Contents

1 Fertilizer **3**

1.1 The Nitrogen Cycle 3

1.2 The Haber-Bosch Process 4

1.3 Other nutrients 4

2 Concrete **7**

2.1 Steel reinforced concrete 7

2.2 Recycling concrete 8

3 Metals **9**

3.1 Steel 9

3.2 What metal for what task? 10

4 A deeper look at Python syntax **13**

4.1 Where did we leave off? 13

4.2 Functions 13

4.2.1 Defining and Calling a Function 13

4.2.2 Parameters and Arguments 14

4.2.3 Return Values 14

4.2.4 Scope 15

4.2.5 Summary 15

4.3 Classes and More Basic Object-Oriented Programming 16

4.3.1 Defining a Class 16

4.3.2 Creating Objects 17

4.3.3 The `__init__` Method 17
4.3.4 Accessing Object Attributes 17
4.3.5 Methods 18
4.3.6 Printing Objects 18
4.3.7 Why Use Classes? 19
4.3.8 Summary 19
4.4 Libraries 20
4.4.1 Importing Libraries 20
4.4.2 Summary 21
4.5 Matplotlib 21
4.5.1 Installing and importing 21
4.5.2 A simple line graph 21
4.5.3 Labels, legends, and grids 22
4.5.4 Visualizing Relationships with scatter plots 23
4.5.5 Histograms and Distributions 23
4.5.6 The object-oriented approach (recommended) 24
4.6 Errors 25
4.6.1 Syntax Errors 26
4.6.2 Exceptions 26
4.6.3 Try-Except 27

5 Angles ***29***
5.1 Radians 31

6 Introduction to Triangles ***33***
6.1 Equilateral, Isosceles, and Scalene Triangles 33
6.2 Interior Angles of a Triangle 35

7 Pythagorean Theorem ***39***
7.1 Distance between Points 40
7.2 Distance in 3 Dimensions 42

A Answers to Exercises ***43***

Index 47

Chapter 1

Fertilizer

One of the biggest problems humans face is: how can we get enough food to feed everyone? In 1950, there were 2.5 billion people on the planet, and about 65% were malnourished. In 2019, there were 7.7 billion people on the planet, and only 15% are malnourished. How did crop yields increase so much? There were several factors: better crop varieties, reliable irrigation, increased mechanization, and affordable fertilizers.

When a plant grows, it takes molecules out of the soil and uses them to build proteins. It primarily needs the elements nitrogen (N), phosphorus (P), and potassium (K).

When you buy a bag of fertilizer at the store, it typically has three numbers on the front. For example, you might buy a bag of "24-22-4". This means that 24% of the mass of the bag is nitrogen, 22% is phosphorus, and 4% is potassium.

Potassium comes as potassium carbonate (K_2CO_3), potassium chloride (KCl), potassium sulfate (K_2SO_4), and potassium nitrate (KNO_3). Any blend of these chemicals is known as "potash". Potash is dug up out of mines.

Phosphorus is also mined, but is refined into phosphoric acid (H_3PO_4) before it is put into fertilizer.

Nitrogen is an especially interesting case for 2 reasons:

- Worldwide, farmers apply more nitrogen to their soil than potassium or phosphorous combined.
- 78% of the air we breathe is nitrogen in the form of N_2, but neither plants nor animals can utilize nitrogen in that form.

1.1 The Nitrogen Cycle

Converting the N_2 in the air into a form that a plant can use is known as *nitrogen fixation*. For billions of years, there were only two ways that nitrogen fixation occurred on earth:

- The energy from lightning causes N_2 and H_2O to reconfigure as ammonia (NH_3) and nitrate (NO_3). This accounts for about 10% of all naturally occurring nitrogen fixation.
- Cyanobacteria are responsible for the rest. They convert N_2 into ammonia.

Let's say that you are eating soybeans. There is a cyanobacteria called *rhizobia* that has a symbiotic relationship with soybean plants. Rhizobia fixes nitrogen for the soybean plant. The soybean plant performs photosynthesis, then gives sugars to the rhizobia.

The proteins in the soybeans contain nitrogen from the rhizobia. When you eat them, you use some of the nitrogen to build new proteins. You probably don't use all the nitrogen, so your cells release ammonia into your blood.

Ammonia likes to react with things, so your liver combines the ammonia with carbon dioxide to make urea $(CO(NH_2)_2)$. Your kidneys take the urea out of your blood and mix it with a bunch of water and salts in your bladder. When you urinate, the urea leaves your body.

If you urinate on the ground, the nearby plants can take the nitrogen out of the urea.

When you die, the nitrogen in your proteins will return to the soil as ammonia and nitrate.

For centuries, farms got their nitrogen from urine, feces, and rotting organic material. There were two challenges with this:

- Human pathogens had to be kept away from human food.
- There was simply not enough to support 7.7 billion people.

This meant we had to figure out how to do nitrogen fixation at an industrial level.

1.2 The Haber-Bosch Process

During World War I, two German scientists, Fritz Haber and Carl Bosch, figured out how to make ammonia from N_2 and H_2 using high temperatures and pressures. This is how nearly all nitrogen fertilizer is created today.

Where do we get the H_2? From methane (CH_4) in natural gas. Today, 3-5% of the world's natural gas production is consumed in the Haber-Bosch process.

The ammonia is converted into ammonium nitrate (NH_4NO_3) or urea before it is shipped to farms.

1.3 Other nutrients

Healthy plants require several other elements that are sometimes applied as fertilizer: calcium, magnesium, and sulfur.

Finally, tiny amounts of copper, iron, manganese, molybdenum, zinc, and boron are sometimes needed.

Chapter 2

Concrete

To make concrete, you mix cement with water and an aggregate (sand or rock). The cement is usually only about 10 to 15 percent of the mixture. The cement reacts with the water, and the resulting solid binds the aggregate together. In 2019, the world consumed 4.5 billion tons of cement.

Concrete is hard and durable. The mortar between the pyramids at Giza is concrete — it is now 5000 years old. Today, we use concrete to build many structures including buildings, bridges, airport runways, and dams. Grout and mortar are related materials but different in that they are used as bonding or filler materials within construction.

There are many kinds of cement, but the most common is Portland cement. It is made by heating limestone (calcium carbonate) with clay (for silicon) in a kiln. Two things come out of the kiln: Carbon dioxide and a hard substance called “clinker”. The clinker is ground up with some gypsum before it is sent to market.

The carbon dioxide is released into the atmosphere, making it an exothermic reaction. Cement manufacturing is responsible for about 8% of the world’s CO_2 emissions; it is a major contributor to climate change.

Especially hard concrete, like that used in a nuclear power plant, can support 3,000 kg per centimeter without being crushed. However, if you pull on two ends of a piece of concrete, it comes apart relatively easily. We say that concrete can handle a lot of *compressive stress*, but not much *tensile* stress.

2.1 Steel reinforced concrete

Many places where we use concrete (like in a bridge), we need both compressive and tensile stress. Often, the top of a beam is undergoing compression and the bottom of the beam is undergoing tension.

FIXME Picture here

Steel has tremendous tensile strength, but not as much compressive strength as concrete. To get both tensile *and* compressive strength, we often bury steel bars or cables inside the concrete. This is known as *steel-reinforced concrete*. The concrete generally does a very good job protecting the steel, which keeps it from rusting.

