AlmaBetter

Introduction to JavaScript

Introduction to JavaScript

JavaScript is a programming language that adds interactivity to your website. This happens in games, in the behavior of responses when buttons are pressed or with data entry on forms; with dynamic styling; with animation, etc. This article helps you get started with JavaScript and furthers your understanding of what is possible.

What is JavaScript?

JavaScript is a powerful programming language that can add interactivity to a website. It was invented by Brendan Eich.

JavaScript is versatile and beginner-friendly. With more experience, you'll be able to create games, animated 2D and 3D graphics, comprehensive database-driven apps, and much more!

JavaScript itself is relatively compact, yet very flexible. Developers have written a variety of tools on top of the core JavaScript language, unlocking a vast amount of functionality with minimum effort. These include

- Browser Application Programming Interfaces (APIs) built into web browsers, providing functionality such as dynamically creating HTML and setting CSS styles; collecting and manipulating a video stream from a user's webcam, or generating 3D graphics and audio samples.
- Third-party APIs allow developers to incorporate functionality in sites from other content providers, such as Twitter or Facebook.
- Third-party frameworks and libraries that you can apply to HTML to accelerate the work of building sites and applications.

A "Hello world!" example

JavaScript is one of the most popular modern web technologies! As your JavaScript skills grow, your websites will enter a new dimension of power and creativity.

However, getting comfortable with JavaScript is more challenging than getting comfortable with HTML and CSS. You may have to start small, and progress gradually. To begin, let's examine how to add JavaScript to your page for creating a ***Hello world!*** example.

1) Go to your test site and create a new folder named scripts. Within the scripts folder, create a new text document called `main.js`, and save it.

2) In your `index.html` file, enter this code on a new line, just before the closing `</body>` tag:

```
<script src="scripts/main.js"></script>
```

3) This is doing the same job as the `<link>` element for CSS. It applies JavaScript to the page, so it can have an effect on the HTML (along with the CSS, and anything else on the page).

4) Add this code to the `main.js` file:

```
const myHeading = document.querySelector("h1");
myHeading.textContent = "Hello world!";
```

5) Make sure the HTML and JavaScript files are saved. Then load index.html in your browser. You should see something like this:

What happened?

The heading text changed to Hello world! using JavaScript. You did this by using a function called `querySelector()` to grab a reference to your heading and then store it in a variable called `myHeading`. This is similar to what we did using CSS selectors. When you want to do something to an element, you need to select it first.

Following that, the code set the value of the `myHeading` variable's `textContent` property (which represents the content of the heading) to Hello world!

Language basics crash course

To give you a better understanding of how JavaScript works, let's explain some of the core features of the language. It's worth noting that these features are common to all programming languages. If you master these fundamentals, you have a head start on coding in other languages too!

a) Variables

Variables are containers that store values. You start by declaring a variable with the let keyword, followed by the name you give to the variable:

```
let myVariable;
```

A semicolon at the end of a line indicates where a statement ends. It is only required when you need to separate statements on a single line. However, some people believe it's good practice to have semicolons at the end of each statement. There are other rules for when you should and shouldn't use semicolons.

You can name a variable nearly anything, but there are some restrictions. If you are unsure, you can check your variable name to see if it's valid.

JavaScript is case-sensitive. This means myVariable is not the same as `myVariable`. If you have problems with your code, check the case!

After declaring a variable, you can give it a value:

```
myVariable = "Bob";
```

Also, you can do both these operations on the same line:

```
let myVariable = "Bob";
```

You retrieve the value by calling the variable name:

```
myVariable;
```

After assigning a value to a variable, you can change it later in the code:

```
let myVariable = "Bob";
myVariable = "Steve";
```

Note that variables may hold values that have different data types:

Variable	Explanation	Example
String	This is a sequence of text known as a string. To signify that the value is a string, enclose it in single quotation marks.	`let myVariable = 'Bob';`
Number	This is a number. Numbers don't have quotes around them.	`let myVariable = 10;`
Boolean	This is a True/False value. The words `true` and `false` are special keywords that don't need quotation marks.	`let myVariable = true;`
Array	This is a structure that allows you to store multiple values in a single reference.	`let myVariable = [1,'Bob','Steve',10];` Refer to each member of the array like this: `myVariable[0]`, `myVariable[1]`, etc.
Object	This can be anything. Everything in JavaScript is an object and can be stored in a variable. Keep this in mind as you learn.	`let myVariable = document.querySelector('h1');` All of the above examples too.

So why do we need variables? Variables are necessary to do anything interesting in programming. If values couldn't change, then you couldn't do anything dynamic, like personalizing a greeting message or changing an image displayed in an image gallery.

b) Comments

Comments are snippets of text that can be added along with the code. The browser ignores text marked as comments. You can write comments in JavaScript just as you can in CSS:

```
/*
Everything in between is a comment.
*/
```

If your comment contains no line breaks, it's an option to put it behind two slashes like this:

```
// This is a comment
```

c) Operators

An `operator` is a mathematical symbol that produces a result based on two values (or variables). In the following table, you can see some of the simplest operators, along with some examples to try in the JavaScript console.

Operator	**Explanation**	**Symbol(s)**	**Example**
Addition	Add two numbers together or combine two strings.	+	`6 + 9;` `'Hello ' + 'world!';`

Subtraction, Multiplication, and Division	These do what you'd expect them to do in basic math.	-, *, /	`9 - 3;`
			`8 * 2; // multiply in JS is an asterisk`
			`9 / 3;`

Assignment	As you've seen already: this assigns a value to a variable.	=	`let myVariable = 'Bob';`
Strict equality	This performs a test to see if two values are equal. It returns a true/false (Boolean) result.	===	`let myVariable = 3;` `myVariable === 4;`

Not, Does-not-equal	This returns the logically opposite value of what it precedes. It turns a `true` into a `false`, etc. When it is used alongside the Equality operator, the negation operator tests whether two values are not equal.	!, !==	For "Not", the basic expression is true, but the comparison returns `false` because we negate it:
			`let myVariable = 3;`
			`!(myVariable === 3);`
			
			"Does-not-equal" gives the same result with different syntax. Here we are testing "is `myVariable` NOT equal to 3". This returns `false` because `myVariable` is equal to 3:
			`let myVariable = 3;`
			`myVariable !== 3;`

There are a lot more operators to explore, but this is enough for now. See Expressions and operators for a complete list.

d) Conditionals

Conditionals are code structures used to test if an expression returns true or not. A very common form of conditionals is the `if...else` statement. For example,

```
let iceCream = "chocolate";
if (iceCream === "chocolate") {
  alert("Yay, I love chocolate ice cream!");
} else {
  alert("Awwww, but chocolate is my favorite…");
}
```

The expression inside the `if ()` is the test. This uses the strict equality operator (as described above) to compare the variable `icecream` with the string `chocolate` to see if the two are equal. If this comparison returns true, the first block of code runs. If the comparison is not true, the second block of code—after the `else` statement—runs instead.

e) Functions

Functions are a way of packaging functionality that you wish to reuse. It's possible to define a body of code as a function that executes when you call the function name in your code. This is a good alternative to repeatedly writing the same code. You have already seen some uses of functions. For example,

```
let myVariable = document.querySelector("h1");
```

```
alert("hello!");
```

These functions, `document.querySelector` and `alert`, are built into the browser.

If you see something which looks like a variable name, but it's followed by parentheses— `()` —it is likely a function. Functions often take arguments: bits of data they need to do their job. Arguments go inside the parentheses, separated by commas if there is more than one argument.

For example, the `alert()` function makes a pop-up box appear inside the browser window, but we need to give it a string as an argument to tell the function what message to display.

You can also define your own functions. In the next example, we create a simple function that takes two numbers as arguments and multiplies them:

```
function multiply(num1, num2) {
  let result = num1 * num2;
  return result;
}
```

Try running this in the console; then test with several arguments. For example

```
multiply(4, 7);
multiply(20, 20);
multiply(0.5, 3);
```

f) Events

Real interactivity on a website requires event handlers. These are code structures that listen for activity in the browser, and run code in response. The most obvious example is handling the click event, which is fired by the browser when you click on something with your mouse. To demonstrate this, enter the following into your console, then click on the current webpage:

```
document.querySelector("html").addEventListener("click", function () {
  alert("Ouch! Stop poking me!");
});
```

There are a number of ways to attach an event handler to an element. Here we select the `<html>` element. We then call its `addEventListener()` function, passing in the event's name to listen to (`'click'`) and a function to run when the event happens.

The function we just passed to addEventListener() here is called an anonymous function because it doesn't have a name. There's an alternative way of writing anonymous functions, which we call an arrow function. An arrow function uses `() =>` instead of `function ()`:

```
document.querySelector("html").addEventListener("click", () => {
  alert("Ouch! Stop poking me!");
});
```

Supercharging our example website

With this review of JavaScript basics completed (above), let's add some new features to our example site.

Before going any further, delete the current contents of your `main.js` file — the bit you added earlier during the "Hello world!" example — and save the empty file. If you don't, the existing code will clash with the new code you are about to add.

Adding an image changer

In this section, you will learn how to use JavaScript and DOM API features to alternate the display of one of two images. This change will happen as a user clicks the displayed image.

- Choose an image you want to feature on your example site. Ideally, the image will be the same size as you added previously, or as close as possible.
- Save this image in your `images` folder.
- Rename the image firefox2.png.

- Add the following JavaScript code to your `main.js` file.

```
const myImage = document.querySelector("img");

myImage.onclick = () => {
  const mySrc = myImage.getAttribute("src");
  if (mySrc === "images/firefox-icon.png") {
    myImage.setAttribute("src", "images/firefox2.png");
  } else {
    myImage.setAttribute("src", "images/firefox-icon.png");
  }
};
```

- Save all files and load `index.html` in the browser. Now when you click the image, it should change to the other one.

This is what happened. You stored a reference to your `<img>` element in `myImage`. Next, you made its `onclick` event handler property equal to a function with no name (an "anonymous" function). So every time this element is clicked:

- The code retrieves the value of the image's `src` attribute.
- The code uses a conditional to check if the `src` value is equal to the path of the original image:
- If it is, the code changes the `src` value to the path of the second image, forcing the other image to be loaded inside the `<img>` element.
- If it isn't (meaning it must already have changed), the src value swaps back to the original image path, to the original state.

Adding a personalized welcome message

Next, let's change the page title to a personalized welcome message when the user first visits the site. This welcome message will persist. Should the user leave the site and return later, we will save the message using the Web Storage API. We will also include an option to change the user, and therefore, the welcome message.

1) In `index.html`, add the following line just before the `<script>` element:

```
<button>Change user</button>
```

2) In `main.js`, place the following code at the bottom of the file, exactly as it is written. This takes references to the new button and the heading, storing each inside variables:

```
let myButton = document.querySelector("button");
let myHeading = document.querySelector("h1");
```

3) Add the following function to set the personalized greeting. This won't do anything yet, but this will change soon.

```
function setUserName() {
  const myName = prompt("Please enter your name.");
  localStorage.setItem("name", myName);
  myHeading.textContent = `Mozilla is cool, ${myName}`;
}
```

The `setUserName()` function contains a `prompt()` function, which displays a dialog box, similar to `alert()`. This `prompt()` function does more than `alert()`, asking the user to enter data, and storing it in a variable after the user clicks OK. In this case, we are asking the user to enter a name. Next, the code calls on an API `localStorage`, which allows us to store data in the browser and retrieve it later. We use localStorage's `setItem()` function to create and store a data item called `'name'`, setting its value to the `myName` variable which contains the user's entry for the name. Finally, we set the `textContent` of the heading to a string, plus the user's newly stored name.

4) Add the following condition block. We could call this initialization code, as it structures the app when it first loads.

```
if (!localStorage.getItem("name")) {
  setUserName();
} else {
  const storedName = localStorage.getItem("name");
  myHeading.textContent = `Mozilla is cool, ${storedName}`;
}
```

This first line of this block uses the negation operator (logical NOT, represented by the `!`) to check whether the `name` data exists. If not, the `setUserName()` function runs to create it. If it exists (that is, the user set a user name during a previous visit), we retrieve the stored name

using `getItem()` and set the `textContent` of the heading to a string, plus the user's name, as we did inside `setUserName()`.

5) Put this `onclick` event handler (below) on the button. When clicked, `setUserName()` runs. This allows the user to enter a different name by pressing the button.

```
myButton.onclick = () => {
  setUserName();
};
```

A user name of null?

When you run the example and get the dialog box that prompts you to enter your user name, try pressing the Cancel button. You should end up with a title that reads Mozilla is cool, null. This happens because—when you cancel the prompt—the value is set as `null`. Null is a special value in JavaScript that refers to the absence of a value.

Also, try clicking OK without entering a name. You should end up with a title that reads Mozilla is cool, for fairly obvious reasons.

To avoid these problems, you could check that the user hasn't entered a blank name. Update your `setUserName()` function to this:

```
function setUserName() {
  const myName = prompt("Please enter your name.");
  if (!myName) {
    setUserName();
  } else {
    localStorage.setItem("name", myName);
    myHeading.textContent = `Mozilla is cool, ${myName}`;
  }
}
```

In human language, this means: If `myName` has no value, run `setUserName()` again from the start. If it does have a value (if the above statement is not true), then store the value in `localStorage` and set it as the heading's text.

