Contents

CHAPTER 1

Basic Statistics

You live near a freeway, and someone asks you, "How fast do cars on that freeway drive?"

You say, "Pretty fast."

They reply, "Can you be more specific?"

So, you pull out your radar gun you happen to always keep on you, and tell them, "That one is going 32.131 meters per second."

To which they say, "I don't want to know about that specific car. I want to know about all the cars."

So, you spend the day beside the freeway measuring the speed of every car that goes by. And you get a list of a thousand numbers, including:

30.462 m/s	29.550 m/s	29.227 m/s
37.661 m/s	27.899 m/s	28.113 m/s
24.382 m/s	35.668 m/s	43.797 m/s
31.312 m/s	37.637 m/s	30.891 m/s

There are 12 numbers here. We say that there are 12 *samples*.

1.1 Mean

We often talk about the *average* of a set of samples, which is the same as the *mean*. To get the mean, sum up the samples and divide that number by the number of samples.

The numbers in that table sum to 388.599. If you divide that by 12, you find that the mean of those samples is 32.217 m/s.

We typically use the greek letter μ ("mu") to represent the mean.

Definition of Mean

If you have a set of samples $x_1, x_2, \ldots, x_n$, the mean is:

$$\mu = \frac{1}{n}\sum_{i=1}^{n} x_i$$

This may be the first time you are seeing a summation ($\sum$). The equation above is equivalent to:

$$\mu = \frac{1}{n}(x_1 + x_2 + \ldots + x_n)$$

Exercise 1 Mean Grade

Working Space

Teachers often use the mean for grading. For example, if you took six quizzes in a class, your final grade might be the mean of the six scores. Find the mean of these six grades: 87, 91, 98, 65, 87, 100.

Answer on Page 63

If you tell your friend, "I measured the speed of 1000 cars, and the mean is 31.71 m/s", your friend will wonder, "Are most of the speeds clustered around 31.71? Or are they all over the place and just happen to have a mean of 31.71?" To answer this question, we use variance.

1.2 Variance

Definition of Variance

If you have n samples $x_1, x_2, \ldots, x_n$ that have a mean of μ, the *variance* is defined to be:

$$v = \frac{1}{n}\sum_{i=1}^{n} (x_i - \mu)^2$$

That is, you figure out how far each sample is from the median, you square that, and then you take the mean of all those squared distances.

x	$x - \mu$	$(x - \mu)^2$
30.462	-1.755	3.079
29.550	-2.667	7.111
29.227	-2.990	8.938
37.661	5.444	29.642
27.899	-4.318	18.642
28.113	-4.104	16.839
24.382	-7.835	61.381
35.668	3.451	11.912
43.797	11.580	134.106
31.312	-0.905	0.818
37.637	5.420	29.381
30.891	-1.326	1.757
$\sum x = 386.599$		$\sum (x - \mu)^2 = 323.605$
mean = 32.217		variance = 26.967

Thus, the variance of the 12 samples is 26.967. The bigger the variances, the farther the samples are spread apart; the smaller the variances, the closer samples are clustered around the mean.

Notice that most of the data points deviate from the mu by 1 to 5 m/s. Isn't it odd that the variance is a big number like 26.967? Remember that it represents the average of the squares. Sometimes, to get a better feel for how far the samples are from the mean, we use the square root of the variance, which is called *the standard deviation*.

The standard deviation of your 12 samples would be $\sqrt{26.9677} = 5.193$ m/s.

The standard deviation is used to figure out a data point is an outlier. For example, if you are asked, "That car that just sped past. Was it going freakishly fast?" You might respond, "No, it was within a standard deviation of the mean." or "Yes, its speed was 2 standard deviations more than the mean. They will probably get a ticket."

A singular μ usually represents the mean. σ usually represents the standard deviation. So σ^2 represents the variance.

Exercise 2 Variance of Grades

Working Space

Now, find the variance for your six grades. As a reminder, they were: 87, 91, 98, 65, 87, 100.

What is your standard deviation?

Answer on Page 63

1.3 Median

Sometimes you want to know where the middle is. For example, you want to know the speed at which half the cars are going faster and half are going slower. To get the median, you sort your samples from smallest to largest. If you have an odd number of samples, the one in the middle is the median. If you have an even number of samples, we take the mean of the two numbers in the middle.

In our example, you would sort your numbers and find the two in the middle:

24.382
27.899
28.113
29.227
29.550
30.462
30.891
31.312
35.668
37.637
37.661
43.797

You take the mean of the two middle numbers: $(30.462 + 30.891)/2 = 30.692$. The median speed would be 30.692 m/s.

Medians are often used when a small number of outliers majorly skew the mean. For example, income statistics usually use the median income because a few hundred billionares

raise the mean a great deal.

Exercise 3 Median Grade

Working Space

Find the median of your six grades: 87, 91, 98, 65, 87, 100.

Answer on Page 63

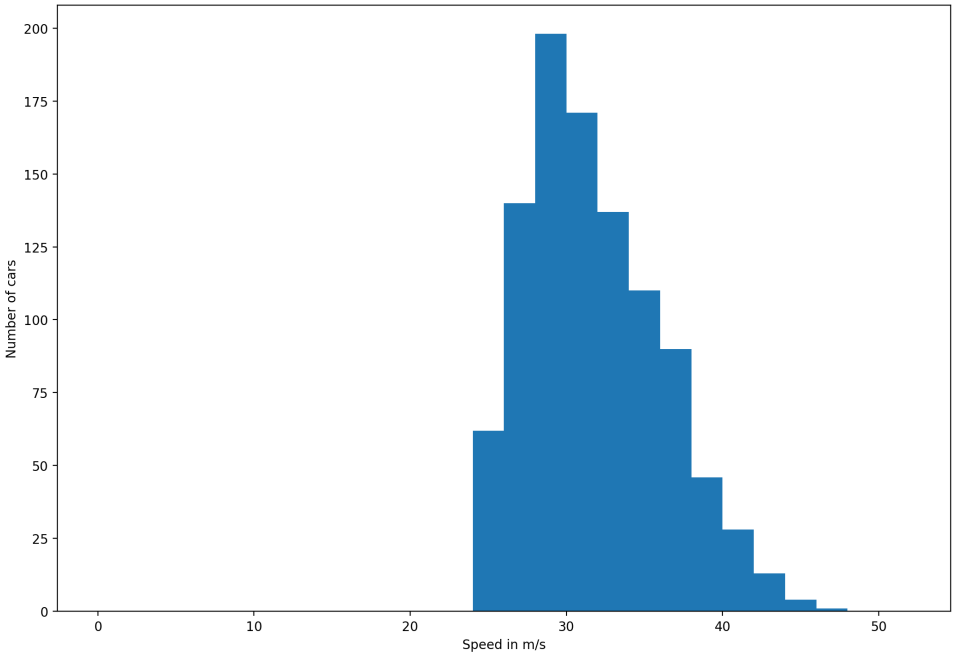
1.4 Histograms and Bell Curves

A histogram is a bar chart that shows how many samples are in each group. In our example, we group cars by speed. Maybe we count the number of cars going between 30 and 32 m/s. Next, we count the cars going between 32 and 34 m/2. Finally, we make a bar chart from that data.

Your 1000 cars would break up into these groups:

0 - 2 m/s	0 cars
2 - 4 m/s	0 cars
4 - 6 m/s	0 cars
...	...
20 - 22 m/s	0 cars
22 - 24 m/s	0 cars
24 - 26 m/s	65 cars
26 - 28 m/s	160 cars
28 - 30 m/s	175 cars
30 - 32 m/s	168 cars
32 - 34 m/s	150 cars
34 - 36 m/s	114 cars
36 - 38 m/s	79 cars
38 - 40 m/s	52 cars
40 - 42 m/s	20 cars
42 - 44 m/s	12 cars
44 - 46 m/s	4 cars
46 - 48 m/s	1 cars
48 - 50 m/s	0 cars

Next, we make a bar chart from that:



Often, a histogram will tell the story of the data. Here, you can see that no one is going less than 24 m/s, but a lot of people travel at 30 m/s. There are a few people who travel over 40 m/s, but there are also a couple of people who drive much faster than anyone else.

If we look closely at the shape of our histogram, we notice something interesting: it has a smooth, rounded hill in the middle and it tapers off on both sides. This is a common pattern in statistics, and it often suggests that the data follows what's called a **normal distribution**, also known as a **bell curve**.

A **bell curve** is a continuous curve that models how data is distributed when:

- Most values are near the average (mean), located at the peak of the curve,
- Fewer values are far from the mean, the tails of the curve,
- The distribution is roughly symmetric.

In our case, the majority of cars are traveling around 30–32 m/s. There are fewer cars going slower or faster than that. If we collected even more data (say, 10,000 cars), the histogram would start to resemble a smooth bell curve even more closely. The units of the x-axis (speed) would still be the same, but the y-axis would represent the *probability density* of finding a car at a certain speed, rather than just the count of cars in each speed

range.

The peak of the bell curve represents the **mean speed**, and the spread of the curve is related to the **standard deviation** — a measure of how spread out the speeds are. Most data falls within:

- 1 standard deviation of the mean ($\mu \pm \sigma$): about 68% of the cars,
- 2 standard deviations: about 95%,
- 3 standard deviations: about 99.7%.

This means that if we know the mean speed and the standard deviation, we can predict how many cars will be within certain speed ranges. If you have ever heard of Six Sigma methodology, that means data falls within six standard deviations of the mean, which is a very high level of quality control (3.4 defects per million measurements!).

So when we say a dataset "looks like a bell curve," we mean that it has a predictable and symmetric structure, often meaning it is a reliable set of data with few outliers and consistent results.

1.5 Root-Mean-Squared

Scientists have a mean-like statistic that they love. It is named quadratic mean, but most just calls it Root-Mean-Squared, or RMS.

Definition of RMS

If you have a list of numbers $x_1, x_2, \ldots, x_n$, their RMS is

$$\sqrt{\frac{1}{n}\left(x_1^2 + x_2^2 + \ldots + x_n^2\right)}$$

You are taking the square root of the mean of squares of the samples, thus the name Root-Mean-Squared.

Using your 12 samples:

x	x^2
30.462	927.933
29.550	873.203
29.227	854.218
37.661	1418.351
27.899	778.354
28.113	790.341
24.382	594.482
35.668	1272.206
43.797	1918.177
31.312	980.441
37.637	1416.544
30.891	954.254
Mean of x^2	1064.875
RMS	32.632

Why is RMS useful? Let's say that all cars had the same mass m, and you need to know what the average kinetic energy per car is. If you know the RMS of the speeds of the cars is v_{rms}, the average kinetic energy for each car is

$$k = \frac{1}{2}mv_{rms}^2$$

(You don't believe me? Let's prove it. Substitute in the RMS:

$$k = \frac{1}{2}m\sqrt{\frac{1}{n}\left(x_1^2 + x_2^2 + \ldots + x_n^2\right)}^2$$

The square root and the square cancel each other out:

$$k = \frac{1}{2}m\frac{1}{n}\left(x_1^2 + x_2^2 + \ldots + x_n^2\right)$$

Use the distributive property:

$$k = \frac{1}{n}\left(\frac{1}{2}mx_1^2 + \frac{1}{2}mx_2^2 + \ldots + \frac{1}{2}mx_n^2\right)$$

That is all the kinetic energy divided by the number of cars, which is the mean kinetic enegy per car. Quod erat demonstrandum! (That is a Latin phrase that means "which is what I was trying to demonstrate". You will sometimes see "QED" at the end of a long mathematic proof.)

Now you are ready for the punchline: Kinetic energy and heat are the same thing. Instead of cars, heat is the kinetic energy of molecules moving around. More on this soon.

Video: Mean, Median, Mode: https://www.youtube.com/watch?v=5C9LBF3b65s

CHAPTER 2

Basic Statistics in Spreadsheets

When you completed the problems in the last section, you probably noticed how long it took to compute statistics like the mean, median, and variance by hand. Luckily, computers were designed to free us from these sorts of tedious tasks. The most basic tool for automating calculations is the spreadsheet program.

The first spreadsheet program (VisiCalc) was introduced in 1979 as a tool for finance people to play "what if" games. For example, a company might make a spreadsheet that told them how much more profit they would make if they changed from using an expensive metal to using a cheaper alloy.

There are lots of spreadsheet programs, including Microsoft's Excel, Google Sheets, and Apple's Numbers. Any spreadsheet program will work; they are all very similar. The instructions and screenshots here will be from Google Sheets — a free spreadsheet program you use through your web browser.

2.1 The Barrel Problem

In honor of this history, let's start by studying a business question: You have a friend who dreams of quitting her job to become a cooper. (A cooper makes barrels that are used for aging wine and whiskey.) According to her:

- It costs $45 dollars in materials to build one barrel.
- A barrel sells for $100 dollars.
- The workshop/warehouse she wants to rent costs $2000 per month.
- Taxes take 20% of her profits.
- She needs to make $4000 monthly after taxes.

She has asked you, "How many barrels do I need to make each month?"

2.2 Solving It Symbolically

Many problems can be solved two ways: symbolically or numerically. To solve this problem symbolically, you would write out the facts as equations or inequalities, then do

symbol manipulations until you ended up with an answer. In this case, you would let b be the number of barrels and create the following inequality:

$$(1.0 - 0.2)\,(b(100 - 45) - 2000) \geq 4000$$

You would simplify it:

$$(0.8)\,(55b - 2000) \geq 4000$$

And simplify it more:

$$44b - 1600 \geq 4000$$

If that is true, then:

$$44b \geq 5600$$

And if that is true, then:

$$b \geq \frac{1400}{11}$$

1400/11 is about 127.27, so she needs to make and sell 128 barrels each month.

That is a perfect answer, and we didn't need a spreadsheet at all. However:

- As problems get larger and more realistic, it gets much more difficult to solve them symbolically.
- As soon as you say "Yes, you need to make and sell 128 barrels each month." Your friend will ask "What if I make and sell 200 barrels? How much money will I make then?"

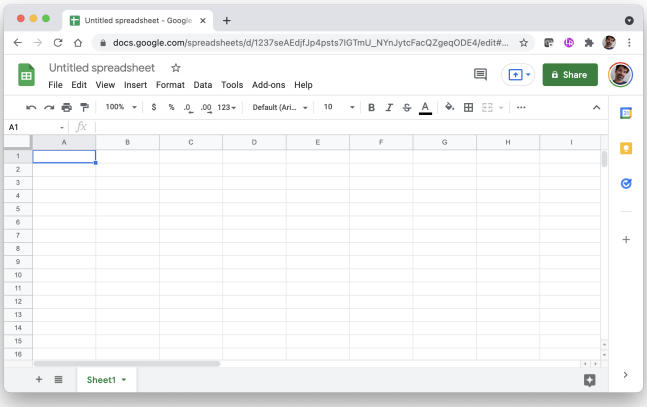
So, we use a spreadsheets to solve the problem numerically.

2.3 Your First Spreadsheet

Let's first make an inital spreadsheet.

In whatever spreadsheet program you are using, create a new spreadsheet document.

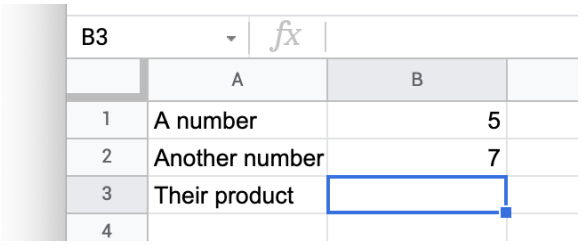
A spreadsheet is essentially a grid of cells. In each cell, you can put data (like numbers or text) and formulas.



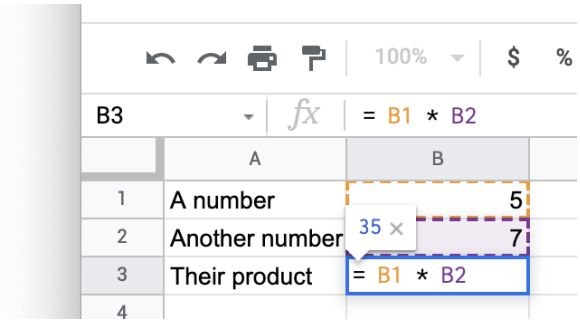
Let's put some labels in the column:

- Select the first cell (A1) and type "A number".
- Select the cell below it (A2) and type "Another number".
- Select the cell below that one (A3) and type "Their product".
- In the next column, type the number 5 in B1 and 7 in B2.

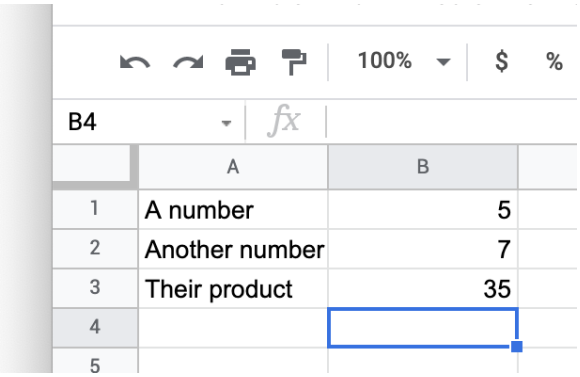
It should look like this:



Now, put a formula in cell B3. Select B3, and type "= B1 * B2". The spreadsheet knows this is a formula because it starts with '='. It will look like this as you type:



When you press Return or Tab, the spreadsheet will remember the formula, but display its value:



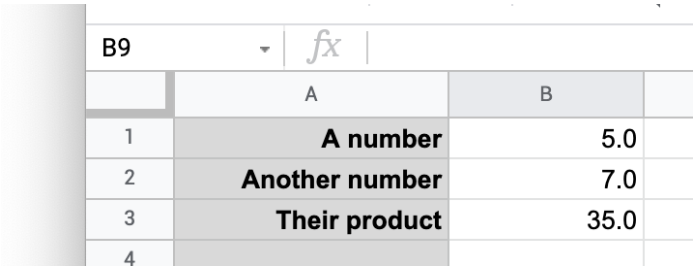
If you change the values of cell B1 or B2, the cell B3 will automatically be recalculated. Try it.

2.4 Formatting

Every spreadsheet lets you change the formatting of your columns and cells. They are all a little different, so play with your spreadsheet a little now. Try to do the following:

- Set the background of the first column to light gray.
- Right-justify the text in the first column.
- Make the text in the first column bold.
- Make the numbers in the second column have one digit after the decimal point.

It should look something like this:



That's a spreadsheet. You have a grid of cells, each of which can hold a value or a formula that uses values from other cells. The cells containing formulas automatically update as you edit the values in the other cells.

2.5 Comma-Separated Values

A large amount of data is exchanged in a file format called *Comma-Separated Values* or just CSV. Each CSV file holds one table of data. It is a text file, and each line of text corresponds to one row of data in the table. The data in each column is separated by a comma. The first line of a CSV is usually the names of the columns. A CSV might look like this:

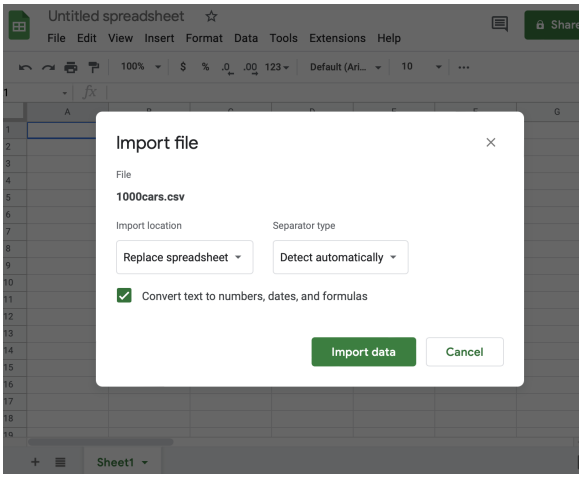
```
studentID,firstName,lastName,height,weight
1,Marvin,Sumner,260,45.3
2,Lucy,Harris,242,42.2
3,James,Boyd,261,44.2
```

In your digital resources for this module, you should have a file called `1000cars.csv`. It is a CSV with only one column called "speed". The first few lines look like this:

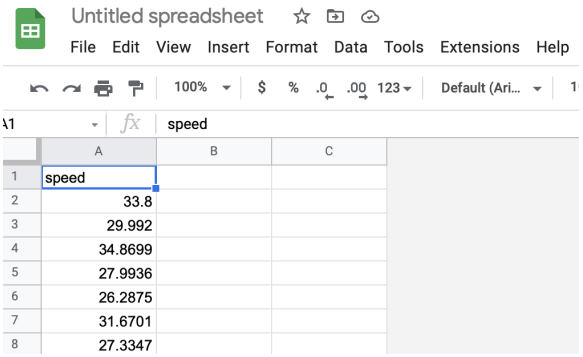
```
speed
33.8000
29.9920
34.8699
27.9936
```

There is a title line and 1000 data lines.