You may have heard of *rebar* before. That is just short for “reinforcing bar”. Typically, rebar has bumps and ridges that keep the bar and the concrete from moving independently.

2.2 Recycling concrete

Many concrete structures only last about 100 years. When they are demolished, the concrete can be reused as aggregate in other projects. Often, the concrete bits are mixed with cement and made into concrete once more.

If the concrete to be reused is reinforced with steel, the steel has to be removed and recycled separately. The concrete is then crushed into small pieces.

Chapter 3

Metals

Elements that transmit electricity well, even at low temperatures, are called *metals*. Many metals are likely familiar to you, such as aluminum, iron, copper, tin, gold, silver, and platinum. Aluminum and iron are particularly common; together they make up about 14% of the earth's crust.

An *alloy* is a mixture of elements that includes at least one metal. Brass, for example, is an alloy of copper and zinc. Bronze is an alloy of copper and tin.

3.1 Steel

One of the most common alloys is steel, which is an alloy of iron and carbon. In pure iron, the molecules slip past each other easily, so pure iron is relatively soft and easily deformed. The carbon in steel prevents that slipping, which is why steel is much, much harder than iron.

How much carbon does steel have? If you have less than 0.002% by weight, you end up with something very much like pure iron. As you increase the carbon, it gets harder and harder. Once it gets above about 2%, the result is very brittle.

If you add about 11% chromium to steel, you get *stainless steel*, which resists rusting.

Exercise 1 Tensile Strength

Working Space

The tensile strength of steel is usually between 400 MPa and 1200 MPa. A Mega Pascal (MPa) is the strength necessary to hold 1,000,000 newtons of force with a cable that has a 1 square meter cross section. Or, equivalently, to hold 1 newton of force with a cable that has a 1 square millimeter cross section.

If you have are buying a round cable that has a tensile strength of 700 Mpa and must hold a 100 kg man aloft, what is the diameter of the smallest cable you can use?

Answer on Page 43

Here are some approximate tensile strengths of other materials:

Material	Tensile strength (MPa)
Iron	3
Concrete	4
Rubber	16
Glass	33
Wood	40
Nylon	100
Human hair	200
Aluminum	300
Steel	700
Spider webs	1000
Carbon fiber	4000

3.2 What metal for what task?

Copper is often used for electrical wires in your house and appliances. This is because it is very efficient at moving electricity (very little power is lost as heat). It is also very good a transmitting heat, so you will often see copper pots and pans.

Aluminum is less dense than copper, and is still a relatively good conductor of electricity. This combination of lighter weight and conductivity is why the overhead wires in a power system are often made of aluminum.

Aluminum is not as strong as steel, but considerably lighter. It is often used structurally where weight is a concern, such as in skyscrapers, cars, airplanes, and ships.

Titanium is about as strong as steel, but it weights about half as much. Titanium is very difficult to work with, so it is used in places where weight and strength are very important and cost is not, such as in airplanes and bicycles. FIXME: We mention airplanes in both examples. Maybe we should clarify the role each one plays?

(Carbon fiber, which is light, strong, and very easy to work with, is replacing aluminum and titanium in many applications. 20 years ago, many expensive bicycles were made of titanium. These days the vast majority are made with carbon fiber.)

Zinc and tin are very resistant to corrosion, so they are often used as a coating to prevent steel from rusting. They are also used in many alloys for the same reason. In the United States, the penny is 97.5% zinc and only 2.5% copper.

Chapter 4

A deeper look at Python syntax

4.1 Where did we leave off?

In the previous Introduction to Python chapter, We started by installing Python and setting up VSCode, then practiced using the console (terminal) to execute `.py` files. Recall that runs code **sequentially**, one line at a time, and that output appears in the console.

We can output to the screen with `print()`, including how Python formats output by default and how to customize it. Next, we introduced variables and common datatypes such as strings, integers, floats, booleans, lists, and dictionaries.

Mathematical operations, including arithmetic operators were demonstrated. We played with in `str()` and the method to get input from the console, `input()`. Finally, we worked with conditionals and loops to control program flow, then introduced strings and lists as sequence types, and worked with them. With these tools, you should now be able to write basic Python programs that read input, store data, make decisions, repeat actions, and produce useful output.

Let's now build on this foundation with some more advanced concepts, such as functions, classes, and libraries. We will also cover error handling, and experiment with basic graphing ability using the Matplotlib library.

4.2 Functions

As our programs get larger, we begin to repeat the same sets of steps again and again. Rather than copying and pasting code (which is hard to maintain and easy to break), we can group instructions into *functions*. A *function* is a named block of code that performs a task. Once a function is defined, it can be *called* (used) as many times as needed.

You have already used functions before, even if you did not know it! For example, `print()`, `input()`, `len()`, and `type()` are built-in Python functions.

4.2.1 Defining and Calling a Function

To define a function in Python, we use the `def` keyword, followed by:

- a function name
- parentheses `()`
- a colon `:`
- an indented block of code

```
def say_hello():
    print("Hello!")
    print("Welcome to Python.")
```

This code *defines* the function, but it does not run the code inside it yet. To execute the function, we must *call* it by using its name followed by parentheses:

```
say_hello()
```

When Python reaches a function call, it temporarily jumps into the function, runs the indented code, and then returns to the line after the call. Just like conditionals and loops, **indentation matters**.

4.2.2 Parameters and Arguments

Most functions are more useful when they can work with different values. A *parameter* is a variable name listed inside the parentheses of a function definition. An *argument* is the actual value you pass into the function when you call it.

```
def greet(name):
    print("Hello", name)
```

Here, `name` is a parameter. We can call the function with different arguments:

```
greet("Alex")
greet("Chicago")
```

4.2.3 Return Values

Some functions *return* a value. Returning a value allows the function to produce a result that can be stored in a variable or used in an expression.

```
def add(a, b):
    return a + b
```

Now we can use the returned value:

```
x = add(3, 4)
print(x)
```

Output:

```
7
```

We can alter the function to expect a certain parameter type, such as integers, and return a specific type as well:

```
def add(a: int, b: int) -> int:
    return a + b
```

It is important to note that Python won't raise a TypeError if you pass in the wrong type, but this is a helpful way to document your code for other programmers, which you may often work with, and yourself.

We can force the parameters to be a certain type, and raise an error if the wrong type is passed in:

```
def add(a: int, b: int) -> int:
    if not isinstance(a, int) or not isinstance(b, int):
        raise TypeError("Both arguments must be integers")
    return a + b
```

Important: `print()` displays information to the console, but it does not return a useful value. `return` sends a value back to where the function was called. Functions that do not have a `return` statement return `None` by default, and are often called *void* functions.

4.2.4 Scope

A variable created inside a function only exists inside that function. This idea is called *scope*. If you define a variable inside a function, you cannot use it outside of the function unless you return it.

```
def make_number():
    x = 10
    return x

y = make_number()
print(y)
```

The variable `x` exists only inside `make_number()`, but the value it returns is stored in `y` outside the function.

4.2.5 Summary

- Functions group code into reusable blocks
- Functions are defined with `def` and called using parentheses
- Parameters receive input values (arguments) when a function is called
- `return` sends a value back to the caller
- Variables inside a function have local scope

Exercise 2 Tracing a Function Call

Consider the following code:

```
def double(x):
    return x * 2

a = 3
b = double(a)
print(b)
```

What is printed? What is the value of `a` after the program runs?