Conclusion

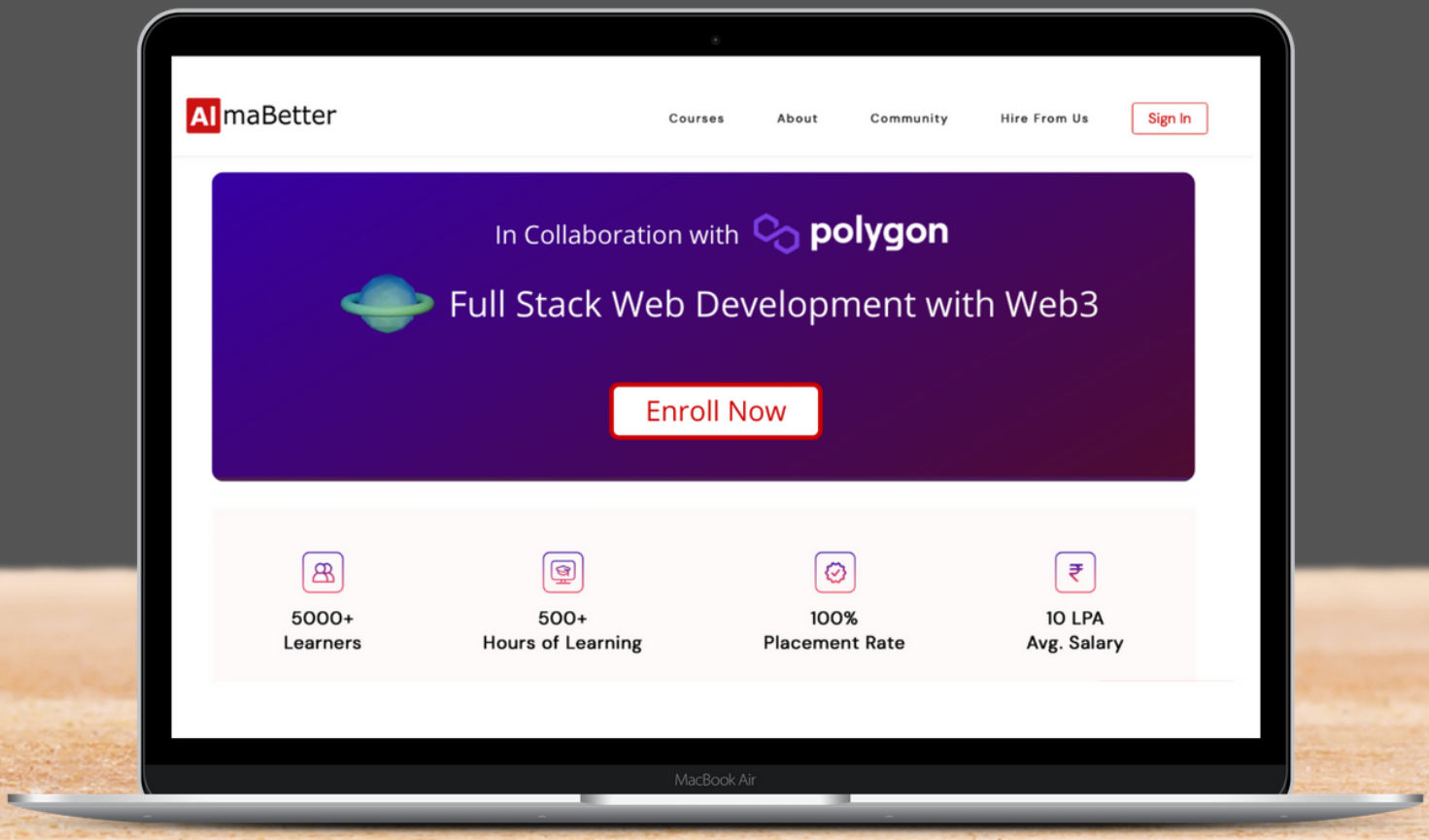
If you have followed all the instructions in this article, you should end up with a page that looks something like the image below.

We have just scratched the surface of JavaScript. If you enjoyed playing, and wish to go further, take advantage of the resources listed below.

If you're considering a career in Web3 Development, there's no better time than now.

Join our **Full Stack Web Development** with Web3 Program and launch your career in tech today!

Link: https://link.almabetter.com/9w63



AlmaBetter

Introduction to HTML

Beginner's Guide

Introduction to HTML: Beginner’s Guide

What is HTML?

HTML stands for Hyper Text Markup Language

1. HTML is the standard markup language for creating Web pages
2. HTML describes the structure of a Web page
3. HTML consists of a series of elements
4. HTML elements tell the browser how to display the content
5. HTML elements label pieces of content such as "this is a heading", "this is a paragraph", "this is a link", etc.

Basic HTML Document

```
<!DOCTYPE html>
<html>
<head>
<title>Page Heading</title>
</head>
<body>
<h1>My First Heading</h1>
<p>My First Paragraph</p>
</body>
</html>
```

Output:

Sample Contents:

1. The declaration defines that this document is an HTML5 document.
2. The element is the root element of an HTML page.
3. The element contains meta information about the HTML page.
4. The element specifies a title for the HTML page (which is shown in the browser's title bar or in the page's tab).
5. The <body> element defines the document's body and is a container for all the visible contents, such as headings, paragraphs, images, hyperlinks, tables, lists, etc.
6. The <h1> element defines a large heading.
7. The <p> element represents a paragraph.
8.

What is an HTML Element?

A start tag, some content, and an end tag define an HTML element:

```
<!DOCTYPE html>
<html>
<head>
<title>Page Title</title>
</head>
<body>

<h1>My First Heading</h1>
<p>My first paragraph.</p>

<tagname> Content goes here... </tagname>

</body>
</html>
```

Output:

The HTML element is everything from the start tag to the end tag:

- `<h1>My First Heading</h1>`
- `<p>My First Paragraph</p>`

HTML Page Structure

```
<html>
<head>
<title>Page title</title>
</head>
<body>
<h1>This is a heading</h1>
<p>This is a paragraph.</p>
<p>This is another paragraph.</p>
</body>
</html>
```

HTML History

Since the early days of the World Wide Web, there have been many versions of HTML:

Year	Version
1989	Tim Berners-Lee invented www
1991	Tim Berners-Lee invented HTML
1993	Dave Raggett drafted HTML+
1995	HTML Working Group defined HTML 2.0
1997	W3C Recommendation: HTML 3.2
1999	W3C Recommendation: HTML 4.01
2000	W3C Recommendation: XHTML 1.0
2008	WHATWG HTML5 First Public Draft
2012	WHATWG HTML5 Living Standard
2014	W3C Recommendation: HTML5
2016	W3C Candidate Recommendation: HTML 5.1
2017	W3C Recommendation: HTML5.1 2nd Edition
2017	W3C Recommendation: HTML5.2

HTML Documents

All HTML documents must start with a document type declaration: `<!DOCTYPE html>`.

The HTML document itself begins with `<html>` and ends with `</html>`.

The visible part of the HTML document is between `<body>` and `</body>`.

The <!DOCTYPE> Declaration

The `<!DOCTYPE>` declaration represents the document type and helps browsers display web pages correctly.

It must only appear once, at the top of the page (before any HTML tags).

The <!DOCTYPE> declaration is not case-sensitive.

The <!DOCTYPE> declaration for HTML5 is: <!DOCTYPE html>

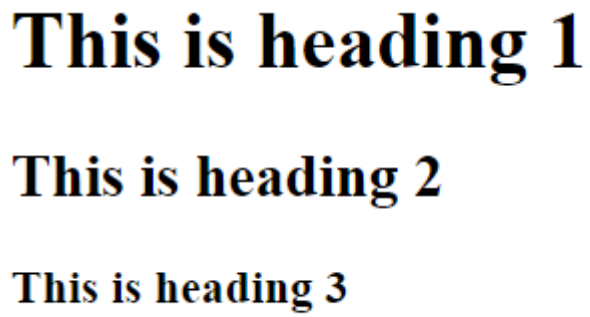
HTML Headings

HTML headings are defined with the <h1> to <h6> tags.

- <h1> defines the most important heading.
- <h6> defines the least important heading:

```
<h1>This is heading 1</h1>
<h2>This is heading 2</h2>
<h3>This is heading 3</h3>
```

Output:

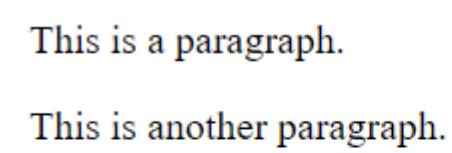


HTML Paragraphs

HTML paragraphs are defined with the <p> tag:

```
<p>This is a paragraph.</p>
<p>This is another paragraph.</p>
```

Output:

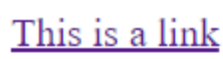


HTML Links

HTML links are defined with the `<a>` tag:

```
<a href="https://www.almabetter.com">This is a link</a>
```

Output:



You will simply be redirected to the linked page source when you click on the link.

The link's destination is specified in the `href` attribute.

Attributes are used to provide additional information about HTML elements.

HTML Images

HTML images are defined with the `<img>` tag.

The source file (`src`), alternative text (`alt`), `width`, and `height` are provided as attributes:

```
<img src="https://link.almabetter.com/gebI" alt="Almabetter.com" width="664"
height="168">
```

Output:

How to View HTML Source?

Have you ever seen a Web page and wondered "Hey! How did they do that?"

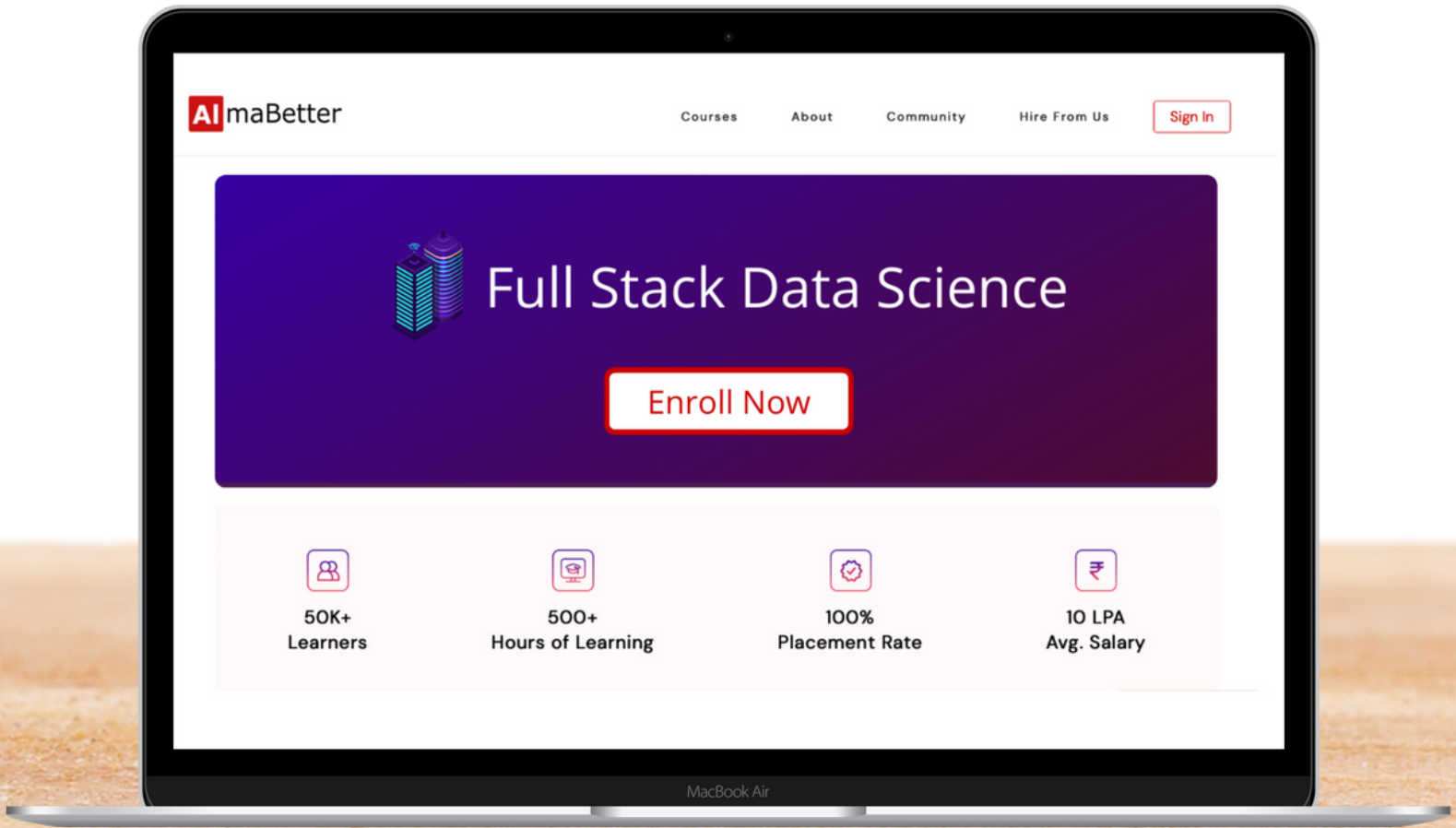
View HTML Source Code: Right-click on an HTML page and select "View Page Source" (in Chrome) or "View Source" (in Edge), or similar in other browsers. This will open a window containing the HTML source code of the page.

Inspect an HTML Element: Right-click on an element (or a blank area), and choose "Inspect" or "Inspect Element" to see what elements are made up of (you will see both the HTML and the CSS). You can also edit the HTML or CSS on-the-fly in the Elements or Styles panel that opens.

If you're looking to get into **Web Development**, then **AlmaBetter** is the best place to start your journey.

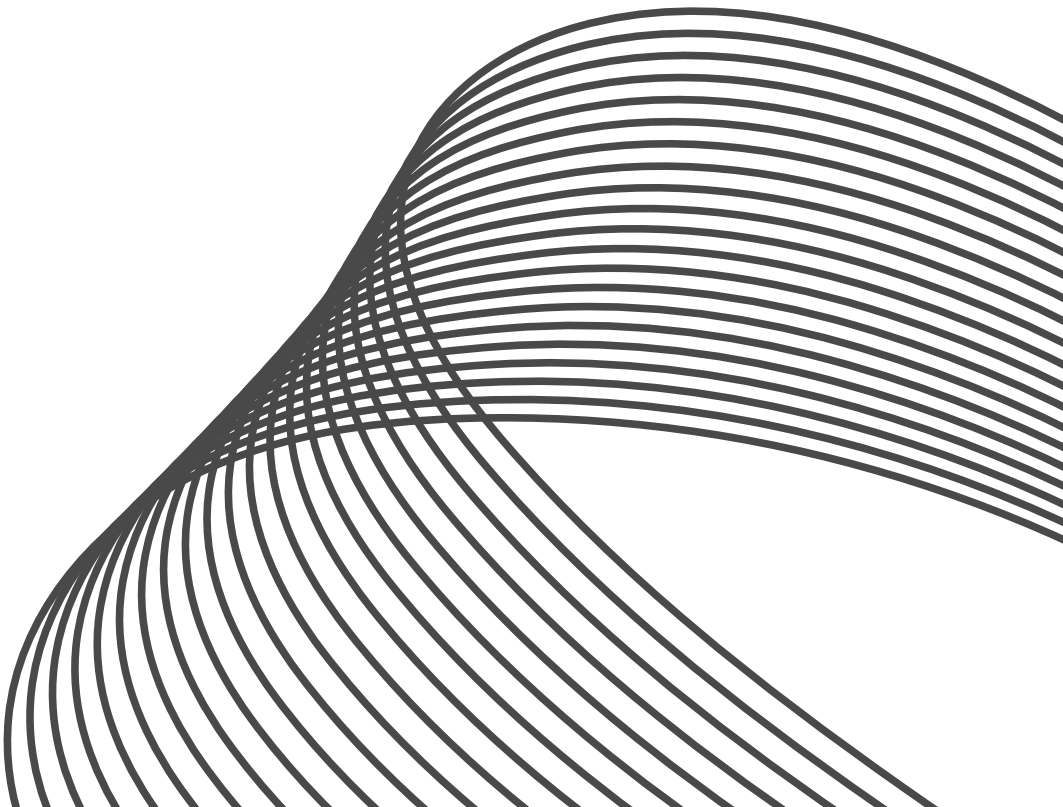
Join our **Full Stack Web Development** with Web3 Program and launch your career in tech today!

Link: https://link.almabetter.com/9w63



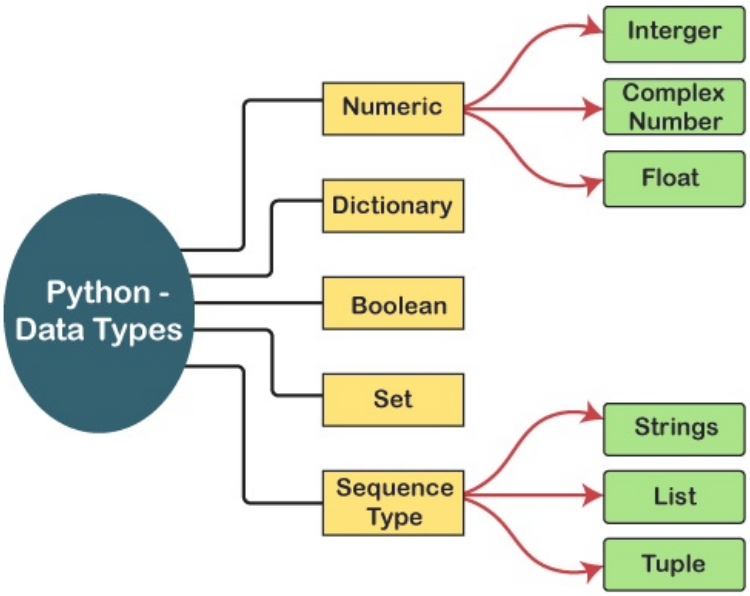
AlmaBetter

Python Basics



Welcome to the Python Basics

Different Types of Data Types in Python



There are eight kinds of types supported by PyTables:

- **bool**: Boolean (true/false) types. Supported precisions: 8 (default) bits.
- **int**: Signed integer types. Supported precisions: 8, 16, 32 (default) and 64 bits.
- **uint**: Unsigned integer types. Supported precisions: 8, 16, 32 (default) and 64 bits.
- **float**: Floating point types. Supported precisions: 16, 32, 64 (default) bits and extended precision floating point (see note on floating point types).
- **complex**: Complex number types. Supported precisions: 64 (32+32), 128 (64+64, default) bits and extended precision complex (see note on floating point types).
- **string**: Raw string types. Supported precisions: 8-bit positive multiples.
- **time**: Data/time types. Supported precisions: 32 and 64 (default) bits.

- **enum**: Enumerated types. Precision depends on base type

bool -> True, False / 1,0

int -> 3,5,-1,-9

float -> 3.45

str -> 'data scientist'

In [ ]:
```
# bool, int, float, str
a = 'Data Science'
print(a)
type(a)
```

```
Data Science
```

Out[ ]:
```
str
```

In [ ]:
```
type(a)
```

Out[ ]:
```
str
```

In [ ]:
```
z = True
type(z)
```

Out[ ]:
```
bool
```

In [ ]:
```
#Autotypecasting
3 + 4.5
```

Out[ ]:
```
7.5
```

In [ ]:
```
True + 2
```

Out[ ]:
```
3
```

In [ ]:
```
int(5.1) #converting float to int
```

Out[ ]:
```
5
```

In [ ]:
```
# bool -> int -> float -> string
bool(-18)
```

Out[ ]:
```
True
```

In [ ]:
```
bool(18) #anything which is not zero is "TRUE" in boolean
```

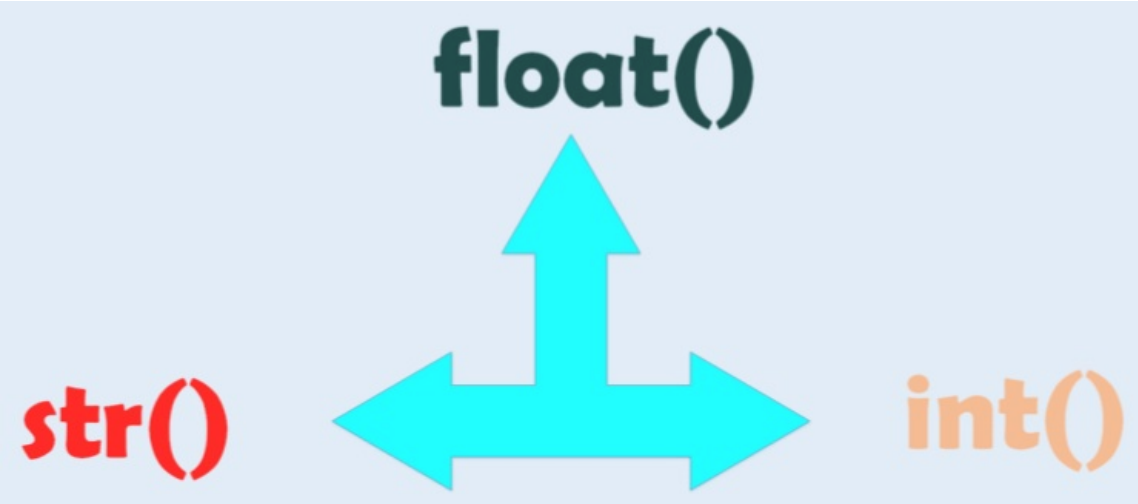
Out[ ]:
```
True
```

In [ ]:
```
name = 'vivek'
```

```
type(name)
```

Out[ ]: str

Type Casting:



The conversion of one data type into the other data type is known as type casting in python or type conversion in python.

In [ ]:
```
3 + 6.5
```

Out[ ]: 9.5

In [ ]:
```
# bool -> int -> float -> str
True + 6 + 7.5
# 1+6+7.5
```

Out[ ]: 14.5

In [ ]:
```
int(7.5) + 3
```

Out[ ]: 10

In [ ]:
```
bool(0)
```

Out[ ]: False

In [ ]:
```
#Auto typecasting
True + 3 + int(4.5)
```

Out[ ]: 8

In [ ]:
```
str(3) + 'vivek'
```

Out[ ]: '3vivek'

In [ ]:
```
#Manual / forced type casting
```

```
4 + float('9.5')
```

Out[ ]: 13.5

In [ ]:
```
int('7') + 5
```

Out[ ]: 12

In [ ]:
```
a = 3.4
type(a)
```

Out[ ]: float

In [ ]:
```
#forced / manual typecasting
int(a)
```

Out[ ]: 3

In [ ]:
```
a= 3
b = 4.5
print(type(a))
print(type(b))
```

```
<class 'int'>
<class 'float'>
```

In [ ]:
```
a + int(b)
```

Out[ ]: 7

In [ ]:
```
3 + int('4')
```

Out[ ]: 7

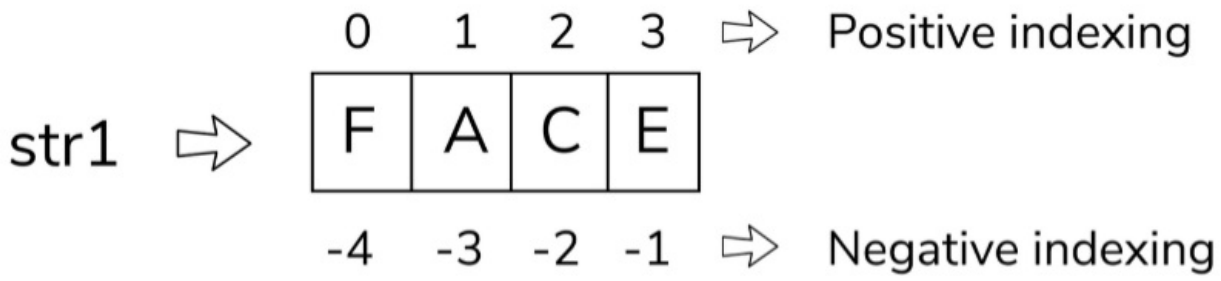
In [ ]:
```
# True -> 1, False -> 0
False + 4
```

Out[ ]: 4

In [ ]:
```
int(3.4 + bool(-20)+int(8.4)) #did you get this
```

Out[ ]: 12

Slicing



str1[1:3] = AC

str1[-3:-1] = AC

Python slice() Function

A slice object is used to specify how to slice a sequence. You can specify where to start the slicing, and where to end. You can also specify the step, which allows you to e.g. slice only every other item.

In [ ]:
```
a = "I am a Data Scientist" #indexing start from 0 from I & we count space as well so, index of a is 2, did you u
a                           # "I=0","space="1","a=2","m=3" & so on.
                           #"t=-1","s=-2" & so on in case of reverse indexing
```

Out[ ]:
```
'I am a Data Scientist'
```

In [ ]:
```
a[2:4]  #it will print a leetr which is on index 2 & 3 excluding index 4
```

Out[ ]:
```
'am'
```

In [ ]:
```
a[-9:]  #if we see index in reverse direction will start from -1
        #so,[-9:] from index -9 it will print all letter including a word at index 9
```

Out[ ]:
```
'Scientist'
```

In [ ]:
```
# : -> slicing operator
a[7:] #starting from index 7(including) it will print till end
```

Out[ ]:
```
'Data Scientist'
```

In [ ]:
```
a[:7] #print everything excluding the letters starting from index 7
```

Out[ ]:
```
'I am a '
```

Math Operators

Python Arithmetic Operators

Arithmetic operators are used with numeric values to perform common mathematical operations:

In [ ]:
```
print(10 + 4) # add (returns 14)
print(10 - 4) # subtract (returns 6)
print(10 * 4) # multiply (returns 40)
print(10**4) # exponent (returns 10000)
print(10 / 4) # divide (returns 2.5)
print(5 % 4) # modulo (returns 1) - also known as the remainder
```

```
14
6
40
10000
2.5
1
```

Logical / Comparision Operators

Logical operators are used to combine conditional statements while Comparison operators are used to compare two values.

In [ ]:
```
# comparisons (these return True)
print(5 > 3 )
print(5 >= 3)
print(5 != 5)
print(5 == 5) # boolean operations (these return True)

# evaluation order: not, and, or
```

```
True
True
False
True
```

Logica Operators

T and T --> T

T and F --> F

F and T --> F

F and F --> F

AND is only True if all are True

T or T --> T

T or F --> T

F or T --> T

F or F --> F

OR is only False if all are False

In [ ]:
```
# Logical operators and or
(5>3) or (10>12)
```

Out[ ]:
```
True
```

In [ ]:
```
print(5 > 3 and 6 < 3 )
print(5 > 3 or 5 < 3 )
```

```
False
```

```
True
```

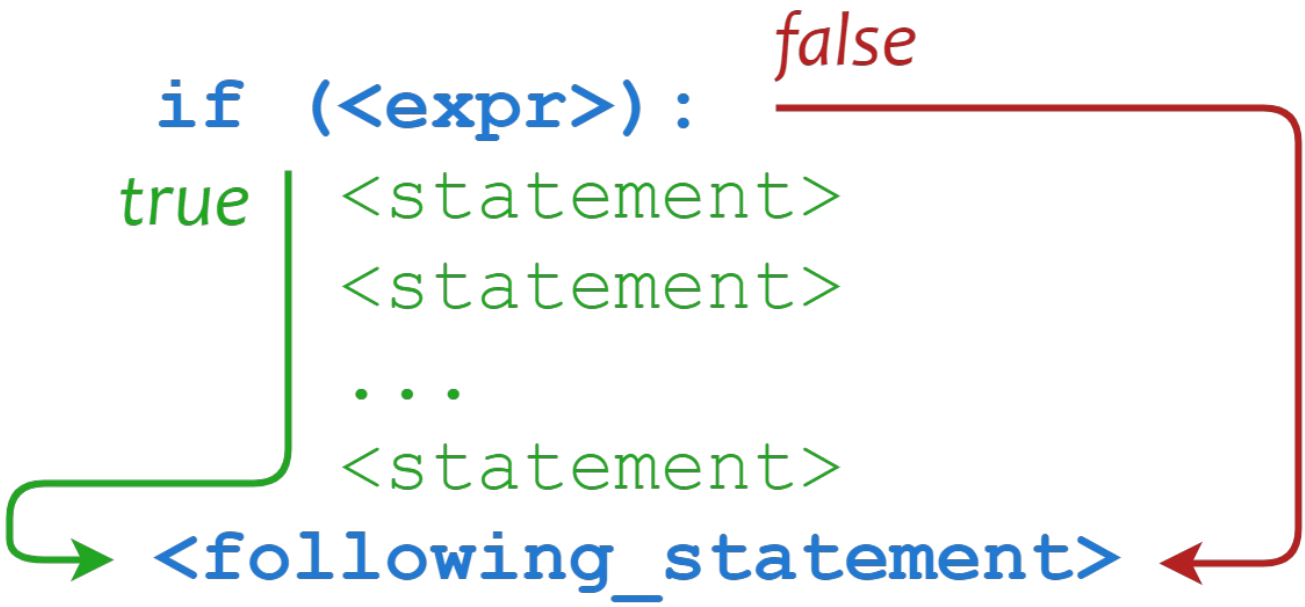
In [ ]:
```
True == bool(-18)
```

Out[ ]:
```
True
```

In [ ]:
```
(5 >= 3 or 6 > 100) and (True == bool(23))
```

Out[ ]:
```
True
```

Conditional Statement



In [ ]:
```
x = 2

if (x>0):
    print("Positive number")
    print("In the IF block")
else:
    print("Negative number")
    print("In the else block")
```

```
Positive number
In the IF block
```

In [ ]:
```
x=5
if (x>0):
    print("X is a positive")
    print("I m if True Block")
else:
    print("X is a Negative")
    print("I m if Else/False Block")

print("I am out of IF-ELSE block")
```

```
X is a positive
I m if True Block
I am out of IF-ELSE block
```

