|  |  |  |
| --- | --- | --- |
| **CURRICULUM FOR SUMMER CODING CAMP** | | |
| **STRUCTUERD CURRICULUM ON INTRODUCTION TO HTML AND PYTHON: FOR AGES 8 - 10** | | |
| **INTRODUCTION TO HTML** | | |
| **DURATION** | **TOPICS** | **EVIDENCE OF MASTERY** |
| **DAY 1:**  Introductory Class | * Meaning of HTML * Structure of a webpage * Syntax for writing HTML * Parent-Child Relationship (Indentation) * Create your first webpage | Assignments/Exercise should be given by the tutor after each class to check students understanding. |
| **DAY 2:**  Some specific HTML tags | * Header Tag * Paragraph Tags * Formatting text(bold, italics e.t.c) * Ordered and Unordered lists | Assignments/Exercise should be given by the tutor after each class to check students understanding. |
| **DAY 3:**  Self-Closing Tags | * <Img>, <br>, <hr> Tags * How to add and customize images in your webpage * Hot-linking * Introduction to Forms: The INPUT element | Assignments/Exercise should be given by the tutor after each class to check students understanding. |
| **DAY 4:**  Anchors | * Links: Anchor Tag * How to link top to bottom on a long webpage * How to link two or more webpages * Working with Iframes | Assignments/Exercise should be given by the tutor after each class to check students understanding. |
| **DAY 5:**  Assessment | * Putting it all together: Create a website using all what you have learnt above * How to setup free web hosting (www.ecowebhosting.co.uk) | Assignments/Exercise should be given by the tutor after each class to check students understanding. |
| **INTRODUCTION TO PYTHON** | | |
| **DURATION** | **TOPICS** | **EVIDENCE OF MASTERY** |
| **DAY 6:**  A quick Introduction to Python Programming language | * What is Python Programming Language, Importance and uses. * Download and Installation of Python and Code Editor * Get familiar with the Code Editor environment. * Run basic arithmetic with Python | Assignments/Exercise should be given by the tutor after each class to check students understanding. |
| **DAY 7:**  Fundamentals of Python | * Variables In Python * Strings: Indexing and Slicing, String Methods, print formatting * Booleans | Assignments/Exercise should be given by the tutor after each class to check students understanding. |
| **DAY 8:**  Lists and Loops | * Lists: Indexing and Slicing, List Methods, Nested List * Tuple * Set * Dictionaries * Loops: while loop, Break and Continue Keyword, for loop | Assignments/Exercise should be given by the tutor after each class to check students understanding. |
| **DAY 9:**  Conditional Statements | * If Statement: syntax for simple If statement * If else Statement * If, elif, else Statement | Assignments/Exercise should be given by the tutor after each class to check students understanding. |
| **DAY 10:**  Functions in Python | * Defining a Function: syntax for defining a Function * Calling a function, Passing values * Function Argument, Return Statement * Global and Local Variables | Assignments/Exercise should be given by the tutor after each class to check students understanding. |

**LESSON NOTE**

**TEENS SUMMER CODING CAMP**

**INTRODUCTION TO HTML**

**DAY 1: Introductory Class**

* **Meaning of HTML**

The meaning of the acronym HTML is Hyper Text Markup Language. HTML is the building block of any website. HTML is a **Markup** language, that is, it is a language that uses tags to define elements within a document and it is human readable. You will need a computer and a text editor (this is where you will write your html code) installed and also a web browser (to preview your code on the website). Now you will need to create your first HTML file and save it as ‘index.html’. This is where you will write all your html code.

* **Structure of a Webpage:**

Every webpage built with HTML has the same structure. It begins and ends with a tag containing html element with the **DOCTYPE** which tells that the type of document we are working with is html document. On the next line we see a tag containing **html lang=”en”** wich tells interprets that the type of language to be used on this document is **English.**

Moving to the next line we see the **head**  tag which represents the head of the web page, and with the opening and closing of the **head**  tag we see the **tittle** tag that holds the title of the web page “**Summer Coding Camp**”. If you look closely you will see that every closing tag is preceded by a forward slash (</head>, </tittle>) except for self-closing tags, we will talk about that later. After the head tag is the **body** tag, within the body tag is where we write all the contents that appears on the web page. So if you want anything to appear on your web page it has to be within the body tag. That’s all about the structure of a webpage for now.

<!DOCTYPE html>

<html lang="en">

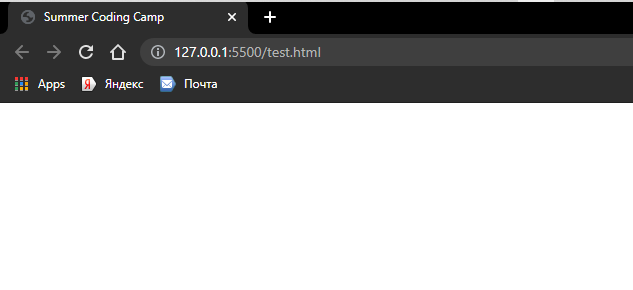
<head>

    <title>Summer Coding Camp</title>

</head>

<body>

</body>

</html>

The title of the webpage

* **Syntax for writing HTML:**

The syntax for writing HTML is very simple:

Start tag end tag

<tag name> content </tag name>

1. **Start tag :** the start tag contains the **tag name** with angle brackets ‘<>’. The Tag name determines what action will be carried out on the contents within the tag.
2. **End tag :** It starts with an opening angle bracket ‘<’ followed by a forward slash ‘/’ then the **tag name**, it ends with a closing angle bracket ‘>’
3. **Content :** this is the text contained in between the **Start** and **End tag.**

* **Parent-Child Relationship (Indentation):**

Indentation is the space at the beginning of a code line. The parent-child relationship is founded on **Indentation.** In the first diagram above, the adult or parent is the **html tag** because it is not indented at all, hence all the content within the **html tag**  is a child of the html tag. So, indentation is what distinguishes between parent and child.

<div class="meta">

                    <ul>

                        <li>

                            <a href="About.html">Articles</a>

                        </li> |

                        <li>

                            <a href="About.html"> web roundofs</a>

                        </li>

                    </ul>

</div>

* **Create your first Webpage:**

Now this is the fun part in today’s class. We create our first webpage.

<!DOCTYPE html>

<html lang="en">

<head>

    <title>Summer Coding Camp</title>

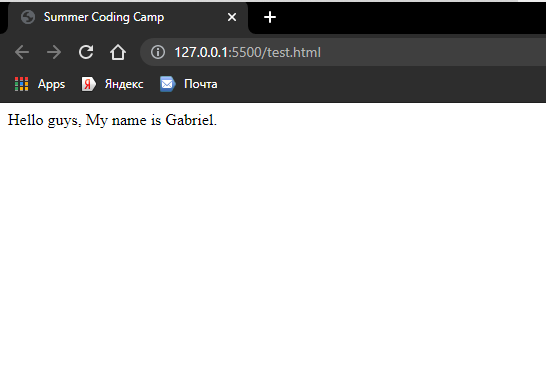
</head>

<body>

    Hello guys, My name is Gabriel.

</body>

</html>

**Output:**

**Day 2: Some specific HTML Tags**

* **Header Tag:**

In HTML, header tags are used to indicate Headings and Subheadings. There are six different types of header tags in HTML and they all differ in font size, they are use based on the choice of the programmer.