Working Space

Answer on Page 43

Exercise 3 Write a Function

Write a function called `is_even(n)` that returns `True` if `n` is even and `False` otherwise. Then show an example call to your function using `n = 7`.

Working Space

Answer on Page 43

4.3 Classes and More Basic Object-Oriented Programming

So far, we have worked mostly with individual variables, lists, and functions. As programs grow larger, it becomes useful to group related data and behavior together. *Object-Oriented Programming* (OOP) is a programming style that organizes code around *objects* rather than individual functions.

In Python, objects are created from *classes*. A *class* is a blueprint for creating objects that contain both data (variables) and behavior (functions).

4.3.1 Defining a Class

A class is defined using the `class` keyword, followed by:

- the class name (by convention, written in `CamelCase`)
- a colon `:`
- an indented block of code

```
class Person:
    pass
```

This defines an empty class. It does not do anything yet, but it gives Python a new type called `Person`.

4.3.2 Creating Objects

An *object* is an instance of a class. Objects are created by calling the class name like a function.

```
p1 = Person()
p2 = Person()
```

Here, `p1` and `p2` are two separate objects, both created from the same class.

4.3.3 The `__init__` Method

Most classes need to store data. This is done using a special function called `__init__`. This function runs automatically when a new object is created.

```
class Person:
    def __init__(self, name, age):
        self.name = name
        self.age = age
```

The parameter `self` refers to the current object being created. Each object gets its own copy of the variables defined using `self`.

```
p = Person("Alex", 20)
```

Now the object `p` has two attributes:

- `p.name`
- `p.age`

4.3.4 Accessing Object Attributes

Attributes are accessed using dot notation.

```
print(p.name)
print(p.age)
```

Output:

```
Alex
20
```

Each object stores its own values. Creating another object does not overwrite existing ones.

4.3.5 Methods

Functions defined inside a class are called *methods*. Methods describe behavior that belongs to the object.

```python
class Person:
    def __init__(self, name, age):
        self.name = name
        self.age = age

    def greet(self):
        print("Hello, my name is", self.name)
```

Calling a method uses dot notation:

```python
p = Person("Alex", 20)
p.greet()
```

Output:

```
Hello, my name is Alex
```

4.3.6 Printing Objects

By default, printing an object produces an unreadable result:

```python
print(p)
```

Output (example):

```
<__main__.Person object at 0x7f9c1a3d>
```

This actually is the object's memory address in the computer's memory, but it may be a bit tedious want to print that and manually search ever byte. Instead, to control how an object is printed, we can define the special method `__str__`.

```python
class Person:
    def __init__(self, name, age):
        self.name = name
        self.age = age

    def __str__(self):
        return f"{self.name} ({self.age} years old)"
```

Now printing the object gives a meaningful result:

```python
p = Person("Alex", 20)
print(p)
```

Output:

```
Alex (20 years old)
```

4.3.7 Why Use Classes?

Classes allow us to:

- group related data and behavior together
- create many objects with the same structure
- write code that is easier to understand and maintain

4.3.8 Summary

- A class is a blueprint for creating objects
- Objects store data using attributes
- `__init__` initializes new objects

- Methods define behavior for objects
- `__str__` controls how objects are printed
- This was only a foundation of OOP; more advance concepts exist and are stronger in other programming languages such as Java and C++

Exercise 4 Tracing an Object

Consider the following code:

```
class Counter:
    def __init__(self, value):
        self.value = value

    def increment(self):
        self.value += 1

c = Counter(5)
c.increment()
c.increment()
print(c.value)
```

What is printed?

Working Space

Answer on Page 43

Exercise 5 Custom Printing

Write a class called `Book` that stores a title and an author. Add a `__str__` method so that printing a book displays:

```
Title by Author
```

Working Space

Answer on Page 44

4.4 Libraries

A *library* is a collection of pre-written code that provides additional functionality without requiring you to write everything from scratch. Python includes many built-in libraries that can be used to perform common tasks such as mathematical calculations, data handling, and visualization. You'll often find that code you are seeking can be found through an open-source library which already exists.

To use a library, it must first be imported.

4.4.1 Importing Libraries

The simplest way to import a library is using the `import` keyword.

```
import math
print(math.sqrt(16))
```

In this example, the `math` library provides access to mathematical functions such as square roots.

It is also possible to import specific values or functions from a library.

```
from math import pi
print(pi)
```

This allows direct access to `pi` without referencing the library name.

4.4.2 Summary

- Libraries extend Python's functionality
- The `import` keyword is used to load libraries
- Specific components can be imported directly when needed

4.5 Matplotlib

Matplotlib is the most widely used plotting library in Python. It can produce simple charts quickly (line plots, scatter plots, bar charts, histograms) and also supports publication-quality figures with precise control over labels, legends, and layout. We will use it widely throughout this course to visualize data, create simulations for proving formulas, and experiment with datasets.

4.5.1 Installing and importing

If you are working locally, you can install Matplotlib with:

```
pip install matplotlib
```

In most scripts, you will import the plotting interface `pyplot`:

```python
import matplotlib.pyplot as plt
```

4.5.2 A simple line graph

A line plot is ideal for showing how a value changes over time or across an ordered variable. Let's create `line.py`

```python
import matplotlib.pyplot as plt # the matplotlib library, which we will call
↪  using plt

# two lists containing values
x = [0, 1, 2, 3, 4, 5]
y = [0, 1, 4, 9, 16, 25]

# a 2d plot, used for line graphs
plt.plot(x, y)
# add a title to our plot
plt.title("A Simple Line Plot")
# add axis labels
plt.xlabel("x")
plt.ylabel("y = x^2")
plt.show() # pops up a new window
```

Running `python3 line.py` will open a new *interactive window* in your computer, with the following image:

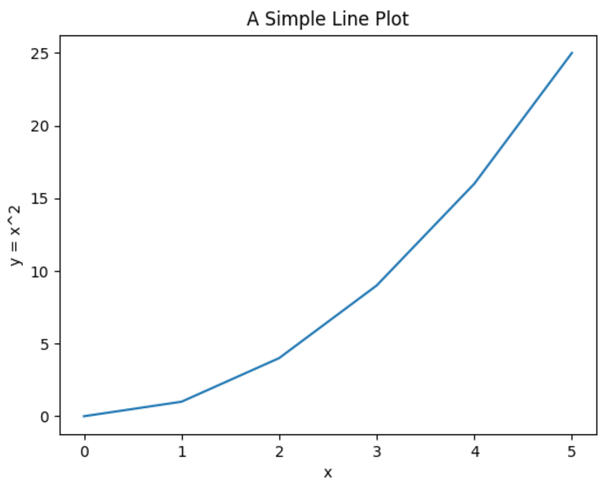


Figure 4.1: The output of line.py.