In [ ]:
```
if 5 < 3:
    print("I am in if block")
    print("So the statement is TRUE")
```

```
else:
    print("I am in ELSE block")

print("I am anyway printed, out of IF")
```

```
I am in ELSE block
I am anyway printed, out of IF
```

In [ ]:

```
if (5<3) :
    print("True")
    print("another statement")
else :
    print("False")
    print("another else st")
print("This prints anyway")
```

```
False
another else st
This prints anyway
```

More examples

In [ ]:

```
x=12
if (x>10) :
    print("This is True or IF block")
    print("I am still in IF")
else :
    print("This is else block")

print("\n ---- \n I am out of IF block")
```

```
This is True or IF block
I am still in IF

 ----
 I am out of IF block
```

In [ ]:

```
if (5<3):
    print("This is IF block")
else :
    print("This is Else Block")
```

```
This is Else Block
```

In [ ]:

```
if (5<3) :
    print("True block statement 1")
    print("True block statement 2")
    print("True block statement 3")
else:
    print("False block")
```

```
False block
```

In [ ]:

```
x = 0
if (x > 0) :
    print("X is Positive")

elif (x<0):
    print("X is Negative")
else:
    print("X is ZERO")
```

```
X is ZERO
```

In [ ]:

```
x=-100
```

```
if ((x>0) or (x==-100)):
    print("X is positive Value or -100")
    print("I am if loop")
elif (x<0):
    print("I am in else if block")
    print("X is negative")
else:
    print("X is Zero")

print("I am out of IF looP")
```

```
X is positive Value or -100
I am if loop
I am out of IF looP
```

In [ ]:

```
x = 6

if x%2 == 0 :
    print(x, " is even number")
    print("hello..")
else :
    print(x, " is ODD number")

print("this is out of IF else block")
```

```
6  is even number
hello..
this is out of IF else block
```

In [ ]:

```
x = -20
# if/elif/else statement
if x > 0:
    print('positive')
    print('hello')
elif x == 0:
    print('zero')
else:
    print('negative')
print("I am out of IF block")
```

```
negative
I am out of IF block
```

In [ ]:

```
# single-line if statement (sometimes discouraged)
x=5
if x > 0: print('positive')
```

```
positive
```

Variables and Strings

Variables are used to store values. A string is a series of characters, surrounded by single or double quotes.

In [ ]:

```
# Hello world
print('Hello world')
```

```
Hello world
```

In [ ]:

```
# Hello world with a variable
msg = "Hello world!"
print(msg)
```

```
Hello world!
```

In [ ]:

```
# Concatenation (combining strings)
first_name = 'albert'
last_name = 'einstein'
full_name = first_name +' ' + last_name
print(full_name)
```

```
albert einstein
```

In [ ]:
```
a='hi' # assigning the strig 'hi' to variable a
a
```

Out[ ]:
```
'hi'
```

In [ ]:
```
strn="""Hi,
"How are you" """      # assigning multi_line string to varibale strn
strn
```

Out[ ]:
```
'Hi,\n"How are you" '
```

In [ ]:
```
len(a) # Return the number of characters in a
```

Out[ ]:
```
2
```

In [ ]:
```
c='GoodMorning'
c.startswith("Go") #Test whether c starts with the substring "Go"
```

Out[ ]:
```
True
```

In [ ]:
```
c.endswith("gn") # Test whether c ends with the substring "gn"
```

Out[ ]:
```
False
```

In [ ]:
```
c.replace("i","y") # Return a new string basedon c with all occurances of "i" replaced with "y"
```

Out[ ]:
```
'GoodMornyng'
```

In [ ]:
```
strn.split(" ") # Split the string strn into a list of strings, separating on the character " " and return that l
```

Out[ ]:
```
['Hi,\n"How', 'are', 'you"', '']
```

In [ ]:
```
"{} plus {} plus {} is {}".format(1,2,4,7) # Return the string with the values 3, 1, and 4 inserted
```

Out[ ]:
```
'1 plus 2 plus 4 is 7'
```

In [ ]:
```
c.lower() # Returns a lowercase version of c
```

Out[ ]:
```
'goodmorning'
```

In [ ]:
```
c.upper() # Returns a uppercase version of c
```

Out[ ]:
```
'GOODMORNING'
```

In [ ]:
```
c.title() # Returns c with the first letter of every word capitalized
```

Out[ ]:
```
'Goodmorning'
```

In [ ]:
```
strn.splitlines() # Returns a list by splitting the string on any newline characters.
```

Out[ ]:
```
['Hi,', '"How are you" ']
```

In [ ]:
```
c[:5] # Returns the first 5 characters of s
```

Out[ ]:
```
'GoodM'
```

In [ ]:
```
"Get" + "Lost" # Returns "GetLost"
```

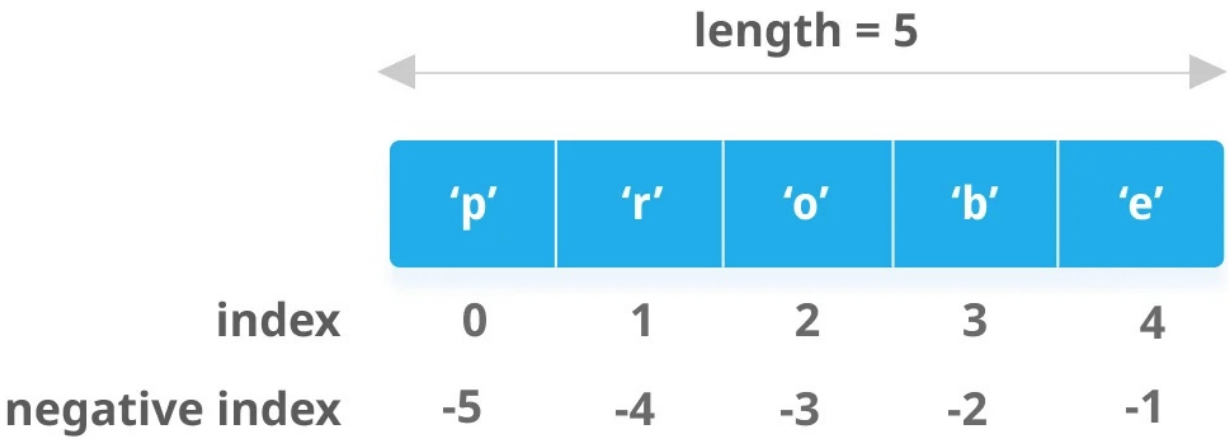
Out[ ]:
```
'GetLost'
```

In [ ]:
```
"Mor" in c # Returns True if the substring "end" is found in c
```

Out[ ]:
```
True
```

Lists

A list stores a series of items in a particular order. You access items using an index, or within a loop.



In [ ]:
```
# Make a list
bikes = ['trek', 'redline', 'giant']
```

In [ ]:
```
# Get the first item in a list
first_bike =bikes[0]
first_bike
```

Out[ ]:
```
'trek'
```

In [ ]:
```
# Get the last item in a list
```

```
last_bike = bikes[-1]
last_bike
```

Out[ ]:
```
'giant'
```

In [ ]:
```
# Looping through a list
for bike in bikes:
    print(bike)
```

```
trek
redline
giant
```

In [ ]:
```
# Adding items to a list
bikes = []
bikes.append('trek')
bikes.append('redline')
bikes.append('giant')
print(bikes)
```

```
['trek', 'redline', 'giant']
```

In [ ]:
```
# Making numerical lists
squares = []
for x in range(1, 11):
    squares.append(x**2)
squares
```

Out[ ]:
```
[1, 4, 9, 16, 25, 36, 49, 64, 81, 100]
```

In [ ]:
```
# List comprehensions
squares = [x**2 for x in range(1, 11)]
squares
```

Out[ ]:
```
[1, 4, 9, 16, 25, 36, 49, 64, 81, 100]
```

In [ ]:
```
# Slicing a list
finishers = ['sam', 'bob', 'ada', 'bea']
first_two = finishers[:2]
first_two
```

Out[ ]:
```
['sam', 'bob']
```

In [ ]:
```
# Copying a list
copy_of_bikes = bikes[:]
copy_of_bikes
```

Out[ ]:
```
['trek', 'redline', 'giant']
```

In [ ]:
```
z=[100,20,30,45,68,54]
# Return the first value in the list z
z[0]
```

Out[ ]:
```
100
```

In [ ]:
```
# Return the last value in the list z
z[-1]
```

Out[ ]:
```
54
```

In [ ]:
```
# Return a slice (list) containing the fourth and fifth values of z
z[3:5]
```

Out[ ]: [45, 68]

In [ ]:
```
len(z) # Return the number of elements in z
```

Out[ ]: 6

In [ ]:
```
Sum=sum(z) # Return the sum of the values of z
print(Sum)
```

317

In [ ]:
```
min(z) # Return the minimum value from a
```

Out[ ]: 20

In [ ]:
```
max(z) # Return the maximum value from a
```

Out[ ]: 100

In [ ]:
```
z.append(21) # Append the value 21 to the end of a
z
```

Out[ ]: [100, 20, 30, 45, 68, 54, 21]

In [ ]:
```
z.sort() # Sort the items of a in ascending order
z
```

Out[ ]: [20, 21, 30, 45, 54, 68, 100]

In [ ]:
```
z.sort(reverse=True) # Sort the items of a in descending order
z
```

Out[ ]: [100, 68, 54, 45, 30, 21, 20]

In [ ]:
```
" ".join(["A","B","C","D"]) # Converts the list["A", "B", "C", "D"] into the string "A B C D"
```

Out[ ]: 'A B C D'

In [ ]:
```
z.pop(3) # Returns the fourth item from a and deletes it from the list
```

Out[ ]: 45

In [ ]:
```
z # 45 got deleted
```

Out[ ]: [100, 68, 54, 30, 21, 20]

In [ ]:
```
z.remove(30) # Removes the first item in a that is equal to 30
```

In [ ]:
```
z # 30 got removed
```

Out[ ]:
```
[100, 68, 54, 21, 20]
```

In [ ]:
```
z.reverse() # Reverses the order of the items in a
z
```

Out[ ]:
```
[20, 21, 54, 68, 100]
```

In [ ]:
```
z[1::2] # Returns every second item from a,commencing from the 1st item
```

Out[ ]:
```
[21, 68]
```

In [ ]:
```
z[-5:] # Returns the last 5 items from a specific axis
```

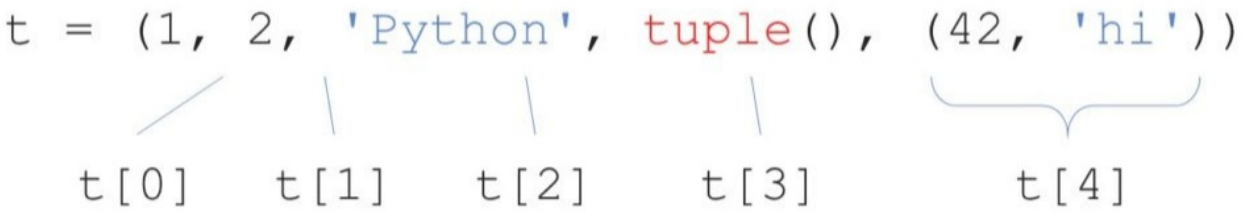
Out[ ]:
```
[20, 21, 54, 68, 100]
```

In [ ]:
```
z[0]=12 # assigning a value to required with its index
z
```

Out[ ]:
```
[12, 21, 54, 68, 100]
```

Tuples

Tuples are similar to lists, but the items in a tuple can't be modified.



In [ ]:
```
# Creating a non empty tuple
tuple='java','anadroid','CSS','HTML'
print(tuple)
```

```
('java', 'anadroid', 'CSS', 'HTML')
```

In [ ]:
```
# Concating 2 tuples
tuple_1='Android','java','HTML'
tuple_2=5,8,6,9
print(tuple_1 + tuple_2)
```

```
('Android', 'java', 'HTML', 5, 8, 6, 9)
```

In [ ]:
```
# repetition
tup=('Hello',)*5
```

```
print(tup)
```

```
('Hello', 'Hello', 'Hello', 'Hello', 'Hello')
```

In [ ]:
```
# Nesting of Tuples
tup_1=('Python','Java','CSS','HTML')
tup_2=(1,5,8,6,7,3)
tup_3=(tup_1,tup_2)
print(tup_3)
```

```
(('Python', 'Java', 'CSS', 'HTML'), (1, 5, 8, 6, 7, 3))
```

In [ ]:
```
# Slicing the tuples
a=(8,3,6,9,45,78,69,12,36)
print(a[1:])
print(a[::-1])
print(a[2:4])
```

```
(3, 6, 9, 45, 78, 69, 12, 36)
(36, 12, 69, 78, 45, 9, 6, 3, 8)
(6, 9)
```

In [ ]:
```
# lenght of tuple
t=('A','B','C','D')
print(len(t))
```

```
4
```

In [ ]:
```
# Tuples in loop
A = ('Hello_World',)
n = 10  #Number of time loop runs
for i in range(int(n)):
    tup = (A,)
    print(tup)
```

```
(('Hello_World',),)
(('Hello_World',),)
(('Hello_World',),)
(('Hello_World',),)
(('Hello_World',),)
(('Hello_World',),)
(('Hello_World',),)
(('Hello_World',),)
(('Hello_World',),)
(('Hello_World',),)
```

In [ ]:
```
# create a tuple
digits1 = (0, 1, 'two',0,1,1,1)  # create a tuple directly
# examine a tuple
print(digits1.count(1))  # counts the number of instances of that value (0) digits.index(1)  # returns the index
len(digits1)
```

```
4
```

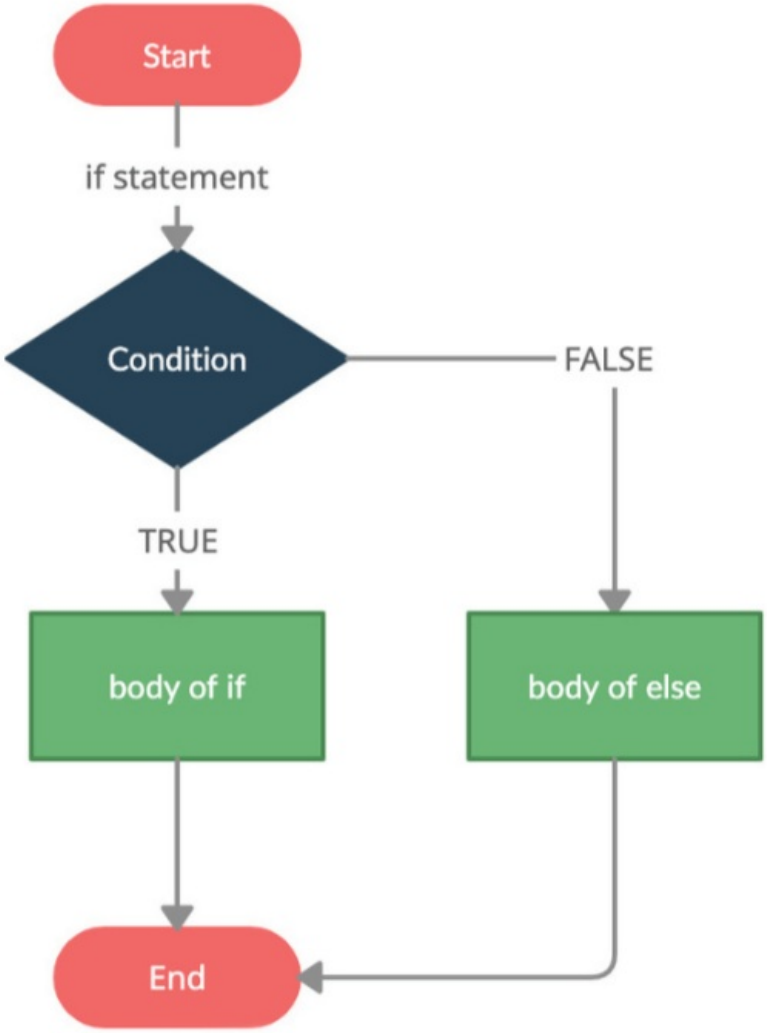
Out[ ]:
```
7
```

In [ ]:
```
# using min(),max()
tuple_A=(5,4,3,2,1,8,7)
tuple_B=('Hello','Hi','Bye','Good Morning')
print('max value in tuple_A and B:'+ str(max(tuple_A)) +  ',' + str(max(tuple_B)))
print('min value in tuple_A and B:'+ str(min(tuple_A))+ ','+ str(min(tuple_B)))
```

```
max value in tuple_A and B:8,Hi
min value in tuple_A and B:1,Bye
```

If Statements

If statements are used to test for particular conditions and respond appropriately.



In [ ]:
```
# if
mark = 10
if (mark > 15):
    print ("mark is less than 15")
print ("I am Not in if")
```

```
I am Not in if
```

In [ ]:
```
# if-else
mark = 20;
if (mark < 15):
    print ("mark is less than 15")
    print ("i'm in if Block")
else:
    print ("mark is greater than 15")
    print ("i'm in else Block")
print ("i'm not in if and not in else Block")
```

```
mark is greater than 15
i'm in else Block
i'm not in if and not in else Block
```

In [ ]:
```
# nested-if
mark = 10
if (mark == 10):
    #  First if statement
    if (mark < 15):
        print ("mark is smaller than 15")
```

```
    # Nested - if statement
    # Will only be executed if statement above
    # it is true
    if (mark < 12):
        print ("mark is smaller than 12 too")
    else:
        print ("mark is greater than 15")
```

```
mark is smaller than 15
mark is smaller than 12 too
```

In [ ]:

```
# if-elif-else ladder
mark = 20
if (mark == 10):
    print ("mark is 10")
elif (mark == 15):
    print ("mark is 15")
elif (mark == 20):
    print ("mark is 20")
else:
    print ("mark is not present")
```

```
mark is 20
```

In [ ]:

```
# A simple if test
age=20
if age >= 18:
    print("You can vote!")
```

```
You can vote!
```

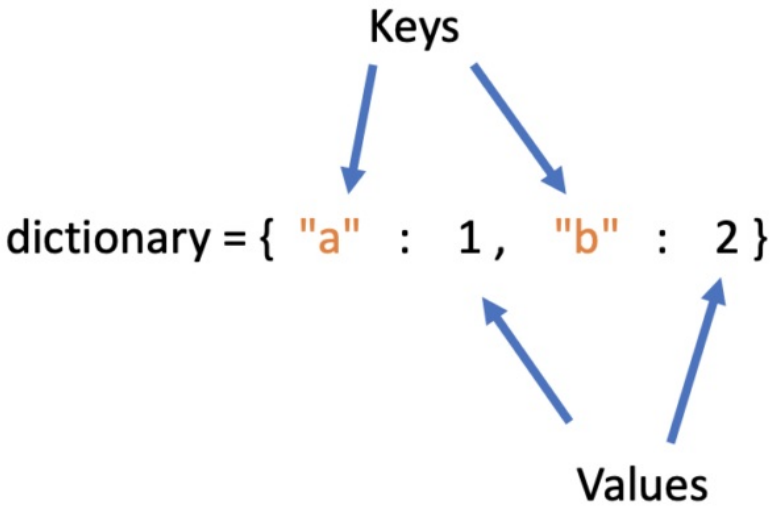
In [ ]:

```
# if-elif-else statements
age=10
if age <=4:
    print('ticket_price = 0')
elif age < 18:
    print('ticket_price = 10')
else:
    print('ticket_price = 15')
```

```
ticket_price = 10
```

Dictionaries

Dictionaries store connections between pieces of information. Each item in a dictionary is a key-value pair.



In [ ]:
```
# A simple dictionary
alien = {'color': 'green', 'points': 5}
```

In [ ]:
```
# Accessing a value
print("The alien's color is " + alien['color'])
```

```
The alien's color is green
```

In [ ]:
```
# Adding a new key-value pair
alien['x_position'] = 0
```

In [ ]:
```
# Looping through all key-value pairs
fav_numbers = {'eric': 17, 'ever': 4}
for name, number in fav_numbers.items():
    print(name + ' loves ' + str(number))
```

```
eric loves 17
ever loves 4
```

In [ ]:
```
# Looping through all keys
fav_numbers = {'eric': 17, 'ever': 4}
for name in fav_numbers.keys():
    print(name + ' loves a number')
```

```
eric loves a number
ever loves a number
```

In [ ]:
```
# Looping through all the values
fav_numbers = {'eric': 17, 'ever': 4}
for number in fav_numbers.values():
    print(str(number) + ' is a favorite')
```

```
17 is a favorite
4 is a favorite
```

In [ ]:
```
# creating a dict with NY,IN,UK as key and their full form as values
dict = {"NY":"New_York","IN":"India","UK":"United_Kingdom"}
```

In [ ]:
```
dict.keys() # Return a list of the keys from dict
```

Out[ ]:
```
dict_keys(['NY', 'IN', 'UK'])
```

In [ ]:
```
dict.values() # Return a list of the values from dict
```

Out[ ]:
```
dict_values(['New_York', 'India', 'United_Kingdom'])
```

In [ ]:
```
dict.items() # Return a list of (key, value) from dict
```

Out[ ]:
```
dict_items([('NY', 'New_York'), ('IN', 'India'), ('UK', 'United_Kingdom')])
```

In [ ]:
```
max(dict, key=dict.get) # Return the key that corresponds to the largest value in dict
```

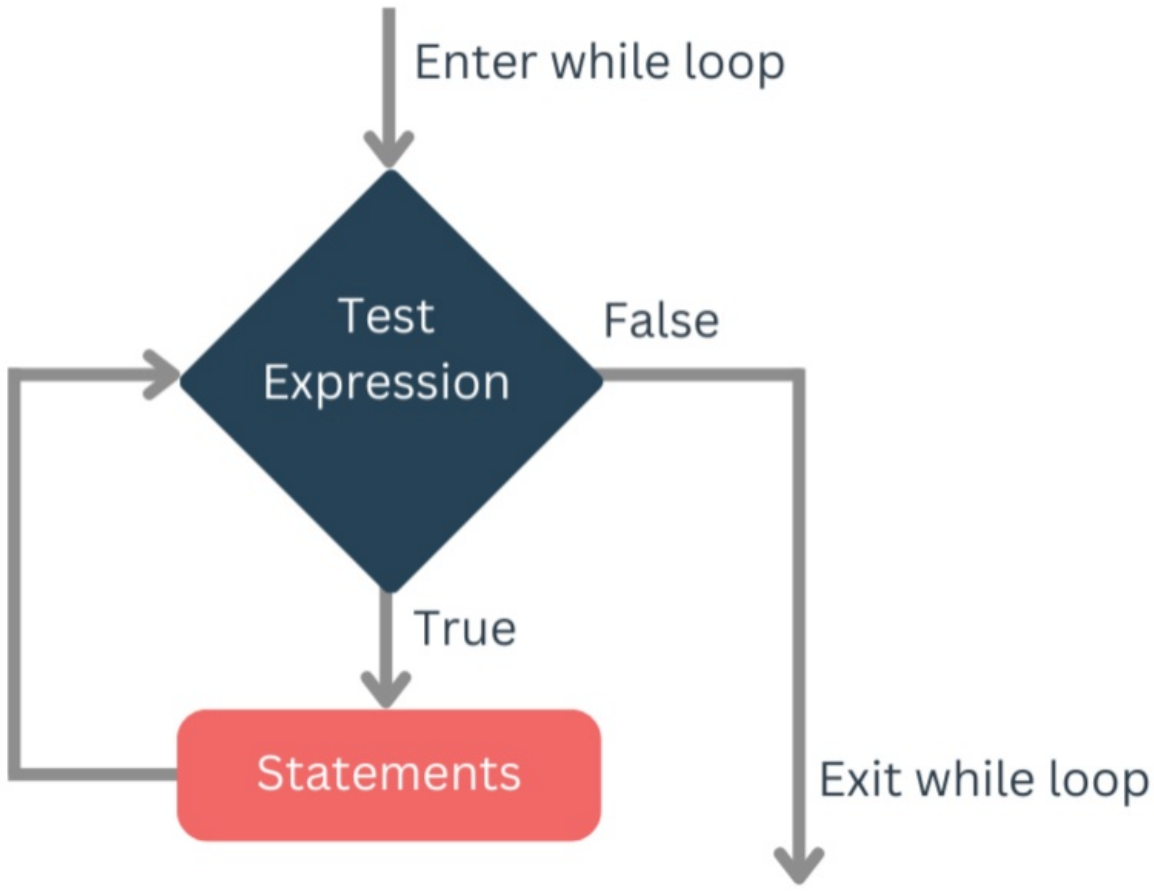
Out[ ]:
```
'UK'
```

In [ ]:
```
min(dict, key=dict.get) # Return the key that corresponds to the smallest value in dict
```

Out[ ]:
```
'IN'
```

While Loop

A while loop repeats a block of code as long as a certain condition is true.



In [ ]:
```
# A simple while loop
current_value = 1
while current_value <= 5:
```

```
    print(current_value)
    current_value += 1
```

```
1
2
3
4
5
```

In [ ]:

```
# Letting the user choose when to quit
msg = ''
while msg != 'quit':
    msg = input("What's your message? ")
    print(msg)
```

```
What's your message? Hi everyone
Hi everyone
What's your message? quit
quit
```

In [ ]:

```
# Single statement while block
count = 0
while (count < 5): count += 1; print("Hello")
```

```
Hello
Hello
Hello
Hello
Hello
```

In [ ]:

```
# loop control statement
# Continue Statement: It returns the control to the beginning of the loop
# Prints all letters except 'w' and 'r'
i = 0
a = 'doyourwork'

while i < len(a):
    if a[i] == 'w' or a[i] == 'sr':
        i += 1
        continue
    print('Current Letter :', a[i])
    i += 1
```

```
Current Letter : d
Current Letter : o
Current Letter : y
Current Letter : o
Current Letter : u
Current Letter : r
Current Letter : o
Current Letter : r
Current Letter : k
```

In [ ]:

```
# break the loop as soon it sees 'e' or 'g'
i = 0
a = 'HappyLearning'

while i < len(a):
    if a[i] == 'e' or a[i] == 'g':
        i += 1
        break
    print('Current Letter :', a[i])
    i += 1
```

```
Current Letter : H
Current Letter : a
Current Letter : p
Current Letter : p
Current Letter : y
Current Letter : L
```

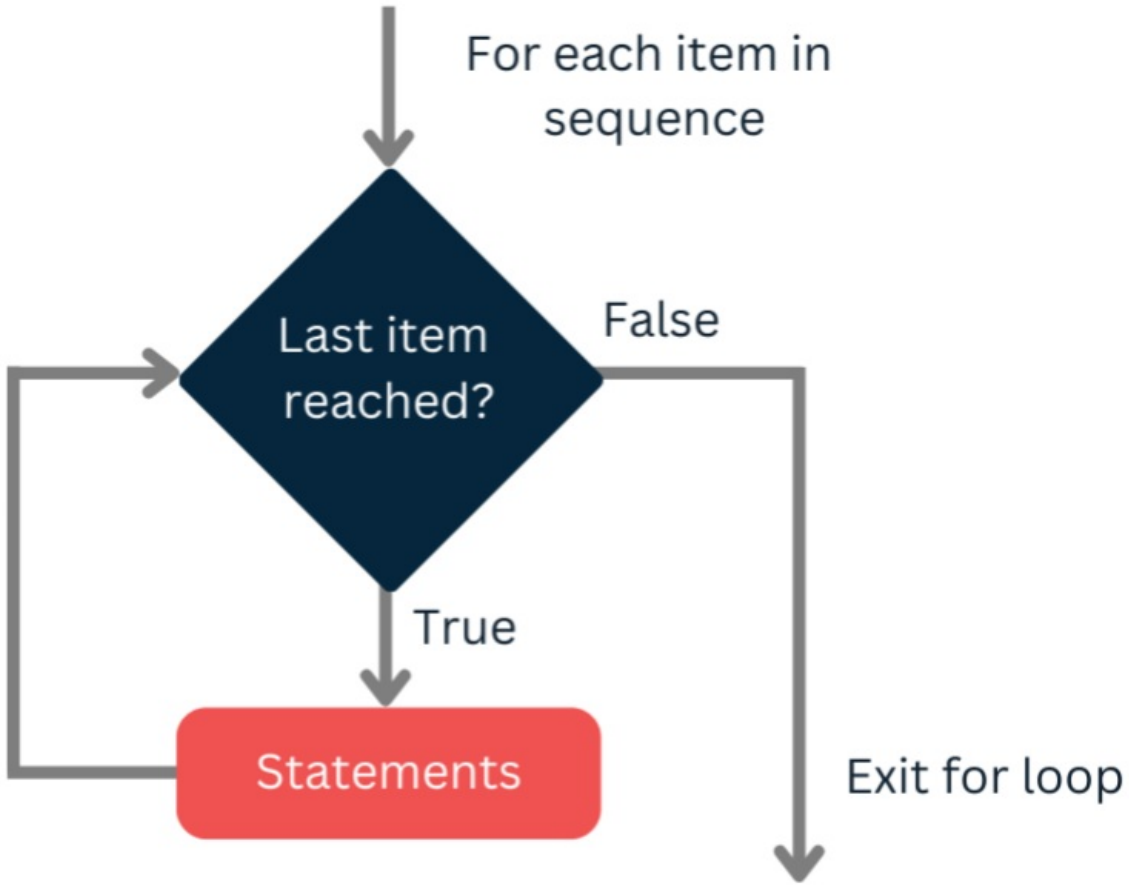
In [ ]:
```
# break the loop as soon it sees 'm' or 'z'
i = 0
a = 'yuweffuygmewedwz'

while i < len(a):
    if a[i] == 'm' or a[i] == 'z':
        i += 1
        break
    print('Current Letter :', a[i])
    i += 1
```

```
Current Letter : y
Current Letter : u
Current Letter : w
Current Letter : e
Current Letter : f
Current Letter : f
Current Letter : u
Current Letter : y
Current Letter : g
```

In [ ]:
```
# while-else loop
i = 0
while i < 8:
    i += 1
    print(i)
else: # Executed because no break in for
    print("No Break\n")

i = 0
while i < 8:
    i += 1
    print(i)
    break
else: # Not executed as there is a break
    print("No Break")
```

```
1
2
3
4
5
6
7
8
No Break

1
```

For Loop



A for loop is used for iterating over a sequence (that is either a list, a tuple, a dictionary, a set, or a string).

In [ ]:
```
# Program to print squares of all numbers present in a list

# List of integer numbers
numbers = [1, 2, 4, 6, 11, 20]

# variable to store the square of each num temporary
sq = 0

# iterating over the given list
for val in numbers:
    # calculating square of each number
    sq = val * val
    # displaying the squares
    print(sq)
```

```
1
4
16
36
121
400
```

In [ ]:
```
# Python for loop example using range() function
# Program to print the sum of first 5 natural numbers

# variable to store the sum
sum = 0

# iterating over natural numbers using range()
for val in range(1, 6):
    # calculating sum
    sum = sum + val

# displaying sum of first 5 natural numbers
print(sum)
```

```
15
```

In [ ]:
```
# For loop with else block
for val in range(5):
    print(val)
else:
    print("The loop has completed execution")
```

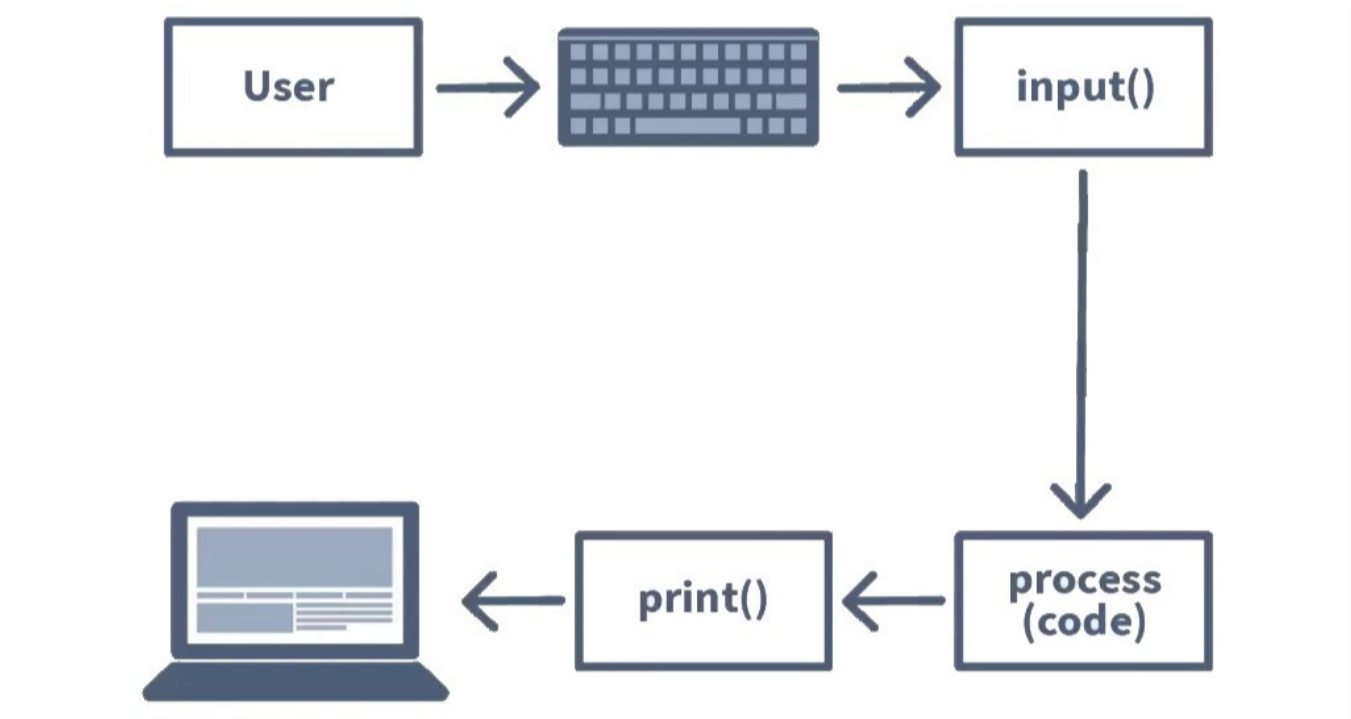
```
0
1
2
3
4
The loop has completed execution
```

In [ ]:
```
#Nested For loop in Python
for num1 in range(3):
    for num2 in range(10, 14):
        print(num1, ",", num2)
```

```
0 , 10
0 , 11
0 , 12
0 , 13
1 , 10
1 , 11
1 , 12
1 , 13
2 , 10
2 , 11
2 , 12
2 , 13
```

User Inputs

Your programs can prompt the user for input. All input is stored as a string.



In [ ]:
```
# Prompting for a value
name = input("What's your name? ")
print("Hello, " + name + "!")
```

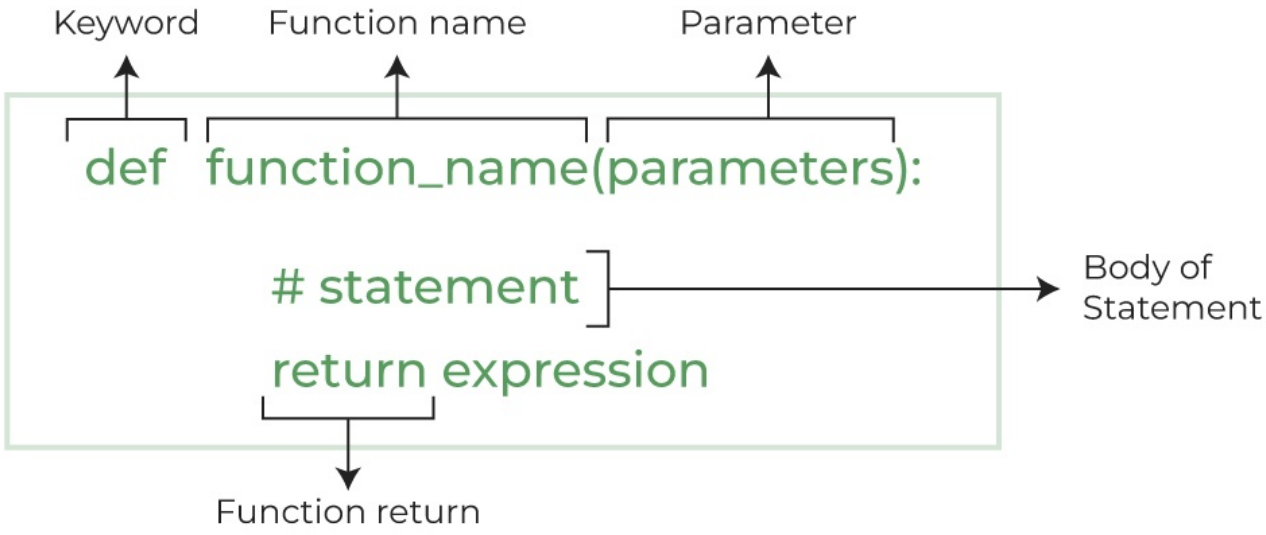
```
What's your name? Vivek
Hello, Vivek!
```

In [ ]:
```
# Prompting for numerical input
```

```
age = input("How old are you? ")
age = int(age)
pi = input("What's the value of pi? ")
pi = float(pi)
```

```
How old are you? 22
What's the value of pi? 3.14
```

Functions



Functions are named blocks of code, designed to do one specific job. Information passed to a function is called an argument, and information received by a function is called a parameter.

In [ ]:
```
#Making a function
def greet_user():
    """Display a simple greeting."""
    print("Hello!")
greet_user()
```

```
Hello!
```

In [ ]:
```
#Passing a single argument
def greet_user(username):
    """Display a simple greeting."""
    print("Hello, " + username + "!")
greet_user('jesse')
greet_user('diana')
greet_user('brandon')
```

```
Hello, jesse!
Hello, diana!
Hello, brandon!
```

In [ ]:
```
#A simple function
def greet_user():
    """Display a simple greeting."""
    print("Hello!")
greet_user()
```

```
Hello!
```

In [ ]:
```
# Passing an argument
def greet_user(username):
    """Display a personalized greeting."""
    print("Hello, " + username + "!")
greet_user('jesse')
```

```
Hello, jesse!
```

In [ ]:
```python
#Default values for parameters
def make_pizza(topping='bacon'):
    """Make a single-topping pizza."""
    print("Have a " + topping + " pizza!")
make_pizza()
make_pizza('pepperoni')
```

```
Have a bacon pizza!
Have a pepperoni pizza!
```

In [ ]:
```python
# Returning a value
def add_numbers(x, y):
    """Add two numbers and return the sum."""
    return x + y
sum = add_numbers(3, 5)
print(sum)
```

```
8
```

In [ ]:
```python
# Using positional arguments
def describe_pet(animal, name):
    """Display information about a pet."""
    print("\nI have a " + animal + ".")
    print("Its name is " + name + ".")
describe_pet('hamster', 'harry')
describe_pet('dog', 'willie')
# Using keyword arguments
def describe_pet(animal, name):
    """Display information about a pet."""
    print("\nI have a " + animal + ".")
    print("Its name is " + name + ".")
describe_pet(animal='hamster', name='harry')
describe_pet(name='willie', animal='dog')
```

```
I have a hamster.
Its name is harry.

I have a dog.
Its name is willie.

I have a hamster.
Its name is harry.

I have a dog.
Its name is willie.
```

In [ ]:
```python
# Using a default value
def describe_pet(name, animal='dog'):
    """Display information about a pet."""
    print("\nI have a " + animal + ".")
    print("Its name is " + name + ".")
describe_pet('harry', 'hamster')
describe_pet('willie')
# Using None to make an argument optional
def describe_pet(animal, name=None):
    """Display information about a pet."""
    print("\nI have a " + animal + ".")
    if name:
        print("Its name is " + name + ".")
describe_pet('hamster', 'harry')
describe_pet('snake')
```

```
I have a hamster.
Its name is harry.

I have a dog.
Its name is willie.

I have a hamster.
Its name is harry.
```

```
I have a snake.
```

In [ ]:

```python
# Using a default value
def describe_pet(name, animal='dog'):
    """Display information about a pet."""
    print("\nI have a " + animal + ".")
    print("Its name is " + name + ".")
describe_pet('harry', 'hamster')
describe_pet('willie')
# Using None to make an argument optional
def describe_pet(animal, name=None):
    """Display information about a pet."""
    print("\nI have a " + animal + ".")
    if name:
        print("Its name is " + name + ".")
describe_pet('hamster', 'harry')
describe_pet('snake')
```

```
I have a hamster.
Its name is harry.

I have a dog.
Its name is willie.

I have a hamster.
Its name is harry.

I have a snake.
```

In [ ]:

```python
#Passing a list as an argument
def greet_users(names):
    """Print a simple greeting to everyone."""
    for name in names:
        msg = "Hello, " + name + "!"
        print(msg)
usernames = ['hannah', 'ty', 'margot']
greet_users(usernames)

#Allowing a function to modify a list
#The following example sends a list of models to a function for
#printing. The original list is emptied, and the second list is filled.
def print_models(unprinted, printed):
    """3d print a set of models."""
    while unprinted:
        current_model = unprinted.pop()
        print("Printing " + current_model)
        printed.append(current_model)
# Store some unprinted designs,
# and print each of them.
unprinted = ['phone case', 'pendant', 'ring']
printed = []
print_models(unprinted, printed)
print("\nUnprinted:", unprinted)
print("Printed:", printed)

#Preventing a function from modifying a list
#The following example is the same as the previous one, except the
#original list is unchanged after calling print_models().
def print_models(unprinted, printed):
    """3d print a set of models."""
    while unprinted:
        current_model = unprinted.pop()
        print("Printing " + current_model)
        printed.append(current_model)
# Store some unprinted designs,
# and print each of them.
original = ['phone case', 'pendant', 'ring']
printed = []
print_models(original[:], printed)
print("\nOriginal:", original)
print("Printed:", printed)
```

```
Hello, hannah!
Hello, ty!
Hello, margot!
Printing ring
Printing pendant
Printing phone case

Unprinted: []
Printed: ['ring', 'pendant', 'phone case']
```

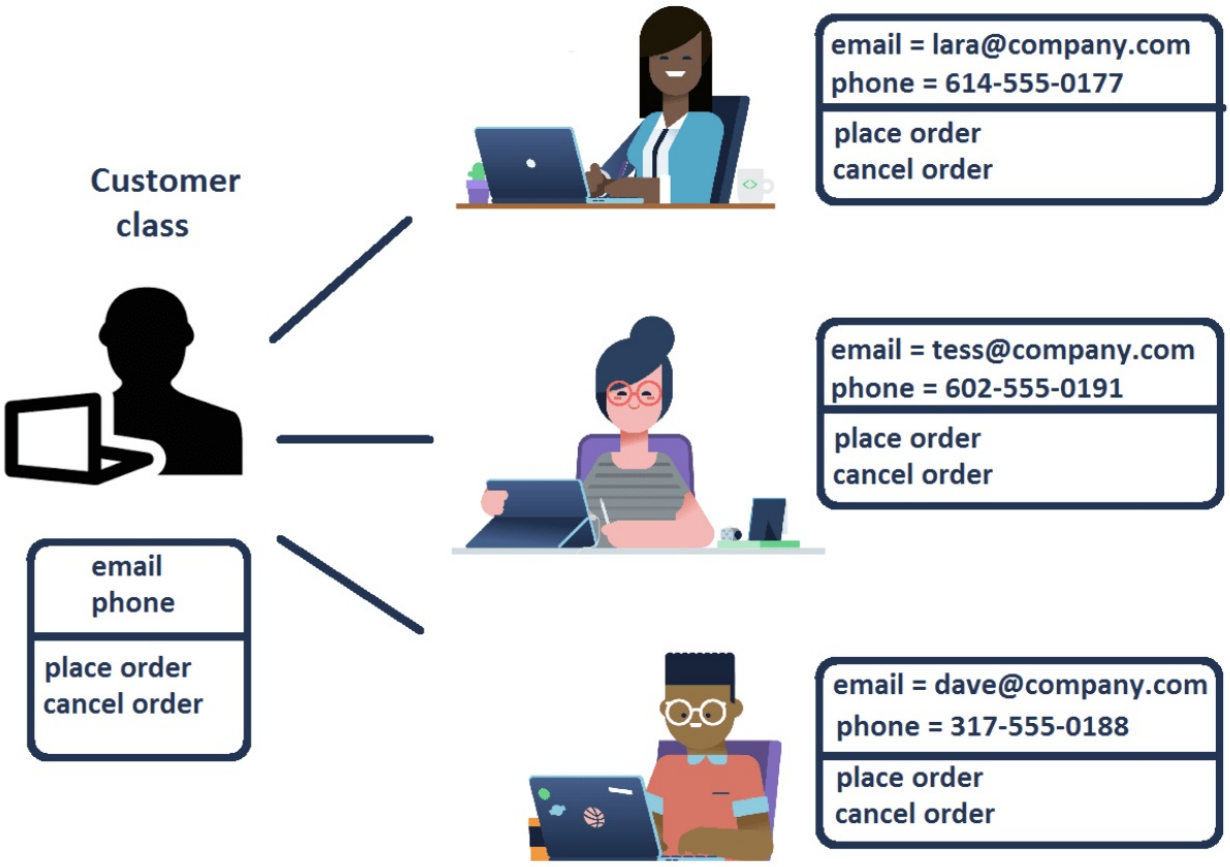
```
Printing ring
Printing pendant
Printing phone case

Original: ['phone case', 'pendant', 'ring']
Printed: ['ring', 'pendant', 'phone case']
```

In [ ]:

```python
#Collecting an arbitrary number of arguments
def make_pizza(size, *toppings):
    """Make a pizza."""
    print("\nMaking a " + size + " pizza.")
    print("Toppings:")
    for topping in toppings:
        print("- " + topping)
# Make three pizzas with different toppings.
make_pizza('small', 'pepperoni')
make_pizza('large', 'bacon bits', 'pineapple')
make_pizza('medium', 'mushrooms', 'peppers','onions', 'extra cheese')

#Collecting an arbitrary number of keyword arguments
def build_profile(first, last, **user_info):
    """Build a user's profile dictionary."""
# Build a dict with the required keys.
    profile = {'first': first, 'last': last}
# Add any other keys and values.
    for key, value in user_info.items():
        profile[key] = value
    return profile
# Create two users with different kinds
# of information.
user_0 = build_profile('albert', 'einstein',
 location='princeton')
user_1 = build_profile('marie', 'curie',
 location='paris', field='chemistry')
print(user_0)
print(user_1)
```

```
Making a small pizza.
Toppings:
- pepperoni

Making a large pizza.
Toppings:
- bacon bits
- pineapple

Making a medium pizza.
Toppings:
- mushrooms
- peppers
- onions
- extra cheese
{'first': 'albert', 'last': 'einstein', 'location': 'princeton'}
{'first': 'marie', 'last': 'curie', 'location': 'paris', 'field': 'chemistry'}
```

Classes



A class defines the behavior of an object and the kind of information an object can store. The information in a class is stored in attributes, and functions that belong to a class are called methods. A child class inherits the attributes and methods from its parent class.

In [ ]:
```
# Creating a dog class
class Dog():
    """Represent a dog."""
    def __init__(self, name):
        """Initialize dog object."""
        self.name = name
    def sit(self):
        """Simulate sitting."""
        print(self.name + " is sitting.")
my_dog = Dog('Peso')
print(my_dog.name + " is a great dog!")
my_dog.sit()
```

```
Peso is a great dog!
Peso is sitting.
```

In [ ]:
```
# Inheritance
class SARDog(Dog):
    """Represent a search dog."""
    def __init__(self, name):
        """Initialize the sardog."""
        super().__init__(name)
    def search(self):
        """Simulate searching."""
        print(self.name + " is searching.")
my_dog = SARDog('Willie')
print(my_dog.name + " is a search dog.")
my_dog.sit()
my_dog.search()
```

```
Willie is a search dog.
Willie is sitting.
Willie is searching.
```

In [ ]:
```
#The Car class
class Car():
```

```
    """A simple attempt to model a car."""

    def __init__(self, make, model, year):
        """Initialize car attributes."""
        self.make = make
        self.model = model
        self.year = year
# Fuel capacity and level in gallons.
        self.fuel_capacity = 15
        self.fuel_level = 0
    def fill_tank(self):
        """Fill gas tank to capacity."""
        self.fuel_level = self.fuel_capacity
        print("Fuel tank is full.")

    def drive(self):
        """Simulate driving."""
        print("The car is moving.")
```

In [ ]:

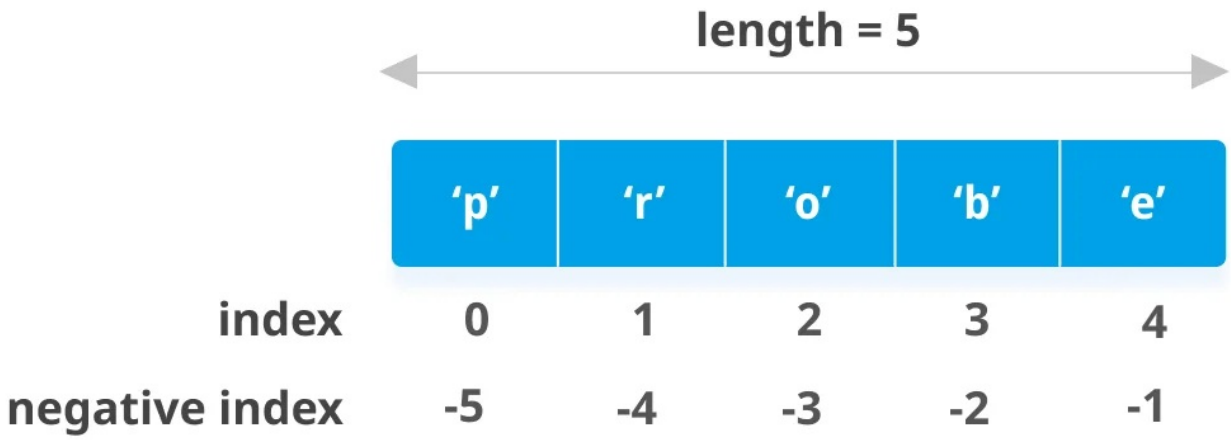
```
#Modifying an attribute directly
my_new_car = Car('audi', 'a4', 2016)
my_new_car.fuel_level = 5
#Writing a method to update an attribute's value
def update_fuel_level(self, new_level):
    """Update the fuel level."""
    if new_level <= self.fuel_capacity:
        self.fuel_level = new_level
    else:
        print("The tank can't hold that much!")
#Writing a method to increment an attribute's value
def add_fuel(self, amount):
    """Add fuel to the tank."""
    if (self.fuel_level + amount<= self.fuel_capacity):
        self.fuel_level += amount
        print("Added fuel.")
    else:
        print("The tank won't hold that much.")
```

In [ ]:

```
#Creating an object from a class
my_car = Car('audi', 'a4', 2016)
#Accessing attribute values
print(my_car.make)
print(my_car.model)
print(my_car.year)
#Calling methods
my_car.fill_tank()
my_car.drive()
#Creating multiple objects
my_car = Car('audi', 'a4', 2016)
my_old_car = Car('subaru', 'outback', 2013)
my_truck = Car('toyota', 'tacoma', 2010)
```

```
audi
a4
2016
Fuel tank is full.
The car is moving.
```

Lists



Lists are used to store multiple items in a single variable.

Lists are one of 4 built-in data types in Python used to store collections of data, the other 3 are Tuple, Set, and Dictionary, all with different qualities and usage.

Lists are created using square brackets:

In [ ]:
```
# Making a list
users = ['val', 'bob', 'mia', 'ron', 'ned']
```

In [ ]:
```
# Getting the first element
first_user = users[0]
print(first_user)
#Getting the second element
second_user = users[1]
print(second_user)
#Getting the last element
newest_user = users[-1]
print(newest_user)
```

```
val
bob
ned
```

In [ ]:
```
# Changing an element
users[0] = 'valerie'
users[-2] = 'ronald'
users
```

Out[ ]:
```
['valerie', 'bob', 'mia', 'ronald', 'ned']
```

In [ ]:
```
# Adding an element to the end of the list
users.append('amy')
print(users)
# Starting with an empty list
users = []
users.append('val')
users.append('bob')
users.append('mia')
print(users)
# Inserting elements at a particular position
users.insert(0, 'joe')
users.insert(3, 'bea')
print(users)
```

```
['valerie', 'bob', 'mia', 'ronald', 'ned', 'amy']
['val', 'bob', 'mia']
['joe', 'val', 'bob', 'bea', 'mia']
```

In [ ]:
```
# Deleting an element by its position
del users[-1]
print(users)
# Removing an item by its value
users.remove('bea')
print(users)
```

```
['joe', 'val', 'bob', 'bea']
['joe', 'val', 'bob']
```

In [ ]:
```
# Pop the last item from a list
most_recent_user = users.pop()
print(most_recent_user)
# Pop the first item in a list
first_user = users.pop(0)
print(first_user)
```

```
bob
joe
```

In [ ]:
```
#Find the length of a list
num_users = len(users)
print("We have " + str(num_users) + " users.")
```

```
We have 1 users.
```

In [ ]:
```
# Sorting a list permanently
users.sort()
# Sorting a list permanently in reverse alphabetical order
users.sort(reverse=True)
# Sorting a list temporarily
print(sorted(users))
print(sorted(users, reverse=True))
# Reversing the order of a list
users.reverse()
```

```
['val']
['val']
```

In [ ]:
```
# Printing all items in a list
for user in users:
    print(user)
#Printing a message for each item, and a separate message afterwards
for user in users:
    print("Welcome, " + user + "!")
print("Welcome, we're glad to see you all!")
```

```
val
Welcome, val!
Welcome, we're glad to see you all!
```

In [ ]:
```
# Printing the numbers 0 to 1000
for number in range(1001):
    print(number)
#Printing the numbers 1 to 1000
for number in range(1, 1001):
    print(number)
#Making a list of numbers from 1 to a million
numbers = list(range(1, 1000001))
numbers

#It will give us output as numbers from 0 to 1000
```

In [ ]:
```
# Finding the minimum value in a list
ages = [93, 99, 66, 17, 85, 1, 35, 82, 2, 77]
youngest = min(ages)
print(youngest)
# Finding the maximum value
ages = [93, 99, 66, 17, 85, 1, 35, 82, 2, 77]
```

```
oldest = max(ages)
print(oldest)
```

```
1
99
```

In [ ]:

```
# Getting the first three items
finishers = ['kai', 'abe', 'ada', 'gus', 'zoe']
first_three = finishers[:3]
print(first_three)
# Getting the middle three items
middle_three = finishers[1:4]
print(middle_three)
# Getting the last three items
last_three = finishers[-3:]
print(last_three)
```

```
['kai', 'abe', 'ada']
['abe', 'ada', 'gus']
['ada', 'gus', 'zoe']
```

In [ ]:

```
# Making a copy of a list
finishers = ['kai', 'abe', 'ada', 'gus', 'zoe']
copy_of_finishers = finishers[:]
print(copy_of_finishers)
```

```
['kai', 'abe', 'ada', 'gus', 'zoe']
```

In [ ]:

```
# Using a loop to generate a list of square numbers
squares = []
for x in range(1, 11):
    square = x**2
    squares.append(square)
print(squares)
# Using a comprehension to generate a list of square numbers
squares = [x**2 for x in range(1, 11)]
print(squares)
# Using a loop to convert a list of names to upper case
names = ['kai', 'abe', 'ada', 'gus', 'zoe']
upper_names = []
for name in names:
    upper_names.append(name.upper())
print(upper_names)
# Using a comprehension to convert a list of names to upper case
names = ['kai', 'abe', 'ada', 'gus', 'zoe']
upper_names = [name.upper() for name in names]
print(upper_names)
```

```
[1, 4, 9, 16, 25, 36, 49, 64, 81, 100]
[1, 4, 9, 16, 25, 36, 49, 64, 81, 100]
['KAI', 'ABE', 'ADA', 'GUS', 'ZOE']
['KAI', 'ABE', 'ADA', 'GUS', 'ZOE']
```

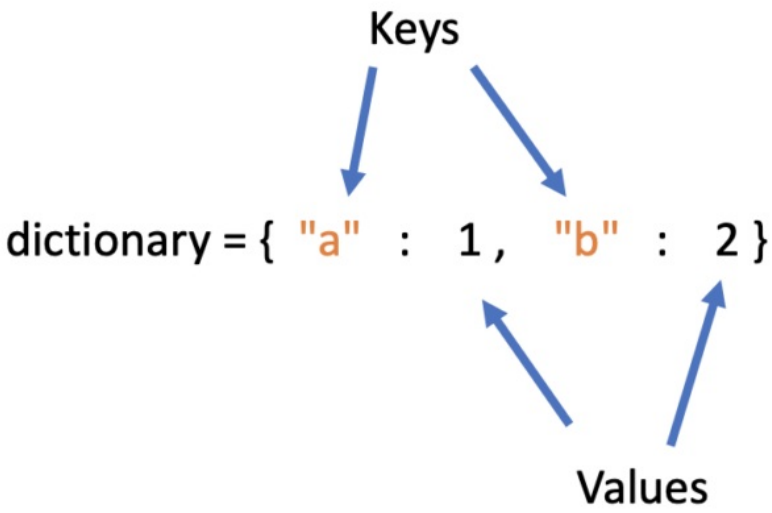
In [ ]:

```
#Build a list and print the items in the list
dogs = []
dogs.append('willie')
dogs.append('hootz')
dogs.append('peso')
dogs.append('goblin')
for dog in dogs:
    print("Hello " + dog + "!")
print("I love these dogs!")
print("\nThese were my first two dogs:")
old_dogs = dogs[:2]
for old_dog in old_dogs:
    print(old_dog)
del dogs[0]
dogs.remove('peso')
print(dogs)
```

```
Hello willie!
Hello hootz!
Hello peso!
Hello goblin!
```

```
I love these dogs!

These were my first two dogs:
willie
hootz
['hootz', 'goblin']
```

Dictionaries



Use curly braces to define a dictionary. Use colons to connect keys and values, and use commas to separate individual key-value pairs.

In [ ]:
```
# Making a dictionary
alien_0 = {'color': 'green', 'points': 5}
```

In [ ]:
```
# Getting the value associated with a key
alien_0 = {'color': 'green', 'points': 5}
print(alien_0['color'])
print(alien_0['points'])
# Getting the value with get()
alien_0 = {'color': 'green'}
alien_color = alien_0.get('color')
alien_points = alien_0.get('points', 0)
print(alien_color)
print(alien_points)
```

```
green
5
green
0
```

In [ ]:
```
# Adding a key-value pair
alien_0 = {'color': 'green', 'points': 5}
alien_0['x'] = 0
alien_0['y'] = 25
alien_0['speed'] = 1.5
print(alien_0)
# Adding to an empty dictionary
alien_0 = {}
alien_0['color'] = 'green'
alien_0['points'] = 5
print(alien_0)
```

```
{'color': 'green', 'points': 5, 'x': 0, 'y': 25, 'speed': 1.5}
{'color': 'green', 'points': 5}
```

In [ ]:
```
#Modifying values in a dictionary
alien_0 = {'color': 'green', 'points': 5}
print(alien_0)
# Change the alien's color and point value.
alien_0['color'] = 'yellow'
alien_0['points'] = 10
print(alien_0)
```

```
{'color': 'green', 'points': 5}
{'color': 'yellow', 'points': 10}
```

In [ ]:
```
# Deleting a key-value pair
alien_0 = {'color': 'green', 'points': 5}
print(alien_0)
del alien_0['points']
print(alien_0)
```

```
{'color': 'green', 'points': 5}
{'color': 'green'}
```

In [ ]:
```
# Looping through all key-value pairs
# Store people's favorite languages.
fav_languages = {
 'jen': 'python',
 'sarah': 'c',
 'edward': 'ruby',
 'phil': 'python',
 }
# Show each person's favorite language.
for name, language in fav_languages.items():
    print(name + ": " + language)
# Looping through all the keys
# Show everyone who's taken the survey.
for name in fav_languages.keys():
    print(name)
# Looping through all the values
# Show all the languages that have been chosen.
for language in fav_languages.values():
    print(language)
# Looping through all the keys in order
# Show each person's favorite language,
# in order by the person's name.
for name in sorted(fav_languages.keys()):
    print(name + ": " + language)
```

```
jen: python
sarah: c
edward: ruby
phil: python
jen
sarah
edward
phil
python
c
ruby
python
edward: python
jen: python
phil: python
sarah: python
```

In [ ]:
```
# Finding a dictionary's length
num_responses = len(fav_languages)
num_responses
```

Out[ ]: 4

In [ ]:
```
#Storing dictionaries in a list
# Start with an empty list.
```

```
users = []
# Make a new user, and add them to the list.
new_user = {
 'last': 'fermi',
 'first': 'enrico',
 'username': 'efermi',
 }
users.append(new_user)
# Make another new user, and add them as well.
new_user = {
 'last': 'curie',
 'first': 'marie',
 'username': 'mcurie',
 }
users.append(new_user)
# Show all information about each user.
for user_dict in users:
    for k, v in user_dict.items():
        print(k + ": " + v)
    print("\n")
```

```
last: fermi
first: enrico
username: efermi


last: curie
first: marie
username: mcurie
```

In [ ]:

```
#You can also define a list of dictionaries directly,
# without using append():
# Define a list of users, where each user
# is represented by a dictionary.
users = [
 {
 'last': 'fermi',
 'first': 'enrico',
 'username': 'efermi',
 },
 {
 'last': 'curie',
 'first': 'marie',
 'username': 'mcurie',
 },
]
# Show all information about each user.
for user_dict in users:
    for k, v in user_dict.items():
        print(k + ": " + v)
    print("\n")
```

```
last: fermi
first: enrico
username: efermi


last: curie
first: marie
username: mcurie
```

In [ ]:

```
# Storing lists in a dictionary
# Store multiple languages for each person.
fav_languages = {
 'jen': ['python', 'ruby'],
 'sarah': ['c'],
 'edward': ['ruby', 'go'],
 'phil': ['python', 'haskell'],
}
# Show all responses for each person.
for name, langs in fav_languages.items():
    print(name + ": ")
    for lang in langs:
        print("- " + lang)
```

```
jen:
```

```
- python
- ruby
sarah:
- c
edward:
- ruby
- go
phil:
- python
- haskell
```

In [ ]:

```
# Storing dictionaries in a dictionary
users = {
 'aeinstein': {
 'first': 'albert',
 'last': 'einstein',
 'location': 'princeton',
 },
 'mcurie': {
 'first': 'marie',
 'last': 'curie',
 'location': 'paris',
 },
 }
for username, user_dict in users.items():
    print("\nUsername: " + username)
    full_name = user_dict['first'] + " "
    full_name += user_dict['last']
    location = user_dict['location']
    print("\tFull name: " + full_name.title())
    print("\tLocation: " + location.title())
```

```
Username: aeinstein
        Full name: Albert Einstein
        Location: Princeton

Username: mcurie
        Full name: Marie Curie
        Location: Paris
```

In [ ]:

```
# Preserving the order of keys and values
from collections import OrderedDict
# Store each person's languages, keeping
# track of who respoded first.
fav_languages = OrderedDict()
fav_languages['jen'] = ['python', 'ruby']
fav_languages['sarah'] = ['c']
fav_languages['edward'] = ['ruby', 'go']
fav_languages['phil'] = ['python', 'haskell']
# Display the results, in the same order they
# were entered.
for name, langs in fav_languages.items():
    print(name + ":")
    for lang in langs:
        print("- " + lang)
```

```
jen:
- python
- ruby
sarah:
- c
edward:
- ruby
- go
phil:
- python
- haskell
```

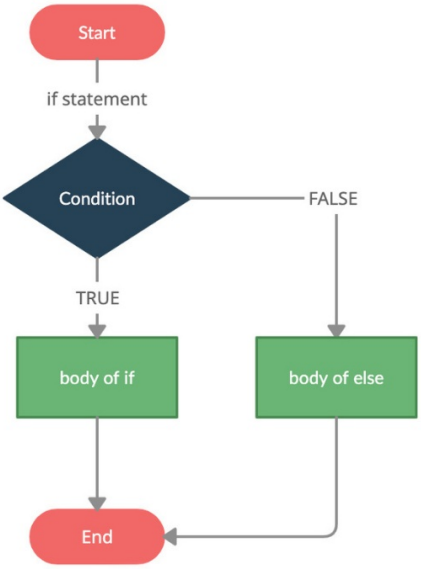
In [ ]:

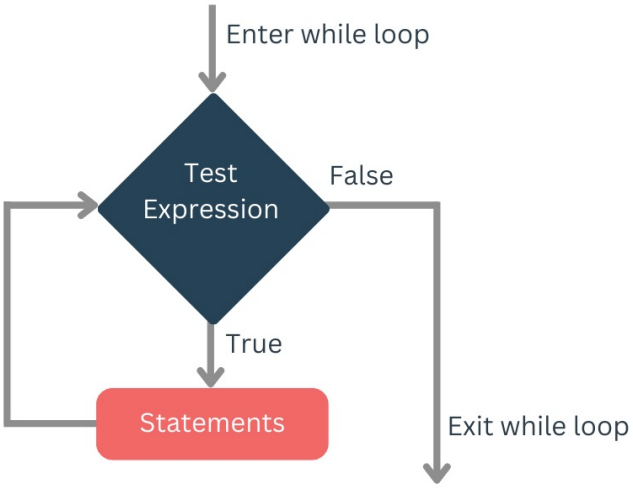
```
# A million aliens
aliens = []
# Make a million green aliens, worth 5 points
# each. Have them all start in one row.
for alien_num in range(1000000):
    new_alien = {}
    new_alien['color'] = 'green'
    new_alien['points'] = 5
    new_alien['x'] = 20 * alien_num
```

```
    new_alien['y'] = 0
    aliens.append(new_alien)
# Prove the list contains a million aliens.
num_aliens = len(aliens)
print("Number of aliens created:")
print(num_aliens)
```

```
Number of aliens created:
1000000
```

If Statements and While Loops





An if statement checks if an expression is true or false, and then runs the code inside the statement only if it is true. The code inside the loop is only run once. A while statement is a loop. Basically, it continues to execute the code in the while statement for however long the expression is true.

In [ ]:
```
#conditional tests
#Checking for equality
#A single equal sign assigns a value to a variable. A double equal
#sign (==) checks whether two values are equal.
car = 'bmw'
print(car == 'bmw')

car = 'audi'
print(car == 'bmw')

# Ignoring case when making a comparison
car = 'Audi'
print(car.lower() == 'audi')

#Checking for inequality
topping = 'mushrooms'
print(topping != 'anchovies')
```

```
True
False
True
True
```

In [ ]:
```
# Testing equality and inequality
age = 18
print(age == 18)
print(age != 18)
```

```
#Comparison operators
age = 19
print(age < 21)
print(age <= 21)
print(age > 21)
print(age >= 21)
```

```
True
False
True
True
False
False
```

In [ ]:

```
#Using and to check multiple conditions
age_0 = 22
age_1 = 18
print(age_0 >= 21 and age_1 >= 21)
age_1 = 23
print(age_0 >= 21 and age_1 >= 21)

#Using or to check multiple conditions
age_0 = 22
age_1 = 18
print(age_0 >= 21 or age_1 >= 21)

age_0 = 18
print(age_0 >= 21 or age_1 >= 21)
```

```
False
True
True
False
```

In [ ]:

```
#Simple boolean values
game_active = True
can_edit = False
```

In [ ]:

```
# if statements
#Simple if statement
age = 19
if age >= 18:
    print("You're old enough to vote!")
# If-else statements
age = 17
if age >= 18:
    print("You're old enough to vote!")
else:
    print("You can't vote yet.")
# The if-elif-else chain
age = 12
if age < 4:
    price = 0
elif age < 18:
    price = 5
else:
    price = 10
print("Your cost is $" + str(price) + ".")
```

```
You're old enough to vote!
You can't vote yet.
Your cost is $5.
```

In [ ]:

```
#Testing if a value is in a list
players = ['al', 'bea', 'cyn', 'dale']
print('al' in players)
print('eric' in players)
```

```
True
False
```

In [ ]:

```
# Testing if a value is not in a list
banned_users = ['ann', 'chad', 'dee']
```

```
user = 'erin'
if user not in banned_users:
    print("You can play!")
# Checking if a list is empty
players = []
if players:
    for player in players:
        print("Player: " + player.title())
else:
    print("We have no players yet!")
```

```
You can play!
We have no players yet!
```

In [ ]:

```
#Simple input
name = input("What's your name? ")
print("Hello, " + name + ".")
#Accepting numerical input
age = input("How old are you? ")
age = int(age)
if age >= 18:
    print("\nYou can vote!")
else:
    print("\nYou can't vote yet.")
```

```
What's your name? vivek
Hello, vivek.
How old are you? 22

You can vote!
```

In [ ]:

```
# While loops
```

In [ ]:

```
# Counting to 5
current_number = 1
while current_number <= 5:
    print(current_number)
    current_number += 1
```

```
1
2
3
4
5
```

In [ ]:

```
#Letting the user choose when to quit
prompt = "\nTell me something, and I'll "
prompt += "repeat it back to you."
prompt += "\nEnter 'quit' to end the program. "
message = ""
while message != 'quit':
    message = input(prompt)
    if message != 'quit':
        print(message)
#Using a flag
prompt = "\nTell me something, and I'll "
prompt += "repeat it back to you."
prompt += "\nEnter 'quit' to end the program. "
active = True
while active:
    message = input(prompt)
    if message == 'quit':
        active = False
    else:
        print(message)
# Using break to exit a loop
prompt = "\nWhat cities have you visited?"
prompt += "\nEnter 'quit' when you're done. "
while True:
    city = input(prompt)
    if city == 'quit':
        break
    else:
        print("I've been to " + city + "!")
```

```
Tell me something, and I'll repeat it back to you.
```

```
Enter 'quit' to end the program. quit

Tell me something, and I'll repeat it back to you.
Enter 'quit' to end the program. quit

What cities have you visited?
Enter 'quit' when you're done. quit
```

In [ ]:

```
#Using continue in a loop
banned_users = ['eve', 'fred', 'gary', 'helen']
prompt = "\nAdd a player to your team."
prompt += "\nEnter 'quit' when you're done. "
players = []
while True:
    player = input(prompt)
    if player == 'quit':
        break
    elif player in banned_users:
        print(player + " is banned!")
    continue
else:
        players.append(player)
        print("\nYour team:")
for player in players:
    print(player)
```

```
Add a player to your team.
Enter 'quit' when you're done. quit
```

In [ ]:

```
#Removing all cats from a list of pets
pets = ['dog', 'cat', 'dog', 'fish', 'cat','rabbit', 'cat']
print(pets)
while 'cat' in pets:
    pets.remove('cat')

print(pets)
```

```
['dog', 'cat', 'dog', 'fish', 'cat', 'rabbit', 'cat']
['dog', 'dog', 'fish', 'rabbit']
```

In [ ]:

```
# An infinite loop
#while True:
#    name = input("\nWho are you? ")
#    print("Nice to meet you, " + name + "!")
```

In [ ]:

In [ ]: