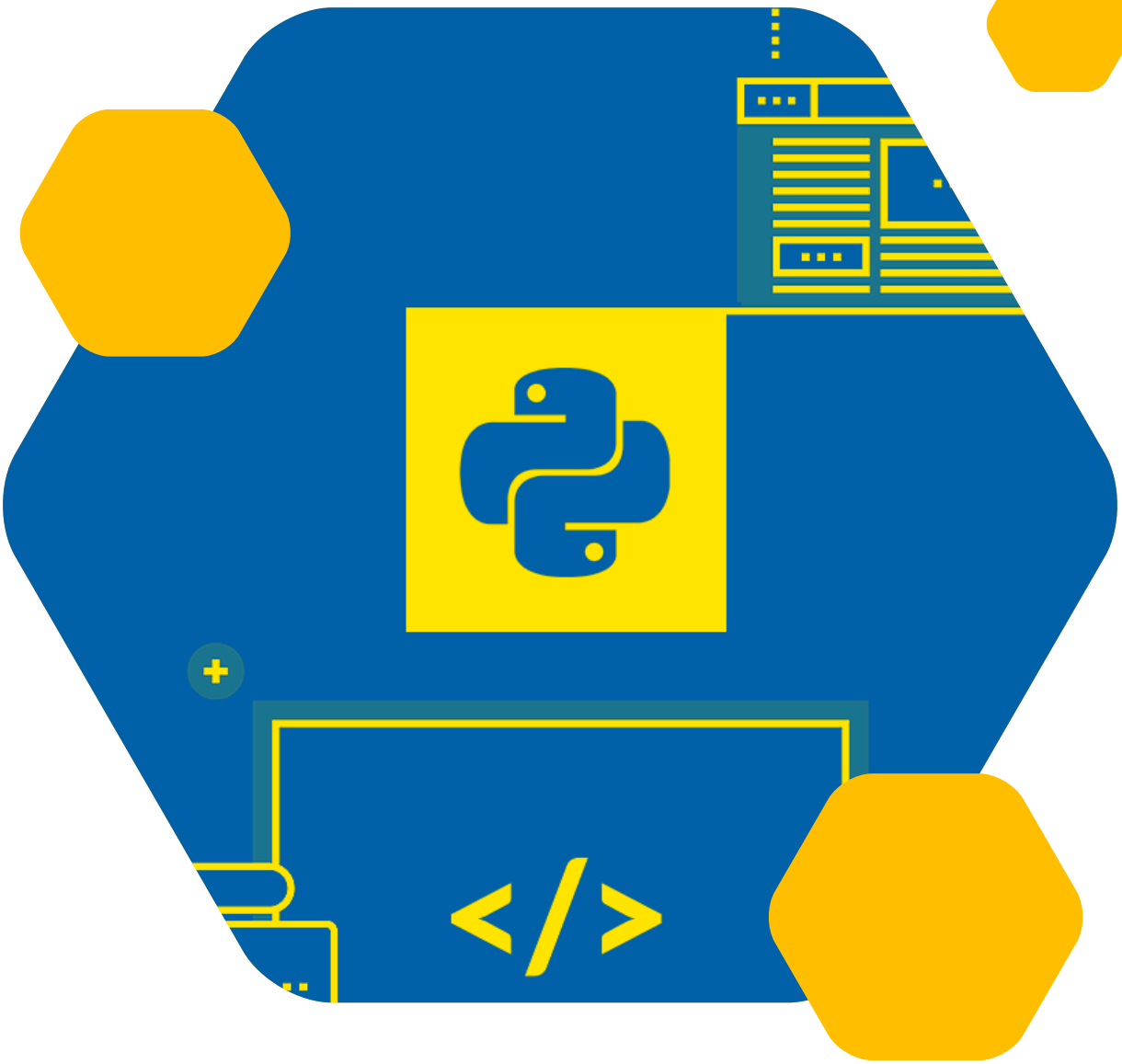
Firmbee

Python from Beginner to Advanced



Python from Beginner to Advanced

Python from Beginner to Advanced

This Python e-book will help you understand all the vital elements of the Python programming language. Anyone who wants to learn Python without any prior experience in programming and anyone who wants to refresh their Python knowledge can read this e-book and get a grip on widely used Python concepts.

1. Introduction to Python

Starting with Python

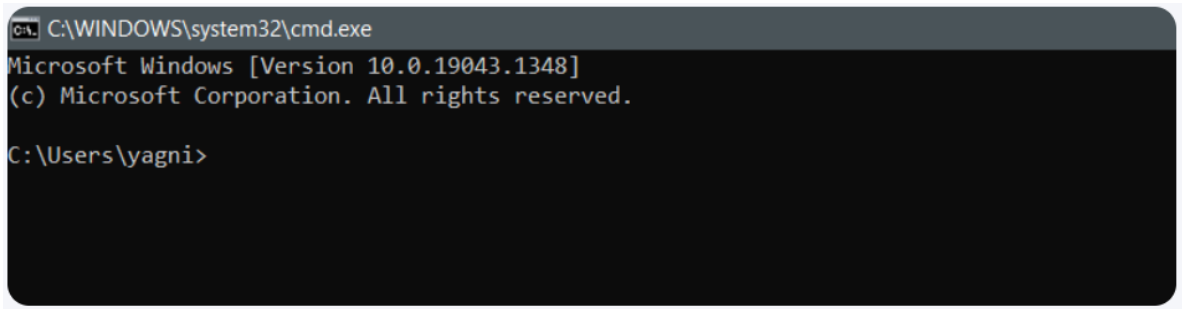
After reading this Python e-book, the you will be able to write programs in Python, use any Python libraries and develop their own packages using Python.

The first step in learning any programming language is to set up the environment for writing programs. As we are going through a Python e-book, we will start with installing Python in three different OS platforms.

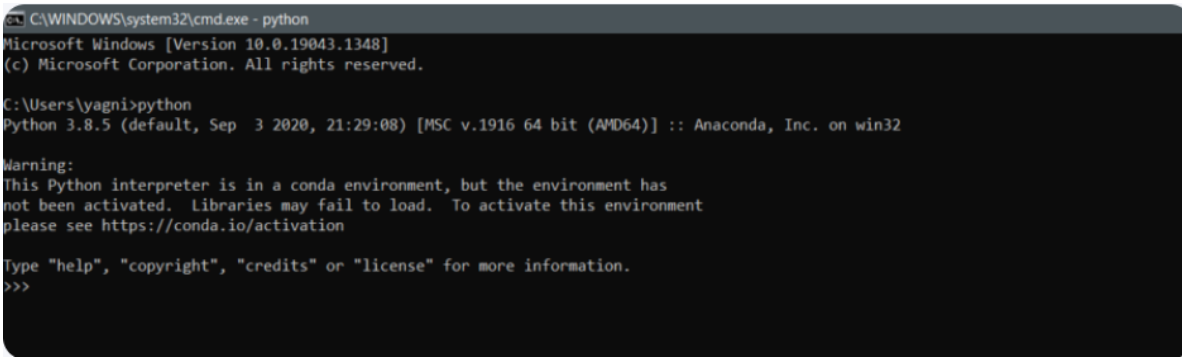
Python Installation

To check if Python is already installed, follow the below-mentioned steps.

- Press Windows + r to get the run.
- Then type cmd and press enter.

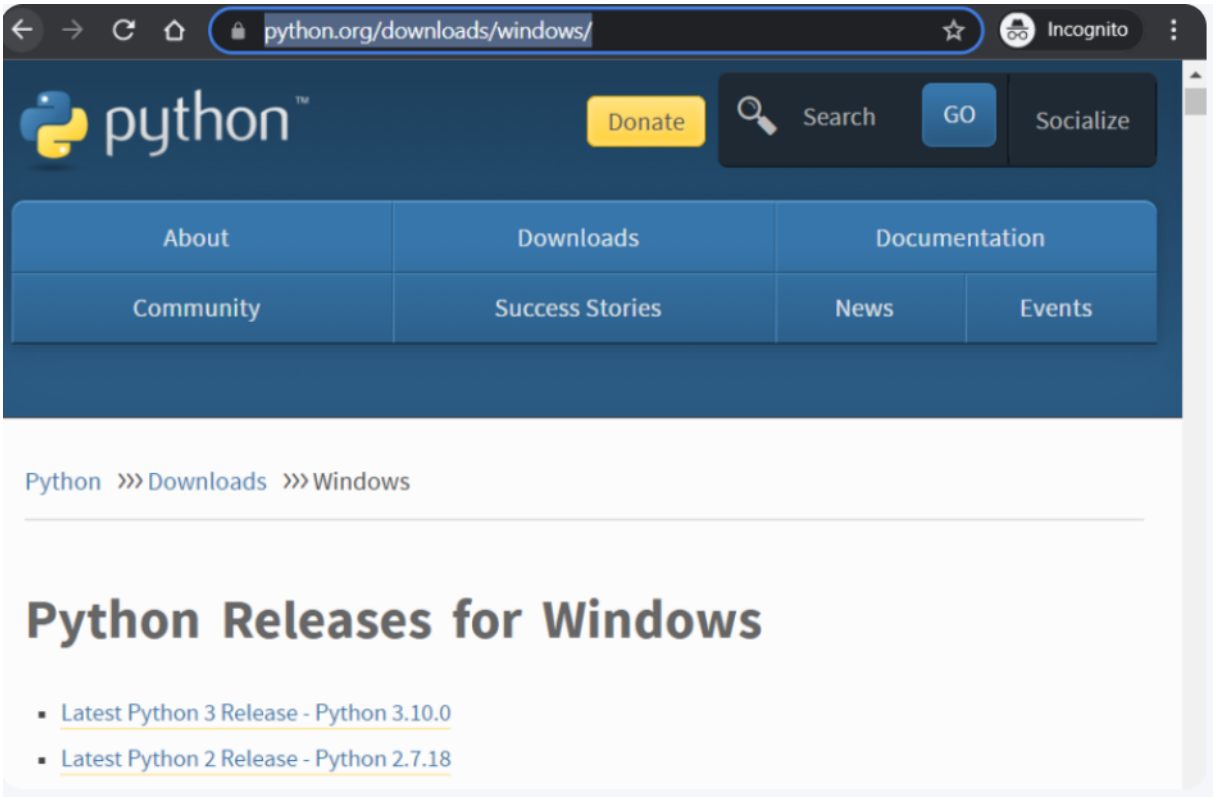


- After opening the cmd. you can check if Python is already installed by using typing Python into the cmd.

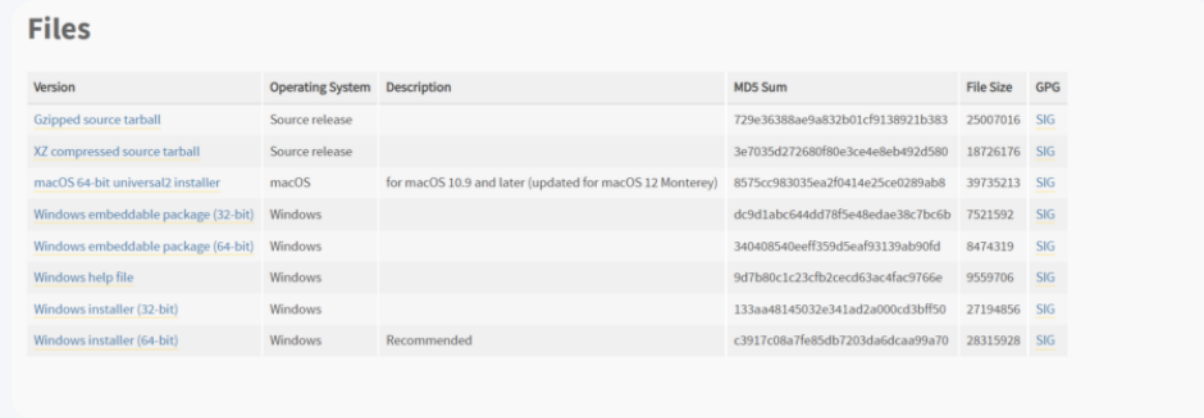


We can also check the version of Python installed by using commands as demonstrated below.

Now we will walk through on how to install Python in Windows. Links are provided for quick navigation when following the e-book. From the Python for Windows weblink, the stable version of Python can be downloaded with your choice between 64 bit or 32-bit Operating system versions.



As we can see, the latest release available for Python 3 is Python 3.10.0. Now click on the Latest Python 3 Release – Python 3.10.0 and it will navigate you to the downloader's page where if we scroll down to the bottom of the page, we will find a table as below.



Files

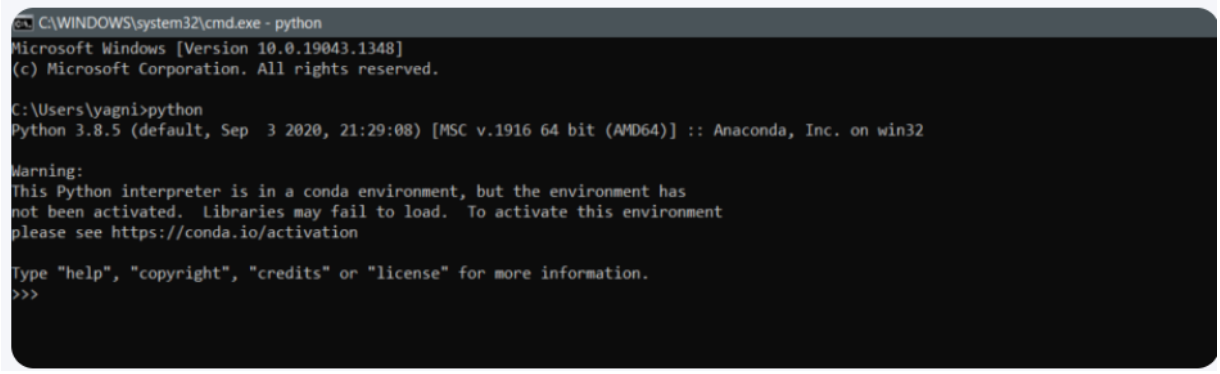
Version	Operating System	Description	MD5 Sum	File Size	GPG
Gzipped source tarball	Source release		729e36388ae9a832b01cf9138921b383	25007016	SIG
XZ compressed source tarball	Source release		3e7035d272680f80e3ce4e8eb492d580	18726176	SIG
macOS 64-bit universal2 installer	macOS	for macOS 10.9 and later (updated for macOS 12 Monterey)	8575cc983035ea2f0414e25ce0289ab8	39735213	SIG
Windows embeddable package (32-bit)	Windows		dc9d1abc644dd78f5e48edae38c7bc6b	7521592	SIG
Windows embeddable package (64-bit)	Windows		340408540eeff359d5eaf93139ab90fd	8474319	SIG
Windows help file	Windows		9d7b80c1c23cfb2cecd63ac4fac9766e	9559706	SIG
Windows installer (32-bit)	Windows		133aa48145032e341ad2a000cd3bff50	27194856	SIG
Windows installer (64-bit)	Windows	Recommended	c3917c08a7fe85db7203da6dcaa99a70	28315928	SIG

Now click on the Windows Installer (32-bit) or Windows Installer (64-bit) according to your desire. A window will open asking you to select the path where you want to download your installer. After downloading the executable file, double-click on the file to start the installation.

Below you can find the steps.

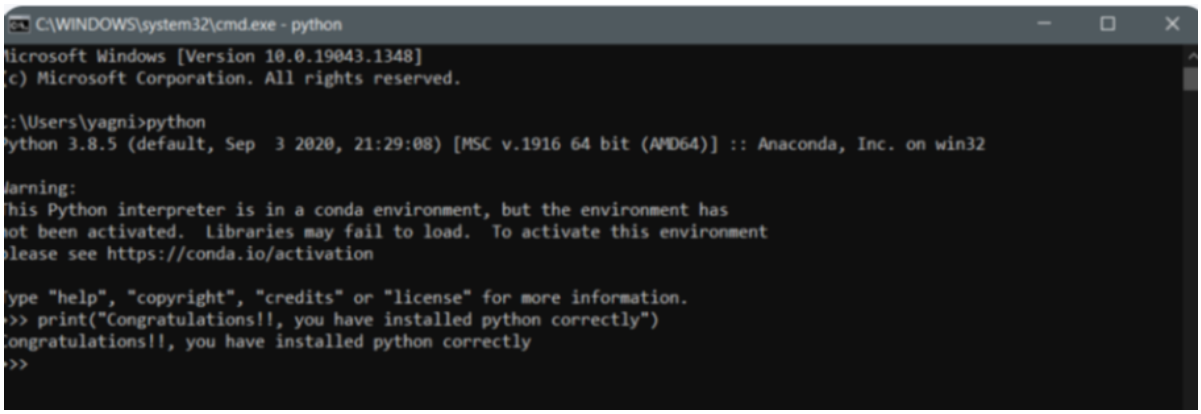
- Run the Python executable file, in our case it will be Python-3.10.0.exe.
- When you double-click on the file, a window will open asking do you want to run this file. Click on run to start the Python installation.
- According to your choice select if you want Python to be installed for all users or for a single user.
- Also, select add Python 3.10 to PATH check box.

- Then select install now. Install now will install Python with all recommended settings which is a good option for beginners.
- Then it will take few minutes for the setup to complete, and you will be taken to next dialog prompt which will ask you to disable the path length limit. This will allow the Python to use long path names without any character
- limit of 260 which is enabled if the path length limit is not disabled.
- To verify if Python is installed, you can use the Python -V or Python –version or just type Python in the cmd.



Congratulations, you have successfully installed Python. Let's write our first program in cmd using Python.

- In our first program we will just print "Congratulations!, you have installed Python correctly".
- To write this, we will use print function of Python.
- Type print("Congratulations!!, you have installed Python correctly").
- Then press enter.
- You will see that the statement we wrote inside the print as displayed below.



Introduction to Python

Python is an interpreted high-level dynamically typed object-oriented programming language.

Before delving into writing programs in Python, it's important to understand what the above terms mean.

High Level Language

A high-level language gives the programmer freedom to code programs which are independent of a particular type of device. They are called high level languages as they are closer to human languages. Python is high level because it is not a compiled language, Python requires another program to run the code unlike C which run directly on local processor.

Interpreted Language

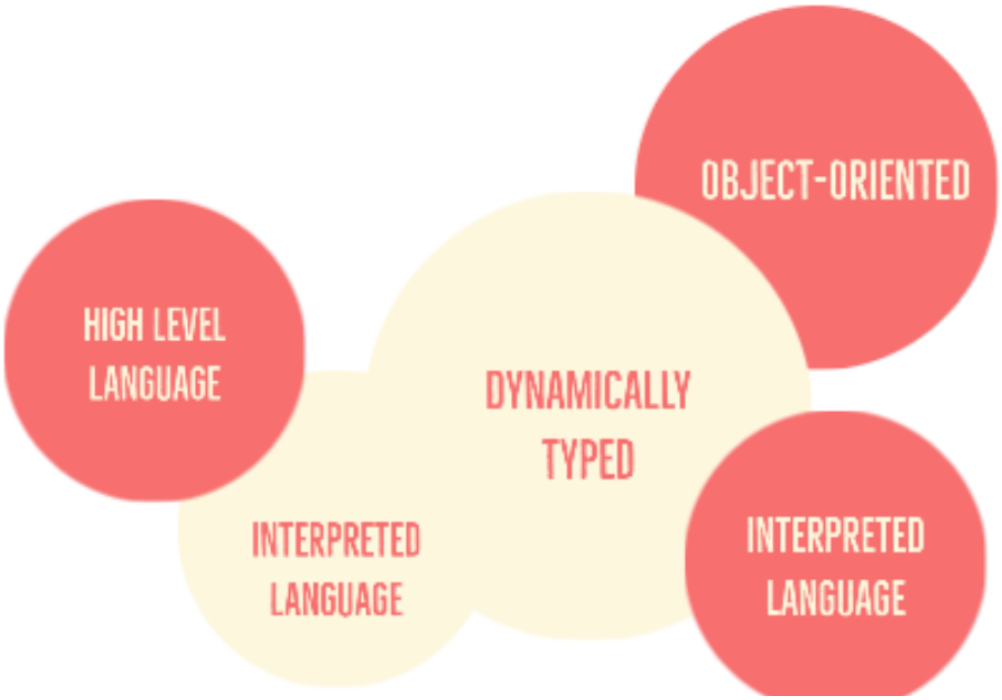
Python is an interpreted language, as the Python program's source code is converted into byte code that is then executed in the Python virtual machine, unlike C or C++.

Dynamically Typed Language

Python is a dynamically typed language because the type of the variable is checked during run time. We will learn about data types in the following chapters.

Object-Oriented Language

Python is an object-oriented language because the Python developer can use classes and objects to write clean and reusable code.



Advantages and disadvantages of using Python

Advantages of using Python

- As Python syntaxes are closer to human language, it is easier to learn, understand and write the code.
- It is both functional and objected-oriented language.
- Python has a large community support and also it has a large number of modules, libraries and packages.
- Due to its simplicity, developing a Python program or application faster than developing in any other language like Java.
- Python is a choice of language in data science, machine learning and artificial intelligence due to its wide variety of machine learning packages and libraries.
- Almost everything can be developed using Python, it also has tools for app development such as kivy, flask, Django and many others.

Disadvantages of using Python

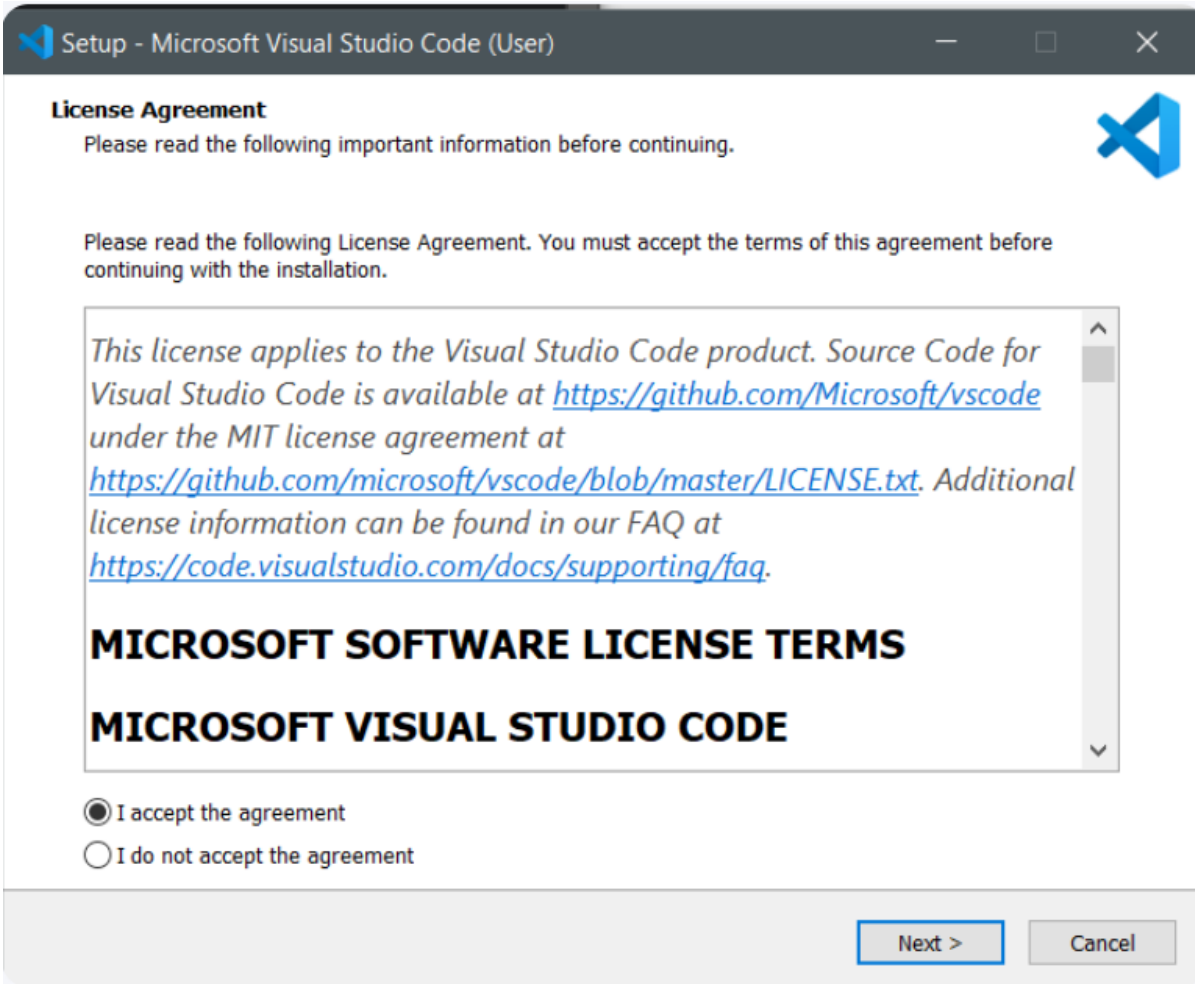
- It's not recommended for communication with hardware components.
- There are no time optimizers in Python, hence it's slower than most of the languages like C, C++, and Java.
- The indentation-based coding makes it a little difficult for people changing their language from C, C++, or Java to Python.\