4.5.3 Labels, legends, and grids

A plot is more useful when it clearly communicates what each element represents. Let's add on to `line.py` by creating `lineTwoOutputs.py` and adding labels, a legend, and another set of data.

```
import matplotlib.pyplot as plt

x = [0, 1, 2, 3, 4, 5]
y1 = [0, 1, 4, 9, 16, 25]
y2 = [0, 1, 8, 27, 64, 125]

plt.plot(x, y1, label="x^2")
plt.plot(x, y2, label="x^3")

plt.title("Two Functions on One
 ↪   Axes")
plt.xlabel("x")
plt.ylabel("value")
plt.grid(True)
plt.legend()
plt.show()
```

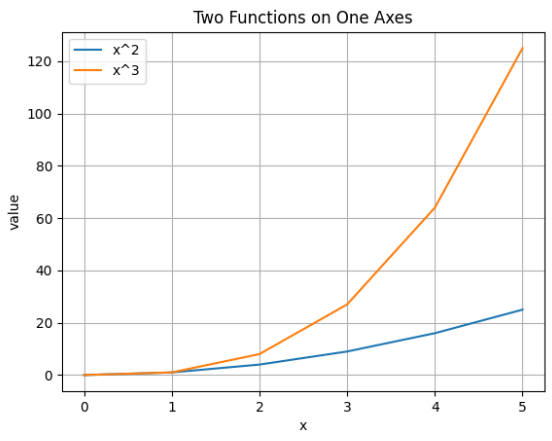


Figure 4.2: $y = x^2$ and $y = x^3$ on the same graphed.

4.5.4 Visualizing Relationships with scatter plots

Scatter plots are excellent for showing how two variables relate (for example, height and weight).

This very basic scatter plot shows the relationship between studying time and score.

```
import matplotlib.pyplot as plt

hours = [1, 2, 3, 4, 5, 6]
scores = [55, 60, 66, 72, 78,
↪  85]

plt.scatter(hours, scores)
plt.title("Study Time vs.
↪  Score")
plt.xlabel("hours studied")
plt.ylabel("score")
plt.show()
```

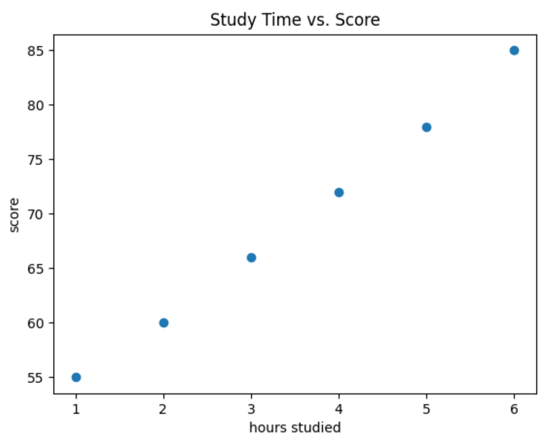


Figure 4.3: A scatterplot generated by matplotlib from the given data.

4.5.5 Histograms and Distributions

A histogram shows how values are distributed and how common different ranges are.

```
import matplotlib.pyplot as plt
# a set of data, for example maybe
↪  these are all test scores
data = [4, 5, 5, 6, 7, 7, 7, 8, 8,
↪  9, 10, 10, 10, 11, 12]

# creates a histogram plot
plt.hist(data, bins=5)
plt.title("Histogram of Values")
plt.xlabel("value")
plt.ylabel("frequency")
plt.show()
```

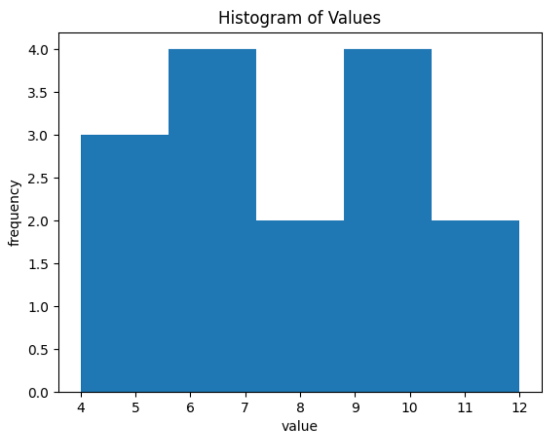


Figure 4.4: A histogram of a set of data.

4.5.6 The object-oriented approach (recommended)

Matplotlib has two main styles:

- **State-based** (using `plt.plot`, `plt.title`, ...): quick and convenient.
- **Object-oriented** (creating a `Figure` and `Axes`): more explicit and easier to scale.

For multi-plot layouts or different analysis of datasets, prefer the object-oriented approach:

```
import matplotlib.pyplot as plt

x = [0, 1, 2, 3, 4, 5]
y1 = [0, 1, 4, 9, 16, 25]
y2 = [0, 1, 8, 27, 64, 125]

# two figures, side by side. axes becomes an indexable list
fig, axes = plt.subplots(nrows=1, ncols=2)

axes[0].plot(x, y1)
axes[0].set_title("x^2")
axes[0].set_xlabel("x")
axes[0].set_ylabel("value")

axes[1].plot(x, y2)
axes[1].set_title("x^3")
axes[1].set_xlabel("x")
axes[1].set_ylabel("value")
```

```
fig.suptitle("Two Subplots")
fig.tight_layout()
plt.show()
```

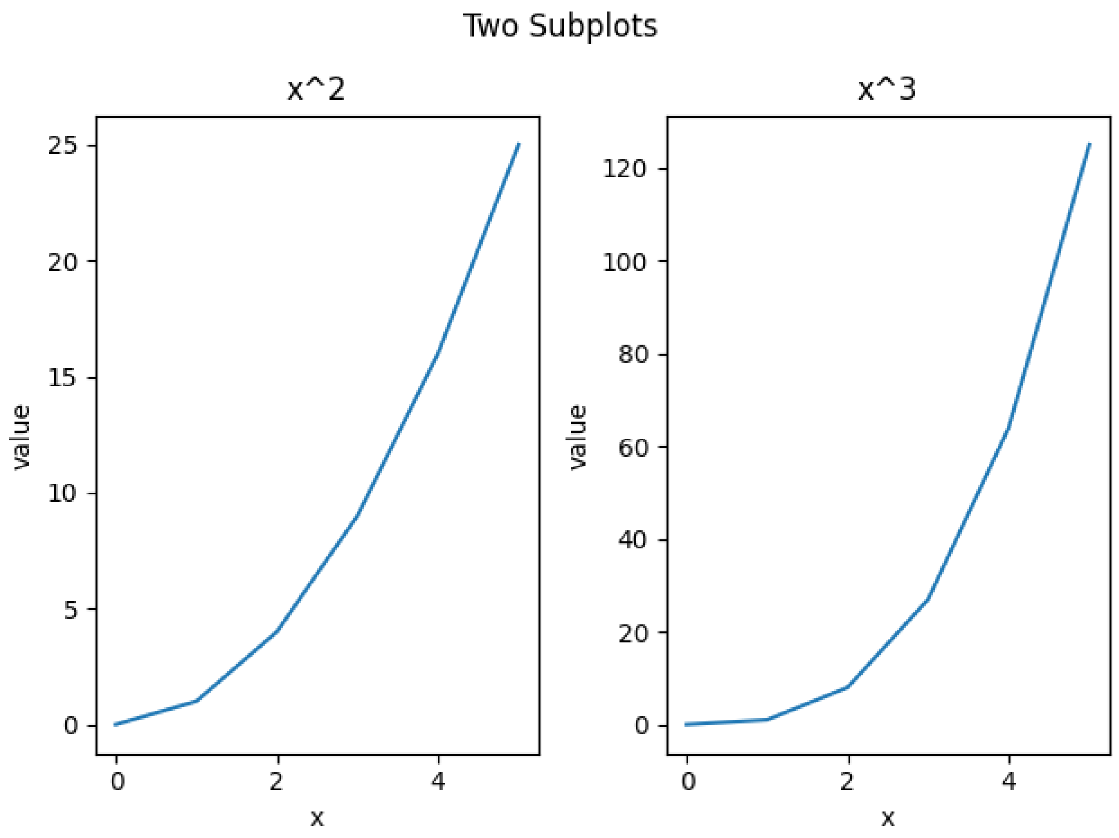


Figure 4.5: The two plots, generated side by side.

