To Mom & Dad

# Table of Contents

Once upon a time you opened this book

Before we start working our brain muscles, let's have a little chat. First of all, I'm so proud of you for taking up something new to learn! Or maybe it isn't new and you just want to read this book. That's cool too! Either way, if you come across something that confuses you or makes you frustrated don't worry! This can be difficult to understand. If it helps go for a walk, play a game on your nintendo switch, eat a cookie, or take a nap!

Everyone's brain works differently, and that means that not everyone is going to automatically understand a certain topic. Don't give up though! Don't let anyone tell you that you don't have the capacity of not understanding this, especially yourself! The hardest part of the entire process is to get to work! Wait...hey! It looks like you already did it! You're on your way to success! I'm so proud of you.

You might not care about my experience, so if you want you can just skip this part! However, I think my experience may help you go into this book with some more confidence. I started coding during 9th grade in my school's coding class. My friend and I were the only two girls in a room full of older boys who already had experience in coding. To be honest, I felt pretty defeated and was sure that I wouldn't be able to get a good grade. As you may be guessing, that didn't happen. After I stopped making excuses for myself for not understanding certain concepts, I told myself, "No one gets anywhere by feeling sorry for themselves and making up silly excuses! Get to work." These complex concepts weren't magically implemented in my brain; I had to use my time to study. I practiced. I asked questions. I watched videos. I learned how to code. After working really hard, I felt confident in my ability to code and, you guessed it, got an A! I promise you that I don't mean to brag. I'm just telling you this to let you know that, like many, I didn't have a lick of talent in coding. Talent is overrated! With hardwork and time, you can always reach your goals (this doesn't just apply to coding).

Lets get started...


# What is Computer Science?

In case you didn't read the book cover or get an idea of what this book is about before purchasing, it's about computer science. So, the obvious question we should answer first is...what is computer science? Well, in simple terms, it is basically learning how to communicate and work with computers. Just like us humans all speak different languages like Chinese, Hebru, English, or Spanish, computers speak different languages that humans do. Have you ever heard of computer languages like java or python? We'll learn more about those later, but those are some of the names in which computers "talk" in. Computer science is used in:

- Technology helping rovers explore Mars
- Data systems keeping us safe in the airport by warning us of potential weapons
- Animations in movies like Frozen!

Now that we have a better understanding of what computer science is, let's talk about why it's important.

Computer science is very relevant in our modern world. It's...everywhere!

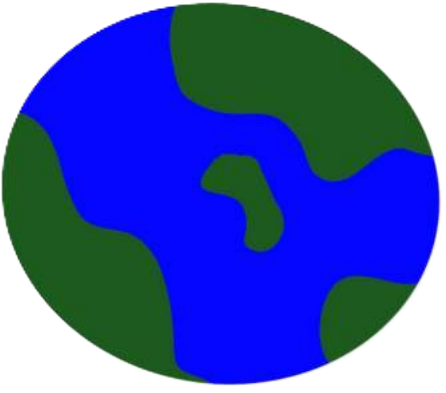
It's implemented in medical software and equipment. For example, it is involved in diagnostic tools such as MRI machines. It also helps us mere humans keep track of patient records which allows us to have a more efficient and fast healthcare service.

It is used in our everyday devices as well! For instance, when you open up your phone to search something like what Justin Beiber's birthday is, the results that pop up are because of many computer algorithms that are powered by computer science. When you open up the gaming apps on your phone like minecraft or candy crush, you're utilizing a game that is built by computer science. Cool, right?


Computer science also contributes to bettering the stability of our community. For example, it creates many job opportunities for the public including IT support and software developers. It helps the wellbeing of our economy!

It protects us! You know those security systems in jewelry stores or maybe in your house? Well, those are built with some help from our trusty friend, computer science. Most technologies that are built for emergency preparedness or management work because of computer science. They are the brains behind these "computer bodyguards".

In today's world, we're affected more and more everyday by climate change. Thankfully we have computer science at our disposal to study and keep track of environmental changes and data. It can also help us come up with remedies to reduce pollution and support a healthier environment.



I love writing poems! Get ready for a lot more hehe...

## Poem Time

While our world is filled with bright screens,
Computer science can help fulfill dreams.
It can remedy problems all over the land,
With just some code and a hand.

From coding to creating cool apps,
It betters our world with only taps.
Innovations galore,
Opportunities more,
Ready for kids and adults - no gaps!

Although we've explained different ways we utilize computer science, sometimes we can still get confused on what is and isn't computer science.

Here are some scenarios. Try and see if you can determine what is computer science and what isn't. Answers are at the bottom.

psst... fill out the answers


- Coding an app about classifying shapes  _____
- Watching a video you searched up on google 

_____
- Writing a program that tell you what to wear for the day  _____
- Playing Minecraft 

_____
- Solving a simple math problem by writing out code 2+2=4 _____
- Liking your friend's instagram post  _____

Answers: Is, Isn't, Is, Isn't, Is, Isn't

A common misconception some have is that they think that just because they are using something that already utilizes computer science means that they are practicing computer science themselves. This is wrong. To be doing computer science requires a lot more than just twiddling your thumbs to move a character in a game. Computer science requires you to create algorithms and methods to solve a problem using code.

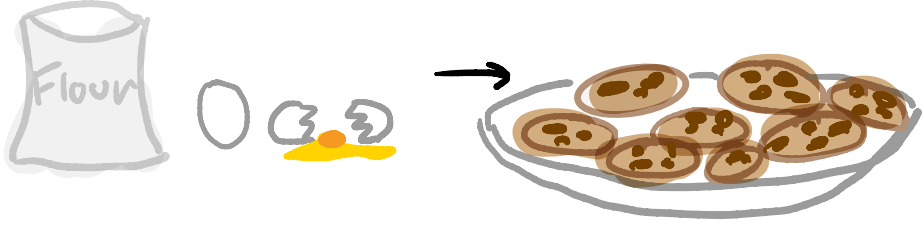
Let's go over the Big 6. You have to know these before we go any further. Don't worry though! This is low stress. If you're a little confused, that's totally normal!

## Coding

Coding is just like writing in a complex language that only computers can understand. When you code, you basically are giving the computer a set of instructions to follow. YOU tell it what to do, like telling a robot what to do. You can tell a computer to draw a certain animation, play a song, or solve a math problem. In a way, it's like creating special spells to cast on a computer! Of course, different spells call for different coding languages. We'll learn more about this throughout the whole book.

# Algorithm

An algorithm is really just a posh word for a series of steps you follow to complete a task or solve a problem. For instance, there is an algorithm for baking cookies. First you mix the butter, sugar, and eggs. Then, you add the dry ingredients like flour and baking soda. After that you make little balls of dough to stuff into the oven to bake. Boom! The cookies are done.

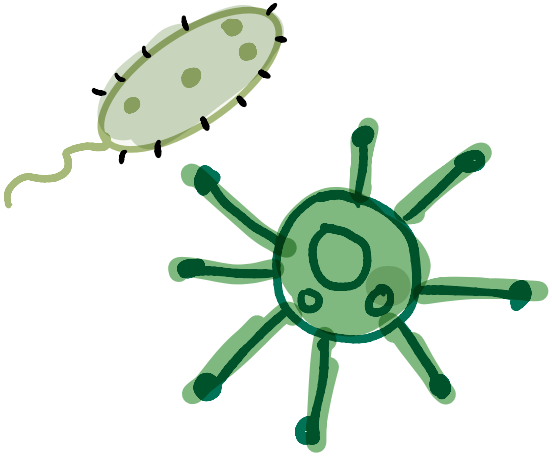



# Internet

With the current status of the world, many people find it hard to see life without the internet. Think of it as this vast spider web that connects every computer in the world together. Through this web, we're able to send videos, memes, messages, and voice memos to our friends and family. It's also how we're able to watch youtube videos, watch tiktoks, and search up information for a research project! It's like a magical library where we can find anything we want and communicate with people out of reach. We take it for granted a lot. A lot of people would go crazy if it ever shut down, even for a day!

# Cyber Security

You may have heard many people say the world isn't a safe place. Well, neither is the internet. A lot of people experience cyberbullying and online danger because of unawareness. Cyber safety is like an online savior that protects users online. When you go online, be sure to not share personal information - like your address, name, age, and birthday - without careful thought. It's important to ask an adult what to do if you're doing something you're not familiar with on the web, such as downloading stuff or clicking links. Not everything is what it seems like on the internet. Some buttons and links allow viruses to gain access to your computer! Be careful!



## What's a virus? Q&A

A computer virus isn't an actual disease where the computer is sick. It's a nasty program that can get into your computer and start causing a lot of problems. It can spread from computer to computer which is similar to human illnesses. If it gains access to your computer, it can make things stop working correctly, it can ravage through your personal information, and it can get rid of important information that you keep stored on your computer. Sometimes downloading an antivirus program can help prevent your computer from "getting sick". If you'd like to do this, ask an adult to help you!

# Data

Data is the information that computers use to complete various tasks. Imagine you have a ginormous box of LEGO pieces. Each piece represents a piece of data like a word, picture, or a number. When you stack these pieces together, you can make something really cool like a boat or an airplane. Computers do a similar thing where they use data to make things like websites, apps, and games. For example, if a program is prompted to ask you how the weather is, you might answer that it is either cold or hot; of course, your answer needs to be remembered, so it's stored somewhere as data. Data helps computers know what to do and how to do it. It is fundamental to the use of computers!

# Storage

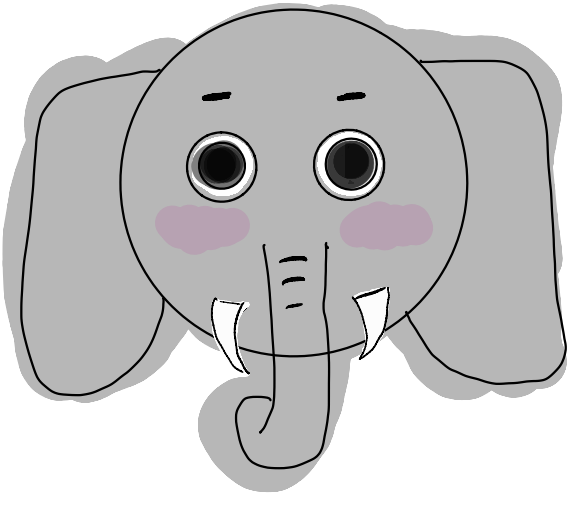
There is no trick or complex meaning behind this word. You may be able to understand this concept without me explaining it to you, but let's go over it just in case.

Storage in computer science is where your computer stores all the information it needs to remember like files, documents, or even previous text messages. It's similar to a closet, but instead of being filled with clothes, games, and sports equipment, it's filled with files and data.

## Two Main Types of Storage:

Storage (RAM): Think of this a white board used in class. It's used to write down things temporarily. At the end of class, the teacher will erase everything on the board, just like how certain information can be completely erased when you shut off your computer.


Permanent Storage (Hard Drive/SSD): Think of a sentimental box where you keep small trinkets that are important to you. When you open the box, everything you left in there hasn't gone anywhere, and when you close the box, everything will still be there. When you turn off your computer, some information remains stored and doesn't get erased.



Storage is vital since it helps computers remember things, so you can come back later to find all your saved work, pictures, texts, and even the place you left off in your Minecraft world!


In the heart of a computer's core
Lies storage where data is stored
Bits and bytes arranged just right,
Keeping memories day and night.

Coding spells in languages across the land,
Demands the machine what to command.
From games we play to apps we like,
Coding brings them all to life.

Algorithms draw the course,
Solving problems with clean force.
Step by step, they lead the way.
Turning chaos into day.

Cyber security stand guard,
Protecting data, working hard.
Shoo malicious eyes that try to peak,
To steal the info that we keep.

The internet, a big expanse,
Connects us all in a global dance.
Sharing knowledge, life, and more,
Bringing stories to every door.

Data flows through every stream,
Fueling every digital dream.
In every area, big or small,
Data drives the tech for all.

Unit 2

# What is a Computer?

# What's a computer?

To put it in simple terms, a computer is a machine that helps us solve different issues, communicate, and create. Similar to a human's brain, a computer completes tasks we ask it to do. For example, if we're in a library, we use our brain to find the correct shelf, then find the corresponding book we want on that shelf. Boom! Task accomplished. With a computer, we can just search up the book on the web and it pops up right on our screen. Boom! Task accomplished.

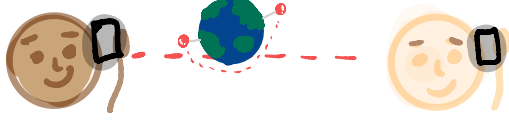
If you're still having trouble grasping this, you can use one of the following analogies!

Think of a computer as a versatile toolbox, filled with multiple specialized tools. Each tool serves a unique purpose. For instance, if you need to draft a letter, the word processor acts like the pencil and paper in your toolbox. When you want to enjoy some music, the music player functions like a radio. What makes the computer exceptional is that it gathers all these tools in one convenient place, streamlining your tasks and making your work more efficient.

Consider your brain your computer. You have input from your eyes and your ears. This translates into what we call sight and hearing. Once these inputs are in your brain, it can process the input in a manner to allow your body to perform. Your body is the one that takes action! This works in a similar way to a computer. When you input something, such as typing on your keyboard, the computer processes the information using its CPU (the brain of a computer) and then produces the results by displaying them on the screen.

## Everyday use of computers

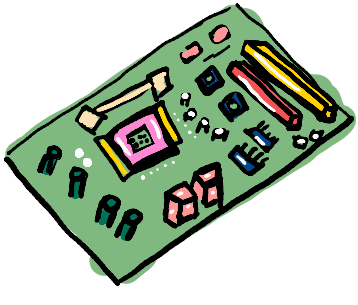
Computers are everywhere, and they have a huge role in how we live, work, and play. Computers enable us to do things more quickly and efficiently, from gaining access to information to keeping in touch with people all over the world.

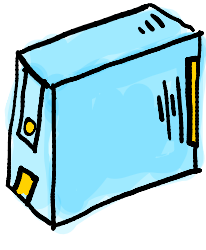


Computers allow us to email co-workers, chat with friends, and video call family living in other countries. They also make it easy to share your experiences with others around the world. With a computer and an Internet connection, school curriculums are available online, and a huge number of children can access more information. They help us learn new things, do homework, even create projects for school.

# What is software & hardware?

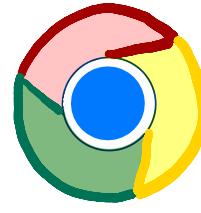
Okay. Let's start off with the easy one: hardware. Hardware are the physical parts of your computer. If you can touch it, it's hardware. For example, if you pull out a computer, you could touch the mouse, the screen, and the keyboard. These are all pieces of hardware. However, hardware also includes the internal parts of a computer, for example, the motherboard and the CPU (central processing unit). These are also considered hardware even though you may not be able to touch it instantly on your computer. You'd have to use a bunch of tools to dissect your computer until you can finally hold it in your hand physically.





Software on the other hand is not physical. It's a list of instructions that tell the computer what to do. This includes web browsers, video games, word processors, and other operating systems. Without software, the applications and programs on your computer wouldn't run. It's the mastermind behind the hardware!

Software is usually kept away in a computer's hard drive. When you start a program, this software is transferred to the computer's RAM (Random Access Memory). Here, it can be quickly handled and carried out by the CPU.

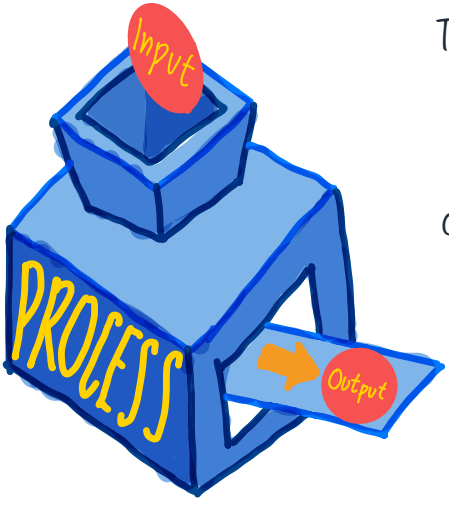


In general try to remember that the hardware is like the body of a person while the software is the brain. For the sake of understanding, pretend it's impossible to touch the brain!

## Input & Output

In computer science terms, input is the data that is used for computer system processing. Hardware allows the user to send data or certain signals that the computer can act on. Whether you type a letter, click the mouse, or swipe on a touchscreen, you are sending information to the computer, which can be referred to as input devices.

Output is the next thing; it is the outcome of processing input. The processed information is then sent back to the user whether it's through a screen, printer, or even earphones.




This is why software AND hardware are both important. They make it easier for us fellow humans to interact with computers. Without one or the other communication could be one sided or there simply might be no way to communicate with computers.

## Examples

Typing the word "dog" on the keyboard -> the word "dog" appears on the screen


Pressing the on button on a computer -> the computer turns on

Clicking on the safari icon on a computer screen -> safari browser pops up 


Tapping on the Candy Crush app on your phone -> candy crush app opens up 

Okay. Now that all the boring stuff is out of the way let's hop into some coding.

Great job! The boring part is over! Now for the fun part.

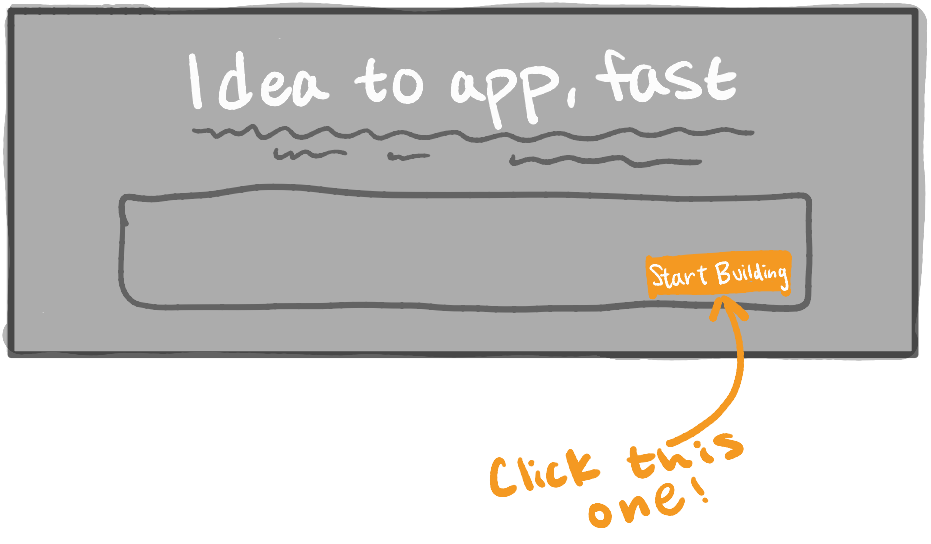

# Your First File

# Get your first file

There are many different platforms that allow you to code ranging from Scratch to Visual Studio Code. However, since we're just beginners we're going to start on a platform called Replit. Replit and its interface are trademarks of Replit, Inc. This book is not affiliated with or endorsed by Replit, Inc.