Setting up an Integrated Development Environment:

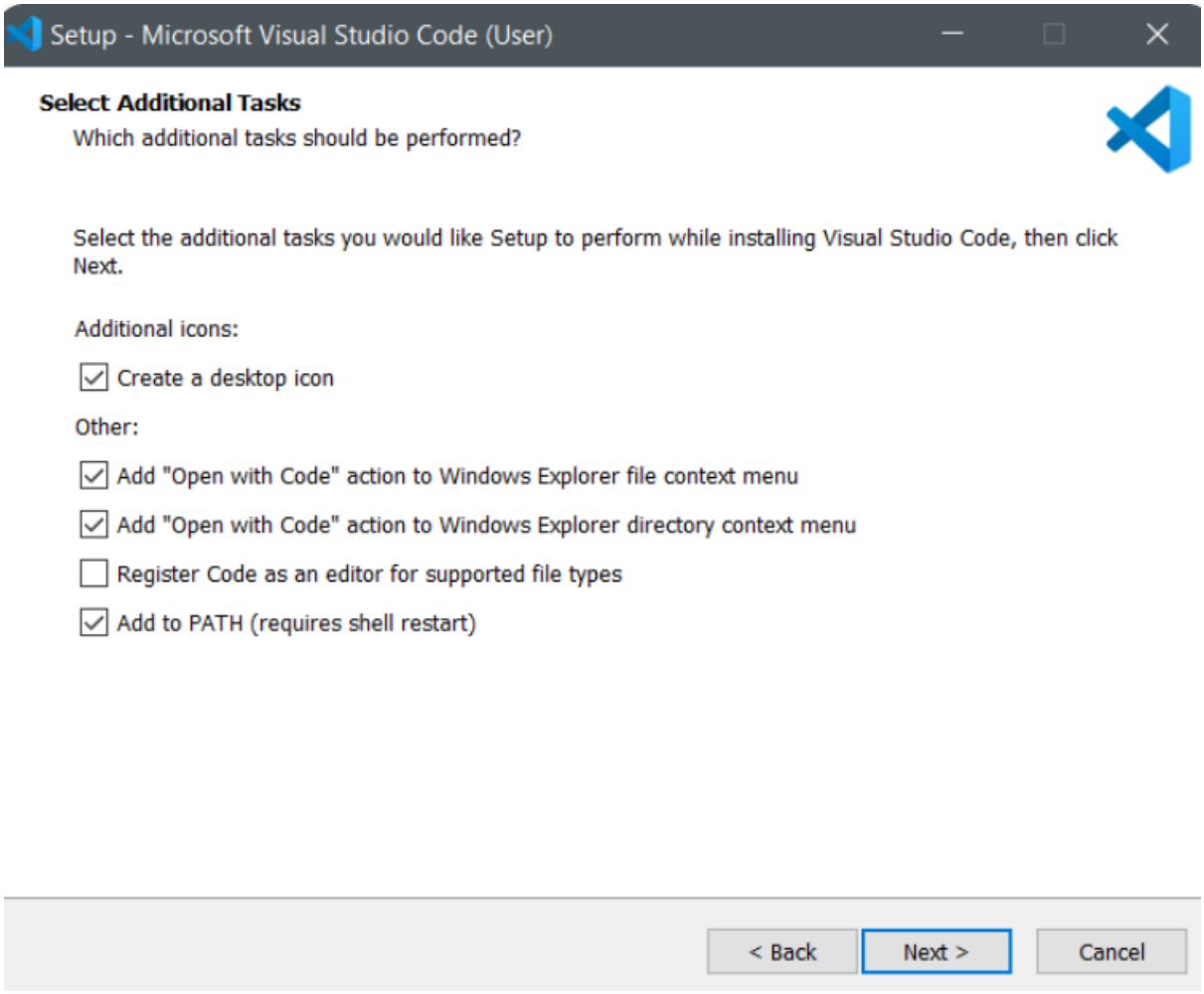
We will be using Visual Studio Code for writing code in Python. Visual Studio Code abbreviated as VS Code is an open-source code editor with many plugins and extensions. These plugins and extensions make writing code in VS Code simpler and more intuitive. Also, VS Code is very light compared to other IDE. It also has various themes for making the development environment interesting for the developer.

Installing VS Code in windows:

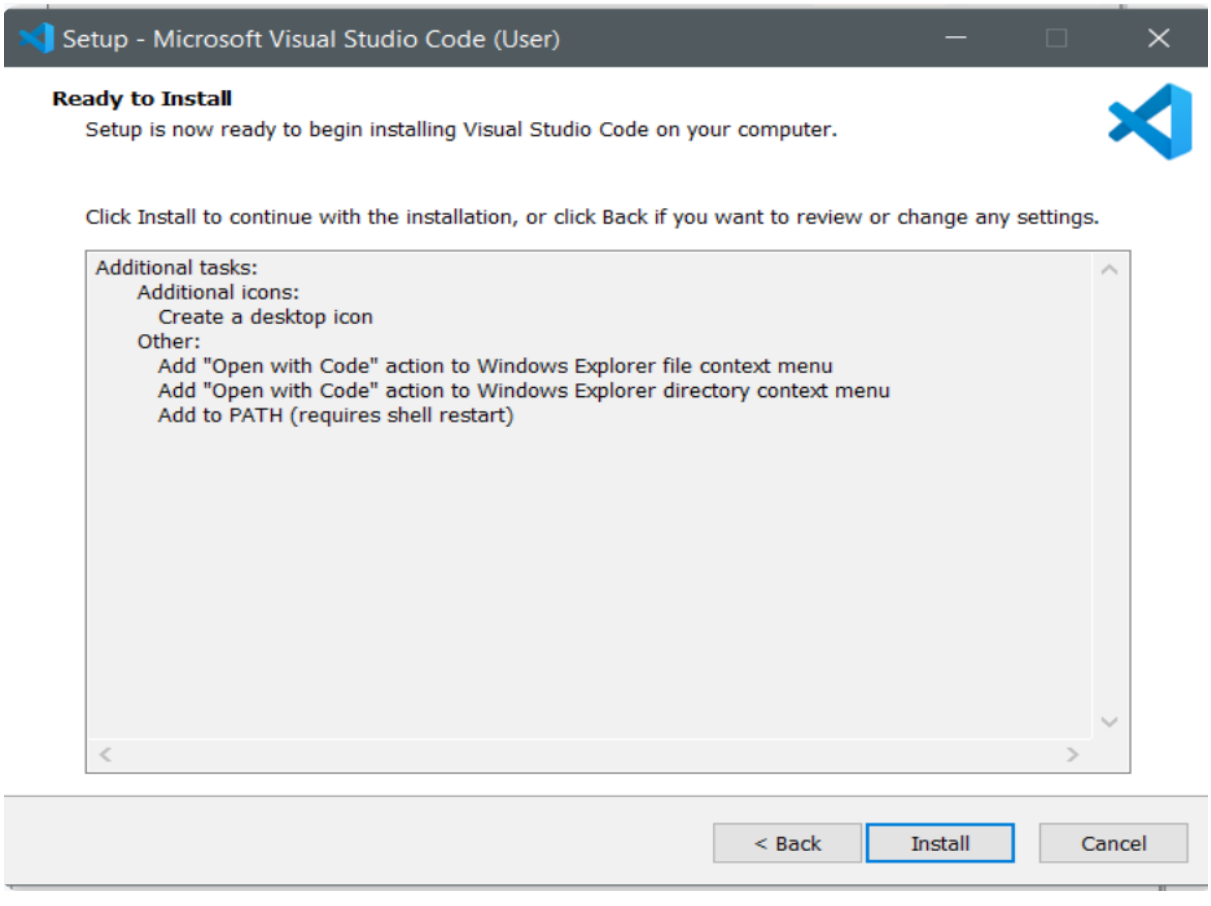
- By using the link below, download the VS code executable file. Link: https://code.visualstudio.com/docs/setup/windows
- Then double-click on the downloaded file to execute it and click run. Then follow the steps as given in the images below.
- Click on I accept the agreement and click next.



- Select the checkboxes as shown in the below image and click next.

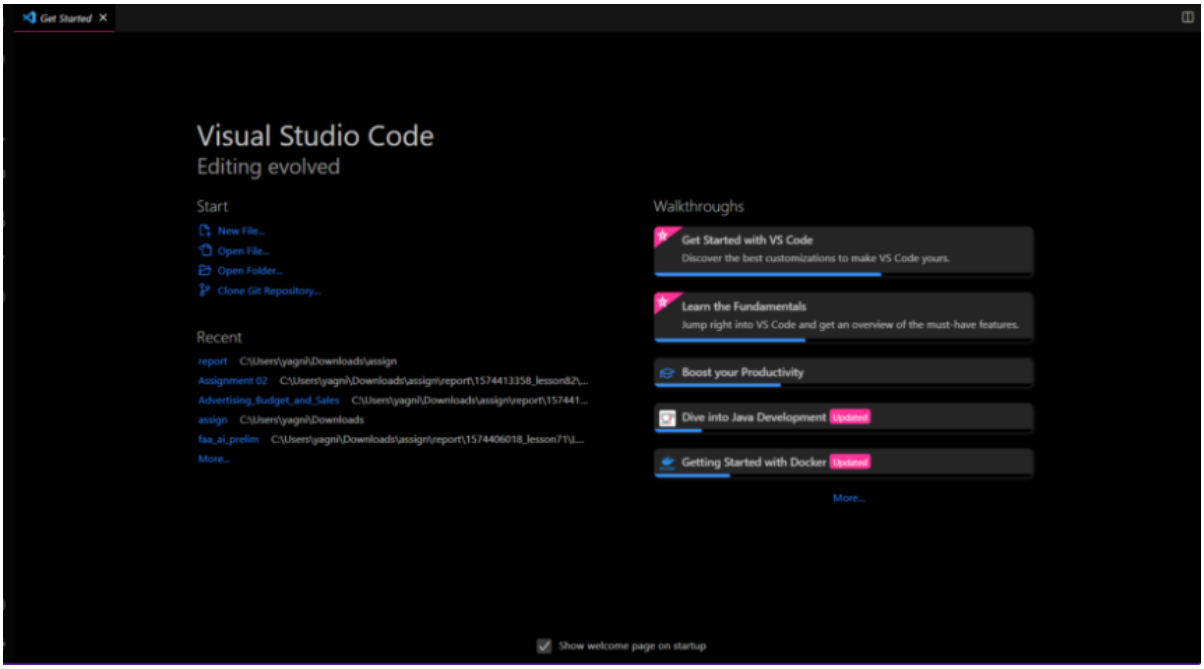


- Then click on install and it will take few minutes for the setup of VS code to complete. After completion of the setup click on finish button.

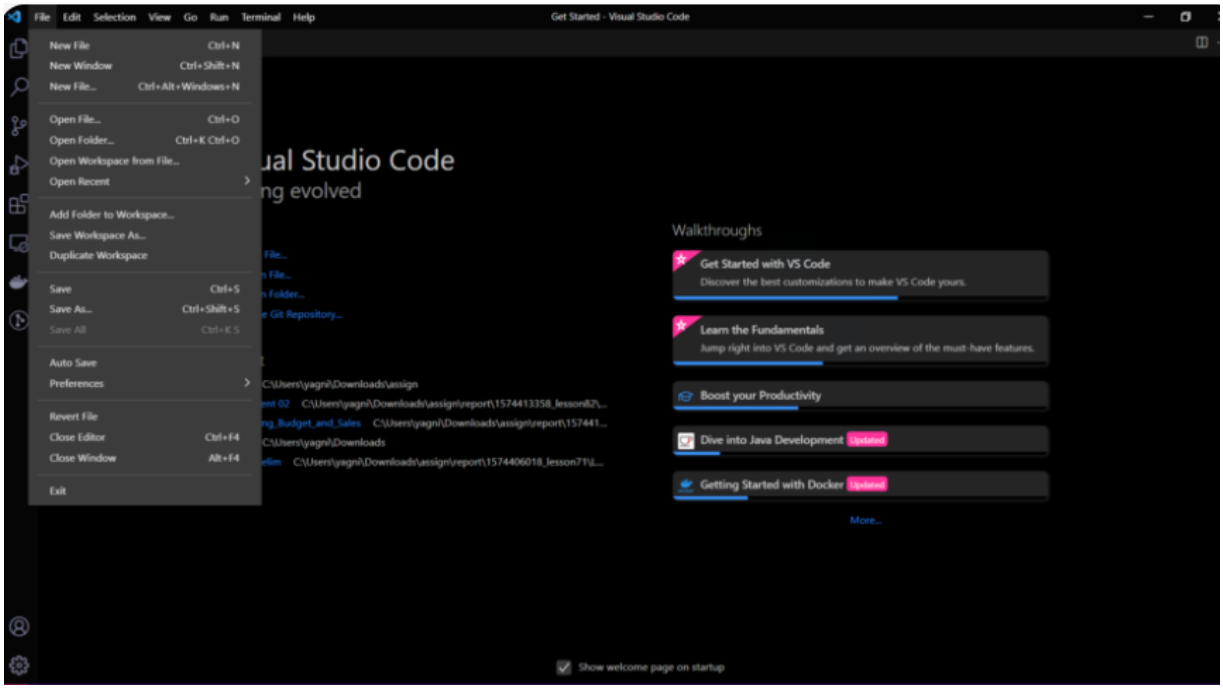


Writing first code using VS Code IDE:

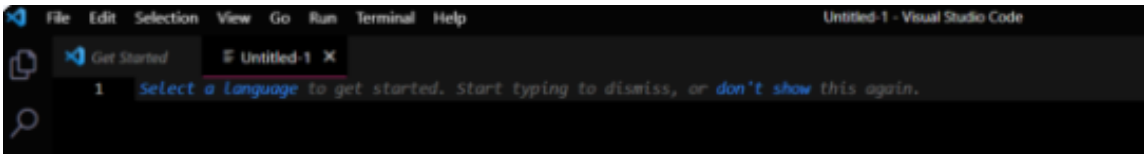
- Open VS Code and you will see a window as shown below.