4.6 Errors

Even the most experienced programmers can make mistakes. If the code seems to run into issues, your code may crash or output an *error*. Python reports errors by raising a halt in the console output, which include a message explaining the problem and the exact line where it occurred. By learning how to read and interpret these messages, you'll be able to *debug*, or fix issues, in your programs more efficiently and write more reliable code.

You will encounter two main types of errors in your code: **Syntax Errors** and **Runtime Errors/Exceptions**.

- Syntax errors occur when your the formatting, or *syntax*. These errors can usually be found before your code is compiled.

- Runtime Errors, or *Exceptions*, occur when operations in your code cause an invalid calculation, or attempt to do an invalid action. These can range anywhere from basic misnamed variables to imported libraries crashing from incorrect data.

4.6.1 Syntax Errors

What do you notice is immediately wrong with this code?

```
    x = "Welcome Home"
    print(type(x)
```

If we run it, we get the output:

```
Traceback (most recent call last):
  File "<string>", line 1, in <module>
  File "/usr/lib/python3.12/py_compile.py", line 150, in compile
    raise py_exc
py_compile.PyCompileError:   File "./prog.py", line 2
    print(type(x)
         ^
SyntaxError: '(' was never closed
```

We notice that every opening parentheses, bracket, or brace must have a closing supplement. Without out, we run into Syntax Errors, letting us know our format is off.

Another type of Syntax Error can be found in incorrect indentation. Take for example the following code:

```
if True:
print("Hello!")
```

Output:

```
Traceback (most recent call last):
  File "<string>", line 1, in <module>
  File "/usr/lib/python3.12/py_compile.py", line 150, in compile
    raise py_exc
py_compile.PyCompileError: Sorry: IndentationError: expected an indented block
↪   after 'if' statement on line 1 (prog.py, line 2)
```

Here, there is no indentation (usually obtained by pressing the `Tab` key on your keyboard) for lines under the conditional statement. This causes a Syntax Error to be raised.

4.6.2 Exceptions

Let's say I try to run the following code:

```python
    print(x)
    x = "hello"
```

You may see the following output:

```
Traceback (most recent call last):
    File "./program.py", line 1, in <module>
NameError: name 'x' is not defined
```

You have encounted a *NameError*, because x was not assigned before it was attempted to be printed. This brings up an important note on Python code: **code is executed line-by-line, sequentially**. So even if you define x after you try and print it, Python will not understand what you are trying to print. Let's look at another example:

4.6.3 Try-Except

```python
try:
    x = int(input("Enter a number: "))
    print(10 / x)
except ValueError:
    print("Please enter a valid integer.")
except ZeroDivisionError:
    print("Cannot divide by zero.")
```

Here, we attempt to divide 10 by some input number x. There are two Exceptions that could cause this to fail:

- The user inputs a string a characters, or anything that isn't an integer
- The user inputs 0, an invalid divisor.

We use a Try-Except block to check for different errors generated, similar to an if statement. The try block surrounds a block of code that may cause faulty runtime output or errors.

Exercise 6 Fix the Code

```
items = ["pen", "book", "eraser",
↪  "ruler"]

index = input("Enter an index (0 to
↪  3): ")
print("You chose:", items[index])

items.remove("pencil")
print("Updated list:", items)
```

Name two errors that could occur when this code is run. For each error, explain why it occurs and how to fix it.

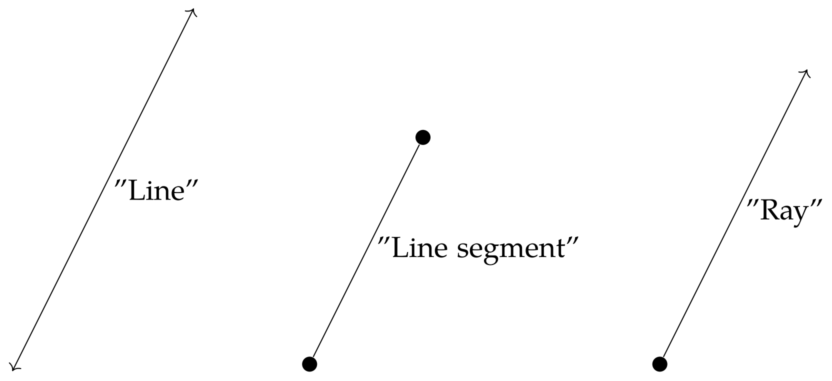
Working Space

Answer on Page 44

CHAPTER 5

Angles

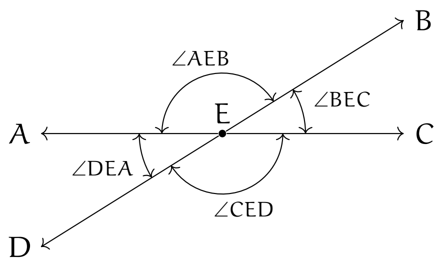
In the following recommend videos, the narrator talks about lines, line segments, and rays. When mathematicians talk about *lines*, they mean a straight line that goes forever in two directions. If you pick any two points on that line, the space between them is a *line segment*. If you take any line, pick a point on it, and discard all the points on one side of the point, that is a *ray*. All three have no width.



Watch the following videos from Khan Academy:

- Introduction to angles: `https://youtu.be/H-de6Tkxej8`
- Measuring angles in degrees: `https://youtu.be/92aLiyeQj0w`

When two lines cross, they form four angles:



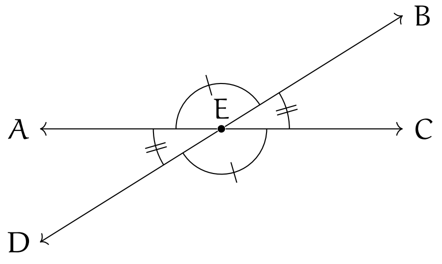
What do we know about those angles? (Note that m∠ means "the measure of angle")

- The sum of any two adjacent angles adds to be 180° So, for example, $m\angle AEB +$

$m\angle BEC = 180°$. We use the phrase "adds to be 180°" so often that we have a special word for it: *supplementary*.

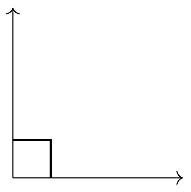
- The sum of all four angles is 360°.
- Angles opposite each other are equal. So, for example, $m\angle AEB = m\angle CED$.
- On any *horizontal* line segment with angles in between it, the sum of the angles must add up to 180°. For example, $\overline{AC}$ is a horizontal line, so angles between it must sum to 180°.

In a diagram, to indicate that two angles are equal we often put hash marks in the angle:



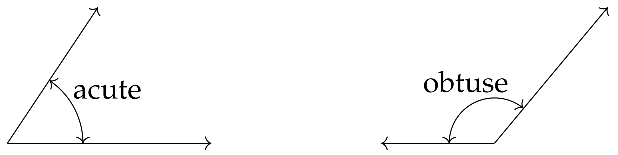
Here, the two angles with a single hash mark are equal, and the two angles with double hash marks are equal.

When two lines are perpendicular, the angle between them is 90°, and we say they meet at a *right angle*. When drawing diagrams, we indicate right angles with an elbow:

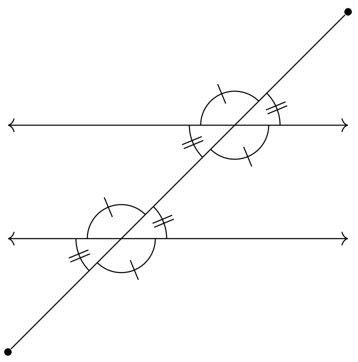


When an angle is less than 90°, it is said to be *acute*. When an angle is more than 90°, it is said to be *obtuse*.

Just as two angles which add up to 180° are referred to as supplementary angles, two (or more) angles which add up to 90° are *complementary* angles. For example, 2 45° angles are complementary, as well as 30° and 60°. If an angle is 65° degrees, we say that its *complement* is $90° - 65° = 25°$.



If two lines are parallel (never intersecting lines with points all the same distance apart), line segments that intersect both lines form the same angles with each line:



5.1 Radians

As you've seen above, angles can be measured in degrees. Just like you can measure length in more than one unit (inches, meters, etc.), there is more than one unit to measure angles in. Angles can also be measured in *radians*. Radians are unitless (that is, you don't have to put a letter after the number) and there are π radians across a straight line. On diagrams and equations, unknown angles in radians are usually represented with the greek letter θ. This means 180° is the same as π radians. Radians come from comparing the length of an arc to the radius of a circle–which we will explore in a future chapter.

Generally, you can use the following formula for your conversions:

$$\text{angle in degrees} = \text{angle in radians} \cdot \frac{180^\circ}{\pi}$$

Example: An angle is measured to be $\frac{\pi}{2}$ radians. What is the angle in degrees?

Solution: Since we know that π radians is the same as 180°, we can set up the unit conversion:

$$\frac{\pi}{2} \cdot \frac{180^\circ}{\pi} = 90^\circ$$

Therefore, a $\frac{\pi}{2}$ angle is 90°.

Exercise 7

Convert the following angles from degrees to radians, or from radians to degrees.

1. $360°$
2. $\frac{\pi}{3}$
3. $225°$
4. $\frac{3\pi}{4}$
5. $30°$
6. $45°$

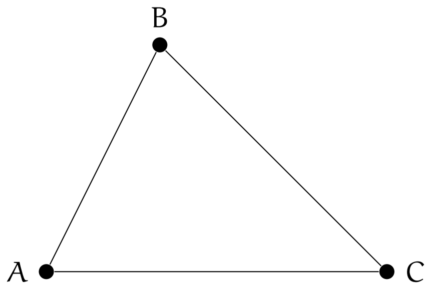
Working Space

Answer on Page 44

CHAPTER 6

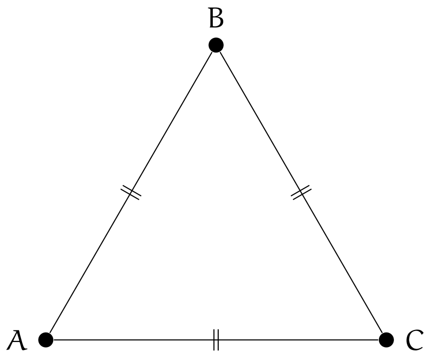
Introduction to Triangles

Connecting any three points with three line segments will get you a triangle. Here is the triangle ABC, which was created by connecting three points A, B, and C:

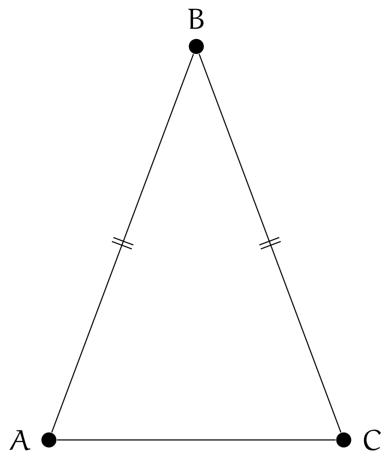


6.1 Equilateral, Isosceles, and Scalene Triangles

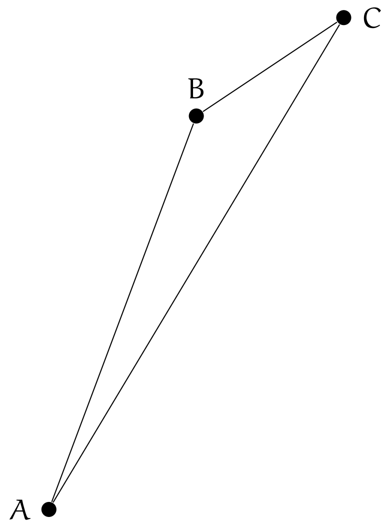
We talk a great deal about the length of the sides of triangles. If all three sides of the triangle are the same length, we say it is an *equilateral triangle*:



If only two sides of the triangle are the same length, we say it is an *isosceles triangle*:

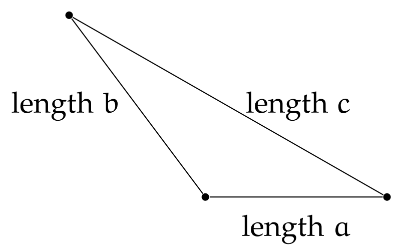


When all three sides of a triangle are different, the triangle is referred to as *scalene*. This consequently means all angle measures are different as well.



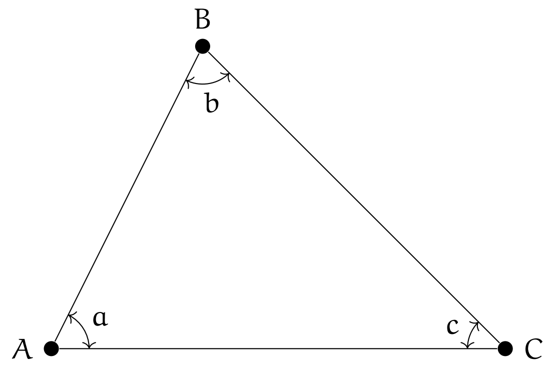
The shortest distance between two points is always the straight line between them. This means you can be certain that the length of one side will *always* be less than the sum of the lengths of the remaining two sides. This is known as the *triangle inequality*.

For example, in this diagram, c must be less than $a + b$.

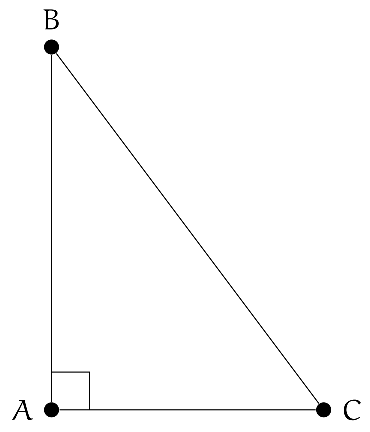


6.2 Interior Angles of a Triangle

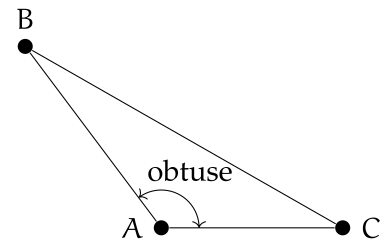
We also talk a lot about the interior angles of a triangle:



A triangle where one of the interior angles is a right angle is said to be a *right triangle*: The side opposite the right angle is the longest side, referred to as the hypotenuse.



If a triangle has an obtuse interior angle, it is said to be an *obtuse triangle*:



If all three interior angles of a triangle are less than 90°, it is said to be an *acute triangle*.

The measures of the interior angles of a triangle always add up to 180°. For example, if we know that a triangle has interior angles of 37° and 56°, we know that the third interior angle is 87°.

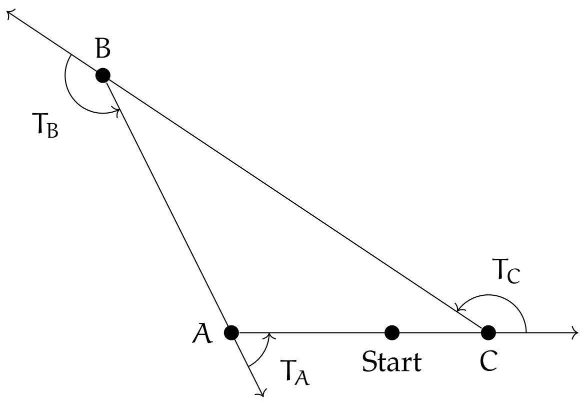
Exercise 8 Missing Angle

One interior angle of a triangle is 92°. The second angle is 42°. What is the measure of the third interior angle?

Working Space

Answer on Page 44

How can you know that the sum of the interior angles is 180°? Imagine that you started on the edge of a triangle and walked all the way around to where you started. (going counter-clockwise.) You would turn three times to the left:



After these three turns, you would be facing the same direction that you started in. Thus, $T_A + T_B + T_C = 360°$. The measures of the interior angles are a, b, and c. Notice that a and T_A are supplementary. So we know that:

- $T_A = 180 - a$
- $T_B = 180 - b$
- $T_C = 180 - c$

So we can rewrite the equation above as

$$(180 - a) + (180 - b) + (180 - c) = 360°$$

Which simplifies to:

$$540° - (a + b + c) = 360°$$

Subtracting both sides from 540:

$$a + b + c = 180°$$

Exercise 9 Interior Angles of a Quadrilateral

Any four-sided polygon is a *quadrilateral*. Using the same "walk around the edge" logic, what is the sum of the interior angles of any quadrilateral?

Working Space

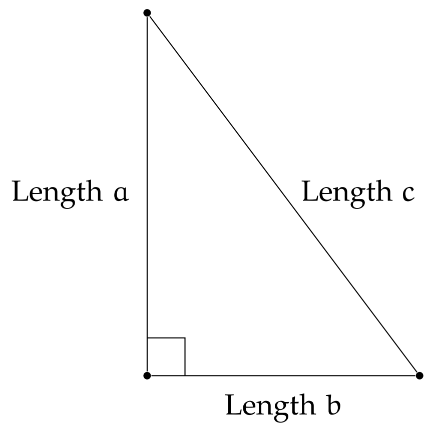
Answer on Page 44

Chapter 7

Pythagorean Theorem

Watch's Khan Academy's Intro to the Pythagorean Theorem video at https://youtu.be/AA6RfgP-AHU.

If you have a right triangle, the edges that touch the right angle are called *the legs*. The third edge, which is always the longest and opposite the right angle, is known as *the hypotenuse*. The Pythagorean Theorem gives us the relationship between the length of the legs and the length of the hypotenuse.



The Pythagorean Theorem tells us that $a^2 + b^2 = c^2$, given that c is the hypotenuse.

For example, if one leg has a length of 3 and the other has a length of 4, then $a^2 + b^2 = 3^2 + 4^2 = 25$. Thus, c^2 must equal 25. This means you know the hypotenuse must be of length 5. This works for any right triangle

In reality, it rarely works out to be such a tidy number. For example, what is the length of the hypotenuse if the two legs are 3 and 6? $a^2 + b^2 = 3^2 + 6^2 = 45$. The length of the hypotenuse is the square root of that: $\sqrt{45} = \sqrt{9 \times 5} = 3\sqrt{5}$, which is approximately 6.708203932499369.

Common side lengths for these triangles are referred to as *Pythagorean triples*, meaning they evaluate to a whole number. Some common examples are $(3, 4, 5)$, $(5, 12, 13)$, and $(8, 15, 17)$. Multiples of right triangles are also triangles ie. $(3, 4, 5) \implies (6, 8, 10)$, which we will touch on next chapter.

There are also angle-based right triangles, consisting of ratios of the angles of the triangles. The most common ones are 45°-45°-90° and the 30°-60°-90°. We will discuss these further in depth, but know for now that they are vital in trigonometry, and consist of Pythagorean triples as side lengths.

Exercise 10 Find the Missing Length

What is the missing measure? All missing values should be whole numbers, except d is an irrational number; answer should be a decimal approximation.

Leg 1 = 6, Leg 2 = 8, Hypotenuse = a

Leg 1 = 5, Leg 2 = b, Hypotenuse = 13

Leg 1 = c, Leg 2 = 15, Hypotenuse = 17

Leg 1 = 3, Leg 2 = 3, Hypotenuse = d

Working Space

Answer on Page 45

A square's diagonal is a special case of the Pythagorean Theorem such that $c = \sqrt{a^2 + b^2} = \sqrt{2s^2}$.

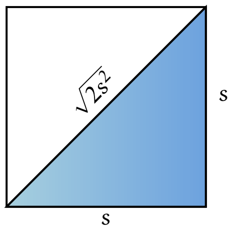
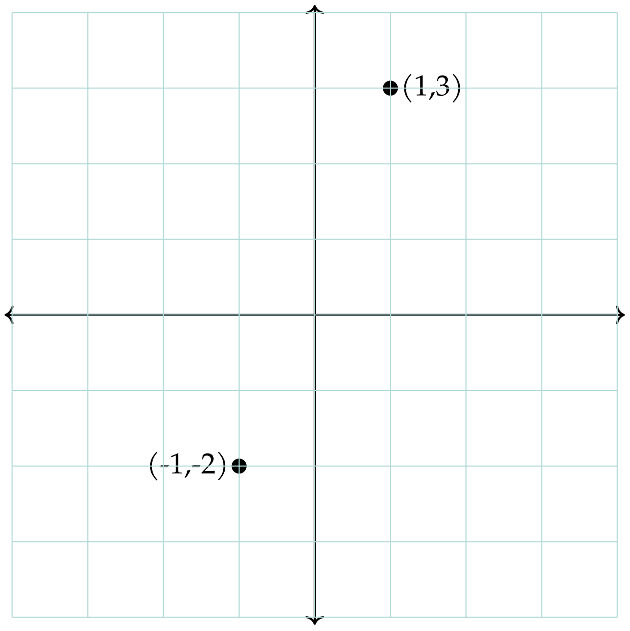


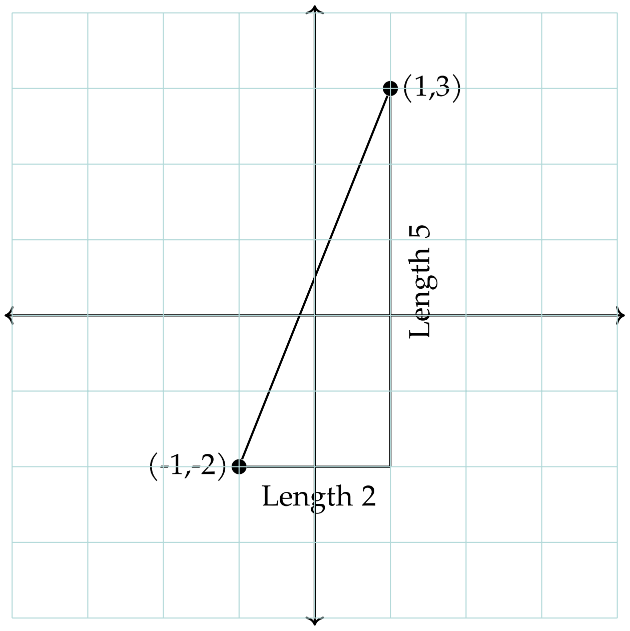
Figure 7.1: A special case of the Pythagorean Theorem where each side is side length s and the hypotenuse is $\sqrt{2s^2}$.

7.1 Distance between Points

What is the distance between these two points?



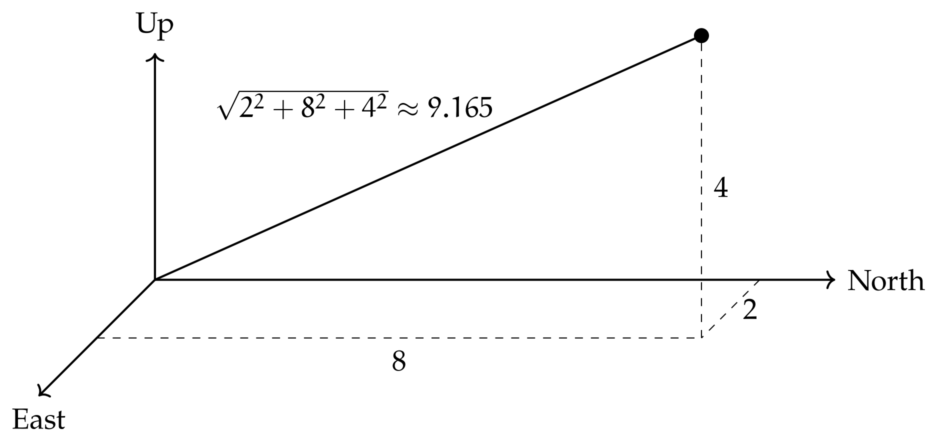
We can draw a right triangle and use the Pythagorean Theorem:



The distance between the two points is $\sqrt{2^2 + 5^2} = \sqrt{29} \approx 5.385165$. In other words, you square the change in x and add it to the square of the change in y. The distance is the square root of that sum.

7.2 Distance in 3 Dimensions

What if the point is in three-dimensional space? For example, you move 2 meters East, 8 meters North, and 4 meters up in the air. How far are you from where you started? You just square each, sum them, and take the square root: $\sqrt{2^2+8^2+4^2} = \sqrt{84} = 2\sqrt{21} \approx 9.165$ meters.



This leads us to a formal definition of the distance formula:

$$d = \sqrt{(x_2-x_1)^2+(y_2-y_1)^2}$$

Or in 3D space:

$$d = \sqrt{(x_2-x_1)^2+(y_2-y_1)^2+(z_2-z_1)^2}$$

Appendix A

Answers to Exercises

Answer to Exercise 1 (on page 10)

On earth, holding a 100 kg man aloft requires 980 Newtons of force.

$980/700 = 1.4$, so you need a cable with a cross-section area of 1.4 square millimeters.

$$\pi r^2 = 1.4$$

$r = \sqrt{1.4/\pi} \approx .67$ millimeters. This means the cable would have to have a diameter of at least 1.34 millimeters.

Answer to Exercise 2 (on page 16)

The function returns `3 * 2`, which is `6`, so the program prints:

```
6
```

The value of `a` is still `3` because the function does not change `a`; it only uses its value to compute a result.

Answer to Exercise 3 (on page 16)

```
def is_even(n):
    return n % 2 == 0

print(is_even(7))
```

This prints `False` because 7 is not divisible by 2.

Answer to Exercise 4 (on page 19)

The initial value is 5. The `increment()` method is called twice, so the value becomes 7. The program prints:

```
7
```

Answer to Exercise 5 (on page 20)

```python
class Book:
    def __init__(self, title, author):
        self.title = title
        self.author = author

    def __str__(self):
        return f"{self.title} by {self.author}"
```

Answer to Exercise 6 (on page 27)

1. **TypeError**: The `input()` function returns a string, so when the user inputs an index, it is treated as a string. Attempting to use this string as an index for the list will raise a `TypeError`. To fix this, we can cast the input to an integer using `int()` and handle the potential `ValueError` if the input is not a valid integer.
2. Pencil is not in the list, so attempting to remove it will raise a `ValueError`. To fix this, we can use a try-except block to catch the `ValueError` and print a message indicating that the item was not found in the list.

Answer to Exercise 7 (on page 32)

1. 2π
2. 60°
3. $\frac{5\pi}{4}$
4. 135°
5. $\frac{\pi}{6}$

6. $\frac{\pi}{4}$

Answer to Exercise 8 (on page 36)

$180° - (92° + 42°) = 46°$

Answer to Exercise 9 (on page 37)

$360°$

Answer to Exercise 10 (on page 40)

a: 10 because $6^2 + 8^2 = 10^2$

b: 12 because $5^2 + 12^2 = 13^2$

c: 8 because $8^2 + 15^2 = 17^2$

d: $3\sqrt{2} \approx 4.24$ because $3^2 + 3^2 = \left(3\sqrt{2}\right)^2$

Index

acute triangle, 38
attributes, 18

cement, 7
classes, 17
  defining, 17
  definition, 17
complementary angles, 32
concrete, 7
creating charts in python, *see* matplotlib

distance
  in 3 dimensions, 44
distance using Pythagorean theorem, 42

equilateral triangle, 35
Errors, 27

fertilizer, 3
functions, 13
  argument, 14
  def, 13
  definition, 13
  parameter, 14
  parameters, 14
  return, 14

Haber-Bosch process, 4

init, 18
isoscelese triangle, 35

libraries, 22
line segment, 31
lines, 31

matplotlib, 23
methods, 19
  definition, 19

nitrogen, 3
nitrogen cycle, 4
nitrogen fixation, 4

object-oriented programming, 17
objects, 18
  definition, 18
obtuse triange, 37

phosphorus, 3
potash, 3
potassium, 3
printing objects, 19
Pythagorean theorem, 41
Pythagorean triples, 41

radians, 33
ray, 31
rebar, 8
right triangle, 37

Scalene Triangle, 36
scope, 15
steel reinforced concrete, 7
supplementary angles, 32

syntax errors, 28

triangle, 35
triangle inequality, 36
try-except, 29

urea, 4
urine, 4