Abstract Algebra

Theory and Applications

JUDSON

PreTeXt Sample Book

Abstract Algebra (SAMPLE ONLY)

PreTeXt Sample Book

Abstract Algebra (SAMPLE ONLY)

Thomas W. Judson
Stephen F. Austin State University

Isaac Newton, Editor
Trinity College

Sage Exercises for Abstract Algebra

Robert A. Beezer
University of Puget Sound

April 25, 2024

Thomas W. Judson did his undergraduate work in Illinois, his graduate work in Oregon and presently teaches in Texas.

He likes to ride his bicycle in France, especially in the high Alps and Pyrenees on the roads of the Tour de France.

About Robert A. Beezer Rob Beezer designed, wrote, and tested the Sage exercises as a contribution to this open source project.

He also likes to ride his bicycle, and once rode with Tom Judson in the high Alps, in addition to some hiking there, up above the passes the cyclists ride.

Cover Design: Covers 4 U
Production Editor: Vilma Mesa

Edition: Annual Edition 2015

Website: abstract.pugetsound.edu

[1] example.com
[2] A second footnote, as well.

To students of algebra everywhere
they are the reason

And to those who teach them

Acknowledgements

I would like to acknowledge the following reviewers for their helpful comments and suggestions.

- David Anderson, University of Tennessee, Knoxville
- Robert Beezer, University of Puget Sound
- Myron Hood, California Polytechnic State University
- Herbert Kasube, Bradley University
- John Kurtzke, University of Portland
- Inessa Levi, University of Louisville
- Geoffrey Mason, University of California, Santa Cruz
- Bruce Mericle, Mankato State University
- Kimmo Rosenthal, Union College
- Mark Teply, University of Wisconsin

I would also like to thank Steve Quigley, Marnie Pommett, Cathie Griffin, Kelle Karshick, and the rest of the staff at PWS Publishing for their guidance throughout this project. It has been a pleasure to work with them.

Robert Beezer encouraged me to make *Abstract Algebra: Theory and Applications* available as an open source textbook, a decision that I have never regretted. With his assistance, the book has been rewritten in PreTeXt (pretextbook.org), making it possible to quickly output print, web, PDF versions and more from the same source. The open source version of this book has received support from the National Science Foundation (Award #DUE-1020957).

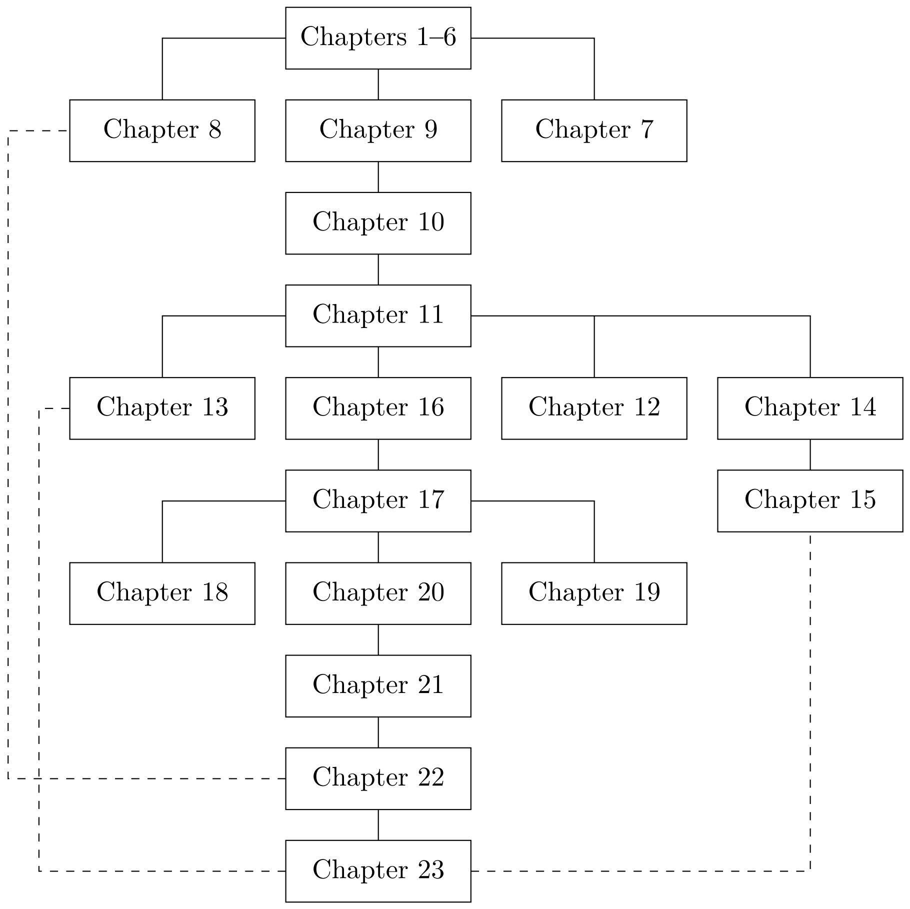
Preface

This text is intended for a one or two-semester undergraduate course in abstract algebra. Traditionally, these courses have covered the theoretical aspects of groups, rings, and fields. However, with the development of computing in the last several decades, applications that involve abstract algebra and discrete mathematics have become increasingly important, and many science, engineering, and computer science students are now electing to minor in mathematics. Though theory still occupies a central role in the subject of abstract algebra and no student should go through such a course without a good notion of what a proof is, the importance of applications such as coding theory and cryptography has grown significantly.

Until recently most abstract algebra texts included few if any applications. However, one of the major problems in teaching an abstract algebra course is that for many students it is their first encounter with an environment that requires them to do rigorous proofs. Such students often find it hard to see the use of learning to prove theorems and propositions; applied examples help the instructor provide motivation.

This text contains more material than can possibly be covered in a single semester. Certainly there is adequate material for a two-semester course, and perhaps more; however, for a one-semester course it would be quite easy to omit selected chapters and still have a useful text. The order of presentation of topics is standard: groups, then rings, and finally fields. Emphasis can be placed either on theory or on applications. A typical one-semester course might cover groups and rings while briefly touching on field theory, using Chapters 1 through 6, 9, 10, 11, 13 (the first part), 16, 17, 18 (the first part), 20, and 21. Parts of these chapters could be deleted and applications substituted according to the interests of the students and the instructor. A two-semester course emphasizing theory might cover Chapters 1 through 6, 9, 10, 11, 13 through 18, 20, 21, 22 (the first part), and 23. On the other hand, if applications are to be emphasized, the course might cover Chapters 1 through 14, and 16 through 22. In an applied course, some of the more theoretical results could be assumed or omitted. A chapter dependency chart appears below. (A broken line indicates a partial dependency.) See the Table of Contents for more.

This real text has been used as the basis of a sample book for testing PreTeXt. So it is slowly migrating away from what the real book looks like and should not be construed as representative. For example, we have reduced the book to four chapters.



Though there are no specific prerequisites for a course in abstract algebra, students who have had other higher-level courses in mathematics will generally be more prepared than those who have not, because they will possess a bit more mathematical sophistication. Occasionally, we shall assume some basic linear algebra; that is, we shall take for granted an elementary knowledge of matrices and determinants. This should present no great problem, since most students taking a course in abstract algebra have been introduced to matrices and determinants elsewhere in their career, if they have not already taken a sophomore or junior-level course in linear algebra.

Exercise sections are the heart of any mathematics text. An exercise set appears at the end of each chapter. The nature of the exercises ranges over several categories; computational, conceptual, and theoretical problems are included. A section presenting hints and solutions to many of the exercises appears at the end of the text. Often in the solutions a proof is only sketched, and it is up to the student to provide the details. The exercises range in difficulty from very easy to very challenging. Many of the more substantial problems require careful thought, so the student should not be discouraged if the solution is not forthcoming after a few minutes of work.

There are additional exercises or computer projects at the ends of many of the chapters. The computer projects usually require a knowledge of programming. All of these exercises and projects are more substantial in nature and allow the exploration of new results and theory.

Sage (`sagemath.org`) is a free, open source, software system for advanced mathematics, which is ideal for assisting with a study of abstract algebra. Sage can be used either on your own computer, a local server, or on SageMathCloud

(cloud.sagemath.com). Robert Beezer has written a comprehensive introduction to Sage and a selection of relevant exercises that appear at the end of each chapter, including live Sage cells in the web version of the book.

Thomas W. Judson
Nacogdoches, Texas 2015

Contributors to the 4th Edition

Many individuals have made this book possible. We will try to thank a few of them here, and hope we have not forgotten anybody really important.

THOMAS JUDSON
Department of Mathematics and Statistics
Stephen F. Austin State University
judsontw@sfasu.edu

DAVID FARMER
American Institute of Mathematics
farmer@aimath.org

ROBERT BEEZER
Department of Mathematics and Computer Science
University of Puget Sound
Tacoma, Washington, USA
beezer@pugetsound.edu

ALEX JORDAN
Department of Mathematics
Portland Community College
Portland, OR
alex.jordan@pcc.edu

THOMAS JUDSON
Department of Mathematics and Statistics
Stephen F. Austin State University
judsontw@sfasu.edu

DAVID FARMER
American Institute of Mathematics
farmer@aimath.org

ROBERT BEEZER
Department of Mathematics and Computer Science
University of Puget Sound
Tacoma, Washington, USA
beezer@pugetsound.edu

ALEX JORDAN
Department of Mathematics
Portland Community College
Portland, OR
alex.jordan@pcc.edu

THOMAS JUDSON
Department of Mathematics and Statistics
Stephen F. Austin State University
judsontw@sfasu.edu

DAVID FARMER
American Institute of Mathematics
farmer@aimath.org

ROBERT BEEZER
Department of Mathematics and Computer Science
University of Puget Sound
Tacoma, Washington, USA
beezer@pugetsound.edu

ALEX JORDAN
Department of Mathematics
Portland Community College
Portland, OR
alex.jordan@pcc.edu

THOMAS JUDSON
Department of Mathematics and Statistics
Stephen F. Austin State University
judsontw@sfasu.edu

DAVID FARMER
American Institute of Mathematics
farmer@aimath.org

Robert Beezer
Department of Mathematics and Computer Science
University of Puget Sound
Tacoma, Washington, USA
beezer@pugetsound.edu

Alex Jordan
Department of Mathematics
Portland Community College
Portland, OR
alex.jordan@pcc.edu

Thomas Judson
Department of Mathematics and Statistics
Stephen F. Austin State University
judsontw@sfasu.edu

David Farmer
American Institute of Mathematics
farmer@aimath.org

Robert Beezer
Department of Mathematics and Computer Science
University of Puget Sound
Tacoma, Washington, USA
beezer@pugetsound.edu

Alex Jordan
Department of Mathematics
Portland Community College
Portland, OR
alex.jordan@pcc.edu

That’s it. Thanks everybody.

Contents

Chapter 1

Preliminaries

A certain amount of mathematical maturity is necessary to find and study applications of abstract algebra. A basic knowledge of set theory, mathematical induction, equivalence relations, and matrices is a must. Even more important is the ability to read and understand mathematical proofs. In this chapter we will outline the background needed for a course in abstract algebra.

It helps when testing to know when this sample document was generated: April 25, 2024, 17:34:00 (-07:00).

1.1 A Short Note on Proofs

Abstract mathematics is different from other sciences. In laboratory sciences such as chemistry and physics, scientists perform experiments to discover new principles and verify theories. Although mathematics is often motivated by physical experimentation or by computer simulations, it is made rigorous through the use of logical arguments. In studying abstract mathematics, we take what is called an axiomatic approach; that is, we take a collection of objects $\mathcal{S}$ and assume some rules about their structure. These rules are called **axioms**. Using the axioms for $\mathcal{S}$, we wish to derive other information about $\mathcal{S}$ by using logical arguments. We require that our axioms be consistent; that is, they should not contradict one another. We also demand that there not be too many axioms. If a system of axioms is too restrictive, there will be few examples of the mathematical structure.

A **statement** in logic or mathematics is an assertion that is either true or false. Consider the following examples:

- $3 + 56 - 13 + 8/2$.
- All cats are black.
- $2 + 3 = 5$.
- $2x = 6$ exactly when $x = 4$.
- If $ax^2 + bx + c = 0$ and $a \neq 0$, then
$$x = \frac{-b \pm \sqrt{b^2 - 4ac}}{2a}.$$
- $x^3 - 4x^2 + 5x - 6$.

All but the first and last examples are statements, and must be either true or false.

A **mathematical proof** is nothing more than a convincing argument about the accuracy of a statement. Such an argument should contain enough detail to convince the audience; for instance, we can see that the statement "$2x = 6$ exactly when $x = 4$" is false by evaluating $2 \cdot 4$ and noting that $6 \neq 8$, an argument that would satisfy anyone. Of course, audiences may vary widely: proofs can be addressed to another student, to a professor, or to the reader of a text. If more detail than needed is presented in the proof, then the explanation will be either long-winded or poorly written. If too much detail is omitted, then the proof may not be convincing. Again it is important to keep the audience in mind. High school students require much more detail than do graduate students. A good rule of thumb for an argument in an introductory abstract algebra course is that it should be written to convince one's peers, whether those peers be other students or other readers of the text.

Let us examine different types of statements. A statement could be as simple as "$10/5 = 2$;" however, mathematicians are usually interested in more complex statements such as "If p, then q," where p and q are both statements. If certain statements are known or assumed to be true, we wish to know what we can say about other statements. Here p is called the **hypothesis** and q is known as the **conclusion**. Consider the following statement: If $ax^2+bx+c = 0$ and $a \neq 0$, then

$$x = \frac{-b \pm \sqrt{b^2 - 4ac}}{2a}.$$

The hypothesis is $ax^2 + bx + c = 0$ and $a \neq 0$; the conclusion is

$$x = \frac{-b \pm \sqrt{b^2 - 4ac}}{2a}.$$

Notice that the statement says nothing about whether or not the hypothesis is true. However, if this entire statement is true and we can show that $ax^2 + bx + c = 0$ with $a \neq 0$ is true, then the conclusion *must* be true. A proof of this statement might simply be a series of equations:

$$\begin{aligned}
ax^2 + bx + c &= 0 \\
x^2 + \frac{b}{a}x &= -\frac{c}{a} \\
x^2 + \frac{b}{a}x + \left(\frac{b}{2a}\right)^2 &= \left(\frac{b}{2a}\right)^2 - \frac{c}{a} \\
\left(x + \frac{b}{2a}\right)^2 &= \frac{b^2 - 4ac}{4a^2} \\
x + \frac{b}{2a} &= \frac{\pm\sqrt{b^2 - 4ac}}{2a} \\
x &= \frac{-b \pm \sqrt{b^2 - 4ac}}{2a}.
\end{aligned}$$

If we can prove a statement true, then that statement is called a **proposition**. A proposition of major importance is called a **theorem**. Sometimes instead of proving a theorem or proposition all at once, we break the proof down into modules; that is, we prove several supporting propositions, which are called **lemmas**, and use the results of these propositions to prove the main result. If we can prove a proposition or a theorem, we will often, with very little effort, be able to derive other related propositions called **corollaries**.

1.1.1 Some Cautions and Suggestions

There are several different strategies for proving propositions. In addition to using different methods of proof, students often make some common mistakes when they are first learning how to prove theorems. To aid students who are studying abstract mathematics for the first time, we list here some of the difficulties that they may encounter and some of the strategies of proof available to them. It is a good idea to keep referring back to this list as a reminder. (Other techniques of proof will become apparent throughout this chapter and the remainder of the text.)

- A theorem cannot be proved by example; however, the standard way to show that a statement is not a theorem is to provide a counterexample.
- Quantifiers are important. Words and phrases such as *only*, *for all*, *for every*, and *for some* possess different meanings.
- Never assume any hypothesis that is not explicitly stated in the theorem. *You cannot take things for granted.*
- Suppose you wish to show that an object *exists* and is *unique*. First show that there actually is such an object. To show that it is unique, assume that there are two such objects, say r and s, and then show that $r = s$.
- Sometimes it is easier to prove the contrapositive of a statement. Proving the statement "If p, then q" is exactly the same as proving the statement "If not q, then not p."
- Although it is usually better to find a direct proof of a theorem, this task can sometimes be difficult. It may be easier to assume that the theorem that you are trying to prove is false, and to hope that in the course of your argument you are forced to make some statement that cannot possibly be true.

Remember that one of the main objectives of higher mathematics is proving theorems. Theorems are tools that make new and productive applications of mathematics possible. We use examples to give insight into existing theorems and to foster intuitions as to what new theorems might be true. Applications, examples, and proofs are tightly interconnected—much more so than they may seem at first appearance.

1.2 Sets and Equivalence Relations

1.2.1 Set Theory

A **set** is a well-defined collection of objects; that is, it is defined in such a manner that we can determine for any given object x whether or not x belongs to the set. The objects that belong to a set are called its **elements** or **members**. We will denote sets by capital letters, such as A or X; if a is an element of the set A, we write $a \in A$.

A set is usually specified either by listing all of its elements inside a pair of braces or by stating the property that determines whether or not an object x belongs to the set. We might write

$$X = \{x_1, x_2, \ldots, x_n\}$$

for a set containing elements $x_1, x_2, \ldots, x_n$ or

$$X = \{x : x \text{ satisfies } \mathcal{P}\}$$

if each x in X satisfies a certain property $\mathcal{P}$. For example, if E is the set of even positive integers, we can describe E by writing either

$$E = \{2, 4, 6, \ldots\} \quad \text{or} \quad E = \{x : x \text{ is an even integer and } x > 0\}.$$

We write $2 \in E$ when we want to say that 2 is in the set E, and $-3 \notin E$ to say that -3 is not in the set E.

Some of the more important sets that we will consider are the following:

$$\begin{aligned}
\mathbb{N} &= \{n : n \text{ is a natural number}\} = \{1, 2, 3, \ldots\} \\
\mathbb{Z} &= \{n : n \text{ is an integer}\} = \{\ldots, -1, 0, 1, 2, \ldots\} \\
\mathbb{Q} &= \{r : r \text{ is a rational number}\} = \{p/q : p, q \in \mathbb{Z} \text{ where } q \neq 0\} \\
\mathbb{R} &= \{x : x \text{ is a real number}\} \\
\mathbb{C} &= \{z : z \text{ is a complex number}\}.
\end{aligned}$$

We can find various relations between sets as well as perform operations on sets. A set A is a **subset** of B, written $A \subset B$ or $B \supset A$, if every element of A is also an element of B. For example,

$$\{4, 5, 8\} \subset \{2, 3, 4, 5, 6, 7, 8, 9\}$$

and

$$\mathbb{N} \subset \mathbb{Z} \subset \mathbb{Q} \subset \mathbb{R} \subset \mathbb{C}.$$

Trivially, every set is a subset of itself. A set B is a **proper subset** of a set A if $B \subset A$ but $B \neq A$. If A is not a subset of B, we write $A \not\subset B$; for example, $\{4, 7, 9\} \not\subset \{2, 4, 5, 8, 9\}$. Two sets are **equal**, written $A = B$, if we can show that $A \subset B$ and $B \subset A$.

It is convenient to have a set with no elements in it. This set is called the **empty set** and is denoted by $\emptyset$. Note that the empty set is a subset of every set.

To construct new sets out of old sets, we can perform certain operations: the **union** $A \cup B$ of two sets A and B is defined as

$$A \cup B = \{x : x \in A \text{ or } x \in B\}$$

and the **intersection** of A and B is defined by

$$A \cap B = \{x : x \in A \text{ and } x \in B\}.$$

If $A = \{1, 3, 5\}$ and $B = \{1, 2, 3, 9\}$, then

$$A \cup B = \{1, 2, 3, 5, 9\} \quad \text{and} \quad A \cap B = \{1, 3\}.$$

We can consider the union and the intersection of more than two sets. In this case we write

$$\bigcup_{i=1}^{n} A_i = A_1 \cup \ldots \cup A_n$$

and

$$\bigcap_{i=1}^{n} A_i = A_1 \cap \ldots \cap A_n$$

for the union and intersection, respectively, of the sets $A_1, \ldots, A_n$.

When two sets have no elements in common, they are said to be **disjoint**; for example, if E is the set of even integers and O is the set of odd integers, then E and O are disjoint. Two sets A and B are disjoint exactly when $A \cap B = \emptyset$.

Sometimes we will work within one fixed set U, called the **universal set**. For any set $A \subset U$, we define the **complement** of A, denoted by A', to be the set

$$A' = \{x : x \in U \text{ and } x \notin A\}.$$

We define the **difference** of two sets A and B to be

$$A \setminus B = A \cap B' = \{x : x \in A \text{ and } x \notin B\}.$$

Example 1.2.1 Set Operations. Let $\mathbb{R}$ be the universal set and suppose that

$$A = \{x \in \mathbb{R} : 0 < x \leq 3\} \quad \text{and} \quad B = \{x \in \mathbb{R} : 2 \leq x < 4\}.$$

Then

$$\begin{aligned} A \cap B &= \{x \in \mathbb{R} : 2 \leq x \leq 3\} \\ A \cup B &= \{x \in \mathbb{R} : 0 < x < 4\} \\ A \setminus B &= \{x \in \mathbb{R} : 0 < x < 2\} \\ A' &= \{x \in \mathbb{R} : x \leq 0 \text{ or } x > 3\}. \end{aligned}$$

□

Proposition 1.2.2 *Let A, B, and C be sets. Then*

1. *$A \cup A = A$, $A \cap A = A$, and $A \setminus A = \emptyset$;*
2. *$A \cup \emptyset = A$ and $A \cap \emptyset = \emptyset$;*
3. *$A \cup (B \cup C) = (A \cup B) \cup C$ and $A \cap (B \cap C) = (A \cap B) \cap C$;*
4. *$A \cup B = B \cup A$ and $A \cap B = B \cap A$;*
5. *$A \cup (B \cap C) = (A \cup B) \cap (A \cup C)$;*
6. *$A \cap (B \cup C) = (A \cap B) \cup (A \cap C)$.*

Proof. We will prove (1) and (3) and leave the remaining results to be proven in the exercises.

(1) Observe that

$$\begin{aligned} A \cup A &= \{x : x \in A \text{ or } x \in A\} \\ &= \{x : x \in A\} \\ &= A \end{aligned}$$

and

$$\begin{aligned} A \cap A &= \{x : x \in A \text{ and } x \in A\} \\ &= \{x : x \in A\} \\ &= A. \end{aligned}$$

Also, $A \setminus A = A \cap A' = \emptyset$.

(3) For sets A, B, and C,

$$\begin{aligned} A \cup (B \cup C) &= A \cup \{x : x \in B \text{ or } x \in C\} \\ &= \{x : x \in A \text{ or } x \in B, \text{ or } x \in C\} \\ &= \{x : x \in A \text{ or } x \in B\} \cup C \end{aligned}$$

$$= (A \cup B) \cup C.$$

A similar argument proves that $A \cap (B \cap C) = (A \cap B) \cap C$. ■

Theorem 1.2.3 De Morgan's Laws. (Augustus De Morgan, 1806–1871) *Let A and B be sets. Then*

1. $(A \cup B)' = A' \cap B'$;
2. $(A \cap B)' = A' \cup B'$.

Proof. (1) We must show that $(A \cup B)' \subset A' \cap B'$ and $(A \cup B)' \supset A' \cap B'$. Let $x \in (A \cup B)'$. Then $x \notin A \cup B$. So x is neither in A nor in B, by the definition of the union of sets. By the definition of the complement, $x \in A'$ and $x \in B'$. Therefore, $x \in A' \cap B'$ and we have $(A \cup B)' \subset A' \cap B'$.
To show the reverse inclusion, suppose that $x \in A' \cap B'$. Then $x \in A'$ and $x \in B'$, and so $x \notin A$ and $x \notin B$. Thus $x \notin A \cup B$ and so $x \in (A \cup B)'$. Hence, $(A \cup B)' \supset A' \cap B'$ and so $(A \cup B)' = A' \cap B'$.
The proof of (2) is left as an exercise. ■

Example 1.2.4 Other Relations on Sets. Other relations between sets often hold true. For example,

$$(A \setminus B) \cap (B \setminus A) = \emptyset.$$

To see that this is true, observe that

$$\begin{aligned} (A \setminus B) \cap (B \setminus A) &= (A \cap B') \cap (B \cap A') \\ &= A \cap A' \cap B \cap B' \\ &= \emptyset. \end{aligned}$$

□

1.2.2 Cartesian Products and Mappings

Given sets A and B, we can define a new set $A \times B$, called the **Cartesian product** of A and B, as a set of ordered pairs. That is,

$$A \times B = \{(a, b) : a \in A \text{ and } b \in B\}.$$

Example 1.2.5 Cartesian Products. If $A = \{x, y\}$, $B = \{1, 2, 3\}$, and $C = \emptyset$, then $A \times B$ is the set

$$\{(x, 1), (x, 2), (x, 3), (y, 1), (y, 2), (y, 3)\}$$

and

$$A \times C = \emptyset.$$

□

We define the **Cartesian product of n sets** to be

$$A_1 \times \cdots \times A_n = \{(a_1, \ldots, a_n) : a_i \in A_i \text{ for } i = 1, \ldots, n\}.$$

If $A = A_1 = A_2 = \cdots = A_n$, we often write A^n for $A \times \cdots \times A$ (where A would be written n times). For example, the set $\mathbb{R}^3$ consists of all of 3-tuples of real numbers.

Subsets of $A \times B$ are called **relations**. We will define a **mapping** or **function** $f \subset A \times B$ from a set A to a set B to be the special type of relation where $(a, b) \in f$ if for every element $a \in A$ there exists a unique element $b \in B$.

Another way of saying this is that for every element in A, f assigns a unique element in B. We usually write $f : A \to B$ or $A \xrightarrow{f} B$. Instead of writing down ordered pairs $(a, b) \in A \times B$, we write $f(a) = b$ or $f : a \mapsto b$. The set A is called the **domain** of f and

$$f(A) = \{f(a) : a \in A\} \subset B$$

is called the **range** or **image** of f. We can think of the elements in the function's domain as input values and the elements in the function's range as output values.

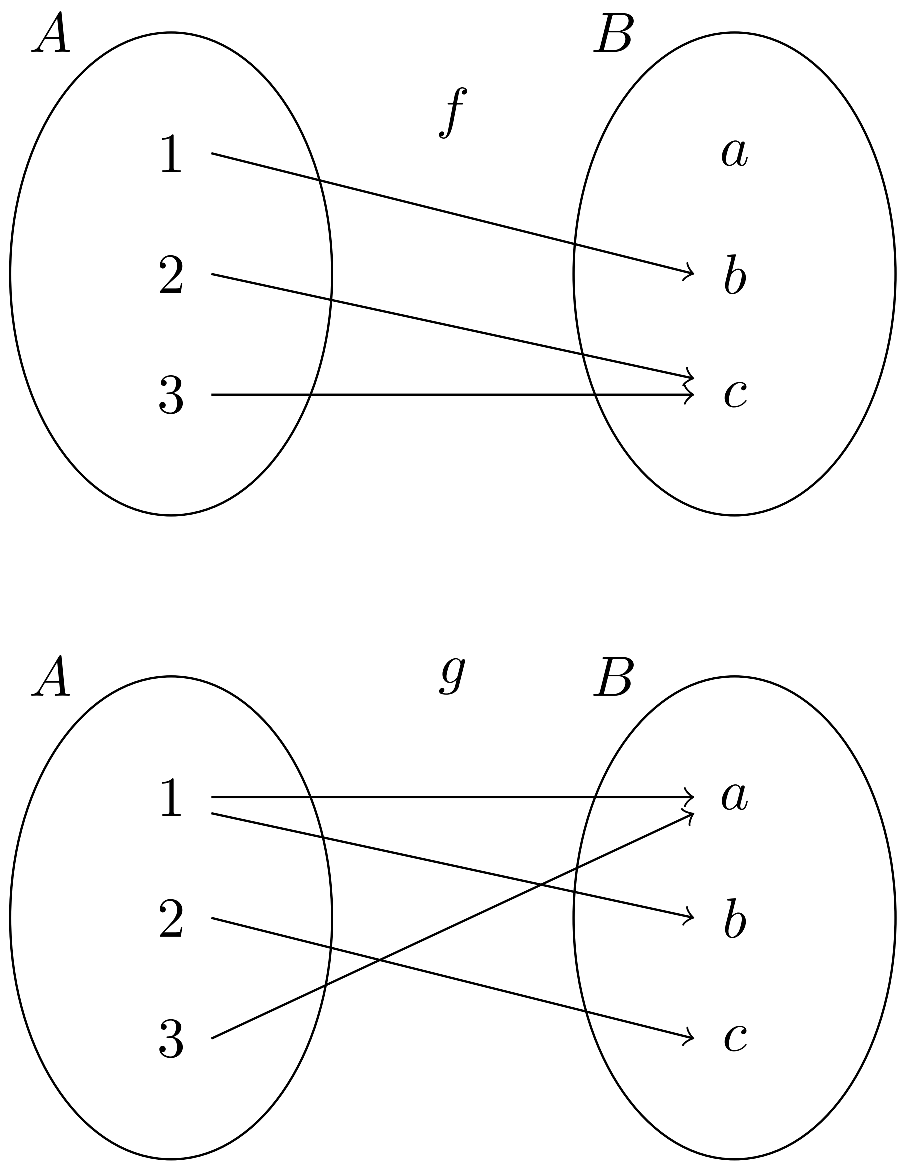


Figure 1.2.6 Mappings and relations

Example 1.2.7 Mappings. Suppose $A = \{1, 2, 3\}$ and $B = \{a, b, c\}$. In Figure 1.2.6 we define relations f and g from A to B. The relation f is a mapping, but g is not because $1 \in A$ is not assigned to a unique element in B; that is, $g(1) = a$ and $g(1) = b$. $\square$

Given a function $f : A \to B$, it is often possible to write a list describing what the function does to each specific element in the domain. However, not all functions can be described in this manner. For example, the function $f : \mathbb{R} \to \mathbb{R}$ that sends each real number to its cube is a mapping that must be described by writing $f(x) = x^3$ or $f : x \mapsto x^3$.

Consider the relation $f : \mathbb{Q} \to \mathbb{Z}$ given by $f(p/q) = p$. We know that $1/2 = 2/4$, but is $f(1/2) = 1$ or 2? This relation cannot be a mapping because it is not well-defined. A relation is **well-defined** if each element in the domain is assigned to a *unique* element in the range.

If $f : A \to B$ is a map and the image of f is B, i.e., $f(A) = B$, then f is said to be **onto** or **surjective**. In other words, if there exists an $a \in A$ for each $b \in B$ such that $f(a) = b$, then f is onto. A map is **one-to-one** or **injective** if $a_1 \neq a_2$ implies $f(a_1) \neq f(a_2)$. Equivalently, a function is one-to-one if $f(a_1) = f(a_2)$ implies $a_1 = a_2$. A map that is both one-to-one and onto is called **bijective**.

Example 1.2.8 One-to-One and Onto Mappings. Let $f : \mathbb{Z} \to \mathbb{Q}$ be defined by $f(n) = n/1$. Then f is one-to-one but not onto. Define $g : \mathbb{Q} \to \mathbb{Z}$ by $g(p/q) = p$ where p/q is a rational number expressed in its lowest terms with a positive denominator. The function g is onto but not one-to-one. □

Given two functions, we can construct a new function by using the range of the first function as the domain of the second function. Let $f : A \to B$ and $g : B \to C$ be mappings. Define a new map, the **composition** of f and g from A to C, by $(g \circ f)(x) = g(f(x))$.

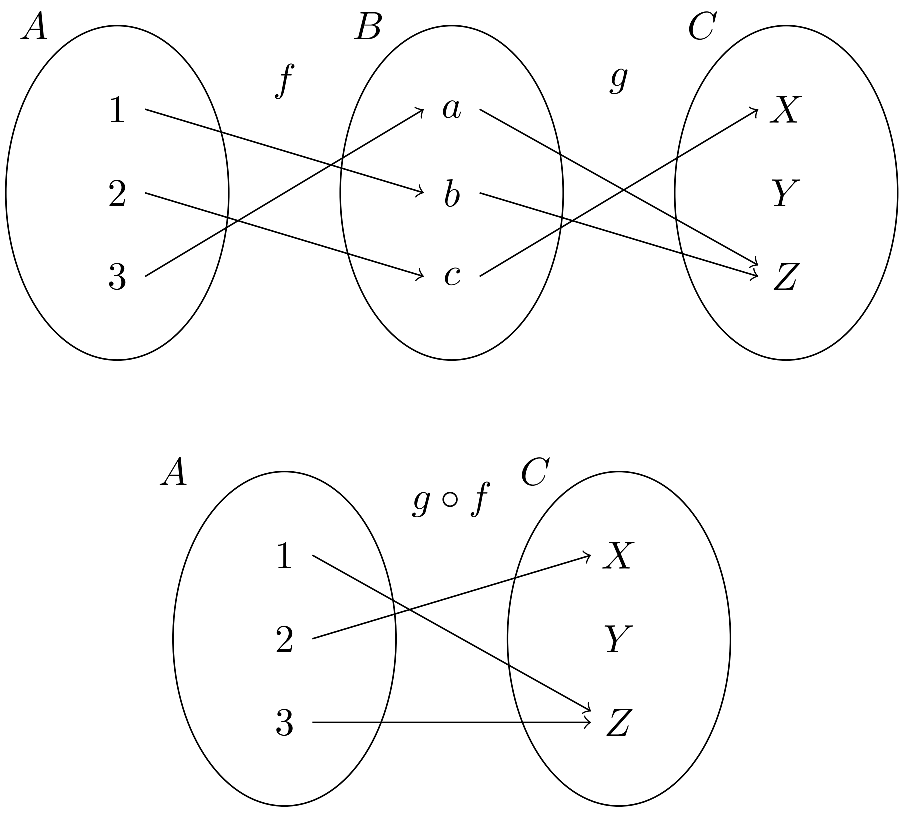


Figure 1.2.9 Composition of maps

Example 1.2.10 Composition of Mappings. Consider the functions $f : A \to B$ and $g : B \to C$ that are defined in Figure 1.2.9 (top). The composition of these functions, $g \circ f : A \to C$, is defined in Figure 1.2.9 (bottom). □

Example 1.2.11 Composition is not Commutative. Let $f(x) = x^2$ and $g(x) = 2x + 5$. Then

$$(f \circ g)(x) = f(g(x)) = (2x+5)^2 = 4x^2 + 20x + 25$$

and

$$(g \circ f)(x) = g(f(x)) = 2x^2 + 5.$$

In general, order makes a difference; that is, in most cases $f \circ g \neq g \circ f$. □

Example 1.2.12 Some Mappings Commute. Sometimes it is the case that $f \circ g = g \circ f$. Let $f(x) = x^3$ and $g(x) = \sqrt[3]{x}$. Then

$$(f \circ g)(x) = f(g(x)) = f(\sqrt[3]{x}\,) = (\sqrt[3]{x}\,)^3 = x$$

and

$$(g \circ f)(x) = g(f(x)) = g(x^3) = \sqrt[3]{x^3} = x.$$

□

Example 1.2.13 A Linear Map. Given a 2×2 matrix

$$A = \begin{pmatrix} a & b \\ c & d \end{pmatrix},$$

we can define a map $T_A : \mathbb{R}^2 \to \mathbb{R}^2$ by

$$T_A(x, y) = (ax + by, cx + dy)$$

for (x, y) in $\mathbb{R}^2$. This is actually matrix multiplication; that is,

$$\begin{pmatrix} a & b \\ c & d \end{pmatrix} \begin{pmatrix} x \\ y \end{pmatrix} = \begin{pmatrix} ax + by \\ cx + dy \end{pmatrix}.$$

Maps from $\mathbb{R}^n$ to $\mathbb{R}^m$ given by matrices are called **linear maps** or **linear transformations**. □

Example 1.2.14 A Permutation. Suppose that $S = \{1, 2, 3\}$. Define a map $\pi : S \to S$ by

$$\pi(1) = 2, \qquad \pi(2) = 1, \qquad \pi(3) = 3.$$

This is a bijective map. An alternative way to write π is

$$\begin{pmatrix} 1 & 2 & 3 \\ \pi(1) & \pi(2) & \pi(3) \end{pmatrix} = \begin{pmatrix} 1 & 2 & 3 \\ 2 & 1 & 3 \end{pmatrix}.$$

For any set S, a one-to-one and onto mapping $\pi : S \to S$ is called a **permutation** of S. □

Theorem 1.2.15 *Let $f : A \to B$, $g : B \to C$, and $h : C \to D$. Then*

1. *The composition of mappings is associative; that is, $(h \circ g) \circ f = h \circ (g \circ f)$;*
2. *If f and g are both one-to-one, then the mapping $g \circ f$ is one-to-one;*
3. *If f and g are both onto, then the mapping $g \circ f$ is onto;*
4. *If f and g are bijective, then so is $g \circ f$.*

Proof. We will prove (1) and (3). Part (2) is left as an exercise. Part (4) follows directly from (2) and (3).
(1) We must show that

$$h \circ (g \circ f) = (h \circ g) \circ f.$$

For $a \in A$ we have

$$\begin{aligned}(h \circ (g \circ f))(a) &= h((g \circ f)(a)) \\ &= h(g(f(a))) \\ &= (h \circ g)(f(a)) \\ &= ((h \circ g) \circ f)(a).\end{aligned}$$

(3) Assume that f and g are both onto functions. Given $c \in C$, we must show that there exists an $a \in A$ such that $(g \circ f)(a) = g(f(a)) = c$. However, since g is onto, there is an element $b \in B$ such that $g(b) = c$. Similarly, there is an $a \in A$ such that $f(a) = b$. Accordingly,

$$(g \circ f)(a) = g(f(a)) = g(b) = c.$$

■

If S is any set, we will use id_S or id to denote the **identity mapping** from S to itself. Define this map by $id(s) = s$ for all $s \in S$. A map $g : B \to A$ is an **inverse mapping** of $f : A \to B$ if $g \circ f = id_A$ and $f \circ g = id_B$; in other words, the inverse function of a function simply "undoes" the function. A map is said to be **invertible** if it has an inverse. We usually write f^{-1} for the inverse of f.

Example 1.2.16 An Inverse Function. The function $f(x) = x^3$ has inverse $f^{-1}(x) = \sqrt[3]{x}$ by Example 1.2.12. □

Example 1.2.17 Exponential and Logarithmic Functions are Inverses. The natural logarithm and the exponential functions, $f(x) = \ln x$ and $f^{-1}(x) = e^x$, are inverses of each other provided that we are careful about choosing domains. Observe that

$$f(f^{-1}(x)) = f(e^x) = \ln e^x = x$$

and

$$f^{-1}(f(x)) = f^{-1}(\ln x) = e^{\ln x} = x$$

whenever composition makes sense. □

Example 1.2.18 A Matrix Inverse Yields an Inverse of a Linear Map. Suppose that

$$A = \begin{pmatrix} 3 & 1 \\ 5 & 2 \end{pmatrix}.$$

Then A defines a map from $\mathbb{R}^2$ to $\mathbb{R}^2$ by

$$T_A(x, y) = (3x + y, 5x + 2y).$$

We can find an inverse map of T_A by simply inverting the matrix A; that is, $T_A^{-1} = T_{A^{-1}}$. In this example,

$$A^{-1} = \begin{pmatrix} 2 & -1 \\ -5 & 3 \end{pmatrix}$$

and hence, the inverse map is given by

$$T_A^{-1}(x, y) = (2x - y, -5x + 3y).$$

It is easy to check that

$$T_A^{-1} \circ T_A(x, y) = T_A \circ T_A^{-1}(x, y) = (x, y).$$

Not every map has an inverse. If we consider the map

$$T_B(x, y) = (3x, 0)$$

given by the matrix

$$B = \begin{pmatrix} 3 & 0 \\ 0 & 0 \end{pmatrix},$$

then an inverse map would have to be of the form

$$T_B^{-1}(x, y) = (ax + by, cx + dy)$$

and

$$(x, y) = T \circ T_B^{-1}(x, y) = (3ax + 3by, 0)$$

for all x and y. Clearly this is impossible because y might not be 0. □

Example 1.2.19 An Inverse Permutation. Given the permutation

$$\pi = \begin{pmatrix} 1 & 2 & 3 \\ 2 & 3 & 1 \end{pmatrix}$$

on $S = \{1, 2, 3\}$, it is easy to see that the permutation defined by

$$\pi^{-1} = \begin{pmatrix} 1 & 2 & 3 \\ 3 & 1 & 2 \end{pmatrix}$$

is the inverse of π. In fact, any bijective mapping possesses an inverse, as we will see in the next theorem. □

Theorem 1.2.20 *A mapping is invertible if and only if it is both one-to-one and onto.*

Proof. Suppose first that $f : A \to B$ is invertible with inverse $g : B \to A$. Then $g \circ f = id_A$ is the identity map; that is, $g(f(a)) = a$. If $a_1, a_2 \in A$ with $f(a_1) = f(a_2)$, then $a_1 = g(f(a_1)) = g(f(a_2)) = a_2$. Consequently, f is one-to-one. Now suppose that $b \in B$. To show that f is onto, it is necessary to find an $a \in A$ such that $f(a) = b$, but $f(g(b)) = b$ with $g(b) \in A$. Let $a = g(b)$.

Conversely, let f be bijective and let $b \in B$. Since f is onto, there exists an $a \in A$ such that $f(a) = b$. Because f is one-to-one, a must be unique. Define g by letting $g(b) = a$. We have now constructed the inverse of f. ■

1.2.3 Equivalence Relations and Partitions

A fundamental notion in mathematics is that of equality. We can generalize equality with equivalence relations and equivalence classes. An **equivalence relation** on a set X is a relation $R \subset X \times X$ such that

- $(x, x) \in R$ for all $x \in X$ (**reflexive property**);
- $(x, y) \in R$ implies $(y, x) \in R$ (**symmetric property**);
- (x, y) and $(y, z) \in R$ imply $(x, z) \in R$ (**transitive property**).

Given an equivalence relation R on a set X, we usually write $x \sim y$ instead of $(x, y) \in R$. If the equivalence relation already has an associated notation such as $=$, $\equiv$, or $\cong$, we will use that notation.

Example 1.2.21 Equivalent Fractions. Let p, q, r, and s be integers, where q and s are nonzero. Define $p/q \sim r/s$ if $ps = qr$. Clearly $\sim$ is reflexive and symmetric. To show that it is also transitive, suppose that $p/q \sim r/s$ and $r/s \sim t/u$, with q, s, and u all nonzero. Then $ps = qr$ and $ru = st$. Therefore,

$$psu = qru = qst.$$

Since $s \neq 0$, $pu = qt$. Consequently, $p/q \sim t/u$. □

Example 1.2.22 An Equivalence Relation From Derivatives. Suppose that f and g are differentiable functions on $\mathbb{R}$. We can define an equivalence relation on such functions by letting $f(x) \sim g(x)$ if $f'(x) = g'(x)$. It is clear that $\sim$ is both reflexive and symmetric. To demonstrate transitivity, suppose that $f(x) \sim g(x)$ and $g(x) \sim h(x)$. From calculus we know that $f(x) - g(x) = c_1$ and $g(x) - h(x) = c_2$, where c_1 and c_2 are both constants. Hence,

$$f(x) - h(x) = (f(x) - g(x)) + (g(x) - h(x)) = c_1 - c_2$$

and $f'(x) - h'(x) = 0$. Therefore, $f(x) \sim h(x)$. □

Example 1.2.23 Equivalent Circles. For (x_1, y_1) and (x_2, y_2) in $\mathbb{R}^2$, define $(x_1, y_1) \sim (x_2, y_2)$ if $x_1^2 + y_1^2 = x_2^2 + y_2^2$. Then $\sim$ is an equivalence relation on $\mathbb{R}^2$. □

Example 1.2.24 Equivalent Matrices. Let A and B be 2×2 matrices with entries in the real numbers. We can define an equivalence relation on the set of 2×2 matrices, by saying $A \sim B$ if there exists an invertible matrix P such that $PAP^{-1} = B$. For example, if

$$A = \begin{pmatrix} 1 & 2 \\ -1 & 1 \end{pmatrix} \quad \text{and} \quad B = \begin{pmatrix} -18 & 33 \\ -11 & 20 \end{pmatrix},$$

then $A \sim B$ since $PAP^{-1} = B$ for

$$P = \begin{pmatrix} 2 & 5 \\ 1 & 3 \end{pmatrix}.$$

Let I be the 2×2 identity matrix; that is,

$$I = \begin{pmatrix} 1 & 0 \\ 0 & 1 \end{pmatrix}.$$

Then $IAI^{-1} = IAI = A$; therefore, the relation is reflexive. To show symmetry, suppose that $A \sim B$. Then there exists an invertible matrix P such that $PAP^{-1} = B$. So

$$A = P^{-1}BP = P^{-1}B(P^{-1})^{-1}.$$

Finally, suppose that $A \sim B$ and $B \sim C$. Then there exist invertible matrices P and Q such that $PAP^{-1} = B$ and $QBQ^{-1} = C$. Since

$$C = QBQ^{-1} = QPAP^{-1}Q^{-1} = (QP)A(QP)^{-1},$$

the relation is transitive. Two matrices that are equivalent in this manner are said to be **similar**. □

A **partition** $\mathcal{P}$ of a set X is a collection of nonempty sets $X_1, X_2, \ldots$ such that $X_i \cap X_j = \emptyset$ for $i \neq j$ and $\bigcup_k X_k = X$. Let $\sim$ be an equivalence relation on a set X and let $x \in X$. Then $[x] = \{y \in X : y \sim x\}$ is called the **equivalence class** of x. We will see that an equivalence relation gives rise to a partition via equivalence classes. Also, whenever a partition of a set exists, there is some natural underlying equivalence relation, as the following theorem demonstrates.

Theorem 1.2.25 *Given an equivalence relation $\sim$ on a set X, the equivalence classes of X form a partition of X. Conversely, if $\mathcal{P} = \{X_i\}$ is a partition of a set X, then there is an equivalence relation on X with equivalence classes X_i.*

Proof. Suppose there exists an equivalence relation $\sim$ on the set X. For any $x \in X$, the reflexive property shows that $x \in [x]$ and so $[x]$ is nonempty. Clearly $X = \bigcup_{x \in X} [x]$. Now let $x, y \in X$. We need to show that either $[x] = [y]$ or $[x] \cap [y] = \emptyset$. Suppose that the intersection of $[x]$ and $[y]$ is not empty and that $z \in [x] \cap [y]$. Then $z \sim x$ and $z \sim y$. By symmetry and transitivity $x \sim y$; hence, $[x] \subset [y]$. Similarly, $[y] \subset [x]$ and so $[x] = [y]$. Therefore, any two equivalence classes are either disjoint or exactly the same.

Conversely, suppose that $\mathcal{P} = \{X_i\}$ is a partition of a set X. Let two elements be equivalent if they are in the same partition. Clearly, the relation is reflexive. If x is in the same partition as y, then y is in the same partition as x, so $x \sim y$ implies $y \sim x$. Finally, if x is in the same partition as y and y is in the same partition as z, then x must be in the same partition as z, and transitivity holds. ■

Corollary 1.2.26 *Two equivalence classes of an equivalence relation are either disjoint or equal.*

Let us examine some of the partitions given by the equivalence classes in the last set of examples.

Example 1.2.27 **A Partition of Fractions.** In the equivalence relation in Example 1.2.21, two pairs of integers, (p, q) and (r, s), are in the same equivalence class when they reduce to the same fraction in its lowest terms. □

Example 1.2.28 **A Partition of Functions.** In the equivalence relation in Example 1.2.22, two functions $f(x)$ and $g(x)$ are in the same partition when they differ by a constant. □

Example 1.2.29 **A Partition of Circles.** We defined an equivalence class on $\mathbb{R}^2$ by $(x_1, y_1) \sim (x_2, y_2)$ if $x_1^2 + y_1^2 = x_2^2 + y_2^2$. Two pairs of real numbers are in the same partition when they lie on the same circle about the origin. □

Example 1.2.30 **A Partition of Integers.** Let r and s be two integers and suppose that $n \in \mathbb{N}$. We say that r is **congruent** to s **modulo** n, or r is congruent to s mod n, if $r - s$ is evenly divisible by n; that is, $r - s = nk$ for some $k \in \mathbb{Z}$. In this case we write $r \equiv s \pmod{n}$. For example, $41 \equiv 17 \pmod{8}$ since $41 - 17 = 24$ is divisible by 8. We claim that congruence modulo n forms an equivalence relation of $\mathbb{Z}$. Certainly any integer r is equivalent to itself since $r - r = 0$ is divisible by n. We will now show that the relation is symmetric. If $r \equiv s \pmod{n}$, then $r - s = -(s - r)$ is divisible by n. So $s - r$ is divisible by n and $s \equiv r \pmod{n}$. Now suppose that $r \equiv s \pmod{n}$ and $s \equiv t \pmod{n}$. Then there exist integers k and l such that $r - s = kn$ and $s - t = ln$. To show transitivity, it is necessary to prove that $r - t$ is divisible by n. However,

$$r - t = r - s + s - t = kn + ln = (k + l)n,$$

and so $r - t$ is divisible by n.

If we consider the equivalence relation established by the integers modulo 3, then

$$\begin{aligned}[0] &= \{\ldots, -3, 0, 3, 6, \ldots\}\\ [1] &= \{\ldots, -2, 1, 4, 7, \ldots\}\\ [2] &= \{\ldots, -1, 2, 5, 8, \ldots\}.\end{aligned}$$

Notice that $[0] \cup [1] \cup [2] = \mathbb{Z}$ and also that the sets are disjoint. The sets $[0]$, $[1]$, and $[2]$ form a partition of the integers.

The integers modulo n are a very important example in the study of abstract algebra and will become quite useful in our investigation of various algebraic structures such as groups and rings. In our discussion of the integers modulo n we have actually assumed a result known as the division algorithm, which will be stated and proved in Chapter 2. □

1.3 Sage

Sage is a powerful system for studying and exploring many different areas of mathematics. In this textbook, you will study a variety of algebraic structures, such as groups, rings and fields. Sage does an excellent job of implementing many features of these objects as we will see in the chapters ahead. But here and now, in this initial chapter, we will concentrate on a few general ways of getting the most out of working with Sage.

You may use Sage several different ways. It may be used as a command-line program when installed on your own computer. Or it might be a web application such as the SageMathCloud. Our writing will assume that you are reading this as a worksheet within the Sage Notebook (a web browser interface), or this is a section of the entire book presented as web pages, and you are employing the Sage Cell Server via those pages. After the first few chapters the explanations should work equally well for whatever vehicle you use to execute Sage commands.

1.3.1 Executing Sage Commands

Most of your interaction will be by typing commands into a *compute cell.* If you are reading this in the Sage Notebook or as a webpage version of the book, then you will see a compute cell just below this paragraph. Click once inside the compute cell and if you are in the Sage Notebook, you will get a more distinctive border around it, a blinking cursor inside, plus a cute little "evaluate" link below.At the cursor, type `2+2` and then click on the evaluate link. Did a `4` appear below the cell? If so, you have successfully sent a command off for Sage to evaluate and you have received back the (correct) answer.

Here is another compute cell. Try evaluating the command `factorial(300)` here.Hmmmmm. That is quite a big integer! If you see slashes at the end of each line, this means the result is continued onto the next line, since there are 615 total digits in the result.

To make new compute cells in the Sage Notebook (only), hover your mouse just above another compute cell, or just below some output from a compute cell. When you see a skinny blue bar across the width of your worksheet, click and you will open up a new compute cell, ready for input. Note that your worksheet will remember any calculations you make, in the order you make them, no matter where you put the cells, so it is best to stay organized and add new cells at the bottom.

Try placing your cursor just below the monstrous value of 300! that you have. Click on the blue bar and try another factorial computation in the new compute cell.

Each compute cell will show output due to only the very last command in the cell. Try to predict the following output before evaluating the cell.

```
a = 10
b = 6
b = b - 10
a = a + 20
a
```

```
30
```

The following compute cell will not print anything since the one command does not create output. But it will have an effect, as you can see when you execute the subsequent cell. Notice how this uses the value of `b` from above. Execute this compute cell *once*. Exactly once. Even if it *appears* to do nothing. If you execute the cell twice, your credit card may be charged twice.

```
b = b + 50
```

Now execute this cell, which will produce some output.

```
b + 20
```

```
66
```

So `b` came into existence as `6`. We subtracted 10 immediately afterward. Then a subsequent cell added `50`. This assumes you executed this cell *exactly* once! In the last cell we create `b+20` (but do not save it) and it is this value (66) that is output, while `b` is still 46.

You can combine several commands on one line with a semi-colon. This is a great way to get multiple outputs from a compute cell. The syntax for building a matrix should be somewhat obvious when you see the output, but if not, it is not particularly important to understand now.

```
A = matrix([[3, 1], [5,2]]); A
```

```
[3 1]
[5 2]
```

```
print A; print ; A.inverse()
```

```
[3 1]
[5 2]
<BLANKLINE>
[ 2 -1]
[-5  3]
```

1.3.2 Immediate Help

Some commands in Sage are “functions,” an example is `factorial()` above. Other commands are “methods” of an object and are like characteristics of objects, an example is `.inverse()` as a method of a matrix. Once you know how to create an object (such as a matrix), then it is easy to see all the available methods. Write the name of the object, place a period (“dot”) and hit the TAB key. If you have `A` defined from above, then the compute cell below is ready to go, click into it and then hit TAB (not "evaluate"!). You should get a long list of possible methods.

```
A.
```

To get some help on how to use a method with an object, write its name after a dot (with no parentheses) and then use a question-mark and hit TAB. (Hit the escape key "ESC" to remove the list, or click on the text for a method.)

```
A.inverse?
```

With one more question-mark and a TAB you can see the actual computer instructions that were programmed into Sage to make the method work, once you scoll down past the documentation delimited by the triple quotes ("""):

```
A.inverse??
```

It is worthwhile to see what Sage does when there is an error. You will probably see a lot of these at first, and initially they will be a bit intimidating. But with time, you will learn how to use them effectively and you will also become more proficient with Sage and see them less often. Execute the compute cell below, it asks for the inverse of a matrix that has no inverse. Then reread the commentary.

```
B = matrix([[2, 20], [5, 50]])
B.inverse()
```

```
Traceback (most recent call last):
...
ZeroDivisionError: Matrix is singular
```

Click just to the left of the error message to expand it fully (another click hides it totally, and a third click brings back the abbreviated form). Read the bottom of an error message first, it is your best explanation. Here a `ZeroDivisionError` is not 100% accurate, but is close. The matrix is not invertible, not dissimilar to how we cannot divide scalars by zero. The remainder of the message begins at the top showing were the error first happened in your code and then the various places where intermediate functions were called, until the actual piece of Sage where the problem occurred. Sometimes this information will give you some clues, sometimes it is totally undecipherable. So do not let it scare you if it seems mysterious, but do remember to always read the last line first, then go back and read the first few lines for something that looks like your code.

1.3.3 Annotating Your Work

It is easy to comment on your work when you use the Sage Notebook. (The following only applies if you are reading this within a Sage Notebook. If you are not, then perhaps you can go open up a worksheet in the Sage Notebook and experiment there.) You can open up a small word-processor by hovering your mouse until you get a skinny blue bar again, but now when you click, also hold the SHIFT key at the same time. Experiment with fonts, colors, bullet lists, etc and then click the "Save changes" button to exit. Double-click on your text if you need to go back and edit it later.

Open the word-processor again to create a new bit of text (maybe next to the empty compute cell just below). Type all of the following *exactly*:

```
Pythagorean Theorem: $c^2=a^2+b^2$
```

and save your changes. The symbols between the dollar signs are written according to the mathematical typesetting language known as TEX — cruise the internet to learn more about this very popular tool. (Well, it is extremely popular among mathematicians and physical scientists.)

1.3.4 Lists

Much of our interaction with sets will be through Sage lists. These are not really sets — they allow duplicates, and order matters. But they are so close to sets, and so easy and powerful to use that we will use them regularly. We will use a fun made-up list for practice, the quote marks mean the items are just text, with no special mathematical meaning. Execute these compute cells as we work through them.

```
zoo = ['snake', 'parrot', 'elephant', 'baboon', 'beetle']
zoo
```

```
['snake', 'parrot', 'elephant', 'baboon', 'beetle']
```

So the square brackets define the boundaries of our list, commas separate items, and we can give the list a name. To work with just one element of the list, we use the name and a pair of brackets with an index. Notice that lists have indices that *begin counting at zero*. This will seem odd at first and will seem very natural later.

```
zoo[2]
```

```
'elephant'
```

We can add a new creature to the zoo, it is joined up at the far right end.

```
zoo.append('ostrich'); zoo
```

```
['snake', 'parrot', 'elephant', 'baboon', 'beetle',
    'ostrich']
```

We can remove a creature.

```
zoo.remove('parrot')
zoo
```

```
['snake', 'elephant', 'baboon', 'beetle', 'ostrich']
```

We can extract a sublist. Here we start with element 1 (the elephant) and go all the way up to, *but not including*, element 3 (the beetle). Again a bit odd, but it will feel natural later. For now, notice that we are extracting two elements of the lists, exactly $3 - 1 = 2$ elements.

```
mammals = zoo[1:3]
mammals
```

```
['elephant', 'baboon']
```

Often we will want to see if two lists are equal. To do that we will need to sort a list first. A function creates a new, sorted list, leaving the original alone. So we need to save the new one with a new name.

```
newzoo = sorted(zoo)
newzoo
```

```
['baboon', 'beetle', 'elephant', 'ostrich', 'snake']
```

```
zoo.sort()
zoo
```

```
['baboon', 'beetle', 'elephant', 'ostrich', 'snake']
```

Notice that if you run this last compute cell your zoo has changed and some commands above will not necessarily execute the same way. If you want to experiment, go all the way back to the first creation of the zoo and start executing cells again from there with a fresh zoo.

A construction called a **list comprehension** is especially powerful, especially since it almost exactly mirrors notation we use to describe sets. Suppose we want to form the plural of the names of the creatures in our zoo. We build a new list, based on all of the elements of our old list.

```
plurality_zoo = [animal+'s' for animal in zoo]
plurality_zoo
```

```
['baboons', 'beetles', 'elephants', 'ostrichs', 'snakes']
```

Almost like it says: we add an “s” to each animal name, for each animal in the zoo, and place them in a new list. Perfect. (Except for getting the plural of “ostrich” wrong.)

1.3.5 Lists of Integers

One final type of list, with numbers this time. The `srange()` function will create lists of integers. (The “s” in the name stands for “Sage” and so will produce integers that Sage understands best. Many early difficulties with Sage and group theory can be alleviated by using only this command to create lists of integers.) In its simplest form an invocation like `srange(12)` will create a list of 12 integers, *starting at zero* and working up to, *but not including*, 12. Does this sound familiar?

```
dozen = srange(12); dozen
```

```
[0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11]
```

Here are two other forms, that you should be able to understand by studying the examples.

```
teens = srange(13, 20); teens
```

```
[13, 14, 15, 16, 17, 18, 19]
```

```
decades = srange(1900, 2000, 10); decades
```

```
[1900, 1910, 1920, 1930, 1940, 1950, 1960, 1970, 1980, 1990]
```

1.3.6 Saving and Sharing Your Work

There is a “Save” button in the upper-right corner of the Sage Notebook. This will save a current copy of your worksheet that you can retrieve your work from within your notebook again later, though you have to re-execute all the cells when you re-open the worksheet.

There is also a “File” drop-down list, on the left, just above your very top compute cell (not be confused with your browser’s File menu item!). You will see a choice here labeled “Save worksheet to a file...” When you do this, you are creating a copy of your worksheet in the `sws` format (short for “Sage

WorkSheet”). You can email this file, or post it on a website, for other Sage users and they can use the “Upload” link on the homepage of their notebook to incorporate a copy of your worksheet into their notebook.

There are other ways to share worksheets that you can experiment with, but this gives you one way to share any worksheet with anybody almost anywhere.

We have covered a lot here in this section, so come back later to pick up tidbits you might have missed. There are also many more features in the Sage Notebook that we have not covered.

1.4 Exercises

Warm-up

This is a meaningless subdivision of the exercises for the sake of testing output.

1. Suppose that

$$\begin{aligned} A &= \{x : x \in \mathbb{N} \text{ and } x \text{ is even}\},\\ B &= \{x : x \in \mathbb{N} \text{ and } x \text{ is prime}\},\\ C &= \{x : x \in \mathbb{N} \text{ and } x \text{ is a multiple of } 5\}. \end{aligned}$$

Describe each of the following sets.

(a) $A \cap B$ (c) $A \cup B$

(b) $B \cap C$ (d) $A \cap (B \cup C)$

2. If $A = \{a, b, c\}$, $B = \{1, 2, 3\}$, $C = \{x\}$, and $D = \emptyset$, list all of the elements in each of the following sets.

(a) $A \times B$ (c) $A \times B \times C$

(b) $B \times A$ (d) $A \times D$

Hint. (a) $A \times B = \{(a, 1), (a, 2), (a, 3), (b, 1), (b, 2), (b, 3), (c, 1), (c, 2), (c, 3)\}$; (d) $A \times D = \emptyset$.

3. Find an example of two nonempty sets A and B for which $A \times B = B \times A$ is true.

4. Prove $A \cup \emptyset = A$ and $A \cap \emptyset = \emptyset$.

5. Prove $A \cup B = B \cup A$ and $A \cap B = B \cap A$.

6. Prove $A \cup (B \cap C) = (A \cup B) \cap (A \cup C)$.

Hint. If $x \in A \cup (B \cap C)$, then either $x \in A$ or $x \in B \cap C$. Thus, $x \in A \cup B$ and $A \cup C$. Hence, $x \in (A \cup B) \cap (A \cup C)$. Therefore, $A \cup (B \cap C) \subset (A \cup B) \cap (A \cup C)$. Conversely, if $x \in (A \cup B) \cap (A \cup C)$, then $x \in A \cup B$ and $A \cup C$. Thus, $x \in A$ or x is in both B and C. So $x \in A \cup (B \cap C)$ and therefore $(A \cup B) \cap (A \cup C) \subset A \cup (B \cap C)$. Hence, $A \cup (B \cap C) = (A \cup B) \cap (A \cup C)$.

7. Prove $A \cap (B \cup C) = (A \cap B) \cup (A \cap C)$.

8. Prove $A \subset B$ if and only if $A \cap B = A$.

9. Prove $(A \cap B)' = A' \cup B'$.

10. Prove $A \cup B = (A \cap B) \cup (A \setminus B) \cup (B \setminus A)$.

Hint. $(A \cap B) \cup (A \setminus B) \cup (B \setminus A) = (A \cap B) \cup (A \cap B') \cup (B \cap A') = [A \cap (B \cup B')] \cup (B \cap A') = A \cup (B \cap A') = (A \cup B) \cap (A \cup A') = A \cup B$.

11. Prove $(A \cup B) \times C = (A \times C) \cup (B \times C)$.

12. Prove $(A \cap B) \setminus B = \emptyset$.

13. Prove $(A \cup B) \setminus B = A \setminus B$.

14. Prove $A \setminus (B \cup C) = (A \setminus B) \cap (A \setminus C)$.

Hint. $A \setminus (B \cup C) = A \cap (B \cup C)' = (A \cap A) \cap (B' \cap C') = (A \cap B') \cap (A \cap C') = (A \setminus B) \cap (A \setminus C)$.

More Exercises

This is a meaningless subdivision of the exercises for the sake of testing output.

15. Prove $A \cap (B \setminus C) = (A \cap B) \setminus (A \cap C)$.

16. Prove $(A \setminus B) \cup (B \setminus A) = (A \cup B) \setminus (A \cap B)$.

17. Which of the following relations $f : \mathbb{Q} \to \mathbb{Q}$ define a mapping? In each case, supply a reason why f is or is not a mapping.

(a) $f(p/q) = \dfrac{p+1}{p-2}$

(b) $f(p/q) = \dfrac{3p}{3q}$

(c) $f(p/q) = \dfrac{p+q}{q^2}$

(d) $f(p/q) = \dfrac{3p^2}{7q^2} - \dfrac{p}{q}$

18. Determine which of the following functions are one-to-one and which are onto. If the function is not onto, determine its range.

(a) $f : \mathbb{R} \to \mathbb{R}$ defined by $f(x) = e^x$

(b) $f : \mathbb{Z} \to \mathbb{Z}$ defined by $f(n) = n^2 + 3$

(c) $f : \mathbb{R} \to \mathbb{R}$ defined by $f(x) = \sin x$

(d) $f : \mathbb{Z} \to \mathbb{Z}$ defined by $f(x) = x^2$

Hint. (a) f is one-to-one but not onto. $f(\mathbb{R}) = \{x \in \mathbb{R} : x > 0\}$. (c) f is neither one-to-one nor onto. $f(\mathbb{R}) = \{x : -1 \leq x \leq 1\}$.

19. Let $f : A \to B$ and $g : B \to C$ be invertible mappings; that is, mappings such that f^{-1} and g^{-1} exist. Show that $(g \circ f)^{-1} = f^{-1} \circ g^{-1}$.

20.

(a) Define a function $f : \mathbb{N} \to \mathbb{N}$ that is one-to-one but not onto.

(b) Define a function $f : \mathbb{N} \to \mathbb{N}$ that is onto but not one-to-one.

Hint. (a) $f(n) = n + 1$.

21. Prove the relation defined on $\mathbb{R}^2$ by $(x_1, y_1) \sim (x_2, y_2)$ if $x_1^2 + y_1^2 = x_2^2 + y_2^2$ is an equivalence relation.

22. Let $f : A \to B$ and $g : B \to C$ be maps.

(a) If f and g are both one-to-one functions, show that $g \circ f$ is one-to-one.

(b) If $g \circ f$ is onto, show that g is onto.

(c) If $g \circ f$ is one-to-one, show that f is one-to-one.

(d) If $g \circ f$ is one-to-one and f is onto, show that g is one-to-one.

(e) If $g \circ f$ is onto and g is one-to-one, show that f is onto.

Hint. (a) Let $x, y \in A$. Then $g(f(x)) = (g \circ f)(x) = (g \circ f)(y) = g(f(y))$. Thus, $f(x) = f(y)$ and $x = y$, so $g \circ f$ is one-to-one. (b) Let $c \in C$, then $c = (g \circ f)(x) = g(f(x))$ for some $x \in A$. Since $f(x) \in B$, g is onto.

23. Define a function on the real numbers by

$$f(x) = \frac{x+1}{x-1}.$$

What are the domain and range of f? What is the inverse of f? Compute $f \circ f^{-1}$ and $f^{-1} \circ f$.

24. Let $f : X \to Y$ be a map with $A_1, A_2 \subset X$ and $B_1, B_2 \subset Y$.

(a) Prove $f(A_1 \cup A_2) = f(A_1) \cup f(A_2)$.

(b) Prove $f(A_1 \cap A_2) \subset f(A_1) \cap f(A_2)$. Give an example in which equality fails.

(c) Prove $f^{-1}(B_1 \cup B_2) = f^{-1}(B_1) \cup f^{-1}(B_2)$, where

$$f^{-1}(B) = \{x \in X : f(x) \in B\}.$$

(d) Prove $f^{-1}(B_1 \cap B_2) = f^{-1}(B_1) \cap f^{-1}(B_2)$.

(e) Prove $f^{-1}(Y \setminus B_1) = X \setminus f^{-1}(B_1)$.

Hint. (a) Let $y \in f(A_1 \cup A_2)$. Then there exists an $x \in A_1 \cup A_2$ such that $f(x) = y$. Hence, $y \in f(A_1)$ or $f(A_2)$. Therefore, $y \in f(A_1) \cup f(A_2)$. Consequently, $f(A_1 \cup A_2) \subset f(A_1) \cup f(A_2)$. Conversely, if $y \in f(A_1) \cup f(A_2)$, then $y \in f(A_1)$ or $f(A_2)$. Hence, there exists an $x \in A_1$ or there exists an $x \in A_2$ such that $f(x) = y$. Thus, there exists an $x \in A_1 \cup A_2$ such that $f(x) = y$. Therefore, $f(A_1) \cup f(A_2) \subset f(A_1 \cup A_2)$, and $f(A_1 \cup A_2) = f(A_1) \cup f(A_2)$.

25. Determine whether or not the following relations are equivalence relations on the given set. If the relation is an equivalence relation, describe the partition given by it. If the relation is not an equivalence relation, state why it fails to be one.

(a) $x \sim y$ in $\mathbb{R}$ if $x \geq y$

(b) $m \sim n$ in $\mathbb{Z}$ if $mn > 0$

(c) $x \sim y$ in $\mathbb{R}$ if $|x - y| \leq 4$

(d) $m \sim n$ in $\mathbb{Z}$ if $m \equiv n \pmod 6$

26. Define a relation $\sim$ on $\mathbb{R}^2$ by stating that $(a, b) \sim (c, d)$ if and only if $a^2 + b^2 \leq c^2 + d^2$. Show that $\sim$ is reflexive and transitive but not symmetric.

27. Show that an $m \times n$ matrix gives rise to a well-defined map from $\mathbb{R}^n$ to $\mathbb{R}^m$.

28. Find the error in the following argument by providing a counterexample. "The reflexive property is redundant in the axioms for an equivalence relation. If $x \sim y$, then $y \sim x$ by the symmetric property. Using the transitive property, we can deduce that $x \sim x$."

Hint. Let $X = \mathbb{N} \cup \{\sqrt{2}\}$ and define $x \sim y$ if $x + y \in \mathbb{N}$.

29. **Projective Real Line.** Define a relation on $\mathbb{R}^2 \setminus \{(0, 0)\}$ by letting $(x_1, y_1) \sim (x_2, y_2)$ if there exists a nonzero real number λ such that

$(x_1, y_1) = (\lambda x_2, \lambda y_2)$. Prove that $\sim$ defines an equivalence relation on $\mathbb{R}^2 \setminus (0, 0)$. What are the corresponding equivalence classes? This equivalence relation defines the projective line, denoted by $\mathbb{P}(\mathbb{R})$, which is very important in geometry.

1.5 Sage Exercises

1. This exercise is just about making sure you know how to use Sage. Login to a Sage Notebook server and create a new worksheet. Do some non-trivial computation, maybe a pretty plot or some gruesome numerical computation to an insane precision. Create an interesting list and experiment with it some. Maybe include some nicely formatted text or $\TeX$ using the included mini-word-processor of the Sage Notebook (hover until a blue bar appears between cells and then shift-click).

Use whatever mechanism your instructor has in place for submitting your work. Or save your worksheet and then trade worksheets via email (or another electronic method) with a classmate.

1.6 References and Suggested Readings

[1] Artin, M. *Abstract Algebra.* 2nd ed. Pearson, Upper Saddle River, NJ, 2011.

[2] Childs, L. *A Concrete Introduction to Higher Algebra.* 2nd ed. Springer-Verlag, New York, 1995.

[3] Dummit, D. and Foote, R. *Abstract Algebra.* 3rd ed. Wiley, New York, 2003.

[4] Ehrlich, G. *Fundamental Concepts of Algebra.* PWS-KENT, Boston, 1991.

[5] Fraleigh, J. B. *A First Course in Abstract Algebra.* 7th ed. Pearson, Upper Saddle River, NJ, 2003.

[6] Gallian, J. A. *Contemporary Abstract Algebra.* 7th ed. Brooks/Cole, Belmont, CA, 2009.

[7] Halmos, P. *Naive Set Theory.* Springer, New York, 1991. One of the best references for set theory.

[8] Herstein, I. N. *Abstract Algebra.* 3rd ed. Wiley, New York, 1996.

[9] Hungerford, T. W. *Algebra.* Springer, New York, 1974. One of the standard graduate algebra texts.

[10] Lang, S. *Algebra.* 3rd ed. Springer, New York, 2002. Another standard graduate text.

[11] Lidl, R. and Pilz, G. *Applied Abstract Algebra.* 2nd ed. Springer, New York, 1998.

[12] Mackiw, G. *Applications of Abstract Algebra.* Wiley, New York, 1985.

[13] Nickelson, W. K. *Introduction to Abstract Algebra.* 3rd ed. Wiley, New York, 2006.

[14] Solow, D. *How to Read and Do Proofs.* 5th ed. Wiley, New York, 2009.

[15] van der Waerden, B. L. *A History of Algebra.* Springer-Verlag, New York, 1985. An account of the historical development of algebra.

Chapter 2

The Integers

The integers are the building blocks of mathematics. In this chapter we will investigate the fundamental properties of the integers, including mathematical induction, the division algorithm, and the Fundamental Theorem of Arithmetic.

2.1 Mathematical Induction and Math in a Title $A \not\subset B$

Suppose we wish to show that

$$1 + 2 + \cdots + n = \frac{n(n+1)}{2}$$

for any natural number n. This formula is easily verified for small numbers such as $n = 1$, 2, 3, or 4, but it is impossible to verify for all natural numbers on a case-by-case basis. To prove the formula true in general, a more generic method is required.

Suppose we have verified the equation for the first n cases. We will attempt to show that we can generate the formula for the $(n+1)$th case from this knowledge. The formula is true for $n = 1$ since

$$1 = \frac{1(1+1)}{2}.$$

If we have verified the first n cases, then

$$\begin{aligned} 1 + 2 + \cdots + n + (n+1) &= \frac{n(n+1)}{2} + n + 1 \\ &= \frac{n^2 + 3n + 2}{2} \\ &= \frac{(n+1)[(n+1)+1]}{2}. \end{aligned}$$

This is exactly the formula for the $(n+1)$th case.

This method of proof is known as **mathematical induction**. Instead of attempting to verify a statement about some subset S of the positive integers $\mathbb{N}$ on a case-by-case basis, an impossible task if S is an infinite set, we give a specific proof for the smallest integer being considered, followed by a generic argument showing that if the statement holds for a given case, then it must also hold for the next case in the sequence. We summarize mathematical induction in the following axiom.

Principle 2.1.1 First Principle of Mathematical Induction. *Let $S(n)$ be a statement about integers for $n \in \mathbb{N}$ and suppose $S(n_0)$ is true for some integer n_0. If for all integers k with $k \geq n_0$, $S(k)$ implies that $S(k+1)$ is true, then $S(n)$ is true for all integers n greater than or equal to n_0.*

Example 2.1.2 An Inequality for Powers of 2. For all integers $n \geq 3$, $2^n > n + 4$. Since

$$8 = 2^3 > 3 + 4 = 7,$$

the statement is true for $n_0 = 3$. Assume that $2^k > k + 4$ for $k \geq 3$. Then $2^{k+1} = 2 \cdot 2^k > 2(k+4)$. But

$$2(k+4) = 2k + 8 > k + 5 = (k+1) + 4$$

since k is positive. Hence, by induction, the statement holds for all integers $n \geq 3$. □

Example 2.1.3 Some Integers Divisible by 9. Every integer $10^{n+1} + 3 \cdot 10^n + 5$ is divisible by 9 for $n \in \mathbb{N}$. For $n = 1$,

$$10^{1+1} + 3 \cdot 10 + 5 = 135 = 9 \cdot 15$$

is divisible by 9. Suppose that $10^{k+1} + 3 \cdot 10^k + 5$ is divisible by 9 for $k \geq 1$. Then

$$\begin{aligned} 10^{(k+1)+1} + 3 \cdot 10^{k+1} + 5 &= 10^{k+2} + 3 \cdot 10^{k+1} + 50 - 45 \\ &= 10(10^{k+1} + 3 \cdot 10^k + 5) - 45 \end{aligned}$$

is divisible by 9. □

Example 2.1.4 The Binomial Theorem. We will prove the binomial theorem using mathematical induction; that is,

$$(a+b)^n = \sum_{k=0}^{n} \binom{n}{k} a^k b^{n-k},$$

where a and b are real numbers, $n \in \mathbb{N}$, and

$$\binom{n}{k} = \frac{n!}{k!(n-k)!}$$

is the binomial coefficient. We first show that

$$\binom{n+1}{k} = \binom{n}{k} + \binom{n}{k-1}.$$

This result follows from

$$\begin{aligned} \binom{n}{k} + \binom{n}{k-1} &= \frac{n!}{k!(n-k)!} + \frac{n!}{(k-1)!(n-k+1)!} \\ &= \frac{(n+1)!}{k!(n+1-k)!} \\ &= \binom{n+1}{k}. \end{aligned}$$

If $n = 1$, the binomial theorem is easy to verify. Now assume that the result is true for n greater than or equal to 1. Then

$$(a+b)^{n+1} = (a+b)(a+b)^n$$

$$
\begin{aligned}
&= (a+b)\left(\sum_{k=0}^{n}\binom{n}{k}a^k b^{n-k}\right)\\
&= \sum_{k=0}^{n}\binom{n}{k}a^{k+1}b^{n-k} + \sum_{k=0}^{n}\binom{n}{k}a^k b^{n+1-k}\\
&= a^{n+1} + \sum_{k=1}^{n}\binom{n}{k-1}a^k b^{n+1-k} + \sum_{k=1}^{n}\binom{n}{k}a^k b^{n+1-k} + b^{n+1}\\
&= a^{n+1} + \sum_{k=1}^{n}\left[\binom{n}{k-1} + \binom{n}{k}\right]a^k b^{n+1-k} + b^{n+1}\\
&= \sum_{k=0}^{n+1}\binom{n+1}{k}a^k b^{n+1-k}.
\end{aligned}
$$

□

We have an equivalent statement of the Principle of Mathematical Induction that is often very useful.

Principle 2.1.5 Second Principle of Mathematical Induction. *Let $S(n)$ be a statement about integers for $n \in \mathbb{N}$ and suppose $S(n_0)$ is true for some integer n_0. If $S(n_0), S(n_0+1), \ldots, S(k)$ imply that $S(k+1)$ for $k \geq n_0$, then the statement $S(n)$ is true for all integers $n \geq n_0$.*

A nonempty subset S of $\mathbb{Z}$ is **well-ordered** if S contains a least element. Notice that the set $\mathbb{Z}$ is not well-ordered since it does not contain a smallest element. However, the natural numbers are well-ordered.

Principle 2.1.6 Principle of Well-Ordering. *Every nonempty subset of the natural numbers is well-ordered.*

The Principle of Well-Ordering is equivalent to the Principle of Mathematical Induction.

Lemma 2.1.7 *The Principle of Mathematical Induction implies that 1 is the least positive natural number.*

Proof. Let $S = \{n \in \mathbb{N} : n \geq 1\}$. Then $1 \in S$. Now assume that $n \in S$; that is, $n \geq 1$. Since $n+1 \geq 1$, $n+1 \in S$; hence, by induction, every natural number is greater than or equal to 1. ■

Theorem 2.1.8 *The Principle of Mathematical Induction implies the Principle of Well-Ordering. That is, every nonempty subset of $\mathbb{N}$ contains a least element.*

Proof. We must show that if S is a nonempty subset of the natural numbers, then S contains a least element. If S contains 1, then the theorem is true by Lemma 2.1.7. Assume that if S contains an integer k such that $1 \leq k \leq n$, then S contains a least element. We will show that if a set S contains an integer less than or equal to $n+1$, then S has a least element. If S does not contain an integer less than $n+1$, then $n+1$ is the smallest integer in S. Otherwise, since S is nonempty, S must contain an integer less than or equal to n. In this case, by induction, S contains a least element. ■

Induction can also be very useful in formulating definitions. For instance, there are two ways to define $n!$, the factorial of a positive integer n.

- The *explicit* definition: $n! = 1 \cdot 2 \cdot 3 \cdots (n-1) \cdot n$.
- The *inductive* or *recursive* definition: $1! = 1$ and $n! = n(n-1)!$ for $n > 1$.

Every good mathematician or computer scientist knows that looking at problems recursively, as opposed to explicitly, often results in better understanding of complex issues.

2.2 The Division Algorithm

An application of the Principle of Well-Ordering that we will use often is the division algorithm.

Theorem 2.2.1 Division Algorithm. *Let a and b be integers, with $b > 0$. Then there exist unique integers q and r such that*

$$a = bq + r$$

where $0 \leq r < b$.
Proof. This is a perfect example of the existence-and-uniqueness type of proof. We must first prove that the numbers q and r actually exist. Then we must show that if q' and r' are two other such numbers, then $q = q'$ and $r = r'$.
Existence of q and r. Let

$$S = \{a - bk : k \in \mathbb{Z} \text{ and } a - bk \geq 0\}.$$

If $0 \in S$, then b divides a, and we can let $q = a/b$ and $r = 0$. If $0 \notin S$, we can use the Well-Ordering Principle. We must first show that S is nonempty. If $a > 0$, then $a - b \cdot 0 \in S$. If $a < 0$, then $a - b(2a) = a(1 - 2b) \in S$. In either case $S \neq \emptyset$. By the Well-Ordering Principle, S must have a smallest member, say $r = a - bq$. Therefore, $a = bq + r$, $r \geq 0$. We now show that $r < b$. Suppose that $r > b$. Then

$$a - b(q + 1) = a - bq - b = r - b > 0.$$

In this case we would have $a - b(q+1)$ in the set S. But then $a - b(q+1) < a - bq$, which would contradict the fact that $r = a - bq$ is the smallest member of S. So $r \leq b$. Since $0 \notin S$, $r \neq b$ and so $r < b$.
Uniqueness of q and r. Suppose there exist integers r, r', q, and q' such that

$$a = bq + r, 0 \leq r < b \quad \text{and} \quad a = bq' + r', 0 \leq r' < b.$$

Then $bq + r = bq' + r'$. Assume that $r' \geq r$. From the last equation we have $b(q - q') = r' - r$; therefore, b must divide $r' - r$ and $0 \leq r' - r \leq r' < b$. This is possible only if $r' - r = 0$. Hence, $r = r'$ and $q = q'$. ■

Let a and b be integers. If $b = ak$ for some integer k, we write $a \mid b$. An integer d is called a **common divisor** of a and b if $d \mid a$ and $d \mid b$. The **greatest common divisor** of integers a and b is a positive integer d such that d is a common divisor of a and b and if d' is any other common divisor of a and b, then $d' \mid d$. We write $d = \gcd(a, b)$; for example, $\gcd(24, 36) = 12$ and $\gcd(120, 102) = 6$. We say that two integers a and b are **relatively prime** if $\gcd(a, b) = 1$.

Theorem 2.2.2 *Let a and b be nonzero integers. Then there exist integers r and s such that*

$$\gcd(a, b) = ar + bs.$$

Furthermore, the greatest common divisor of a and b is unique.
Proof. Let

$$S = \{am + bn : m, n \in \mathbb{Z} \text{ and } am + bn > 0\}.$$

Clearly, the set S is nonempty; hence, by the Well-Ordering Principle S must have a smallest member, say $d = ar + bs$. We claim that $d = \gcd(a, b)$. Write $a = dq + r'$ where $0 \leq r' < d$. If $r' > 0$, then

$$r' = a - dq$$

$$\begin{aligned}
&= a - (ar + bs)q \\
&= a - arq - bsq \\
&= a(1 - rq) + b(-sq),
\end{aligned}$$

which is in S. But this would contradict the fact that d is the smallest member of S. Hence, $r' = 0$ and d divides a. A similar argument shows that d divides b. Therefore, d is a common divisor of a and b.
Suppose that d' is another common divisor of a and b, and we want to show that $d' \mid d$. If we let $a = d'h$ and $b = d'k$, then

$$d = ar + bs = d'hr + d'ks = d'(hr + ks).$$

So d' must divide d. Hence, d must be the unique greatest common divisor of a and b. ■

Corollary 2.2.3 *Let a and b be two integers that are relatively prime. Then there exist integers r and s such that $ar + bs = 1$.*

2.2.1 The Euclidean Algorithm

Among other things, Theorem 2.2.2 allows us to compute the greatest common divisor of two integers.

Example 2.2.4 Greatest Common Divisor of Two Integers. Let us compute the greatest common divisor of 945 and 2415. First observe that

$$\begin{aligned}
2415 &= 945 \cdot 2 + 525 \\
945 &= 525 \cdot 1 + 420 \\
525 &= 420 \cdot 1 + 105 \\
420 &= 105 \cdot 4 + 0.
\end{aligned}$$

Reversing our steps, 105 divides 420, 105 divides 525, 105 divides 945, and 105 divides 2415. Hence, 105 divides both 945 and 2415. If d were another common divisor of 945 and 2415, then d would also have to divide 105. Therefore, $\gcd(945, 2415) = 105$.

If we work backward through the above sequence of equations, we can also obtain numbers r and s such that $945r + 2415s = 105$. Observe that

$$\begin{aligned}
105 &= 525 + (-1) \cdot 420 \\
&= 525 + (-1) \cdot [945 + (-1) \cdot 525] \\
&= 2 \cdot 525 + (-1) \cdot 945 \\
&= 2 \cdot [2415 + (-2) \cdot 945] + (-1) \cdot 945 \\
&= 2 \cdot 2415 + (-5) \cdot 945.
\end{aligned}$$

So $r = -5$ and $s = 2$. Notice that r and s are not unique, since $r = 41$ and $s = -16$ would also work. □

To compute $\gcd(a, b) = d$, we are using repeated divisions to obtain a decreasing sequence of positive integers $r_1 > r_2 > \cdots > r_n = d$; that is,

$$\begin{aligned}
b &= aq_1 + r_1 \\
a &= r_1 q_2 + r_2 \\
r_1 &= r_2 q_3 + r_3 \\
&\vdots
\end{aligned}$$

$$\begin{aligned} r_{n-2} &= r_{n-1}q_n + r_n \\ r_{n-1} &= r_n q_{n+1}. \end{aligned}$$

To find r and s such that $ar + bs = d$, we begin with this last equation and substitute results obtained from the previous equations:

$$\begin{aligned} d &= r_n \\ &= r_{n-2} - r_{n-1}q_n \\ &= r_{n-2} - q_n(r_{n-3} - q_{n-1}r_{n-2}) \\ &= -q_n r_{n-3} + (1 + q_n q_{n-1})r_{n-2} \\ &\vdots \\ &= ra + sb. \end{aligned}$$

The algorithm that we have just used to find the greatest common divisor d of two integers a and b and to write d as the linear combination of a and b is known as the **Euclidean algorithm**.

2.2.2 Prime Numbers

Let p be an integer such that $p > 1$. We say that p is a **prime number**, or simply p is **prime**, if the only positive numbers that divide p are 1 and p itself. An integer $n > 1$ that is not prime is said to be **composite**.

Lemma 2.2.5 (Euclid) *Let a and b be integers and p be a prime number. If $p \mid ab$, then either $p \mid a$ or $p \mid b$.*

Proof. Suppose that p does not divide a. We must show that $p \mid b$. Since $\gcd(a, p) = 1$, there exist integers r and s such that $ar + ps = 1$. So

$$b = b(ar + ps) = (ab)r + p(bs).$$

Since p divides both ab and itself, p must divide $b = (ab)r + p(bs)$. ■

Theorem 2.2.6 (Euclid) *There exist an infinite number of primes.*

Proof. We will prove this theorem by contradiction. Suppose that there are only a finite number of primes, say $p_1, p_2, \ldots, p_n$. Let $P = p_1 p_2 \cdots p_n + 1$. Then P must be divisible by some p_i for $1 \leq i \leq n$. In this case, p_i must divide $P - p_1 p_2 \cdots p_n = 1$, which is a contradiction. Hence, either P is prime or there exists an additional prime number $p \neq p_i$ that divides P. ■

Theorem 2.2.7 Fundamental Theorem of Arithmetic. *Let n be an integer such that $n > 1$. Then*

$$n = p_1 p_2 \cdots p_k,$$

where $p_1, \ldots, p_k$ are primes (not necessarily distinct). Furthermore, this factorization is unique; that is, if

$$n = q_1 q_2 \cdots q_l,$$

then $k = l$ and the q_i's are just the p_i's rearranged.

Proof. Uniqueness. To show uniqueness we will use induction on n. The theorem is certainly true for $n = 2$ since in this case n is prime. Now assume that the result holds for all integers m such that $1 \leq m < n$, and

$$n = p_1 p_2 \cdots p_k = q_1 q_2 \cdots q_l,$$

where $p_1 \leq p_2 \leq \cdots \leq p_k$ and $q_1 \leq q_2 \leq \cdots \leq q_l$. By Lemma 2.2.5, $p_1 \mid q_i$ for some $i = 1, \ldots, l$ and $q_1 \mid p_j$ for some $j = 1, \ldots, k$. Since all of the p_i's and q_i's are prime, $p_1 = q_i$ and $q_1 = p_j$. Hence, $p_1 = q_1$ since $p_1 \leq p_j = q_1 \leq q_i = p_1$. By the induction hypothesis,

$$n' = p_2 \cdots p_k = q_2 \cdots q_l$$

has a unique factorization. Hence, $k = l$ and $q_i = p_i$ for $i = 1, \ldots, k$.
Existence. To show existence, suppose that there is some integer that cannot be written as the product of primes. Let S be the set of all such numbers. By the Principle of Well-Ordering, S has a smallest number, say a. If the only positive factors of a are a and 1, then a is prime, which is a contradiction. Hence, $a = a_1 a_2$ where $1 < a_1 < a$ and $1 < a_2 < a$. Neither $a_1 \in S$ nor $a_2 \in S$, since a is the smallest element in S. So

$$\begin{aligned} a_1 &= p_1 \cdots p_r \\ a_2 &= q_1 \cdots q_s. \end{aligned}$$

Therefore,

$$a = a_1 a_2 = p_1 \cdots p_r q_1 \cdots q_s.$$

So $a \notin S$, which is a contradiction. ■

2.2.3 Historical Note

Prime numbers were first studied by the ancient Greeks. Two important results from antiquity are Euclid's proof that an infinite number of primes exist and the Sieve of Eratosthenes, a method of computing all of the prime numbers less than a fixed positive integer n. One problem in number theory is to find a function f such that $f(n)$ is prime for each integer n. Pierre Fermat (1601?–1665) conjectured that $2^{2^n} + 1$ was prime for all n, but later it was shown by Leonhard Euler (1707–1783) that

$$2^{2^5} + 1 = 4{,}294{,}967{,}297$$

is a composite number. One of the many unproven conjectures about prime numbers is Goldbach's Conjecture. In a letter to Euler in 1742, Christian Goldbach stated the conjecture that every even integer with the exception of 2 seemed to be the sum of two primes: $4 = 2 + 2$, $6 = 3 + 3$, $8 = 3 + 5$, $\ldots$. Although the conjecture has been verified for the numbers up through 4×10^{18}, it has yet to be proven in general. Since prime numbers play an important role in public key cryptography, there is currently a great deal of interest in determining whether or not a large number is prime.

Remark 2.2.8 Sage. Sage's original purpose was to support research in number theory, so it is perfect for the types of computations with the integers that we have in this chapter.

2.3 Sage

Many properties of the algebraic objects we will study can be determined from properties of associated integers. And Sage has many powerful functions for analyzing integers.

2.3.1 Division Algorithm

The code `a % b` will return the remainder upon division of a by b. In other words, the result is the unique integer r such that (1) $0 \leq r < b$, and (2) $a = bq + r$ for some integer q (the quotient), as guaranteed by the Division Algorithm (Theorem 2.2.1). Then $(a - r)/b$ will equal q. For example,

```
r = 14 % 3
r
```

```
2
```

```
q = (14 - r)/3
q
```

```
4
```

It is also possible to get both the quotient and remainder at the same time with the `.quo_rem()` method (quotient and remainder).

```
a = 14
b = 3
a.quo_rem(b)
```

```
(4, 2)
```

A remainder of zero indicates divisibility. So `(a % b) == 0` will return `True` if b divides a, and will otherwise return `False`.

```
(20 % 5) == 0
```

```
True
```

```
(17 % 4) == 0
```

```
False
```

The `.divides()` method is another option.

```
c = 5
c.divides(20)
```

```
True
```

```
d = 4
d.divides(17)
```

```
False
```

2.3.2 Greatest Common Divisor

The greatest common divisor of a and b is obtained with the command `gcd(a, b)`, where in our first uses, a and b are integers. Later, a and b can be other objects with a notion of divisibility and "greatness," such as polynomials. For example,

```
gcd(2776, 2452)
```

```
4
```

We can use the `gcd` command to determine if a pair of integers are relatively prime.

```
a = 31049
b = 2105
gcd(a, b) == 1
```

```
True
```

```
a = 3563
b = 2947
gcd(a, b) == 1
```

```
False
```

The command `xgcd(a,b)` (“eXtended GCD”) returns a triple where the first element is the greatest common divisor of a and b (as with the `gcd(a,b)` command above), but the next two elements are values of r and s such that $ra + sb = \gcd(a, b)$.

```
xgcd(633,331)
```

```
(1, -137, 262)
```

Portions of the triple can be extracted using `[ ]` (“indexing”) to access the entries of the triple, starting with the first as number `0`. For example, the following should always return the result `True`, even if you change the values of `a` and `b`. Try changing the values of `a` and `b` below, to see that the result is always `True`.

```
a = 633
b = 331
extended = xgcd(a, b)
g = extended[0]
r = extended[1]
s = extended[2]
g == r*a + s*b
```

```
True
```

Studying this block of code will go a long way towards helping you get the most out of Sage’s output. Note that `=` is how a value is *assigned* to a variable, while as in the last line, `==` is how we compare two items for *equality*.

2.3.3 Primes and Factoring

The method `.is_prime()` will determine if an integer is prime or not.

```
a = 117371
a.is_prime()
```

```
True
```

```
b = 14547073
b.is_prime()
```

```
False
```

```
b == 1597 * 9109
```

```
True
```

The command `random_prime(a, proof=True)` will generate a random prime number between 2 and a. Experiment by executing the following two compute cells several times. (Replacing `proof=True` by `proof=False` will speed up the search, but there will be a very, very, very small probability the result will not be prime.)

```
a = random_prime(10^21, proof=True)
a
```

```
424729101793542195193
```

```
a.is_prime()
```

```
True
```

The command `prime_range(a, b)` returns an ordered list of all the primes from a to $b-1$, inclusive. For example,

```
prime_range(500, 550)
```

```
[503, 509, 521, 523, 541, 547]
```

The commands `next_prime(a)` and `previous_prime(a)` are other ways to get a single prime number of a desired size. Give them a try below if you have an empty compute cell there (as you will if you are reading in the Sage Notebook, or are reading the online version). (The hash symbol, `#`, is used to indicate a “comment” line, which will not be evaluated by Sage. So erase this line, or start on the one below it.)In addition to checking if integers are prime or not, or generating prime numbers, Sage can also decompose any integer into its prime factors, as described by the Fundamental Theorem of Arithmetic (Theorem 2.2.7).

```
a = 2600
a.factor()
```

```
2^3 * 5^2 * 13
```

So $2600 = 2^3 \times 5^2 \times 13$ and this is the unique way to write 2600 as a product of prime numbers (other than rearranging the order of the primes themselves in the product).

While Sage will print a factorization nicely, it is carried internally as a list of pairs of integers, with each pair being a base (a prime number) and an exponent (a positive integer). Study the following carefully, as it is another good exercise in working with Sage output in the form of lists.

```
a = 2600
factored = a.factor()
first_term = factored[0]
first_term
```

```
(2, 3)
```

```
second_term = factored[1]
second_term
```

```
(5, 2)
```

```
third_term = factored[2]
third_term
```

```
(13, 1)
```

```
first_prime = first_term[0]
first_prime
```

```
2
```

```
first_exponent = first_term[1]
first_exponent
```

```
3
```

The next compute cell reveals the internal version of the factorization by asking for the actual list. And we show how you could determine exactly how many terms the factorization has by using the length command, `len()`.

```
list(factored)
```

```
[(2, 3), (5, 2), (13, 1)]
```

```
len(factored)
```

```
3
```

Can you extract the next two primes, and their exponents, from `a`?

2.4 Exercises

1. Prove that

$$1^2 + 2^2 + \cdots + n^2 = \frac{n(n+1)(2n+1)}{6}$$

for $n \in \mathbb{N}$.

Answer. The base case, $S(1) : [1(1+1)(2(1)+1)]/6 = 1 = 1^2$ is true.

Assume that $S(k) : 1^2 + 2^2 + \cdots + k^2 = [k(k+1)(2k+1)]/6$ is true. Then

$$\begin{aligned} 1^2 + 2^2 + \cdots + k^2 + (k+1)^2 &= [k(k+1)(2k+1)]/6 + (k+1)^2 \\ &= [(k+1)((k+1)+1)(2(k+1)+1)]/6, \end{aligned}$$

and so $S(k+1)$ is true. Thus, $S(n)$ is true for all positive integers n.

2. Prove that

$$1^3 + 2^3 + \cdots + n^3 = \frac{n^2(n+1)^2}{4}$$

for $n \in \mathbb{N}$.

3. Prove that $n! > 2^n$ for $n \geq 4$.

Answer. The base case, $S(4) : 4! = 24 > 16 = 2^4$ is true. Assume $S(k) : k! > 2^k$ is true. Then $(k+1)! = k!(k+1) > 2^k \cdot 2 = 2^{k+1}$, so $S(k+1)$ is true. Thus, $S(n)$ is true for all positive integers n.

4. Prove that

$$x + 4x + 7x + \cdots + (3n-2)x = \frac{n(3n-1)x}{2}$$

for $n \in \mathbb{N}$.

5. Prove that $10^{n+1} + 10^n + 1$ is divisible by 3 for $n \in \mathbb{N}$.

6. Prove that $4 \cdot 10^{2n} + 9 \cdot 10^{2n-1} + 5$ is divisible by 99 for $n \in \mathbb{N}$.

7. Show that

$$\sqrt[n]{a_1 a_2 \cdots a_n} \leq \frac{1}{n} \sum_{k=1}^{n} a_k.$$

8. Use induction to prove that $1 + 2 + 2^2 + \cdots + 2^n = 2^{n+1} - 1$ for $n \in \mathbb{N}$.

9. Prove the Leibniz rule for $f^{(n)}(x)$, where $f^{(n)}$ is the nth derivative of f; that is, show that

$$(fg)^{(n)}(x) = \sum_{k=0}^{n} \binom{n}{k} f^{(k)}(x) g^{(n-k)}(x).$$

Hint. Follow the proof in Example 2.1.4.

10. Prove that

$$\frac{1}{2} + \frac{1}{6} + \cdots + \frac{1}{n(n+1)} = \frac{n}{n+1}$$

for $n \in \mathbb{N}$.

11. If x is a nonnegative real number, then show that $(1+x)^n - 1 \geq nx$ for $n = 0, 1, 2, \ldots$.

Hint. The base case, $S(0) : (1+x)^0 - 1 = 0 \geq 0 = 0 \cdot x$ is true. Assume $S(k) : (1+x)^k - 1 \geq kx$ is true. Then

$$\begin{aligned} (1+x)^{k+1} - 1 &= (1+x)(1+x)^k - 1 \\ &= (1+x)^k + x(1+x)^k - 1 \\ &\geq kx + x(1+x)^k \\ &\geq kx + x \\ &= (k+1)x, \end{aligned}$$

so $S(k+1)$ is true. Therefore, $S(n)$ is true for all positive integers n.

12. **Power Sets.** Let X be a set. Define the **power set** of X, denoted $\mathcal{P}(X)$, to be the set of all subsets of X. For example,

$$\mathcal{P}(\{a, b\}) = \{\emptyset, \{a\}, \{b\}, \{a, b\}\}.$$

For every positive integer n, show that a set with exactly n elements has a power set with exactly 2^n elements.

13. Prove that the two principles of mathematical induction stated in Section 2.1 are equivalent.

14. Show that the Principle of Well-Ordering for the natural numbers implies that 1 is the smallest natural number. Use this result to show that the Principle of Well-Ordering implies the Principle of Mathematical Induction; that is, show that if $S \subset \mathbb{N}$ such that $1 \in S$ and $n + 1 \in S$ whenever $n \in S$, then $S = \mathbb{N}$.

15. For each of the following pairs of numbers a and b, calculate $\gcd(a, b)$ and find integers r and s such that $\gcd(a, b) = ra + sb$.

(a) 14 and 39

(b) 234 and 165

(c) 1739 and 9923

(d) 471 and 562

(e) 23,771 and 19,945

(f) -4357 and 3754

16. Let a and b be nonzero integers. If there exist integers r and s such that $ar + bs = 1$, show that a and b are relatively prime.

17. **Fibonacci Numbers.** The Fibonacci numbers are

$$1, 1, 2, 3, 5, 8, 13, 21, \ldots .$$

We can define them inductively by $f_1 = 1$, $f_2 = 1$, and $f_{n+2} = f_{n+1} + f_n$ for $n \in \mathbb{N}$.

(a) Prove that $f_n < 2^n$.

(b) Prove that $f_{n+1}f_{n-1} = f_n^2 + (-1)^n$, $n \geq 2$.

(c) Prove that $f_n = [(1 + \sqrt{5}\,)^n - (1 - \sqrt{5}\,)^n]/2^n\sqrt{5}$.

(d) Show that $\lim_{n\to\infty} f_n/f_{n+1} = (\sqrt{5} - 1)/2$.

(e) Prove that f_n and f_{n+1} are relatively prime.

Hint. For Item 2.4.17.a and Item 2.4.17.b use mathematical induction. Item 2.4.17.c Show that $f_1 = 1$, $f_2 = 1$, and $f_{n+2} = f_{n+1} + f_n$. Item 2.4.17.d Use part Item 2.4.17.c. Item 2.4.17.e Use part Item 2.4.17.b and Exercise 2.4.16.

18. Let a and b be integers such that $\gcd(a, b) = 1$. Let r and s be integers such that $ar + bs = 1$. Prove that

$$\gcd(a, s) = \gcd(r, b) = \gcd(r, s) = 1.$$

19. Let $x, y \in \mathbb{N}$ be relatively prime. If xy is a perfect square, prove that x and y must both be perfect squares.

Hint. Use the Fundamental Theorem of Arithmetic.

20. Using the division algorithm, show that every perfect square is of the form $4k$ or $4k + 1$ for some nonnegative integer k.

21. Suppose that a, b, r, s are pairwise relatively prime and that

$$a^2 + b^2 = r^2$$
$$a^2 - b^2 = s^2.$$

Prove that a, r, and s are odd and b is even.

22. Let $n \in \mathbb{N}$. Use the division algorithm to prove that every integer is congruent mod n to precisely one of the integers $0, 1, \ldots, n-1$. Conclude that if r is an integer, then there is exactly one s in $\mathbb{Z}$ such that $0 \leq s < n$ and $[r] = [s]$. Hence, the integers are indeed partitioned by congruence mod n.

23. Define the **least common multiple** of two nonzero integers a and b, denoted by $\operatorname{lcm}(a, b)$, to be the nonnegative integer m such that both a and b divide m, and if a and b divide any other integer n, then m also divides n. Prove that any two integers a and b have a unique least common multiple.

Hint. Let $S = \{s \in \mathbb{N} : a \mid s,\ b \mid s\}$. Then $S \neq \emptyset$, since $|ab| \in S$. By the Principle of Well-Ordering, S contains a least element m. To show uniqueness, suppose that $a \mid n$ and $b \mid n$ for some $n \in \mathbb{N}$. By the division algorithm, there exist unique integers q and r such that $n = mq + r$, where $0 \leq r < m$. Since a and b divide both m, and n, it must be the case that a and b both divide r. Thus, $r = 0$ by the minimality of m. Therefore, $m \mid n$.

24. If $d = \gcd(a, b)$ and $m = \operatorname{lcm}(a, b)$, prove that $dm = |ab|$.

25. Show that $\operatorname{lcm}(a, b) = ab$ if and only if $\gcd(a, b) = 1$.

26. Prove that $\gcd(a, c) = \gcd(b, c) = 1$ if and only if $\gcd(ab, c) = 1$ for integers a, b, and c.

27. Let $a, b, c \in \mathbb{Z}$. Prove that if $\gcd(a, b) = 1$ and $a \mid bc$, then $a \mid c$.

Hint. Since $\gcd(a, b) = 1$, there exist integers r and s such that $ar + bs = 1$. Thus, $acr + bcs = c$. Since a divides both bc and itself, a must divide c.

28. Let $p \geq 2$. Prove that if $2^p - 1$ is prime, then p must also be prime.

29. Prove that there are an infinite number of primes of the form $6n + 5$.

Hint. Every prime must be of the form 2, 3, $6n + 1$, or $6n + 5$. Suppose there are only finitely many primes of the form $6k + 5$.

30. Prove that there are an infinite number of primes of the form $4n - 1$.

31. Using the fact that 2 is prime, show that there do not exist integers p and q such that $p^2 = 2q^2$. Demonstrate that therefore $\sqrt{2}$ cannot be a rational number.

2.5 Programming Exercises

1. **The Sieve of Eratosthenes.** One method of computing all of the prime numbers less than a certain fixed positive integer N is to list all of the numbers n such that $1 < n < N$. Begin by eliminating all of the multiples of 2. Next eliminate all of the multiples of 3. Now eliminate all of the multiples of 5. Notice that 4 has already been crossed out. Continue in this manner, noticing that we do not have to go all the way to N; it suffices to stop at $\sqrt{N}$. Using this method, compute all of the prime numbers less than $N = 250$. We can also use this method to find all of the integers that are relatively prime to an integer N. Simply eliminate the prime factors of N and all of their multiples. Using this method, find all of the numbers that are relatively prime to $N = 120$. Using the Sieve of Eratosthenes, write a program that will compute all of the primes less than an integer N.

2. Let $\mathbb{N}^0 = \mathbb{N} \cup \{0\}$. Ackermann's function is the function $A : \mathbb{N}^0 \times \mathbb{N}^0 \to \mathbb{N}^0$ defined by the equations

$$\begin{aligned} A(0, y) &= y + 1 \\ A(x + 1, 0) &= A(x, 1) \\ A(x + 1, y + 1) &= A(x, A(x + 1, y)). \end{aligned}$$

Use this definition to compute $A(3, 1)$. Write a program to evaluate Ackermann's function. Modify the program to count the number of statements executed in the program when Ackermann's function is evaluated. How many statements are executed in the evaluation of $A(4, 1)$? What about $A(5, 1)$?

3. Write a computer program that will implement the Euclidean algorithm. The program should accept two positive integers a and b as input and should output $\gcd(a, b)$ as well as integers r and s such that

$$\gcd(a, b) = ra + sb.$$

2.6 Sage Exercises

These exercises are about investigating basic properties of the integers, something we will frequently do when investigating groups. Use the editing capabilities of a Sage worksheet to annotate and explain your work.

1. Use the `next_prime()` command to construct two different 8-digit prime numbers and save them in variables named `a` and `b`.

2. Use the `.is_prime()` method to verify that your primes `a` and `b` are really prime.

3. Verify that 1 is the greatest common divisor of your two primes from the previous exercises.

4. Find two integers that make a "linear combination" of your two primes equal to 1. Include a verification of your result.

5. Determine a factorization into powers of primes for $c = 4\,598\,037\,234$.

6. Write a compute cell that defines the same value of `c` again, and then defines a candidate divisor of `c` named `d`. The third line of the cell should return `True` if and only if `d` is a divisor of `c`. Illustrate the use of your cell by testing your code with $d = 7$ and in a new copy of the cell, testing your code with $d = 11$.

2.7 References and Suggested Readings

[1] Brookshear, J. G. *Theory of Computation: Formal Languages, Automata, and Complexity.* Benjamin/Cummings, Redwood City, CA, 1989. Shows the relationships of the theoretical aspects of computer science to set theory and the integers.

[2] Hardy, G. H. and Wright, E. M. *An Introduction to the Theory of Numbers.* 6th ed. Oxford University Press, New York, 2008.

[3] Niven, I. and Zuckerman, H. S. *An Introduction to the Theory of Numbers.* 5th ed. Wiley, New York, 1991.

[4] Vanden Eynden, C. *Elementary Number Theory.* 2nd ed. Waveland Press, Long Grove IL, 2001.

Chapter 3

Groups

David Farmer

We begin our study of algebraic structures by investigating sets associated with single operations that satisfy certain reasonable axioms; that is, we want to define an operation on a set in a way that will generalize such familiar structures as the integers $\mathbb{Z}$ together with the single operation of addition, or invertible 2×2 matrices together with the single operation of matrix multiplication. The integers and the 2×2 matrices, together with their respective single operations, are examples of algebraic structures known as groups.[1]

The theory of groups occupies a central position in mathematics. Modern group theory arose from an attempt to find the roots of a polynomial in terms of its coefficients. Groups now play a central role in such areas as coding theory, counting, and the study of symmetries; many areas of biology, chemistry, and physics have benefited from group theory.

Alex Jordan helped improve this chapter.

3.1 Integer Equivalence Classes and Symmetries

Carl Friedrich Gauß, Leonhard Euler

Let us now investigate some mathematical structures that can be viewed as sets with single operations.

3.1.1 The Integers mod n

Gottfried Wilhelm Leibniz

The integers mod n have become indispensable in the theory and applications of algebra. In mathematics they are used in cryptography, coding theory, and the detection of errors in identification codes.

We have already seen that two integers a and b are equivalent mod n if n divides $a - b$. The integers mod n also partition $\mathbb{Z}$ into n different equivalence classes; we will denote the set of these equivalence classes by $\mathbb{Z}_n$. Consider the integers modulo 12 and the corresponding partition of the integers:

$$[0] = \{\ldots, -12, 0, 12, 24, \ldots\}$$
$$[1] = \{\ldots, -11, 1, 13, 25, \ldots\}$$

[1]A test footnote, with no real content.

$$\vdots$$
$$[11] = \{\ldots, -1, 11, 23, 35, \ldots\}.$$

When no confusion can arise, we will use $0, 1, \ldots, 11$ to indicate the equivalence classes $[0], [1], \ldots, [11]$ respectively. We can do arithmetic on $\mathbb{Z}_n$. For two integers a and b, define addition modulo n to be $(a + b) \pmod{n}$; that is, the remainder when $a + b$ is divided by n. Similarly, multiplication modulo n is defined as $(ab) \pmod{n}$, the remainder when ab is divided by n.

Example 3.1.1 Modular Addition. The following examples illustrate integer arithmetic modulo n:

$$\begin{aligned} 7 + 4 &\equiv 1 \pmod{5} & 7 \cdot 3 &\equiv 1 \pmod{5} \\ 3 + 5 &\equiv 0 \pmod{8} & 3 \cdot 5 &\equiv 7 \pmod{8} \\ 3 + 4 &\equiv 7 \pmod{12} & 3 \cdot 4 &\equiv 0 \pmod{12}. \end{aligned}$$

In particular, notice that it is possible that the product of two nonzero numbers modulo n can be equivalent to 0 modulo n. □

Example 3.1.2 Modular Arithmetic. Most, but not all, of the usual laws of arithmetic hold for addition and multiplication in $\mathbb{Z}_n$. For instance, it is not necessarily true that there is a multiplicative inverse. Consider the multiplication table for $\mathbb{Z}_8$ in Figure 3.1.3. Notice that 2, 4, and 6 do not have multiplicative inverses; that is, for $n = 2$, 4, or 6, there is no integer k such that $kn \equiv 1 \pmod{8}$.

$\cdot$	0	1	2	3	4	5	6	7
0	0	0	0	0	0	0	0	0
1	0	1	2	3	4	5	6	7
2	0	2	4	6	0	2	4	6
3	0	3	6	1	4	7	2	5
4	0	4	0	4	0	4	0	4
5	0	5	2	7	4	1	6	3
6	0	6	4	2	0	6	4	2
7	0	7	6	5	4	3	2	1

Figure 3.1.3 Multiplication table for $\mathbb{Z}_8$

□

Proposition 3.1.4 *Let $\mathbb{Z}_n$ be the set of equivalence classes of the integers mod n and $a, b, c \in \mathbb{Z}_n$.*

1. *Addition and multiplication are commutative:*

$$\begin{aligned} a + b &\equiv b + a \pmod{n} \\ ab &\equiv ba \pmod{n}. \end{aligned}$$

2. *Addition and multiplication are associative:*

$$\begin{aligned} (a + b) + c &\equiv a + (b + c) \pmod{n} \\ (ab)c &\equiv a(bc) \pmod{n}. \end{aligned}$$

3. *There are both additive and multiplicative identities:*

$$\begin{aligned} a + 0 &\equiv a \pmod{n} \\ a \cdot 1 &\equiv a \pmod{n}. \end{aligned}$$

4. *Multiplication distributes over addition:*

$$a(b+c) \equiv ab + ac \pmod{n}.$$

5. *For every integer a there is an additive inverse $-a$:*

$$a + (-a) \equiv 0 \pmod{n}.$$

6. *Let a be a nonzero integer. Then $\gcd(a, n) = 1$ if and only if there exists a multiplicative inverse b for $a \pmod{n}$; that is, a nonzero integer b such that*

$$ab \equiv 1 \pmod{n}.$$

Proof. We will prove (1) and (6) and leave the remaining properties to be proven in the exercises.
(1) Addition and multiplication are commutative modulo n since the remainder of $a+b$ divided by n is the same as the remainder of $b+a$ divided by n.
(6) Suppose that $\gcd(a, n) = 1$. Then there exist integers r and s such that $ar + ns = 1$. Since $ns = 1 - ar$, it must be the case that $ar \equiv 1 \pmod{n}$. Letting b be the equivalence class of r, $ab \equiv 1 \pmod{n}$.
Conversely, suppose that there exists an integer b such that $ab \equiv 1 \pmod{n}$. Then n divides $ab - 1$, so there is an integer k such that $ab - nk = 1$. Let $d = \gcd(a, n)$. Since d divides $ab - nk$, d must also divide 1; hence, $d = 1$. ∎

3.1.2 Symmetries

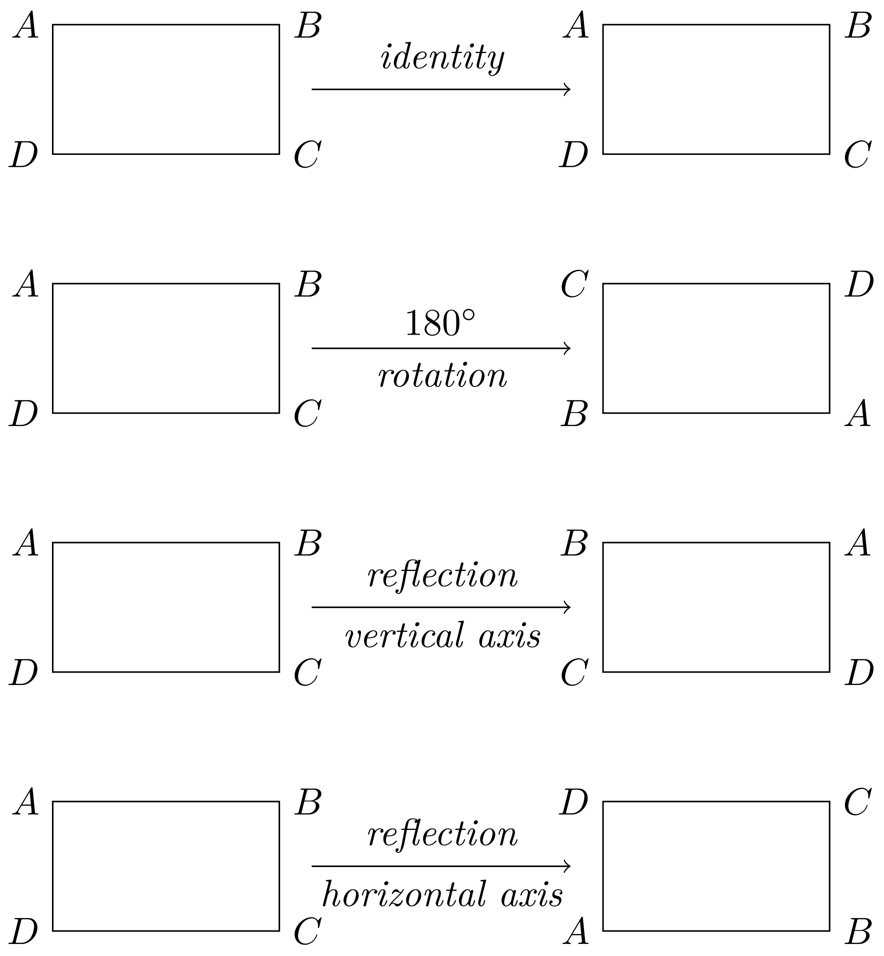
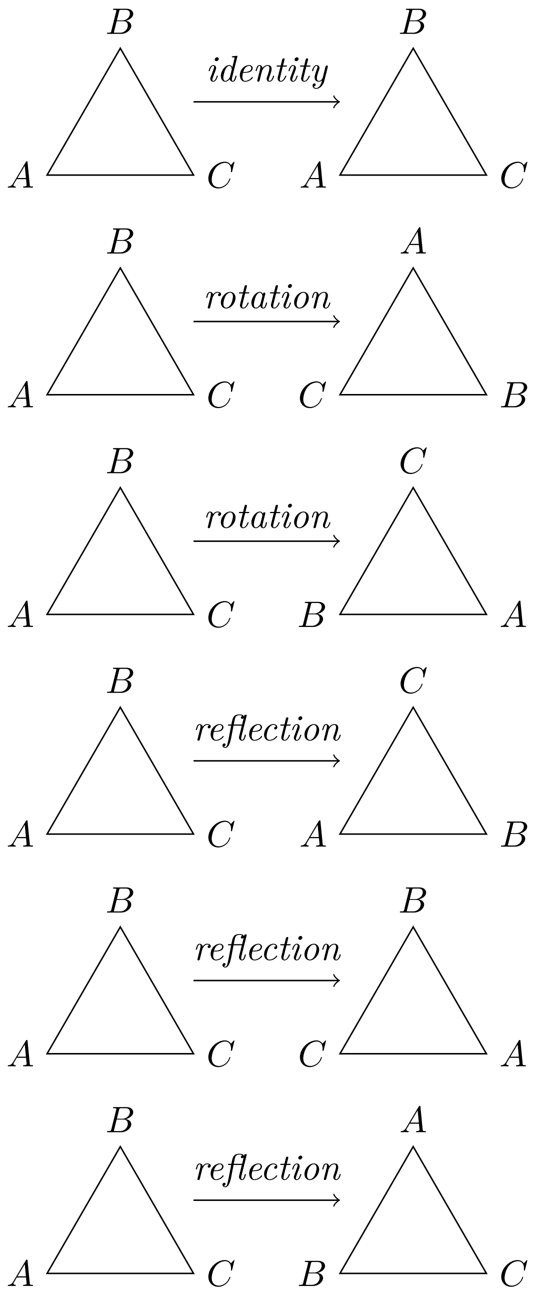


Figure 3.1.5 Rigid motions of a rectangle

A **symmetry** of a geometric figure is a rearrangement of the figure preserving the arrangement of its sides and vertices as well as its distances and angles. A map from the plane to itself preserving the symmetry of an object is called a **rigid motion**. For example, if we look at the rectangle in Figure 3.1.5, it is easy to see that a rotation of 180° or 360° returns a rectangle in the plane with the same orientation as the original rectangle and the same relationship among the vertices. A reflection of the rectangle across either the vertical axis or the horizontal axis can also be seen to be a symmetry. However, a 90° rotation in either direction cannot be a symmetry unless the rectangle is a square.



$$id = \begin{pmatrix} A & B & C \\ A & B & C \end{pmatrix}$$

$$\rho_1 = \begin{pmatrix} A & B & C \\ B & C & A \end{pmatrix}$$

$$\rho_2 = \begin{pmatrix} A & B & C \\ C & A & B \end{pmatrix}$$

$$\mu_1 = \begin{pmatrix} A & B & C \\ A & C & B \end{pmatrix}$$

$$\mu_2 = \begin{pmatrix} A & B & C \\ C & B & A \end{pmatrix}$$

$$\mu_3 = \begin{pmatrix} A & B & C \\ B & A & C \end{pmatrix}$$

Figure 3.1.6 Symmetries of a triangle

Let us find the symmetries of the equilateral triangle $\triangle ABC$. To find a symmetry of $\triangle ABC$, we must first examine the permutations of the vertices A, B, and C and then ask if a permutation extends to a symmetry of the triangle. Recall that a **permutation** of a set S is a one-to-one and onto map $\pi : S \to S$. The three vertices have $3! = 6$ permutations, so the triangle has at most six symmetries. To see that there are six permutations, observe there are three different possibilities for the first vertex, and two for the second, and the remaining vertex is determined by the placement of the first two. So we have $3 \cdot 2 \cdot 1 = 3! = 6$ different arrangements. To denote the permutation of the vertices of an equilateral triangle that sends A to B, B to C, and C to A, we write the array

$$\begin{pmatrix} A & B & C \\ B & C & A \end{pmatrix}.$$

Notice that this particular permutation corresponds to the rigid motion of rotating the triangle by 120° in a clockwise direction. In fact, every permutation gives rise to a symmetry of the triangle. All of these symmetries are shown in Figure 3.1.6.

A natural question to ask is what happens if one motion of the triangle $\triangle ABC$ is followed by another. Which symmetry is $\mu_1\rho_1$; that is, what happens when we do the permutation ρ_1 and then the permutation μ_1? *Remember that we are composing functions here. Although we usually multiply left to right, we compose functions right to left.* We have

$$\begin{aligned}(\mu_1\rho_1)(A) &= \mu_1(\rho_1(A)) = \mu_1(B) = C\\ (\mu_1\rho_1)(B) &= \mu_1(\rho_1(B)) = \mu_1(C) = B\\ (\mu_1\rho_1)(C) &= \mu_1(\rho_1(C)) = \mu_1(A) = A.\end{aligned}$$

This is the same symmetry as μ_2. Suppose we do these motions in the opposite order, ρ_1 then μ_1. It is easy to determine that this is the same as the symmetry μ_3; hence, $\rho_1\mu_1 \neq \mu_1\rho_1$. A multiplication table for the symmetries of an equilateral triangle $\triangle ABC$ is given in Figure 3.1.7.

Notice that in the multiplication table for the symmetries of an equilateral triangle, for every motion of the triangle α there is another motion β such that $\alpha\beta = id$; that is, for every motion there is another motion that takes the triangle back to its original orientation.

$\circ$	id	ρ_1	ρ_2	μ_1	μ_2	μ_3
id	id	ρ_1	ρ_2	μ_1	μ_2	μ_3
ρ_1	ρ_1	ρ_2	id	μ_3	μ_1	μ_2
ρ_2	ρ_2	id	ρ_1	μ_2	μ_3	μ_1
μ_1	μ_1	μ_2	μ_3	id	ρ_1	ρ_2
μ_2	μ_2	μ_3	μ_1	ρ_2	id	ρ_1
μ_3	μ_3	μ_1	μ_2	ρ_1	ρ_2	id

Figure 3.1.7 Symmetries of an equilateral triangle

3.2 Definitions and Examples

3.2.1 Definition of a Group

The integers mod n and the symmetries of a triangle or a rectangle are examples of groups. A **binary operation** or **law of composition** on a set G is a function $G \times G \to G$ that assigns to each pair $(a, b) \in G \times G$ a unique element $a \circ b$, or ab in G, called the composition of a and b. A **group** $(G, \circ)$ is a set G together with a law of composition $(a, b) \mapsto a \circ b$ that satisfies the following axioms.

- The law of composition is **associative**. That is,

$$(a \circ b) \circ c = a \circ (b \circ c)$$

  for $a, b, c \in G$.

- There exists an element $e \in G$, called the **identity element**, such that for any element $a \in G$
$$e \circ a = a \circ e = a.$$
- For each element $a \in G$, there exists an **inverse element** in G, denoted by a^{-1}, such that
$$a \circ a^{-1} = a^{-1} \circ a = e.$$

A group G with the property that $a \circ b = b \circ a$ for all $a, b \in G$ is called **abelian** or **commutative**. Groups not satisfying this property are said to be **nonabelian** or **noncommutative**.

3.2.2 Examples of Groups

Example 3.2.1 Group of All Integers. The integers $\mathbb{Z} = \{\ldots, -1, 0, 1, 2, \ldots\}$ form a group under the operation of addition. The binary operation on two integers $m, n \in \mathbb{Z}$ is just their sum. Since the integers under addition already have a well-established notation, we will use the operator $+$ instead of $\circ$; that is, we shall write $m + n$ instead of $m \circ n$. The identity is 0, and the inverse of $n \in \mathbb{Z}$ is written as $-n$ instead of n^{-1}. Notice that the set of integers under addition have the additional property that $m+n = n+m$ and therefore form an abelian group. □

Most of the time we will write ab instead of $a \circ b$; however, if the group already has a natural operation such as addition in the integers, we will use that operation. That is, if we are adding two integers, we still write $m+n$, $-n$ for the inverse, and 0 for the identity as usual. We also write $m - n$ instead of $m + (-n)$.

It is often convenient to describe a group in terms of an addition or multiplication table. Such a table is called a **Cayley table**.

Example 3.2.2 Group of Integers Modulo 5. The integers mod n form a group under addition modulo n. Consider $\mathbb{Z}_5$, consisting of the equivalence classes of the integers 0, 1, 2, 3, and 4. We define the group operation on $\mathbb{Z}_5$ by modular addition. We write the binary operation on the group additively; that is, we write $m + n$. The element 0 is the identity of the group and each element in $\mathbb{Z}_5$ has an inverse. For instance, $2 + 3 = 3 + 2 = 0$. Figure 3.2.3 is a Cayley table for $\mathbb{Z}_5$ (Contributed by Robert Beezer). By Proposition 3.1.4, $\mathbb{Z}_n = \{0, 1, \ldots, n-1\}$ is a group under the binary operation of addition mod n.

+	0	1	2	3	4
0	0	1	2	3	4
1	1	2	3	4	0
2	2	3	4	0	1
3	3	4	0	1	2
4	4	0	1	2	3

Figure 3.2.3 Cayley table for $(\mathbb{Z}_5, +)$

□

Example 3.2.4 Multiplication of Integers Modulo 6. Not every set with a binary operation is a group. For example, if we let modular multiplication be the binary operation on $\mathbb{Z}_n$, then $\mathbb{Z}_n$ fails to be a group. The element 1 acts as a group identity since $1 \cdot k = k \cdot 1 = k$ for any $k \in \mathbb{Z}_n$; however, a multiplicative inverse for 0 does not exist since $0 \cdot k = k \cdot 0 = 0$ for every k in $\mathbb{Z}_n$. Even if we consider the set $\mathbb{Z}_n \setminus \{0\}$, we still may not have a group. For instance, let $2 \in \mathbb{Z}_6$. Then 2 has no multiplicative inverse since

$$\begin{aligned} 0 \cdot 2 &= 0 & 1 \cdot 2 &= 2 \\ 2 \cdot 2 &= 4 & 3 \cdot 2 &= 0 \\ 4 \cdot 2 &= 2 & 5 \cdot 2 &= 4. \end{aligned}$$

By Proposition 3.1.4, every nonzero k does have an inverse in $\mathbb{Z}_n$ if k is relatively prime to n. Denote the set of all such nonzero elements in $\mathbb{Z}_n$ by $U(n)$. Then $U(n)$ is a group called the **group of units** of $\mathbb{Z}_n$. Figure 3.2.5 is a Cayley table for the group $U(8)$.

$\cdot$	1	3	5	7
1	1	3	5	7
3	3	1	7	5
5	5	7	1	3
7	7	5	3	1

Figure 3.2.5 Multiplication table for $U(8)$

□

Example 3.2.6 Symmetries of a Triangle is not Abelian. The symmetries of an equilateral triangle described in Section 3.1 form a nonabelian group. As we observed, it is not necessarily true that $\alpha\beta = \beta\alpha$ for two symmetries α and β. Using Figure 3.1.7, which is a Cayley table for this group, we can easily check that the symmetries of an equilateral triangle are indeed a group. We will denote this group by either S_3 or D_3, for reasons that will be explained later. □

Example 3.2.7 Matrix Multiplication of 2×2 Matrices is a Group. We use $\mathbb{M}_2(\mathbb{R})$ to denote the set of all 2×2 matrices. Let $GL_2(\mathbb{R})$ be the subset of $\mathbb{M}_2(\mathbb{R})$ consisting of invertible matrices; that is, a matrix

$$A = \begin{pmatrix} a & b \\ c & d \end{pmatrix}$$

is in $GL_2(\mathbb{R})$ if there exists a matrix A^{-1} such that $AA^{-1} = A^{-1}A = I$, where I is the 2×2 identity matrix. For A to have an inverse is equivalent to requiring that the determinant of A be nonzero; that is, $\det A = ad - bc \neq 0$. The set of invertible matrices forms a group called the **general linear group**. The identity of the group is the identity matrix

$$I = \begin{pmatrix} 1 & 0 \\ 0 & 1 \end{pmatrix}.$$

The inverse of $A \in GL_2(\mathbb{R})$ is

$$A^{-1} = \frac{1}{ad - bc} \begin{pmatrix} d & -b \\ -c & a \end{pmatrix}.$$

The product of two invertible matrices is again invertible. Matrix multiplication is associative, satisfying the other group axiom. For matrices it is not true in general that $AB = BA$; hence, $GL_2(\mathbb{R})$ is another example of a nonabelian group. □

Example 3.2.8 Group of Quaternions. Let

$$1 = \begin{pmatrix} 1 & 0 \\ 0 & 1 \end{pmatrix} \qquad I = \begin{pmatrix} 0 & 1 \\ -1 & 0 \end{pmatrix}$$
$$J = \begin{pmatrix} 0 & i \\ i & 0 \end{pmatrix} \qquad K = \begin{pmatrix} i & 0 \\ 0 & -i \end{pmatrix},$$

where $i^2 = -1$. Then the relations $I^2 = J^2 = K^2 = -1$, $IJ = K$, $JK = I$, $KI = J$, $JI = -K$, $KJ = -I$, and $IK = -J$ hold. The set $Q_8 = \{\pm 1, \pm I, \pm J, \pm K\}$ is a group called the **quaternion group**. Notice that Q_8 is noncommutative. □

Example 3.2.9 Group of Nonzero Complex Numbers. Let $\mathbb{C}^*$be the set of nonzero complex numbers. Under the operation of multiplication $\mathbb{C}^*$ forms a group. The identity is 1. If $z = a + bi$ is a nonzero complex number, then

$$z^{-1} = \frac{a - bi}{a^2 + b^2}$$

is the inverse of z. It is easy to see that the remaining group axioms hold. □

A group is **finite**, or has **finite order**, if it contains a finite number of elements; otherwise, the group is said to be **infinite** or to have **infinite order**. The **order** of a finite group is the number of elements that it contains. If G is a group containing n elements, we write $|G| = n$. The group $\mathbb{Z}_5$ is a finite group of order 5; the integers $\mathbb{Z}$ form an infinite group under addition, and we sometimes write $|\mathbb{Z}| = \infty$.

3.2.3 Basic Properties of Groups

Proposition 3.2.10 *The identity element in a group G is unique; that is, there exists only one element $e \in G$ such that $eg = ge = g$ for all $g \in G$.*

Proof. Suppose that e and e' are both identities in G. Then $eg = ge = g$ and $e'g = ge' = g$ for all $g \in G$. We need to show that $e = e'$. If we think of e as the identity, then $ee' = e'$; but if e' is the identity, then $ee' = e$. Combining these two equations, we have $e = ee' = e'$. (David Farmer helped with this proof.) ■

Inverses in a group are also unique. If g' and g'' are both inverses of an element g in a group G, then $gg' = g'g = e$ and $gg'' = g''g = e$. We want to show that $g' = g''$, but $g' = g'e = g'(gg'') = (g'g)g'' = eg'' = g''$. We summarize this fact in the following proposition.

Proposition 3.2.11 *If g is any element in a group G, then the inverse of g, denoted by g^{-1}, is unique.*

Proposition 3.2.12 *Let G be a group. If $a, b \in G$, then $(ab)^{-1} = b^{-1}a^{-1}$.*

Proof. Let $a, b \in G$. Then $abb^{-1}a^{-1} = aea^{-1} = aa^{-1} = e$. Similarly, $b^{-1}a^{-1}ab = e$. But by the previous proposition, inverses are unique; hence, $(ab)^{-1} = b^{-1}a^{-1}$. ■

Proposition 3.2.13 *Let G be a group. For any $a \in G$, $(a^{-1})^{-1} = a$.*

Proof. Observe that $a^{-1}(a^{-1})^{-1} = e$. Consequently, multiplying both sides of this equation by a, we have

$$(a^{-1})^{-1} = e(a^{-1})^{-1} = aa^{-1}(a^{-1})^{-1} = ae = a.$$

■

It makes sense to write equations with group elements and group operations. If a and b are two elements in a group G, does there exist an element $x \in G$ such that $ax = b$? If such an x does exist, is it unique? The following proposition answers both of these questions positively.

Proposition 3.2.14 *Let G be a group and a and b be any two elements in G. Then the equations $ax = b$ and $xa = b$ have unique solutions in G.*
Proof. Suppose that $ax = b$. We must show that such an x exists. Multiplying both sides of $ax = b$ by a^{-1}, we have $x = ex = a^{-1}ax = a^{-1}b$.
To show uniqueness, suppose that x_1 and x_2 are both solutions of $ax = b$; then $ax_1 = b = ax_2$. So $x_1 = a^{-1}ax_1 = a^{-1}ax_2 = x_2$. The proof for the existence and uniqueness of the solution of $xa = b$ is similar. ■

Proposition 3.2.15 *If G is a group and $a, b, c \in G$, then $ba = ca$ implies $b = c$ and $ab = ac$ implies $b = c$.*

This proposition tells us that the **right and left cancellation laws** are true in groups. We leave the proof as an exercise.

We can use exponential notation for groups just as we do in ordinary algebra. If G is a group and $g \in G$, then we define $g^0 = e$. For $n \in \mathbb{N}$, we define

$$g^n = \underbrace{g \cdot g \cdots g}_{n \text{ times}}$$

and

$$g^{-n} = \underbrace{g^{-1} \cdot g^{-1} \cdots g^{-1}}_{n \text{ times}}.$$

Theorem 3.2.16 *In a group, the usual laws of exponents hold; that is, for all $g, h \in G$,*

1. *$g^m g^n = g^{m+n}$ for all $m, n \in \mathbb{Z}$;*
2. *$(g^m)^n = g^{mn}$ for all $m, n \in \mathbb{Z}$;*
3. *$(gh)^n = (h^{-1}g^{-1})^{-n}$ for all $n \in \mathbb{Z}$. Furthermore, if G is abelian, then $(gh)^n = g^n h^n$.*

We will leave the proof of this theorem as an exercise. Notice that $(gh)^n \neq g^n h^n$ in general, since the group may not be abelian. If the group is $\mathbb{Z}$ or $\mathbb{Z}_n$, we write the group operation additively and the exponential operation multiplicatively; that is, we write ng instead of g^n. The laws of exponents now become

1. $mg + ng = (m + n)g$ for all $m, n \in \mathbb{Z}$;
2. $m(ng) = (mn)g$ for all $m, n \in \mathbb{Z}$;
3. $m(g + h) = mg + mh$ for all $n \in \mathbb{Z}$.

It is important to realize that the last statement can be made only because $\mathbb{Z}$ and $\mathbb{Z}_n$ are commutative groups.

3.2.4 Historical Note

Although the first clear axiomatic definition of a group was not given until the late 1800s, group-theoretic methods had been employed before this time in the development of many areas of mathematics, including geometry and the theory of algebraic equations.

Joseph-Louis Lagrange used group-theoretic methods in a 1770–1771 memoir to study methods of solving polynomial equations. Later, Évariste Galois

(1811–1832) succeeded in developing the mathematics necessary to determine exactly which polynomial equations could be solved in terms of the polynomial's coefficients. Galois' primary tool was group theory.

The study of geometry was revolutionized in 1872 when Felix Klein proposed that geometric spaces should be studied by examining those properties that are invariant under a transformation of the space. Sophus Lie, a contemporary of Klein, used group theory to study solutions of partial differential equations. One of the first modern treatments of group theory appeared in William Burnside's *The Theory of Groups of Finite Order* [1], first published in 1897.

3.3 Subgroups

3.3.1 Definitions and Examples

Sometimes we wish to investigate smaller groups sitting inside a larger group. The set of even integers $2\mathbb{Z} = \{\ldots, -2, 0, 2, 4, \ldots\}$ is a group under the operation of addition. This smaller group sits naturally inside of the group of integers under addition. We define a **subgroup** H of a group G to be a subset H of G such that when the group operation of G is restricted to H, H is a group in its own right. Observe that every group G with at least two elements will always have at least two subgroups, the subgroup consisting of the identity element alone and the entire group itself. The subgroup $H = \{e\}$ of a group G is called the **trivial subgroup**. A subgroup that is a proper subset of G is called a **proper subgroup**. In many of the examples that we have investigated up to this point, there exist other subgroups besides the trivial and improper subgroups.

Example 3.3.1 A Subgroup of the Reals. Consider the set of nonzero real numbers, $\mathbb{R}^*$, with the group operation of multiplication. The identity of this group is 1 and the inverse of any element $a \in \mathbb{R}^*$ is just $1/a$. We will show that

$$\mathbb{Q}^* = \{p/q : p \text{ and } q \text{ are nonzero integers}\}$$

is a subgroup of $\mathbb{R}^*$. The identity of $\mathbb{R}^*$ is 1; however, $1 = 1/1$ is the quotient of two nonzero integers. Hence, the identity of $\mathbb{R}^*$ is in $\mathbb{Q}^*$. Given two elements in $\mathbb{Q}^*$, say p/q and r/s, their product pr/qs is also in $\mathbb{Q}^*$. The inverse of any element $p/q \in \mathbb{Q}^*$ is again in $\mathbb{Q}^*$ since $(p/q)^{-1} = q/p$. Since multiplication in $\mathbb{R}^*$ is associative, multiplication in $\mathbb{Q}^*$ is associative. □

Example 3.3.2 A Subgroup of the Nonzero Complex Numbers. Recall that $\mathbb{C}^*$ is the multiplicative group of nonzero complex numbers. Let $H = \{1, -1, i, -i\}$. Then H is a subgroup of $\mathbb{C}^*$. It is quite easy to verify that H is a group under multiplication and that $H \subset \mathbb{C}^*$. □

Example 3.3.3 A Subgroup of Matrices With Determinant One. Let $SL_2(\mathbb{R})$ be the subset of $GL_2(\mathbb{R})$consisting of matrices of determinant one; that is, a matrix

$$A = \begin{pmatrix} a & b \\ c & d \end{pmatrix}$$

is in $SL_2(\mathbb{R})$ exactly when $ad - bc = 1$. To show that $SL_2(\mathbb{R})$ is a subgroup of the general linear group, we must show that it is a group under matrix multiplication. The 2×2 identity matrix is in $SL_2(\mathbb{R})$, as is the inverse of the matrix A:

$$A^{-1} = \begin{pmatrix} d & -b \\ -c & a \end{pmatrix}.$$

It remains to show that multiplication is closed; that is, that the product of two matrices of determinant one also has determinant one. We will leave this task as an exercise. The group $SL_2(\mathbb{R})$ is called the **special linear group**. □

Example 3.3.4 Groups, Subsets, Operations. It is important to realize that a subset H of a group G can be a group without being a subgroup of G. For H to be a subgroup of G it must inherit G's binary operation. The set of all 2×2 matrices, $\mathbb{M}_2(\mathbb{R})$, forms a group under the operation of addition. The 2×2 general linear group is a subset of $\mathbb{M}_2(\mathbb{R})$ and is a group under matrix multiplication, but it is not a subgroup of $\mathbb{M}_2(\mathbb{R})$. If we add two invertible matrices, we do not necessarily obtain another invertible matrix. Observe that

$$\begin{pmatrix} 1 & 0 \\ 0 & 1 \end{pmatrix} + \begin{pmatrix} -1 & 0 \\ 0 & -1 \end{pmatrix} = \begin{pmatrix} 0 & 0 \\ 0 & 0 \end{pmatrix},$$

but the zero matrix is not in $GL_2(\mathbb{R})$. □

Example 3.3.5 Structurally Different Groups. One way of telling whether or not two groups are the same is by examining their subgroups. Other than the trivial subgroup and the group itself, the group $\mathbb{Z}_4$ has a single subgroup consisting of the elements 0 and 2. From the group $\mathbb{Z}_2$, we can form another group of four elements as follows. As a set this group is $\mathbb{Z}_2 \times \mathbb{Z}_2$. We perform the group operation coordinatewise; that is, $(a, b)+(c, d) = (a+c, b+d)$. Figure 3.3.6 is an addition table for $\mathbb{Z}_2 \times \mathbb{Z}_2$. Since there are three nontrivial proper subgroups of $\mathbb{Z}_2 \times \mathbb{Z}_2$, $H_1 = \{(0, 0), (0, 1)\}$, $H_2 = \{(0, 0), (1, 0)\}$, and $H_3 = \{(0, 0), (1, 1)\}$, $\mathbb{Z}_4$ and $\mathbb{Z}_2 \times \mathbb{Z}_2$ must be different groups.

$+$	$(0,0)$	$(0,1)$	$(1,0)$	$(1,1)$
$(0,0)$	$(0,0)$	$(0,1)$	$(1,0)$	$(1,1)$
$(0,1)$	$(0,1)$	$(0,0)$	$(1,1)$	$(1,0)$
$(1,0)$	$(1,0)$	$(1,1)$	$(0,0)$	$(0,1)$
$(1,1)$	$(1,1)$	$(1,0)$	$(0,1)$	$(0,0)$

Figure 3.3.6 Addition table for $\mathbb{Z}_2 \times \mathbb{Z}_2$

□

3.3.2 Some Subgroup Theorems

Let us examine some criteria for determining exactly when a subset of a group is a subgroup.

Proposition 3.3.7 *A subset H of G is a subgroup if and only if it satisfies the following conditions.*

1. *The identity e of G is in H.*
2. *If $h_1, h_2 \in H$, then $h_1h_2 \in H$.*
3. *If $h \in H$, then $h^{-1} \in H$.*

Proof. First suppose that H is a subgroup of G. We must show that the three conditions hold. Since H is a group, it must have an identity e_H. We must show that $e_H = e$, where e is the identity of G. We know that $e_He_H = e_H$ and that $ee_H = e_He = e_H$; hence, $ee_H = e_He_H$. By right-hand cancellation,

$e = e_H$. The second condition holds since a subgroup H is a group. To prove the third condition, let $h \in H$. Since H is a group, there is an element $h' \in H$ such that $hh' = h'h = e$. By the uniqueness of the inverse in G, $h' = h^{-1}$.
Conversely, if the three conditions hold, we must show that H is a group under the same operation as G; however, these conditions plus the associativity of the binary operation are exactly the axioms stated in the definition of a group. ■

Proposition 3.3.8 *Let H be a subset of a group G. Then H is a subgroup of G if and only if $H \neq \emptyset$, and whenever $g, h \in H$ then gh^{-1} is in H.*

Proof. First assume that H is a subgroup of G. We wish to show that $gh^{-1} \in H$ whenever g and h are in H. Since h is in H, its inverse h^{-1} must also be in H. Because of the closure of the group operation, $gh^{-1} \in H$.
Conversely, suppose that $H \subset G$ such that $H \neq \emptyset$ and $gh^{-1} \in H$ whenever $g, h \in H$. If $g \in H$, then $gg^{-1} = e$ is in H. If $g \in H$, then $eg^{-1} = g^{-1}$ is also in H. Now let $h_1, h_2 \in H$. We must show that their product is also in H. However, $h_1(h_2^{-1})^{-1} = h_1 h_2 \in H$. Hence, H is a subgroup of G. ■

Remark 3.3.9 Sage. The first half of this text is about group theory. Sage includes Groups, Algorithms and Programming (GAP), a program designed primarly for just group theory, and in continuous development since 1986. Many of Sage's computations for groups ultimately are performed by GAP.

3.4 Sage

Many of the groups discussed in this chapter are available for study in Sage. It is important to understand that sets that form algebraic objects (groups in this chapter) are called "parents" in Sage, and elements of these objects are called, well, "elements." So every element belongs to a parent (in other words, is contained in some set). We can ask about properties of parents (finite? order? abelian?), and we can ask about properties of individual elements (identity? inverse?). In the following we will show you how to create some of these common groups and begin to explore their properties with Sage.

3.4.1 Integers mod n

```
Z8 = Integers(8)
Z8
```

```
Ring of integers modulo 8
```

```
Z8.list()
```

```
[0, 1, 2, 3, 4, 5, 6, 7]
```

```
a = Z8.an_element(); a
```

```
0
```

```
a.parent()
```

```
Ring of integers modulo 8
```

We would like to work with elements of `Z8`. If you were to type a `6` into a compute cell right now, what would you mean? The integer 6, the rational number $\frac{6}{1}$, the real number 6.00000, or the complex number $6.00000+0.00000i$? Or perhaps you really do want the integer 6 mod 8? Sage really has no idea what you mean or want. To make this clear, you can "coerce" `6` into `Z8` with the syntax `Z8(6)`. Without this, Sage will treat a input number like `6` as an integer, the simplest possible interpretation in some sense. Study the following carefully, where we first work with "normal" integers and then with integers mod 8.

```
a = 6
a
```

```
6
```

```
a.parent()
```

```
Integer Ring
```

```
b = 7
c = a + b; c
```

```
13
```

```
d = Z8(6)
d
```

```
6
```

```
d.parent()
```

```
Ring of integers modulo 8
```

```
e = Z8(7)
f = d+e; f
```

```
5
```

```
g = Z8(85); g
```

```
5
```

```
f == g
```

```
True
```

`Z8` is a bit unusual as a first example, since it has two operations defined, both addition and multiplication, with addition forming a group, and multiplication not forming a group. Still, we can work with the additive portion, here forming the Cayley table for the addition.

```
Z8.addition_table(names='elements')
```

```
+  0 1 2 3 4 5 6 7
 +----------------
0| 0 1 2 3 4 5 6 7
1| 1 2 3 4 5 6 7 0
2| 2 3 4 5 6 7 0 1
```

```
3| 3 4 5 6 7 0 1 2
4| 4 5 6 7 0 1 2 3
5| 5 6 7 0 1 2 3 4
6| 6 7 0 1 2 3 4 5
7| 7 0 1 2 3 4 5 6
```

When n is a prime number, the multipicative structure (excluding zero), will also form a group.

The integers mod n are very important, so Sage implements both addition and multiplication together. Groups of symmetries are a better example of how Sage implements groups, since there is just one operation present.

3.4.2 Groups of symmetries

The symmetries of some geometric shapes are already defined in Sage, albeit with different names. They are implemented as “permutation groups” which we will begin to study carefully in Chapter 5.

Sage uses integers to label vertices, starting the count at 1, instead of letters. Elements by default are printed using “cycle notation” which we will see described carefully in Chapter 5. Here is an example, with both the mathematics and Sage. For the Sage part, we create the group of symmetries and then create the symmetry ρ_2 with coercion, followed by outputting the element in cycle notation. Then we create just the *bottom row* of the notation we are using for permutations.

$$\rho_2 = \begin{pmatrix} A & B & C \\ C & A & B \end{pmatrix} = \begin{pmatrix} 1 & 2 & 3 \\ 3 & 1 & 2 \end{pmatrix}$$

```
triangle = SymmetricGroup(3)
rho2 = triangle([3,1,2])
rho2
```

```
(1,3,2)
```

```
[rho2(x) for x in triangle.domain()]
```

```
[3, 1, 2]
```

The final list comprehension deserves comment. The `.domain()` method gives a lait of the symbols used for the permutation group `triangle` and then `rho2` is employed with syntax like it is a function (it *is* a function) to create the images that would occupy the bottom row.

With a double list comprehension we can list all six elements of the group in the “bottom row” format. A good exercise would be to pair up each element with its name as given in Figure 3.1.6.

```
[[a(x) for x in triangle.domain()] for a in triangle]
```

```
[[1, 2, 3], [2, 1, 3], [2, 3, 1], [3, 1, 2], [1, 3, 2], [3,
   2, 1]]
```

Different books, different authors, different software all have different ideas about the order in which to write multiplication of functions. This textbook builds on the idea of composition of functions, so that fg is the composition $(fg)(x) = f(g(x))$ and it is natural to apply g first. Sage takes the opposite view and since we write fg, Sage will understand that we want to do f first. Neither approach is wrong, and neither is necessarily superior, they are just different and there are good arguments for either one. When you consult other books that work with permutation groups, you want to first determine which approach it takes.

The translation here between the text and Sage will be worthwhile practice. Here we will reprise the discussion at the end of Section 3.1, but reverse the order on each product to compute Sage-style and exactly mirror what the text does.

```
mu1 = triangle([1,3,2])
mu2 = triangle([3,2,1])
mu3 = triangle([2,1,3])
rho1 = triangle([2,3,1])
product = rho1*mu1
product == mu2
```

```
True
```

```
[product(x) for x in triangle.domain()]
```

```
[3, 2, 1]
```

```
rho1*mu1 == mu1*rho1
```

```
False
```

```
mu1*rho1 == mu3
```

```
True
```

Now that we understand that Sage does multiplication in reverse, we can compute the Cayley table for this group. Default behavior is to just name elements of a group as letters, `a`, `b`, `c`, \dots{} in the same order that the `.list()` command would produce the elements of the group. But you can also print the elements in the table as themselves (that uses cycle notation here), or you can give the elements names. We will use `u` as shorthand for μ and `r` as shorthand for ρ.

```
triangle.cayley_table()
```

```
*  a b c d e f
 +------------
a| a b c d e f
b| b a f e d c
c| c e d a f b
d| d f a c b e
e| e c b f a d
f| f d e b c a
```

```
triangle.cayley_table(names='elements')
```

```
      *       ()   (1,2) (1,2,3) (1,3,2)   (2,3)   (1,3)
       +------------------------------------------------
    ()|      ()   (1,2) (1,2,3) (1,3,2)   (2,3)   (1,3)
  (1,2)|   (1,2)      ()   (1,3)   (2,3) (1,3,2) (1,2,3)
(1,2,3)| (1,2,3)   (2,3) (1,3,2)      ()   (1,3)   (1,2)
(1,3,2)| (1,3,2)   (1,3)      () (1,2,3)   (1,2)   (2,3)
  (2,3)|   (2,3) (1,2,3)   (1,2)   (1,3)      () (1,3,2)
  (1,3)|   (1,3) (1,3,2)   (2,3)   (1,2) (1,2,3)      ()
```

```
triangle.cayley_table(names=['id','u1','u3','r1','r2','u2'])
```

```
  *  id u1 u3 r1 r2 u2
   +------------------
 id| id u1 u3 r1 r2 u2
 u1| u1 id u2 r2 r1 u3
 u3| u3 r2 r1 id u2 u1
 r1| r1 u2 id u3 u1 r2
 r2| r2 u3 u1 u2 id r1
 u2| u2 r1 r2 u1 u3 id
```

You should verify that the table above is correct, just like Table 3.2 is correct. Remember that the convention is to multiply a row label times a column label, in that order. However, to do a check across the two tables, you will need to recall the difference in ordering between your textbook and Sage.

3.4.3 Quaternions

Sage implements the quaternions, but the elements are not matrices, but rather are permutations. Despite appearances the structure is identical. It should not matter which version you have in mind (matrices or permutations) if you build the Cayley table and use the default behavior of using letters to name the elements. As permutations, or as letters, can you identify -1, I, J and K?

```
Q = QuaternionGroup()
[[a(x) for x in Q.domain()] for a in Q]
```

```
[[1, 2, 3, 4, 5, 6, 7, 8], [2, 3, 4, 1, 6, 7, 8, 5],
 [5, 8, 7, 6, 3, 2, 1, 4], [3, 4, 1, 2, 7, 8, 5, 6],
 [6, 5, 8, 7, 4, 3, 2, 1], [8, 7, 6, 5, 2, 1, 4, 3],
 [4, 1, 2, 3, 8, 5, 6, 7], [7, 6, 5, 8, 1, 4, 3, 2]]
```

```
Q.cayley_table()
```

```
*  a b c d e f g h
 +----------------
a| a b c d e f g h
b| b d f g c h a e
c| c e d h g b f a
d| d g h a f e b c
e| e h b f d a c g
f| f c g e a d h b
g| g a e b h c d f
h| h f a c b g e d
```

It should be fairly obvious that `a` is the identity element of the group (1), either from its behavior in the table, or from its "bottom row" representation in the list above. And if you prefer, you can ask Sage.

```
id = Q.identity()
[id(x) for x in Q.domain()]
```

```
[1, 2, 3, 4, 5, 6, 7, 8]
```

Now -1 should have the property that $-1 \cdot -1 = 1$. We see that the identity element `a` is on the diagonal of the Cayley table only when we compute `c*c`.

We can verify this easily, borrowing the third "bottom row" element from the list above. With this information, once we locate I, we can easily compute $-I$, and so on.

```
minus_one = Q([3, 4, 1, 2, 7, 8, 5, 6])
minus_one*minus_one == Q.identity()
```

```
True
```

See if you can pair up the letters with all eight elements of the quaternions. Be a bit careful with your names, the symbol `I` is used by Sage for the imaginary number i (which we will use below), but Sage will silently let you redefine it to be anything you like. Same goes for lower-case `i`. So call your elements of the quaternions something like `QI, QJ, QK` to avoid confusion.

As we begin to work with groups it is instructive to work with the actual elements. But many properties of groups are totally independent of the order we use for multiplication, or the names or representations we use for the elements. Here are facts about the quaternions we can compute without any knowledge of just how the elements are written or multiplied.

```
Q.is_finite()
```

```
True
```

```
Q.order()
```

```
8
```

```
Q.is_abelian()
```

```
False
```

3.4.4 Subgroups

The best techniques for creating subgroups will come in future chapters, but we can create some groups that are naturally subgroups of other groups.

Elements of the quaternions were represented by certain permutations of the integers 1 through 8. We can also build the group of *all* permutations of these eight integers. It gets pretty big, so do not list it unless you want a lot of output! (I dare you.)

```
S8 = SymmetricGroup(8)
a = S8.random_element()
[a(x) for x in S8.domain()]     # random
```

```
[5, 2, 6, 4, 1, 8, 3, 7]
```

As a demonstration of reusing chunks of Sage code, we duplicate the previous example. But in each case the code lives in an external file, just once. So if you wanted to use setup code in more than one division, you could put it in a file and incorporate it similarly. Here we do not test the second instance, and so do not include expected output either. Typically, you would do this copy somewhere further away. Note that need to supply a path for the file that is relative to the location of teh source file containing the `<pretext>` element, since the repeated material is source XML.

```
S8 = SymmetricGroup(8)
a = S8.random_element()
[a(x) for x in S8.domain()]      # random
```

```
S8.order()
```

```
40320
```

The quaternions, `Q`, is a subgroup of the full group of all permutations, the symmetric group S_8 or `S8`, and Sage regards this as a property of `Q`.

```
Q.is_subgroup(S8)
```

```
True
```

In Sage the complex numbers are known by the name `CC`. We can create a list of the elements in the subgroup described in Example 3.2.9. Then we can verify that this set is a subgroup by examining the Cayley table, using multiplication as the operation.

```
H = [CC(1), CC(-1), CC(I), CC(-I)]
CC.multiplication_table(elements=H,
                        names=['1', '-1', 'i', '-i'])
```

```
*    1  -1   i  -i
  +------------
 1|   1  -1   i  -i
-1|  -1   1  -i   i
 i|   i  -i  -1   1
-i|  -i   i   1  -1
```

3.5 Exercises

1. Find all $x \in \mathbb{Z}$ satisfying each of the following equations.

(a) $3x \equiv 2 \pmod{7}$
(b) $5x + 1 \equiv 13 \pmod{23}$
(c) $5x + 1 \equiv 13 \pmod{26}$
(d) $9x \equiv 3 \pmod{5}$
(e) $5x \equiv 1 \pmod{6}$
(f) $3x \equiv 1 \pmod{6}$

Hint. (a) $3 + 7\mathbb{Z} = \{\ldots, -4, 3, 10, \ldots\}$; (c) $18 + 26\mathbb{Z}$; (e) $5 + 6\mathbb{Z}$.

2. Which of the following multiplication tables defined on the set $G = \{a, b, c, d\}$ form a group? Support your answer in each case.

(a)

$\circ$	a	b	c	d
a	a	c	d	a
b	b	b	c	d
c	c	d	a	b
d	d	a	b	c

(c)

$\circ$	a	b	c	d
a	a	b	c	d
b	b	c	d	a
c	c	d	a	b
d	d	a	b	c

(b)

$\circ$	a	b	c	d
a	a	b	c	d
b	b	a	d	c
c	c	d	a	b
d	d	c	b	a

(d)

$\circ$	a	b	c	d
a	a	b	c	d
b	b	a	c	d
c	c	b	a	d
d	d	d	b	c

Hint. (a) Not a group; (c) a group.

3. Write out Cayley tables for groups formed by the symmetries of a rectangle and for $(\mathbb{Z}_4, +)$. How many elements are in each group? Are the groups the same? Why or why not?

4. Describe the symmetries of a rhombus and prove that the set of symmetries forms a group. Give Cayley tables for both the symmetries of a rectangle and the symmetries of a rhombus. Are the symmetries of a rectangle and those of a rhombus the same?

5. Describe the symmetries of a square and prove that the set of symmetries is a group. Give a Cayley table for the symmetries. How many ways can the vertices of a square be permuted? Is each permutation necessarily a symmetry of the square? The symmetry group of the square is denoted by D_4.

6. Give a multiplication table for the group $U(12)$.

Hint.

$\cdot$	1	5	7	11
1	1	5	7	11
5	5	1	11	7
7	7	11	1	5
11	11	7	5	1

7. Let $S = \mathbb{R} \setminus \{-1\}$ and define a binary operation on S by $a * b = a + b + ab$. Prove that $(S, *)$ is an abelian group.

8. Give an example of two elements A and B in $GL_2(\mathbb{R})$ with $AB \neq BA$.

Hint. Pick two matrices. Almost any pair will work.

9. Prove that the product of two matrices in $SL_2(\mathbb{R})$ has determinant one.

10. Prove that the set of matrices of the form

$$\begin{pmatrix} 1 & x & y \\ 0 & 1 & z \\ 0 & 0 & 1 \end{pmatrix}$$

is a group under matrix multiplication. This group, known as the **Heisenberg group**, is important in quantum physics. Matrix multiplication in the Heisenberg group is defined by

$$\begin{pmatrix} 1 & x & y \\ 0 & 1 & z \\ 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} 1 & x' & y' \\ 0 & 1 & z' \\ 0 & 0 & 1 \end{pmatrix} = \begin{pmatrix} 1 & x+x' & y+y'+xz' \\ 0 & 1 & z+z' \\ 0 & 0 & 1 \end{pmatrix}.$$

11. Prove that $\det(AB) = \det(A)\det(B)$ in $GL_2(\mathbb{R})$. Use this result to show that the binary operation in the group $GL_2(\mathbb{R})$ is closed; that is, if A and B are in $GL_2(\mathbb{R})$, then $AB \in GL_2(\mathbb{R})$.

12. Let $\mathbb{Z}_2^n = \{(a_1, a_2, \ldots, a_n) : a_i \in \mathbb{Z}_2\}$. Define a binary operation on $\mathbb{Z}_2^n$ by

$$(a_1, a_2, \ldots, a_n) + (b_1, b_2, \ldots, b_n) = (a_1 + b_1, a_2 + b_2, \ldots, a_n + b_n).$$

Prove that $\mathbb{Z}_2^n$ is a group under this operation. This group is important in algebraic coding theory.

13. Show that $\mathbb{R}^* = \mathbb{R} \setminus \{0\}$ is a group under the operation of multiplication.

14. Given the groups $\mathbb{R}^*$ and $\mathbb{Z}$, let $G = \mathbb{R}^* \times \mathbb{Z}$. Define a binary operation $\circ$ on G by $(a, m) \circ (b, n) = (ab, m + n)$. Show that G is a group under this operation.

15. Prove or disprove that every group containing six elements is abelian.

Hint. There is a nonabelian group containing six elements.

16. Give a specific example of some group G and elements $g, h \in G$ where $(gh)^n \neq g^n h^n$.

Hint. Look at the symmetry group of an equilateral triangle or a square.

17. Give an example of three different groups with eight elements. Why are the groups different?

Hint. The are five different groups of order 8.

18. Show that there are $n!$ permutations of a set containing n items.

Hint. Let

$$\sigma = \begin{pmatrix} 1 & 2 & \cdots & n \\ a_1 & a_2 & \cdots & a_n \end{pmatrix}$$

be in S_n. All of the a_is must be distinct. There are n ways to choose a_1, $n-1$ ways to choose a_2, …, 2 ways to choose a_{n-1}, and only one way to choose a_n. Therefore, we can form σ in $n(n-1)\cdots 2 \cdot 1 = n!$ ways.

19. Show that

$$0 + a \equiv a + 0 \equiv a \pmod{n}$$

for all $a \in \mathbb{Z}_n$.

20. Prove that there is a multiplicative identity for the integers modulo n:

$$a \cdot 1 \equiv a \pmod{n}.$$

21. For each $a \in \mathbb{Z}_n$ find an element $b \in \mathbb{Z}_n$ such that

$$a + b \equiv b + a \equiv 0 \pmod{n}.$$

22. Show that addition and multiplication mod n are well defined operations. That is, show that the operations do not depend on the choice of the representative from the equivalence classes mod n.

23. Show that addition and multiplication mod n are associative operations.

24. Show that multiplication distributes over addition modulo n:

$$a(b + c) \equiv ab + ac \pmod{n}.$$

25. Let a and b be elements in a group G. Prove that $ab^n a^{-1} = (aba^{-1})^n$ for $n \in \mathbb{Z}$.

Hint.

$$\begin{aligned}(aba^{-1})^n &= (aba^{-1})(aba^{-1}) \cdots (aba^{-1}) \\ &= ab(aa^{-1})b(aa^{-1})b \cdots b(aa^{-1})ba^{-1} \\ &= ab^n a^{-1}.\end{aligned}$$

26. Let $U(n)$ be the group of units in $\mathbb{Z}_n$. If $n > 2$, prove that there is an element $k \in U(n)$ such that $k^2 = 1$ and $k \neq 1$.

27. Prove that the inverse of $g_1 g_2 \cdots g_n$ is $g_n^{-1} g_{n-1}^{-1} \cdots g_1^{-1}$.

28. Prove the remainder of Proposition 3.2.14: if G is a group and $a, b \in G$, then the equation $xa = b$ has a unique solution in G.

29. Prove Theorem 3.2.16.

30. Prove the right and left cancellation laws for a group G; that is, show that in the group G, $ba = ca$ implies $b = c$ and $ab = ac$ implies $b = c$ for elements $a, b, c \in G$.

31. Show that if $a^2 = e$ for all elements a in a group G, then G must be abelian.

Hint. Since $abab = (ab)^2 = e = a^2 b^2 = aabb$, we know that $ba = ab$.

32. Show that if G is a finite group of even order, then there is an $a \in G$ such that a is not the identity and $a^2 = e$.

33. Let G be a group and suppose that $(ab)^2 = a^2 b^2$ for all a and b in G. Prove that G is an abelian group.

34. Find all the subgroups of $\mathbb{Z}_3 \times \mathbb{Z}_3$. Use this information to show that $\mathbb{Z}_3 \times \mathbb{Z}_3$ is not the same group as $\mathbb{Z}_9$. (See Example 3.3.5 for a short description of the product of groups.)

35. Find all the subgroups of the symmetry group of an equilateral triangle.

Hint. $H_1 = \{id\}$, $H_2 = \{id, \rho_1, \rho_2\}$, $H_3 = \{id, \mu_1\}$, $H_4 = \{id, \mu_2\}$, $H_5 = \{id, \mu_3\}$, S_3.

36. Compute the subgroups of the symmetry group of a square.

37. Let $H = \{2^k : k \in \mathbb{Z}\}$. Show that H is a subgroup of $\mathbb{Q}^*$.

38. Let $n = 0, 1, 2, \ldots$ and $n\mathbb{Z} = \{nk : k \in \mathbb{Z}\}$. Prove that $n\mathbb{Z}$ is a subgroup of $\mathbb{Z}$. Show that these subgroups are the only subgroups of $\mathbb{Z}$.

39. Let $\mathbb{T} = \{z \in \mathbb{C}^* : |z| = 1\}$. Prove that $\mathbb{T}$ is a subgroup of $\mathbb{C}^*$.

40.

$$\begin{pmatrix} \cos\theta & -\sin\theta \\ \sin\theta & \cos\theta \end{pmatrix}$$

where $\theta \in \mathbb{R}$. Prove that G is a subgroup of $SL_2(\mathbb{R})$.

41. Prove that

$$G = \{a + b\sqrt{2} : a, b \in \mathbb{Q} \text{ and } a \text{ and } b \text{ are not both zero}\}$$

is a subgroup of $\mathbb{R}^*$ under the group operation of multiplication.

Hint. The identity of G is $1 = 1 + 0\sqrt{2}$. Since $(a + b\sqrt{2})(c + d\sqrt{2}) = (ac + 2bd) + (ad + bc)\sqrt{2}$, G is closed under multiplication. Finally, $(a + b\sqrt{2})^{-1} = a/(a^2 - 2b^2) - b\sqrt{2}/(a^2 - 2b^2)$.

42. Let G be the group of 2×2 matrices under addition and

$$H = \left\{ \begin{pmatrix} a & b \\ c & d \end{pmatrix} : a + d = 0 \right\}.$$

Prove that H is a subgroup of G.

43. Prove or disprove: $SL_2(\mathbb{Z})$, the set of 2×2 matrices with integer entries and determinant one, is a subgroup of $SL_2(\mathbb{R})$.

44. List the subgroups of the quaternion group, Q_8.

45. Prove that the intersection of two subgroups of a group G is also a subgroup of G.

46. Prove or disprove: If H and K are subgroups of a group G, then $H \cup K$ is a subgroup of G.

Hint. Look at S_3.

47. Prove or disprove: If H and K are subgroups of a group G, then $HK = \{hk : h \in H \text{ and } k \in K\}$ is a subgroup of G. What if G is abelian?

48. Let G be a group and $g \in G$. Show that

$$Z(G) = \{x \in G : gx = xg \text{ for all } g \in G\}$$

is a subgroup of G. This subgroup is called the **center** of G.

49. Let a and b be elements of a group G. If $a^4 b = ba$ and $a^3 = e$, prove that $ab = ba$.

Hint. Since $a^4 b = ba$, it must be the case that $b = a^6 b = a^2 ba$, and we can conclude that $ab = a^3 ba = ba$.

50. Give an example of an infinite group in which every nontrivial subgroup is infinite.

51. If $xy = x^{-1} y^{-1}$ for all x and y in G, prove that G must be abelian.

52. Prove or disprove: Every proper subgroup of an nonabelian group is nonabelian.

53. Let H be a subgroup of G and

$$C(H) = \{g \in G : gh = hg \text{ for all } h \in H\}.$$

Prove $C(H)$ is a subgroup of G. This subgroup is called the **centralizer** of H in G.

54. Let H be a subgroup of G. If $g \in G$, show that $gHg^{-1} = \{g^{-1}hg : h \in H\}$ is also a subgroup of G.

Exercise Group. In each group, how many solutions are there to $x^2 = e$?

55. C_n, n odd.

Answer. 1

56. C_n, n even.

Answer. 2

57. D_n, n odd.

Answer. n

58. D_n, n even.

Answer. $n + 1$

59. This is an odd-numbered exercise with tasks.

(a) What is $1 + 1$?

Answer. 2

(b) This task has subtasks.

(i) What is $3 + 3$?

Answer. 6

(ii) What is $5 + 5$?

Answer. 10

60. This is an even-numbered exercise with tasks.

(a) What is $2 + 2$?

Answer. 4

(b) This task has subtasks.

(i) What is $4 + 4$?

Answer. 8

(ii) What is $6 + 6$?

Answer. 12

3.6 Additional Exercises: Detecting Errors

1. **UPC Symbols.** Universal Product Code (UPC) symbols are found on most products in grocery and retail stores. The UPC symbol is a 12-digit code identifying the manufacturer of a product and the product itself (Figure 3.6.1). The first 11 digits contain information about the product; the twelfth digit is used for error detection. If $d_1 d_2 \cdots d_{12}$ is a valid UPC number, then

$$3 \cdot d_1 + 1 \cdot d_2 + 3 \cdot d_3 + \cdots + 3 \cdot d_{11} + 1 \cdot d_{12} \equiv 0 \pmod{10}.$$

(a) Show that the UPC number 0-50000-30042-6, which appears in Figure 3.6.1, is a valid UPC number.

(b) Show that the number 0-50000-30043-6 is not a valid UPC number.

(c) Write a formula to calculate the check digit, d_{12}, in the UPC number.

(d) The UPC error detection scheme can detect most transposition errors; that is, it can determine if two digits have been interchanged. Show that the transposition error 0-05000-30042-6 is not detected. Find a transposition error that is detected. Can you find a general rule for the types of transposition errors that can be detected?

(e) Write a program that will determine whether or not a UPC number is valid.

0 50000 30042 6

Figure 3.6.1 A UPC code

2. It is often useful to use an inner product notation for this type of error detection scheme; hence, we will use the notion

$$(d_1, d_2, \ldots, d_k) \cdot (w_1, w_2, \ldots, w_k) \equiv 0 \pmod{n}$$

to mean

$$d_1w_1 + d_2w_2 + \cdots + d_kw_k \equiv 0 \pmod{n}.$$

Suppose that $(d_1, d_2, \ldots, d_k) \cdot (w_1, w_2, \ldots, w_k) \equiv 0 \pmod{n}$ is an error detection scheme for the k-digit identification number $d_1d_2 \cdots d_k$, where $0 \leq d_i < n$. Prove that all single-digit errors are detected if and only if $\gcd(w_i, n) = 1$ for $1 \leq i \leq k$.

3. Let $(d_1, d_2, \ldots, d_k) \cdot (w_1, w_2, \ldots, w_k) \equiv 0 \pmod{n}$ be an error detection scheme for the k-digit identification number $d_1d_2 \cdots d_k$, where $0 \leq d_i < n$. Prove that all transposition errors of two digits d_i and d_j are detected if and only if $\gcd(w_i - w_j, n) = 1$ for i and j between 1 and k.

4. **ISBN Codes.** Every book has an International Standard Book Number (ISBN) code. This is a 10-digit code indicating the book's publisher and title. The tenth digit is a check digit satisfying

$$(d_1, d_2, \ldots, d_{10}) \cdot (10, 9, \ldots, 1) \equiv 0 \pmod{11}.$$

One problem is that d_{10} might have to be a 10 to make the inner product zero; in this case, 11 digits would be needed to make this scheme work. Therefore, the character X is used for the eleventh digit. So ISBN 3-540-96035-X is a valid ISBN code.

(a) Is ISBN 0-534-91500-0 a valid ISBN code? What about ISBN 0-534-91700-0 and ISBN 0-534-19500-0?

(b) Does this method detect all single-digit errors? What about all transposition errors?

(c) How many different ISBN codes are there?

(d) Write a computer program that will calculate the check digit for the first nine digits of an ISBN code.

(e) A publisher has houses in Germany and the United States. Its German prefix is `3-540`. If its United States prefix will be `0-abc`, find `abc` such that the rest of the ISBN code will be the same for a book printed in Germany and in the United States. Under the ISBN coding method the first digit identifies the language; German is `3` and English is `0`. The next group of numbers identifies the publisher, and the last group identifies the specific book.

3.7 Sage Exercises

These exercises are about becoming comfortable working with groups in Sage.

1. Create the groups `CyclicPermutationGroup(8)` and `DihedralGroup(4)` and name these groups `C` and `D`, respectively. We will understand these constructions better shortly, but for now just understand that both objects you create are actually groups.

2. Check that `C` and `D` have the same size by using the `.order()` method. Determine which group is abelian, and which is not, by using the `.is_abelian()` method.

3. Use the `.cayley_table()` method to create the Cayley table for each group.

4. Write a nicely formatted discussion identifying differences between the two groups that are discernible in properties of their Cayley tables. In other words, what is {\em different} about these two groups that you can "see" in the Cayley tables? (In the Sage notebook, a Shift-click on a blue bar will bring up a mini-word-processor, and you can use use dollar signs to embed mathematics formatted using TEX syntax.)

5. For `C` locate the one subgroup of order 4. The group `D` has three subgroups of order 4. Select one of the three subgroups of `D` that has a different structure than the subgroup you obtained from `C`.

The `.subgroups()` method will give you a list of all of the subgroups to help you get started. A Cayley table will help you tell the difference between the two subgroups. What properties of these tables did you use to determine the difference in the structure of the subgroups?

6. The `.subgroup(elt_list)` method of a group will create the smallest subgroup containing the specified elements of the group, when given the elements as a list `elt_list`. Use this command to discover the shortest list of elements necessary to recreate the subgroups you found in the previous exercise. The equality comparison, `==`, can be used to test if two subgroups are equal.

3.8 References and Suggested Readings

[1] Burnside, W. *Theory of Groups of Finite Order.* 2nd ed. Cambridge University Press, Cambridge, 1911; Dover, New York, 1953. A classic. Also available at books.google.com.

[2] Gallian, J. A. and Winters, S. "Modular Arithmetic in the Marketplace," *The American Mathematical Monthly* **95** (1988): 548–51.

[3] Gallian, J. A. *Contemporary Abstract Algebra.* 7th ed. Brooks/Cole, Belmont, CA, 2009.

[4] Hall, M. *Theory of Groups.* 2nd ed. American Mathematical Society, Providence, 1959.

[5] Kurosh, A. E. *The Theory of Groups*, vols. I and II. American Mathematical Society, Providence, 1979.

[6] Rotman, J. J. *An Introduction to the Theory of Groups.* 4th ed. Springer, New York, 1995.

Chapter 4

Cyclic Groups

The groups $\mathbb{Z}$ and $\mathbb{Z}_n$, which are among the most familiar and easily understood groups, are both examples of what are called cyclic groups. In this chapter we will study the properties of cyclic groups and cyclic subgroups, which play a fundamental part in the classification of all abelian groups.

4.1 Cyclic groups

Often a subgroup will depend entirely on a single element of the group; that is, knowing that particular element will allow us to compute any other element in the subgroup.

Example 4.1.1 An Infinite Cyclic Subgroup, Modular Addition. Suppose that we consider $3 \in \mathbb{Z}$ and look at all multiples (both positive and negative) of 3. As a set, this is

$$3\mathbb{Z} = \{\ldots, -3, 0, 3, 6, \ldots\}.$$

It is easy to see that $3\mathbb{Z}$ is a subgroup of the integers. This subgroup is completely determined by the element 3 since we can obtain all of the other elements of the group by taking multiples of 3. Every element in the subgroup is "generated" by 3. □

Example 4.1.2 An Infinite Cyclic Subgroup, Multiplication of Rational Numbers. If $H = \{2^n : n \in \mathbb{Z}\}$, then H is a subgroup of the multiplicative group of nonzero rational numbers, $\mathbb{Q}^*$. If $a = 2^m$ and $b = 2^n$ are in H, then $ab^{-1} = 2^m 2^{-n} = 2^{m-n}$ is also in H. By Proposition 3.3.8, H is a subgroup of $\mathbb{Q}^*$ determined by the element 2. □

Theorem 4.1.3 *Let G be a group and a be any element in G. Then the set*

$$\langle a \rangle = \{a^k : k \in \mathbb{Z}\}$$

is a subgroup of G. Furthermore, $\langle a \rangle$ is the smallest subgroup of G that contains~a.

Proof. The identity is in $\langle a \rangle$ since $a^0 = e$. If g and h are any two elements in $\langle a \rangle$, then by the definition of $\langle a \rangle$ we can write $g = a^m$ and $h = a^n$ for some integers m and n. So $gh = a^m a^n = a^{m+n}$ is again in $\langle a \rangle$. Finally, if $g = a^n$ in $\langle a \rangle$, then the inverse $g^{-1} = a^{-n}$ is also in $\langle a \rangle$. Clearly, any subgroup H of G containing a must contain all the powers of a by closure; hence, H contains $\langle a \rangle$. Therefore, $\langle a \rangle$ is the smallest subgroup of G containing a. ■

Remark 4.1.4 If we are using the "+" notation, as in the case of the integers under addition, we write $\langle a \rangle = \{na : n \in \mathbb{Z}\}$.

For $a \in G$, we call $\langle a \rangle$ the **cyclic subgroup** generated by a. If G contains some element a such that $G = \langle a \rangle$, then G is a **cyclic group**. In this case a is a **generator** of G. If a is an element of a group G, we define the **order** of a to be the smallest positive integer n such that $a^n = e$, and we write $|a| = n$. If there is no such integer n, we say that the order of a is infinite and write $|a| = \infty$ to denote the order of a.

Example 4.1.5 Generators of a Finite Cyclic Group. Notice that a cyclic group can have more than a single generator. Both 1 and 5 generate $\mathbb{Z}_6$; hence, $\mathbb{Z}_6$ is a cyclic group. Not every element in a cyclic group is necessarily a generator of the group. The order of $2 \in \mathbb{Z}_6$ is 3. The cyclic subgroup generated by 2 is $\langle 2 \rangle = \{0, 2, 4\}$. □

The groups $\mathbb{Z}$ and $\mathbb{Z}_n$ are cyclic groups. The elements 1 and -1 are generators for $\mathbb{Z}$. We can certainly generate $\mathbb{Z}_n$ with 1 although there may be other generators of $\mathbb{Z}_n$, as in the case of $\mathbb{Z}_6$.

Example 4.1.6 A Cyclic Group of Units. The group of units, $U(9)$, in $\mathbb{Z}_9$ is a cyclic group. As a set, $U(9)$ is $\{1, 2, 4, 5, 7, 8\}$. The element 2 is a generator for $U(9)$ since

$$\begin{array}{ll} 2^1 = 2 & 2^2 = 4 \\ 2^3 = 8 & 2^4 = 7 \\ 2^5 = 5 & 2^6 = 1. \end{array}$$

□

Example 4.1.7 A Group That is Not Cyclic. Not every group is a cyclic group. Consider the symmetry group of an equilateral triangle S_3. The subgroups of S_3 are shown in Figure 4.1.8. Notice that every subgroup is cyclic; however, no single element generates the entire group. □

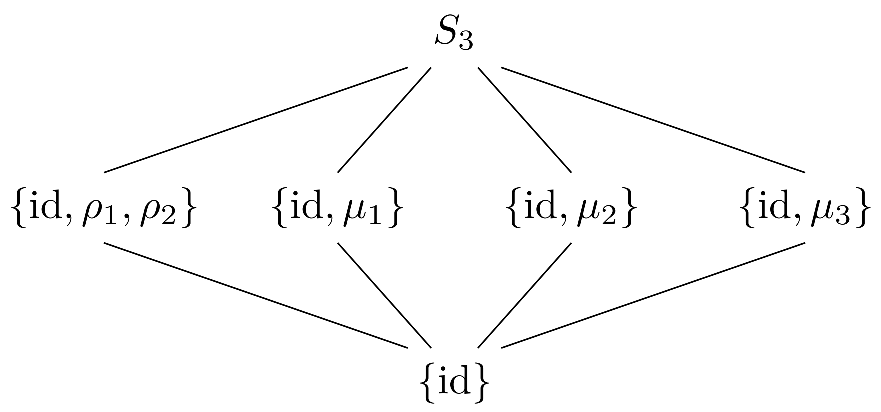


Figure 4.1.8 Subgroups of S_3

Theorem 4.1.9 *Every cyclic group is abelian.*
Proof. Let G be a cyclic group and $a \in G$ be a generator for G. If g and h are in G, then they can be written as powers of a, say $g = a^r$ and $h = a^s$. Since

$$gh = a^r a^s = a^{r+s} = a^{s+r} = a^s a^r = hg,$$

G is abelian. ■

4.2 Subgroups of a Cyclic Group

We can ask some interesting questions about cyclic subgroups of a group and subgroups of a cyclic group. If G is a group, which subgroups of G are cyclic? If G is a cyclic group, what type of subgroups does G possess?

Theorem 4.2.1 *Every subgroup of a cyclic group is cyclic.*

Proof. The main tools used in this proof are the division algorithm and the Principle of Well-Ordering. Let G be a cyclic group generated by a and suppose that H is a subgroup of G. If $H = \{e\}$, then trivially H is cyclic. Suppose that H contains some other element g distinct from the identity. Then g can be written as a^n for some integer n. Since H is a subgroup, $g^{-1} = a^n$ must also be in H. Since either n or $-n$ is positive, we can assume that H contains positive powers of a and $n > 0$. Let m be the smallest natural number such that $a^m \in H$. Such an m exists by the Principle of Well-Ordering.

We claim that $h = a^m$ is a generator for H. We must show that every $h' \in H$ can be written as a power of h. Since $h' \in H$ and H is a subgroup of G, $h' = a^k$ for some integer k. Using the division algorithm, we can find numbers q and r such that $k = mq + r$ where $0 \leq r < m$; hence,

$$a^k = a^{mq+r} = (a^m)^q a^r = h^q a^r.$$

So $a^r = a^k h^{-q}$. Since a^k and h^{-q} are in H, a^r must also be in H. However, m was the smallest positive number such that a^m was in H; consequently, $r = 0$ and so $k = mq$. Therefore,

$$h' = a^k = a^{mq} = h^q$$

and H is generated by h. ■

Corollary 4.2.2 *The subgroups of $\mathbb{Z}$ are exactly $n\mathbb{Z}$ for $n = 0, 1, 2, \ldots$.*

Proposition 4.2.3 *Let G be a cyclic group of order n and suppose that a is a generator for G. Then $a^k = e$ if and only if n divides k.*

Proof. First suppose that $a^k = e$. By the division algorithm, $k = nq + r$ where $0 \leq r < n$; hence,

$$e = a^k = a^{nq+r} = a^{nq} a^r = e a^r = a^r.$$

Since the smallest positive integer m such that $a^m = e$ is n, $r = 0$.

Conversely, if n divides k, then $k = ns$ for some integer s. Consequently,

$$a^k = a^{ns} = (a^n)^s = e^s = e.$$

■

Theorem 4.2.4 *Let G be a cyclic group of order n and suppose that $a \in G$ is a generator of the group. If $b = a^k$, then the order of b is n/d, where $d = \gcd(k, n)$.*

Proof. We wish to find the smallest integer m such that $e = b^m = a^{km}$. By Proposition 4.2.3, this is the smallest integer m such that n divides km or, equivalently, n/d divides $m(k/d)$. Since d is the greatest common divisor of n and k, n/d and k/d are relatively prime. Hence, for n/d to divide $m(k/d)$ it must divide m. The smallest such m is n/d. ■

Corollary 4.2.5 *The generators of $\mathbb{Z}_n$ are the integers r such that $1 \leq r < n$ and $\gcd(r, n) = 1$.*

Example 4.2.6 A Finite Cyclic Group of Order 16. Let us examine the group $\mathbb{Z}_{16}$. The numbers 1, 3, 5, 7, 9, 11, 13, and 15 are the elements of $\mathbb{Z}_{16}$ that are relatively prime to 16. Each of these elements generates $\mathbb{Z}_{16}$. For example,

$$\begin{array}{ccc} 1 \cdot 9 = 9 & 2 \cdot 9 = 2 & 3 \cdot 9 = 11 \\ 4 \cdot 9 = 4 & 5 \cdot 9 = 13 & 6 \cdot 9 = 6 \\ 7 \cdot 9 = 15 & 8 \cdot 9 = 8 & 9 \cdot 9 = 1 \\ 10 \cdot 9 = 10 & 11 \cdot 9 = 3 & 12 \cdot 9 = 12 \\ 13 \cdot 9 = 5 & 14 \cdot 9 = 14 & 15 \cdot 9 = 7. \end{array}$$

□

4.3 Cyclic Groups of Complex Numbers

The **complex numbers** are defined as

$$\mathbb{C} = \{a + bi : a, b \in \mathbb{R}\},$$

where $i^2 = -1$. If $z = a + bi$, then a is the **real part** of z and b is the **imaginary part** of z.

To add two complex numbers $z = a + bi$ and $w = c + di$, we just add the corresponding real and imaginary parts:

$$z + w = (a + bi) + (c + di) = (a + c) + (b + d)i.$$

Remembering that $i^2 = -1$, we multiply complex numbers just like polynomials. The product of z and w is

$$(a + bi)(c + di) = ac + bdi^2 + adi + bci = (ac - bd) + (ad + bc)i.$$

Every nonzero complex number $z = a + bi$ has a multiplicative inverse; that is, there exists a $z^{-1} \in \mathbb{C}^*$ such that $zz^{-1} = z^{-1}z = 1$. If $z = a + bi$, then

$$z^{-1} = \frac{a - bi}{a^2 + b^2}.$$

The **complex conjugate** of a complex number $z = a + bi$ is defined to be $\overline{z} = a - bi$. The **absolute value** or **modulus** of $z = a + bi$ is $|z| = \sqrt{a^2 + b^2}$.

Example 4.3.1 Complex Number Operations. Let $z = 2 + 3i$ and $w = 1 - 2i$. Then

$$z + w = (2 + 3i) + (1 - 2i) = 3 + i$$

and

$$zw = (2 + 3i)(1 - 2i) = 8 - i.$$

Also,

$$\begin{aligned} z^{-1} &= \frac{2}{13} - \frac{3}{13}i \\ |z| &= \sqrt{13} \\ \overline{z} &= 2 - 3i. \end{aligned}$$

□

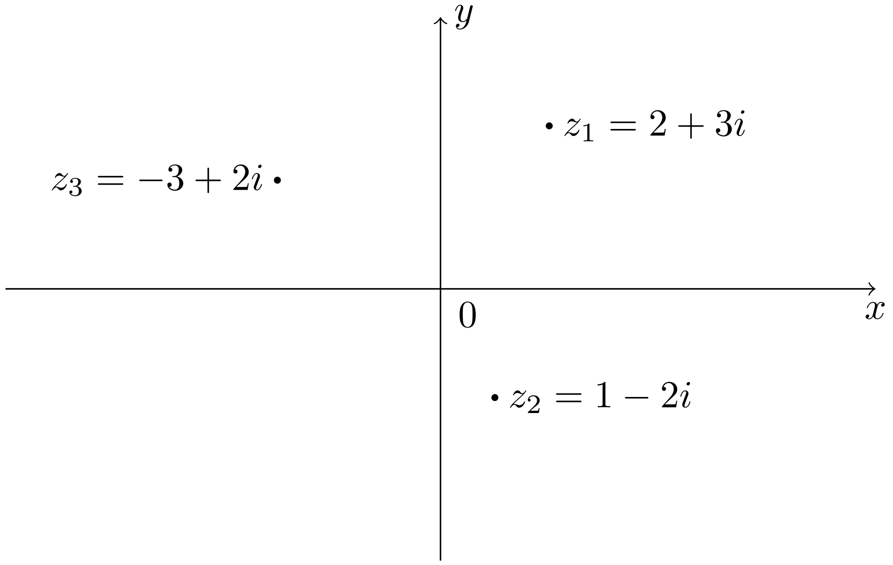


Figure 4.3.2 Rectangular coordinates of a complex number

There are several ways of graphically representing complex numbers. We can represent a complex number $z = a + bi$ as an ordered pair on the xy plane where a is the x (or real) coordinate and b is the y (or imaginary) coordinate. This is called the **rectangular** or **Cartesian** representation. The rectangular representations of $z_1 = 2 + 3i$, $z_2 = 1 - 2i$, and $z_3 = -3 + 2i$ are depicted in Figure 4.3.2.

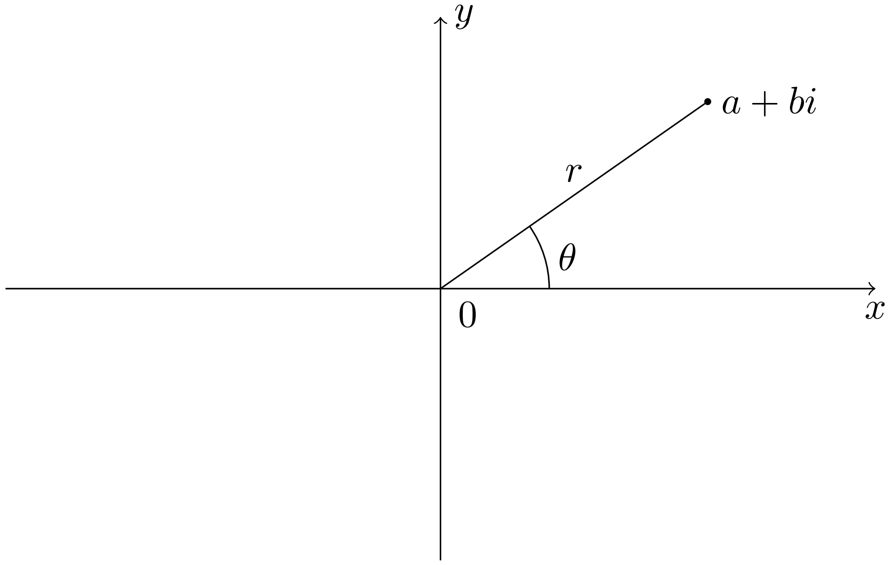


Figure 4.3.3 Polar coordinates of a complex number

Nonzero complex numbers can also be represented using **polar coordinates**. To specify any nonzero point on the plane, it suffices to give an angle θ from the positive x axis in the counterclockwise direction and a distance r from the origin, as in Figure 4.3.3. We can see that

$$z = a + bi = r(\cos\theta + i\sin\theta).$$

Hence,

$$r = |z| = \sqrt{a^2 + b^2}$$

and

$$a = r\cos\theta$$
$$b = r\sin\theta.$$

We sometimes abbreviate $r(\cos\theta + i\sin\theta)$ as $r\operatorname{cis}\theta$. To assure that the representation of z is well-defined, we also require that $0^\circ \leq \theta < 360^\circ$. If the measurement is in radians, then $0 \leq \theta < 2\pi$.

Example 4.3.4 Complex Numbers in Polar Form. Suppose that $z = 2\operatorname{cis}60^\circ$. Then

$$a = 2\cos 60^\circ = 1$$

and

$$b = 2\sin 60^\circ = \sqrt{3}.$$

Hence, the rectangular representation is $z = 1 + \sqrt{3}\,i$.

Conversely, if we are given a rectangular representation of a complex number, it is often useful to know the number's polar representation. If $z = 3\sqrt{2} - 3\sqrt{2}\,i$, then

$$r = \sqrt{a^2 + b^2} = \sqrt{36} = 6$$

and

$$\theta = \arctan\left(\frac{b}{a}\right) = \arctan(-1) = 315^\circ,$$

so $3\sqrt{2} - 3\sqrt{2}\,i = 6\operatorname{cis}315^\circ$. □

The polar representation of a complex number makes it easy to find products and powers of complex numbers. The proof of the following proposition is straightforward and is left as an exercise.

Proposition 4.3.5 *Let $z = r\operatorname{cis}\theta$ and $w = s\operatorname{cis}\phi$ be two nonzero complex numbers. Then*

$$zw = rs\operatorname{cis}(\theta + \phi).$$

Example 4.3.6 Multiplication of Complex Numbers in Polar Form. If $z = 3\operatorname{cis}(\pi/3)$ and $w = 2\operatorname{cis}(\pi/6)$, then $zw = 6\operatorname{cis}(\pi/2) = 6i$. □

Theorem 4.3.7 DeMoivre. *Let $z = r\operatorname{cis}\theta$ be a nonzero complex number. Then*

$$[r\operatorname{cis}\theta]^n = r^n\operatorname{cis}(n\theta)$$

for $n = 1, 2, \ldots$.

Proof. We will use induction on n (see Section 2.1). For $n = 1$ the theorem is trivial. Assume that the theorem is true for all k such that $1 \leq k \leq n$. Then

$$\begin{aligned} z^{n+1} &= z^n z \\ &= r^n(\cos n\theta + i\sin n\theta)r(\cos\theta + i\sin\theta) \\ &= r^{n+1}[(\cos n\theta\cos\theta - \sin n\theta\sin\theta) + i(\sin n\theta\cos\theta + \cos n\theta\sin\theta)] \\ &= r^{n+1}[\cos(n\theta + \theta) + i\sin(n\theta + \theta)] \\ &= r^{n+1}[\cos(n+1)\theta + i\sin(n+1)\theta]. \end{aligned}$$

■

Example 4.3.8 Powers of Complex Numbers. Suppose that $z = 1 + i$ and we wish to compute z^{10}. Rather than computing $(1+i)^{10}$ directly, it is much easier to switch to polar coordinates and calculate z^{10} using DeMoivre's

Theorem:

$$\begin{aligned} z^{10} &= (1+i)^{10} \\ &= \left(\sqrt{2}\operatorname{cis}\left(\frac{\pi}{4}\right)\right)^{10} \\ &= (\sqrt{2}\,)^{10}\operatorname{cis}\left(\frac{5\pi}{2}\right) \\ &= 32\operatorname{cis}\left(\frac{\pi}{2}\right) \\ &= 32i. \end{aligned}$$

□

The multiplicative group of the complex numbers, $\mathbb{C}^*$, possesses some interesting subgroups. Whereas $\mathbb{Q}^*$ and $\mathbb{R}^*$ have no interesting subgroups of finite order, $\mathbb{C}^*$ has many. We first consider the **circle group**,

$$\mathbb{T} = \{z \in \mathbb{C} : |z| = 1\}.$$

The following proposition is a direct result of Proposition 4.3.5.

Proposition 4.3.9 *The circle group is a subgroup of $\mathbb{C}^*$.*

Although the circle group has infinite order, it has many interesting finite subgroups. Suppose that $H = \{1, -1, i, -i\}$. Then H is a subgroup of the circle group. Also, 1, -1, i, and $-i$ are exactly those complex numbers that satisfy the equation $z^4 = 1$. The complex numbers satisfying the equation $z^n = 1$ are called the **nth roots of unity**.

Theorem 4.3.10 *If $z^n = 1$, then the nth roots of unity are*

$$z = \operatorname{cis}\left(\frac{2k\pi}{n}\right),$$

where $k = 0, 1, \ldots, n-1$. Furthermore, the nth roots of unity form a cyclic subgroup of $\mathbb{T}$ of order n

Proof. By DeMoivre's Theorem,

$$z^n = \operatorname{cis}\left(n\frac{2k\pi}{n}\right) = \operatorname{cis}(2k\pi) = 1.$$

The z's are distinct since the numbers $2k\pi/n$ are all distinct and are greater than or equal to 0 but less than 2π. We will leave the proof that the nth roots of unity form a cyclic subgroup of $\mathbb{T}$ as an exercise. ■

A generator for the group of the nth roots of unity is called a **primitive nth root of unity**.

Example 4.3.11 Roots of Unity. The 8th roots of unity can be represented as eight equally spaced points on the unit circle (Figure 4.3.12). The primitive 8th roots of unity are

$$\begin{aligned} \omega &= \frac{\sqrt{2}}{2} + \frac{\sqrt{2}}{2}i \\ \omega^3 &= -\frac{\sqrt{2}}{2} + \frac{\sqrt{2}}{2}i \\ \omega^5 &= -\frac{\sqrt{2}}{2} - \frac{\sqrt{2}}{2}i \\ \omega^7 &= \frac{\sqrt{2}}{2} - \frac{\sqrt{2}}{2}i. \end{aligned}$$

□

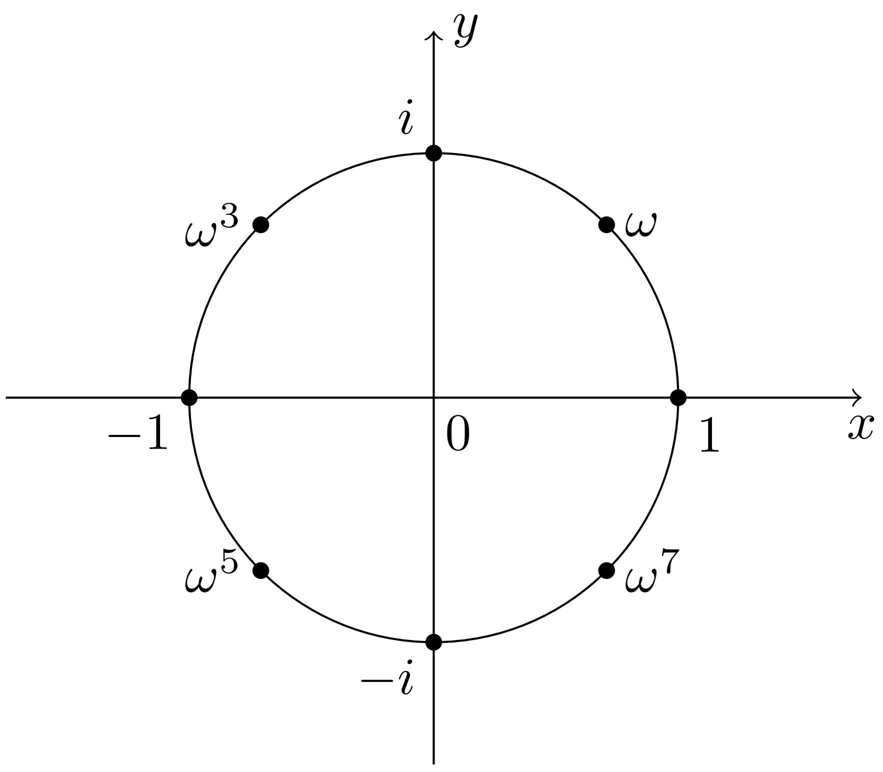


Figure 4.3.12 8th roots of unity

We interrupt this exposition to repeat the previous diagram, wrapped as different figure with a different caption. The TikZ code to produce these diagrams lives in an external file, `tikz/cyclic-roots-unity.tex`, which is pure text, freed from any need to format for XML processing. So, in particular, there is no need to escape ampersands and angle brackets, nor is there employment of the `CDATA` mechanism. But the real value is that there is just one version to edit, and any changes will be reflected in both copies.

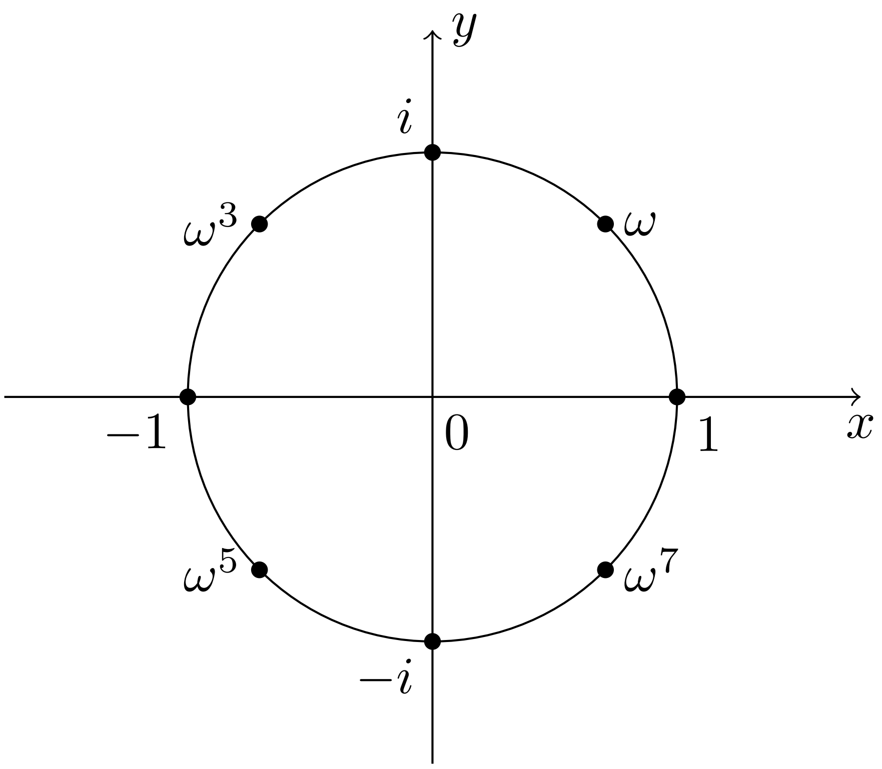


Figure 4.3.13 Repeat: 8th roots of unity

4.4 Large Powers of Integers

Computing large powers can be very time-consuming. Just as anyone can compute 2^2 or 2^8, everyone knows how to compute

$$2^{2^{1000000}}.$$

However, such numbers are so large that we do not want to attempt the calculations; moreover, past a certain point the computations would not be feasible even if we had every computer in the world at our disposal. Even writing down the decimal representation of a very large number may not be reasonable. It could be thousands or even millions of digits long. However, if we could compute something like $2^{37398332} \pmod{46389}$, we could very easily write the result down since it would be a number between 0 and 46,388. If we want to compute powers modulo n quickly and efficiently, we will have to be clever.[1]

The first thing to notice is that any number a can be written as the sum of distinct powers of 2; that is, we can write

$$a = 2^{k_1} + 2^{k_2} + \cdots + 2^{k_n},$$

where $k_1 < k_2 < \cdots < k_n$. This is just the binary representation of a. For example, the binary representation of 57 is 111001, since we can write $57 = 2^0 + 2^3 + 2^4 + 2^5$.

The laws of exponents still work in $\mathbb{Z}_n$; that is, if $b \equiv a^x \pmod{n}$ and $c \equiv a^y \pmod{n}$, then $bc \equiv a^{x+y} \pmod{n}$. We can compute $a^{2^k} \pmod{n}$ in k

[1]The results in this section are needed only in Chapter 2 (not really).

multiplications by computing

$$\begin{gathered} a^{2^0} \pmod{n} \\ a^{2^1} \pmod{n} \\ \vdots \\ a^{2^k} \pmod{n}. \end{gathered}$$

Each step involves squaring the answer obtained in the previous step, dividing by n, and taking the remainder.

Example 4.4.1 Repeated Squares. We will compute $271^{321} \pmod{481}$. Notice that

$$321 = 2^0 + 2^6 + 2^8;$$

hence, computing $271^{321} \pmod{481}$ is the same as computing

$$271^{2^0+2^6+2^8} \equiv 271^{2^0} \cdot 271^{2^6} \cdot 271^{2^8} \pmod{481}.$$

So it will suffice to compute $271^{2^i} \pmod{481}$ where $i = 0, 6, 8$. It is very easy to see that

$$271^{2^1} = 73{,}441 \equiv 329 \pmod{481}.$$

We can square this result to obtain a value for $271^{2^2} \pmod{481}$:

$$\begin{aligned} 271^{2^2} &\equiv (271^{2^1})^2 \pmod{481} \\ &\equiv (329)^2 \pmod{481} \\ &\equiv 108{,}241 \pmod{481} \\ &\equiv 16 \pmod{481}. \end{aligned}$$

We are using the fact that $(a^{2^n})^2 \equiv a^{2 \cdot 2^n} \equiv a^{2^{n+1}} \pmod{n}$. Continuing, we can calculate

$$271^{2^6} \equiv 419 \pmod{481}$$

and

$$271^{2^8} \equiv 16 \pmod{481}.$$

Therefore,

$$\begin{aligned} 271^{321} &\equiv 271^{2^0+2^6+2^8} \pmod{481} \\ &\equiv 271^{2^0} \cdot 271^{2^6} \cdot 271^{2^8} \pmod{481} \\ &\equiv 271 \cdot 419 \cdot 16 \pmod{481} \\ &\equiv 1{,}816{,}784 \pmod{481} \\ &\equiv 47 \pmod{481}. \end{aligned}$$

□

The method of repeated squares will prove to be a very useful tool when we explore RSA cryptography. To encode and decode messages in a reasonable manner under this scheme, it is necessary to be able to quickly compute large powers of integers mod n.

Remark 4.4.2 Sage. Sage support for cyclic groups is a little spotty — but we can still make effective use of Sage and perhaps this situation could change soon.

4.5 Exercises

1. Prove or disprove each of the following statements.

(a) All of the generators of $\mathbb{Z}_{60}$ are prime.

(b) $U(8)$ is cyclic.

(c) $\mathbb{Q}$ is cyclic.

(d) If every proper subgroup of a group G is cyclic, then G is a cyclic group.

(e) A group with a finite number of subgroups is finite.

2. Find the order of each of the following elements.

(a) $5 \in \mathbb{Z}_{12}$

(b) $\sqrt{3} \in \mathbb{R}$

(c) $\sqrt{3} \in \mathbb{R}^*$

(d) $-i \in \mathbb{C}^*$

(e) 72 in $\mathbb{Z}_{240}$

(f) 312 in $\mathbb{Z}_{471}$

3. List all of the elements in each of the following subgroups.

(a) The subgroup of $\mathbb{Z}$ generated by 7

(b) The subgroup of $\mathbb{Z}_{24}$ generated by 15

(c) All subgroups of $\mathbb{Z}_{12}$

(d) All subgroups of $\mathbb{Z}_{60}$

(e) All subgroups of $\mathbb{Z}_{13}$

(f) All subgroups of $\mathbb{Z}_{48}$

(g) The subgroup generated by 3 in $U(20)$

(h) The subgroup generated by 5 in $U(18)$

(i) The subgroup of $\mathbb{R}^*$ generated by 7

(j) The subgroup of $\mathbb{C}^*$ generated by i where $i^2 = -1$

(k) The subgroup of $\mathbb{C}^*$ generated by $2i$

(l) The subgroup of $\mathbb{C}^*$ generated by $(1+i)/\sqrt{2}$

(m) The subgroup of $\mathbb{C}^*$ generated by $(1+\sqrt{3}\,i)/2$

4. Find the subgroups of $GL_2(\mathbb{R})$ generated by each of the following matrices.

(a) $\begin{pmatrix} 0 & 1 \\ -1 & 0 \end{pmatrix}$

(b) $\begin{pmatrix} 0 & 1/3 \\ 3 & 0 \end{pmatrix}$

(c) $\begin{pmatrix} 1 & -1 \\ 1 & 0 \end{pmatrix}$

(d) $\begin{pmatrix} 1 & -1 \\ 0 & 1 \end{pmatrix}$

(e) $\begin{pmatrix} 1 & -1 \\ -1 & 0 \end{pmatrix}$

(f) $\begin{pmatrix} \sqrt{3}/2 & 1/2 \\ -1/2 & \sqrt{3}/2 \end{pmatrix}$

5. Find the order of every element in $\mathbb{Z}_{18}$.

6. Find the order of every element in the symmetry group of the square, D_4.

7. What are all of the cyclic subgroups of the quaternion group, Q_8?

8. List all of the cyclic subgroups of $U(30)$.

9. List every generator of each subgroup of order 8 in $\mathbb{Z}_{32}$.

10. Find all elements of finite order in each of the following groups. Here the "$*$" indicates the set with zero removed.

(a) $\mathbb{Z}$ (b) $\mathbb{Q}^*$ (c) $\mathbb{R}^*$

11. If $a^{24} = e$ in a group G, what are the possible orders of a?

12. Find a cyclic group with exactly one generator. Can you find cyclic groups with exactly two generators? Four generators? How about n generators?

13. For $n \leq 20$, which groups $U(n)$ are cyclic? Make a conjecture as to what is true in general. Can you prove your conjecture?

14. Let

$$A = \begin{pmatrix} 0 & 1 \\ -1 & 0 \end{pmatrix} \qquad \text{and} \qquad B = \begin{pmatrix} 0 & -1 \\ 1 & -1 \end{pmatrix}$$

be elements in $GL_2(\mathbb{R})$. Show that A and B have finite orders but AB does not.

15. Evaluate each of the following.

(a) $(3 - 2i) + (5i - 6)$

(b) $(4 - 5i) - \overline{(4i - 4)}$

(c) $(5 - 4i)(7 + 2i)$

(d) $(9 - i)\overline{(9 - i)}$

(e) i^{45}

(f) $(1 + i) + \overline{(1 + i)}$

16. Convert the following complex numbers to the form $a + bi$.

(a) $2 \operatorname{cis}(\pi/6)$

(b) $5 \operatorname{cis}(9\pi/4)$

(c) $3 \operatorname{cis}(\pi)$

(d) $\operatorname{cis}(7\pi/4)/2$

17. Change the following complex numbers to polar representation.

(a) $1 - i$

(b) -5

(c) $2 + 2i$

(d) $\sqrt{3} + i$

(e) $-3i$

(f) $2i + 2\sqrt{3}$

18. Calculate each of the following expressions.

(a) $(1 + i)^{-1}$

(b) $(1 - i)^6$

(c) $(\sqrt{3} + i)^5$

(d) $(-i)^{10}$

(e) $((1 - i)/2)^4$

(f) $(-\sqrt{2} - \sqrt{2}\, i)^{12}$

(g) $(-2 + 2i)^{-5}$

19. Prove each of the following statements.

(a) $|z| = |\overline{z}|$

(b) $z\overline{z} = |z|^2$

(c) $z^{-1} = \overline{z}/|z|^2$

(d) $|z + w| \leq |z| + |w|$

(e) $|z - w| \geq ||z| - |w||$

(f) $|zw| = |z||w|$

20. List and graph the 6th roots of unity. What are the generators of this group? What are the primitive 6th roots of unity?

21. List and graph the 5th roots of unity. What are the generators of this group? What are the primitive 5th roots of unity?

22. Calculate each of the following.

(a) $292^{3171} \pmod{582}$
(b) $2557^{341} \pmod{5681}$
(c) $2071^{9521} \pmod{4724}$
(d) $971^{321} \pmod{765}$

23. Let $a, b \in G$. Prove the following statements.

(a) The order of a is the same as the order of a^{-1}.

(b) For all $g \in G$, $|a| = |g^{-1}ag|$.

(c) The order of ab is the same as the order of ba.

24. Let p and q be distinct primes. How many generators does $\mathbb{Z}_{pq}$ have?

25. Let p be prime and r be a positive integer. How many generators does $\mathbb{Z}_{p^r}$ have?

26. Prove that $\mathbb{Z}_p$ has no nontrivial subgroups if p is prime.

27. If g and h have orders 15 and 16 respectively in a group G, what is the order of $\langle g \rangle \cap \langle h \rangle$?

28. Let a be an element in a group G. What is a generator for the subgroup $\langle a^m \rangle \cap \langle a^n \rangle$?

29. Prove that $\mathbb{Z}_n$ has an even number of generators for $n > 2$.

30. Suppose that G is a group and let a, $b \in G$. Prove that if $|a| = m$ and $|b| = n$ with $\gcd(m, n) = 1$, then $\langle a \rangle \cap \langle b \rangle = \{e\}$.

31. Let G be an abelian group. Show that the elements of finite order in G form a subgroup. This subgroup is called the **torsion subgroup** of G.

32. Let G be a finite cyclic group of order n generated by x. Show that if $y = x^k$ where $\gcd(k, n) = 1$, then y must be a generator of G.

33. If G is an abelian group that contains a pair of cyclic subgroups of order 2, show that G must contain a subgroup of order 4. Does this subgroup have to be cyclic?

34. Let G be an abelian group of order pq where $\gcd(p, q) = 1$. If G contains elements a and b of order p and q respectively, then show that G is cyclic.

35. Prove that the subgroups of $\mathbb{Z}$ are exactly $n\mathbb{Z}$ for $n = 0, 1, 2, \ldots$.

36. Prove that the generators of $\mathbb{Z}_n$ are the integers r such that $1 \leq r < n$ and $\gcd(r, n) = 1$.

37. Prove that if G has no proper nontrivial subgroups, then G is a cyclic group.

38. Prove that the order of an element in a cyclic group G must divide the order of the group.

39. Prove that if G is a cyclic group of order m and $d \mid m$, then G must have a subgroup of order d.

40. For what integers n is -1 an nth root of unity?

41. If $z = r(\cos\theta + i\sin\theta)$ and $w = s(\cos\phi + i\sin\phi)$ are two nonzero complex numbers, show that

$$zw = rs[\cos(\theta + \phi) + i\sin(\theta + \phi)].$$

42. Prove that the circle group is a subgroup of $\mathbb{C}^*$.

43. Prove that the nth roots of unity form a cyclic subgroup of $\mathbb{T}$ of order n.

44. Let $\alpha \in \mathbb{T}$. Prove that $\alpha^m = 1$ and $\alpha^n = 1$ if and only if $\alpha^d = 1$ for $d = \gcd(m, n)$.

45. Let $z \in \mathbb{C}^*$. If $|z| \neq 1$, prove that the order of z is infinite.

46. Let $z = \cos\theta + i\sin\theta$ be in $\mathbb{T}$ where $\theta \in \mathbb{Q}$. Prove that the order of z is infinite.

4.6 Programming Exercises

1. Write a computer program that will write any decimal number as the sum of distinct powers of 2. What is the largest integer that your program will handle?

2. Write a computer program to calculate $a^x \pmod{n}$ by the method of repeated squares. What are the largest values of n and x that your program will accept?

4.7 Sage Exercises

This group of exercises is about the group of units mod n, $U(n)$, which is sometimes cyclic, sometimes not. There are some commands in Sage that will answer some of these questions very quickly, but instead of using those now, just use the basic techniques described. The idea here is to just work with elements, and lists of elements, to discern the subgroup structure of these groups.

1. Execute the statement `R = Integers(40)` to create the set `[0,1,2,...,39]` This is a group under addition mod 40, which we will ignore. Instead we are interested in the subset of elements which have an inverse under *multiplication* mod 40. Determine how big this subgroup is by executing the command `R.unit_group_order()`, and then obtain a list of these elements with `R.list_of_elements_of_multiplicative_group()`.

2. You can create elements of this group by coercing regular integers into `U`, such as with the statement `a = U(7)`. (Don't confuse this with our mathematical notation $U(40)$.) This will tell Sage that you want to view 7 as an element of U, subject to the corresponding operations. Determine the elements of the cyclic subgroup of U generated by 7 with a list comprehension as follows:

```
R = Integers(40)
a = R(7)
[a^i for i in srange(16)]
```

What is the order of 7 in $U(40)$?

3. The group $U(49)$ is cyclic. Using only the Sage commands described previously, use Sage to find a generator for this group. Now using *only* theorems about the structure of cyclic groups, describe each of the subgroups of $U(49)$ by specifying its order and by giving an explicit generator. Do not repeat any of the subgroups — in other words, present each subgroup *exactly* once. You can use Sage to check your work on the subgroups, but your answer about the subgroups should rely only on theorems and be a nicely written paragraph with a table, etc.

4. The group $U(35)$ is not cyclic. Again, using only the Sage commands described previously, use computations to provide irrefutable evidence of this. How many of the 16 different subgroups of $U(35)$ can you list?

5. Again, using only the Sage commands described previously, explore the structure of $U(n)$ for various values of n and see if you can formulate an interesting conjecture about some basic property of this group. (Yes, this is a *very* open-ended question, but this is ultimately the real power of exploring mathematics with Sage.)

4.8 References and Suggested Readings

[1] Koblitz, N. *A Course in Number Theory and Cryptography.* 2nd ed. Springer, New York, 1994.

[2] Pomerance, C. "Cryptology and Computational Number Theory—An Introduction," in *Cryptology and Computational Number Theory*, Pomerance, C., ed. Proceedings of Symposia in Applied Mathematics, vol. 42, American Mathematical Society, Providence, RI, 1990. This book gives an excellent account of how the method of repeated squares is used in cryptography.

Chapter 5

Runestone Testing

We collect Runestone interactive items for testing here, in sections of their own.

5.1 ActiveCode

Python programs are made interactive in HTML, on request.

```
print("Hello, World!")
```

Listing 5.1.1 An interactive Python program, using *Runestone*

A C program will only be interactive if hosted on a Runestone server.

```
#include <stdio.h>

int main(void)
{
    puts("Hello, world!");
}
```

Listing 5.1.2 An C program, interactive on a *Runestone* server

A Java program will only be interactive if hosted on a Runestone server.

```
import javax.swing.JFrame;  //Importing class JFrame
import javax.swing.JLabel;  //Importing class JLabel
public class HelloWorld {
    public static void main(String[] args) {
        JFrame frame = new JFrame();           //Creating
            frame
        frame.setTitle("Hi!");                 //Setting
            title frame
        frame.add(new JLabel("Hello, world!"));//Adding
            text to frame
        frame.pack();                          //Setting
            size to smallest
        frame.setLocationRelativeTo(null);     //Centering
            frame
        frame.setVisible(true);                //Showing
            frame
    }
}
```

Listing 5.1.3 A Java program, interactive on a *Runestone* server

Javascript programs are made interactive in HTML, on request.

```
document.write('Hello, world!');
```

Listing 5.1.4 An interactive JavaScript program, using *Runestone*

A `<program>` is not interactive, by default, no matter where it is hosted.

```
import javax.swing.JFrame;  //Importing class JFrame
import javax.swing.JLabel;  //Importing class JLabel
public class HelloWorld {
    public static void main(String[] args) {
        JFrame frame = new JFrame();           //Creating
            frame
        frame.setTitle("Hi!");                 //Setting
            title frame
        frame.add(new JLabel("Hello, world!"));//Adding
            text to frame
        frame.pack();                          //Setting
            size to smallest
        frame.setLocationRelativeTo(null);     //Centering
            frame
        frame.setVisible(true);                //Showing
            frame
    }
}
```

Listing 5.1.5 A static Java program

An Octave program will only be interactive if hosted on a Runestone server. Octave is meant to be a drop-in replacement for Matlab.

```
x = 2 + 2
printf("%d\n", x)
```

Listing 5.1.6 A simple Octave program

A language not supported by Runestone Services will be rendered static.

```
program HelloWorld;
begin
  WriteLn('Hello, world!');
end.
```

Listing 5.1.7 A Pascal program that cannot be interactive on Runestone

The following Python program is in a `<listing>` since we will want to reference it shortly. The program does not do very much, it just defines four variables whose values are lists of statistics. It should run, and there will be no syntax errors, but it is a bit boring since there is no output.

```
loan_amount = [1250.0, 500.0, 1450.0, 200.0, 700.0, 100.0,
    250.0, 225.0, 1200.0, 150.0, 600.0, 300.0, 700.0,
    125.0, 650.0, 175.0, 1800.0, 1525.0, 575.0, 700.0,
    1450.0, 400.0, 200.0, 1000.0, 350.0]

country_name = ['Azerbaijan', 'El Salvador', 'Bolivia',
    'Paraguay', 'El Salvador', 'Philippines',
    'Philippines', 'Nicaragua', 'Guatemala', 'Philippines',
    'Paraguay', 'Philippines', 'Bolivia', 'Philippines',
    'Philippines', 'Madagascar', 'Georgia', 'Uganda',
    'Kenya', 'Tajikistan', 'Jordan', 'Kenya',
    'Philippines', 'Ecuador', 'Kenya']

time_to_raise = [193075.0, 1157108.0, 1552939.0, 244945.0,
    238797.0, 1248909.0, 773599.0, 116181.0, 2288095.0,
    51668.0, 26717.0, 48030.0, 1839190.0, 71117.0,
    580401.0, 800427.0, 1156218.0, 1166045.0, 2924705.0,
    470622.0, 24078.0, 260044.0, 445938.0, 201408.0,
    2370450.0]

num_lenders_total = [38, 18, 51, 3, 21, 1, 10, 8, 42, 1,
    18, 6, 28, 5, 16, 7, 54, 1, 18, 22, 36, 12, 8, 24, 8]
```

Listing 5.1.8 A Python program that defines some statistics

Now a programming exercise. The program upcoming is going to **include** all the code of the program preceding. This is accomplished with an `@include` attribute on the *including* program whose value is the `@xml:id` of the *included* program. So by running the next program, it should pass all of its three tests (for example another example using unit tests, see Checkpoint 5.3.4). Now reload the page, do not run the program in the listing, and then see that the program in the exercise still runs correctly.

You'll see nothing that tells the reader that the one chunk of code is prefacing the other. And in static formats it might be even less obvious. So you will want to say *something* to alert the reader. Here it is easy: Checkpoint 5.1.9 includes all the code from Listing 5.1.8.

Checkpoint 5.1.9 A Python program, including another. Compute the total amount of money loaned and store it in the variable `loan_total`.

```
loan_total = 0
for loan in loan_amount:
    loan_total += loan
print(loan_total)
```

Exact same exercise again, but now we include *two* programs. We first get the simple "Hello, world!" program at Listing 5.1.1 and then the same program defining the variables with lists of statistics at Listing 5.1.8. So the output just includes the extra result from the `print()` statement.

Checkpoint 5.1.10 A Python program, including two others. Compute the total amount of money loaned and store it in the variable `loan_total`.

```
loan_total = 0
for loan in loan_amount:
    loan_total += loan
print(loan_total)
```

Here is an activecode with `@language` set to `sql` uses the `@database` to load a SQLite database file.

Checkpoint 5.1.11 An SQL program that uses an SQLite database file. Select all the columns of all the rows in the `test` database table.

```
SELECT * FROM test
```

A nonsense paragraph just to check on spacing. A nonsense paragraph just to check on spacing. A nonsense paragraph just to check on spacing. A nonsense paragraph just to check on spacing.

5.2 Code Lens

CodeLens is an interactive tool for following program execution, much like a debugger, without the ability to influence flow control or variable values. For use without a server, traces must be computed beforehand. First, we have some trivial programs, to provide minimal testing.

```
print('Hello, World!')
```

Listing 5.2.1 A Python program, stepable with CodeLens

```
#include <stdio.h>

int main(void)
{
    puts("Hello, World!");
}
```

Listing 5.2.2 An C program, stepable with CodeLens

```
public class HelloWorld {
    public static void main(String[] args) {
        System.out.println("Hello, World!");
    }
}
```

Listing 5.2.3 A Java program, stepable with CodeLens

Now some moderately more complicated programs to find teh prime numbers less than 20. We do not vouch for the quality of these, or even their correctness!

```
def SieveOfEratosthenes(n):
   # array of type boolean with True values in it
   prime = [True for i in range(n + 1)]
   p = 2
   while (p * p <= n):
      # If it remain unchanged it is prime
      if (prime[p] == True):
         # updating all the multiples
         for i in range(p * 2, n + 1, p):
            prime[i] = False
      p += 1
   prime[0]= False
   prime[1]= False
   # Print
   for p in range(n + 1):
      if prime[p]:
         print (p,end=" ")
# main
if __name__=='__main__':
   n = 20
   print ("The prime numbers smaller than or equal to",
       n,"is")
   SieveOfEratosthenes(n)
```

Listing 5.2.4 Sieve of Eratosthenes[1], Java

```
#include <stdio.h>
const int len = 20;
int main() {
   int arr[20] = {0};
   for (int i = 2; i < len; i++) {
      for (int j = i * i; j < len; j+=i) {
         arr[j - 1] = 1;
      }
   }
   for (int i = 1; i < len; i++) {
      if (arr[i - 1] == 0)
        printf(" %d", i);
   }
}
```

Listing 5.2.5 Sieve of Eratosthenes[2], C++

[1] www.tutorialspoint.com/python-program-for-sieve-of-eratosthenes
[2] www.tutorialspoint.com/cplusplus-program-to-implement-sieve-of-eratosthenes-to-generate-prime-numbers-

```
public class SievePrimeFactors  {
   public static void main(String args[]) {
      int num = 20;
      boolean[] bool = new boolean[num];

      for (int i = 0; i< bool.length; i++) {
         bool[i] = true;
      }
      for (int i = 2; i < Math.sqrt(num); i++) {
         if(bool[i] == true) {
            for(int j = (i*i); j < num; j = j+i) {
               bool[j] = false;
            }
         }
      }
      System.out.println("List of prime numbers: ");
      for (int i = 2; i< bool.length; i++) {
         if(bool[i]==true) {
            System.out.println(i);
         }
      }
   }
}
```

Listing 5.2.6 Sieve of Eratosthenes[3], Java

5.3 Coding Exercises

Program listings can be more that just live demonstrations, they can be exercises. The first two also occur in the sample article where they just get a static rendering, if at all.

Checkpoint 5.3.1 Inline Coding Exercise, No Help. An exercise might ask a reader to write a computer program, that would go here in the `<statement>`. But you can also add a `<program>` element after a `<statement>`. Here we place no code at all, but we do say we want it to be interactive. The purpose is to make it a live coding environment for a version of your output that allows the reader to perhaps submit a solution. The `<program>` element is necessary so you can specify a programming language.

In interactive formats, try creating and running a Python program below. Use CodeLens to step through the program.

Hint. We didn't really ask you to do anything.

Checkpoint 5.3.2 Inline Coding Exercise, Partial. Similar to above, but we provide a starting point for the exercise.

```
#include <stdio.h>

int main(void)
```

Answer. We're not really sure. But it would begin as follows:

```
#include <stdio.h>

int main(void)
```

[3] www.tutorialspoint.com/Sieve-of-Eratosthenes-in-java

Activity 5.3.1 Activity Coding Exercise. Similar to above, but now as a complete Python program inside an `<activity>`. This demonstrates the possibility to use any "project-like" block (`<project>`, `<activity>`, `<exploration>`, `<investigation>`), but not in the case when structured with `<task>`. (There is an empty `<tests>` element here, designed to test relief for an error this will cause on a Runestone server.)

```
for i in range(10):
    print(i)
```

Answer. We're still not really sure.

Checkpoint 5.3.3 An Exercise with a Static Program. Similar to above, again, but we place the `<program>` element *inside* the `<statement>`, not after it as a peer. This signals that this is not a coding exercise and the program will render static, since it is explicitly labeled as not being interactive.

```
#include <stdio.h>

int main(void)
```

Solution. We're not really sure. Still.

Checkpoint 5.3.4 Coding Exercise, with Unit Tests. Fix the following code so that it always correctly adds two numbers. [Ed. Unit test support is experimental.]

```
def add(a,b):
    return 4
```

Answer. We're not really sure. But it would begin as follows:

```
#include <stdio.h>

int main(void)
```

5.4 Data Files

In the following file of climate data, the first column is Year, second column is Global Average Temperature (Celcius), and the third column is Global Emmisions C02 (Giga-tons). [Normally you might place this inside a block with the `<datafile>`.]

```
Data: ccdata1.txt

1850    -0.37     2.24E-7
1860    -0.34     3.94E-7
1870    -0.28     6.6E-7
1880    -0.24     1.1
1890    -0.42     1.72
1900    -0.2      2.38
1910    -0.49     3.34
1920    -0.25     4.01
1930    -0.14     4.53
1940     0.01     5.5
1950    -0.17     6.63
1960    -0.05     10.5
1970    -0.03     16
1980     0.09     20.3
```

```
1990     0.3      22.6
2000     0.29     24.9
2010     0.56     32.7
2019     0.74     33.3
```

```python
ccfile = open("ccdata1.txt", "r")

for aline in ccfile:
    values = aline.split()
    print('In', values[0], 'the average temp. was',
        values[1], '°C and CO2 emmisions were', values[2],
        'gigatons.')

ccfile.close()
```

Data 5.4.1 Stack Overflow Developer Survey. [A data file can go lots of places. But to make it more prominent, and easy to cross-reference, it would be natural to put it into a `<data>` block.]

Now that you are experienced with working with files lets look at a bit of the data set. The survey had 98,855 respondents. We will work with a sample of 2000 of those responses for this lab. In addition we have narrowed down the questions from 129 to just 13. The columns we have included in this data set are:

1. Respondent
2. Country
3. JobSatisfaction
4. UndergradMajor
5. ConvertedSalary
6. Exercise
7. Gender
8. RaceEthnicity
9. EducationParents
10. HoursOutside
11. Age
12. LastNewJob
13. LanguageWorkedWith

Data: so_survey.csv

```
Respondent|Country|JobSatisfaction|UndergradMajor|ConvertedSalar→
51900|United Kingdom|Moderately satisfied|Computer science, comp→
95836|Argentina|Slightly satisfied|A business discipline (ex. ac→
51710|Germany|Slightly dissatisfied|A social science (ex. anthro→
44125|United States|Moderately dissatisfied|A social science (ex→
35167|United Kingdom|Extremely satisfied|A humanities discipline→
31721|Japan|Slightly dissatisfied|Information systems, informati→
36729|Brazil|Moderately dissatisfied|Computer science, computer →
38620|Germany|Moderately dissatisfied|Computer science, computer→
54695|Netherlands|Moderately satisfied|Computer science, compute→
```

Data: luther-bell.jpg

```
import image
img = image.Image("luther-bell.jpg")

print(img.getWidth())
print(img.getHeight())

p = img.getPixel(45, 55)
print(p.getRed(), p.getGreen(), p.getBlue())
```

Computation 5.4.2 Golden Gate Bridge Image processing. This image has a Creative Commons license, but we've lost track of the exact terms.

[Now a data file and a program to process it, all bundled up inside a `<computation>`, since an `<example>` gets knowled and the ActiveCode does not fill.]

Data: golden_gate.png

This program changes every pixel of the image.

```
import image

img = image.Image("golden_gate.png")
win = image.ImageWin(img.getWidth(), img.getHeight())
img.draw(win)

# img.setDelay(delay, number of pixels between delay)
# setDelay(1, 400) will speed up a lot
img.setDelay(1,15)

for row in range(img.getHeight()):
    for col in range(img.getWidth()):
        p = img.getPixel(col, row)

        newred = p.red * 1.4
        newgreen =  p.green * .75
        newblue =  p.blue * 1.1

        newpixel = image.Pixel(newred, newgreen, newblue)

        img.setPixel(col, row, newpixel)

img.draw(win)
win.exitonclick()
```

The examples above all use Python, which will run in your browser. Other languages will only run when a project is hosted on Runestone Academy servers. And in this case there is a small twist. You need to indicate which existing `<datafile>` your program needs, even if that seems obvious by reading the code. Use a `@datafile` attribute on `<program>` that has a list of filenames. These are the filenames you set via the `@filename` attribute of the `<datafile>` element, and are the names you use in your program's code. As before, no path information is neede, nor allowed.

[2023-02-21: testing for single files first, list of several not yet implemented.]

When the `@language` attribute of a `<program>` is set to `python3` that means in-browser Python is not good enough, and you want the greater power and flexibility of having your code run on a Runestone Academy server. So this is our first example of using the `@datafile` attribute.

The data file is an abbreviated version of the example above, just to be different. And is not editable.

`Data: ccdata2.txt`

```
1900    -0.2     2.38
1910    -0.49    3.34
1920    -0.25    4.01
1930    -0.14    4.53
```

The program is identical to the above, but we specify `python3` as the language, and use the smaller file. So this example is only active when this content is hosted on a Runestone Academy server.

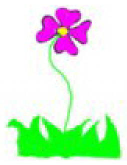
```
ccfile = open("ccdata2.txt", "r")

for aline in ccfile:
    values = aline.split()
    print('In', values[0], 'the average temp. was',
        values[1], '°C and CO2 emmisions were', values[2],
        'gigatons.')
```

```
ccfile.close()
```

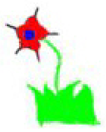
[2023-02-24 Testing Java with multiple data files]

Data 5.4.3 Flowers. Two flower images as `<datafile>` for use in upcoming Java program.

Data: flower1.jpg



Data: flower2.jpg



```
import java.awt.*;
import java.awt.font.*;
import java.awt.geom.*;
import java.awt.image.BufferedImage;
import java.text.*;
import java.util.*;
import java.util.List;

/**
 * A class that represents a picture.  This class inherits
     from
 * SimplePicture and allows the student to add
     functionality to
 * the Picture class.
 *
 * @author Barbara Ericson ericson@cc.gatech.edu
 */
public class Picture extends SimplePicture
{
  ///////////////////// constructors
      //////////////////////////////////

  /**
   * Constructor that takes no arguments
   */
  public Picture ()
  {
    /* not needed but use it to show students the implicit
        call to super()
     * child constructors always call a parent constructor
     */
    super();
  }

  /**
   * Constructor that takes a file name and creates the
       picture
   * @param fileName the name of the file to create the
```

```
        picture from
     */
    public Picture(String fileName)
    {
      // let the parent class handle this fileName
      super(fileName);
    }

    /**
     * Constructor that takes the height and width
     * @param height the height of the desired picture
     * @param width the width of the desired picture
     */
    public Picture(int width, int height)
    {
      // let the parent class handle this width and height
      super(width,height);
    }

    /**
     * Constructor that takes a picture and creates a
     * copy of that picture
     * @param copyPicture the picture to copy
     */
    public Picture(Picture copyPicture)
    {
      // let the parent class do the copy
      super(copyPicture);
    }

    /**
     * Constructor that takes a buffered image
     * @param image the buffered image to use
     */
    public Picture(BufferedImage image)
    {
      super(image);
    }
    ////////////////////// methods
        ///////////////////////////////////////

    /**
     * Method to return a string with information about this
        picture.
     * @return a string with information about the picture
        such as fileName,
     * height and width.
     */
    public String toString()
    {
      String output = "Picture, filename " + getFileName() +
        " height " + getHeight()
        + " width " + getWidth();
      return output;
    }

    /**
      zeroBlue() method sets the blue values at all pixels to
         zero
```

```
  */
   public void zeroBlue()
   {
     Pixel[][] pixels = this.getPixels2D();

     for (Pixel[] rowArray : pixels)
      {
        for (Pixel p: rowArray)
        {
               p.setBlue(0);
        }
     }
   }

   /* mirrorVertical() */
   public void mirrorVertical()
   {
        Pixel[][] pixels = this.getPixels2D();
        Pixel leftPixel = null;
        Pixel rightPixel = null;
        int width = pixels[0].length;
        for (int row = 0; row < pixels.length; row++)
        {
             for (int col = 0; col < width / 2; col++)
             {
                  leftPixel = pixels[row][col];
                  rightPixel = pixels[row][width - 1 - col];
                  rightPixel.setColor(leftPixel.getColor());
             }
        }
   }

    /** copy from the passed fromPic to the
      * specified startRow and startCol in the
      * current picture
      * @param fromPic the picture to copy from
      * @param startRow the start row to copy to
      * @param startCol the start col to copy to
      */
    public void copy(Picture fromPic,
                   int startRow, int startCol)
    {
      Pixel fromPixel = null;
      Pixel toPixel = null;
      Pixel[][] toPixels = this.getPixels2D();
      Pixel[][] fromPixels = fromPic.getPixels2D();
      for (int fromRow = 0, toRow = startRow;
           fromRow < fromPixels.length &&
           toRow < toPixels.length;
           fromRow++, toRow++)
      {
        for (int fromCol = 0, toCol = startCol;
             fromCol < fromPixels[0].length &&
             toCol < toPixels[0].length;
             fromCol++, toCol++)
        {
          fromPixel = fromPixels[fromRow][fromCol];
          toPixel = toPixels[toRow][toCol];
          toPixel.setColor(fromPixel.getColor());
        }
      }
    }

   public void createCollage()
   {
        // You can also try butterfly.jpg and snowflake.jpg
        Picture flower1 = new Picture("flower1.jpg");
        Picture flower2 = new Picture("flower2.jpg");
        this.copy(flower1,0,0);
        this.copy(flower2,100,0);
        this.copy(flower1,200,0);
        Picture flowerNoBlue = new Picture(flower2);
        flowerNoBlue.zeroBlue();
        this.copy(flowerNoBlue,300,0);
        this.copy(flower1,400,0);
        this.copy(flower2,500,0);
        this.mirrorVertical();
        this.show();
   }

   /* Main method for testing
    */
   public static void main(String[] args)
   {
      Picture p = new Picture(500,500);
      p.createCollage();
   }
 }
```

The following is experimental, as of 2023-07-05, and needs some organization, plus some credit to CSAwesome and Barb Ericson.

```
Data: beach.jpg
```

```
Data: pictureClasses1.jar

import java.awt.Image;
import java.awt.image.BufferedImage;

/**
 * Interface to describe a digital picture.  A digital pict→
 * associated file name.  It can have a title.  It has pixe→
 * associated with it and you can get and set the pixels.  →
 * can get an Image from a picture or a BufferedImage.  You→
 * it from a file name or image.  You can show a picture.  →
 * explore a picture.  You can create a new image for it.
 *
 * @author Barb Ericson ericson@cc.gatech.edu
 */
public interface DigitalPicture
{
  public String getFileName(); // get the file name that th→
  public String getTitle(); // get the title of the picture
  public void setTitle(String title); // set the title of t→
  public int getWidth(); // get the width of the picture in→
  public int getHeight(); // get the height of the picture →
```

```
import java.awt.*;
import java.awt.font.*;
import java.awt.geom.*;
import java.awt.image.BufferedImage;
import java.text.*;
import java.util.*;
import java.util.List;

/**
 * A class that represents a picture.  This class inherits
     from
 * SimplePicture and allows the student to add
     functionality to
 * the Picture class.
 *
 * @author Barbara Ericson ericson@cc.gatech.edu
 */
public class Picture extends SimplePicture
{
  ///////////////////// constructors
      //////////////////////////////////

  /**
   * Constructor that takes no arguments
   */
  public Picture ()
  {
    /* not needed but use it to show students the implicit
        call to super()
     * child constructors always call a parent constructor
     */
    super();
  }

  /**
   * Constructor that takes a file name and creates the
       picture
   * @param fileName the name of the file to create the
       picture from
   */
  public Picture(String fileName)
  {
    // let the parent class handle this fileName
    super(fileName);
  }

  /**
   * Constructor that takes the height and width
   * @param height the height of the desired picture
   * @param width the width of the desired picture
   */
  public Picture(int height, int width)
  {
    // let the parent class handle this width and height
    super(width,height);
  }

  /**
   * Constructor that takes a picture and creates a
   * copy of that picture
   * @param copyPicture the picture to copy
   */
  public Picture(Picture copyPicture)
  {
    // let the parent class do the copy
    super(copyPicture);
  }

  /**
   * Constructor that takes a buffered image
   * @param image the buffered image to use
   */
  public Picture(BufferedImage image)
  {
    super(image);
  }
  ////////////////////// methods
      ///////////////////////////////////////

  /**
   * Method to return a string with information about this
       picture.
   * @return a string with information about the picture
       such as fileName,
   * height and width.
   */
  public String toString()
  {
    String output = "Picture, filename " + getFileName() +
      " height " + getHeight()
      + " width " + getWidth();
    return output;

  }

  /**
    zeroBlue() method sets the blue values at all pixels to
        zero
 */
  public void zeroBlue()
  {
    Pixel[][] pixels = this.getPixels2D();

    for (Pixel[] rowArray : pixels)
     {
       for (Pixel p: rowArray)
       {
              p.setBlue(0);
       }
    }
  }
```

```
    /*
       keepOnlyBlue() method sets the blue values at all
           pixels to zero.

       Add new method here and call it from main.
    */

     /* Main method for testing
      */
     public static void main(String[] args)
     {
       Picture pict = new Picture("beach.jpg");
       pict.show();
       pict.zeroBlue(); // Change this to call keepOnlyBlue()
       pict.show();
     }
  }
```

5.5 YouTube Videos

Runestone uses a YouTube API that broadcasts events when a reader interacts with a video. In this way, Runestone can record video-watching as an activity. As PreTeXt output we concede better behavior on small screens ("responsiveness") to enable this feature in a Runestone version. A `@label` attribute is necessary for persistence in the Runestone database. Various PreTeXt features, such as playlists, are not yet supported—make a request.

Standalone

The margins here are asymmetric just as a test. And this text is here to see where the video ends.

5.6 Deeper

This is a stub of a `<section>`, but it contains two `<subsection>` which each contain a PROJECT-LIKE item that includes a `<program>` element, and thus is a coding exercise. This tests migration to the Runestone assignment area, though Runestone only has divisions two-deep ("chapter" and "subchapter", so both will show up associated with the same `<section>`—this one.

The first is a start of a C program, so will only be interactive on a Runestone server. The second is a Python program, so will be interactive in all HTML outputs.

5.6.1 Subsection One

An `<activity>` next. This one has a `<program>` so will be made interactive whenever possible.

Activity 5.6.1 Activity in a Subsection. We would suggest you do something here.

```
#include <stdio.h>

int main(void)
```

Nothing suggests this next `<project>` is interactive.

Project 5.6.2 Project in a Subsection. You would work this project on paper, most likely. It is *never* interactive since there is no indication of a desire for that, even if hosted on a capable platform.

A little bit of markup, to test a bug: x^2. (The bug was only apparent under a deprecated method for specify a question to be an interactive short-answer question on Runestone Academy.)

5.6.2 Subsection Two

An `<exploration>` next.

Exploration 5.6.3 Exploration in a Subsection. We would suggest now that you explore something here. In this case we simply provide a CodeLens, which would be graded as an "interaction".

```
for i in range(10):
    print(i)
```

The next simple `<exercise>` will be a "short answer" question on a capable interactive platform since we have included a `<response>` element.

Checkpoint 5.6.1 Inline Exercise in a Subsection. I am an interactive short answer question, but only on a capable platform.

5.7 True/False Exercises

1. **True/False.** Every vector space has finite dimension.

   True or False?

   Hint. P_n, the vector space of polynomials with degree at most n, has dimension $n+1$ by Theorem 3.2.16. [Cross-reference is just a demo, content is not relevant.] What happens if we relax the defintion and remove the parameter n?

   Answer. False.

   Solution. False.

   The vector space of all polynomials with finite degree has a basis, $B = \{1, x, x^2, x^3, \ldots\}$, which is infinte.

5.8 Multiple Choice Exercises

When this `<exercises>` division is hosted on Runestone Academy, it will be enabled for group work. See group selection and submission features at the end of the division. (2023-07-03: experimental.)

1. **Multiple-Choice, Not Randomized, One Answer.** What color is a stop sign?

   A. Green

   B. Red

C. White

Hint 1. What did you see last time you went driving?

Hint 2. Maybe go out for a drive?

Answer. B.

Solution.

A. *Incorrect.*

Green means “go!”.

B. *Correct.*

Red is universally used for prohibited activities or serious warnings.

C. *Incorrect.*

White might be hard to see.

2. Multiple-Choice, Not Randomized, Multiple Answers. Which colors might be found in a rainbow? (Note that the radio buttons now allow multiple buttons to be selected.)

A. Red

B. Yellow

C. Black

D. Green

Hint. Do you know the acronym...ROY G BIV for the colors of a rainbow, and their order?

Answer. A, B, D.

Solution.

A. *Correct.*

Red is a definitely one of the colors.

B. *Correct.*

Yes, yellow is correct.

C. *Incorrect.*

Remember the acronym...ROY G BIV. “B” stands for blue.

D. *Correct.*

Yes, green is one of the colors.

3. Multiple-Choice, Randomized, One Answer. What color is a stop sign? [Static versions retain the order as authored.]

A. Green

B. Red

C. White

Hint 1. What did you see last time you went driving?

Hint 2. Maybe go out for a drive?

Answer. B.

Solution.

A. *Incorrect.*

   Green means “go!”.

B. *Correct.*

   Red is universally used for prohibited activities or serious warnings.

C. *Incorrect.*

   White might be hard to see.

4. **Multiple-Choice, Randomized, Multiple Answers.** Which colors might be found in a rainbow? (Note that the radio buttons now allow multiple buttons to be selected.) [Static versions retain the order as authored.]

A. Red

B. Yellow

C. Black

D. Green

Hint. Do you know the acronym...ROY G BIV for the colors of a rainbow, and their order?

Answer. A, B, D.

Solution.

A. *Correct.*

   Red is a definitely one of the colors.

B. *Correct.*

   Yes, yellow is correct.

C. *Incorrect.*

   Remember the acronym...ROY G BIV. “B” stands for blue.

D. *Correct.*

   Yes, green is one of the colors.

5. **Mathematical Multiple-Choice, Not Randomized, Multiple Answers.** Which of the following is an antiderivative of $2\sin(x)\cos(x)$?

A. $\sin^2(x) + 832$

B. $\sin^2(x)$

C. $-\cos^2(x)$

D. $-2\cos(x)\sin(x)$

Hint. You can take a derivative on any one of the choices to see if it is correct or not, rather than using techniques of integration to find *a single* correct answer.

Answer. A, B, C.

Solution.

A. *Correct.*

   Remember that when we write $+C$ on an antiderivative that this is the way we communicate that there are *many* possible derivatives, but they all "differ by a constant".

B. *Correct.*

   The derivative given in the statement of the problem looks exactly like an application of the chain rule to $\sin^2(x)$.

C. *Correct.*

   Take a derivative on $-\cos^2(x)$ to see that this answer is correct. Extra credit: does this answer "differ by a constant" when subtracted from either of the other two correct answers?

D. *Incorrect.*

   The antiderivative of a product is not the product of the antiderivatives. Use the product rule to take a derivative and see that this answer is not correct.

5.9 Parsons Exercises

1. Parsons Problem, Mathematical Proof. Create a proof of the theorem: If n is an even number, then $n \equiv 0 \mod 2$.

- Click the heels of your ruby slippers together three times.
- Suppose n is even.
- Either:
  Then n is a prime number.
  Or:
  Then there exists an m so that $n = 2m$.
  Or:
  Then there exists an m so that $n = 2m + 1$.
- Thus $n \equiv 0 \mod 2$.
- So $n = 2m + 0$.

  This is a superfluous second paragraph in this block.
- And a little bit of irrelevant multi-line math

$$c^2 a^2 + b^2$$
$$x^2 + y^2.$$

Solution.

- Suppose n is even.
- Then there exists an m so that $n = 2m$.
- So $n = 2m + 0$.

  This is a superfluous second paragraph in this block.
- Thus $n \equiv 0 \mod 2$.

2. Parsons Problem, Programming. The Sieve of Eratosthenes computes prime numbers by starting with a finite list of the integers bigger than 1. The first member of the list is a prime and is saved/recorded. Then all multiples of that prime (which not a prime, excepting the prime itself!) are removed from the list. Now the first number remaining in the list is the next prime number. And the process repeats.

The code blocks below can be rearranged to form one of the many possible programs to implement this algorithm to compute a list of all the primes less than 250. [Ed. this version of this problem requires the reader to provide the necessary indentation.]

This reprises Exercise 2.5.1.

- `for nonprime in range(p, n, p):`
- Either:

```
primes = []
candidates = list(range(2,n))
```

  Or:

```
candidates = []
primes = list(range(2,n))
```

-
  ```
  p = candidates[0]
  primes.append(p)
  ```

-
  ```
  print(primes)
  ```

-
  ```
  if nonprime in candidates:
      candidates.remove(nonprime)
  ```

-
  ```
  n = 250
  ```

-
  ```
  primes = candidates + [p]
  ```

-
  ```
  while candidates:
  ```

Solution.

```
n = 250
primes = []
candidates = list(range(2,n))
while candidates:
    p = candidates[0]
    primes.append(p)
    for nonprime in range(p, n, p):
        if nonprime in candidates:
            candidates.remove(nonprime)
print(primes)
```

3. Parsons Problem, Programming. The Sieve of Eratosthenes computes prime numbers by starting with a finite list of the integers bigger than 1. The first member of the list is a prime and is saved/recorded. Then all multiples of that prime (which not a prime, excepting the prime itself!) are removed from the list. Now the first number remaining in the list is the next prime number. And the process repeats.

The code blocks below can be rearranged to form one of the many possible programs to implement this algorithm to compute a list of all the primes less than 250. [Ed. this version of this problem does not require the reader to provide the necessary indentation, which is the default.]

This reprises Exercise 2.5.1.

-
  ```
  ␣␣␣␣for␣nonprime␣in␣range(p,␣n,␣p):
  ```
- Either:
  ```
  primes = []
  candidates = list(range(2,n))
  ```
  Or:
  ```
  candidates = []
  primes = list(range(2,n))
  ```

-
  ```
  ␣␣␣␣p␣=␣candidates[0]
  ␣␣␣␣primes.append(p)
  ```

-
  ```
  print(primes)
  ```

-
  ```
  ␣␣␣␣␣␣␣␣if␣nonprime␣in␣candidates:
  ␣␣␣␣␣␣␣␣␣␣␣␣candidates.remove(nonprime)
  ```

-
  ```
  n␣=␣250
  ```

- `primes␣=␣candidates␣+␣[p]`
- `while␣candidates:`

Solution.

```
n = 250
primes = []
candidates = list(range(2,n))
while candidates:
    p = candidates[0]
    primes.append(p)
    for nonprime in range(p, n, p):
        if nonprime in candidates:
            candidates.remove(nonprime)
print(primes)
```

4. Parsons Problem, Mathematical Proof, Numbered Blocks. Create a proof of the theorem: If n is an even number, then $n \equiv 0 \mod 2$. [Ed. This version has numbered blocks, online they are on the right end of the block.]

1. Click the heels of your ruby slippers together three times.
2. Suppose n is even.
3. (a) Then n is a prime number.
   (b) Then there exists an m so that $n = 2m$.
   (c) Then there exists an m so that $n = 2m + 1$.
4. Thus $n \equiv 0 \mod 2$.
5. So $n = 2m + 0$.

   This is a superfluous second paragraph in this block.

Answer. 2, 3b, 5, 4

Solution.

2	Suppose n is even.
3b	Then there exists an m so that $n = 2m$.
5	So $n = 2m + 0$. This is a superfluous second paragraph in this block.
4	Thus $n \equiv 0 \mod 2$.

5. Parsons Problem, Programming. The Sieve of Eratosthenes computes prime numbers by starting with a finite list of the integers bigger than 1. The first member of the list is a prime and is saved/recorded. Then all multiples of that prime (which not a prime, excepting the prime itself!) are removed from the list. Now the first number remaining in the list is the next prime number. And the process repeats.

The code blocks below can be rearranged to form one of the many possible programs to implement this algorithm to compute a list of all the primes less than 250. [Ed. This version has numbered blocks, online they are on the left end of the block.]

This reprises Exercise 2.5.1.

1. `for nonprime in range(p, n, p):`
2. (a)
   ```
   primes = []
   candidates = list(range(2,n))
   ```
   (b)
   ```
   candidates = []
   primes = list(range(2,n))
   ```

3.
   ```
   p = candidates[0]
   primes.append(p)
   ```
4. `print(primes)`

5.
   ```
   if nonprime in candidates:
       candidates.remove(nonprime)
   ```
6. `n = 250`
7. `primes = candidates + [p]`
8. `while candidates:`

Answer. 6, 2a, 8, 3, 1, 5, 4

Solution.

```
n = 250
primes = []
candidates = list(range(2,n))
while candidates:
    p = candidates[0]
    primes.append(p)
    for nonprime in range(p, n, p):
        if nonprime in candidates:
            candidates.remove(nonprime)
print(primes)
```

5.10 Horizontal Parsons Exercises

1. **Parsons Problem, SQL statement.** Form the SQL statement by rearranging the four blocks.

```
| * | SELECT | test | FROM |
```

Solution.

```
SELECT * FROM test
```

2. **Parsons Problem, SQL statement, no randomization.** Form the SQL statement by rearranging the four blocks. This version of this problem will *always* present the blocks in the same fixed order (but incorrect, hopefully!), as prescribed by the author in the source.

```
| * | SELECT | test | FROM |
```

Solution.

```
SELECT * FROM test
```

3. **Parsons Problem, SQL statement, automatic feedback.** Form the SQL statement by rearranging the four blocks.

```
| * | SELECT | test | FROM |
```

Solution.

```
SELECT * FROM test
```

4. **Parsons Problem, Natural Language.** Form the sentence often used to show font samples. You can reuse blocks as needed.

```
| jumped | brown | fox | dog | over | the | lazy | quick |
```

Solution.

```
the quick brown fox jumped over the lazy dog
```

5. **Parsons Problem, Natural Language, with Distractors.** Form the sentence often used to show font samples. Again, but now with distractors.

```
| jumped | quick | brown | fox | dog | bar | over | the | foo | lazy |
```

Solution.

```
the quick brown fox jumped over the lazy dog
```

6. **Parsons Problem, SQL statement, reusable.** Form the SQL statement by rearranging the four blocks. Same problem as above, but we allow blocks to be reused (even though the solution does not require that).

```
| * | SELECT | test | FROM |
```

Solution.

```
SELECT * FROM test
```

5.11 Matching Exercises

1. **Matching Problem, Dates.** Match each event in United States history with the year it happened.

Monroe Doctrine	1803
Haymarket Riot	1863
Louisiana Purchase	1886
Battle of Gettysburg	1823

Solution.

Monroe Doctrine	1823
Haymarket Riot	1886
Louisiana Purchase	1803
Battle of Gettysburg	1863

2. **Matching Problem, Derivatives.** Match each function with its derivative.

$x^3 - 6x^2 + 5$	$3x^2 - 12x$
x^{-3}	$2x + 2$
$(x+1)^2$	$-3x^{-4}$

Solution.

$x^3 - 6x^2 + 5$	$3x^2 - 12x$
x^{-3}	$-3x^{-4}$
$(x+1)^2$	$2x + 2$

3. **Matching Problem, Linear Algebra.** Match each subspace with a basis for that subspace. (You may assume that each set is really a basis for at least one of the subspaces.)

$\{\langle x,y,z\rangle \mid -y+z=0\}$	
$\{\langle x,y,z\rangle \mid -3x-5y+z=0\}$	$\{\langle -4,3,3\rangle, \langle 3,-2,-2\rangle\}\{\langle -4,3,3\rangle, \langle 5,-4,-5\rangle\}\{\langle 3,-2,-2\rangle, \langle 5,-4,-5\rangle$
$\{\langle x,y,z\rangle \mid -2x-5y+2z=0\}$	

Hint. For openers, a basis for a subspace must be a *subset* of the subspace.

Solution.

$\{\langle x,y,z\rangle \mid -y+z=0\}$	$\{\langle -4,3,3\rangle, \langle 3,-2,-2\rangle\}$
$\{\langle x,y,z\rangle \mid -3x-5y+z=0\}$	$\{\langle -4,3,3\rangle, \langle 5,-4,-5\rangle\}$
$\{\langle x,y,z\rangle \mid -2x-5y+2z=0\}$	$\{\langle 3,-2,-2\rangle, \langle 5,-4,-5\rangle\}$

5.12 Clickable Area Exercises

1. **Clickable Areas, "Regular" Text.** Identify (by clicking, or by circling) all of the nouns in this quotation by Eleanor Roosevelt.

   "The future belongs to those who believe in the beauty of their dreams."

   Answer. Correct: *future*; *beauty*; *dreams*. Incorrect: ~~those~~; ~~their~~.

   The incorrect words are pronouns.

   Solution. "The *future* belongs to ~~those~~ who believe in the *beauty* of ~~their~~ *dreams*."

2. **Clickable Areas, Code.** Identify (by clicking, or by circling) all of the assignment statements in this Python function.

```
def main():
    x = 4
    for i in range(5):
        y = i
        if y > 2:
            print(y)
```

   Answer. Correct: `x = 4`; `y = i`. Incorrect: `def main():`; `if y > 2:`.

   Remember, the operator `=` is used for assignment.

3. **Clickable Areas, Text in a Table.** A two-dimensional array was created in Python with the list comprehension:

```
[[0 for x in range(3)] for y in range(2)]
```

   Then the values were (mostly) changed from zeros and the final array is shown below.

Identify (by clicking, or by circling) all of the boolean values in the array.

`42`	`True`	`'towel'`
`'true'`	`0`	`False`

This second table has no `<area>`, in order to test CSS for tables.

`42`	`True`	`'towel'`
`'true'`	`0`	`False`

Hint. Python boolean variables begin with capital latters.

Answer. Correct: `True`; `False`. Incorrect: ~~`'towel'`~~; ~~`'true'`~~.

Python boolean variables are `True` and `False`. A value in quotation marks is a string, not a boolean.

Solution.

`42`	`True`	~~`'towel'`~~
~~`'true'`~~	`0`	`False`

This second table has no `<area>`, in order to test CSS for tables.

`42`	`True`	`'towel'`
`'true'`	`0`	`False`

5.13 Select Exercises

1. Mock exercise, just to say this is all testing, 2023-05-19.

2.

3.

5.14 Short Answer Exercises

1. **Short Answer.** This sample book is configured to make some simple questions interactive on a capable platform, by adding a `<response>` element as a signal.

5.15 Polling

(2024-04-24) This section is experimental, and may not be fully functional as you view it. No markup in the source document is final and may change at any time.

We begin with standalone queries, using the `<query>` element. Runestone calls these **polls**. We plan structures/blocks such as `<poll>`, `<survey>`, and `<questionnaire>`, that will be structured collections of `<query>`.

Query. In my town, the stoplights have:

1. Solid yellow lights.
2. Yellow lights that are solid left-turn arrows.
3. Yellow lights that are blinking left-turn arrows.