Import this CSV into your spreadsheet program. In Google Sheets, it looks like this:



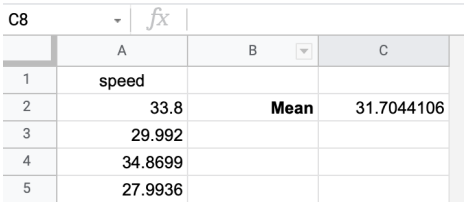
You should see a long, long column of data appear. (Mine goes from cell A2 through A1001.)



2.6 Statistics in Spreadsheets

Let's take the mean of all 1000 numbers. In cell B2, type in a label: "Mean". (Feel free to format your labels as you wish. Bolding is recommended.)

In cell C2, enter the formula "=AVERAGE(A2:A1001)". When you press return, the cell will show the mean: 31.70441, if done correctly.



Notice that by specifying that the function `AVERAGE` was to be performed on a range of cells: cells A2 through (":") A1001.

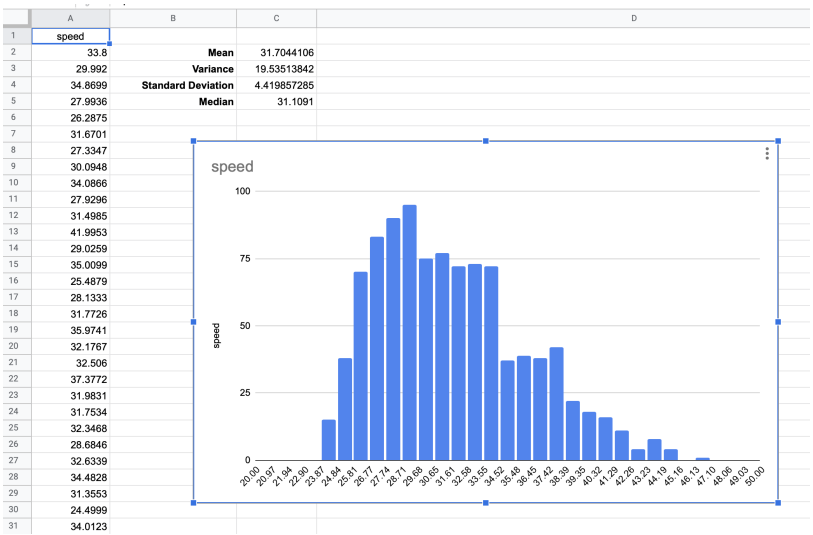
Do the calculations for variance, standard deviation, and median.

- The function for variance is `VAR`.
- The function for standard deviation is `STDEV`.
- The function for median is `MEDIAN`.

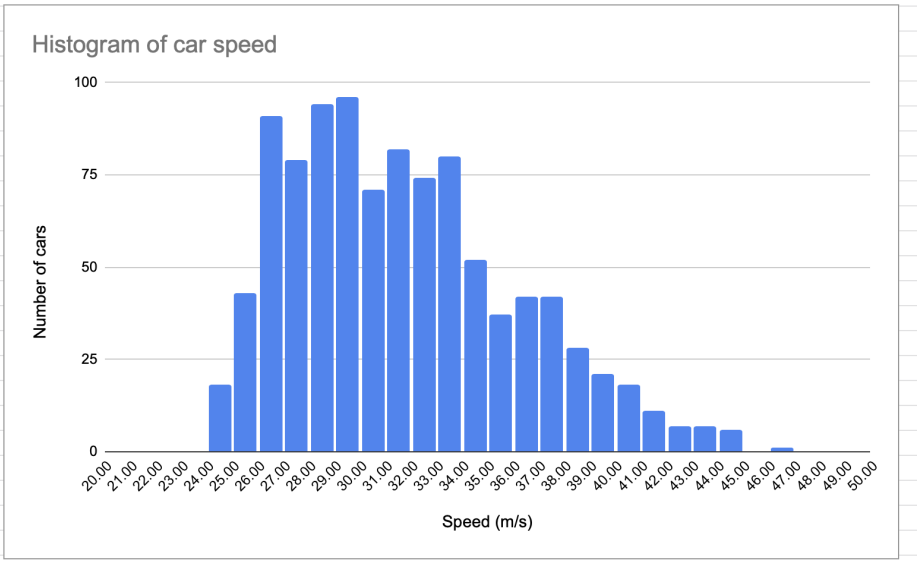
	A	B	C
1	speed		
2	33.8	**Mean**	31.7044106
3	29.992	**Variance**	19.53513842
4	34.8699	**Standard Deviation**	4.419857285
5	27.9936	**Median**	31.1091
6	26.2875		
7	31.6701		

2.7 Histogram

Most spreadsheets have the ability to create a histogram. In Google Sheets, you select the entire range A2:A1001 by selecting the first cell and then shift-clicking the last. Next, you choose Insert→Chart. In the inspector, change the type of the chart to a histogram (at the bottom under "other"). This will get you a basic histogram.



Play with the formatting to see how unique you can make data. Here is an example:



Exercise 4 RMS

Working Space

In your spreadsheet, calculate the quadratic mean (the root-mean-squared) of the speeds.

You will need the following three functions:

- `SUMSQ` returns the sum of the squares of a range of cells.
- `COUNT` returns the number of cells in a range that contains numbers.
- `SQRT` returns the square root of a number.

Answer on Page 63

2.8 The Return of the Barrel Problem

Let's get back to our example. Put labels in the A column:

- Barrels produced (per month)
- Materials cost (per barrel)
- Sale price (per barrel)
- Pre-tax earnings (per month)
- Taxes (per month)
- Take home pay (per month)

Format them any way you like. It should look something like this:

	A
1	**Barrels Produced** (per month)
2	**Materials cost** (per barrel)
3	**Sale price** (per barrel)
4	**Rent** (per month)
5	**Pretax Earnings** (per month)
6	**Taxes** (per month)
7	**Take home pay** (per month)
8	

In the B column, the first four cells are values (not formulas):

- 115 formatted as a number with no decimal point
- 45 formatted as currency
- 100 formatted as currency
- 2000 formatted as currency

It should look something like this:

	A	B
1	**Barrels Produced** (per month)	115
2	**Materials cost** (per barrel)	$45.00
3	**Sale price** (per barrel)	$100.00
4	**Rent** (per month)	$2,000.00

The next three cells in the B column will have formulas:

- B1 * (B3 - B2) - B4
- 0.2 * B5

- B5 - B6

It should look something like this:

	A	B
1	**Barrels Produced** (per month)	115
2	**Materials cost** (per barrel)	$45.00
3	**Sale price** (per barrel)	$100.00
4	**Rent** (per month)	$2,000.00
5	**Pretax Earnings** (per month)	$4,325.00
6	**Taxes** (per month)	$865.00
7	**Take home pay** (per month)	$3,460.00
8		

Now you can share this spreadsheet with your friend, and she can put different values into the cells for what-if games, such as "If I can get my materials cost down to $42 per barrel, what happens to my take home pay?"

Sometimes it is nice to show a range of values for a variable or two. In this case, it might be nice to show your friend what the numbers look like if she produces 115, 120, 125, 130, 135, or 140 barrels per month.

We have one column, and now we need six. How do we duplicate cells?

1. Click B1 to select it, then shift-click on B7 to select all seven cells.
2. Copy them. (Depending on what program you are using, there is likely a menu item for this.)
3. Click C1 to select it
4. Paste them.

	A	B	C
1	**Barrels Produced** (per month)	115	115
2	**Materials cost** (per barrel)	$45.00	$45.00
3	**Sale price** (per barrel)	$100.00	$100.00
4	**Rent** (per month)	$2,000.00	$2,000.00
5	**Pretax Earnings** (per month)	$4,325.00	$4,325.00
6	**Taxes** (per month)	$865.00	$865.00
7	**Take home pay** (per month)	$3,460.00	$3,460.00
8			

We want the first cell in the new column to be 120. You could just type in 120, but let's do something more clever. Put a formula into that cell: = B1 + 5. Now, the cell should show 120.

Why did we put in a formula? When we duplicate this column, this cell will always have 5 more barrels than the cell to its left.

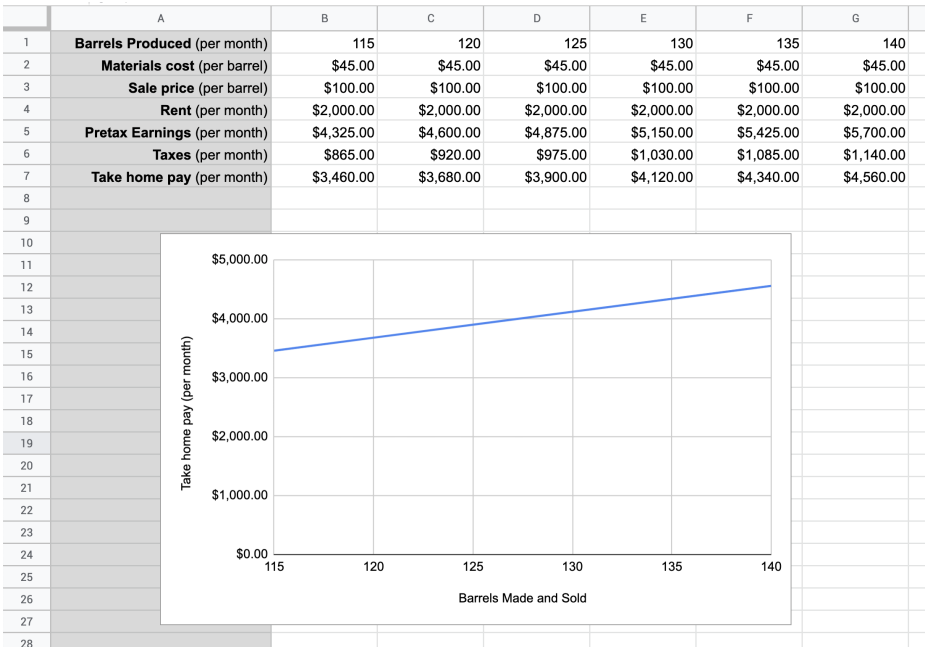
Next, let's duplicate the second column a few times. The easy way to do this is to select the cells as you did before, then drag the lower-right corner to the right until column G is in the selection. When you end the drag, the copies will appear:

	A	B	C	D	E	F	G
1	**Barrels Produced** (per month)	115	120	125	130	135	140
2	**Materials cost** (per barrel)	$45.00	$45.00	$45.00	$45.00	$45.00	$45.00
3	**Sale price** (per barrel)	$100.00	$100.00	$100.00	$100.00	$100.00	$100.00
4	**Rent** (per month)	$2,000.00	$2,000.00	$2,000.00	$2,000.00	$2,000.00	$2,000.00
5	**Pretax Earnings** (per month)	$4,325.00	$4,600.00	$4,875.00	$5,150.00	$5,425.00	$5,700.00
6	**Taxes** (per month)	$865.00	$920.00	$975.00	$1,030.00	$1,085.00	$1,140.00
7	**Take home pay** (per month)	$3,460.00	$3,680.00	$3,900.00	$4,120.00	$4,340.00	$4,560.00

Nice, right? Now your friend can easily see how many barrels correspond to how much take-home pay. But do you know what would be even more helpful? A graph.

2.9 Graphing

Graphing is a little different on every different platform. Here is what you want the graph to look like:



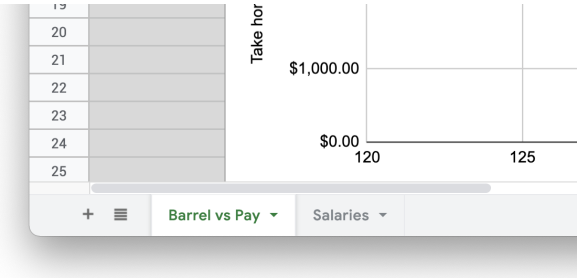
On Google Sheets:

1. Select cells B7 through G7.
2. Choose the menu item Insert -> Chart.
3. Choose the chart type (Line)

4. Add the X-axis to be B1 through G1.
5. Under the Customize tab, Set the label for the X-axis to be "Barrels Made and Sold".
6. Delete the chart title (which is the same as the Y-axis label).

2.10 Other Things You Should Know About Spreadsheets

Your spreadsheet document can have several "Sheets". Each has its own grid of cells. The sheet has a name; usually, you call it something like "Salaries". When you need to use a value from the "Salaries" sheet in another sheet, you can specify "Salaries!A2" — that is, cell A2 on sheet "Salaries". To flip between the sheets, there is usually a tab for each at the bottom of the document.



By default, the cell references are relative. In other words, when you write a formula in cell H5 that references the value in cell G4, the cell remembers "The cell that is one up and one to the left of me." Thus, if you copy that formula into B9, now that formula reads the value from A8.

If you want an absolute reference, you use $. If H5 references G4, G4 will be used no matter where on the sheet the formula is copied to.

You can use the $ on the row or column. In $A4, the column is absolute and the row is relative. In A$4, the row is absolute and the column is relative.

2.11 Challenge: Make a spreadsheet

You have a company that bids on painting jobs. Make a spreadsheet to help you do bids. Here are the parameters:

- The client will tell you how many square meters of wall needs to be painted.
- Paint costs $0.02 per square meter of wall
- On average, a square meter of wall takes 0.02 hours to paint.

- You can hire painters at $15 per hour.
- You add 20% to your estimated costs for a margin of error and profit.

Make a spreadsheet such that when you type in the square meters to be painted, the spreadsheet tells you how much you will spend on paint and labor. It should also tell you what your bid should be.

Chapter 3

Introduction to Python

In this chapter we will discuss the very basics of Python. No we are not talking about snake anatomy, but the programming language Python. Python is a high-level, interpreted programming language known for its readability and versatility. It is widely used in various fields such as web development, data analysis, artificial intelligence, scientific computing, and more. It can be used for data parsing, visualization, and even machine learning. Python's syntax is designed to be easy to read and write, making it an excellent choice for beginners and experienced programmers alike. Throughout this chapter, we encourage you to write out the provided lines of code yourself! This will reinforce skills like computer literacy, Python syntax and troubleshooting, and of course, typing speed.

3.1 Getting Started with Python

To get started with Python, you need to have it installed on your computer. You can download the latest version of Python from the official website: https://www.python.org/downloads/. Follow the installation instructions for your operating system.

Once you have python installed, you can write and run Python code using various methods:

- **Interactive Shell:** You can open a terminal or command prompt and type `python` or `python3` to start an interactive Python shell where you can type and execute Python commands line by line.
- **Script Files:** You can write your Python code in a text file with a `.py` extension and run it using the command `python filename.py` or `python3 filename.py` in the terminal.
- **Integrated Development Environments (IDEs):** There are several IDEs available for Python, such as PyCharm, VSCode, and Jupyter Notebooks, which provide a more user-friendly environment for writing and running Python code. We will assume you will use VSCode, as it is widely used among computer scientists and devlopers around the world, and has direct integration to GitHub.

We are going to install VSCode as our IDE for this course. You can download it from https://code.visualstudio.com/. After installing VSCode, you will also need to install the Python extension for VSCode, which can be found in the Extensions Marketplace within VSCode.

3.1.1 The Console and Running Python Programs

When working with Python, it is important to understand the difference between *writing code* and *executing code*. Writing code simply means typing instructions into a file using a text editor or IDE. Executing code means telling Python to read those instructions and perform them.

The *console*, also called the *terminal* or *command prompt*, is a text-based interface where you can run programs and view their output. Any text printed using the `print()` function will appear in the console.

When you run a Python program, Python reads your file from top to bottom and executes each line in order. This is an important idea that will come up often: **Python code is executed sequentially, one line at a time**.

3.1.2 Running a Python File

To run a Python file, you must use the console. In VSCode, you can open the integrated terminal by selecting `Terminal` → `New Terminal` from the menu. Make sure your terminal is open in the same folder as your Python file.

If your file is named `hello.py`, you can run it by typing:

```
python hello.py
```

or, on some systems:

```
python3 hello.py
```

After pressing Enter, Python will execute the file and display any output in the console. If there are errors in your code, Python will stop execution and display an error message instead. You may also see a green play button at the top of your window, you could also run your program with that!

At this point, it is helpful to remember:

- Writing code does nothing by itself
- Code only runs when you explicitly execute the file
- Output from `print()` appears in the console

3.2 Output `print`

Let's talk more about the `print()` statement. It can take a string, multiple variables, or a combination of strings and variables. We are going to introduce the syntax first, and then expand on the definitions later.

```
x = "This is a variable."
print("This is a string.", x)
```

This should output `This is a string. This is a variable`. You can also do multiple variables:

```
x = "Good"
y = "Morning,"
z = "Chicago"
print(x, y, z)
```

This will output `Good Morning, Chicago`. Alternatively, printing can have addition within it:

```
x = "Good"
y = "Morning,"
z = "Chicago"
print(x+y+z)
```

This will output `GoodMorning,Chicago`. Notice the difference!

If you want to combine numbers and strings,

```
x = "it is"
y=78
z="degrees outside"
print(x+y+z)
```

This will not work! Instead, we should use commas:

```
x = "it is"
y=78
z="degrees outside"
print(x, y, z)
```

This will correctly output "it is 78 degrees outside"

By default, a space is added in between each argument (where the commas would be).

We can change what seperates each variable by the `sep=` argument.

```
x = "it is"
y=78
z="degrees outside"
print(x, y, z, sep="..!..")
```

This will output `it is..!..78..!..degrees outside`

By default, Python includes a new line at the end of each print statement. We can alter this to anything else by using the `end=` argument:

```
print("Good Morning, Chicago.", end="!!")
```

Outputting `Good Morning, Chicago.!!` to the console, with no new line character inserted. Note that the next print statement will continue on the same line unless it contains a new line character.

```
print("Hello World!", end=" ")
print("I will print on the same line.")
```

We have used all this terminology like strings and variables a lot, let's dive a bit deeper into it!

3.3 Datatypes and Variables

There are many different datatypes in Python, and all of them can be passed to `print` function. Here's the main few:

Strings Strings are just a sequence of characters. Anything closed between quotes gets included in the string, such as "This is the Python Chapter for the Kontinua Sequence!!%& × ()-+=1234567890 &"

Integers Integers are any whole number, positive negative or 0. Theoretically, integer values can go on for infinity, but it is important to understand they take up space in computer memory.

Floats Floats, or floating point values, are numbers that include decimal places. Division can be done between integers and floats but may result in a different value.

Booleans Booleans only two values: `True` or `False`. The output of conditional statements

(such as if's or while's) are booleans. They only answer Yes and No questions, and are important on deciding between two possible paths in code.

Sequence Types Sequences are consecutive lists or pairs of other data types can be represented in various froms: `list`, `set`, and `tuple`. Values are accessed by positions.

Mapping Types The main mapping type is a `dict`, short for dictionary. Mapping types store values as key–value pairs instead of positions. You access items using a key, not an index.

Let's create some of these in our code:

```
x = 5
y = 5.6
z = True
a = [x, y, z]
```

A *variable* is a container for information, usually containing one of the datatypes established above. In Python, variables are assigned using the assignment operator =, with the name of the variable on the left side of the operator and the value it is assigned to on the right side.

Python has specific naming rules for variables. The name

- Must start with a letter or underscore
- Cannot contain spaces
- Are case-sensitive

In our code above, `x` is a *variable* containing the number 5, an Integer. `y` contains the float 5.6, while `z` is a boolean with the value `True`. We then created a list containing the variables `x`, `y`, and `z`.

A unique feature of Python is that variables can change their datatypes even after assignment. We can see the datatype of an object using the `type()`

```
x = 10
print(x)
print(type(x))

x = "Hello!"
print(x)
print(type(x))
```

Output:

```
10
<class 'int'>
Hello!
<class 'str'>
```

Notice that, because of that feature, the datatype of a variable is not defined beforehand. This is a core difference between Python and other programming languages like Java or C++, which we will explore in the future.

We can also define our floats using scientific notation.

```
x = 1e3      # 1000.0
y = 2.5e-4   # 0.00025
```

Exercise 5 Identify the variable type

Working Space

Below are 5 variables defined in Python. Identify their type as `str`, `float`, `bool`, `list`, or `int` by filling in the table below.

```
a = 10
b = 3.5
c = "10"
d = True
e = [1, 2, 3]
```

Variable	Value	Data Type
a	10	
b	3.5	
c	”10”	
d	True	
e	[1, 2, 3]	

Answer on Page 64

Exercise 6 What does this do?

What is the output of the following code?

```
print(type(3.0) == type(3))
```

Working Space

Answer on Page 64

3.4 Operations

Operations can be done on both variables and numbers alike. Most commonly they will be used for mathematical operations or simplifying operations of equivalence.

3.4.1 Arithmetic Operations

Math operations in Python are very similar to standard math operations. We need them for any type of data operation or math parsing.

```
a = 100
b = 3
c = 100.0
d = -100
print(a + b) # addition
print(a - b) # subtraction
print(a * b) # multiplication
print(a / b) # division (results in a float)
print(a % b) # modulus (remainder) operator - useful for division checks
print(a ** b) # exponentiation
print(a // b) # floor division
print(c // b) # floor division as a float
print(d // b) # negative flooring goes further DOWN
```

Output:

```
103
97
300
33.333333333333336
```

```
1
1000000
33
33.0
34
```

Note that division by `0` is not a valid operation, just as in algebra. This will cause an `ZeroDivisionError`.

3.4.2 Augmented Assignment Operators

You may want to do the following operation

```
x = 0
while True:
    print(x)
    x = x + 1
```

This would print the counting numbers, increasing by 1, forever. The `x = x + 1` rewrites the value of `x` equal to the current value of `x`, and adds one to it. There is a short hand for this kind of operation, called an *augmented assignment operator*, which combines assignment operators and operations. These are commonly used in loops and accumulation counters:

```
x = 10
x += 5   # same as x = x + 5
x -= 3   # same as x = x - 3
x *= 2   # same as x = 2 * x
x /= 4   # same as x = x / 4
```

Following order of operations, `x = 6` after all lines are executed.

3.4.3 Summary

- Math operations are done with `+`, `-`, `*`, and `/`
- Augmented Operations `+=`, `-=`, `*=`, `/=` are common for loops and counting operations.

Exercise 7 Tracing Chart

Let's create a tracing chart. What happens after each line of code is run?

Working Space

```
n = 5
m = "3"
n = n + 1
m = m + "1"
```

Line Executed	n (value, type)	m (value, type)
After line 2		
After line 3		
After line 4		

Answer on Page 64

3.5 Input and Output

So far, we have only displayed information using the `print()` function. While output is useful, most programs also need a way to receive information from the user. In Python, this is done using the built-in `input()` function.

3.5.1 Getting User Input

The `input()` function pauses the program and waits for the user to type something into the console. Whatever the user types is then returned and can be stored in a variable.

```
name = input("Enter your name: ")
print("Hello,", name)
```

In this example:

- The text inside `input()` is displayed as a prompt in the console
- The program waits for the user to type a response and press Enter

- The entered text is stored in the variable `name`
- The `print()` function then outputs a greeting using that value

3.5.2 Input Always Returns a String

A very important rule in Python is that `input()` **always returns a string**, even if the user types a number. This means that mathematical operations cannot be performed on input values unless they are converted to a numeric type.

Consider the following incorrect example:

```
age = input("Enter your age: ")
print(age + 1)
```

This code will result in an error, because Python cannot add a number to a string.

To perform calculations, the input must be *cast* to a different datatype.

3.5.3 Casting Input Values

Casting is the process of converting one datatype into another. Python provides built-in functions such as `int()`, `float()`, and `str()` for this purpose.

```
age = int(input("Enter your age: "))
print(age + 1)
```

Here:

- `input()` returns a string
- `int()` converts that string into an integer
- The program can now perform arithmetic operations

3.5.4 Common Input Errors

If the user enters something that cannot be converted to the requested datatype, Python raises a runtime error called a `ValueError`. For example:

```
age = int(input("Enter your age: "))
```

If the user types `hello` instead of a number, Python will produce an error similar to the following:

```
ValueError: invalid literal for int() with base 10
```

This error occurs because the string `"hello"` cannot be converted into an integer. Handling these types of errors will be discussed later when we introduce `try-except` statements.

3.5.5 Summary

- `print()` is used to display output to the console
- `input()` is used to receive text input from the user
- `input()` always returns a string
- Casting is required to convert input into numeric types

Exercise 8 User Input Types

Working Space

```
x = input("Enter a number: ")
print(x + 5)
```

What will happen when this code is run and the user enters `10`? If there is an issue, explain what it is, why it occurs, and how to fix it.

Answer on Page 64

3.6 Conditionals, Loops, and Match-Case

Most programs need to make decisions and repeat actions. Programmers do not want to write the same code multiple times, as this introduces *redundancy*. In Python, this is done using *conditionals* and *loops*. Conditionals allow the program to choose between different paths of execution, while loops allow code to be run multiple times.

3.6.1 Conditional Statements

Conditional statements execute code only if a given condition is true. Python uses the keywords `if`, `elif`, and `else` to define these conditions.

```
x = 10

if x > 0:
    print("x is positive")
elif x == 0:
    print("x is zero")
else:
    print("x is negative")
```

In this example:

- The expression after `if` is evaluated as a boolean
- If the condition is `True`, the indented block runs
- If it is `False`, Python checks the next condition. The `elif` block is the only way to check another branch after one is returned `False`. The `else` branch is a last resort, and only runs if all previous statements are false
- Only one branch of the conditional will execute

It is important to note that **indentation is required** in Python. All statements belonging to a conditional block must be indented at the same level.

3.6.2 Boolean Expressions

Conditions are built using comparison and logical operators. These expressions always evaluate to either `True` or `False`.

```
a = 5
```

```
b = 10

print(a < b)   # less than
print(a == b)  # equal to
print(a != b)  # not equal to
```

Logical operators can be used to combine conditions. Condtions can be *strung together* using `and`, `or`, and `not`

```
x = 7

if x > 0 and x < 10:
    print("x is between 1 and 9")
if x > 0 or x < -3:
    print("x is greater than 0 or less than -3")

y = False
print(not y)
```

3.6.3 Loops

Loops allow code to be repeated multiple times. Python provides two main types of loops: `for` loops and `while` loops.

To define a loop, Python uses the keywords `for` and `while`, followed by a condition or sequence to iterate over.

- the loop header, with correctly defined endpoints
- a colon :
- an indented block of code to iterate over

3.6.4 For Loops

A `for` loop is used to iterate over a sequence, such as a list or a range of numbers.

```
for i in range(5):
    print(i)
```

This code will print the numbers 0 through 4. The `range()` function generates a sequence

of integers starting at 0 and stopping before the given value.

```
numbers = [10, 20, 30]

for n in numbers:
    print(n)
```

3.6.5 While Loops

A `while` loop repeats as long as a condition remains true.

```
x = 5

while x > 0:
    print(x)
    x -= 1
```

In this example:

- The condition is checked before each iteration
- The loop stops once the condition becomes `False`

Care must be taken to ensure that the condition eventually becomes false. Otherwise, the program will enter an infinite loop, one which never stops and the computer will run forever, risking hardware component issues.

3.6.6 Breaking and Continuing Loops

Python provides keywords to control loop execution.

```
for i in range(10):
    if i == 5:
        break
    print(i)
```

The `break` statement immediately exits the loop.

```
for i in range(5):
```

```
    if i == 2:
        continue
    print(i)
```

The `continue` statement skips the current iteration and moves to the next one.

3.6.7 Match-Case Statements

Python also provides the `match-case` statement, which allows a value to be compared against several possible patterns. This is similar to a switch statement in other programming languages.

```
command = input("Enter a command: ")

match command:
    case "start":
        print("Starting program")
    case "stop":
        print("Stopping program")
    case "pause":
        print("Pausing program")
    case _:
        print("Unknown command")
```

The underscore (`\_`) acts as a default case if no other patterns match.

3.6.8 Summary

- Conditionals allow programs to make decisions
- Boolean expressions evaluate to `True` or `False`
- Loops allow code to be repeated efficiently
- `for` loops iterate over sequences
- `while` loops repeat based on a condition
- `match-case` provides a structured way to handle multiple cases

Exercise 9 Predict the Output

Fill in the table with the outputs for the given inputs.

Working Space

```
x = int(input("Enter a number: "))

if x > 10:
    print("Large")
elif x == 10:
    print("Ten")
else:
    print("Small")
```

Input	Output
5	
10	
15	

Answer on Page 65

Exercise 10 Fill in the Blanks: Loop Edition

Fill in the following code to print all even numbers from 0 to 20 (inclusive).

Working Space

```
for i in __________:
    print(__________)
```

Answer on Page 65

3.7 Lists and Strings

We talked about creating strings, any set of characters between quotes. These can either be single quotes ' ' or double quotes " ".

Multiline strings must be surrounded by three quotations on each side of the text sequence.

```
a = """Lorem ipsum dolor sit amet,
consectetur adipiscing elit,
sed do eiusmod tempor incididunt
ut labore et dolore magna aliqua."""
```

Strings are an example of a *sequence type*, meaning they store values in a specific order and allow individual elements to be accessed using an index.

3.7.1 Indexing Strings

Each character in a string has a position, called an *index*. Indexing in Python begins at 0, so the first character is at index 0. Think of it like asking the question for each letter: "How far from the start is this element?" The first element is zero positions away from the start, so its index is 0. The last element is always at index $n-1$, where n is the length of string. In memory, all of these characters are stored consecutively.

```
word = "Python"

print(word[0])
print(word[1])
print(word[5])
```

Output:

```
P
y
n
```

Attempting to access an index that does not exist will result in a runtime error.

3.7.2 Length of a String

The number of characters in a string can be found using the built-in `len()` function.

```
word = "Python"
print(len(word))
```

Output:

```
6
```

3.7.3 Iterating Over Strings

Strings can be iterated over one character at a time using a `for` loop.

```
for char in "Hello":
    print(char)
```

This loop runs once for each character in the string.

3.7.4 String Methods

Strings come with built-in functions, called *methods*, that perform common operations. These methods are called using dot notation.

```
text = "  Python Programming  "

print(text.upper())
print(text.lower())
print(text.strip())
```

Common string methods include:

- `upper()` converts all characters to uppercase
- `lower()` converts all characters to lowercase
- `strip()` removes leading and trailing whitespace

3.7.5 Lists

A list is another sequence type used to store multiple values in a single variable. Lists are created using square brackets, with elements separated by commas.

```
numbers = [1, 2, 3, 4]
```

Lists can store values of different datatypes, including strings, numbers, booleans, and even other lists.

```
data = [10, 3.14, True, "Python"]
```

3.7.6 Indexing Lists

Like strings, lists are indexed starting at 0.

```
numbers = [10, 20, 30]

print(numbers[0])
print(numbers[2])
```

Output:

```
10
30
```

For nested lists, you search an indexe using a bracket for every nested list until you reach the element you are aiming to reach. Later, we will talk about `.json` files, which are files essentially comprised of strung-nested lists.

```
nested_numbers = [0, [1, 2, 3], [4, 5, 6], [7, 8, 9]]

print(numbers[0])
print(numbers[1][2])
print(numbers[2][1])
```

Output:

```
0
3
5
```

Slicing Strings

You can also use *slicing* in Python to get all elements between two indices. This syntax is Python-specific, although there are equivalences in other programming languages. This syntax works for *both* lists and strings.

For any sequence, we can slice it or operate on it with the following index notation:

```
sequence[start : stop : step]
```

where `start` is the index to begin at (inclusive), `end` is the index to end at (exclusive), and `step`, an optional argument for how many items to skip.

```
text = "abcdef"

text[1:4]    # 'bcd'
text[:3]     # 'abc'
text[3:]     # 'def'
text[::2]    # 'ace'
text[::-1]   # 'fedcba'  (reverse string)
```

The above example slices a string in a few different ways. Let's look at list slicing:

```
nums = [0, 1, 2, 3, 4, 5]

nums[2:5]    # [2, 3, 4]
nums[:4]     # [0, 1, 2, 3]
nums[4:]     # [4, 5]
nums[::2]    # [0, 2, 4]
nums[::-1]   # [5, 4, 3, 2, 1, 0]
```

Negative indices start from the end, and work the way up the sequence:

```
text = "python"

text[-3:]    # 'hon'
text[:-2]    # 'pyth'
```

Note that setting a variable to a sliced string creates a new string object, while leaving the original unchanged.

Exercise 11 Mystery Sequence

What will the code `seq[:]` output, assuming `seq = [a, f, j, k]` is a valid list?

Working Space

Answer on Page 65

Exercise 12 Computer... Indexing!

Working Space

You are given the following code:

```
word = "computer"
```

What is the output of each of the following expressions?

```
print(word[0])
print(word[2])
print(word[-1])
print(word[-3])
print(word[3:6])
```

Answer on Page 65

3.7.7 Modifying Lists

Unlike strings, lists are *mutable*, meaning their contents can be changed after creation.

```
numbers = [1, 2, 3]
numbers[1] = 10
print(numbers)
```

Output:

```
[1, 10, 3]
```

3.7.8 List Length and Iteration

The number of elements in a list can be found using the `len()` function.

```
numbers = [1, 2, 3]
print(len(numbers))
```

Lists can be iterated over using a `for` loop.

```
numbers = [1, 2, 3]

for n in numbers:
    print(n)
```

3.7.9 Common List Methods

Lists include built-in methods that allow elements to be added or removed.

```
numbers = [1, 2, 3]
numbers.append(4)
numbers.remove(2)
print(numbers)
```

Common list methods include:

- `append()` - adds an element to the end of the list
- `remove()` - removes the first occurrence of a value
- `pop()` – removes and returns an element by index

3.7.10 Strings vs Lists

Although strings and lists are both sequence types, there is an important difference between them.

- Strings are immutable and cannot be modified after creation
- Lists are mutable and can be changed

3.7.11 Summary

- Strings and lists are sequence types
- Indexing starts at 0
- The `len()` function returns the size of a sequence
- Strings are immutable
- Lists are mutable and support modification

Exercise 13 What's in your backpack?

Let's say the following code is run.

Working Space

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

items.append("ruler")
items.pop()
items.remove("book")
items.append("marker")
```

Fill in the following chart with the contents of the `items` list after each operation.

Line Execution	`items` content
After Line 1	`items = ["pen", "book", "eraser"]`
After Line 3	
After Line 4	
After Line 5	
After Line 6	

Answer on Page 65

3.8 Is there more?

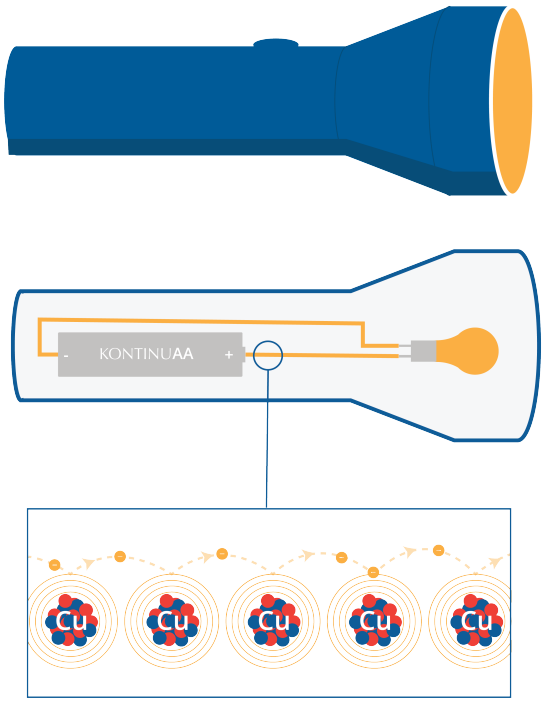
Of course, there is always something to learn with Python. We are going to do a deeper dive into functions, classes, and a few useful libraries in the next workbook. For now, this should be enough to get you started with Python. We will revisit many of these concepts in later chapters as we build more complex programs.

Chapter 4

Introduction to Electricity

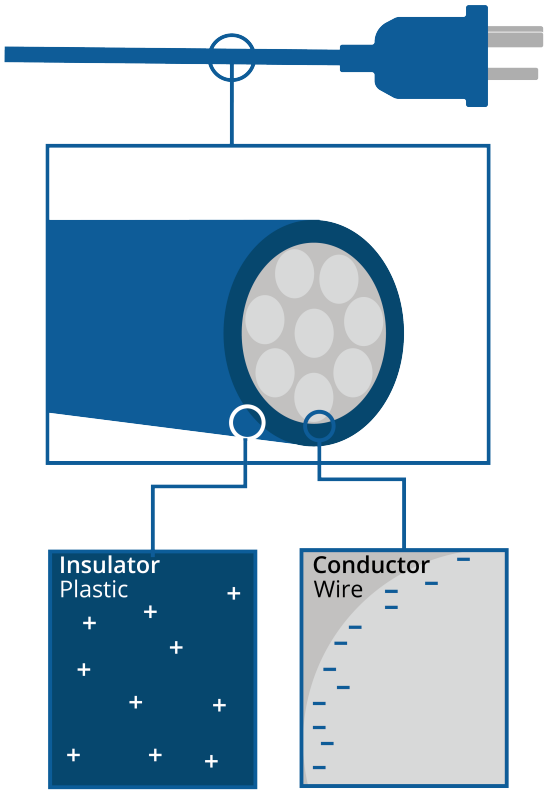
What happens when you turn on a flashlight? The battery in the flashlight acts as an electron pump, and the electrons flow through the wires to the lightbulb (or LED). As the electrons pass through the lightbulb, they excite the molecules within, which gives off light and heat. (LEDs also give off light and heat, but they give off much less heat.) The electrons then return to the battery to be pumped around again.

When electricity is flowing through a copper wire, the protons and neutrons of the copper stay put, while the electrons jump between the atoms on their way from the battery to the lightbulb and back again.



In some materials, like copper and iron, electrons are loosely bound to their nuclei, forming a sea of electrons, which allows energy to flow. These are good *electrical conductors*. In other materials, like glass and plastic, electrons don't leave their nuclei as easily. This makes them are terrible electrical conductors – we call these materials *electrical insulators*.

For example, the plastic around a wire is used for electrical insulation.



4.1 Units

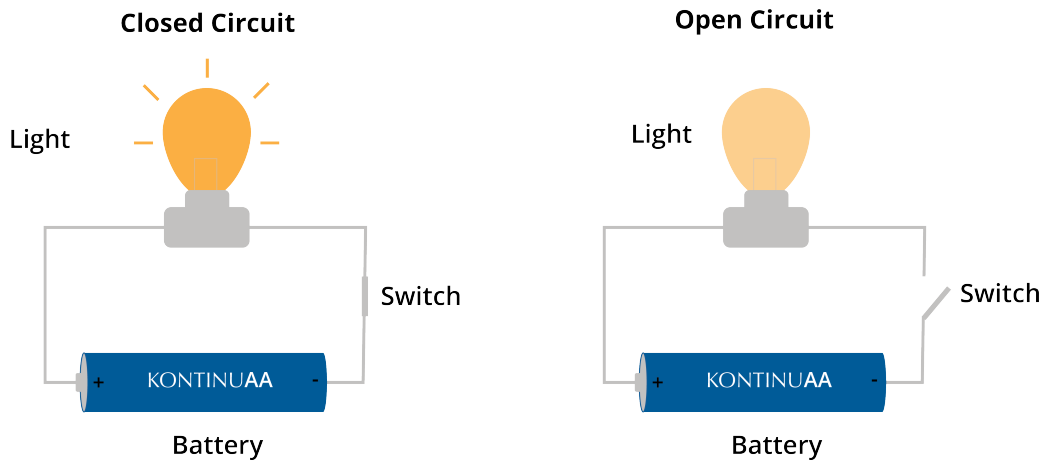
Electrons are very small, so to study them, scientists came up with a unit that represents *a lot* of electrons. 1 *coulomb* is about 6,241,509,074,460,762,608 electrons. When 5 coulombs enter one end of the wire every second (and simultaneously 5 coulombs exit the other end), we say "This wire is carrying 5 amperes of current."

(Truthfully, we usually shorten ampere to just "amp". This is sometimes a little awkward, because we also often shorten the word "amplifier" to "amp", but you should generally be able to tell which is which from the context.)

If you look at the circuit breakers or fuses for your home's electrical system, you'll see that each one is rated in amps. For example, maybe the circuit that supplies power to your kitchen has a 10 amp circuit breaker. If, for some reason, more than 10 amps tries to pass through that wire, the circuit breaker will turn off the whole circuit.

When your flashlight is on, it pushes about 1 amp of current through the lightbulb (When

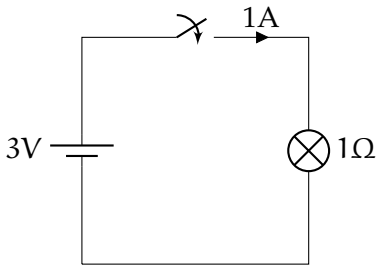
it is off, there is no current in the lightbulb).



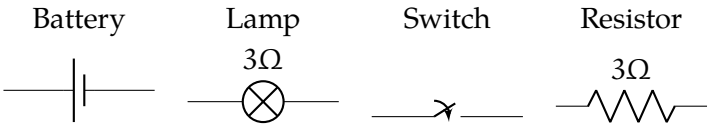
The lightbulb creates *resistance* that the current pushes through. Think of it like plumbing: The current is the amount of water passing through a pipe. The resistance is something that tries to stop the current – like a ball of hair, similar how different surfaces apply friction when pushing a box. The battery is what allows the current to push through the resistance; we call that pressure *voltage*. Especially in physics, voltage is often referred to as *electromagnetic potential*.

4.2 Circuit Diagrams

Here is a circuit diagram of your flashlight:



The lines are wires. The symbols that we will use are:



The battery pushes the electrons from the positive end (the larger line) to the negative end (the smaller line), so the circuit must go around in a circle for the current to flow. This is why the current stops flowing when the switch breaks the circuit.

You can think of a switch as having zero resistance when it is closed and infinite resistance when it is open.

For our purposes, a lamp is just a resistor that gives off light.

4.3 Ohm's Law

Resistance is measured in *ohms*, and we use a Greek capital omega for that: Ω

Voltage is measured in *volts*.

Ohm's Law

Whenever a voltage V is pushing a current I through a resistance of R, the following is true:

$$V = IR$$

where V is in volts, I is in amps, and R is in ohms.

4.4 Power and Watts

Joule's Law

When a current I is passing through a resistance R, the power consumed is

$$W = I^2R$$

where W is in watts, I is in amps, and R is in ohms.

Of course $V = IR$, so we can extend this to:

$$W = I^2R = IV = \frac{V^2}{R}$$

Your flashlight's batteries provide about 3 volts. How much battery power is the flashlight using when it is on? The power (in watts) produced by the battery is the product of the voltage (in volts) and the current (in amps). This means your flashlight is giving off $3volts \times 1amp = 3watts$ of power. Some of that power is given off as light, some as heat.

A watt is 1 joule of energy per second. We say that a watt is a measure of *power*.

When we talk about how much energy is stored in a battery, we use a unit like a kilowatt-hour. A kilowatt-hour is equivalent to 3.6 million joules.

4.5 Another great use of RMS

In many electrical problems, the voltage fluctuates a great deal. For example, the fluctuations in voltage makes the sound that comes out of an audio speaker.

You can use the root-mean-squared of the voltage to figure out the average power your speaker is consuming.

Let's say that the RMS of the voltage you are sending to the speaker is V_{rms} and the resistance of the speaker is R ohms. This means the power consumed by the speaker is:

$$P = \frac{V_{rms}^2}{R}$$

Similarly, if you know the RMS of the current you are pushing through the speaker is I_{rms}, then the power consumed by the speaker is:

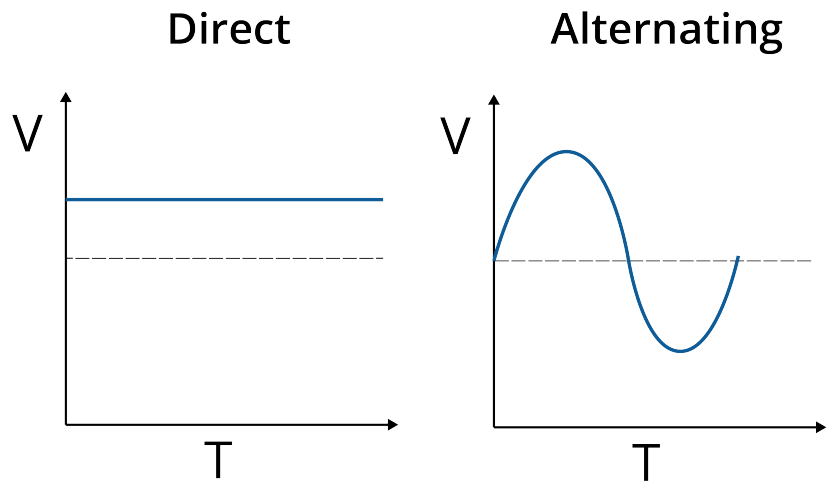
$$P = I_{rms}R$$

4.6 Electricity Dangers

Large amounts of electricity moving through your body can hurt or even kill you. You must be careful around electricity.

That said, your body is not a very good conductor, so low-voltage systems (like a flashlight) don't have enough voltage to move significant amounts of current through your body.

However, the electricity in a power outlet has much more voltage. The voltage in these outlets is fluctuating between positive and negative, so we call it *Alternating Current* or AC.



In most countries, the RMS of the voltage between 110 and 240 V. (The peak voltage is always $\sqrt{2}$ times the RMS value. In the US, for example, people say "Our outlets supply 120 V." They mean that the RMS of the voltage difference between the wire and the earth is 120V. The peak voltage is almost 170V.)

How much current can a human handle? Not much. You can barely feel 1 mA moving through your body, but at 16 mA, your muscles will clench and you won't be able to relax them — many people die from electrocution because they grab a wire which pushes enough current through their body to prevent them from letting go of the wire. At 20 mA, a human's respiratory muscles become paralyzed.

The fuse breaker in a house will often allow 20 A to flow through the circuit before it shuts off the power. Always be very sure to shut off the power before touching any of the wiring in your house.

While water is actually a mediocre conductor, it can still deliver enough current to kill you. If you see a wire in a puddle, you should not touch the puddle. Interestingly, because of the salt, sea water is more than 100 times better at conducting electricity than the water you drink.

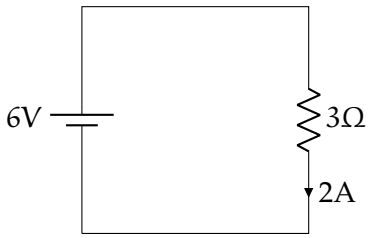
If you hold a wire in each hand, how many Ohms of resistance will your body have? Once it gets past your skin, you will look like a bag of salt water to the electricity. After the

skin, your body will have a resistance of about 300Ω. However, the skin is a pretty good insulator. If you have dry, calloused hands, your skin may add 100,000Ω to the resistance.

Chapter 5

DC Circuit Analysis

In the most basic circuit, you have only a battery and a resistor:

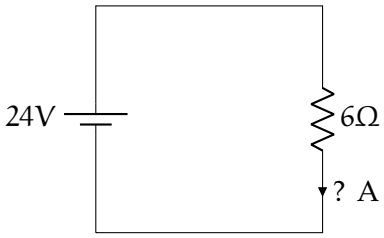


For this situation, you only need Ohm's Law: $V = IR$. In this case, $6V = 3\Omega \times 2A$.

Exercise 14 Ohm's Law

Working Space

How many amps are going around the circuit?

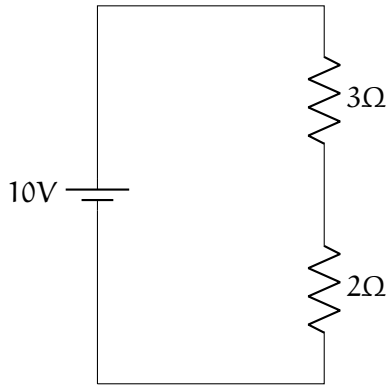


Answer on Page 66

5.1 Resistors in Series

When you have two resistors wired together in a long line, we say they are "in series." If you have two resistors R_1 and R_2 wired in series, the total resistance is $R_1 + R_2$.

In this diagram, for example, the total resistance is 5Ω.



The current flowing through the circuit, then, is $10/5 = 2A$.

By Ohm's law, the voltage drop across the upper resistor is $IR = 2A \times 3\Omega = 6V$.

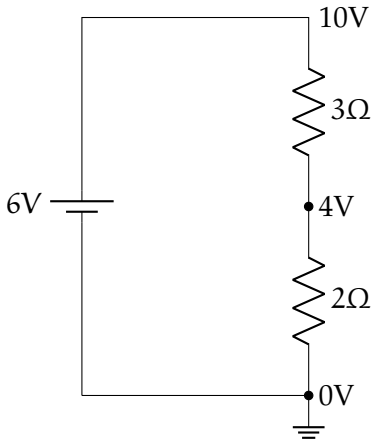
The voltage drop across the lower resistor is $IR = 2A \times 2\Omega = 4V$.

Notice that the battery pumps the voltage up to $10V$, then the two resistors drop it by exactly $10V$. This is known as "Kirchhoff's Voltage Law":

Kirchhoff's Voltage Law

As you make a loop around a circuit, the sum of the voltage increase must equal the sum of the voltage decrease.

The negative end of the battery is connected to "ground" (it has zero voltage). We can then draw a diagram with the voltages (That symbol in the lower right represents a connection to ground).

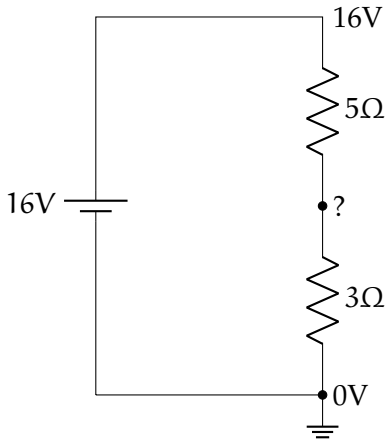


Exercise 15 Resistors In Series

Working Space

What is the current going around the circuit?

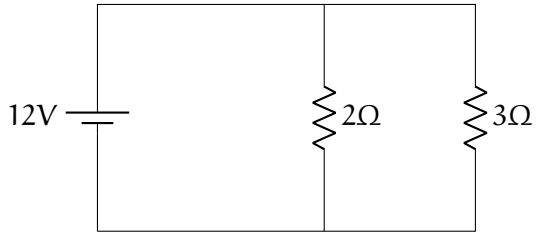
What is the voltage drop across each resistor?



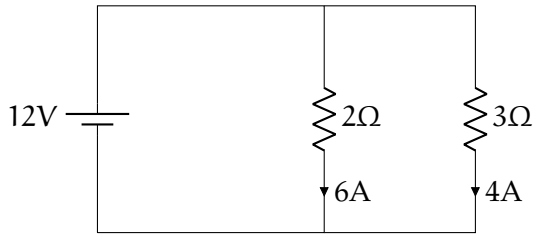
Answer on Page 66

5.2 Resistors in Parallel

Observe the following circuit. Note that the current can go two different paths.



There is 12 volts pushing current through both resistors. So 6A will go through the 2Ω resistor and 4A will go through the 3Ω resistor. Note that even though there is a path of least resistance (2Ω), the current is still devided evenly among both branches.



Thus, a total of 10 A will be going through the battery.

Imagine you are a battery. You can't see that you have two resistors. What does it feel like to you? $\frac{V}{I} = R$, and $V = 12$ and $I = 10$. This means the effective resistance of the two resistors in parallel is $\frac{12}{10}$ or $\frac{6}{5}\Omega$.

Resistance in Parallel

If you have several resistances $R_1, R_2, \ldots, R_n$ wired in parallel, their effective resistance R_t is given by

$$\frac{1}{R_t} = \frac{1}{R_1} + \frac{1}{R_2} + \ldots + \frac{1}{R_n}$$

In our example:

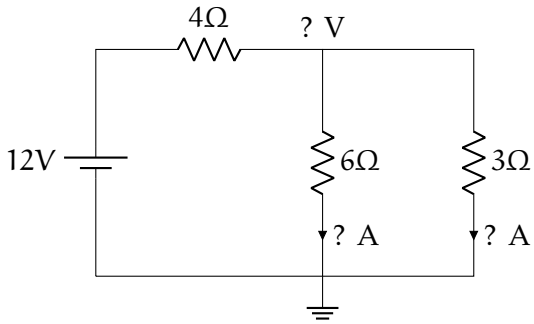
$$\frac{1}{R_t} = \frac{1}{2} + \frac{1}{3} = \frac{5}{6}$$

Thus, $R_t = \frac{6}{5}\Omega$.

Exercise 16 Resistors In Parallel

Working Space

What is the current going through the battery? What is the drop over the 4Ω resistor? What is the current in each branch?



Answer on Page 66

5.3 Kirchhoff's Current Law

States that all currents coming into a junction (or node) must equal the currents leaving that junction. Why? Because first of all, energy is always conserved, meaning it cannot be lost or destroyed within the equations in the first place. Secondly, each circuit is conservative, meaning it cannot lose voltage, and thus, cannot lose current. As you go around a loop, the starting point must have the same chargable amount as it did when you began, so any increases and decreases along the loop have to cancel out for a total change of zero.

Kirchhoff's Current Law

The sum of current into a junction equals the sum of current out of the junction.

Appendix A

Answers to Exercises

Answer to Exercise 1 (on page 6)

$$\mu = \frac{1}{6}(87 + 91 + 98 + 65 + 87 + 100) = 88$$

Answer to Exercise 2 (on page 8)

The mean of your grades is 88.

The variance, then, is

$$\sigma^2 = \frac{1}{6}\left((87-88)^2 + (91-88)^2 + (98-88)^2 + (61-88)^2 + (87-88)^2 + (100-88)^2\right) = \frac{784}{6} = 65\frac{1}{3}$$

The standard deviation is the square root of that: $\sigma = 8.083$ points.

Answer to Exercise 3 (on page 9)

In order the grades are 65, 87, 87, 91, 98, 100. The middle two are 87 and 91. The mean of those is 89. (Speed trick: The mean of two numbers is the number that is halfway between.)

Answer to Exercise 4 (on page 22)

The formula for the RMS is "=SQRT(SUMSQ(A2:A1001)/COUNT(A2:A1001))".

Answer to Exercise 5 (on page 33)

Variable	Value	Data Type
a	10	`int`
b	3.5	`float`
c	”10”	`str`
d	True	`bool`
e	[1, 2, 3]	`list`

Answer to Exercise 6 (on page 34)

```
False
```

Answer to Exercise 7 (on page 35)

Line Executed	n (value, type)	m (value, type)
After line 2	(5, int)	(”3”, str)
After line 3	(6, int)	(”3”, str)
After line 4	(6, int)	(”31”, str)

Answer to Exercise 8 (on page 37)

The code will raise a `TypeError` because `x` is a string and cannot be added to the integer `5`. The error message will be similar to:

```
TypeError: can only concatenate str (not "int") to str
```

To fix this, we need to cast `x` to an integer using `int()`:

```
x = int(input("Enter a number: "))
print(x + 5)
```

Answer to Exercise 9 (on page 41)

Input	Output
5	Small
10	Ten
15	Large

Answer to Exercise 10 (on page 41)

```
for i in range(0, 21, 2):
    print(i)
```

Answer to Exercise 11 (on page 44)

The start index is omitted and the end index is omitted, which means they are 0 and `len(seq)` $= 4$ respectively. Since the bounds are 0 and $4 - 1 = 3$, every index is included, in the given order. Therefore, `seq[:]` returns a copy of the given string, `[a, f, j, k]`

Answer to Exercise 12 (on page 45)

```
c
m
r
t
put
```

Answer to Exercise 13 (on page 47)

Line Execution	`items` content
After Line 1	`items = ["pen", "book", "eraser"]`
After Line 3	`items = ["pen", "book", "eraser", "ruler"]`
After Line 4	`items = ["pen", "book", "eraser"]`
After Line 5	`items = ["pen", "eraser"]`
After Line 6	`items = ["pen", "eraser", "marker"]`

Answer to Exercise 14 (on page 57)

$V = IR$ so $I = \frac{V}{R} = \frac{24V}{6\Omega} = 4A$.

Answer to Exercise ?? (on page 59)

There is a total resistance of 8Ω, so your 16V will push 2A of current around the circuit.

2A going through a 5Ω resistor represents a 10V drop.

2A going through a 3Ω resitor represents a 6V drop.

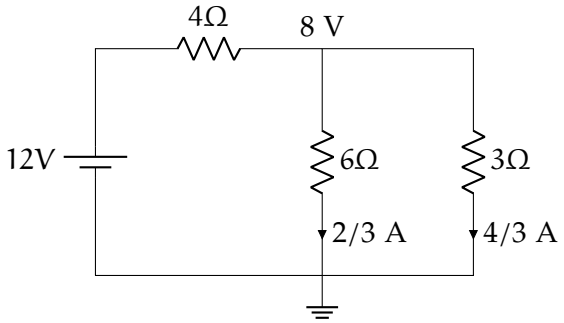
Answer to Exercise 16 (on page 61)

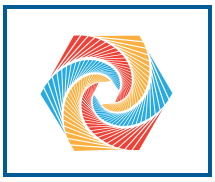
The effective resistance of the 6Ω and the 3Ω is 2Ω because

$$\frac{1}{R_T} = \frac{1}{6} + \frac{1}{3} == \frac{1}{2}$$

This means the battery experiences a resistance of $4\Omega + 2\Omega = 6\Omega$. A 12V will push 2A through a resistance of 6Ω.

The voltage drop across the 4Ω resistor is $2A \times 4\Omega = 8V$. Thus there will be a 4V drop across the two resistors in parallel. So 2/3 A will flow through the 6Ω resistor. 4/3 A will flow through the 3Ω resistor.





Index