<!DOCTYPE html>

<html lang="en">

<head>

    <title>Summer Coding Camp</title>

</head>

<body>

    <h1>Welcome to my first webpage</h1>

    <h2>Welcome to my first webpage</h2>

    <h3>Welcome to my first webpage</h3>

    <h4>Welcome to my first webpage</h4>

    <h5>Welcome to my first webpage</h5>

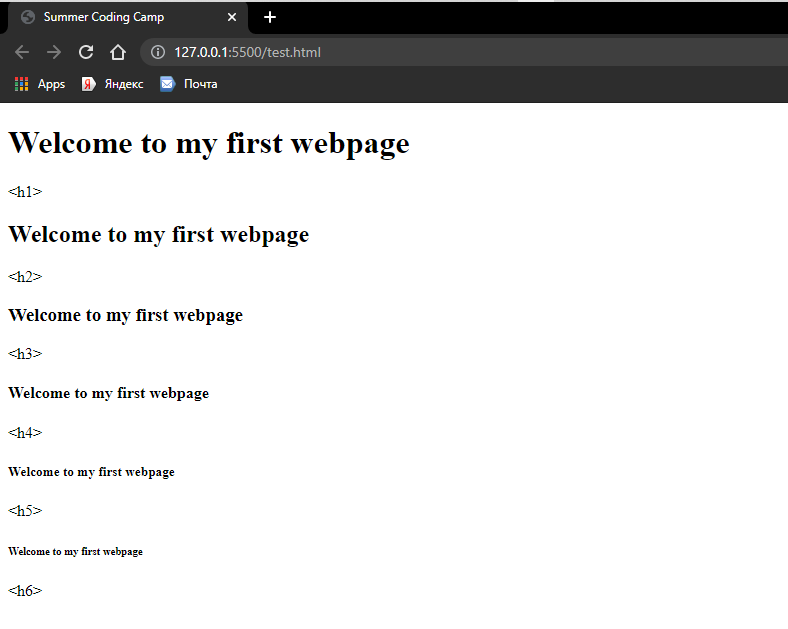
    <h6>Welcome to my first webpage</h6>

</body>

</html>

1. **H1 Tag:**  written as <h1> content </h1>
2. **H2 Tag:**  written as <h2> content </h2>
3. **H3 Tag:**  written as <h3> content </h3>
4. **H4 Tag:**  written as <h4> content </h4>
5. **H5 Tag:**  written as <h5> content </h5>
6. **H6 Tag:**  written as <h6> content </h6>

The output :



* **Paragraph Tag:**

Paragraph Tag as the name implies is used to write a block of text in a new paragraph. The Paragraph Tag does not restyle the text but it helps to start the text within the paragraph tag in a new line. Paragraph Tag is written as:

<p> content </p>

* **Formatting Text:**

Formatting is used to add a little bit of style to some specific text or a block of text on the webpage. Examples include:

1. **Bold/Strong Text:** Written as <b> **content** </b> or <strong> **content** </strong>
2. **Italics/Emphasis Text:**  Written as <i> *content* </i> or <em> *content* </em>
3. **Underline/Inserted Text:** Written as <u> content </u> or <ins> content <>
4. **Superscript Text:** Written as <sup> content </sup>
5. **Subscript Text:** Written as <sub> content </sub>
6. **Strike-Out Text:** Written as <del> ~~content~~ </del>

<!DOCTYPE html>

<html lang="en">

<head>

    <title>Summer Coding Camp</title>

</head>

<body>

    <p>Welcome to my first <b>Bold Text</b> <strong>Bold Text</strong></p>

    <p>Welcome to my first <i>Italized Text</i> <em>Italized Text</em></p>

    <p>Welcome to my first <u>Underlined Text</u> <ins>Underlined Text</ins></p>

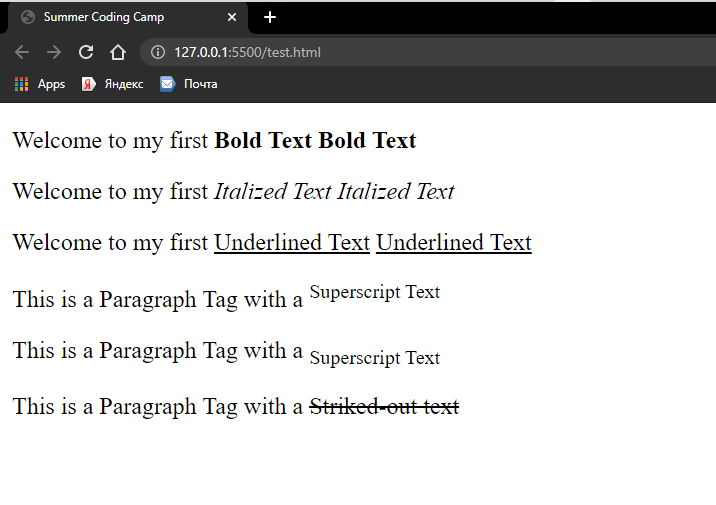
    <p>This is a Paragraph Tag with a <sup>Superscript Text</sup></p>

    <p>This is a Paragraph Tag with a <sub>Superscript Text</sub></p>

    <p>This is a Paragraph Tag with a <del>Striked-out text</del></p>

</body>

</html>

Output:

* **Ordered and Unordered Lists:**

**Ordered List**

An Ordered list in HTML is used to list items in an orderly manner and numbering them. The item is written within a list tag (<li> </li>) and you can add as many items as you want to the list but each item has to be written in a different list tag. Then the list tag is within an ordered list tag (<ol> </ol>). Written as:

<ol>

<li> content </li>

</ol>

<!DOCTYPE html>

<html lang="en">

<head>

    <title>Summer Coding Camp</title>

</head>

<body>

    <p>A list of colors</p>

    <ol>

        <li>blue</li>

        <li>black</li>

        <li>red</li>

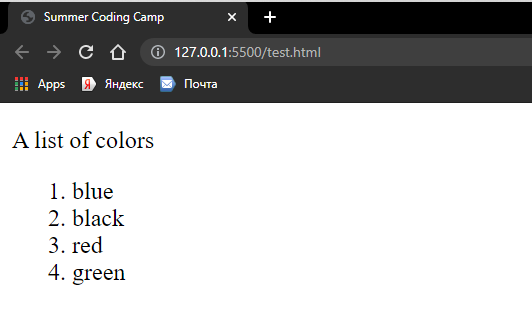
        <li>green</li>

    </ol>

</body>

</html>

Output:



**Unordered List**

An Unordered list in HTML is used to list items without considering the order in which they appear. The item is written within a list tag (<li> </li>) and you can add as many items as you want to the list but each item has to be written in a different list tag. Then the list tag is within an ordered list tag (<ul> </ul>). Written as:

<ul>

<li> content </li>

</ul>

<!DOCTYPE html>

<html lang="en">

<head>

    <title>Summer Coding Camp</title>

</head>

<body>

    <p>A list of colors</p>

    <ul>

        <li>blue</li>

        <li>black</li>

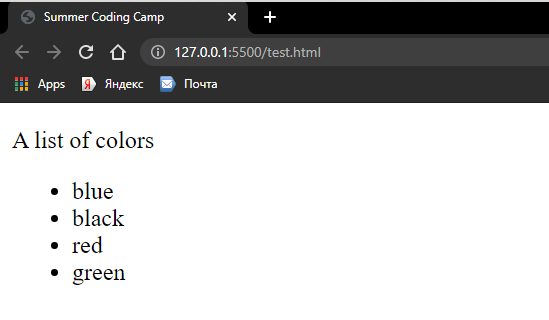
        <li>red</li>

        <li>green</li>

    </ul>

</body>

</html>

Output:

**DAY 3: Self-Closing Tags**

* **<Img>, <br>, <hr> Tags:**

Self-closing Tags are HTML elements that only have a start tag but no end tag. This is because self-closing tags represent void element. They are called void elements because they cannot contain any content. Example include:

* Image tag = <img src=” ”>

It is used to add images to the webpage and the “scr”, it stands for “source” i.e. the location of the image file.

* Line Break Tag = <br>

All the text after the “br” tag are written in a new line.

* Horizontal Tag = <hr>

The “hr” tag draws a horizontal line across the webpage, it is used to separate sections on a webpage.

<!DOCTYPE html>

<html lang="en">

<head>

    <title>Summer Coding Camp</title>

</head>

<body>

    <p>This image was added using an &lt;img tag&gt;</p>

    <img src="img/img\_195.jpg" alt="test-image">

    <p>This is how</p> <br>

    <p>the &lt;br tag&gt; works</p>

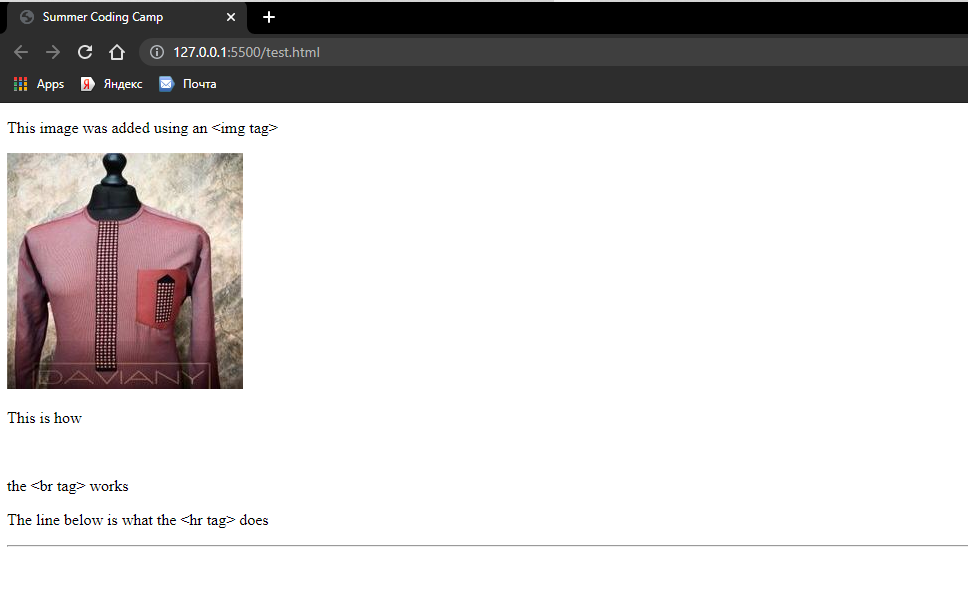
    <p>The line below is what the &lt;hr tag&gt; does</p>

    <hr>

</body>

</html>

Output:

****

* **How to add and customize images in your webpage:**

From the previous topic we added an image to the webpage. But here we will be learning how to add/display and customize the image like, setting the height and width that we want for the image, placing it on the left or right side of the webpage. The syntax for the image tag is:

 <img src="" alt="">

Within the img tag:

* Src=”” : this is where we input the address of the image we want to add, this tells the browser where to get the image from. So make sure you add the address of the image correctly else nothing will display on the web browser. And make sure you specify the correct image extension (e.g. .jpg, .png, .jpeg e.t.c.)
* alt=”” : it is optional to enter any text here, this is where you write an alternative text the should display on the webpage incase the image does not load due to network issues or there about. And it is a best practice to always add an alternative text that is related to the image, so that the user can have an idea of the image if the image does not come up.

Now to customize the image we need to specify the width and height we want for the image. This is very helpful when you are adding an image that is too large on the webpage, hence you might want to specify the size you want. The syntax to do this is:

<!DOCTYPE html>

<html lang="en">

<head>

    <title>Summer Coding Camp</title>

</head>

<body>

    <img src="img/IMG\_7264.JPG.jpg" alt="test-image" width="200px" height="300px">

</body>

</html>

* **Hot Linking:**

Hot linking is used to link image from the web to your webpage, that is, the image you want to display is not on your computer but you copy the address of that image on another website and place it in the “src”, so the browser will copy the image from that webpage and display it on your webpage. But it is bad practice to do Hot linking because the image might be removed from that site and then nothing will be displayed don your webpage, so it is better to download the image to your computer .

* **Introduction to Forms: The INPUT element:**

Here we will look at how we can add forms to our webpage. Forms allow your webpage to be interactive, allowing users to enter data, tick boxes and click buttons. This is achieved by using the “**form tag**”, so everything that will be on the form will be within the form tag. The form tag looks like this:

 <form> content </form>

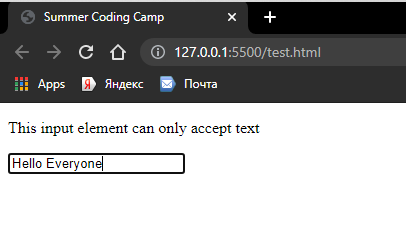
This takes us to the **input** element. The input element provides the space for the use to enter data and it is also a self-closing tag. This is the syntax:

<p>This input element can only accept text</p>

form>

     <input type="text">

</form>

type=”text” tells the browse the type of action the input element will perform or the type of value it will accept. This means that the input element can only accept text.

Output:

We can also add a checkbox to the form, all we need to do is to set the type=”checkbox”. Like this:

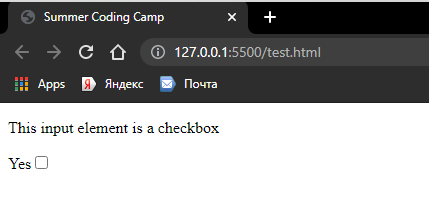
  <p>This input element is a checkbox</p>

    <form>

        <p>Yes<input type="checkbox"></p>

    </form>

Output:



We can also add a radio button to the form, all we need to do is to set the type=”radio”. Like this:

 <p>This input element is a radio button</p>

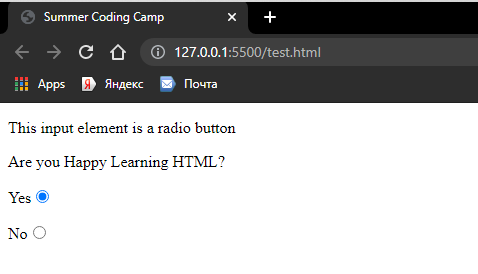
    <p>Are you Happy Learning HTML?</p>

    <form>

        <p>Yes<input type="radio" name="same"></p>

        <p>No<input type="radio" name="same"></p>

    </form>

The name=”same” is used to link the two radio buttons together, telling the browser that the two buttons are working together , hence the user can only select one radio button at a time.

Output:

Next is the **Drop down Element.** This element works differently, it is not an input element but a select element. The list of items to be selected from are written within an **option tag** , a separate tag for each item, all within the **select tag.** As shown below:

 <p>Which is your favorite food?</p>

    <form>

        <select>

         <option>Rice</option>

        <option>Beans</option>

         <option>Pounded Yam</option>

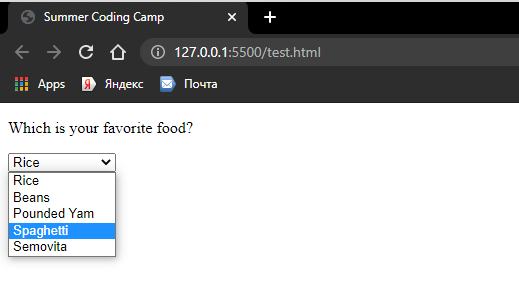
         <option>Spaghetti</option>

         <option>Semovita</option>

        </select>

    </form>

Output:



Finally on this topic is the **Submit Button.** There is always a submit button on every web form. It is after clicking the submit that the data entered is received in and this can be done it two ways:

1. The **Button Tag:** We do this using the **button tag**

 <p>

    <button>Submit</button>

 </p>

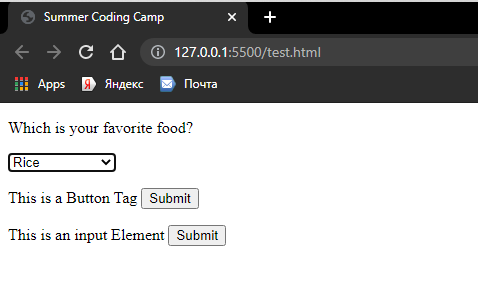
1. **The input element**

<p>

 <input type="submit">

</p>

**Output:**

****

**Day 4: Anchors**

* **Links: Anchor Tags**

Anchor Tags are used to link from one page to another. This is done using the **a Tag** as shown below:

  <a href=""> content </a>

**href=””** is short for Hyperlink Reference, that is were the address of the page or site we want to go to is written. We can also include a text which the user can click to go to the webpage/website. For example, we can link to google.com:

 <a href="https://google.com">Click here to visit google</a>

* **How to link top to bottom on a long webpage:**

This has to do with linking within a page and it is achieved using **“id”** (short for identifier). We use **id** to target a particular tag or a group of tags having the same **id.** For example let us add nine image files to our webpage, now with this number of images on the page the user is going to have to scroll down to get to the bottom of the page. What if the user wants to go straight to the bottom without having to keep scrolling?, to solve that problem we will introduce two buttons to the webpage, one at the top that says “Click To Go down” and another at the bottom of the page that says “Click To Go up”. Now to implement this, we assign an **id** to the first image and to the last image, then we link the buttons with these **id.** As shown below:

 <a href="#down"><button>Click To Go Down</button></a>

    <p id="top"><img src="img/IMG\_7264.JPG.jpg" width="400px" height="600px"></p>

    <p><img src="img/IMG\_7264.JPG.jpg" alt="test-image" width="400px" height="600px"></p>

    <p><img src="img/IMG\_7264.JPG.jpg" alt="test-image" width="400px" height="600px"></p>

    <p><img src="img/IMG\_7264.JPG.jpg" alt="test-image" width="400px" height="600px"></p>

    <p><img src="img/IMG\_7264.JPG.jpg" alt="test-image" width="400px" height="600px"></p>

    <p><img src="img/IMG\_7264.JPG.jpg" alt="test-image" width="400px" height="600px"></p>

    <p><img src="img/IMG\_7264.JPG.jpg" alt="test-image" width="400px" height="600px"></p>

    <p><img src="img/IMG\_7264.JPG.jpg" alt="test-image" width="400px" height="600px"></p>

    <p id="down"><img src="img/IMG\_7264.JPG.jpg" width="400px" height="600px"></p>

    <a href="#top"><button>Click To Go Up</button></a>

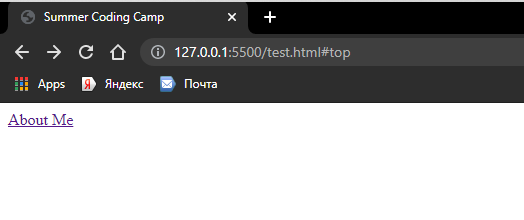
The ‘#’ before the **id name**  is to tell the browser that the hyperlink is an **id.**

* **How to link two or more webpages together:**

In the previous class you were introduced to Anchor tags and how they work. To link webpages together is very easy and one of the ways to do this is by linking to a webpage you created on your computer. First you need to create a new html file (you will be linking to this new file) give it a name of your choice, let’s say ‘about.html’. Now go back to your **index.html** file. To link your about page to the home page you use an **Anchor tag** then input the name of the page you want to link in the **href** space, then let the content read “About Me” like this:

<a href="About.html">About Me</a>

Output:



Another way to link pages is by Hot linking too, let’s say you want to link to Google, all you need to do is write the address to Google website in the Hyperlink Reference (href=””).

* **Working with Iframes:**

Iframes are used to display contents from other sites that can be readily seen on your website (e.g Facebook, YouTube e.t.c.) and you can interact with the website from your website. This is done using the **iframe tag** :

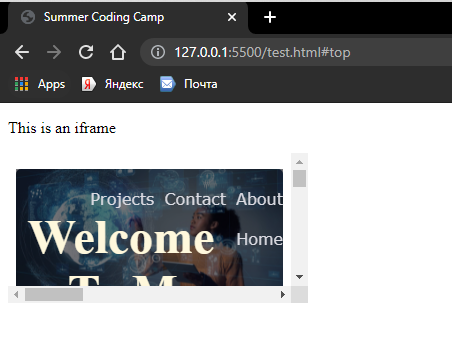
<iframe src="" frameborder="0"></iframe>

src=””, the source is where you input the correct address of the website you want to display on your webpage.

frameboder=”0”, this where you specify the size of the border of the iframe. For instance:

<p>This is an iframe</p>

<iframe src="home.html" frameborder="0"></iframe>

Output:

Note: You can set the iframe to a width and height of your choice.

**Day 5: Assessment: Putting it all together**

* **Create a website using all what you were taught from Day 1.**
* **How to set up free webhosting (www.ecowebhosting.co.uk)**

This will be a practical class, the tutor will walk you through the process.

**INTRODUCTION TO PYTHON**

**Day 6: A quick Introduction to Python Programming language.**

* **What is Python Programming Language, it’s importance and uses?**

Unlike HTML, Python is a high level programming language that is executed by an interpreter, it is the world’s fastest growing programming language and most popular programming language not just among computer scientist and software developers but also Engineers, Mathematicians, Data Analysts, accountants, Scientists, Network Engineers and even kids. In fact it is the ideal programming language to learn first. But what makes python so **important**?, here are some reasons:

* With python you can solve complex problems in less time and with fewer lines of code than many other programming languages.
* Python is a multipurpose language and can be used for a wide range of jobs.
* Python is a cross-platform programming language, i.e. you can build and run python apps on Windows, Mac and Linux operating systems.
* Python has a huge community, so whenever you get stuck there is someone out there to help, all you nee to do is ask.
* Python has a large ecosystem of frameworks, libraries and tools, so whatever you want to do it is likely that somebody has done it before.

**Uses**

* Python is used by huge companies like Google, Spotify, Facebook.
* You can use it for a wide range of jobs like Data Analysis, Artificial Intelligence/Machine Leaning, writing Automation Scripts, Building Web applications, Mobile applications, Desktop applications, Hacking.
* **Download and installation of python and Code Editor:**

**How to download and install python**

* First open your web browser and type the web address <https://python.org> to visit the official website for python
* On the website go under the **Download** tab, you will see the latest version of python, go ahead and download python for you operating system (Windows, Mac, Linux).
* After downloading python, go ahead and RUN the execution file.
* Make sure you “**add python to path**”.
* Install python following the Recommended settings.

**How to download and install python Code Editor**

When it comes to typing python code we make use of code editors or IDEs (Integrated Development Environment). But for this class we will be making use of a code editor called **VSCODE (**visual studio code**).** A code editor is a text editor program designed specifically for editing source code of computer programs.

So, to download vscode head over to **code.visualstudio.com** to download the latest version of visual studio code.

* **Getting familiar with the Vscode Environment:**

After downloading vscode, open the desktop app. On the top, from the **File menu,** click **Open Folder** to create a new folder on your computer (maybe under Documents), you can name the folder ”**HelloWorld**” and then open it. On the panel at the left side of the vscode environment, click on the first icon named **Explorer**  to see the current directory you are working on, this consist of all the files and the folder you are working on.

Now let’s add a new file and call it **test.py,** to do this, click on the **Explorer** and inside the **HelloWorld** folder create the python file. **.py** is the file extension for all your python, any file without the .py file extension will not be read as a python file by the text editor. Now, since we have created a folder and a python fine, we are all set to start writing and executing our python code.

* **Run basic arithmetic with python:**

You can use python to perform various arithmetic operations (addition, subtraction, multiplication, division, square root, exponential, e.t.c.). We are going to be using one of the built-in functions of python called “**print()**” and we can use this to print/display our result on the screen. Whenever you want to use a function make sure you open and close parenthesis “()”, in programming we say you are calling the **print** function. Calling a function means executing it. Let’s perform some arithmetic operations in python to get our legs wet on code:

print(2+2)

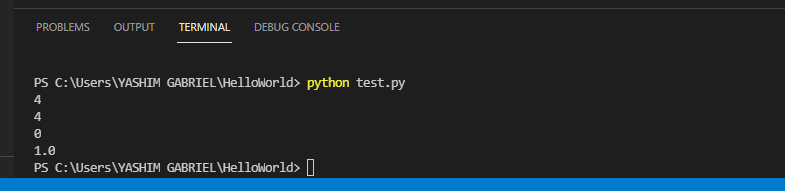
print(2\*2)

print(2-2)

print(2/2)

Note: after writing your code you must save using **“ctrl + s”** before you executer the code. To execute the code you need to make use of the integrated terminal, press **“ctrl + ` ”,** then on the integrated terminal type “**python test.py**” and it executes the code.

Output:



So you can perform any type of arithmetic operation on python.

**Day 7: Fundamentals of python**

* **Variables in python:**

Variable is a reserved memory location to store values. Variables are like containers that stores/hold values that we can make use of anytime in the program all we need to do is to call the variable name. The content of a variable can be changed or another value can be reassigned to the variable. We give variable name to a variable so that we can call it whenever we need to make use of it. There are some rules for declaring a variable, they are:

1. All variable names must begin with a letter of the alphabet or an underscore. It is bad practice to begin a variable name with special characters or numbers (e.g. #,%,\*,2 ,4 e.t.c.)
2. After the first letter, variable names can contain numbers.
3. There must be no space within a variable name, if you must include a space you can make use of an underscore (e.g. my\_name) or you can make use of camel case to distinguish the words in a variable name (e.g. myName)
4. When naming a variable, the variable name should be related to the kind of variable you are storing (e.g Age = 18).
5. Your variable name must be simple and short as possible (stay within the range of 1 to 31 characters).
6. Python is case sensitive when it comes to variable names, that is, Name is not the same as name.
7. You cannot use a keyword or a reserved word as a variable name. Keyword is a word reserved by the program because it has a special meaning (e.g. if, as, in, True, e.t.c.)

Below are some example of how to declare variables and use them (using **print function**) in python:

# Declara a variable to sto your age

Age = 18

print(Age)

Output:

PS C:\Users\YASHIM GABRIEL\HelloWorld> & C:/Python39/python.exe "c:/Users/YASHIM GABRIEL/HelloWorld/test.py"

18

# Declara a variable to store your name

Name = "Yashim Gabriel"

print(Name)

Output:

PS C:\Users\YASHIM GABRIEL\HelloWorld> & C:/Python39/python.exe "c:/Users/YASHIM GABRIEL/HelloWorld/test.py"

Yashim Gabriel

**Class work:** Assign values to a variable and use them to perform arithmetic operations.

Before you start declaring variable name you need to know the data types in python. They include:

* **String**
* **Integer**
* **Float**
* **Boolean**

Next we will be looking at some data types in python.

* **Strings: Indexing and Slicing, String Methods, print formatting:**

**A string is an ordered sequence of characters**. Two keywords here, **ordered** and **characters.** Ordered means that we will be able to use indexing and slicing to grab elements from the string.

**How to create strings in python:**

In python you write as string or sets of strings within single or double quotes (“ ” or ‘ ’). **Example:**

“Welcome To Teens Coding Camp” or ‘Welcome To Teens Coding Camp’

To print a string:

print('Welcome To Teens Coding Camp')

Output:

PS C:\Users\YASHIM GABRIEL\HelloWorld> python -u "c:\Users\YASHIM GABRIEL\HelloWorld\test.py"

Welcome To Teens Coding Camp

print('this is a new line \n notice how this is on a new line')

print("Python Code")

print('this is a tab \t notice how this prints with space between')

Output:

PS C:\Users\YASHIM GABRIEL\HelloWorld> python -u "c:\Users\YASHIM GABRIEL\HelloWorld\test.py"

this is a new line

notice how this is on a new line

Python Code

this is a tab notice how this prints with space between

**Comments:** these are lines of code that are not executed by the interpreter in runtime, they are text written by the programmer to explain the code, it is optional to use comment, but it is best practice to use comments in your code. A comment is preceded by a hash tag “#”, For example

# Declare a variable to store your name

# Name = "Yashim Gabreil"

# print(Name)

**Indexing and Slicing:**

**Indexing:** this means to select a single character in a string. Indexing starts at 0 in python, that is python starts counting a string or group of strings from 0 not 1. Example:

**character: h e l l o**

**index: 0 1 2 3 4**

EXAMPLES:

> word = "hello"

> print(word)

**Output:** hello

> word[0]

**Output: ‘**h’

> word[3]

**Output: ‘**l’

Python also supports reverse indexing:

**character: h e l l o**

**index: 0 1 2 3 4**

**reverse index: 0 -4 -3 -2 -1**

Reverse indexing is used commonly to grab the last "chunk" of a sequence.

> word[-2]

**Output: ‘**l’

**Slicing:** this means to grab a sub-section of a string.

We can grab entire sub-section of a string with *slice* notation.

This is the syntax:

[start:stop:step]

Key things to note:

1. The starting index direclty corresponds to where your slice will start
2. The stop index corresponds to where your slice will go up to. **It does not include this index character!**
3. The step size is how many characters you skip as you go grab the next one.

Let's see some examples

> alpha = 'abcdef'

# NOTICE HOW d IS NOT INCLUDED!

> alpha[0:3]

**Output: ‘**abc**’**

> alpha[0:4]

**Output: ‘**abcd**’**

> alpha[2:4]

**Output: ‘**cd**’**

> alpha[2:]

**Output: ‘**cdef**’**

> alpha[:2]

**Output: ‘**ab**’**

> alpha[0:6:2]

**Output: ‘**ace**’**

**String Methods:**

Basically methods are actions you can call off an object. A string method is written as:

**variableName.methodName()**

Take note of the open and close parenthesis when calling a method in python. String has many methods, we will be making use of a few.

EXAMPLE:

> basic = "hello world I am still a beginner pythonista"

> basic.upper() #This method will make the whole string uppercase

**Output: ‘**HELLO WORLD I AM STILL A BEGINNER PYTHONISTA**’**

> basic.lower() #This method will make the whole string lowercase

**Output: ‘**hello world i am still a beginner pythonista**’**

> basic.split() #This method will split the string by each word and put it in a list. We will learn list later on.

**Output:** [**‘**hello’, ‘world’, ‘I’, ‘am’, ‘still’, ‘a’, ‘beginner’, ‘pythonista**’**]

> basic.split(‘i’) #This method will split the string by letter ‘i’, not ‘ I’

**Output:** [**‘**hello world I am st’, ‘ll a beg’, ’nner python’, ’sta**’**]

**Print formatting:**

You can use the **.format()** method off a string, to perform what is formally known as **string interpolation**, essentially inserting variables when printing a string.

EXAMPLE:

> user\_name = "Newbie"

> password = 12345

> print("Welcome {} and your password is {}".format(user\_name, password))

**Output:** welcome Newbie and your password is 12345

> action = 'learn'

> print("The {} needs to {}".format(user\_name,action))

**Output:** The Newbie needs to learn

> print("The {a} needs to {b}".format(a=user\_name,b=action))

**Output:** The Newbie needs to learn

> print("The {b} needs to {a}".format(a=user\_name,b=action))

**Output:** The learn needs to Newbie

* **Booleans:**

Booleans are data types that indicate a logical state of **True** or **False**. Python also has a placeholder object called **None**. Let's explore what these look like. We will work with them a lot more once we begin to learn about Conditional Statements in Python, but until then, let's just get to understand what they look like (notice the syntax highlighting).

EXAMPLE:

> a = True

> print(a)

**Output:** True

> type(a) # this is used to check for the data type of a variable

**Output:** bool

> b = False

**Output:** bool

> 1 > 2

**Output:** False

**Day 8: Lists and Loops**

* **Lists: Indexing and Slicing, List Methods, Nested List:**

Similarly, lists are sequences of objects, they can hold a variety of data types in order, and they follow the same sequence and indexing bracket rules like strings. They can also take in mixed data types. Some examples include:

> alist = [] #In python a list is defined by placing all the elements in a square bracket

> type(alist)

**Output:** liist

> my\_list = [1,2,3]

> print(my\_list)

**Output:** [1,2,3]

> my\_list2 = [‘a’, ’b’, ’c’, 1, 2, 3.2, True, “true”] #This is to show that a list can hold different data types

> print(my\_list2)

**Output:** [‘a’, ’b’, ’c’, 1, 2, 3.2, True, “true”]

> a = 100

> b = 200

> c = 300

> my\_list3 = [a, b, c]]

> print(my\_list3)

> print(my\_list2)

**Output:** [100, 200, 300]

[‘a’, ’b’, ’c’, 1, 2, 3.2, True]

**Indexing and Slicing**

Indexing and Slicing works exactly the same way as in Strings. Let’s look at some examples:

> my\_list = [‘a’, ’b’, ’c’, ‘d’, ‘e’, ‘f’]

> print(my\_list)

**Output:** [‘a’, ’b’, ’c’, ‘d’, ‘e’, ‘f’]

> mylist[0:5:2]

**Output:** [‘a’, ’c’, ’e’]

> anotherlist = mylist[0:3]

> pint(anotherlist)

**Output:** [‘a’, ’b’, ’c’]

**The len function**

Python has built in functions that you can call. We'll slowly introduce more of them as we need them. One useful built in function is the **len** function which returns back the length of an object.

> len(‘python’)

**Output:** 5

> len(anothrlist)

**Output:** 3

**List Methods**

Methods are actions you can call off a function. The syntax is:

mylist = [1,2,3]

mylist.method\_name)

You must call the parenthesis to execute the method. Examples of list method include:

> mylist = [1,2,3]

> mylist.append(6) #This method will add 6 to the list from the end.

> mylist.append(7)

> mylist.append(8)

> mylist.append(9)

> mylist.append(10)

> print(mylist)

**Output:** [1,2,3,6,7,8,9.10]

> mylist.pop() #This method removes the last item on the list except you specify the index you want to pop, But in doing that we don’t need to add the square brackets in this case.

**output:** 10

> print(mylist)

**Output:** [1,2,3,6,7,8,9]

> First\_item = mylist.pop() #we are creating a variable name to store the first item we just popped

> Print(First\_item)

**Output:** 1

> mylist.reverse() #This method doesn't return anything. Instead it performs the action "in-place" or on the list itself without returning anything. So when we print the list it will be in reverse order.

> print(mylist)

**Output:** [9,8,7,6,3,2]

mylist.sort(reverse=False) #This method sorts the list in ascending by default and descending order if you set “reverse=True”

> print(mylist)

**Output:** [2,3,6,7,8,9]

You can change the content of a list by using the **.insert()** method. This is the syntax:

Var\_name.method\_name( index, new-item)

Example:

> mylist.insert(3, ‘cat’)

> print(mylist)

**Output:** [2, 3, 6, ’cat’, 8, 9]

**Nested List:**

A nested list can simply be define a list within a list. Example of nested list include:

> mylist = [1.2.3,[‘a’,’b’,’c’]]

> print(mylist[3])

**Output:** ['a', 'b', 'c']

> print(mylist[3][1])

**Output:**  'b'

> list\_1 = [2,3,"four", [20,30,40, ["one", "two", "three"]]] #note that this list has only 4 indexes

**Exercise:** how can we index “two”

* **Tuple:**

Tuples are ordered sequences just like a list, but have one major difference, they are **immutable**. Meaning you can not **change** them. So in practice what does this actually mean? It means that you can not reassign an item once its in the tuple, unlike a list, where you can do a reassignment.

> atuple = () #In python a list is defined by placing all the elements in a parenthesis.

> t = (1,2,3)

> print(type(t))

**Output:** tuple

>t = (1,2,’a’,’b’)

>print(t[2])

**Output:** ‘a’

Let us look at the immutability property of a tuple.

> mylist = [1,2,3] #we can secede to change the element in index 0 on a list but not on a tuple

> mylist[0] = ‘a’

> print(mylist)

**Output:** [‘a’,2,3]

> t = (1,2,3) #if we try to reassign a value to any of the index, we will get an error message.

> t[1] = ‘cat’ # when we try to execute this we get an error message,

**Output: PS C:\Users\YASHIM GABRIEL\HelloWorld> python -u "c:\Users\YASHIM GABRIEL\HelloWorld\test.py"**

**Traceback (most recent call last):**

**File "c:\Users\YASHIM GABRIEL\HelloWorld\test.py", line 2, in <module>**

**t[1] = 'cat'**

**TypeError: 'tuple' object does not support item assignment**

**Some tuple methods include:**

> t = (‘a’,’b’,’c’,;d’)

> t.index(‘b’) #this method will return the index of item ‘b’

**Output:** 1

> t.count(‘c’)

**Output:** 2

* **Sets:**

Another fundamental Data Structure is The Set!

Sets are an unordered collection of unique elements. We can construct them by using the **set()** function or by placing all the elements in curly brackets. Let’s see how set works.

> x = set()

> x.add(1)

> x.add(2)

> print(x)

**Output:** {1,2}

Note that a set only consists of unique elements, that is, it does not store duplicate elements unlike a list.

> mylist=[1,1,1,1,1,2,2,2,2,3,3,3,4,4,4,5,5,5,6,6,6,7]

> x=set(mylist)

> print(x)

**Output:** {1,2,3,4,5,6,7}

> print(type(x))

**Output:** set

**Dictionaries:**

So far we've only seen how to store data types in sequences like storing characters in a string or items in a list. But what if we want to store information another way? Python support Dictionaries which is a **key-item** data structure.

The choice of deciding between sequences like a list and mappings like a dictionary often depends on the specific situation. As you become a stronger programmer, choosing the right storage format will become more intuitive. Let’s see how dictionaries work.

Let us create our first dictionary:

> my\_dict = {"Nigeria":0, 'Ghana':3, 'China':4}

> print(my\_dict['Ghana'])

**Output:** 3

> new\_dict = {"item1": 1, "item2" : 2,"item3" : "Abuja"}

> new\_dict["item5"] = 4 #this is how to add new item to a dictionary

> print(new\_dict)

**Output:** {'item1': 1, 'item2': 2, 'item3': 'Kano', 'item4': 'Lokoja', 'item5': 4}

Make a dictionary with {} and : to signify a key and a value

> d = {'key1':'value1','key2':'value2'}

# Call values by their key

> d['key1']

> print(d[‘key2’])

**Output:** ‘value2’

### **Adding New Key-Item Pairs**

> d['new\_key'] = 'new item'

> print(d)

**Output:**

{'key1': 'value1', 'key2': 'value2', 'new\_key': 'new item'}

**Dictionaries are unordered! This may not be clear at first with smaller dictionaries, but as dictionaries get larger they won't retain order, which means they can not be sorted! If you need order and the ability to sort, stick with a sequence, like a list!**

Dictionaries are very flexible in the data types they can hold, they can hold numbers, strings, lists, and even other dictionaries!

> = {'k1':10,'k2':'stringy','k3':[1,2,3,],'k4':{'inside\_key':[20,30,40, {"keyinside": [60,20, "thirty"]}]}}

> print(d["k4"]["inside\_key"][3]["keyinside"][1:])

**Output:** [20, “thirty”]

> print(d['k4']['inside\_key'])

**Output:** [20,30,40, {"keyinside": [60,20, "thirty"]

**Dictionary Methods**

short\_names = {"AAU": "Ambrose Alli University","OAU": "Obafemi Awolowo University", "UNILAG": "University of Lagos","NDA": "National Defence Academy"}

print(short\_names.value())

**Output:**

dict\_values(['Ambrose Alli University', 'Obafemi Awolowo University', 'University of Lagos', 'National Defence Academy'])

print(short\_names.keys())

**Output:** dict\_keys(['AAU', 'OAU', 'UNILAG', 'NDA'])

Print(short\_names.items())

**Output:**

dict\_items([('AAU', 'Ambrose Alli University'), ('OAU', 'Obafemi Awolowo University'), ('UNILAG', 'University of Lagos'), ('NDA', 'National Defence Academy')])

Now let’s do something interesting , let us look at the **input element** in python. The input element is used to get input/data from the user. And you can also print out this data. Let us see how the input element is implemented:

print("What is your name?")

prompt = "my name is "

new = input("")

print(prompt, new)

When we run this code it will ask a question and provide us the space to type in the answer and it will print out the result.

**Note:** Tutors should try this out practically in the class and involve the students as much as possible.

* **Loops: while loop, Break and Continue Keyword, for loop:**

**While Loops**

A while loop will repeatedly execute a single statement or group of statements as long as the condition being checked is **true**. The reason it is called a '**loop**' is because the code statements are repeated over and over again until the condition is no longer met or until the condition becomes **false**.

Code indentation becomes very important as we begin to work with loops and control flow. Now let’s start with a simple while loop.

a = 1

while a <= 10:

    print(a)

    a = a + 1

**Output:**

1

2

3

4

5

6

7

8

9

10

# Start by setting variable x to 0

x = 0

while x < 3:

    print('X is currently')

    print(x)

    print("Adding 1 to x")

    x = x + 1 #alternatively you could write x += 1

**Output:**

X is currently

0

Adding 1 to x

X is currently

1

Adding 1 to x

X is currently

2

Adding 1 to x

**EXERCISE:** write a code to print all odd and even numbers from 1 – 100

#### Note This!

Be careful with while loops! There is a potential to write a condition that always remains True, meaning you have an infinite running while loop. If this happens to you, exit vscode.

**Break Keyword**

The break keyword allows you to "break" out of the loop that contains the break keyword. For example

x = 0

while x < 10:

    print(x)

    print('adding one to x')

    x = x + 1

    if x == 3:

        # This will cause to break out of the top loop

        # Note that if statements don't count as loops

        break

**Output:**

0

adding one to x

1

adding one to x

2

adding one to x

**for loop**

A **for loop** acts as an iterator in Python, it goes through items that are in a sequence or any other iterable item. Objects that we've learned about that we can iterate over include strings,lists,tuples, and even built in iterables for dictionaries, such as the keys or values.

Here's the general syntax for a for loop in Python,take note of the indentation:

for item in object:

statements to do stuff

The variable name used for the item is completely up to the coder, so use your best judgment for choosing a name that makes sense and you will be able to understand when revisiting your code.

**For loop with a list**

mylist = [1,2,3,4,5,6,7,8,9,10]

for item in mylist:

    print(item\*\*2) # each item in the list is multiplied to the power of 2

**Output:**

1

4

9

16

25

36

49

64

81

100

mylist = [1,2,3,4]

for a in mylist:

    print("{} raise to the power of 2 is {}".format(a,a\*\*2))

**Output:**

1 raise to the power of 2 is 1

2 raise to the power of 2 is 4

3 raise to the power of 2 is 9

4 raise to the power of 2 is 16

for num in mylist:

    print("I am in a for loop")

**Output:**

I am in a for loop

I am in a for loop

I am in a for loop

I am in a for loop

**For loops with strings**

for letter in "This is a string":

    print(letter)

**Output:**

T

h

i

s

i

s

a

s

t

r

i

n

g

mystring = 'This is a string'

for word in mystring.split():

    print(word)

**Output:**

This

is

a

string

for a in range(4,100,10):

    print(a)

**Output:**

4

14

24

34

44

54

64

74

84

94

**for loop with dictionaries**

my\_dictionary = {'a': 1, 'b': 2, 'c': 3}

for item in my\_dictionary:

    print(item)

**Output:**

a

b

c

my\_dictionary = {'a': 1, 'b': 2, 'c': 3}

for x in my\_dictionary.values():

    print(k)

**Output:**

1

2

3

my\_dictionary = {'a': 1, 'b': 2, 'c': 3}

for k in my\_dictionary.keys():

    print(k)

    print(my\_dictionary[k])

    print('\n')  # '\n' means new line

**Output:**

a

1

b

2

c

3

**Continue Keyword**

The continue keyword can be a bit tricky to see its usefulness, but it allows you to continue with the top level loop, basicaly the opposite of break. It will take time before you realize a good situation to use it in, but here is a simple example:

for letter in 'code':

    if letter == 'e':

        continue

    print('Current Letter is:',letter)

**Output:**

Current Letter is: c

Current Letter is: o

Current Letter is: d

**Day 9: Conditional Statements**

* **If Statement: syntax for simple If statement:**

Majority of the time when programming, we'll need to control the flow of our logic. Our program will want to perform an action in only certain cases, we can use the **if**, **elif**, and **else** statements to control for these cases.

The syntax for an if statement

if some\_condition:

#List of expressions to be executed

Let’s look at some examples:

if 3<2:

    print('Two is lesser')

**Output:** Tow is lesser

if 1>2:

    print("One is greater than two")

This line of code will not produce any output. Notice what happens, the indented block of code only runs when the if condition is True!

* **if else statement**

Let's now add in an alternate action in case the **if** is not True using the **else** statement.

Notice the format and how the code blocks line up, this is crucial in Python! Code indentation let's Python know what blocks and statements correspond together.

if 1==1:

    print("One is equal to One")

else:

    print("First if was not True")

**Output:** One is equal to One

if 1==2:

    print("One is equal to Two")

else:

    print("First if was not True")

**Output:** First if was not True

saved\_password = 123456

new\_password = input("Enter your password: ")

if int(new\_password) == saved\_password:

    print("Password correct")

else:

    print("You're a criminal")

**Output:**

Enter your password: 222

You’re a criminal

**If, elif, else statement**

Now let's imagine we have multiple conditions to check before the final **else** statement, this is where we can use the **elif** keyword to check for as many individual conditions as necessary:

if 2 == 0:

    print('First condition is True')

elif 2 == 1:

    print("Second condition is True")

elif 2 == 100:

    print("Third condition is True")

else:

    print("None of the above conditions were True")

**Output:** None of the above conditions were True

if 2 == 2:

    print('First condition is True')

elif 2 == 1:

    print("Second condition is True")

elif 2 == 100:

    print("Third condition is True")

else:

    print("None of the above conditions were True")

**Output:** First condition is True

**Day 10: Functions in python**

* **Defining a Function: syntax for defining a Function:**

A function is a block of organized, reusable code that is used to perform a single, related action. Functions provide better modularity for your application and a high degree of code reusing.

### Defining a Function

You can define functions by following the rules below:

* Function blocks begin with the keyword **def** followed by the function name and parentheses ( ( ) ).
* Any input parameters or arguments should be placed within these parentheses. You can also define parameters inside these parentheses.
* The first statement of a function can be an optional statement - the documentation string of the function or docstring.
* The code block within every function starts with a colon (:) and is indented.
* The statement return [expression] exits a function, optionally passing back an expression to the caller. A return statement with no arguments is the same as return None.

**def** functionname( parameters ):

"function\_docstring descrinng what the function does"

*# list of expressions to be executed*

**return** [expression]

### Example

The following function takes a two numbers as input parameters and prints out the sum:

def print\_sum(a, b):

    "This prints the sum of two numbers a and b"

    print(a + b)

    return

### **Calling a Function**

Defining a function only gives it a name, specifies the parameters that are to be included in the function and structures the blocks of code. Inorder to use the function, you must call the name and pass in the required parameters.

To demonstrate this, we call the **print\_sum** function we created above:

print\_sum(7, 13)

**Output:** 20

## **Passing values**

### **Pass by reference**

All parameters (arguments) in the Python language are passed by reference. It means if you change what a parameter refers to within a function, the change also reflects back in the calling function. For example:

def change\_name(name):

    print("The passed name is {}".format(name))

    #change the name

    name = "Harry Potter"

    print("The new name in this function is {}".format(name))

    return

#call the function|

name = "John Stone"

change\_name(name)

**Output:** The passed name is John Stone

The new name in this function is Harry Potter

* **Function Argument, Return Statement:**

You can call a function by using the following types of formal arguments:

* Required arguments
* Keyword arguments
* Default arguments
* Variable-length arguments

### **Required arguments**

Required arguments are the arguments passed to a function in correct positional order. Here, the number of arguments in the function call should match exactly with the function definition. For example, the function change\_name above requires one argument for it to work. If you call the function without the required argument it throws an error.

change\_name("Opeyemi")

**Output:**

The passed name is Opeyemi

The new name in this function is Harry Potter

### **Keyword arguments**

Keyword arguments are related to the function calls. When you use keyword arguments in a function call, the caller identifies the arguments by the parameter name.

change\_name(name="Mercy")

**Output:**

The passed name is Mercy

The new name in this function is Harry Potter

def printinfo(name,age):

    "This prints the info about a person"

    print("Name: ", name)

    print("Age ", age)

    return

printinfo(name="Jesse", age=20)

**Output:**

Name: Jesse

Age 20

### **Default arguments**

A default argument is an argument that assumes a default value if a value is not provided in the function call for that argument. The following example gives an idea on default arguments, it prints default age if it is not passed.

def printinfo(age=25):

    "This prints the info about a person"

    print("Age ", age)

    return

printinfo()

**Output:**

Age 25

**Exercise:** printinfo(45)

### **Variable-length arguments**

You may need to process a function for more arguments than you specified while defining the function. These arguments are called variable-length arguments and are not named in the function definition, unlike required and default arguments.

def functionname([formal\_args,] \*var\_args\_tuple ):

       "function\_docstring"

   function\_expressions

   return [expression]

An asterisk (\*) is placed before the variable name that holds the values of all non-keyword variable arguments. This tuple remains empty if no additional arguments are specified during the function call. Examples include:

def printargs(name, \*scores):

   "This prints a variable passed arguments"

   print("Name is: {}".format(name))

   #print info in variable leanght arguments

   for var in scores:

        print(var)

printargs("John", 80,90,98,97)

**Output:**

Name is: John

80

90

98

97

### **Return Statement**

The statement return [expression] exits a function, optionally passing back an expression to the caller. A return statement with no arguments is the same as return None.

def sum(a, b ):

       # Add both the parameters and return total"

   total = a + b

   return total

# Now you can call sum function and pass the result to a variable

total = sum(10,20)

print(total)

**Output:**

30

### **Global vs. Local variables**

Variables that are defined inside a function body have a local scope, and those defined outside have a global scope.

This means that local variables can be accessed only inside the function in which they are declared, whereas global variables can be accessed throughout the program body by all functions. When you call a function, the variables declared inside it are brought into scope.

sum = 0  # This is global variable.

def add\_num(a, b):

    # Add both the parameters and return total"

    sum = a + b

    print("The total inside the function is: {}".format(sum))

    return sum

# Now you can call sum function

add\_num(10, 20)

print("The total outside the function is: {}".format(sum))

**Output:**

The total inside the function is: 30

The total outside the function is: 0