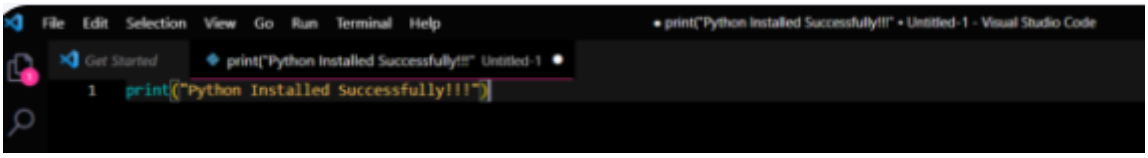
- Click on the file to open file menu and click on new file as shown below.



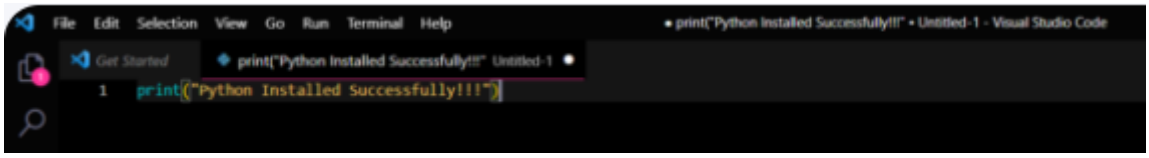
- Then a tab will open in VS Code named Untitled-1 as shown below.



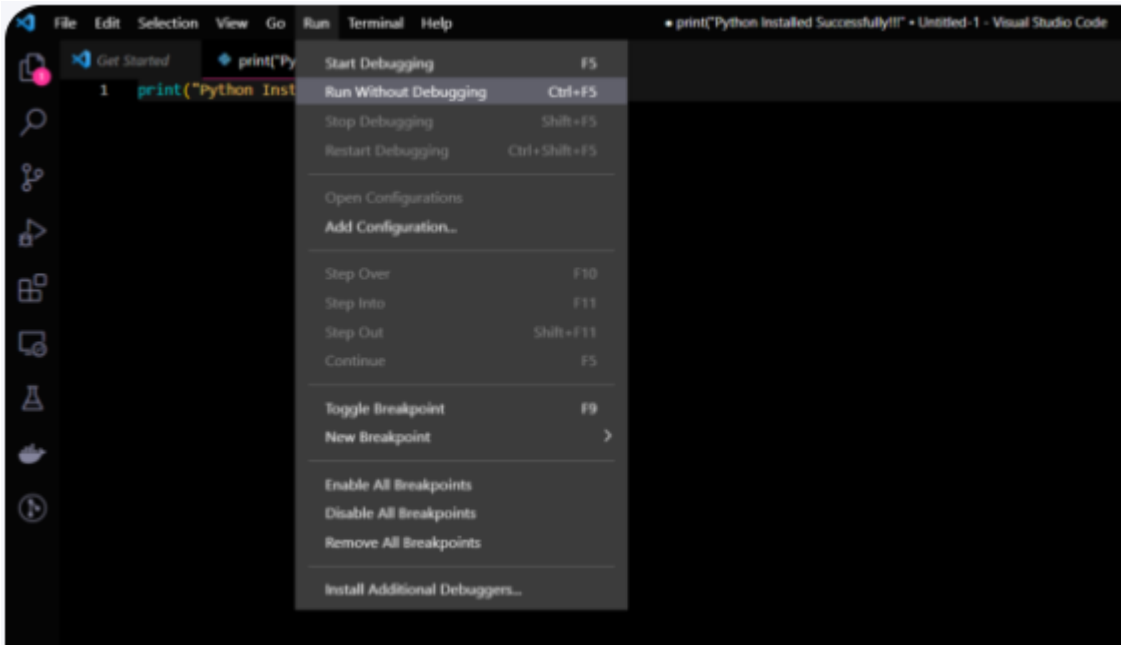
- Click on select a language and the below window will open where you have to select Python.



- Then type the code print("Python Installed Successfully!!!") as shown below.



- Then Go to run tab as shown below and select run without debugging.



- Then VS code will ask you to save the file. Save the file in the directory you desire. It will run the file after saving and shows you the result as below.

2. Variables and Data Types in Python

As we have learned in the previous chapter - Python is a high-level, interpreted, dynamically typed, and object-oriented language. Due to its high-level nature, the language is very easy to learn, and syntax is also simple. There are a variety of applications of Python in real-world like for machine learning, data science, game development, web applications, and many more.

In the previous chapter, we learned on how to print text in Python. We used to print ("your desired text") as the syntax. Let's start with what are variables and why do we use variables.

Variables in Python

A variable is an entity that stores a value. The value may be a number, integer, real number, text, or a character. Let's see some examples of how you can use a variable to store values in Python.

```
1  # variables
2  x = 1  # storing integer
3  y = 2.5  # storing real number
4  z = "string"  # storing string or text
5  n = "a"  # storing a character
6  b = True  # storing a boolean value
7  print(x,y,z,n,b)
```

```
1  Output:
2  1 2.5 string a True
```

We have seen how to store variables, now let's see how to print their values. You already know the answer, which is to use print(), which we used in the first chapter to print the desired text. Also, see that we are using the variables without using any double quotes or single quotes as opposed to before. This is because a variable is recognized by print directly as it is stored in the memory when it is declared. Now, let's print the variables.

We can see that the variables are printed as highlighted in the above image. As we can see Python supports most of the different data types in Python like integer, float (real numbers), string (text or characters) and Boolean (True or False).

Data types in Python

Strings

What operations can be performed using strings?

- **title()**

This function can be used to capitalize the starting letter of each word in the string as seen below the output is highlighted.

```
text="this blog is awesome"
print(text.title())
```

Output:

```
This Blog Is Awesome
```

- **upper()**

This function can be used to capitalize the whole words in the string. The example is illustrated in the below image.

```
text="this blog is awesome"
print(text.upper())

Output:
THIS BLOG IS AWESOME
```

- **lower()**

This function can be used to convert the whole words in the string into lowercase alphabets. The example is illustrated in the below image.

```
text = "this blog is awesome"
print(text.lower())
```

```
Output:

this blog is awesome
```

- **Concatenation of strings**

To combine two different strings "+" can be used. The example is illustrated in the below image.

```
text = "this blog is awesome"
text2="for beginners"
print(text+text2)
```

```
Output:

this blog is awesome for beginners
```

- **White Spaces**

There are times when you don't want to print text in a single line but to have multiple lines, and sometimes you want text to be having tab space. This can be done in Python by using "\n"(new line) and "\t"(tab space). The example is illustrated below.

```
print("this blog is \nawesome")
print("\tthis blog is awesome")
```

```
Output:

this blog is
awesome
 this blog is awesome
```

- **Strips functions**

This is a function in Python that removes any white space in the string. Using strip both left and right white spaces can be removed. But sometimes for the specific requirements for removing white space in left "lstrip()" can be used and for right "rstrip()" can be used. The example with code is illustrated below

```
z=" hello "
print(z)
print(z.strip())
print(z.rstrip())
print(z.lstrip())
Output:
" hello "
"hello"
" hello"
"hello "
```

- **String Length**

By using len() function, a string length can be determined. The example with code is illustrated below. You can see for string "hello", the output is 5.

```
Z="awesome"
Print(len(Z))
```

```
Output:
5
```

Integers

The integers data types in Python are used only when whole numbers are to be stored. There are several operations, which can be performed on integers. Let's learn about type() function here. The type() function tells you about the variable's datatype. The example for the type() function with code is illustrated below.

```
1  a=1
2
3  type(a)
```

```
1  Output:
2
3  int
```

Floats

In integer data type variables only, whole numbers can be stored but to include real numbers or to store real numbers we use float. In essence, float is used for decimals.

```
1  a=1.6
2
3  type(a)
```

```
1  Output:
2
3  float
```

Operations on floats and integers

In our basic mathematics during our high school, we have learned numerous operations which can be performed on numbers like Addition, Subtraction, Multiplication, Division and many more. We can perform all those operations on floats and integers as illustrated below with code.

```
1  # variables
2
3  x = 1  # storing integer
4  y = 2.5  # storing real number
5  z = "string"  # storing string or text
6  n = "a"  # storing a character
7  x = True  # sprint(x,y,z,n,b)toring a boolean value
```

print(type(x),type(y),type(z),type(n),type(b)) [/code]

```
Output:


<class 'bool'> <class 'float'> <class 'str'> <class 'str'> <class 'bool'>
```

Boolean

In Python there are times when a developer needs to know if a statement is true or false. Mostly when using conditions, the Boolean are used. Boolean consists of True and False. Not that Python is case-sensitive when using Booleans, hence they need to be in the True and False format only.

Let's test our knowledge!

- Type Conversion
  The type conversion is a process where you convert one datatype variable into another datatype variable.

- int()
  This converts a number that is in string form or a float to integer value. The example is illustrated below with the code.

```
a="6"
b=6.5
print(int(a),int(b))
```

```
output:


6 6
```

- Note: The int() can only convert numbers in string form to integers but not characters. If characters are used in int(). then it will give an error as illustrated below.

```
a="a"


print(int(a))
```

```
output:


---------------------------------------------------------------------------ValueError Traceback (most recent call last) <iPython-input-128-d5a3b8380653> in <module> 1 a="a" 2 ----> 3 print(int(a)) ValueError: invalid literal for int() with base 10: 'a'
```

- float()
  This is used for converting any real number in string form or any integer to float as illustrated in below code.

```
a=”6.5”
b=7
print(float(a),float(b))
```

```
Output:

6.5 7.0
```

- str()
  This function can convert any integer or float value to string form. The example is illustrated below.

```
a = 6
b = 6.7
c = True

print(str(a), str(b), str(c))
```

```
Output:

6 6.7 True
```

- bool()
  This function can convert any integer, string, float value into a Boolean value.

- Note: If the values in integer or float are 0, then the bool() will give False. For strings, if the string is empty then False.
  The example is illustrated below.

```
a = 0
b = 0
c = “”

print(bool(a), bool(b), bool(c))
```

Output: FalseFalseFalse

3. Python tuples, lists, sets and dictionaries

This chapter will help in developing the understanding of Python tuples, lists, sets and dictionaries. We will see some examples of their implementations and their use cases for some tasks. The coding part will be done in VS Code. If you have not installed VS Code or want to start from scratch, please check the previous chapter.

Introduction to Python tuples, lists, sets and dictionaries

In the previous chapter, we saw how we can use the variables and data types in Python. We also investigated some useful functions related to data types and variables.

Python is a powerful scripting language. It has many built-in data structures available for use. These structures are so powerful in handling the data, yet they are simple to implement.

These basic structures are of four types – list, tuple, dictionary and set.

Lists in Python

Lists are in-built in Python. These are mutable, so items can be added or removed from them without altering their original contents, and elements can be accessed through index.

They are so general that they can be used to store any type of object, from strings to numbers, even the objects also. Moreover, it is not required to have all the elements of the same type, A list can have elements of different types.

To use list, you need to initialize a variable by [].