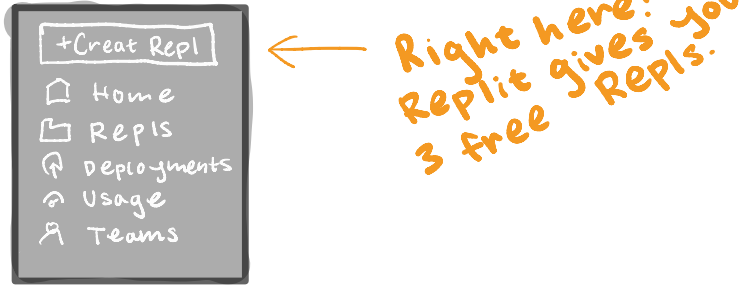
First, find a computer at the library or a laptop at home and search up and type what's below into the search bar of an available search engine like Safari or Google.


In the middle of the screen, you should now see an orange button that says to sign up for free. Click it!

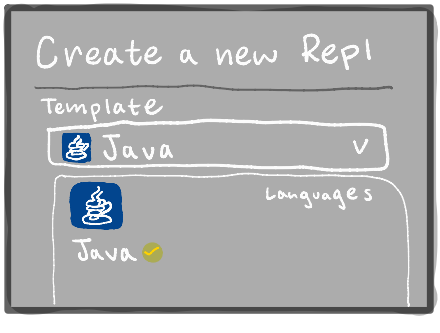


Next, ask a parent to help you put in an email and password to create the account! If you don't require parental permission, put in an email and password of your choice! After this, you should be able to see Replit start to get ready for you. It may ask you a couple of questions regarding your name or coding level; however you'd like to fill that out is up to you!

After filling out any questions Replit may ask you, you're ready to start! Look at the top left corner of your screen. You should be able to see a button that says, "+ Create Repl" Click on it!



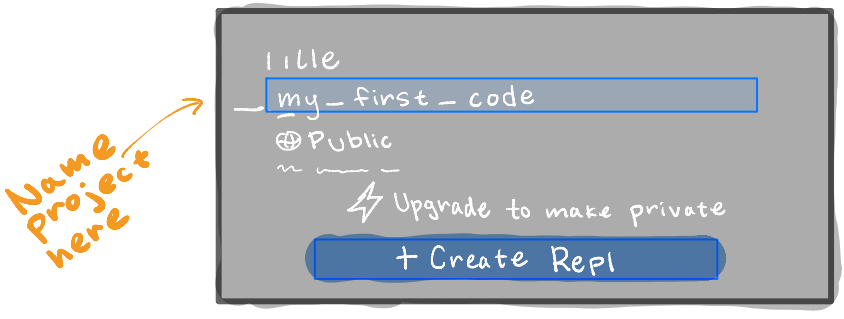

Now it asks you to choose a template. This is basically asking what language you'd like to code in. Right now we're going to start with java so type that in! Now click the icon that has a little coffee cup on it.




**Note:**

There are many different versions of java. For now, we're going to stick to the basics and choose JUST "Java" with no add-ons. Below is a reference to make sure you choose the right one!

Once you've clicked the language you want to code in, you can title your project anything you'd like. I suggest to title it "my_first_code," but it is completely optional. If you want you could title it "ilovecookies" or "lakersarethebest." However, when titling your projects, you can't have spaces! To mimic a space, coders usually use an underscore.




Now that you've chosen your language and title, go ahead and click the create button.

I know what you're thinking! This may look really difficult and confusing, but hang in there. Let's go over each line one by one. By the way, the lines that are green and have '//' at the beginning of them are called comments. These lines don't run any code. There kinda like little notes you can write while you're coding. In fact, you can just delete them!

```
1
2  public class Main {
3    public static void main(String[] args) {
4      System.out.println("Hello World!");
5    }
6
7  }
```

```
public class Main {
```

This line defines a class named Main. In Java, everything starts with a class, which is like a blueprint for creating objects. Don't worry if you don't understand what objects are; we'll get to that later.

```
public static void main(String[] args){
```

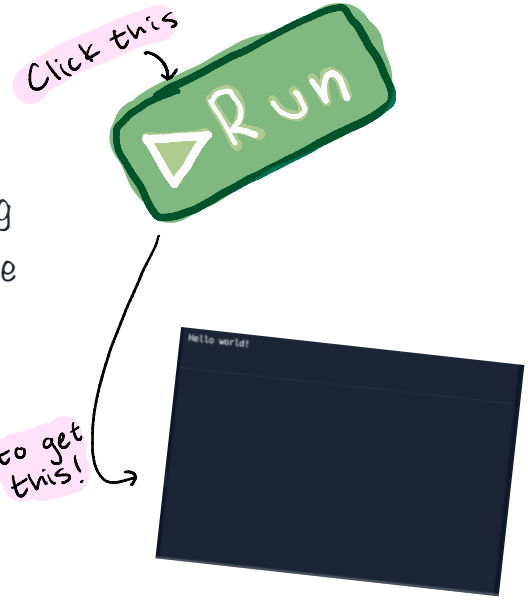
This is the main method. It's a special method where the program starts running. We'll go over all those crazy word like "void" and "static" later on.

```
System.out.println("Hello world");
```

This last line is called a print statement! This prints stuff to the console for everyone to see!

## In action

Go ahead and run the code by clicking the big green button at the top and see what happens! Whatever is inside the print statement should print to the console.




The right box on your screen

# Strings

Before we continue, let's go over what a string is. A string in programming is a type of data used to store text. Think of it as a sequence of characters (letters, numbers, symbols) that are put together to form words, sentences, or even whole paragraphs. In java, strings are written between quotation marks (" ").

"I love listening to Billie Eilish." is a string that contains text.
"12345" is also a string, even though it looks like numbers. Since it's in quotes, it's treated as text, not as actual numbers you can do math with.


Now you give it a shot! Inside the print statement delete the statement in between the quotation marks and type something you'd like to see print to the console! It could be a word, a sentence, or even a whole essay! Sky's the limit. Look at the example below if you need some help.

Now let's learn about how to create a new line with print statements.

When you print something in programming, all the text usually appears on the same line. But what if you want to split it into multiple lines? For example, instead of printing:


You might want it to look like this:

To create a new line in a print statement, you can use escape sequences. In Java, the most common escape sequence for a new line is \n.

### Escape Sequence

An escape sequence is a combination of characters that represents a special character or behavior in a string. In programming, escape sequences allow you to include characters in a string that would otherwise be hard to type directly, such as new lines, tabs, or even quotation marks.


Here's how you can use it:

The \n tells the computer to move the text after it to a new line. When you run this code, you'll see the output split into two lines, like this:


This is so helpful when you want to organize the output to make it more readable. You can even add more \n to create multiple blank lines in between!

## A simpler way

Java actually provides an even simpler way to automatically move to the next line after printing something: by using println()instead of print().

Here's the difference:

System.out.print("Hello world!"); will print the text but stay on the same line.


System.out.println("Hello world!"); will print the text and then move the cursor to the next line, ready for the next output.

Isn't it cool how each println() automatically starts a new line after printing the message?! You don't have to add \n—it does it for you! So, if you want each message on its own line, using println() is an easy way to do it.

## Ending a line of code

Remember that unless a line ends in a bracket, put a semicolon at the end! Without it your code won't run. Let's do some practice now!

1. Print a sentence that says what your favorite pizza flavor is.
2. Print your name, then make a new line that prints your age.
3. Print your favorite flower, 5 blank lines after, then why you love that flower so much.

## Answer examples

1

2


3


Did you get them right? There's different ways to answer these prompts, too! For example, instead of using 'System.out.println("");', you can use \n inside the brackets! The cool thing about code is that there are so many different ways to solve the same problem. You get to explore your creativity! Keep going. You got this!


In code we write, so bold and bright,
With print we share our text in sight.
println() comes with a graceful flair,
Moving to the next line, leaving space to spare.

Escape with \n, create new lines,
Formatting output with perfect designs.
Print your words, let them flow,
In the world of code, watch them grow!

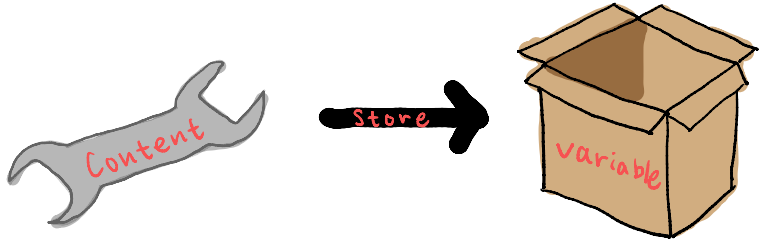
Debug, express, or simply play,
With print statements leading the way!

Good job! You're making such good progress. Now you're ready to move onto variables!


# Variables

# What are variables?

Think of a variable like a container or a storage box. When you have something you want to keep track of—whether it's a number, some text, or other information—you can "store" it in a variable, just like you'd put things in a box and label it. This way, you can come back and use or update that information easily. For example, you might have a box for storing someone's name or their age, and you label those boxes so you know what's inside.



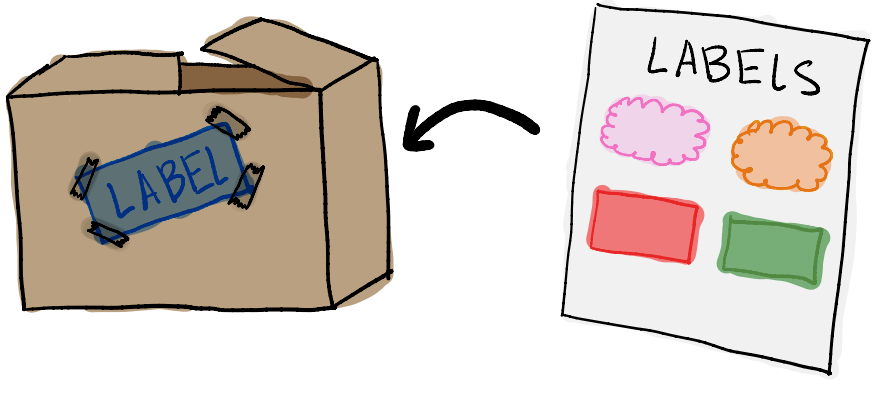

# Why use variables?

Let's say you're writing a program, and you need to remember things like a player's name, their age, or their score in a game. Instead of writing these details over and over throughout your code, you can store them in variables. Think of the variable as the label on your box—whenever you need that information, you just call the variable by name. It makes your code much simpler to read and quicker to change later.

# Creating a variable

In Java, creating a variable means you:

1. Decide what kind of data you want to store. Is it a name? A number Java has different types of boxes for different kinds of information.
2. Choose a name for your variable. This is like the label you put on the box, so you can remember what's inside.




In Java, there are different kinds of variables for storing different types of data. Just like you use different-sized boxes for different items, Java has specific types of variables for numbers, text, true/false values, and more. Let's go over the most common ones on the next page.

## String

A String is used to store text—anything inside quotation marks.

## Integer

int is short for "integer" and is used for storing whole numbers (no decimals).

## Double

double is used to store numbers with decimals, like 3.14 or 5.99.

## Boolean

A boolean variable can only store two values: true or false. It's great for decisions, like whether something is on or off.

## Character

A char (short for "character") stores a single letter, digit, or symbol.

Note: You use single quotes (' ') for characters, unlike the double quotes for strings.

# Float

Similar to double, but float stores smaller decimal numbers. It's less precise than double, but it uses less memory.

Variables make your code flexible and reusable. You can change the value inside the variable without having to rewrite the code everywhere! Below you can see an example on how to make variables.

```
public class Main {
  public static void main(String[] args) {
   String name = "Alice";
   int age = 10;
  }

  // @Test
  // void addition() {
  //     assertEquals(2, 1 + 1);
  // }
```

The the text Alice is stored in the String variable -> name

The number 10 is stored in the integer variable -> age

## Utilizing variables example

# Recap of Variable Types

String: Stores text, like names or words ("Hello").
int: Stores whole numbers, like age or count (10).
double: Stores numbers with decimals, like prices (19.99).
boolean: Stores true/false values (true or false).
char: Stores a single character ('A').
float: Stores smaller decimal numbers (98.6f).


floats can have 'f' at the end to denote it's a float

By choosing the right type of variable, you can store data efficiently and ensure your program runs smoothly!


In Java's realm, where code takes flight,
Variables shine, each one a delight.
String holds the words, so rich and so bright,
While int counts the ages, with numbers in sight.

Double brings decimals, precise as can be,
And float dances lightly, with less memory.
Boolean decides, true or false in the fray,
Char holds a letter, just one on display.

Together they work, a colorful crew,
Organizing data, making sense anew.
In the world of code, they play their part,
Variables unite, a programmer's art!


# Working with Variables

# Introduction to variables in Action

Let's do an introduction to variables. Now knowing what variables are and how to create them, it's finally time to see how they can put your code in action. Let's go over how to declare, initialize, and use variables. We're going to see how they can make your programs more exciting and engaging.

## Declaring & Initializing Variables

When you declare a variable, you're creating a little space in your program to store some information. Think of it like labeling a box where you keep important things. To initialize a variable means you're putting something inside that box for the first time. Let's look at a few examples:

```
2   public class Main {
3     public static void main(String[] args) {
4       String favoriteFood = "Pizza";
5       int numberOfSlices = 3;
6       double pizzaPrice = 9.99;
7
8     }
9
10  }
```

Here, we have three variables that store different kinds of information. We've labeled one box with favoriteFood, and inside it, we've placed the value "Pizza".

## Using Variables in Calculations

Variables are powerful tools when it comes to doing math in your code. You can use numerical variables to perform calculations that adjust based on the data you have. For instance:

```
2   public class Main {
3     public static void main(String[] args) {
4       String favoriteFood = "Pizza";
5       int numberOfSlices = 3;
6       double pizzaPrice = 9.99;
7
8       int totaSlices = numberOfSlices *2;
9       double totalPrize = pizzaPrice *2;
10
11    }
12
13  }
```

In this example, we're using the variables we created earlier to calculate the total slices and the total price for two pizzas. It's a cool way to make your code dynamic and responsive!

# Changing Variable Values

The best part about variables is that they aren't set in stone. You can change their values whenever you need to, allowing your program to adapt to new information. Here's how you can update variable values:

```
1
2   public class Main {
3     public static void main(String[] args) {
4       String favoriteFood = "Pizza";
5       int numberOfSlices = 3;
6       double pizzaPrice = 9.99;
7
8       int totalSlices = numberOfSlices * 2;
9       double totalPrice = pizzaPrice * 2;
10    }
11
12  }
```

used to be 3

used to be 9.99

After these updates, if you run your calculations again, you'll see different results based on the new values. It's like rearranging th e items in your box whenever you get something new! Let's print the before and after values to see the difference!

Before

```
1
2   public class Main {
3     public static void main(String[] args) {
4       String favoriteFood = "Pizza";
5       int numberOfSlices = 3;
6       double pizzaPrice = 9.99;
7
8       int totalSlices = numberOfSlices * 2;
9       double totalPrice = pizzaPrice * 2;
10
11      System.out.println(totalSlices);
12      System.out.println(totalPrice);
13
14    }
15
16  }
```

Run

6
19.98

## After


```java
public class Main {
  public static void main(String[] args) {
    String favoriteFood = "Pizza";
    int numberOfSlices = 5;
    double pizzaPrice = 12.99;

    int totalSlices = numberOfSlices * 2;
    double totalPrice = pizzaPrice * 2;

    System.out.println(totalSlices);
    System.out.println(totalPrice);

  }
}
```

See! After changing our variables, the output to the console was different. Since numberOfSlices and pizzaPrice had their values changed, the values of totalSlices and totalPrice changed! As a result, after the update, instead of '6' and '19.98' getting printed to the console, '10' and '25.98' got printed.

## Combining Variables in Output

Variables can make your output much more interesting. By combining text and variable values in your print statements, you can create messages that feel personal and engaging. Look at the next page for an example!

# Example

```java
public class Main {
  public static void main(String[] args) {
    String favoriteFood = "Pizza";
    int numberOfSlices = 3;
    double pizzaPrice = 9.99;

    int totalSlices = numberOfSlices * 2;
    double totalPrice = pizzaPrice * 2;

    System.out.println("I ordered " + numberOfSlices + " slices of "
+ favoriteFood + " for $" + pizzaPrice + " each.");

  }
}
```

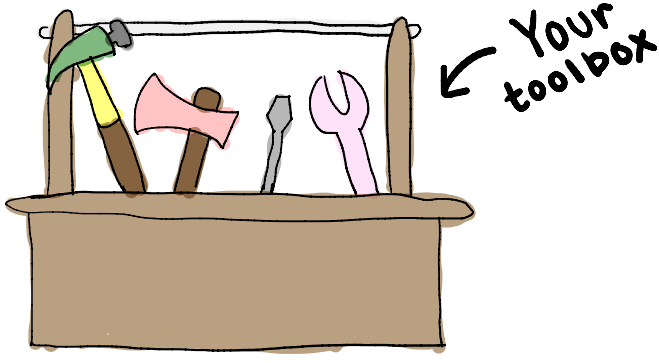
Run  Ask AI

```
I ordered 3 slices of Pizza for $9.99 each.
```

This line puts together a sentence that includes the values stored in your variables, giving you a clear and personalized message.


Don't worry if you're confused on this print statement. The next chapter will go over what these '+' signs mean and how to use them. However, this line of code should make sense intuitively. Essentially, we are printing a statement that combines both Strings and variables. To combine these, we are using the '+' sign.

Variables are a fundamental concept in programming, allowing you to store, manipulate, and display information dynamically. As you continue your coding journey, keep experimenting with variables to see how they can enhance your programs. Whether you're calculating scores, storing user input, or making decisions, variables will always play a crucial role in your coding toolbox.




Let's do some practice with a quick project!

1. Make a String variable called favorite_food and assign it to whatever dish you'd like!

   For example:

   String favorite_food = "Spaghetti";

2. Create an int variable called money and assign it any number.

   For example:

   money = 3;

Now create a boolean variable called hungry and assign it a true of false. True means you are hungry and false means that you're not.

For example:

boolean hungry = true;

Now print out a line of code with your variables! Look below for an example to see how to combine them!

## Example Print Statements

```
System.out.println("I want to order" + favorite_food + ", and I have
    around" + money + "dollars.");

System.out.println("Let's check the next statement to see if we
    should get" + favorite_food + "or not. \n I am hungry:" +
    hungry);
```

Good job! I know this seems really simple, but it's important to test out new things you learn! We're going to get into some more complex stuff and I won't be able to go line by line with you. However, there is always an answer key if you get stuck! The answer for this prompt is on the next page!

# Example Answer

## Code

```
1  public class Main {
2      public static void main(String[] args) {
3          // Declare and assign variables
4          String favorite_food = "Tacos";
5          int money = 5;
6          boolean hungry = true;
7
8          // Print out the combined lines using the variables
9          System.out.println("I want to order " + favorite_food + ",
   and I have around " + money + " dollars.");
10         System.out.println("Let's check the next statement to see if
   we should get " + favorite_food + " or not. \nI am hungry: " +
   hungry);
11     }
12 }
```

## Output / Console (right side of screen)

```
I want to order Tacos, and I have around 5 dollars.
Let's check the next statement to see if we should get Taco
s or not.
I am hungry: true
```

# Unit 6

# Operators

# Operators

In programming, operators are special symbols that perform operations on variables and values. Just like mathematical operators you encounter in math classes (like addition and subtraction), Java has its own set of operators for various tasks. In this chapter, we'll explore the different types of operators available in Java and how to use them effectively.

## Arithmetic Operators

Arithmetic operators are used to perform basic mathematical operations. Here's a quick overview:

| Operator | Description | Example |
|---|---|---|
| + | Addition | 5+3 → 8 |
| - | Subtraction | 5-2 → 3 |
| * | Multiplication | 5*3 → 15 |
| / | Division | 6/3 → 2 |
| % | Modulus | 5%2 → 1 |

remainder

Let's write a simple program that demonstrates arithmetic operations:

Delete or comment out any previous code you've written in your main method so we can start from scratch. To comment out something, place two slashes in front of a line (//). If you don't want to do either of these options, you can also start a new file in Replit!

This next project is going to help us practice how to use arithmetic operations in code! This is a fundamental piece, so make sure you really understand it.

## New Project Alert 

1. Make two int variables and assign any values you want to them.

2. Make a new 5 new int variables and title them the following: sum, difference, product, quotient, remainder.

3. When defining each variable, use the operations to give an accurate value to each variable. I'll give you the first two:

```
int sum = x + y;

int difference = x - y;
```

4. Once you've finished making your variables, print them out!
   Here's an examples:

System.out.println("Sun:" + sum);

System.out.println("Difference:" + difference);

# Example Answer

Code:

```
1  public class Main {
2      public static void main(String[] args) {
3          // Declare and assign variables
4          String favorite_food = "Tacos";
5          int money = 5;
6          boolean hungry = true;
7
8          // Print out the combined lines using the variables
9          System.out.println("I want to order " + favorite_food + ",
   and I have around " + money + " dollars.");
10         System.out.println("Let's check the next statement to see if
   we should get " + favorite_food + " or not. \nI am hungry: " +
   hungry);
11     }
12 }
```

Output:

```
I want to order Tacos, and I have around 5 dollars.
Let's check the next statement to see if we should get Taco
s or not.
I am hungry: true
```

Good job! Now you understand how to work with these arithmetic operations! Let's move onto Relational Operators now.

# Relational Operators

Relational operators are used to compare two values. They return a boolean result: true or false. Here's a list of common relational operators:

| Operator | Description | Example |
|---|---|---|
| == | Equal to | 5 == 5 → true |
| != | Not equal to | 5 != 3 → true |
| > | Greater than | 5 > 3 → true |
| < | Less than | 5 < 3 → false |
| >= | Greater than or equal to | 5 >= 5 → true |
| <= | Less than equal to | 5 <= 3 → false |

This next project will help us practice using relational operators in code! Understanding how to compare values is crucial for making decisions in your programs, so let's dive in!

# New Project Alert 

1. Make two integer variables and assign any values you want to them. You can name them x and y.

2. Next, create five new boolean variables that will store the results of the relational comparisons. Name them as follows: isEqual, isNotEqual, isGreater, isLess, isGreaterOrEqual

3. Now assign the variables to their according values. I'll give you the first two.

```
boolean isEqual = (x == y);

boolean isNotEqual = (x != y);
```

4. Once you've defined your boolean variables, print out each result to see the comparisons in action! Here's are some example of how to print the values:

```
System.out.println("Is x equal to y? " + isEqual);

System.out.println("Is x not equal to y? " + isNotEqual);
```

How'd you do? Check the answer example on the next page for help or to compare!

# Answer Example

Code:

```java
public class Main {
    public static void main(String[] args) {
        // Declare and assign values to two integer variables
        int x = 10;
        int y = 7;

        // Perform relational comparisons and assign results to boolean variables
        boolean isEqual = (x == y);
        boolean isNotEqual = (x != y);
        boolean isGreater = (x > y);
        boolean isLess = (x < y);
        boolean isGreaterOrEqual = (x >= y);

        // Print out the results of the comparisons
        System.out.println("Is x equal to y? " + isEqual);
        System.out.println("Is x not equal to y? " + isNotEqual);
        System.out.println("Is x greater than y? " + isGreater);
        System.out.println("Is x less than y? " + isLess);
        System.out.println("Is x greater than or equal to y? " + isGreaterOrEqual);
    }
}
```

Output:

```
Is x equal to y? false
Is x not equal to y? true
Is x greater than y? true
Is x less than y? false
Is x greater than or equal to y? true
```

# Logical Operators

Now we're going to go over Logical Operators! Just like in real life, where we often make decisions based on multiple conditions, programming allows us to do the same. Logical operators are the tools we use to combine different conditions and make our code smarter. Let's break down these operators so you can understand how to use them effectively!

## What are they?

Logical operators help us evaluate multiple conditions at once. In Java, there are three main logical operators you'll use:

## Logical AND

Think of the logical AND operator as the "team player" of the bunch. It only returns true when both conditions are true. If either condition is false, it's a no-go!

```
1  public class Main {
2    public static void main(String[] args) {
3      boolean isSunny = true;
4      boolean isWeekend = false;
5      boolean canGoOut = isSunny && isWeekend;
6    }
7  }
8
```

AND operator

In the case of the last page, you can only go out if it's both sunny and the weekend. If one of those conditions isn't met, you're stuck inside!

## Logical OR

Now, the logical OR operator is a bit more flexible. It returns true if at least one of the conditions is true. It's like saying, "If I can do this or that, I'm happy!"

Below, you're good to go if it's sunny or it's the weekend. So, even if one condition is false, you still have a reason to enjoy your day!

```
1  public class Main {
2    public static void main(String[] args) {
3      boolean isSunny = true;
4      boolean isWeekend = false;
5      boolean canGoOut = isSunny || isWeekend;
6    }
7  }
8
```

OR operator

Unlike the AND operator, the OR operator just needs one of the conditions to be true rather than both.

## Logical NOT

Finally, we have the logical NOT operator, which is like a switch that flips the truth. If something is true, ! makes it false, and vice versa.

```
1  public class Main {
2    public static void main(String[] args) {
3      boolean isRaining = false;
4      boolean isDry = !isRaining;
5    }
6  }
```

NOT operator

In this case, if it's not raining, then you're dry! It's a simple way to change the perspective of a condition. The NOT operator essentially is an opposite operator.

## Combining Operators

You can mix and match these logical operators to create more complex conditions. Just like putting together pieces of a puzzle, you can combine them to get the complete picture!

```java
public class Main {
  public static void main(String[] args) {
    boolean isSunny = true;
    boolean isWeekend = false;
    boolean canGoOut = isSunny && isWeekend;
    boolean goToMovies = isSunny || isWeekend;
  }
}
```

In this example, going out to enjoy the day requires both conditions to be true. However, you can still go to the movies if at least one condition holds!

## Short Circuit Evaluation

Java uses a clever technique called short-circuit evaluation. This means that when evaluating expressions, Java will skip checking the second condition if the first one already determines the outcome.

```java
public class Main {
  public static void main(String[] args) {
    boolean isSunny = true;
    boolean goToPark = isSunny && (5 / 0 > 1);
  }
}
```

On the page before, because isSunny is false, Java doesn't even bother checking the second condition. It knows the whole expression can't be true!

## Re-cap!

Logical operators are your best friends when it comes to decision-making in programming. They allow you to create more nuanced and flexible conditions, making your code smarter and more capable.

In our next project, we'll dive into some hands-on practice with logical operators, giving you a chance to put your new knowledge to work!

Unit 7

# If-Else Statements

# Making Decisions with If-Else

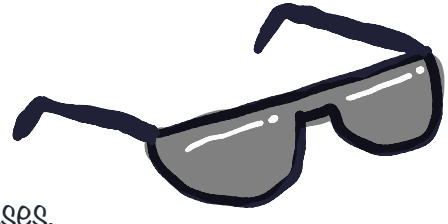
Now that we've learned about variables and logical operators, it's time to take things up a notch! Imagine you're writing a program where you need to make decisions—just like in real life, where you might choose what to wear based on the weather or decide whether to study based on how much time you have. In coding, we use if-else statements to make those decisions.

## What's an If-Else Statement?

An if-else statement is like asking a question in your code: "If this condition is true, do something; otherwise, do something else." It helps your program take different actions based on whether a condition is met or not.

Think of it like this:

- If it's sunny, you'll wear sunglasses.
- Else (if it's not sunny), you'll grab an umbrella.



```java
public class Main {
  public static void main(String[] args) {
    boolean isSunny = true;

    if(isSunny){
      System.out.println("It's sunny outside!");
    } else {
      System.out.println("It's not sunny outside!");
    }
  }
}
```

# Breaking it down

Let's look at what's happening in this code:

The condition: Inside the parentheses of if, you put a condition (in this case, isSunny). This is the question your code is asking.

The action: Inside the curly braces {}, you write the code that runs if the condition is true.

The else block: After the else, there's another set of curly braces. This is what happens if the condition is false.

# Why If-Else?

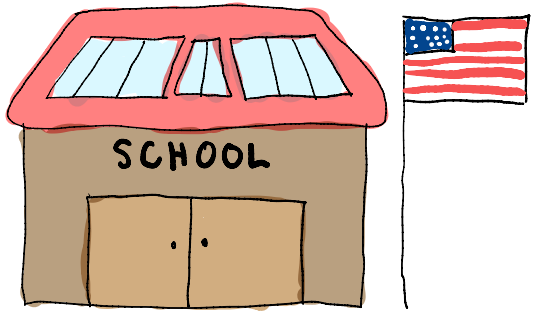
If-else statements are super useful when you need to make decisions or check conditions in your programs. For example:

If a player's score is above a certain number, they win the game.


If someone enters the correct password, they can log in; otherwise, they can't.

password

If it's a weekday, you go to school; else, you can relax.



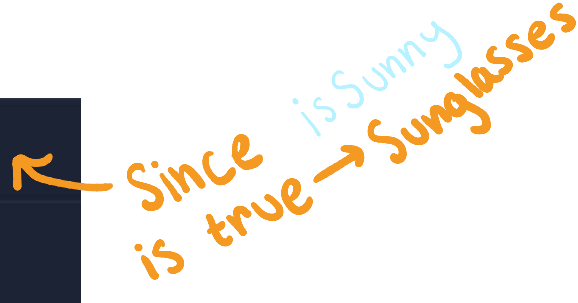
# More Options

Sometimes, you need more than just two options. Maybe it's not just about sunny or not sunny—what if it's cloudy, or what if you're not sure? In this case, you can use else if to add more conditions.

```java
public class Main {
  public static void main(String[] args) {
    boolean isSunny = true;
    boolean isCloudy = false;

    if(isSunny){
      System.out.println("Wear sunglasses");
    } else if (isCloudy) {
      System.out.println("It's not sunny outside!");
    } else {
      System.out.print("Stay inside.");
    }
  }
}
```

```
Wear sunglasses
```



In the example on the page before :

If it's sunny, you wear sunglasses.
If it's cloudy, you take a light jacket.
If it's neither sunny nor cloudy, you stay inside.

## New Project Alert 

Let's put this all together with a simple example in code. Write a program to decide whether someone passed or failed a test based on their score!

Let's go through some steps!

1. Create an int value variable that stores a test score

2. Create if statements that give a grade based off the score! To do this, create if statements that check to see if a score is of certain value, then what letter grade to give that score. I'll give you the first one.

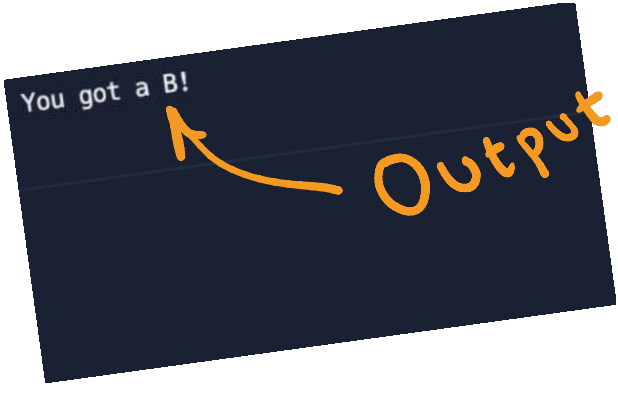
```
if (score >= 90) {
System.out.println("You got an A!");
}
```

# Answer-Key

```java
public class Main {
    public static void main(String[] args) {

        int score = 85;

        if (score >= 90) {
            System.out.println("You got an A!");
        } else if (score >= 80) {
            System.out.println("You got a B!");
        } else if (score >= 70) {
            System.out.println("You got a C!");
        } else if (score >= 60) {
            System.out.println("You got a D!");
        } else {
            System.out.println("You got an F!");
        }
    }
}
```



In this program, based on the score, the program decides what grade the student gets. Notice how the conditions are stacked to check different ranges of scores.

## Re-cap!

If-else statements are the decision-makers in your code. They allow your programs to react to different situations and choose the right actions. When you keep building more complex projects, you can see how valuable these simple structures can be for controlling the flow of your program. Trust me, I don't know where I would be without these functions.


# Lists and Arrays

# What Are Arrays?

An array is a fixed-size collection of elements, all of the same type, stored in contiguous memory locations. Think of it as a row of lockers, each capable of holding a single value.

## Key features

- Fixed size: Once created, the size of an array cannot be changed.
- Indexed: Each element in an array is assigned a unique index starting from 0.
- Same type: All elements must be of the same data type (e.g., all integers, all strings).

## Array Example

# Advantages

- Easy to use when the number of elements is known beforehand.
- Memory-efficient for fixed-size data.

# Quick Method Explanation

Before diving into lists, let's briefly talk about methods. A method is a reusable block of code that performs a specific task, like adding an item to a list or removing an item. Think of methods as tools or commands you can use to interact with data or objects in your program. Don't worry—later, we'll cover how to create and use methods in detail. For now, we'll work on understanding how to use some built-in methods to work with lists. Don't be scared when you see this in a couple of pages!

# Calling a Method

To call (or invoke) a method, you use its name followed by parentheses. If the method requires inputs (parameters), you pass them inside the parentheses. You'll see this better later, so don't be worried if you're a little confused.

# What Are Lists?

A list is a dynamic collection of elements, meaning its size can grow or shrink as needed. In Java, lists are typically implemented using the ArrayList class from the java.util package.

## Key features

- Dynamic size: Lists can grow or shrink as elements are added or removed.
- Indexed: Like arrays, lists are indexed starting from 0.
- Same type: Elements in a list must be of the same data type.

# List Example

```java
import java.util.ArrayList;

public class Main {
    public static void main(String[] args) {

        ArrayList<String> groceryList = new ArrayList<>();


        groceryList.add("Apples");
        groceryList.add("Bananas");
        groceryList.add("Carrots");


        System.out.println("First item: " + groceryList.get(0));

        groceryList.set(1, "Blueberries");

        groceryList.remove(2);

        System.out.println("Total items: " + groceryList.size());

        System.out.println("Grocery List: " + groceryList);
    }
}
```

Methods in ArrayList

```
First item: Apples
Total items: 2
Grocery List: [Apples, Blueberries]
```

Output

# Advantages

- Flexible size for dynamic programs.
- Rich methods for adding, removing, and manipulating elements.

# Built-in Methods for Lists

## 1. add method

Adds an element to the end of the list.

Common Syntax: add(element)

```java
import java.util.ArrayList;

public class Main {
    public static void main(String[] args) {
        ArrayList<String> colors = new ArrayList<>();
        colors.add("Red");
        colors.add("Blue");
        colors.add("Green");
        System.out.println(colors);
    }
}
```

output

```
[Red, Blue, Green]
```

## 2. add specifically method

Inserts an element at the specified index, shifting the current elements to the right.

Common Syntax: add(index, element)

```java
import java.util.ArrayList;

public class Main {
    public static void main(String[] args) {
        ArrayList<String> colors = new ArrayList<>();
        colors.add("Red");
        colors.add("Blue");
        colors.add("Green");

        colors.add(1, "Yellow");
        System.out.println(colors);
    }
}
```

output

```
[Red, Yellow, Blue, Green]
```

## 3. get method

Retrieves the element at the specified index.

Common Syntax: get(index)

```java
import java.util.ArrayList;

public class Main {
    public static void main(String[] args) {
        ArrayList<String> colors = new ArrayList<>();
        colors.add("Red");
        colors.add("Blue");
        colors.add("Green");
        colors.add(1, "Yellow");

        String firstColor = colors.get(0);

        System.out.println("First color: " + firstColor);

    }
}
```

output

```
First color: Red
```

## 4. set method

Replaces the element at the specified index with a new element.

Common Syntax: set(index, element)

```java
import java.util.ArrayList;

public class Main {
    public static void main(String[] args) {
        ArrayList<String> colors = new ArrayList<>();
        colors.add("Red");
        colors.add("Blue");
        colors.add("Green");
        colors.add(1, "Yellow");
        colors.set(2, "Purple");

        System.out.println(colors);

    }
}
```

output

```
[Red, Yellow, Purple, Green]
```

## 5. remove method

Removes the element at the specified index. The elements to the right of the removed element shift left.

Common Syntax: remove(index)

```java
import java.util.ArrayList;

public class Main {
    public static void main(String[] args) {
        ArrayList<String> colors = new ArrayList<>();
        colors.add("Red");
        colors.add("Blue");
        colors.add("Green");
        colors.add(1, "Yellow");
        colors.set(2, "Purple");
        colors.remove(3);

        System.out.println(colors);

    }
}
```

```
[Red, Yellow, Purple]
```

output

## 6. size method

Returns the number of elements in the list.

Common Syntax: size()

```java
import java.util.ArrayList;

public class Main {
    public static void main(String[] args) {
        ArrayList<String> colors = new ArrayList<>();
        colors.add("Red");
        colors.add("Blue");
        colors.add("Green");
        colors.add(1, "Yellow");
        colors.set(2, "Purple");
        colors.remove(3);

        int totalColors = colors.size();

        System.out.println("Total colors: " + totalColors);
    }
}
```

output

```
Total colors: 3
```

## 7. Contains method

Checks if the list contains a specific element. Returns true if found, otherwise false.

Common Syntax: contains(element)

```java
import java.util.ArrayList;

public class Main {
    public static void main(String[] args) {
        ArrayList<String> colors = new ArrayList<>();
        colors.add("Red");
        colors.add("Blue");
        colors.add("Green");
        colors.add(1, "Yellow");
        colors.set(2, "Purple");
        colors.remove(3);

        boolean hasRed = colors.contains("Red");
        System.out.println("Does the list have Red? " + hasRed);

    }
}
```

output

```
Does the list have Red? true
```

## 8. Is-it-Empty method

Checks if the list is empty. Returns true if there are no elements, otherwise false.

Common Syntax: isEmpty()

```java
import java.util.ArrayList;

public class Main {
    public static void main(String[] args) {
        ArrayList<String> colors = new ArrayList<>();
        colors.add("Red");
        colors.add("Blue");
        colors.add("Green");
        colors.add(1, "Yellow");
        colors.set(2, "Purple");
        colors.remove(3);

        boolean isEmpty = colors.isEmpty();
        System.out.println("Is the list empty? " + isEmpty);
    }
}

```

```
Does the list have Red? true
```

output

## 9. clearing method

Removes all elements from the list, making it empty. If you were to print the list, nothing would return to the console besides some brackets. However, you can still add things to it! This method doesn't indefinitely empty the list.

Common Syntax: clear()

```java
import java.util.ArrayList;

public class Main {
    public static void main(String[] args) {
        ArrayList<String> colors = new ArrayList<>();
        colors.add("Red");
        colors.add("Blue");
        colors.add("Green");
        colors.add(1, "Yellow");
        colors.set(2, "Purple");
        colors.remove(3);

        colors.clear();
        System.out.println("After clear: " + colors);

        System.out.println("Is the list empty? " + colors.isEmpty());
    }
}
```

```
After clear: []
Is the list empty?
true
```

output

Good job! Those are all the major methods you need to know for lists. Don't worry if you're still confused as to what a method is. We'll understand them better soon, but for now, think of them as shortcuts to performing difficult things. While these methods are ready to use after `import java.util.ArrayList`, we can also make methods of our own. The possibilities for methods are endless, making code fun and unpredictable.

# Key Differences

| Feature | Arrays | Lists |
| --- | --- | --- |
| Size | Fixed after creation | Dynamic, can grow or shrink |
| Data Access | Use `array[index]` | Use `list.get(index)` |
| Manipulation | Manual (ex. value reassign) | Methods (`add`, `remove`) |
| Performance | Faster for fixed-size data | Slight overhead bc flexibility |

## Overhead

In programming, overhead refers to the extra resources (such as memory, time, or processing power) required to perform a task beyond the essential work itself. It is the "hidden cost" of using certain features, tools, or abstractions in your code.


# When to Use What?

Use arrays when:

The number of elements is known and fixed.

You need performance-critical, lightweight collections.

Use lists when:

The collection size changes dynamically.

You need powerful built-in methods for manipulation.

# Some More Examples

## 1a)

```java
import java.util.ArrayList;

public class Main {
    public static void main(String[] args) {
        ArrayList<String> groceryList = new ArrayList<>();

        // Add items
        groceryList.add("Milk");
        groceryList.add("Bread");
        groceryList.add("Eggs");

        // Insert item at a specific position
        groceryList.add(1, "Butter");

        // Display items
        System.out.println("Grocery List: " + groceryList);

        // Remove an item
        groceryList.remove("Bread");

        // Check if an item is in the list
        System.out.println("Do I need Milk? " + groceryList.contains("Milk"));

        // Check the total number of items
        System.out.println("Total items: " + groceryList.size());
    }
}
```

1b)

```
Grocery List: [Milk, Butter, Bread, Eggs]
Do I need Milk? true
Total items: 3
```

output

2a)

```java
import java.util.ArrayList;

public class Main {
    public static void main(String[] args) {
        ArrayList<String> tasks = new ArrayList<>();

        // Add tasks
        tasks.add("Finish homework");
        tasks.add("Clean room");
        tasks.add("Prepare for meeting");

        // Print initial tasks
        System.out.println("Initial tasks: " + tasks);

        // Complete a task and remove it
        tasks.remove(0);

        // Add a new task
        tasks.add("Buy groceries");

        // Check remaining tasks
        System.out.println("Remaining tasks: " + tasks);
    }
}
```

## 2b)

```
Initial tasks: [Finish homework, Clean room, Prepare for meeting]
Remaining tasks: [Clean room, Prepare for meeting, Buy groceries]
```

output

# New Projects Alert

## Student grades

1. Create a list of student names.
2. Add 5 names to the list.
3. Print the list.
4. Remove the third name and print the updated list.
5. Replace the first name with a new one.

## Creating a Playlist

1. Create a list for your favorite songs.

2. Add 3 songs, print the list.

3. Add a new song at position 2.

4. Remove the last song.

5. Print the final list and the number of songs.

How'd you do? Don't worry if you're struggling a bit! These are difficult concepts to grasp. Practice makes perfect. If you're completely lost, take a look at the answer key on the next page or review this chapter again.

# Answer Key

## Student grades

```java
import java.util.ArrayList;

public class Main {
    public static void main(String[] args) {
        // Step 1: Create a list of student names
        ArrayList<String> studentNames = new ArrayList<>();

        // Step 2: Add 5 names to the list
        studentNames.add("Alice");
        studentNames.add("Bob");
        studentNames.add("Charlie");
        studentNames.add("David");
        studentNames.add("Eve");

        // Step 3: Print the list
        System.out.println("Student Names: " + studentNames);

        // Step 4: Remove the third name
        studentNames.remove(2); // Removes "Charlie"
        System.out.println("After removing third name: " + studentNames);

        // Step 5: Replace the first name with a new one
        studentNames.set(0, "Zara");
        System.out.println("After replacing first name: " + studentNames);
    }
}

```

```
Student Names: [Alice, Bob, Charlie, David, Eve]
After removing third name: [Alice, Bob, David, Eve]
After replacing first name: [Zara, Bob, David, Eve]
```

output

# Creating a Playlist

```java
import java.util.ArrayList;

public class Main {
    public static void main(String[] args) {
        // Step 1: Create a list for your favorite songs
        ArrayList<String> playlist = new ArrayList<>();

        // Step 2: Add 3 songs
        playlist.add("Song A");
        playlist.add("Song B");
        playlist.add("Song C");

        // Step 3: Print the list
        System.out.println("Playlist: " + playlist);

        // Step 4: Add a new song at position 2
        playlist.add(1, "Song D"); // Inserts "Song D" at index 1
        System.out.println("After adding a song at position 2: " + playlist);

        // Step 5: Remove the last song
        playlist.remove(playlist.size() - 1); // Removes the last song
        System.out.println("After removing the last song: " + playlist);

        // Step 6: Print the final list and the number of songs
        System.out.println("Final Playlist: " + playlist);
        System.out.println("Number of Songs: " + playlist.size());
    }
}
```

```
Playlist: [Song A, Song B, Song C]
After adding a song at position 2: [Song A, Song D, Song B, Song C]
After removing the last song: [Song A, Song D, Song B]
Final Playlist: [Song A, Song D, Song B]
Number of Songs: 3
```

output

# Re-cap!

Lists and arrays are very important tools for managing collections of data. While arrays are simple and efficient, lists provide powerful functionality for dynamic programs. By mastering these structures, you'll unlock the ability to organize and manipulate data in your programs effectively.

# Unit 9

# Methods

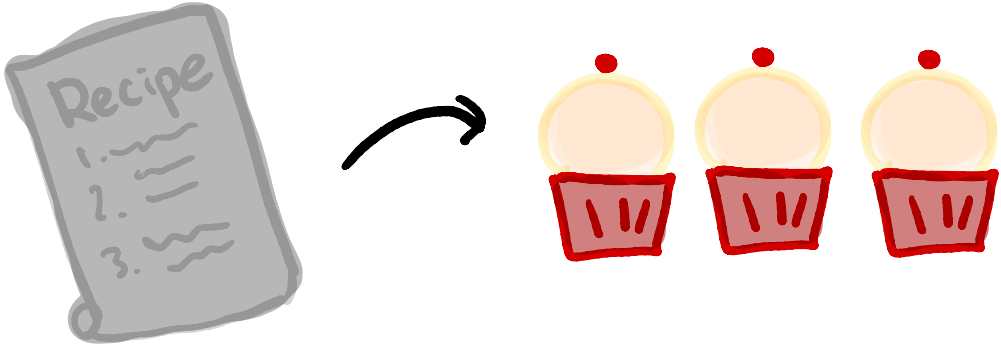
# What Is a Method?

Methods are one of the most important concepts in programming. They allow you to organize code into reusable blocks, making your programs cleaner, easier to read, and more efficient.

A method is a block of code that performs a specific task. It is similar to a function in other programming languages, but in Java, methods are part of classes. Methods are used to:

1. Perform specific actions.
2. Reuse code without rewriting it.
3. Improve readability and maintainability of your programs.

**Analogy:** A chef call for a recipe whenever he needs to make a new batch of cupcakes.



# Key Features of Methods

Name: Identifies the method.

Return Type: Specifies what the method gives back after execution (or void if it doesn't return anything).

Parameters: Inputs the method needs to perform its task (optional).

Body: The block of code that runs when the method is called.

# Common Syntax

```
public static returnType methodName(parameters){
    // previous code
    return value;
}
```

## Example

```
public static int addNumbers(int a, int b) {
    return a + b;
}
```

# Calling a Method

To call a method, you use its name followed by parentheses. If the method requires inputs, you pass them inside the parentheses.

```java
public class Main {
    // Method to greet a user
    public static void greet(String name) {
        System.out.println("Hello, " + name + "!");
    }

    public static void main(String[] args) {
        // Call the method directly
        greet("Alice"); // Outputs: Hello, Alice!
    }
}
```

# Types of Methods

## 1. methods without return values

Use the keyword void as the return type if the method doesn't return a value.

```java
public class Main {
    // Make the method static
    public static void printMessage(String message) {
        System.out.println(message);
    }
```

Call the method:

```
printMessage("Cooper"); // Outputs: Cooper
```

## 2. methods with return values

Specify the return type (e.g., int, String) and use the return keyword to send a value back.

```
1   public class Main {
2       // Make the method static
3       public static int getFive() {
4           return 5;
5       }
6
```

Call the method:

```
System.out.print(getFive()); // Outputs: 5
```

## 3. methods with parameters

Methods can take inputs (parameters) to perform their task.

```
1   public class Main {
2       // Make the method static
3       public static void order(String name) {
4           System.out.println("Order for " + name + "!");
5       }
6
```

Call the method:

```
order("Ashley"); // Outputs: Order for Ashley!
```

## 4. Overloading methods

You can define multiple methods with the same name but different parameter lists. This is called method overloading

```
1   public class Main {
2
3       public static void printNumber(int num) {
4           System.out.println("Integer: " + num);
5       }
6
7       public static void printNumber(double num) {
8           System.out.println("Double: " + num);
9       }
10
```

Call the method:

```
printNumber(5);      // Outputs: Integer: 5
printNumber(3.14);   // Outputs: Double: 3.14
```

# Why Methods?

Code Reusability: Write once, use many times.

Readability: Break down complex tasks into smaller, manageable pieces.

Maintainability: Easier to update or debug code.

# Example

```java
public class Main {
    // Add two numbers
    public static int add(int a, int b) {
        return a + b;
    }

    // Subtract two numbers
    public static int subtract(int a, int b) {
        return a - b;
    }

    // Multiply two numbers
    public static int multiply(int a, int b) {
        return a * b;
    }

    // Divide two numbers
    public static double divide(double a, double b) {
        if (b == 0) {
            System.out.println("Error: Division by zero");
            return 0;
        }
        return a / b;
    }

    public static void main(String[] args) {
        // Call methods directly without creating an object
        System.out.println("Addition: " + add(10, 5));         // Outputs: 15
        System.out.println("Subtraction: " + subtract(10, 5)); // Outputs: 5
        System.out.println("Multiplication: " + multiply(10, 5)); // Outputs: 50
        System.out.println("Division: " + divide(10, 2));     // Outputs: 5.0
    }
}

```

output

```
Addition: 15
Subtraction: 5
Multiplication: 50
Division: 5.0
```

# Your Turn!

Let's put your knowledge to use! Create two methods to help make a temperature converter.

Here's the conversion: Fahrenheit = Celsius (9/5) +32

Methods:

celsiusToFahrenheit(double celsius): Converts Celsius to Fahrenheit.

fahrenheitToCelsius(double fahrenheit): Converts Fahrenheit to Celsius

# Answer-key

```java
public class Main {

    // Method to convert Celsius to Fahrenheit
    public static double celsiusToFahrenheit(double celsius) {
        return (celsius * 9 / 5) + 32;
    }

    // Method to convert Fahrenheit to Celsius
    public static double fahrenheitToCelsius(double fahrenheit) {
        return (fahrenheit - 32) * 5 / 9;
    }

    public static void main(String[] args) {
        // Example temperatures
        double celsius = 25.0;
        double fahrenheit = 77.0;

        // Convert Celsius to Fahrenheit
        double convertedToFahrenheit = celsiusToFahrenheit(celsius);
        System.out.println(celsius + "°C in Fahrenheit: " + convertedToFahrenheit + "°F");

        // Convert Fahrenheit to Celsius
        double convertedToCelsius = fahrenheitToCelsius(fahrenheit);
        System.out.println(fahrenheit + "°F in Celsius: " + convertedToCelsius + "°C");
    }
}
```

output →

```
25.0°C in Fahrenheit: 77.0°F
77.0°F in Celsius: 25.0°C
```

# Re-cap

Methods are reusable blocks of code that perform specific tasks. They can return values, take inputs, and even be overloaded for flexibility. Breaking programs into methods improves readability, maintainability, and reusability.


# Objects and Classes

# What Are Objects?

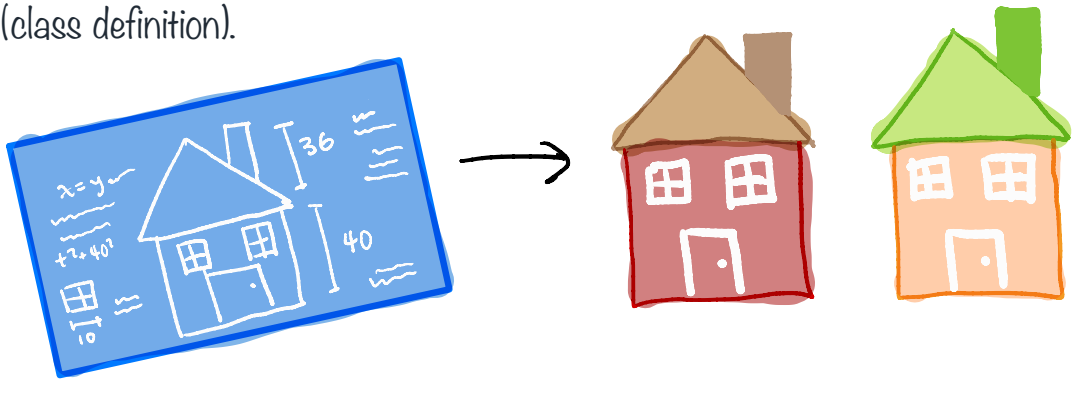
In Java, objects are the building blocks of object-oriented programming (OOP). Think of an object as a "thing" that has properties (attributes) and behaviors (actions it can perform). Objects make it easier to organize, model, and interact with real-world concepts in your code.

An object is an instance of a class. You can think of a class as a blueprint, and an object as the actual thing created from that blueprint.

# Analogy

Class: A blueprint for a house.
Object: A specific house built using the blueprint.

Each house (object) built from the same blueprint (class) can have its own unique features (property values) but shares the same structure (class definition).



# Key Features of Objects

Attributes: The data that defines the state of an object.
Example: A car has attributes like color, brand, and speed.

Methods: The actions or behaviors the object can perform.
Example: A car can accelerate, brake, or honk.

# How to Make an Object

Define a class (the blueprint).
Use the new keyword to create an instance of the class.

Common Syntax: `ClassName objectName= new ClassName();`

To assign your object a certain attribute, first make sure you initialize the attributes at the top of your code as global, meaning that no matter where you are in your code, you can see them. To make the attributes global, make sure they're not trapped in any method and are correctly aligned with everything else! Then, to use the attribute with your object just call them after your desired object!

Common Syntax: `object.wantedAttribute= value;`

# Example

```java
class Main {
    // Attributes
    String brand;
    String color;
    int speed;

    // Method to display car details
    void displayDetails() {
        System.out.println("Brand: " + brand + ", Color: " + color + ", Speed: " + speed + " km/h");
    }

    // Method to accelerate the car
    void accelerate(int increase) {
        speed += increase;
        System.out.println("The car accelerated to " + speed + " km/h.");
    }

    // Method to brake the car
    void brake(int decrease) {
        speed -= decrease;
        if (speed < 0) {
            speed = 0;
        }
        System.out.println("The car slowed down to " + speed + " km/h.");
    }

    public static void main(String[] args) {
        // Create a Car object
        Car myCar = new Car();

        // Set attributes
        myCar.brand = "Toyota";
        myCar.color = "Red";
        myCar.speed = 50;

        // Use methods
        myCar.displayDetails(); // Outputs: Brand: Toyota, Color: Red, Speed: 50 km/h
        myCar.accelerate(20);   // Outputs: The car accelerated to 70 km/h.
        myCar.brake(30);        // Outputs: The car slowed down to 40 km/h.
    }
}

```

global

assigning attributes to objects

```
Brand: Toyota, Color: Red, Speed: 50 km/h
The car accelerated to 70 km/h.
The car slowed down to 40 km/h.
```

output

# Why Use Objects?

Objects make your code:

Modular: You can break your program into smaller, reusable pieces.
Scalable: Adding new features or functionality becomes easier.
Organized: Related data and methods are grouped together.

## New Project Alert 

This project involves creating a Student class to track student details and grades. You'll create objects from the class, set the attributes manually, and use methods to interact with the data.

1. Create a Student class with attributes for the student's name, grade level, and average grade.

2. Add methods to:
   Display the student's information.
   Update the student's average grade.

# Answer-Key

```java
class Student {
    // Attributes
    String name;
    int gradeLevel;
    double averageGrade;

    // Method to display student details
    void displayDetails() {
        System.out.println("Name: " + name);
        System.out.println("Grade Level: " + gradeLevel);
        System.out.println("Average Grade: " + averageGrade);
    }

    // Method to update the average grade
    void updateGrade(double newGrade) {
        averageGrade = newGrade;
        System.out.println(name + "'s new average grade: " + averageGrade);
    }

    public static void main(String[] args) {
        // Create student objects
        Student student1 = new Student();
        Student student2 = new Student();

        // Set attributes manually
        student1.name = "Alice";
        student1.gradeLevel = 10;
        student1.averageGrade = 88.5;

        student2.name = "Bob";
        student2.gradeLevel = 11;
        student2.averageGrade = 92.3;

        // Display initial details
        System.out.println("Initial Student Details:");
        student1.displayDetails();
        System.out.println();
        student2.displayDetails();
        System.out.println();

        // Update grades
        System.out.println("Updating Grades:");
        student1.updateGrade(90.0);
        student2.updateGrade(95.5);
    }
}
```

Keep in mind that Replit is an introductory platform for coding. This code may not run if the class name isn't 'Main'. However, on other platforms, this code will correctly compile.

## How It Works

Attributes:

name: The student's name.

gradeLevel: The student's grade level.

averageGrade: The student's average grade.

Methods:

displayDetails(): Prints the student's current details.

updateGrade(double newGrade): Updates the student's average grade with a new value.

Manual Attribute Assignment:

Attributes are set manually after creating objects, without using constructors.

Execution:

Create two Student objects.

Set their attributes manually.

Display their details, update grades, and display the changes.

# Re-cap!

Objects are the foundation of object-oriented programming. They bundle data (attributes) and behaviors (methods) into a single unit. You can create multiple objects from the same class and make them interact.

Congrats! You did it! Unfortunately, this book has come to an end. I really hope you were able to follow along and learn from this book. I spent hours crafting a curriculum that I wish I had when I started coding. There are still many things you should know in order to code, but the topics in this book should give you a good grasp on how to start! There are so many great sources online to learn from, so don't be discouraged if this book didn't answer all your questions. I'm so proud of you for sticking through this. Be proud! You just learned how to code.

Bye, for now!