For Example:

```
# An empty list
empty_list = []
# List with same type of elements
same_type_list = ['1', '3', '7', '10']
# List with different types of elements
diff_type_list = ['John', 'Dev', 1.90, True]
```

Now we know how to initialize the variable with list. Let's see some basic operations.

Basic operations with lists

Ever wanted to traverse the items in a list without going through them one-by-one? Python provides several useful functions. They allow you to manipulate them without iterating over the list or looping through each item.

The following are Python's five most used list operations:

1. len(list) – It returns the length of the list. It also helps in iteration when one wants to traverse the list.

For Example:

```
# Printing the length of the list
some_list = ['k', 'u',  'm', 'a', 'r']
print(len(some_list))
# Traversal of list
for i in range(len(some_list)):
        print(some_list[i])
```

```
# Output

5
k
u
m
a
r
```

2. max(list) – It returns the item in the given list with the highest value, if there is no tie then it returns an error.

For Example:

```
# Printing the maximum of the number stored in list
num_list = [1, 2, 3, 4, 5, 12, 78, 900, 100]
print(max(num_list))

```

```
# Output

900
```

3. min(list) – it returns the item in the given list with the lowest value, if there is no tie then it returns an error

For Example:

```
# Printing the minimum of the number stored in list
num_list = [1,2,3,4,5,12,78,900,100]
print(min(num_list))
```

```
# Output

1
```

4. sort(list) – This function sorts through all these data and puts them in ascending/descending order by default but if a key parameter is passing it sorts the list based on the evaluation of the function on the elements.

Reverse parameter controls whether the sorted(ascending order) list be given as it is sorted, or it gets reversed, i.e., in descending order.

The syntax is list.sort(reverse=True|False, key= some function)

For example:

```
num_list = [1,2,3,4,5,12,78,900,100]
print(num_list)
num_list.sort()
print(num_list)
num_list.sort(reverse = True)
print(num_list)
```

```
Output:

[1, 2, 3, 4, 5, 12, 78, 900, 100]
[1, 2, 3, 4, 5, 12, 78, 100, 900]
[900, 100, 78, 12, 5, 4, 3, 2, 1]
```

5. map(function, sequence) – This function here applies a function on each element of the list. The syntax is given by map(fun, iter). Here, 'fun' is the function that is supposed to be applied on every element of 'iter'.

For Example:

```
def square(n):
        return n * n

numbers = [1, 2, 3, 4]
result = map(square, numbers)
print(list(result))
```

```
output:
[1, 4, 9, 16]
```

There are so many other functions are there for lists. Now let's see what tuples are.

Python tuples

They can be created by simply declaring a tuple within parentheses, (), or by converting any sequence into a tuple using the built-in constructor tuple().

```
# Creating an empty tuple
empty_tuple = ()

seq_set = {1,2,3}
seq_list = [2,3,4,5]
print(type(seq))
print(type(seq_list))
# Converting set into tuple
seq_set_tuple = tuple(seq_set)
```

```
Output:
<class 'set'> <class 'list'>
# Creating an empty tuple
empty_tuple = ()

seq_set = {1, 2, 3}
seq_list = [2, 3, 4, 5]
print(type(seq_set))
print(type(seq_list))
# Converting set into tuple
seq_set_tuple = tuple(seq_set)
print(type(seq_set_tuple))

output:

<class 'set'> <class 'list'> <class 'tuple'>
```

Tuples are like lists with the difference that tuples are immutable. Then why do we use the tuples.

Difference between Python tuples and lists

Tuples are immutable, while lists are mutable. This means that tuples cannot be changed after they have been created, while lists can be edited to add or remove items.

Like a list, a tuple is also a sequence of data elements, which are not necessarily of the same type.

For Example:

```
# Tuple with same type of elements
same_type_list = ('1', '3', '7', '10')
print(same_type_list)
```

```
Output:

('1', '3', '7', '10')
```

```
# List with different types of elements
diff_type_list = ('John', 'Dev', 1.90, True)
print(diff_type_list)
```

```
# Output

('John', 'Dev', 1.9, True)
```

4. Python sets and dictionaries

This chapter will help the reader understand the basic Python sets and dictionaries with some basic applications in real world. We will be using Visual Studio Code as our code editor. If you have not installed Visual Studio Code, the instructions are given in the previous chapter.

Python sets

A set is a mutable and unordered collection of unique elements. Set is written with curly brackets ({}), being the elements separated by commas.

It can also be defined with the built-in function set([iterable]). This function takes as argument an iterable (i.e., any type of sequence, collection, or iterator), returning a set containing unique items from the input (duplicated elements are removed).

For Example:

```
# Create a Set using
# A string
print(set('Dev'))
```

```
Output:
{'e', 'v', 'D'}
```

```
# a list
set(['Mayank', 'Vardhman', 'Mukesh', 'Mukesh'])
```

```
Output:
{'Mayank', 'Mukesh', 'Vardhman'}
```

```
# A tuple
set(('Lucknow', 'Kanpur', 'India'))
```

```
Output:
{'India', 'Kanpur', 'Lucknow'}
```

```
# a dictionary
set({'Sulphur': 16, 'Helium': 2, 'Carbon': 6, 'Oxygen': 8})
```

```
Output:
{'Carbon', 'Helium', 'Oxygen', 'Sulphur'}
```

Now, we know how to create Sets. Let's see what the common operations in sets are.

Operations in Python sets

Adding an element in a set:

Syntax for adding element is set.add(element).

The method works in-place and modifies the set and returns 'None'.

For Example:

```
locations = set(('Lucknow','kanpur','India'))
locations.add('Delhi')
print(locations)
```

```
Output:
{'India', 'Delhi', 'Lucknow', 'kanpur'}
```

In Python sets, we cannot insert an element in a particular index because it is not ordered.

Removing an element from a set:

There are three methods using which you can perform the removal of an element form a set.

They are given below:

- set.remove(element)
- set.descard(element)
- set.pop()

Let's understand this by looking at an example for each implementation:

set.remove(element)

```
locations = set(('Lucknow', 'kanpur', 'India'))
#Removes Lucknow from the set
locations.remove('Lucknow')
print(locations)
```

```
Output:
{'India', 'kanpur'}
```

set.discard(element)

```
locations = set(('Lucknow', 'kanpur', 'India'))
# Removes 'Lucknow' from the set
locations.discard('Lucknow')
print(locations)
```

```
Output:
{'India', 'kanpur'}
```

As you can see that both the 'remove' and 'discard' method work in-place and modify the same set on which they are getting called. They return 'None'.

The only difference that is there in the 'remove' and 'discard' function is that 'remove' function throw an exception (KeyError) is raised, if 'element' is not present in set. The exception is not thrown in case of 'discard'.

set.pop()

```
locations = set(("Lucknow", 'Kanpur', 'India'))
# Removes 'Lucknow' from the set
removed_location = locations.pop()
print(locations)
print(removed_location)
```

```
Output:
{'Kanpur', 'Lucknow'}
India
```

'pop' function does not take any arguments and removes any arbitrary element from set. It also works in-place but unlike other methods it returns the removed element.

So, we have covered lists, tuples, and Python sets. Now, finally let's see how things work in Python dictionaries.

Dictionaries in Python

Python dictionaries are a fundamental data type for data storage and retrieval.

The dictionary is a built-in data structure that stores key:value pairs and can be accessed by either the key or the value. Python dictionaries are unordered, and keys cannot be negative integers. On top of that, while keys must be immutable, values do not have to be.

The syntax for creating a dictionary is to place two square brackets after any sequence of characters followed by a colon (e.g., {'a': 'b'}); if you are passing in more than one sequence then you need to put them in separate sets of brackets (e.g., {'a': 'b', 'c': 'd'}).

For Example:

```
# Creating an empty Dictionary
Dictionary = {}
print("Empty Dictionary: ")
print(Dictionary)
```

```
Output:
Empty Dictionary: {}
```

We can also create a dictionary using in=built function known as 'dict()'.

Let's see how we can create it:

```
# Creating a Dictionary
# With dict() method
Dictionary = dict({1: 'Hello', 2: 'World', 3: '!!!'})
print("\nDictionary by using dict() method: ")
print(Dictionary)
```

```
Output:
Dictionary by using dict() method:
1: 'Hello', 2: 'World', 3: '!!!'}
```

Now, let's create the dictionary using a list of tuples of key and value pair:

```
# Creating a Dictionary
Dict = dict([(1, 'Hello'), (2, 'World')])
print("\nDictionary by using list of tuples of key and value as a pair: ")
print(Dict)
```

```
Output:
Dictionary by using list of tuples of key and value as a pair:
{1: 'Hello', 2: 'World'}
```

Remember that the keys are case-sensitive.

Let's see briefly what are the methods that are present in dictionary of Python.

Difference Between Python sets and dictionaries

A set is a collection of values, not necessarily of the same type, whereas a dictionary stores key-value pairs.

Python sets are collections of data that do not have any order or keys. A set does not store any data about its members other than their identity. Dictionaries are collections that map unique keys to values. Furthermore, dictionaries store information about their members, including the key and value pair.

So, we built some basic understanding about Lists, Tuples, Sets and Dictionaries in Python. We also investigated some functions and their implementations.

5. Conditional statements in Python

We have covered the basic data types and advanced data types in Python in the previous chapter. In this chapter, the conditional statements will be covered.

Conditional Statements in Python – what do they do?

The conditional statements in Python regulate the flow of the code execution. In a very layman term, these statements are used when you want the program to do a task if a condition is satisfied and not to do the same task when the condition is not fulfilled.

Python input()

Up till now, we have just printed out the output but never gave any input to our program. In Python input() is used for giving input to the program in Python. The example is illustrated below.

For Example:

```
1 | # Take input
2 | x=input()
3 | print(x)
```

The above code will ask for an input which will be stored in the X variable for further usage.

```
1 | Output:
2 | 5
3 | 5
```

The input can also have a string query in it. The example is illustrated below.

```
1 | # Take input
2 | x=input("please enter your age?")
3 | print(x)
```

```
1 | Output:
2 | please enter your age. 5
3 | 5
```

Even the input can be modified using the datatype functions used in the typecast of a datatype. The example is illustrated below.

```
# Take input
x=int(input("please enter your age?"))
y=input("please enter your age?")
print(type(x))
print(type(y))
```

```
Output:
please enter your age. 5
please enter your age. 5
<class 'int'>
<class 'str'>
```

In the above example, we can see that the input without any typecast function is a string value. Hence, the default value for input is string.

If statement in Python

If a program has only a single decision to make, then one "if" statement is used. Let's take an example where we want to allow a person only if he or she has a mask.

```
#if statement
mask=True
if mask==True:
  print("the person can enter")
```

Syntax in Python

The syntax is quite simple, it's followed by the condition and indentation of one tab space whenever there is something in the if statement. When we discussed the operators in the variable's chapter, we discussed comparison operators, logical operators, and mathematical operators.

In this condition, both comparison operators and logical operators can be used. In the above example, we can see that we used "==" operator for comparison. In the above program if the mask is True then the statement will be printed, otherwise it will not print anything.
Let's execute the program and examine the output.

```
Output:
the person can enter
```

What will happen if we change the make value to False? The output will be as given below. Which is empty – nothing will be printed as the condition is not fulfilled.

```
Output:
```

If else in Python

In the above example, we just have a condition, which says if a person has a mask, they can enter. But there is not, otherwise, what to do if the person doesn't have a mask. Hence, it seems to be an incomplete program. Let's say, if they don't have a mask, we want them to get a mask to enter. For this we will be using else statement which executes only when the "if" statement condition is not fulfilled.

An example is illustrated below.

```
#if else statement
mask=True
if mask==True:
  print("the person can enter")
else:
  print("please, get a mask to enter")
```

Now if we change the value of the mask to False, then we will get "please, get a mask to enter")

```
#if else statement

mask=False

if mask==True:
  print("the person can enter")
else:
  print("please, get a mask to enter")
```

```
Output:
please, get a mask to enter
```

This can also be written in the below format.

```
#if else statement

mask=False

if mask==True:
  print("the person can enter")
print("please, get a mask to enter")
```

In Python, whenever you write a statement after the if without indentation, it taken to be under else statement.

Now let's add a case, where if a person doesn't have a mask but is willing to buy it , can buy the mask from the guard itself and enter. For this, we will change our earlier code a bit. We will give string values such as "nobuy" ,"buy", "yes". Now we will use these to write our if statements.

Now the according to the mask value, the execution will be done. If the mask value is "nobuy", we will get the output to be "please, get a mask to enter".

```
#if else statement

mask="nobuy"

if mask=="yes":
  print("the person can enter")
elif mask=="buy":
  print("person bought the mask and can enter")
print("please, get a mask to enter")
```

```
Output:
please, get a mask to enter
```

Even if the mask is given any other value, we will get the result to be "please, get a mask to enter". This is because above two if statement conditions will not be fulfilled.

```
#if else statement
mask="yes"

if mask=="yes":
  print("the person can enter")
elif mask=="buy":
  print("person bought the mask and can enter")
print("please, get a mask to enter")
```

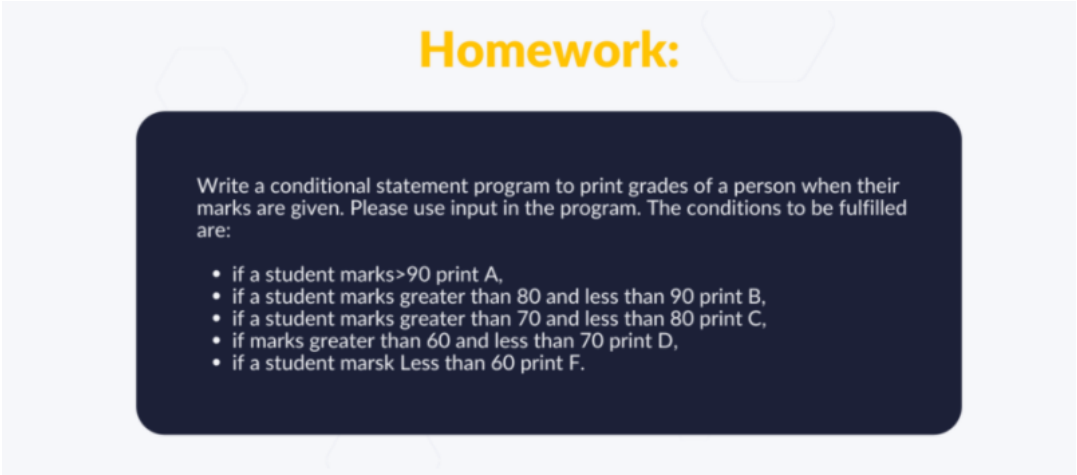
```
Output:
the person can enter
```

For "buy" in the mask, the output will be ("person bought the mask and can enter").

```
#if else statement
mask=”yes”
if mask==”yes”:
  print(“the person can enter”)
elif mask==”buy”
  print(“person bought the mask and can enter”)
print(“please, get a mask to enter”)
```

```
Output:
the person bought the mask and can enter
```

In this chapter, we have covered some basics of conditional statements in Python, the further topics on functions will be covered in the next chapter. From this chapter onward, the reader will be given some practice questions.



6. Loops in Python

We have covered the basic data types, advanced data types and conditional statements in Python in our previous chapters. In this chapter, the loops will be covered. If you are new to Python, please start from the first chapter to get a better understanding of this blog.

Loops in Python

Loops are used when there is a need to do a task more than one time. For example, printing numbers from 1 to 100 or a better example would be to sum all the elements in a list or an array. Sometimes there is a need to write more than 1 loop or loop inside a loop. In Python writing these loops is very simple, and even the syntax is easy to understand. As we have seen, in Python we don’t need to declare a variable first before using it. The basic looping starts with for loop. Let’s understand “for” loop.

For loop in Python

In a for loop, we have three things which needs to be mentioned. First one is the initial value of the variable on which the iteration needs to be done, the stopping condition and the last one is by how many steps you want to increment or decrement the iterator.

Let's see the syntax of a "for" loop:

```
# For Loop

for var in range(10):
  print(var)

for var in range(0,10,1):
  print(var)
```

In the above code illustration, we can see that for loops are giving the same result. The syntax in the end where we provided the function range has three arguments, which we discussed in the previous paragraph. In the above example the range has 0,10,1 in which 0 is the initial value of the iterator, 10 is the final value, but the range actually iterates till 10-1 which is 9 and 1 is the incrementing of the iterator every time the loop runs.

Let's run the above program.

```
Output:
0
1
2
3
4
5
6
7
8
9

0
1
2
3
4
5
6
7
8
9
```

As we can see from the output illustration, it's printing 0 to 9 numbers.

For loops in List

In a list we have a collection of items and below is the illustration on how to use for loops to iterate through a lis

```
X=[1,2,3,4,5,6]
for i in X:
  print(i)
```

```
Output:
This will print all the elements in the list.
1,2,3,4,5,6
```

To include the index also while printing, the code is illustrated below.

```
X=[1,2,3,4,5,6]
for i in range(len(X)):
  print(i,X[i]
```

This will print both index and the value in the list.

There is an easy way to get the index and value using enumerate function. The enumerate function use is illustrated below.

```
X=[1,2,3,4,5,6]
for i,value in enumerate(X):
  print(i,value)
```

```
Output:
0,1
1,2
2,3
3,4
4,5
5,6
```

Iterating a set using for loop

Iterating a set is like the list iteration using for loop. An example is illustrated below.

```
X={1,2,3,4,5,6}
for i,value in enumerate(X):
  print(i,value)
```

```
Output:
0,1
1,2
2,3
3,4
4,5
5,6
```

Iterating a tuple using for loop

Iterating a tuple is like the list iteration using for loop. An example is illustrated below.

```
X=(1,2,3,4,5,6)
for i,value in enumerate(X):
  print(i,value)
```

```
Output:
0,1
1,2
2,3
3,4
4,5
5,6
```

Iterating a dictionary using for loop

Iterating a dictionary is different from the other data types, as the dictionary contains key-value pairs. Hence to get just keys we use dictionaryname.keys() and for values we use dictionaryname.values(). An example is illustrated below.

```
X={"1":1,"2":2}
for key in X.keys():
  print(key)
for value in X.values():
  print(value)
for key,value in X.items():
  print(key,value)
```

```
Output:
1
2

1
2

1,1
2,2
```

Nested loops in Python

Nested loops are useful when building a brute force solution to a given problem. They increase time complexity of the program and decrease readability.

```
a = [1, 2]
b = [10, 13]
# getting numbers whose product is 13

for i in a:
  for j in b:
    if i*j == 13:
      print(i, j)
```

In the above coding block, we defined 2 lists and each list has some collection of numbers. The main aim was to find what numbers product will be 13 from both the lists and also to print those numbers. For this purpose, we have to iterate through 2 lists, hence 2 for loops were used.

Alternative way:

There is a function in itertools which is called product. This helps in keeping the nested for loops if present in the program readable. The example is illustrated below.
from itertools import product

a = [1, 2]
b = [10, 13]
# getting numbers whose product is 13

for i, j in product(a, b):

```
if(i*j == 13):
  print(i, j)
```

While Loops in Python

Up till now, we have just printed out the output but never gave any input to our program. In Python the input() is used for giving input to the program in Python. The exam is illustrated below. The while loop is used when you want to execute a program if the condition is fulfilled. While loop examples are illustrated below.

Printing 0-9 using while loop:

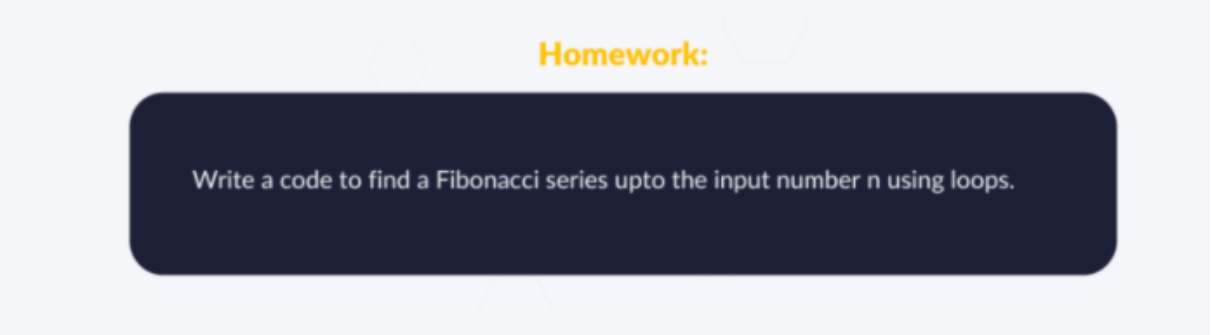
i = 0

```
while(i < 10):
  print(i)

  i += 1
```

As you can see, the syntax is while followed by a condition, and inside the loop we increment the iterator according to the desired number.

```
Output:
0
1
2
3
4
5
6
7
8
9
```

In this chapter, we have covered some basics of looping statements in Python, the further topics on functions will be covered in the next chapter. The question to be solved is given below.



7. Python

Python functions are objects that means the functions can be used as return value for other functions, can be stored in a variable, can be stored in data structures, or can be used as an argument in other functions.

Python functions are defined using “def” keyword following the function name. Then inside these brackets “()” , the arguments are defined. The basic syntax of Python functions is illustrated below.

For Example:

```
# Create a function
# def keyword
def functioname():
```

Note:

Function name is also having the same norms as the variable declaration. Let's write our first function.

```
# first function

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

In the above code block, we have written a function that gives us the sum of two numbers. As you can see, we have used "def" keyword, a and b are the arguments which in our case would be the numbers we want the sum for. Now, we have used a keyword here called "return" which is used to return the desired value or string from the function after performing the desired task. The values which are returned by using returned keywords can be further assigned to other variables or can be used in functions as an argument.

Let's now see, how to use this function on our desired numbers.

```
# first function

def sum(a,b):
  return a+b

sum(6,7)

x=sum(6,7)
print(x)
```

As you can see if we just use the function, the function will not show any value, but when we store the functions return value in another variable and print it, it gives the desired result.

Let's run the program and see the output.

```
# Output

13
```

We have got the output as 13, which is the sum of 6 and 7. Let's write another function which gives us fullname given firstname and lastname.

```
# second function
def fullname(fn,ln):
  return fn+ln

x=fullname("Python","language")
print(x)
```

As you can see, we have just defined the function fullname and gave it parameters firstname and lastname. We are returning fullname using "+" which is a concatenation operator in string which we learned in the variables chapter.

Let's explore the output.

```
#Output

Pythonlanguage
```

Python functions as objects

Most of the data in Python is represented in the form of objects. In Python, strings, modules, functions are all represented in the form of objects. Let's see how we can use functions as objects.

Assigning functions to a variable

As a function is an object, it can be assigned to a variable. An example is illustrated below.

```
# first function

def sum(a,b):
  return a+b

sumab=sum
```

In the above example, we can see that assigning it to a new variable doesn't call the function, instead it just assigns the function to the variable "sumab". The actual meaning of the above example is that the variable "sumab" takes the sum function object as a reference, and the "sumab" now points to that object. Hence, the sumab can also be used as a function now. An example is illustrated below.

```
# New function

def sum(a,b):
  return a+b

sumab=sum

s=sumab(7,8)
print(s)
```

Output:

```
#output

15
```

Note:
The function name which we give in the declaration and the function objects work very differently. Even if we delete the original function name, if there is another name pointing to that reference function object, still the function will work. An example is illustrated below.

```
# New function

def sum(a,b):
  return a+b

sumab=sum

del sum

sum(8,7)
```

Output:

```
#Output

NameError: “name ‘sum’ is not defined”
```

But when we use the sumab function, then the result is illustrated below.

```
# New function

def sum(a,b):
  return a+b

sumab=sum

del sum

sumab(8,7)
```

Output:

15

Storing Python functions in Data Structures

As the functions are objects in Python, we can store them in data structures in the same way we store our variables and constants. The syntax changes a little, but it's like how we stored elements in the datatypes.

```
#function storing in datastructures

Storedfunctionslist=[len,str.upper(),str.strip(),str.lower()]

Storedfunctionslist
```

Iterating through functions is just like iterating objects. Example illustrated below.

```
#function storing in datastructures

Storedfunctionslist=[len,str.upper(),str.strip(),str.lower()]

for fun in Storedfunctionslist:
        print(fun, fun('Hello'))
```

In this chapter, we have covered some basics Python functions, the further detailed topics on functions will be covered in the next chapter.

8. Advanced functions in Python

This chapter will help the reader continue from the previous Python functions chapter along with some basic applications in the real world. We will be using Visual Studio Code as our code editor. If you have not installed Visual Studio Code, the instructions are given in the first chapter.

Passing functions to other functions

As discussed in the previous blog, functions in Python are treated as objects.Just like objects in Python, functions can also be passed as an argument to another function.

For Example:

```
def mul(x, y):
  return x*y


def add(mul, y):
  return mul+y


x = add(mul(9, 10), 10)
print(x)
```

In the above code block, it can be seen that the mul function is passed as an argument to the add function and it is stored in the x variable which is then printed to verify the answer.

```
100
```

Using functions inside a function

In Python, we can define a function inside another function. These functions are called as nested functions. But in this use case, the inside function or nested function cannot be called separately. Both the examples are illustrated in the following code block.

Let's write our first function.

```
def mul(x, y):

  m = x*y

  def square(m):
    return m*m

  return square(m)


x = mul(9, 10)

print(x)
```

```
Output:
8100
```

In the above code block the outer function is "mul" which returns the function square which takes an argument "m" which is the multiplication of two arguments given to "mul" function. The code execution first starts with calling "mul" function, then the product of "x" and "y" gets stored in "m" variable. As this function is returning square function, the "square" function is called and the final product which is the square of "m" is returned.

Let's learn a few important things in Python, which will make your coding journey with Python much better.

*Args in Python

These are the arguments which we use as function parameters. Let's write a usual function using what we learned till now. We will be writing a function that can give us the maximum area of a rectangle given 2 rectangle areas as parameters to the function.

```
def maxarea(a, b):
  if a > b:
    return f'rectangle a has the more area which is {a}'
  else:
    return f'rectangle a has the more area which is {b}'


x = maxarea(100, 60)
print(x)
```

```
Output:
rectangle a has the more area which is 100
```

This function is good for 2 parameters or arguments, but what if we need to compare more than 2 areas? One approach would be passing a list of areas into the function.

```
def maxarea(lis):

  max = 0
  for i in lis:
    if i > max:
      max = i

  return f"rectangle which has the more area is {max}"


x = maxarea([100, 60, 50])
print(x)
```

```
Output:
rectangle which has the more area is 100
```

This approach is good, but we should know the number of parameters or arguments to give beforehand. In real-time code execution, this would be a hassle. Hence, to make the programmer's life easy, Python uses *args and **kwargs.

"*" operator in Python

This operator is an unpacking operator which is usually used to pass an unspecified number of parameters or arguments.

Unpacking arguments into tuple using * operator

As we have seen, that "*" operator is used for unpacking values. The example is illustrated below.

```
x = [1, 2, 3, 4]
y = [5, 6, 7, 8]

z = *x, *y

print(type(z))
print(z)
```

```
Output:
<class 'tuple'>
(1, 2, 3, 4, 5, 6, 7, 8)
```

As we can see, the unpacking operator has unpacked list x and list y into a tuple that is z. We can also see that the result is a tuple.

Let's write the same function using *Args.

```
def maxarea(*lis):

    max = 0
    for i in lis:
        if i > max:
            max = i

    return f"rectangle which has the more area is {max}"


x = maxarea(100, 60, 50, 200)
y = maxarea(100, 60, 50, 200, 9000)
z = maxarea(100, 60, 50, 20, 90)
print(x)
print(y)
print(z)
```

```
Output:
rectangle which has the more area is 200
rectangle which has the more area is 9000
rectangle which has the more area is 100
```

In this code block we can see that now the arguments are dynamic, we can add any number of arguments that will be unpacked in the maxarea function to give us the desired result. Also, we can compare any number of areas in this context.

**kwargs in Python

The kwargs is like args, but it accepts positional arguments. It uses ** operator which has some attributes like unpacking multiple positional arguments of any length, can also unpack dictionaries, can also be used for combining two dictionaries.

Merging dictionaries

```
a = {"h": 1, "n": 2}
b = {"m": 5, "l": 10}

c = {**a, **b}

print(type(c))
print(c)
```

We can see from the above code that we have 2 dictionaries a and b which are merged using ** operator to give another dictionary.

Output:

```
<class 'dict'>
{'h': 1, 'n': 2, 'm': 5, 'l': 10}
```

When we use * operator instead of ** operator, the code for this case is illustrated below.

```
a = {"h": 1, "n": 2}
b = {"m": 5, "l": 10}

c = {*a, *b}

print(type(c))
print(c)
```

```
Output:
<class 'set'>
{'n', 'l', 'm', 'h'}
```

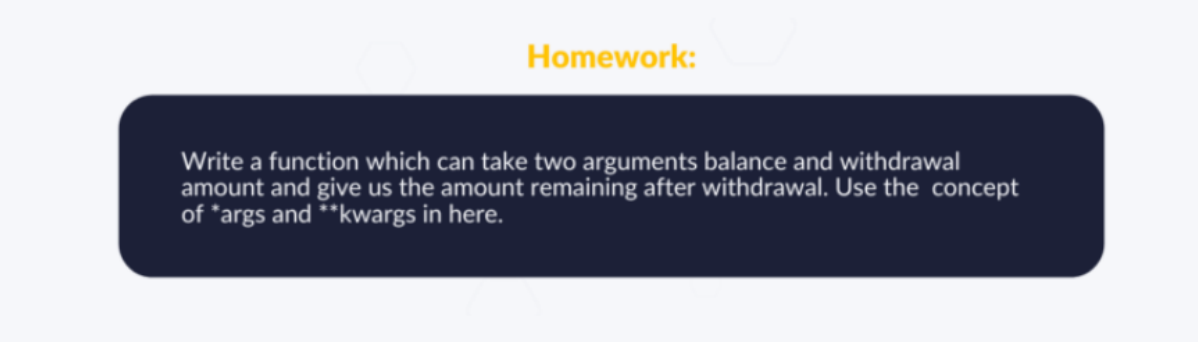
Hence, when the * operator is used on two dictionaries to merge, the out will be a set with only the keys from the dictionary.

The maxarea function using **kwargs is illustrated in the below code block.

```
def maxarea(**lis):

    max = 0
    for i in lis.values():
        if i > max:
            max = i

    return f"rectangle which has the more area is {max}"


x = maxarea(a=1, b=2, c=3)
y = maxarea(a=1, b=2)
z = maxarea(a=1, b=2, c=3, d=9)
print(x)
print(y)
print(z)
```

```
Output:
rectangle which has the more area is 3
rectangle which has the more area is 2
rectangle which has the more area is 9
```

In this chapter about advanced functions in Python, we have covered topics like passing functions to other functions, using functions inside a function, *Args in Python, "*" operator in Python, **kwargs in Python, and more. The further topics which include classes will be covered in the next chapter. Homework regarding advanced functions in Python is given below.



9. Python classes and objects

This chapter will help the reader understand the basic Python classes along with some basic applications in real world.

As we have discussed in the first chapter, Python is an object-oriented programming language. There are three phrases that are very important while discussing object-oriented programming in Python. The first one is class, the second one would is an object, the third one would be the inheritance. Let's start with what is a class.

Python classes – definition

A class is a blueprint or an extensible program that helps in the creation of objects. A class consists of behavior and state. The behavior of a class is demonstrated through functions inside the class, which are called methods. The state of the class is demonstrated using the variables inside the class, which are called attributes.

Initialization of Python classes

A class can be initialized using the following syntax.

A class in Python is defined using "class" keyword following the class name. The basic syntax of Python function is illustrated below.

For Example:

```
1  <img src="https://firmbee.com/wp-content/uploads/Python_9-800x600.png" alt="Python_classes" width="800"
   height="600" class="alignnone size-medium wp-image-21409 img-fluid" />
2  # Create a function
3  # class Monkey
4  class classname:
```

Note: class name is also having the same norms as the variable declaration.

Let's write our first Python class

```
1  # first class
2
3  class Animals:
4    pass
5
```

Let's now see, how to add components to the animal's class. But before that let's learn about the "__init__()" constructor. Constructors are used for object instantiation. Here the __init__() is used for object instantiation. The constructor can be default with only self as an argument or parametrized with required arguments.

Attributes

There are two different types of attributes, the first ones are class variables and the second ones are instance variables. The class variables are the variables that are owned by the class. Also, these variables are available to all instances of the class. Hence, their value will not change even if the instance changes.

```
1  # class variables
2
3  class Animals:
4    type=”mammals”
```

The instance variables are the variables that belong to the instances itself. Hence, they will change their value as the instance changes.

```
1  # class variables
2
3  class Animals:
4    def __init__(self,legs):
5      self.legs=legs
6
```

Note: Instance variables are not available for access using class name, because they change depending on the object accessing it.

Let’s make a program that has both class and instance variables declared.

```
1  class Animals:
2    type=”mammals”
3    def __init__(self,name,legs):
4      self.name=name
5      self.legs=legs
6
7
```

In the above program, we have used both instance and class variables. So, these variables form attributes of the class.

Behavior of the class

As discussed, behavior of the class is defined by the methods inside the class. But before going into the discussion on behavior, we have to start discussing the “self” parameter, which we used in the __init__().

Self:

In a very simple term, whenever we attach anything to self it says that the variable or function belongs to that class. Also, with “self”, the attributes or methods of the class can access.

Methods:

Class methods are functions inside the class which will have their first argument as “self”. A method inside the class is defined using “def” keyword.

```
01  class Animals:
02    type=”mammals”
03    def __init__(self,name,legs):
04      self.name=name
05      self.legs=legs
06    def bark(self):
07      if self.name==”dog”:
08        print(“woof woof!!!”)
09      else:
10        print(“not a dog”)
11
```

In the above method “bark”, as we are using the name variable, which is an instance variable, we are accessing it using “self” and this function would print “woof woof!!!”, only if the name provided to the object, is dog.

We have discussed most of the components of a class, but you might be thinking how to see if the class is working. The answer to this is unless we create an object of the class, we will not be able to see what the class is doing. Now, Let’s define and create an object of the class.

Objects in Python

An object is an instance of the class. A class is just a blueprint, but the object is an instance of the class which has actual values.

The code for defining or creating an object is illustrated below.

```
01  class Animals:
02    type=”mammals”
03    def __init__(self,name,legs):
04      self.name=name
05      self.legs=legs
06    def bark(self):
07      if self.name==”dog”:
08        print(“woof woof!!!”)
09      else:
10        print(“not a dog”)
11
12  dog=Animals(“dog”,4)
13
```

To create an object, the syntax is the objectname=classname(arguments). Hence, here we are giving the name of the animal to be dog and number of legs to be 4. Now, the object of the class is created, the next step is to use the object to access its attributes. To access the attributes of a class using the object, remember only the instance variables can be accessed using the object. The instance variables in our class are name and legs.

```
01  class Animals:
02    type=”mammals”
03    def __init__(self,name,legs):
04      self.name=name
05      self.legs=legs
06    def bark(self):
07      if self.name==”dog”:
08        print(“woof woof!!!”)
09      else:
10        print(“not a dog”)
11
12  dog=Animals(“dog”,4)
13  print(dog.name)
14  print(dog.legs)
15
```

As we can see, we are able to access instance variables using dot notation.

Let’s explore the output.

```
1  #Output
2
3  dog
4  4
5
```

To access the functions inside the class or methods, we will be using the dot notation. The example is illustrated below.

```
class Animals:
  type="mammals"
  def __init__(self,name,legs):
    self.name=name
    self.legs=legs
  def bark(self):
    if self.name=="dog":
      print("woof woof!!!")
    else:
      print("not a dog")

dog=Animals("dog",4)
print(dog.name)
print(dog.legs)
print(dog.bark())
```

```
#Output

dog
4
woof woof!!!
```

In the above example, we can see that we are accessing the class method "bark" using the dog object we created. We can see that we are not using the "self" parameter in the function arguments. This is because we don't require the use of "self" outside the class as the object itself is acting as self.

Inheritance

Inheritance is a process through which the class attributes and methods can be passed to a child class. The class from where the child class is inheriting is the parent class. The syntax for inheritance is illustrated below.

```
#Inheritance

class parent:

class child(parent):

```

From the above illustration, we can see that for the inheritance syntax we are placing the parent class name as an argument to the child class. Let's use the Animals class and make a child class called dog. This is illustrated below.

```
class Animals:
  type="mammals"
  def __init__(self,name,legs):
    self.name=name
    self.legs=legs
  def bark(self):
    if self.name=="dog":
      print("woof woof!!!")
    else:
      print("not a dog")

class Dog(Animals):
def __init__(self,name,legs,breed):
Animals.__init__(self,name,legs)
self.breed=breed
```

In the above example code, we made a dog class which is extending the animals class which we created before. We are also using the parameters from the Animals using the Animals.__init__(arguments) which has name and legs which will be inherited to the dog class. Then we are making an instance attribute for the dog class which is breed.
Now let's make an object for the dog class and access the attributes and methods of the animals class.

```
class Animals:
  type="mammals"
  def __init__(self,name,legs):
    self.name=name
    self.legs=legs
  def bark(self):
    if self.name=="dog":
      print("woof woof!!!")
    else:
      print("not a dog")

class Dog(Animals):
def __init__(self,name,legs,breed):
Animals.__init__(self,name,legs)
self.breed=breed


pug=Dog("dog",4,"pug")
pug.breed
pug.name
pug.legs
pug.bark()
```

```
#Output

pug
dog
4
woof woof!!!
```

As we can see from the output the parent class attributes and methods are being accessed by the child class object.

In this chapter, we have covered some basics of classes in Python. In the next chapter we will cover the topic of file handling.

10. Files in Python

This chapter will help the reader understand the basic Python files and file handling along with some basic applications in real world.

Files in Python – definition:

A file is an entity that stores information. This information may be of any type such as text, images, videos, or any music. In Python, there are functions inbuilt which can be used to perform operations on files.

Examples of binary files in Python:

- Document files: .pdf, .doc, .xls etc.
- Image files: .png, .jpg, .gif, .bmp etc.
- Video files: .mp4, .3gp, .mkv, .avi etc.
- Audio files: .mp3, .wav, .mka, .aac etc.
- Database files: .mdb, .accde, .frm, .sqlite etc.
- Archive files: .zip, .rar, .iso, .7z etc.
- Executable files: .exe, .dll, .class etc

Examples of text files in Python:

- Web standards: html, XML, CSS, JSON etc.
- Source code: c, app, js, py, java etc.
- Documents: txt, tex, RTF etc.
- Tabular data: csv, tsv etc.
- Configuration: ini, cfg, reg etc

Operations on files in Python

Opening a file in Python:

The open() function in Python is used for opening files. This function takes two arguments, one is the filename and the other one is the mode of opening. There are many modes of opening such as read mode, write mode, and others.

Let's explore the syntax:

```
# File opening in Python

File=open("filename","mode")

```

Modes of file opening:

- **"r":**– this is used for opening a file in read mode.
- **"w":** – this is used for opening a file in write mode.
- **"x":** – this is used for exclusive file creation. If the file is not present, it fails.
- **"a":** – this is used when you want to append a file without truncating the file. If the file is not present, then this creates a new file.
- **"t":** – this is used for opening file in text mode.
- **"b":** – this is used for opening file in binary mode.
- **"+":** – this is used when the user wants to update a file.

Note:
The operations for binary files are as given below.

Let's open a file using above discussed methods. The code is illustrated below. As we don't have any file, we will create a file and then open it.

```
x="new file opening"

with open("new","w") as f:
  f.write(x)
```

In the above code, we are creating a string variable x which contains the text "new file opening", this string variable is being written into a file "new" using write method. We are using "with" here as it handles closing of the file. So, we are opening a file in write format and writing the string x to the file.

Now, let's read the same file.

```
x="new file opening \n writing new file"

with open("new","r") as f:
  print(f.read())
```

In the above code, we are opening the file new which we wrote in the previous code and opening it in read format. Note that, we are using read() function to read the file. Let's run and see the output.

```
#output

New file is opening
```

Functions involved in reading files in Python

There are three functions involved in the reading operation performed on files.

Read():

This function is used when the user wants to read all the information inside the file.

```
x="new file opening \n writing new file"

with open("new","r") as f:
  print(f.read())
```

Readline():

This function is used when the user wants to read the file line by line.

```
x="new file opening \n writing new file"

with open("new","r") as f:
  print(f.readline())
```

Readlines():

This function reads all the lines but in a line by line fashion which increases its efficiency in handling memory.

```
x="new file opening \n writing new file"

with open("new","r") as f:
  print(f.readlines())
```

Appending a file:

As discussed above, we will be opening a file in append mode which "a+" for appending it. The code is illustrated below.

```
x="new file opening"

with open("new","a+") as f:

  f.write("Hello world")
```

```
Reading the file to see the appended line:
x="new file opening"

with open("new","r") as f:

  print(f.read())

```

Let's explore the output:

```
new file openingHello world
```

Renaming a file:

For renaming a file, we will be using the methods present in the "os" module of Python. The code is illustrated below.

```
import os

os.rename("new.txt","example.txt")
```

In the above code, we are importing the "os" module and using "rename" method to rename the file we create from "new" to "example".

Removing a file:

For removing files, we will be using the same module "os" which we have used for renaming the file. The example of the code is illustrated below.

```
1  import os
2
3  os.remove("example.txt")
```

Copying a file:

For copying the file, we will be using the same module "os" which we have used for renaming and removing a file. The example of the code is illustrated below.

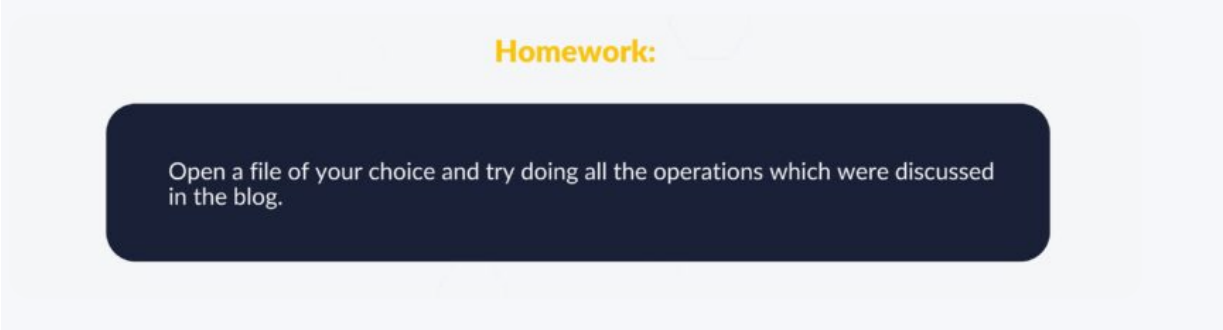
```
1  import os
2
3  os.system("cp example example1")
```

Moving a file:

For moving the file, we will be using the same module "os" which we have used above. The example of the code is illustrated below.

```
1  import os
2
3  os.system("mv source destination")
```

In this chapter, we have covered some basics when it comes to files in Python. In the next chapter we will use all the gathered knowledge in practice.



11. Python applications in practice

In this chapter, we will help the reader use the learningS from all the previous chapters to make a mini-project. You'll discover Python applications in practice.

Creating a guessing numbers game

This mini-project will be exciting to learn on how we can use functions and most of the other things which we learned in the previous chapters. This mini-project game generates a random number from 1 to 1000 or if you want it to be easy you can decrease the range and the user who is playing the game must guess the number. Sounds exciting, doesn't it? What will make it more exciting is that we can give the user some cues if he guesses the number wrong so that they can guess the number correctly.

Let's make a blueprint for the game with Python applications in practice.

Intro command line:

In the intro command line, we will ask the user to guess a number. We will ask his name and age. Then we will ask him if he wants to play the game or not. Let's do this in the code.

```
# Intro Panel Command line

print("Welcome to the guessnum")

name=input("what is your name?")
print(f"Hello {name}")
```

```
Output:
Welcome to the guessnum
Hello john
```

As can be seen, we first introduced our game to the user, and then we asked the user their name. We greeted them using the saved name. Now let's ask the user the age.

```
# Intro Panel Command line

print("Welcome to the guessnum")

name=input("what is your name?")
age=int(input(f"Hello {name}, what is your age?"))
print(f"Hello {name}")
```

```
Output:
Welcome to the guessnum
Hello john
```

In here we are seeing fstring, this is alternative to format, if we write f followed by a string, we can use our stored variables inside the "{}" directly.

Now we can see most of the intro panel. Now let's ask the user if he wants to play the game and if he wants to play the game, let's ask him to guess a number, so that we can say if it's right or not. But before asking the user to guess the number, we must have the number of the program ready.

Let's see how it is done in code.

```
# Intro Panel Command line

print("Welcome to the guessnum")

name=input("what is your name?")
age=int(input(f"Hello {name}, what is your age?"))
choice=input(f"Hello {name}, would you like to play the game? y/n")

if choice=="y":
  pass
else:
  print("exiting")
  exit

```

Now we are making another prompt which will ask the user, whether he wants to play the game, and we will be using the conditionals which we learned in the previous chapter to continue if he says yes and if it's no, to exit the game.

Now let's continue expanding our game and ask the user for the number, but before that let's make our code select a random number.

```
# Intro Panel Command line
import random
print("Welcome to the guessnum")

name=input("what is your name?")
age=int(input(f"Hello {name}, what is your age?"))
choice=input(f"Hello {name}, would you like to play the game? y/n")

if choice=="y":
  number=int(random.randint(1,5))
  guess=int(input("Please input your guess"))
  print(f"your guess is {guess}")
else:
  print("exiting")
  exit


```

```
Output:
Welcome to the guessnum
your guess is 2
```

Now we added an import known as random, which selects a random number from a given range.

The function is random.randint(start,end). Then we are asking our user to guess the number and we are printing our user's guesses.

Let's also print our program's guess.

```
# Intro Panel Command line
import random
print("Welcome to the guessnum")

name=input("what is your name?")
age=int(input(f"Hello {name}, what is your age?"))
choice=input(f"Hello {name}, would you like to play the game? y/n")

if choice=="y":
  number=int(random.randint(1,5))
  guess=int(input("Please input your guess"))
  print(f"your guess is {guess} and program's guess is {number}")
else:
  print("exiting")
  exit
```

```
Output:
Welcome to the guessnum
your guess is 2 and the program's guess is 5
```

So, we can see that we are almost halfway, we have the guess of the program and the guess of the user.

Now we can just compare and print if the user is correct or not.

```
# Intro Panel Command line
import random
print("Welcome to the guessnum")

name=input("what is your name?")
age=int(input(f"Hello {name}, what is your age?"))
choice=input(f"Hello {name}, would you like to play the game? y/n")

if choice=="y":
  number=int(random.randint(1,5))
  guess=int(input("Please input your guess"))

  if guess==number:
    print("you guessed it right!!!")



  print(f"your guess is {guess} and program's guess is {number}. Sorry!!! your guess is wrong")

else:
  print("exiting")
  exit
```

```
Output:
Welcome to the guessnum
your guess is 2 and the program's guess is 1. Sorry!!! your guess is wrong
```

As you can see, I have guessed wrong, but maybe you can guess it right? This game can be made more interesting by adding the score factor.

Now let's code for the score factor.

```
# Intro Panel Command line
import random
print("Welcome to the guessnum")

name=input("what is your name?")
age=int(input(f"Hello {name}, what is your age?"))
choice=input(f"Hello {name}, would you like to play the game? y/n")
correct=0




while(choice=="y"):
    number=int(random.randint(1,5))
    guess=int(input("Please input your guess"))

    if guess==number:
        print("you guessed it right!!!")
        correct+=1
        choice=input(f"Hello {name}, would you like to continue the game? y/n")




    print(f"your guess is {guess} and program's guess is {number}. Sorry!!! your guess is wrong")
    choice=input(f"Hello {name}, would you like to continue the game? y/n")



else:
    print(f"your score is {correct}")
    print("exiting")
    exit

```

```
output:
Welcome to the guessnum
your guess is 1 and program's guess is 5.
Sorry!!! your guess is wrong your guess is 2 and program's guess is 3.
Sorry!!! your guess is wrong your guess is 3 and program's guess is 2.
Sorry!!! your guess is wrong your guess is 4 and program's guess is 3.
Sorry!!! your guess is wrong your guess is 1 and program's guess is 2.
Sorry!!! your guess is wrong your guess is 2 and program's guess is 5.
Sorry!!! your guess is wrong your guess is 3 and program's guess is 4.
Sorry!!! your guess is wrong your guess is 3 and program's guess is 2.
Sorry!!! your guess is wrong your guess is 3 and program's guess is 5.
Sorry!!! your guess is wrong your guess is 4 and program's guess is 2.
Sorry!!! your guess is wrong your guess is 3 and program's guess is 1.
Sorry!!! your guess is wrong your guess is 4 and program's guess is 5.
Sorry!!! your guess is wrong your guess is 2 and program's guess is 2.
you guessed it right!!!
Sorry!!! your guess is wrong your score is 1 exiting
```

As you can see, we utilized while loops, and we used a new variable called correct, which is giving us the score of the user. Which we are printing to the output.

Homework:

Try to print the number of attempts of the user and modify the code to do so.

Congratulations!

Now you know how to put Python applications in practice, and you officially finished the e-book: Python Course from Beginner to Advanced in 11 blog posts.

Head to our blog to explore our second course: JavaScript Course.

Gain a 360' perspective of all your IT projects

We are the creators of Firmbee: an all-in-one project management software which manages your firm's issues, finances, supports remote team work and HR processes.

With our tool IT teams, freelancers, company owners and people from different industries can achieve peak productivity and move project forward in a planned and organized way.

Sign up | Discover more

Follow us on social media for similar content: