Basic Python

Introductions

Comments, Escape Sequences & Print Statement

Comments are used to write something which the programmer does not want to execute. Comments can be written to mark the author's name, date when the program is written, adding notes for your future self, etc.

- Comments are used to make the code more understandable for the programmer.
- The Interpreter does not execute comments.

There are two types of comments in Python Language -:

- Single Line Comment
- Multi-Line Comment

Single Line Comment: Single Line comments are the comments which are written in a single line, i.e., they occupy the space of a single line.

- We use # (hash/pound to write single-line comments).
- E.g., the below program depicts the usage of comments.

```
import os
#This is a comment
print("Main code started")

#Now I will write my code here:
print(os.listdir())
```

Multi-Line Comment: Multi-Line comments are the comments which are created by using multiple lines, i.e., they occupy more than one line in a program.

- We use ' ' '..... Comment ....' ' ' for writing multi-line comments in Python (Use lines enclosed with three quotes for writing multi-line comments). An example of a multi-line comment is shown below:

```
import os
'''This is a comment
Author: Harry
Date: 27 November 2020
Multi-line comment ends here
'''
print("Main code started")

#Now I will write my code here:
print(os.listdir())
```

Python Print() Statement:

print() is a function in Python that allows us to display whatever is written inside it. In case an operation is supplied to print, the value of the expression after the evaluation is printed in the terminal. For example,

```
# print statement for printing strings
print("Harry is a programmer")

# Print statement with a literal
print(1+87)

#This will print "Harry is a programmer" and 88 on the screen respectively!
```

end: end argument allows us to put something at the end of the line after it is printed. In simple words, it allows us to continue the line with " " or ',' or anything we want to put inside these quotes of the end.
It simply joins two different print statements using some string or even by space. Example:

```
# print statement for printing strings
print("Harry is a programmer", end="**")

# Print statement with a literal
print(1+87)

#This will print "Harry is a programmer**88" on the screen
```

Escape Sequences :

- An **Escape Sequence character** in Python is a sequence of characters that represents a single character.
- It doesn't represent itself when used inside string literal or character.
- It is composed of two or more characters starting with backslash \ but acts as a single character. Example \n depicts a new line character.

Some more examples of escape sequence characters are shown below:

Commonly Used Escape Sequences:

Escape Sequences	Description
\n	Inserts a new line in the text at the point
\	Inserts a backslash character in the text at the point
"	Inserts a double quote character in the text at that point
'	Inserts a single quote character in the text at that point
\t	Inserts a tab in the text at that point
\f	Inserts a form feed In the text at that point
\r	Inserts a carriage return in the text at that point
\b	Inserts a backspace in the text at that point

Variables, Datatypes and Typecasting

Variable:

A variable is a name given to any storage area or memory location in a program.

In simple words, we can say that a variable is a container that contains some information, and whenever we need that information, we use the name of that container to access it. Let's create a variable:

```
a = 34 # Variable storing an integer
b = 23.2 # Variable storing real number
```

Here *a* and *b* are variables, and we can use *a* to access 34 and *b* to access 23.2. We can also overwrite the values in *a* and *b*

Data Types in Python:

Primarily there are the following data types in Python.

1. Integers (<class 'int'>): Used to store integers
2. Floating point numbers (<class 'float'>): Used to store decimal or floating-point numbers
3. Strings (<class 'str'>): Used to store strings
4. Booleans (<class 'bool'>): Used to store True/False type values
5. None: None is literal to describe 'Nothing' in Python

Rules for defining a variable in Python:

- A variable name can contain alphabets, digits, and underscores (_). For E.g. : demo_xyz = 'It's a string variable'
- A variable name can only start with an alphabet and underscore.
- It can't start with a digit. For example, *5harry* is illegal and not allowed.
- No white space is allowed to be used inside a variable name.
- Also, reserved keywords are not recommended to be used as variable names.

Examples of few valid variable names are *harry*, *_demo*, *de_mo*, etc.

Python is a fantastic language that automatically identifies the type of data for us. It means we need to put some data in a variable, and Python automatically understands the kind of data a variable is holding. Cool, isn't it?

Have a look at the code below:

```
# Variable in Python:
abc = "It's a string variable"
_abcnum = 40 # It is an example of int variable
abc123 = 55.854 # It is an example of float variable
print(_abcnum + abc123) # This will give sum of 40 + 55.854
```

type() Function in Python: type() function is a function that allows a user to find data type of any variable. It returns the data type of any data contained in the variable passed to it.
Have a look at the code below, which depicts the use of type function:

```
# type() Function in Python:
harry = "40"
demo = 55.5
demo = 40
print (type(harry)) #It will give output as string type
demo3 = type(demo) #It will return data type as float
print(demo3) #It will print that data type
print(type(demo2)) #It will give output as int type
```

Note – We can't do numbers with strings arithmetic operations, i.e., we can't add a string to any number. Have a look at the example below:

```
var1 = "It's a String"
var2 = 5
print(var1+var2) ''' It will give an error as we can't add string to any number. '''
```

Note – We can add (concatenate) two or more strings, and the strings will be concatenated to return another string. Here is the example showing that:

```
var1 = "My Name is "
var2 = "Harry"
var3 = var1+var2+" & I am a Good Boy."
print(var1+var2) # It will give output 'My Name is Harry'
print(var3)
```

Typecasting :

Typecasting is the way to change one data type of any data or variable to another datatype, i.e., it changes the data type of any variable to some other data type.

I know it's a bit confusing but let me tell you in a simple manner. Suppose there is a string "34" Note: String is not integer since it is enclosed in double-quotes) and as we know, we can't add this to an integer number, let's say 6. But to do so, we can typecast this string to int data type, and then we can add 34+6 to get the output as 40. Have a look at the program below:

```
# Typecasting in Python :
abc = 5
abc2 = '45'
abc3 = 55.95
xyz = 5.0

abc4=int(abc2)

print(abc+abc4) # Output : 50
print(abc+int(abc2)) # Output : 50

print(float(abc)+xyz) # It will add 5.0 + 5.0 and will return 10.0

print(str(abc)+45) # It will give an error as abc has been changed into string.
```

There are many functions to convert one data type into another type :

str() – this function allows us to convert some other data type into a string.

int() – this function allows us to convert some other data type into an integer. For example, str("34") returns 34 which is of type integer (int)

float() – this function allows us to convert some other data type into a floating-point number, i.e., a number with decimals.

input() Function – This function allows the user to receive input from the keyboard into the program as a string.

input() function always takes input as a string, i.e., if we ask the user to take a number as input, even then, it will take it as a string, and we will have to typecast it into another data type as per the use case.
If you enter 45 when the input() is called, you will get "45" as a string

```
# Input Function in Python:
print("Enter your name : ")
name = input() #It will take input from user
print(Your Name is",name) # It will show the name
xyz = input(Enter your age : ")
print("Your age is ",xyz)
```

Quick Quiz: Create a program that takes two numbers as input from the user and then prints the sum of these numbers.

Solution :

```
# Quiz :
print("Enter First Number : ")
num1= input()
print("Enter Second Number : ")
num2=input()
print("The Sum is",num2) #It will give output as sum of two numbers.
```

String Slicing And Other Functions In Python

The string is a data type in Python. **Strings in Python programming language** are arrays of bytes representing a sequence of characters. In simple terms, Strings are the combination or collection of characters enclosed in quotes. Strings are one of the most used data types in any programming language because most of the real-world data such as name, address, or any sequence which contains alphanumeric characters are mostly of type 'String'.

Primarily, you will find three types of strings in Python :

- Single Quote String – ('Single Quote String')
- Double Quote String – ("Double Quote String")
- Triple Quote String – (''' Triple Quote String ''')

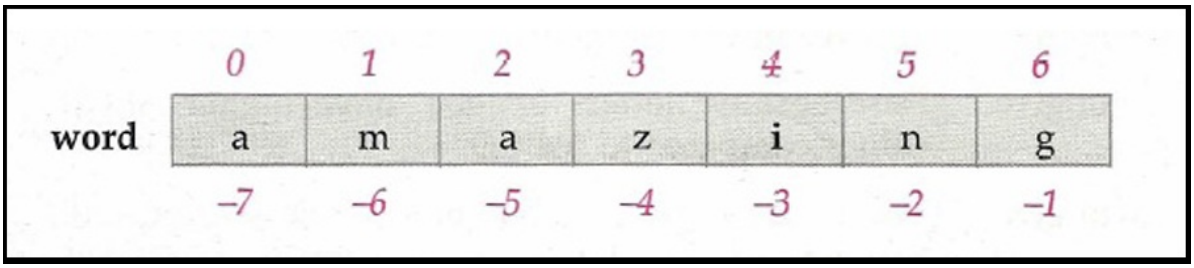
Let us now look into some functions you will use to manipulate or perform operations on strings.

len() Function : This len() function returns the total no. of characters in a string. E.g., for string a="abc", len(a) will return three as the output as it is a string variable containing 3 characters

E.g., Consider this string variable x

```
x = "String Demo"
```

This string variable x contains a string containing 11 characters (including spaces). Since the index in a string starts from 0 to length-1, this string can be looked at as:



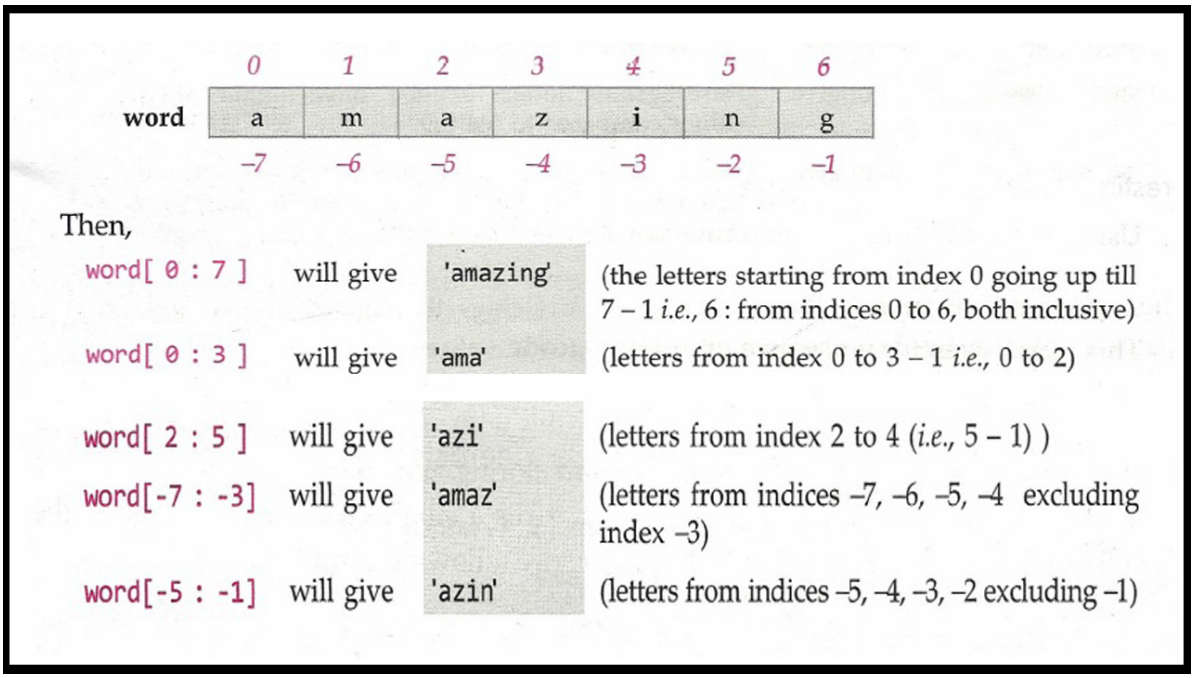
Note: The indexes of a string begin from 0 to (length-1) in the forward direction and -1,-2,-3,..., -length in the backward direction.

String Slicing :

As we know, the meaning of the word 'slice' is 'a part of.' I am sure you have sliced paneer cubes at home!

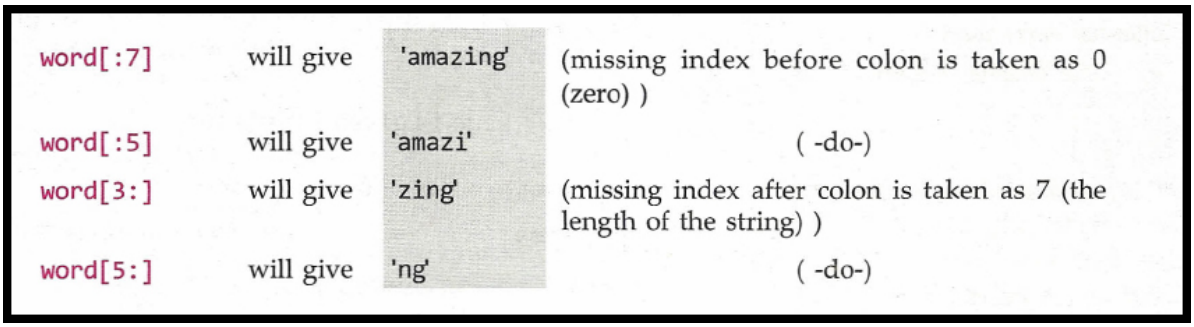
Just like paneer slice refers to the part of the paneer cube; In Python, the term 'string slice' refers to a part of the string, where strings are sliced using a range of indices.

To do string slicing, we just need to put the name of the string followed by [n:m]. It means 'n' denotes the index from which slicing should start, and 'm' denotes the index at which slicing should terminate or complete. Let's look into an example!

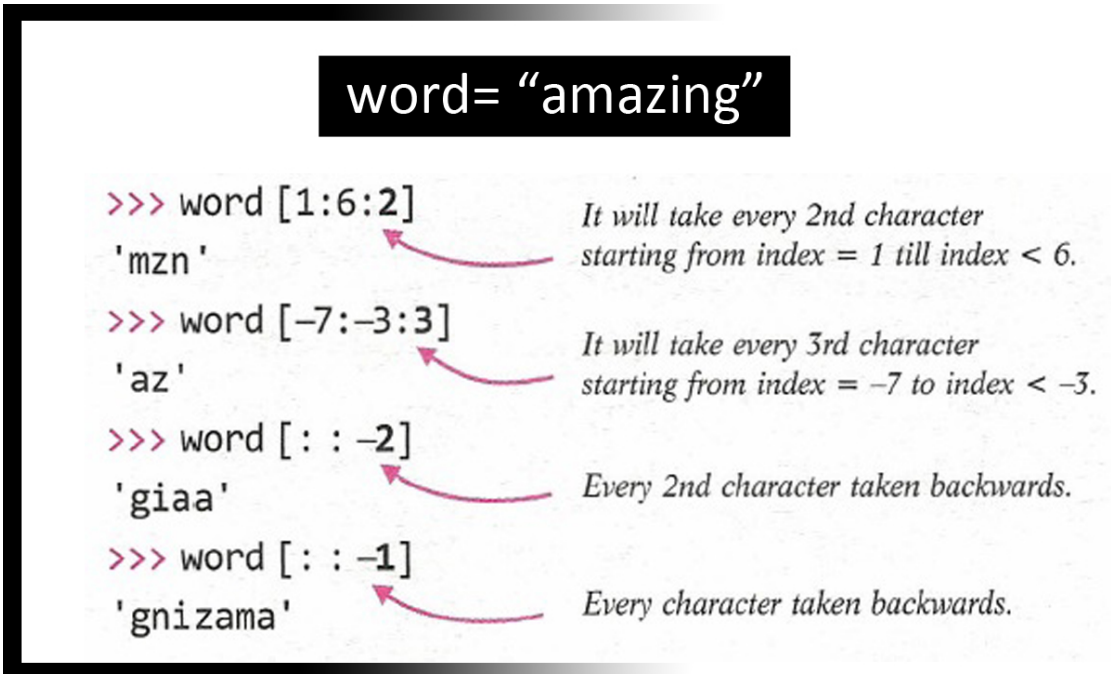


In Python, string slicing s[n:m] for a string s is done as *characters of s from n to m-1*. It means characters are taken from the first index to the second index-1.

E.g., abc="Demo" then abc[0:3] will give 'Dem' and will not give 'Demo' coz index number of 'D' is 0, 'e' is 1, 'm' is 2, and 'o' is 3. So it will give a range from n to m-1, i.e., 0 to 3-1=2. That's why we got the output 'Dem'.



> In string slicing, we sometimes need to give a skip value i.e. string[n Ⓜ skip_value]. This simply takes every skip_valueth character. By default, the skip value is 1 but if we want to choose alternate characters of a string then we can give it as 2. Have a look at the example below:



Let's end this tutorial by looking into some of the most used string functions :

- string.endswith(): This function allows the user to check whether a given string ends with a passed argument or not. It returns True or False.
- string.count(): This function counts the total no. of occurrences of any character in the string. It takes the character whose occurrence you want to find as an argument.
- string.capitalize(): This function capitalizes the first character of any string. It doesn’t take any argument.
- string.upper(): It returns the copy of the string converted to the uppercase.
- string.lower(): It returns the copy of the string converted to lower case.
- string.find(): This function finds any given character or word in the entire string. It returns the index of the first character from that word.
- string.replace(“old_word”, “new_word”): This function replaces the old word or character with a new word or character from the entire string.

```
# String Functions:

demo = "Aakash is a good boy"
print(demo.endswith("boy"))
print(demo.count('o'))
print(demo.capitalize())
print(demo.upper())
print(demo.lower())
print(demo.find("is")
print(demo.find("good","nice"))
```

Data Structure

Python Lists And List Functions

Python lists are containers used to store a list of values of any data type. In simple words, we can say that a list is a collection of elements from any data type. E.g.

```
list1 = ['harry', 'ram', 'Aakash', 'shyam', 5, 4.85]
```

The above list contains strings, an integer, and even an element of type float. A list can contain any kind of data, i.e., it's not mandatory to form a list of only one data type. The list can contain any kind of data in it.

Do you remember we saw indexing in strings? List elements can also be accessed by using Indices, i.e., the first element of the list has 0 index and the second element has one as its index, and so on.

Note: If you put an index that isn't in the list, you will get an error. i.e., if a list named list1 contains four elements, list1[4] will throw an error because the list index starts from 0 and goes up to (index-1) or 3.
Have a look at the examples below:

```
                        Lists in Python

[]                                          # list with no member, empty
list
[1, 2, 3]                                  # list of integers
[1, 2.5, 3.7, 9]                           # list of numbers (integers and
floating point)
['a', 'b', 'c']                             # lisst of characters
['a', 1, 'b', 3.5, 'zero']                 # list of mixed value types
['One', 'Two', 'Three']                 # list of strings
```

List Methods :

Here is the list of list methods in Python. These methods can be used in any python list to produce the desired output.

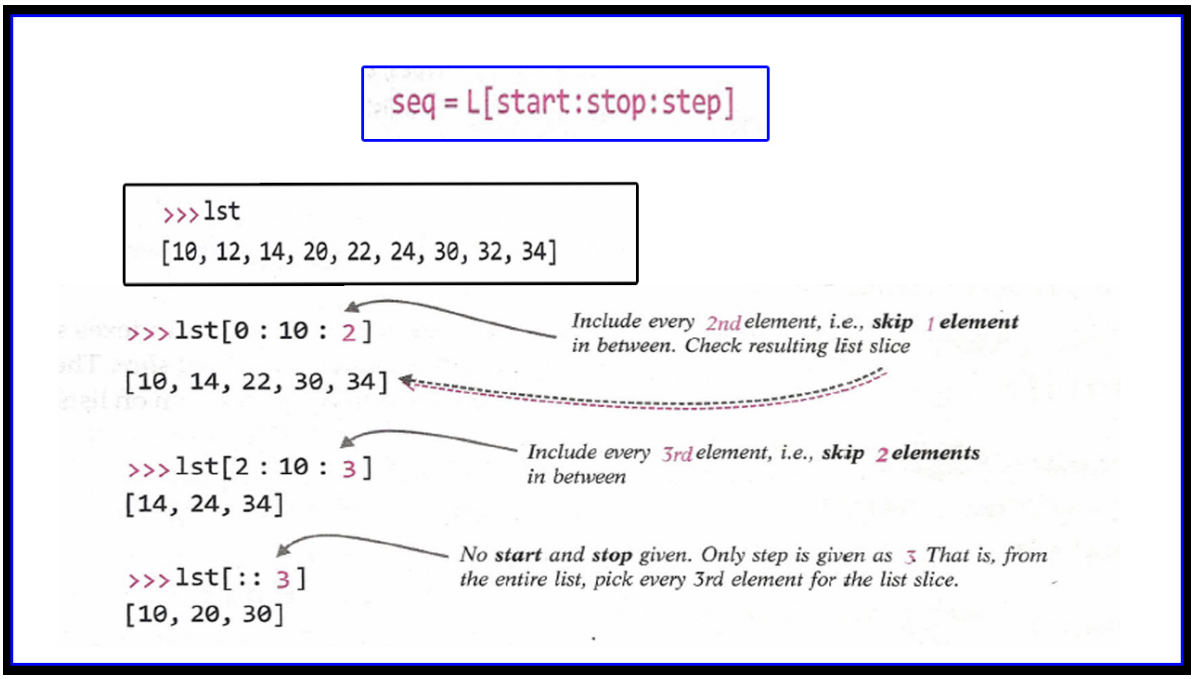
```
# List Methods :
l1=[1,8,4,3,15,20,25,89,65]       #l1 is a list
print(l1)

l1.sort()
print(l1)      #l1 after sorting
l1.reverse()
print(l1)      #l1 after reversing all elements
```

List Slicing :

List slices, like string slices, return a part of a list extracted out of it. Let me explain; you can use indices to get elements and create list slices as per the following format :

```
seq = list1[start_index:stop_index]
```

Just like we saw in strings, slicing will go from a start index to stop_index-1. The seq list, a slice of list1, contains elements from the specified start_index to specified (stop_index – 1).



List Methods:

There are a lot of list methods that make our life easy while using lists in python. Let's have a look at a few of them below:

```
# List Methods :-
list1=[1,2,3,6,,5,4]     #list1 is a list

list1.append(7)     # This will add 7 in the last of list
list1.insert(3,8)     # This will add 8 at 3 index in list
list1.remove(1)     #This will remove 1 from the list
list1.pop(2)          #This will delete and return index 2 value.
```

Tuples in Python:

A tuple is an immutable data type in Python. A tuple in python is a collection of elements enclosed in () (parentheses). Tuple, once defined, can't be changed, i.e., its elements or values can't be altered or manipulated.

```
# Tuples in Python :
a=()     # It's an example of empty tuple
x=(1,)    # Tuple with single value i.e. 1
tup1 = (1,2,3,4,5)
tup1 = ('harry', 5, 'demo', 5.8)
```

Note: To create a tuple of one element, it is necessary to put a comma ',' after that one element like this tup=(1,) because if we have only 1 element inside the parenthesis, the python interpreter will interpret it as a single entity which is why it's important to use a ',' after the element while creating tuples of a single element.

Swapping of two numbers

Python provides a very handy way of swapping two numbers like below:

```
# Swapping of two numbers :
a = 10
b = 15
print(a,b)     #It will give output as: 10 15
a,b = b,a
print(a,b)     #It will give output as: 15 10
```

Dictionary & Its Functions Explained

Before going through the actual content, i.e., the implementation of a dictionary, it is essential to know some fundamental theories so that we can understand what we are going to learn and why we are spending our precious time learning it.

Let's start with the basic definition of a Python Dictionary:

“Python dictionary is an unordered collection of items. Each item of the dictionary has a key and value pair/ key-value pair.”

Now coming to the more formal approach:

Every programming language has its distinct features, commonly known as its key features. With that said, Python’s one out-of-the-box feature is “dictionaries”. Dictionaries may look very similar to a “List”. Still, dictionaries have some distinct features that do not hold true for other data types like lists, and those features make it (python dictionary) special.

Here are a few important features of a python dictionary:

- It is unordered (no sequence is required - data or entries have no order)
- It is mutable (values can be changed even after its formation, or new data/information can be added to the already existing dictionary, we can also pop/remove an entry completely)
- It is indexed (Dictionary contains key-value pairs, and indexing is done with keys. Also, after the Python 3.7th update, the compiler stores the entries in the order they are created)
- No duplication of data (each key is unique; no two keys can have the same name, so there is no chance for a data being overridden)

If we talk a little about how it works, its syntax comprises of key and values separated by colons in curly brackets, where the key is used as a keyword, as we see in real life dictionaries, and the values are like the explanation of the key or what the key holds (the value). And for the successful retrieval of the data, we must know the key so that we can access its value, just like in a regular oxford dictionary where if we don't know the word or its spelling, we cannot obtain its definition. Let's look into the syntax of a Python dictionary:

```
a = {'key', 'value', 'cow':'mooh'}
print(a['cow'])
#will print "mooh" on the screen
```

With the help of dictionaries, we do not have to do most of our work manually through code like in C or C++. I mean that Python provides us with a long list of already defined methods for dictionaries that can help us do our work in a shorter span of time with a minimal amount of code. Some of these methods are, clear(), copy(), popitem(), etc. The best part about them is that

no extra effort is required to be put in order to learn the functionality as their names explain their functions (in most of the cases), such as clear() will clear all the data from the dictionary, making it empty, copy() will make a copy of the dictionary, etc.

Some distinct features that a dictionary provides are:

- We can store heterogeneous data into our dictionary, i.e., numbers, strings, tuples, and the other objects can be stored in the same dictionary.
- Different data types can be used in a single list, making the value of some keys in the dictionary.

Sets In Python

Let's start with the basic definition of sets in Mathematics. People already familiar with the concept of sets in mathematics know that as per the mathematical definition - A set is a collection of well-defined objects and non-repetitive elements that is - a set with 1,2,3,4,3,4,5,2, 2, and 3 as its elements can be written as {1,2,3,4,5}

No repetition of elements is allowed in sets.

In Python programming, sets are more or less the same. Let's look at the Python programming definition of sets:

"A set is a data structure, having unordered, unique, and unindexed elements."

Elements in a set are also called entries, and no two entries could be the same within a set.

If your curiosity isn't satisfied with the basic definition, do not worry, as we are approaching a more formal illustrative approach to an understanding of python sets.

Well, now that you have a basic idea about sets in mathematics. Let me tell you that a mathematical set has some basic operations which can be performed on them. For example, the union of two sets is a set made using all the elements from both sets. The intersection is an operation that creates a set containing common elements from both sets. A python set has all the properties and attributes of the mathematical set. The union, intersection, disjoint, etc., all methods are exactly the same and can be performed on sets in python language.

Note: If you are a programming beginner who doesn't know much about sets in mathematics. You can simply understand that sets in python are data types containing unique elements.

If you want to use sets in your python programs, you should know the following properties of sets in Python:

- Sets are iterable(iterations can be performed using loops)
- They are mutable (can be updated by adding or removing entries)
- There is no duplication (two same entries do not occur)

Structure:

- Elements of the sets are written in between two curly brackets and are separated with a comma, and in this simple way, we can create a set in Python.
- The other way of forming a set is by using a built-in set constructor function.

Both of these approaches are defined in the video above.

Let me now share some basic information about sets so that you can know why they are so important and why you should learn them.

Unlike the dictionary (that we have learned in tutorials 10 and 11), sets are not just restricted to Python language, but nearly all commonly used programming languages have sets included in them as a data type. Examples of these languages include C++, Java, etc., even languages such as Swift and JavaScript support sets. One of the earliest languages that supported sets was Pascal. I hope you now have a rough idea of how important these sets actually are because whichever language you choose to code in, you must have a very basic understanding of sets!

Restrictions:

Everything has a limit to its functionality, and there are some limitations on working with sets too.

- Once a set is created, you can not change any of its items. Although you can add new items or remove previous but updating an already existing item is not possible.
- There is no indexing in sets, so accessing an item in order or through a key is not possible, although we can ask the program if the specific keyword we are looking for is present in the set by using the "in" keyword or by looping through the set by using a for loop(we will cover for loops in tutorial # 16 and 17)

Despite these restrictions, sets play a crucial role in the life of a python programmer. In most cases, these restrictions are never a problem for the programmer, given he knows which data type to use when. And this skill is something you will learn with time after writing a lot of python programs

Set Methods:

There are already a lot of built-in methods that you can use for your ease, and they are easily accessible through the internet. You might want to peep into python's official documentation at times as well to check for some updates they might push down the line. Some of the methods you can use with sets include **union(), discard(), add(), isdisjoint(),** etc., and their functionality is the same as in the sets in mathematics. Also, the purpose of these functions can easily be understood by their names.

Conditional Statements and Loops

If Else & Elif Conditionals In Python

Let's start today's topic of “If else and elif in python”, with a definition:

“If, else and elif statement can be defined as a multiway decision taken by our program due to the certain conditions in our code.”

For few viewers, the term “elif” is new as they are not familiar with the word, and it is also not like most of the other words such as list or loops, etc., that have the same meaning in the English language and in Python programming. In fact, in English, “elif” means honest. But if you have ever done programming in any language, you must be familiar with the “else-if” statement; well, “elif” is just that.

We are now coming towards a more formal sort of description. “If and else” are known as decision-making statements for our program. They are very similar to the decision-making we apply in our everyday life that depends on certain conditions. The everyday example is thoroughly explained in the tutorial, so I will not waste your time on that. Instead, I would now like to focus on more technical details.

Let us focus a little on the working:

Our compiler will execute the if statement to check whether it is true or false now; if it’s true, the compiler will execute the code in the “if” section of the program and skip the bunch of code written in “elif” and “else.” But if the “if” condition is false, then the compiler will move towards the elif section and keep on running the code until it finds a true statement(there could be multiple elif statements). If this does not happen, then it will execute the code written in the “else” part of the program.

An “if” statement is a must because without an if, we cannot apply “else” or “else-if” statements. On the other hand else or else if statement is not necessary because if we have to check between only two conditions, we use only “if and else” and even though if we require code to run only when the statement returns true and do nothing if it returns false then an else statement is not required at all.

Now Let’s talk about some technical issues related to the working of decision statements:

- There is no limit to the number of conditions that we could use in our program. We can apply as many elif statements as we want, but we can only use one “else” and one “if” statement.
- We can use nested if statements, i.e., if statement within an if statement. It is quite helpful in many cases.
- Decision statements can be written using logical conditions, which are:
- Equal to
- Not equal to
- Less than
- Greater than
- Greater than equal to
- Less than equal to

We can also use Boolean or our custom-made conditions too.

Bonus part:

As we know that an “if” statement is necessary, and you can’t have an “else” or “else-if” without it, but let’s suppose you have a large amount of code and for some reason, you have to remove the “if” part of the code (because maybe your code is better without it) but you do not want to do lots of coding again. Then the solution is just to write pass instead of the code, and this will help your code run without any error without executing the if part.

For Loops In Python

Starting with the technical definition of for loops:

Like all the other functions we have seen so far in the video tutorials, for loop is also just a programming function that iterates a statement or a number of statements based on specific boundaries under certain defined conditions, which are the loop's basis.

Note that the statement that the loop iterates must be present inside the body of the loop.

Regarding loops, iteration means going through some chunk of code again and again. In programming, it has the same meaning, the only difference is that the iteration depends upon certain conditions, and upon its fulfillment, the iteration stops, and the compiler moves forward.

For a beginner or layman, the concept of the loop could be easily understood using an example of the song's playlist. When we like a song, we set it on repeat, and then it automatically starts playing again and again. The same concept is used in programming, we set a part of code for looping, and the same part of the code executes until the certain condition that we provided is fulfilled. You must be thinking that in the song playlist, the song keeps on playing until we stop it. The same scenario can be made in the case of loops, and if we put a certain condition that the loop could not fulfill, then it will continue to iterate endlessly until stopped by force.

An example of where a loop could be helpful to us could be in areas where a lot of data has to be printed on the screen, and physically writing that many printing statements could be difficult or, in some cases, impossible. Loops are also helpful in searching data from lists, dictionaries, and tuples.

Why do we use loops?

- Complex problems can be simplified using loops
- Less amount of code required for our program
- Lesser code so lesser chance or error
- Saves a lot of time
- Can write code that is practically impossible to be written
- Programs that require too many iterations such as searching and sorting algorithms can be simplified using loops

How to write a for loop?

For loop basically depends upon the elements it has to iterate instead of the statement being true or false. The latter one is for the While loop, which is the topic for the next tutorial, i.e., tutorial# 17. In different programming languages, the way to write a loop is different; in java and C, it could be a little technical and difficult to grasp for a beginner, but it's simple and easy in Python. We just have to declare a variable so we can print the output through it during different iterations and use the keywords “for” and “in”. More explanation could be easily obtained about working and syntax through the video tutorial.

Advantages of loops:

- The reusability of code is ensured

- We do not have to repeat the code, again and again; we just have to write it one time
- We can transverse through data structures like list, dictionary, and tuple
- We apply most of the finding algorithms through loops

Example of a for loop:

```
dict1= {"Best Python Course": "CodeWithHarry",
        "Best C Languge Course": "CodeWithHarry",
        "Harry Sir":"Tom Cruise Of Programming"
        }

for x,y in dict1.items():
    print(x, y)
```

Output:

```
Best Python Course: CodeWithHarry
Best C Languge Course: CodeWithHarry
Harry Sir: Tom Cruise Of Programming
```

You can see that how I've printed the key-value pairs of the dictionary **dict1** using a for loop.

While Loops In Python

In the previous tutorial, we discussed about loops in general, what they are and their benefits, why we should use them, along with syntax, and the working of the “for” loop. If you haven’t gone through that tutorial till now, I will recommend you to go through that first, so it will be easier for you to learn the concept of while loop, once you have the concept, what loops generally are.

Now, as we know now what loops are and why they are used, in this section, I will go directly towards the working and syntax of while loop along with its comparison with for loop and where to use it.

“A while loop in python runs a bunch of code or statements again and again until the given condition is true when the condition becomes false, the loop terminates its repetition.”

The syntax for a while loop is simple and very much like for loop. We have to use the keyword “while”, along with it, we have to put a condition in parenthesis, and after that, a colon is placed. The condition could be either true or false. Until the condition is true, the loop will keep on executing again and again. If we use a certain condition in our while loop that never becomes false, the program will continue running endlessly until we stop it by force. So, this kind of mistake in our syntax is known as logical/human error.

To terminate an infinite loop, you can press **Ctrl+C** on your system.

Syntax of while loop:

```
while codition_is_true:
    Code inside the loop body
```

For vs. While:

A “for” statement loop runs until the iteration through, set, lists, tuple, etc., or a generator function is completed. In the case of a while loop, the statement simply loops until the condition we have provided becomes false. We generally use for loops for areas where we are already familiar with the number of iterations and use while loop where the number of iterations are unknown. Because while the loop is solely based on the state of its condition.

Let us understand the concept of the while loop and why we use it in areas where the number of iterations are not defined, or we do not have any idea how many iterations would take place with the help of an example. Suppose that we have created an application for an ATM from where a customer can only withdraw money up to 5000 rupees. Now our condition in the while loop would be to iterate unless the input is between 1 to 5000. So, the condition will be true unless the user inputs a number between 1 to 5000, so the loop will iterate depending upon the time until the user submits the wrong input. The example is a bit rough, but I hope it helps you clear your concepts.

For a While loop to run endlessly, we can pass true or 1 as a condition. 1 is also used in place of writing true as a whole. So, in this case, the condition will never become false, so the program will run endlessly. But these kinds of programs do not output anything because they can never complete their execution.

Break & Continue Statements In Python

“Break and continue statements are used to alter the flow or normal working of a loop, that could be either a “for loop” or “while loop”.

If you are not familiar with the concept of loops, I recommend you go through Tutorial numbers 16 and 17 of Python Tutorials For Absolute Beginners In Hindi Playlist.

You all know what break and continue mean in the basic English language. Break means interrupt and continue means resuming after an interrupt as the meanings of the words can describe that both these functions are totally opposite to each other. Hence, both of these keywords are mostly defined together, like “if and else” or “try and except,” but unlike the others, their functionality is quite the opposite of each other. They aren’t even used together in most of the cases like “except”, could only be used if there is “try” or “else” condition and if there isn’t “if” statement present, but in cases of “break and continue", they both do not have any such relation. They may be defined together but not mostly used together.

Defining break statement, break statement alters the normal functionality of the loops by terminating or exiting the loop containing it. The compiler then moves on to the code that is placed after the body of the loop. The syntax of break is only a single word, i.e., “break”. A break statement is relatively easier to understand than a continue statement as it just leaves the bunch of code written after it inside the loop. The control of the program is then shifted to the statement written after the loop.

Example Of Break Statement:

```
i=0;
while(True):
    print(f"The value of i is : {i}")
    i=i+1
    if(i>10):
        print("Breaking the loop. ")
        break;
```

Output:

```
The value of i is : 0
The value of i is : 1
The value of i is : 2
The value of i is : 3
The value of i is : 4
The value of i is : 5
The value of i is : 6
The value of i is : 7
The value of i is : 8
The value of i is : 9
The value of i is : 10
Breaking the loop.
```

Continue statement also alters the flow of a normal working loop, but unlike the break statement, it takes the compiler to the start of the code that has already been executed before the statement but is inside the loop boundary. All the code written after the continue statement is skipped, but it is worth noting that the continue statement works only for a single iteration. Unless in situations where it's written with decision statements such as if, else, etc., in those situations, the continue statement will totally be dependent upon the condition of the decision statement. Its syntax is also plain and easy, like a break statement as you only have to use the keyword “continue”.

Example Of Continue Statement:

```
i=0;
while(True):
    i=i+1
    if(i==5):
        continue
    if(i>10):
        break
    print(f"The value of i is : {i}")
```

Output:

```
The value of i is : 1
The value of i is : 2
The value of i is : 3
The value of i is : 4
The value of i is : 6
The value of i is : 7
The value of i is : 8
The value of i is : 9
The value of i is : 10
```

Where and when can these statements come in handy:

When you are working with a big project, there might occur a situation where you only need to use the loop partially without adding new or removing already existing lines of code. You can easily apply the break statement in your code so that you can partially run it without any sort of error.

Or in another situation, let's suppose you are working with while loop printing some lines on the screen and you want to skip an iteration in between others, then you can write the continue statement using an “if” statement that matches your need so that you can skip the specific iteration.

Operators In Python

Today's tutorial is more about theoretical learning than coding because we must have basic knowledge about what we are actually implementing in our code because if our basis is strong then only we can make an efficient program of a larger scale.

“Operators in Python can be defined as symbols that assist us to perform certain operations. The operations can be between variable and variable, variable and value, value and value”

Operators that Python Language supports are:

- Arithmetic Operators
- Assignment Operators
- Comparison Operators
- Logical Operators
- Identity Operators
- Membership Operators
- Bitwise Operators

We must remember most of them from our basic mathematics that we studied in school. May be except the last one, that might be new for some of us. To understand bitwise fully, we must have the basic knowledge of the conversion of decimal into binary. For example, the binary for the first five number is:

- 0001
- 0010
- 0011
- 0100
- 0101

And so on.

Now I will give you a theoretical understanding of each of these operators. There is no theory related difference in them and the one we studied during school time. The only difference you will see will be in the syntax i.e. how to write them for perfect execution.

Arithmetic Operators:

Basic mathematical operations such as addition, multiplication, subtraction, division, etc. are performed with the help of arithmetic Operations. It contains nearly all operations that we can perform with the help of a calculator. Symbols for such operators include *, /, %, -, //, etc.

Assignment Operators:

The assignment operator is used to assign values to a variable. In some cases, we have to assign a variable’s value to another variable, in such cases the value of the right operand is assigned to the left operand. One of the basic signs from which we can recognize an assignment operator is that it must have an equal-to(=) sign. Some commonly used assignment operators include +=, -=, /=, etc.

Comparison Operators:

They are also known as relational operators. They compare the values on either side of the operator and decide the relation among them. Commonly used comparison operators include ==, >, < , >=,etc.

Logical Operators:

Logical operators perform logical AND, OR and NOT, operations. They are usually used in conditional statements to join multiple conditions. AND, OR and NOT keywords are used to perform logical operations.

Identity Operations:

Identity operator checks if two operands share the same identity or not, which means that they share the same location in memory or different. “is” and “is not” are the keywords used for identity operands.

Membership Operands:

Membership operand checks if the value or variable is a part of a sequence or not. The sequence could be string, list, tuple, etc. “in” and “not in” are keywords used for membership operands.

Bitwise Operand:

Bitwise operands are used to perform bit by bit operation on binary numbers. First, we have to change the format of the number from decimal to binary and then compare them using AND, OR, XOR, NOT, etc.

```
x %=7 # x = x%7
print(x)

# Bitwise Operators
# 0 - 00
# 1 - 01
# 2 - 10
# 3 - 11
print(0 & 2)
print(0 | 3)
```

Short Hand If Else Notation In Python

Before going through the if-else short-hand statement, we must be familiar with the basic definition, so that we can know what the short-hand statement actually is. In basic English, shorthand can be said as “a basic way of writing using abbreviations or symbols”. For example, we do not use the full name, Kentucky Fried Chicken, instead we use its abbreviation that is KFC. Writing in such a compact manner is known as shorthand. Now let us move towards the definition of shorthand with respect to the Python language.

“An executable statement, written in so compact manner that it comprises of only a single line of code is known as shorthand statement.”

Python supports many sorts of shorthand statements and with the help of these statements, we can write our code in a more compact sort or form using less space. We learned the original form of “if and else” statement in tutorial #13 of our playlist, we can also write it in a shorthand form of code.

If we write the statement:

```
If-expression if(Condition) else else-expression
```

It will be same as:

```
if condition:
   if-expression
else:
   else-expression
```

Now let us come to the question that why we should use shorthand if-else statement instead of the original one. First of all, you have to do less typing while using the shorthand form. If you are making a project that requires thousands of lines of code then, it is very hectic if you are writing something in four or five lines that could easily be written in a single line. This advantage would not be too convincing for a lot of viewers now and they would not like to adopt the shorthand method instead of the one they are using currently. In this case, this tutorial is still very important for them because if you are a coder then reading other people’s code is a must. Even though you are clearing your concepts by going through some code on Stack Overflow or making a group project with other programmers, you will surely encounter shorthand code at any time, so it is best to have a piece of prior knowledge about it.

The second advantage of using the shorthand way of programming is that the compiler handles it in an efficient way. It does not matter for the short program even though if they are a couple of thousand lines long but when working on a huge project, like Amazon (12 Million lines of code) or Facebook (62 Million lines of code) or Google (2 Billion lines of code), then it is very important to save as many lines of code as we can by writing it in a compact form.

There is also a disadvantage of using shorthand, that it is not easily readable. When we are working on a group project, we always try to choose the approach that is the easiest to understand, and writing a compact code always bring difficulties when we share our code with someone. Even though whichever approach we follow, we should always have knowledge about all possible approaches, because everyone is not using the same approach.

Code file as described in the video

```
a = int(input("enter a\n"))
b = int(input("enter b\n"))

# if a>b: print("A B se bada hai bhai")
print("B A se bada hai bhai") if a
```

Functions And Docstrings

Function

"Functions in Python can be defined as lines of codes that are built to create a specific task and can be used again and again in a program when called."

There are two types of functions in the Python language:

- Built-in functions
- User-defined functions

We have used a lot of built-in functions in our code till now, these functions include print(), or sum(), etc. So, we have a good idea about how to call a function. Built-in functions are already present in our python program, and we just have to call them whenever we need them to execute. Being familiar with built-in functions we will now look into User-defined function mostly in this tutorial.

We must know how to define a function in Python. In order to create one ourselves, we have to use the def keyword in order to define a function accompanied by the function's name with a pair of parentheses. The function of parenthesis is to send arguments or parameters to a function. In simple words, parameters can be defined as values that are sent in the parenthesis. For example, if a function is used to add two numbers then both the numbers will be passed as parameters in the parenthesis. After parenthesis, a colon is used to get in the body of the function. Some functions may return a value to the caller, so in order to get the value, a return statement is put at the end of the body of the function that gives the value that has been calculated by the function.

Calling a function is very simple, we just have to write the name of the function along with the closing parenthesis. If the function requires some arguments then we write those in the parenthesis, but if it does not return anything, then we leave them empty.

We have discussed a lot about what functions are and how to use them, now let us move to the advantages of using a function:

- If we are working on a big project then we will prefer to make as many functions as possible, so every other member of our team could use that.
- By using functions, we can avoid the repetition of code to an extent. As we have discussed in the previous tutorial i.e. Tutorial #22, more lines of code mean less efficiency. Also repeating the same code at different places will just make the code more crowded than required.
- The reusability of code is ensured by using functions. We can even use a function inside another function or in any part of our code.
- By making a function of code that we are going to use again and again, we can save a lot of time.

Docstrings

Docstring is a short form of documentation string. Its purpose is to give the programmer a brief knowledge about the functionality of the function. It must be the first string in a function, and it is also an optional string but always good to have it while working on programs having multiple functions. The syntax for writing a docstring is very simple as it is just a string written in between three double quotes placed three times (""" """) on either side of the string. But it has to be the first line of code in the function's body. To call a docstring we write the name of the function followed by *.doc*.

Code file as described in the video

```python
# a = 9
# b = 8
# c = sum((a, b)) # built in function

def function1(a, b):
    print("Hello you are in function 1", a+b)

def function2(a, b):
    """This is a function which will calculate average of two numbers
    this function doesnt work for three numbers"""
    average = (a+b)/2
    # print(average)
    return average

# v = function2(5, 7)
# print(v)
print(function2.__doc__)
```

Advanced Python

Try Except Exception Handling In Python

Before discussing exceptional handling, let us discuss, what an exception is actually.

“Exception can be said as an error, that causes a program to crash. Unlike syntax error, it is syntactically correct and occurs mostly due to our negligence”

For example, assigning a string value to an int data type variable or dividing a number by zero or also when a name of a variable or a function is not found, an exception occurs. Python has built-in support for dealing with many sorts of exceptions automatically, but we can also define our own.

The solution to exception related problems is simple. We just have to alter the normal flow of the program by a bit of code known as an exception handler. This code will save the state of the program up to the point where the exception occurred and will continue the normal flow from the code written outside the area of occurrence of an exception. We can also print the exception by converting it into a string. This way program does not terminate but executes completely except for the area where the exception occurred.

Now as we have covered the basics, let us move towards an in-depth understanding of exception handling. Try and except blocks are used in Python to handle the exception. If you are familiar with any other programming language, the except block is the same as the catch block. In the try block, we write the code about which we have doubt that exception could occur in it, and in except block we just write the code that we want to execute in case the exception error occurs. In such cases where no exception occurs, the except block will not execute. In simple words, in the try block, we write the code where chances of exception are high, and in except block, we handle the error, maybe through printing a warning or just skipping the exception part completely by completely ignoring it as a part of the program.

We can also use an else keyword to print something if no exception occurs. For example. In case of an exception, the except block is printing the error, likewise, if the exception does not occur, we could print a statement that no error occurred, using an else keyword.

There are also many sorts of predefines exceptions that we could find in Python such as EOF or End of File Error (occurs when the end of a file is reached but the operation is not completed) or ZeroDivisionError (occurs when the number is divided by zero). We can code such expect blocks that catch only specific sort of exception and ignore the rest. For this purpose, we have to specify the error name after the keyword except, before putting the colon.

Advantages of using try and catch

- Without a try block if an exception occurs the program will surely crash.
- If we have some part of code that is not much important but can cause an exception, then we must write it down in the try block so it does not cause the whole program to crash.

Code file as described in the video

```
print("Enter num 1")
num1 = input()
print("Enter num 2")
num2 = input()
try:
    print("The sum of these two numbers is",
          int(num1)+int(num2))
except Exception as e:
    print(e)



print("This line is very important")
```

File Handling

Python File IO Basics

You must have noticed that till now we have been learning one new concept per tutorial. For some important concepts like loops we had to allocate two tutorials so we can grasp the concept of both loops (for and while) separately. But now in the case of the file, we have allocated the next five tutorials (excluding the exercise and their solutions). So, from this, you can take a hint that how important file handling is in programming.

In this tutorial we are not getting into files in detail, instead, we are discussing the basics of the file and its modes in a theoretical manner. In computer terms, “a file is a resource for saving data and information in computer hardware”. A file is stored in the form of bytes in hardware. A file is opened in the RAM, but it is stored in the hardware because the hardware is non-volatile i.e. it stores its data permanently. On the other hand, RAM is volatile, it loses its data when the system is shut down.

Unlike C or C++, file handling in python is relatively easy and simple. Python treats files differently as text or binary and this is important. There are two types of files that we normally encounter in our computer daily. The first one is a text file and the second one is a binary file. We can understand by the name of the text file that it must contain text in it. The extension for the text file is .txt. All other forms of files are mostly binary even a .doc file, that we open in Microsoft Word is a binary file because it requires special software for accessing it.

The second sort of files are binary files. They are almost all the other files that we come in contact with while using our computer. These files include images, PDFs, Excel files, etc.

Modes of opening file in Python:

There are many modes of opening a file in Python, unlike other languages Python has provided its users a variety of options. We will discuss seven of them in this tutorial.

- r : r mode opens a file for read-only. We do not have permission to update or change any data in this mode.
- w : w mode does not concern itself with what is present in the file. It just opens a file for writing and if there is already some data present in the file, it overwrites it.
- x : x is used to create a new file. It does not work for an already existing file, as in such cases the operation fails.
- a : a stands for append, which means to add something to the end of the file. It does exactly the same. It just adds the data we like in write(w) mode but instead of overwriting it just adds it to the end of the file. It also does not have the permission of reading the file.
- t : t mode is used to open our file in text mode and only proper text files can be opened by it. It deals with the file data as a string.
- b : b stands for binary and this mode can only open the binary files, that are read in bytes. The binary files include images, documents, or all other files that require specific software to be read.
- + : In plus mode, we can read and write a file simultaneously. The mode is mostly used in cases where we want to update our file.

Open(), Read() & Readline() For Reading File

As we now have an idea of what files(text or binary) are and their access modes, we are now ready to dive into the discussion of file handling methods. When we want to read or write a file (say on our hard drive), we must first open the file. When we open a file, we are asking the operating system to find the file by name, making sure the file exists.

How to open a file**?**

Python has a built-in open() function to open a file.

The syntax of the function is:

```
1 open("filename" ,"mode")
```

To open a file, we must specify two things,

- Name of the file and its extension
- Access mode where we can specify in which mode file has to be opened, it could either be read (r), write (w) or append(a), etc. For more information regarding access modes, refer to the previous tutorial.

For Example,

```
1 open("myfile.txt")
```

The file “myfile.txt” will open in "rt" mode as it is the default mode. But the best practice is to follow the syntax to avoid errors.

The open function returns a file object. We store this file object into a variable which is generally called as a file pointer/file handler. Here is a code snippet to open the file using file handing in Python,

```
1   f=open("myfile.txt," "w")
```

You can use this file pointer to further add modifications in the file. An error could also be raised if the operation fails while opening the file. It could be due to various reasons like trying to access a file that is already closed or trying to read a file open in write mode.

How to read a file?

To read a file in Python, there are various methods available,

- We can read a whole file line by line using a **for loop in combination with an iterator.** This will be a fast and efficient way of reading data.
- When opening a file for reading, Python needs to know exactly how the file should be opened. Two access modes are available reading (r), and reading in binary mode (rb). They have to be specified during opening a file with the built-in open() method.

```
1 f = open("myfile.txt", "r")
```

The read() method reads the whole file by default. We can also use the read(size) method where you can specify how many characters we want to return i.e.

```
f.read(2); #Here, you will get the first two characters of the file.
```

- You can use the readline() method to read individual lines of a file. By calling readline() a second time, you will get the next line.
- readlines() method reads until the end the file ends and returns a list of lines of the entire file. It does not read more than one line.

```
f=open("myfile.txt","r");
f.readlines() #Returns a list object
```

Note: The default mode to read data is text mode. If you want to read data in binary format, use "rb".

Is it necessary to close a file**?**

The answer is yes, it is always the best practice to close a file after you are done performing operations on it. However, Python runs a garbage collector to clean up the unused objects, but as good programmers, we must not rely on it to close the file. Python has a build-in close() function to close a file i.e;

```
f.close()
```

I hope you like this tutorial. Here, we have discussed different file handling in Python with examples that will help you while working on real-world projects.

Source Code:

```
f = open("harry.txt", "rt")
print(f.readlines())

content = f.read()
for line in f:
    print(line, end="")
print(content)
content = f.read(34455)
print("1", content)

content = f.read(34455)
print("2", content)
f.close()
```

Writing And Appending To A File

We have already discussed how to open and read a file in Python, in the last tutorial. If you haven't seen the video, please go and see that one first. Now we will discuss how to write or insert text to a file and also how we can simultaneously read and write a file. Now, as we know, there are different modes of opening a file, we will cover three of them in this tutorial.

Note: f is an object for the file. It is not a method or a special character. You will notice me using f.write() or f.read() or f.close(), in further description or tutorial, but you can use character or word of your own choice instead.

MODES	PURPOSE
“w” mode:	Here “w” stands for write. After opening or creating a file, a function, f.write() is used to insert text into the file. The text is written inside closed parenthesis surrounded by double quotations. There is a certain limitation to the write mode of the opening file that it overrides the existing data into the file. For a newly created file, it does no harm, but in case of already existing files, the previous data is lost as f.write() overrides it.
“a” mode:	“a” symbolizes append mode here. In English, appends mean adding something at the end of an already written document, and the same is the function the mode performs here. Unlike write mode, when we use "a" keyword, it adds more content at the end of the existing content. The same function i.e., f.write() is used to add text to the file in append mode. It is worth noting that append mode will also create a new file if the file with the same name does not exist and can also be used to write in an empty file.
“r+” mode:	At the beginning of the description, I told you that we would learn reading and writing a file simultaneously. Well, r+ mode is more of a combination of reading and append than read and write. By opening a file in this mode, we can print the existing content on to the screen by printing f.read() function and adding or appending text to it using f.write() function.

A very helpful tip for beginners:

If you are writing in append mode, start your text by putting a **blank space** or **newline character (\n)** else the compiler will start the line from the last word or full stop without any blank space because the curser in case of append mode is placed right after the last character. So, it is always considered a good practice to adopt certain habits that could help you in the future, even though they are not much helpful now.

f.close():

f.close() is used to close a file when we are done with it. It is a good practice to close a file after use because whichever mode you opened it for, the file will be locked in for that specific purpose and could not be accessed outside the program, even though the file browser.

Code file as described in the video

```
# f = open("harry.txt", "w")
# a = f.write("Harry bhai bahut achhe hain\n")
# print(a)
# f.close()

# f = open("harry2.txt", "a")
# a = f.write("Harry bhai bahut achhe hain\n")
# print(a)
# f.close()


# Handle read and write both
f = open("harry2.txt", "r+")
print(f.read())
f.write("thank you")

```

Seek(), tell() & More On Python Files

Python file objects give us many methods and attribute that we can use to analyze a file, including tools to figure out the name of the file associated with the file object, whether it is closed or opened, writable, readable and how it handles errors.

We have already discussed the file, its access modes, how to open, close, read, and write files in the previous tutorials.

Now it is time to pay attention to some of the most useful and important functions used in file handling.

In today's tutorial, we are going to study why there is a need for tell() and seek() functions and how we can use them?

tell()

When we are working with python files, there are several ways to present the output of a program, it could be in human-readable form or binary form, or we use read() function with specified sie to read some data.

What if we want to know the position of the file(read/write) pointer.

For this purpose, we use **the tell() function**. f.tell() returns an integer giving the file pointer current position in the file represented as a number of bytes. File Pointer/File Handler is like a cursor, which defines from where the data has to be read or written in the file. Sometimes it becomes important for us to know the position of the File Pointer. With the help of tell(), this task can be performed easily

Description:

- **Syntax:** seek()
- **Parameters Required:** No parameters are required.
- **Return Value:** seek() function returns the current position of the file pointer within the file.

Example:

```
 f = open("myfile.txt", "r")
print(f.readline() )
print(f.tell())
```

Here the question arises, **what if we want to change the position of the file pointer**.

Here the concept of seek() function comes.

seek()

When we open a file, the system points to the beginning of the file. Any read or write will happen from the start. To change the file object's position, use seek(offset, whence) function. The position will compute by adding *offset* to a reference point, and the whence argument selects the reference point. It is useful when operating over an open file. If we want to read the file but skip the first 5 bytes, open the file, use function seek(5) to move to where you want to start reading, and then continue reading the file.

Description:

- **Syntax:** file_pointer .seek(offset, whence).
- **Offset:** In seek() function, offset is required. Offset is the position of the read/write pointer within the file.
- **Whence:** This is optional. It defines the point of reference. The default is 0, which means absolute file positioning.

Value	Meaning
0	Absolute file positioning. The position is relative to the start of the file. The first argument cannot be negative.
1	Seek relative to the current position. The first argument can be negative to move backward or positive to move forward
2	Seek relative to the file's end. The first argument must be negative.

Example:

This code will change the current file position to 5, and print the rest of the line.

```
f = open("myfile.txt", "r")
f.seek(5)
print( f.readline() )
```

Note: not all file objects are seekable.

Code file as described in the video

```
f = open("harry.txt")
f.seek(11)
print(f.tell())
print(f.readline())
# print(f.tell())

print(f.readline())
# print(f.tell())
f.close()

```

Using With Block To Open Python Files

There is more than one way to open and close a file. The one we have studied till now is using the **open()** and **close()** function. In today's tutorial, we will go through another yet better and straightforward approach of opening and closing files. We will see how we can use, with block to open and close a file, including syntax and benefits. **We will be using f as our file's object**.

Opening and closing of files are necessary and crucial steps in file handling. We cannot read, write, or perform any task on a file without opening it first. Everyone is familiar with it because it is the first step in file handling. But what most of us are not familiar with is how vital closing a file is. If we do not close our file after we are done using it, then the file object will keep on consuming processor memory, and also, there will be more chances of exceptions as the file is still open hence, more chances of bugs.

To save ourselves from such situations, we could use a with block to open files.

Now how the with block works?

Its syntax is simple:

```
With open(“file_name.txt”) as f:
```

f being the object of the file. The important thing to note is, there is no **close() function** required. After running the code in the block, the file will automatically be closed.

Now at this level, closing a file does not seem like a big issue, but when we are working on more significant projects with a higher number of files then there are instances where we tend to forget that we have closed our file. In such situations, chances of bugs occurrence increase, and we can not access a file elsewhere until it is closed properly. Also, the program will require more processing power. So, even if we are not dealing with a more significant project now, still closing a file is a good practice because, as a programmer, I can tell you that the practices you adopt now will soon become habits, and they will be difficult to let go.

What opening a file with **"With block"** actually does is to create a context manager that automatically closes a file after processing it. Another benefit of using a “With block” is that we can open multiple files in a single block by separating them using a comma. All the files could have different modes of opening. For example, we can access one file for reading and another one for writing purposes. Both files should have different objects referring to them.

The syntax would be:

```
With open(“file1txt”) as f, open(“file2.txt”) as g
```

Both files will be simultaneously closed together.

Let us once again briefly go over the advantages of With block:

- Multiple files can be opened.
- The files that are opened together can have different modes
- Automatically closes file
- Saves processing power by opening and closing file only when running code inside the block
- Creates a context manager, so lesser chances of an exception occurring

Code file as described in the video

```
with open("harry.txt") as f:
    a = f.readlines()
    print(a)

# f = open("harry.txt", "rt")
#Question of the day - Yes or No and why?
# f.close()

```

Scope, Global Variables and Global Keyword

What is Scope in Python?

Scope refers to the coding area where a particular Python variable is accessible. Hence one cannot access any particular variable from anywhere from the code. We have studied about variables in previous lectures. Recall that a variable is a label for a location in memory. It holds a value. Not all variables are accessible, and not all variables exist for the same amount of time.

Where the variables defined determines that is it accessible or not, and how long it will exist.

Local vs. Global Variables

Local Variable:-

A variable that is declared inside a function or loop is called a *local variable*. In the case of functions, when we define a variable within a function, its scope lies within the function only. It is accessible from the point where it is defined until the end of the function. It will exist for as long as the function is executing. Local variables cannot be accessed outside the function. The parameter names in the function, they behave like a local variable.

For Example,

```
def sum():

      a=10 #local variable cannot be accessed outside the function
      b=20
      sum=a+b
      print( sum)

print(a) #this gives an error
```

When you try to access variable “a” outside the function, it will give an error. It is accessible within the function only.

Global Variable:-

On the other hand, a global variable is easier to understand; it is not declared inside the function and can be accessed anywhere within a program. It can also be defined as a variable defined in the main body of the program. Any function or loop can access it. Its scope is anywhere within the program.

For Example,

```
a=1  #global variable

def print_Number():

            a=a+1;
            print(a)

 print_number()
```

This is because we can only access the global variable, but we cannot modify it from inside of the function.

What if we want to modify the global variable inside the function?

For this purpose, we use the **global keyword.** In Python, the global keyword allows us to modify the global variable. It is used to create a global variable and make changes to the variable in a local scope.

Rules of global keyword:

- If we assigned a value to a variable within the function body, it would be local unless explicitly declared as global.
- Those variables that are referenced only inside a function are implicitly global.
- There is no need to use the global keyword outside a function.

What if we have a nested function. How does the scope change?

When we define a function inside another function, it becomes a nested function. We already know how to access a global variable from a function by using a global keyword. When we declare a local variable in a function, its scope is usually restricted to that function alone. This is because each function and subfunction stores its variables in its separate workspace.

A nested function also has its own workspace. But it can be accessed to the workspaces of all functions in which it is nested. A variable whose value is assigned by the primary function can be read or overwritten by a function nested at any level within the primary.

Code file as described in the video

```
l = 10 # Global

def function1(n):
    l = 5 #Local
    m = 8 #Local
    global l
    l = l + 45
    print(l, m)
    print(n, "I have printed")

function1("This is me")
print(m)

x = 89
def harry():
    x = 20
    def rohan():
        global x
        x = 88
    print("before calling rohan()", x)
    rohan()
    print("after calling rohan()", x)

harry()
print(x)
```

Recursions: Recursive Vs Iterative Approach

We have worked with iteration in previous lectures, related to loops, while recursion is a new topic for us. Let's start with the definition:

"Recursion occurs when a function calls itself."

Mostly both recursion and iteration are used in association with loops, but both are very different from each other. In both recursion and iteration, the goal is to execute a statement again and again until a specific condition is fulfilled. An iterative loop ends when it has reached the end of its sequence; for example, if we are moving through a list, then the loop will stop executing when it reached the end of the list. But in the case of recursion, the function stops terminating when a base condition is satisfied. Let us understand both of them in detail.

Recursion:

There are two essential and significant parts of a recursive function. The first one is the **base case**, and the second one is the **recursive case**. In the base case, a conditional statement is written, which the program executes at the end, just before returning values to the users. In the recursive case, the formula or logic the function is based upon is written. A recursive function terminates to get closer to its base case or base condition. As in case of loops, if the condition does not satisfy the loop could run endlessly, the same is in recursion that if the base case is not met in the call, the function will repeatedly run, causing the system to crash.

In case of recursion, each recursive call is stored into a stack until it reaches the base condition, then the stack will one by one return the calls printing a series or sequence of numbers onto the screen. It is worth noting that stack is a LIFO data structure i.e., last in first out. This means that the call that is sent into the stack at the end will be executed first, and the first one that was inserted into the stack will be executed at last.

Iteration:

We have a basic idea about iteration as we have already discussed it in tutorial # 16 and 17 relating loops. Iteration runs a block of code again and again, depending on a user-defined condition. Many of the functions that recursion performs can also be achieved by using iterations but not all, and vice versa.

Recursion vs. Iteration:

- Recursion can only be applied to a function, while iteration can be used for any number of lines of code, we want to repeat
- Lesser coding has to be done in case of recursion than iteration
- Back tracing or reverse engineering in case of recursion can be difficult.
- In the case of recursion, if the condition is not met, the system will repeat a few times and then crash while in case of iteration it will continue to run endlessly.
- Even though less coding has to be written in case of recursion, it is still slower in execution because the function has to be called again, and again, storing data into the stack also increases the time of execution.

In my opinion for smaller programs where there are lesser lines of codes, we should use a recursive approach and in complex programs, we should go with iteration to reduce the risk of bugs.

Code file as described in the video

```python
# n! = n * n-1 * n-2 * n-3.......1
# n! = n * (n-1)!
def factorial_iterative(n):
    """
        :param n: Integer
        :return: n * n-1 * n-2 * n-3.......1
    """
    fac = 1
    for i in range(n):
        fac = fac * (i+1)
    return fac

def factorial_recursive(n):
    """
        :param n: Integer
        :return: n * n-1 * n-2 * n-3.......1
    """
    if n ==1:
        return 1
    else:
        return n * factorial_recursive(n-1)
    # 5 * factorial_recursive(4)
    # 5 * 4 * factorial_recursive(3)
    # 5 * 4 * 3 * factorial_recursive(2)
    # 5 * 4 * 3 * 2 * factorial_recursive(1)
```

Anonymous/Lambda Functions In Python

As we have studied function in the previous lecture, we know that if a program has a large piece of code that is required to be executed repeatedly, then it will be better to implement that piece of code as a function. Functions improve code reusability, modularity, and integrity.

In this tutorial, we will see how to declare Anonymous functions and how to return values from them.

As we have studied in the previous lecture, the syntax for function declaration is as follows:

```
def function_name ():
```

What is Anonymous function?

In Python programming, an **anonymous function** or **lambda** expression is a **function definition** that is not bound to an identifier (def).

The anonymous function is an inline function. The anonymous functions are created using a lambda operator and cannot contain multiple expressions.

The following example will show how anonymous functions work:

```
result = lambda n1, n2, n3: n1 + n2 + n3;
print ("Sum of three values : ", result( 10, 20, 25 ))
```

In the above code, we have created an anonymous function that adds three numbers. The function is stored in a variable named result. The function can then be called using this variable. In the above code, the function has been called with three different parameter values for the function call.

Anonymous functions can accept inputs and return the outputs, just like other **functions do.**

Why do we use Python Lambda Functions?

The main objective of anonymous functions is that, when we need a function just once, anonymous functions come in handy. Lambda operator is not only used to create anonymous functions, but there are many other uses of the lambda expression. We can create anonymous functions wherever they are needed. Due to this reason, Python Lambda Functions are also called as throw-away functions which are used along with other predefined functions such as **reduce(), filter(),** and **map().**

These functions help reduce the number of lines of the code when compared to named Python functions.

Significant Differences Between Lambda Expressions And Simple Functions.

1. Can be passed immediately with or without variables.
2. They are inline functions.
3. Automatic return of results.
4. There is neither a document string nor a name.

Python List sort():

Sorting means arranging data systematically. If the data we want to work with is not sorted, we will face problems finding the desired element.

The Python language, like other programming languages, can sort data.

Python has an in-built method i.e. **sort(),** which is used to sort the elements of the given list in a specified ascending or descending order. There is also a built-in function i.e. sorted(), that builds a new sorted list from an iterable like list, dictionary, etc.

The syntax of the sort() method is:

```
list.sort(key=myFunc ,reverse=True|False)
```

Parameter Values:-

Parameters	Values
key	In the key parameter, we specify a function that is called on each list element before making comparisons.
reverse	This is optional. False will sort the list in ascending order, and true will sort the list in descending order. Default is reverse=False.

Sort() does not return any value, but it changes from the original list.

Code file as described in the video

```
# Lambda functions or anonymous functions
# def add(a, b):
#     return a+b
#
# # minus = lambda x, y: x-y
#
# def minus(x, y):
#     return x-y
#
# print(minus(9, 4))


a =[[1, 14], [5, 6], [8,23]]
a.sort(key=lambda x:x[1])
print(a)

```

Using Python External & Built In Modules

In Python, a module can be defined as a file containing a runnable python code. **The extension used by modules in Python is .py.** Hence any simple file containing Python code can be considered as a module if its extension is .py. The file and module names are the same; hence we can call a module only by name without using the extension. Along with this, a module can define functions, classes, and variables.

There are two types of modules.

- Built-in
- External

Built-in Modules:

Built-in modules are already installed in Python by default. We can import a module in Python by using the import statement along with the specific module name. We can also access the built-in module files and can read the Python code present in them. Newly integrated modules are added in Python with each update.

Some important modules names are as follows

Modules Names	Purpose
calendar	used in case we are working with calendars
random	used for generating random numbers within certain defined limits
enum	used while working with enumeration class
html	for handling and manipulating code in HTML
math	for working with math functions such as sin, cos, etc.
runpy	runpy is an important module as it locates and runs python modules without importing them first

External modules:

External modules have to be downloaded externally; they are not already present like the built-in ones. Installing them is a rather easy task and can be done by using the command "**pip install module_name**" in the compiler terminal. Being familiar with all the modules seems like a long shot for even the best of programmers because there are so many modules available out there. So, we can search and find modules according to our needs and use them as there is no need to remember all of them when we can simply look for them on the internet when the need occurs.

Being programmers, module makes our life a lot easy. Unlike programming languages in the past, also known as low-level programming languages, Python provides us with a lot of modules that make our coding much easy because we do not have to write all the code ourselves. We can directly access a module for a specific task. For example, to generate a random number within two numbers, known as its limit, we do not have to write a function ourselves with loops and a large number of lines of codes. Instead, we can directly import a module, and this makes our work simple.

We should not concern ourselves with the code written inside the module; instead, we can search the internet for the functions and properties. If we are not satisfied with the available module's work or could not find a module that could help us in the required manner, we can create our own module by making a file with **.py extension**. The module file will be like any other file you see in Python, the difference will just arise in the extension.

Code file as described in the video

```
import random
random_number = random.randint(0, 1)
# print(random_number)
rand = random.random() *100
# print(rand)
lst = ["Star Plus", "DD1", "Aaj Tak", "CodeWithHarry"]
choice = random.choice(lst)
print(choice)

```

F-Strings & String Formatting In Python

String formatting is used to design the string using formatting techniques provided by the particular programming language. From the % formatting to the format() method, to format string literals, there is no limit as to the potential of string crafting. There are four significant ways to do string formatting in Python. In this tutorial, we will learn about the four main approaches to string formatting in Python.

#1 String Formatting (% Operator)

Python has a built-in operation that we can access with the % operator. This will help us to do simple positional formatting. If anyone knows a little bit about C programming, then they have worked with printf statement, which is used to print the output. This statement uses the % operator. Similarly, in Python, we can perform string formatting using the % operator. For Example:

```
name="Jack”

 n="%s My name is %s” %name

print(n)

 Output: "My name is Jack."
```

The problem with this method is when we have to deal with large strings. If we specify the wrong type of input type operator, then it will throw an error. For Example, %d will throw a TypeError if the input is not an integer.

#2 Using Tuple ()

The string formatting syntax, which uses % operator changes slightly if we want to make multiple substitutions in a single string. The % operator takes only one argument, to mention more than one argument, use tuples. Tuples are better than using the old formatting string method. However, it is not an ideal way to deal with large strings. For Example:

```
name=”Jack”
class=5
s=”%s is in class %d”%(name,class)
print(s)
```

Output: Jack is in class 5.

#3 String Formatting (str.format)

Python 3 introduced a new way to do string formatting. format() string formatting method eliminates the %-operator special syntax and makes the syntax for string formatting more regular. str.format() allows multiple substitutions and value formatting. We can use format() to do simple positional formatting, just like you could with old-style formatting:

In str.format(), we put one or more replacement fields and placeholders defined by a pair of curly braces { } into a string.

Syntax: **{}.format(values)**

For Example,

```
str = "This article is written in {} "

print (str.format("Python"))
```

Output: This article is written in Python.

This string formatting method is preferred over %-style formatting. Using the format() method, we can deal with large strings, and the code will become more readable.

#4 Using f-Strings (f):

Python added a new string formatting approach called **formatted string literals or "f-strings**." This is a new way of formatting strings. A much more simple and intuitive solution is the use of Formatted string literals.**f-string** has an easy syntax as compared to previous string formatting techniques of Python. They are indicated by an **"f"** before the first quotation mark of a string. Put the expression inside { } to evaluate the result. Here is a simple example

```
## declaring variables

 str1="Python”

 str2="Programming”

print(f"Welcome to our {str1}{str2} tutorial”)
```

Output: Welcome to our Python Programming tutorial.

Time Module:-

The time module provides time-related functions. It handles the time-related tasks. To use functions that are defined in this module, we need to import the module first.

```
import time
```

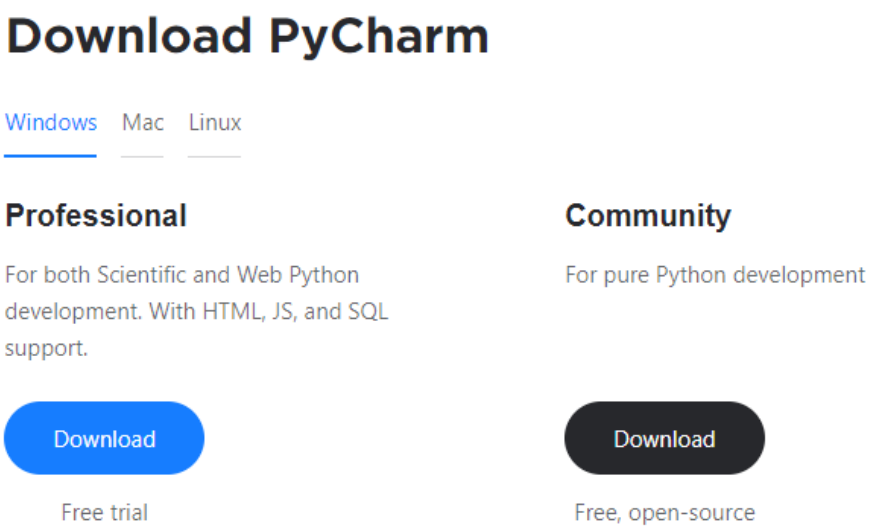
It is mostly used in measuring the time elapsed during program execution.

Code file as described in the video

```
# F strings
import math

me = "Harry"
a1 =3
# a = "this is %s %s"%(me, a1)
# a = "This is {1} {0}"
# b = a.format(me, a1)
# print(b)
a = f"this is {me} {a1} {math.cos(65)}"
# time
print(a)

```

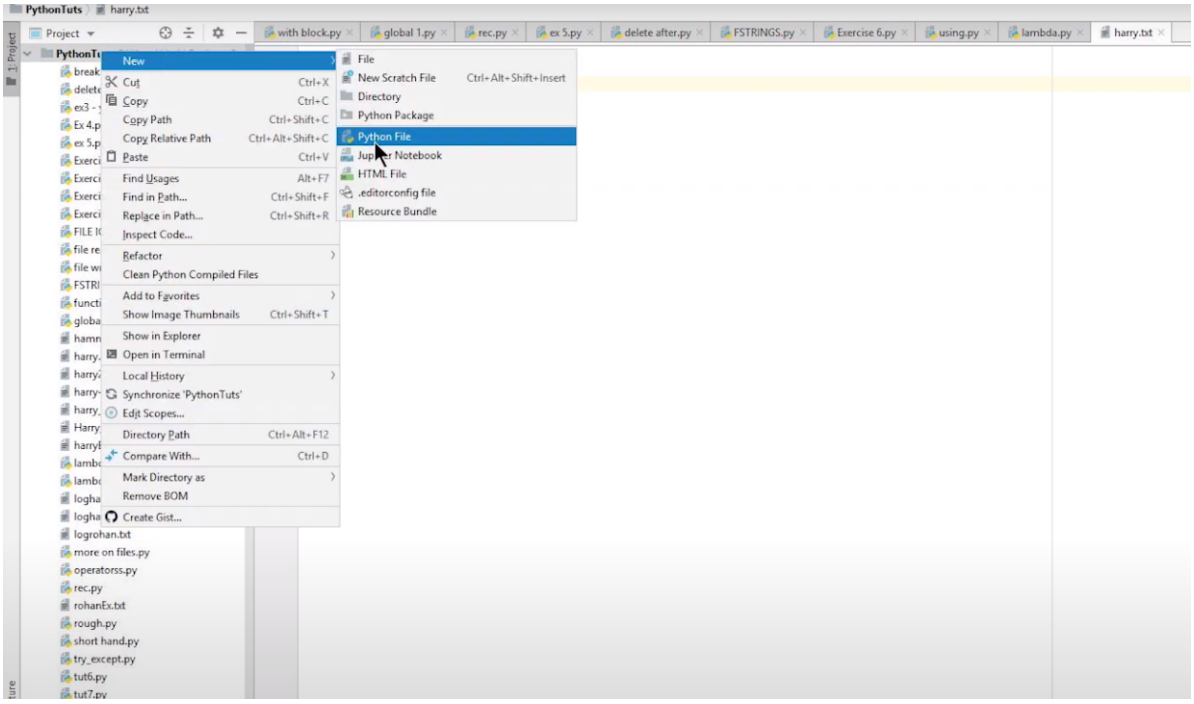
*args and **kwargs In Python

So, guys in this course we are working on Pycharm compiler. There are also many other options available like Spyder, Idle, Wing, etc. but we will go with Pycharm for this series and you will see its benefits in the upcoming tutorials. If you haven't downloaded it yet then download it by clicking on the download. This will take you to Pycharm official site.

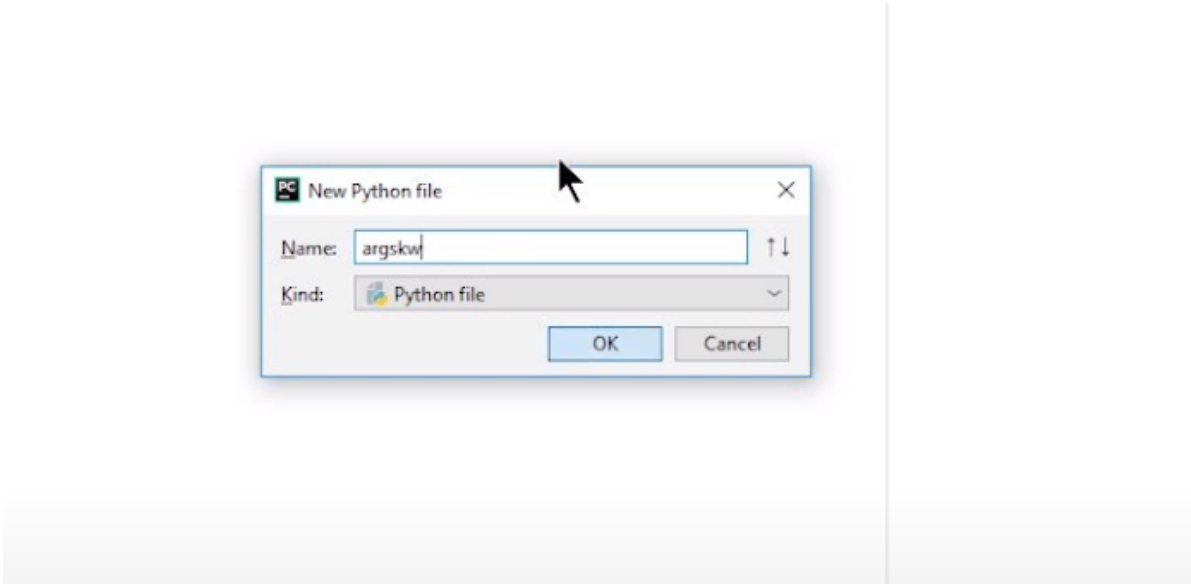


I will recommend you installing the community version as the other one will expire after a month trial and after that, you have to pay to use its features.

Moving forward let's just open our pycharm and create a new file.



As always I am again going to repeat: not to name our file similar to a module name. The reason for that, I have discussed in Tutorial #45. You can give it a read or watch the video for further clarification.



We are going to name our file argskw here.

We have worked with functions lately and also have made our own functions. We have seen that a function can only pass a certain number of arguments. The number of arguments has to be decided while defining the function, and it can not be changed while calling it. In simple terms, the number of arguments passed should be the same as the ones that are defined. If we dislike this restriction and do not want ourselves to be bound by certain limits, then we are lucky to have ***args** and ****kwargs** with us.

Before discussing *args and **kwargs, we should have a basic knowledge about types of arguments. In Python programming, there are two types of arguments that can be passed in a function.

- positional arguments
- keyword arguments

Positional arguments are the one in which an order has to be followed while passing arguments. We have been dealing with them until now.

We know how normal functions work with positional arguments, so now we will directly start exploring keyword arguments. But to understand them, we have to know about asterisk or more commonly known in Perl 6 and ruby as a splat operator.

Asterisk is used in python as a mathematical symbol for multiplication, but in case of arguments, it refers to unpacking. The unpacking could be for a list, tuple, or a dictionary. We will discover more about it by defining *args and **kwargs.

*args:

args is a short form used for arguments. It is used to unpack an argument. In the case of *args, the argument could be a list or tuple. Suppose that we have to enter the name of students who attended a particular lecture. Each day the number of students is different, so positional arguments would not be helpful because we can not leave an argument empty in that case. So the best way to deal with such programs is to define the function using the class name as formal positional argument and student names with parameter *args. In this way, we can pass student's names using a tuple.

Note that the name args does not make any difference, we can use any other name, such as *myargs. The only thing that makes a difference is the Asterisk()*.

**kwargs:

The full form of **kwargs is keyword arguments. It passes the data to the argument in the form of a dictionary. Let's take the same example we used in the case of *args. The only difference now is that the student's registration, along with the student's name, has to be entered. So what** kwargs does is, it sends argument in the form of key and value pair. So the student's name and their registration both can be sent as a parameter using a single ** kwargs statement.

Same as we discussed for args*, the name kwargs does not matter. We can write any other name in its place, such as **attendance. The only mandatory thing is the double asterisks we placed before the name.

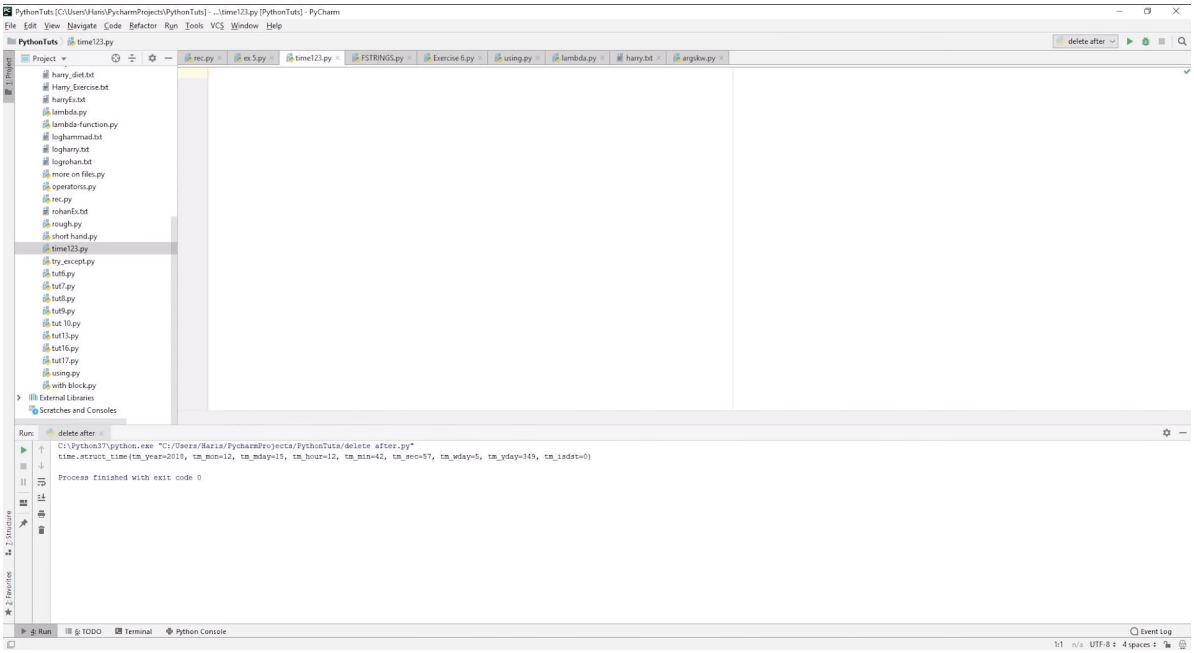
One of the instances where there will be a need for these keyword arguments will be when we are modifying our code, and we have to make a change in the number of parameters or a specific function.

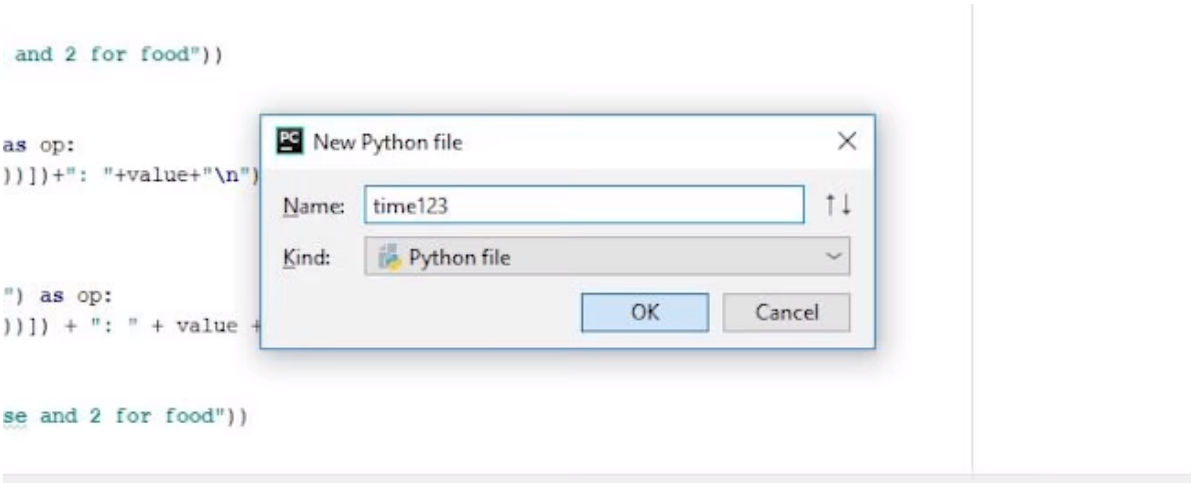
Code file as described in the video

```
# def function_name_print(a, b, c, d, e):
#     print(a, b, c, d, e)

def funargs(normal, *argsrohan, **kwargsbala):
    print(normal)
    for item in argsrohan:
        print(item)
    print("\nNow I would Like to introduce some of our heroes")
    for key, value in kwargsbala.items():
        print(f"{key} is a {value}")


# function_name_print("Harry", "Rohan", "Skillf", "Hammad", "Shivam")

har = ["Harry", "Rohan", "Skillf", "Hammad",
       "Shivam", "The programmer"]
normal = "I am a normal Argument and the students are:"
kw = {"Rohan":"Monitor", "Harry":"Fitness Instructor",
      "The Programmer": "Coordinator", "Shivam":"Cook"}
funargs(normal, *har, **kw)

```

Time Module In Python

Welcome to another tutorial guys. We are just going to open our Pycharm ide and start this tutorial without wasting any time.





We are going to name our file time123 here. Moving towards our today's topic.

In Tutorial #38, we discussed modules in detail, along with their types. We also discussed a few important modules in that tutorial. If you have not gone through it, then I would recommend you to go and see that one first. The **execution time of a program** is defined as the system's time to execute the task. As we know, all program takes some **execution time**, but we don't know how much. So, don't worry. In this tutorial, we will learn it by using a very helpful module known as the "Time module."

As can be defined by the name, ***"time module handles time-related tasks."***

It can be accessed by simply using an import statement.

```
1 import time
```

In Python, time can be tracked through its built-in libraries. The time module consists of all time-related functions. It also allows us to access several types of clocks required for various purposes. We will be discussing some of these important functions that are commonly used and come handy for further programming.

time.time():

It returns us the seconds of time that have elapsed since the Unix epoch. In simple words, it tells us the time in seconds that have passed since 1 January 1970. Its syntax is simple and easy to use.

```
import time
seconds = time.time()
print("Seconds since epoch =", seconds)
time.asctime():
```

We use the function time.asctime() to print the local time onto the screen. There are a lot of other ways to do it but time.asctime() prints the time in a sequence using a 24 characters string.

The format will be something like this: **Mon Feb 10 08:01:02 2020**

Time.sleep():

What sleep() function does is, it delays the execution of further commands for given specific seconds. In simple terms, it sends the program to sleep for some defined number of seconds. sleep() function is mostly used in programs directly connected to the operating system and in-game development. It halts the program execution, giving other programs a chance to get executed simultaneously.

The syntax is :

```
time.sleep(5)
```

The number of seconds is sent as a parameter within parenthesis. The program will go to sleep for 5 seconds after getting to this line of code and will continue its execution afterward.

time localtime():

The time.localtime() is used to convert the number of seconds to local time. This function takes seconds as a parameter and returns the date and time in time.struct_time format. It is optional to pass seconds as a parameter. If seconds is not provided, the current time will be returned by time() is used.

The syntax is:

```
time.localtime([ sec ])
```

For Example:

```
import time

print "time.localtime() returns: %s",%time.localtime()
```

Uses of time modules:

We can use the time module

- In games where missions depend on a certain time limit.
- To check the execution time a certain part of our code is taking.
- To print the date or local time onto the screen
- To suspend the execution of python thread.
- To measure the efficiency of the code.

There are many built-in functions in the *time* module. Not all of them are discussed in this tutorial. Explore more time module functions and use them in your code, so that you can measure the execution time of your code.

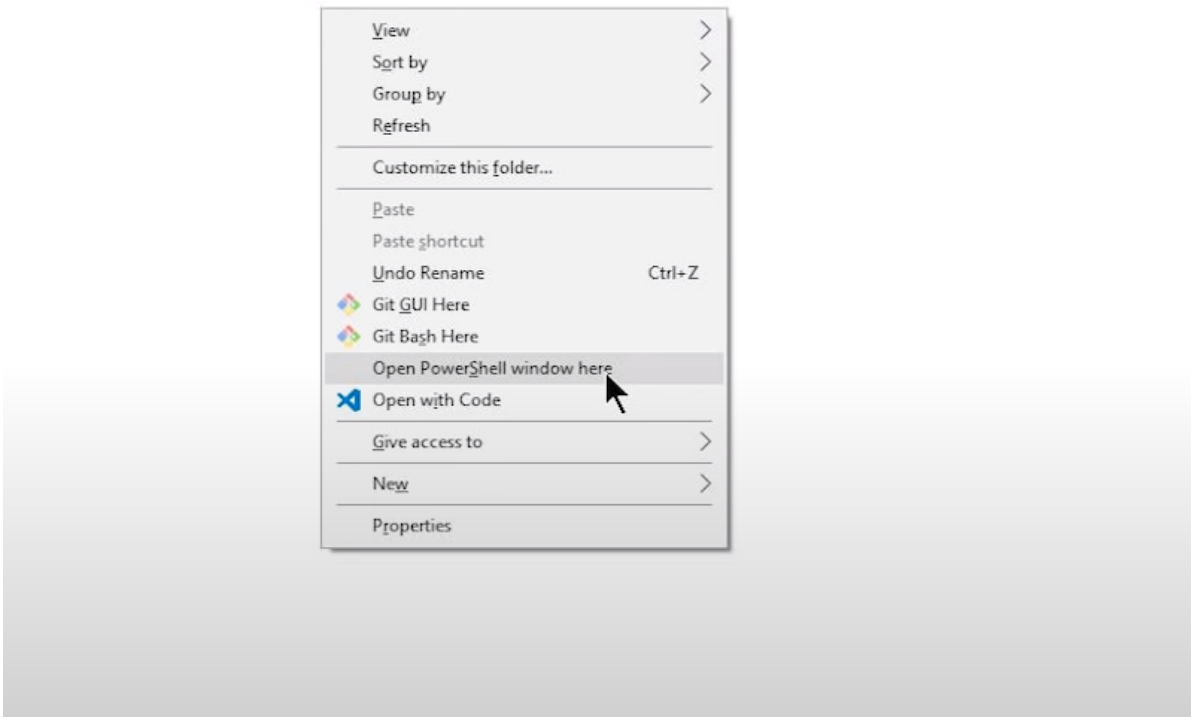
Code file as described in the video

```
import time
initial = time.time()

k = 0
while(k<45):
    print("This is harry bhai")
    time.sleep(2)
    k+=1
print("While loop ran in", time.time() - initial, "Seconds")

initial2 =time.time()
for i in range(45):
    print("This is harry bhai")
print("For loop ran in", time.time() - initial2, "Seconds")


# localtime = time.asctime(time.localtime(time.time()))
# print(localtime)

```

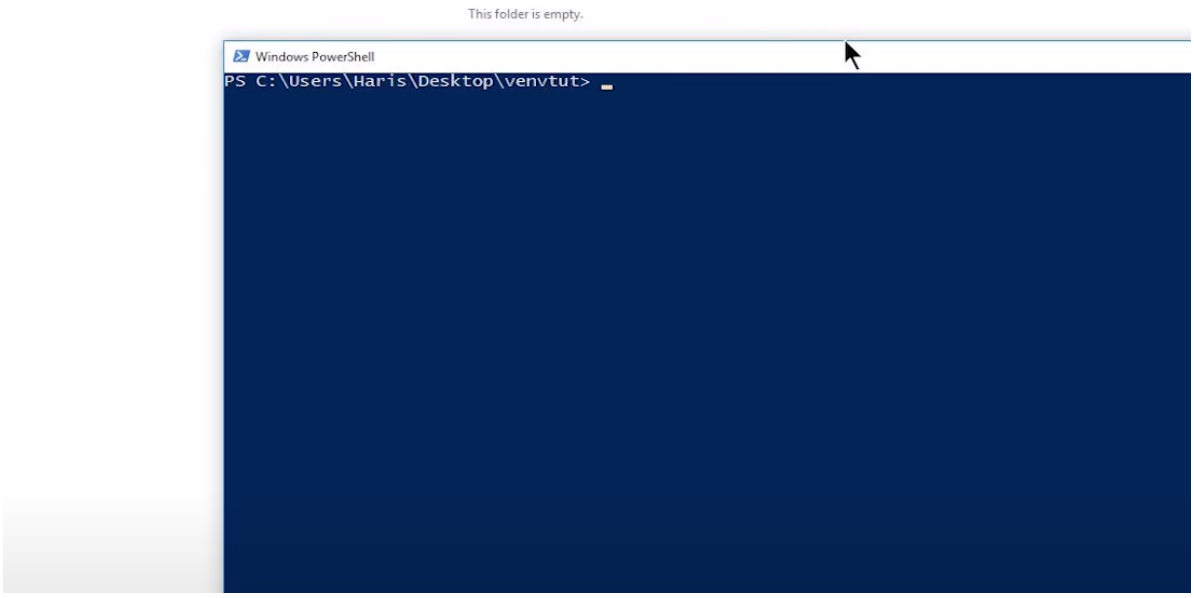
Virtual Environment & Requirements.txt

So, guys in this tutorial our introduction will be a little different than usual as we are not going be to using Pycharm. Instead today we are going to work with PowerShell. It may be a little unusual but don't worry, as I am here to guide you through every step.

So, let's get started. First, we will create a new folder. After opening the folder we will now open our PowerShell window using shift + mouse right-click.



Our PowerShell window will appear like this:



Note that the path it is showing is of our current folder. That is actually the benefit of using shift + mouse right-click.

Now let us move on to some theoretical concept.

A virtual environment is a tool or an aid provided to us by Python to keep the dependencies that we have utilized earlier in a few projects, constant. In simple terms, after some duration, Python keeps on launching and upgrading its versions. The new versions yet being better can have certain disadvantages for few users, because, in every new update, new functions are added to the modules, and previous ones may be upgraded. So, there is a chance that a function that used to work earlier will not work as it used to.

To save ourselves from such situations, Python has allowed us to use of a virtual environment.

What virtual environment does?

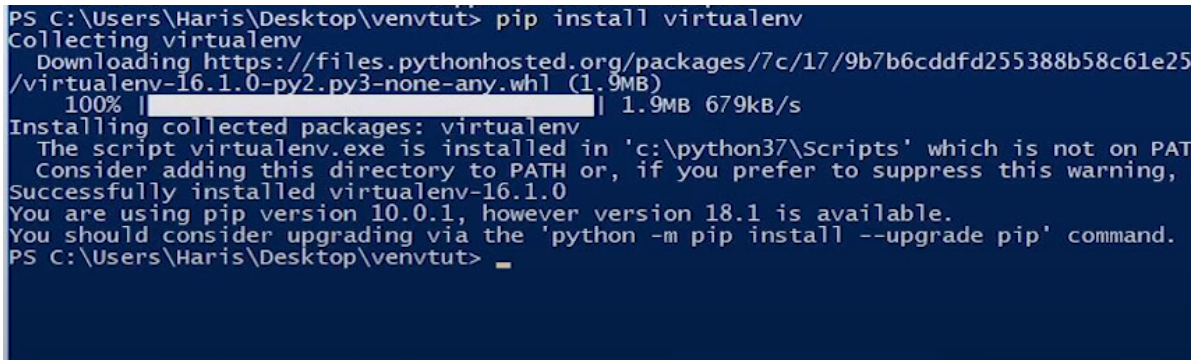
Virtual Environment saves the current state of our compiler along with the state of their modules and libraries. So in this way even if Python has made certain changes in its module, our virtual environment can still work as before even after years. We can also install different packages and **"dataframes"** in our virtual environment.

To be more clear, the virtual environment works exactly the same way as the Python we have installed on our windows/mac/Linux currently because a virtual environment is just a clone of the original product.

Using Virtual Environments

To get started, install the virtualenv tool with pip:

```
$ pip install virtualenv
```

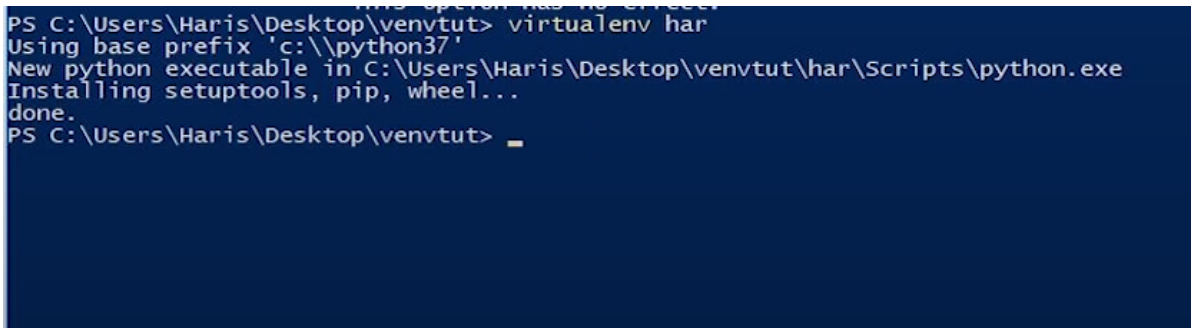


virtualenv is a tool. It is used to create isolated Python environments.

To assign a name to your virtual environment, use command

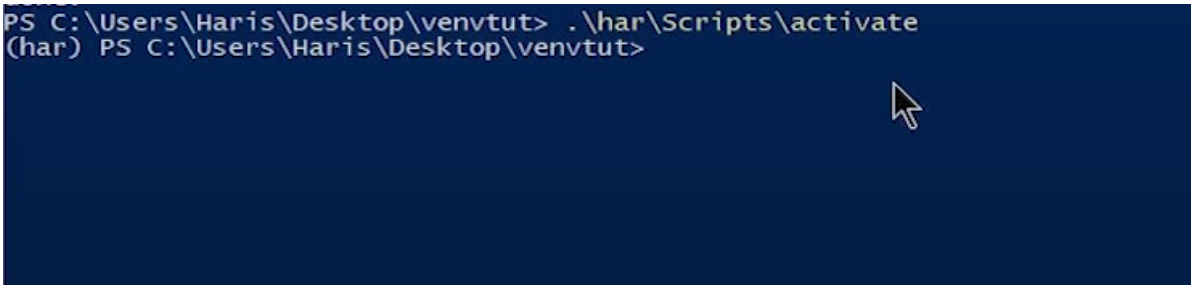
```
$ virtualenv virtualenv_name
```

After running this command, a directory named folder_name will be created. This directory will contain all the necessary executables to use the packages that the Python project would need. Python packages will be installed in this directory.



Now, after creating a virtual environment, you need to activate it. When you open the directory, it contains folders like include, lib, script, and tcl. When you open the script folder, you'll find a file activate.bat. You can activate the virtual environment by simply clicking on this file or using the command prompt and writing the following command.

```
$ .\virtualenv_name\Scripts\activate
```



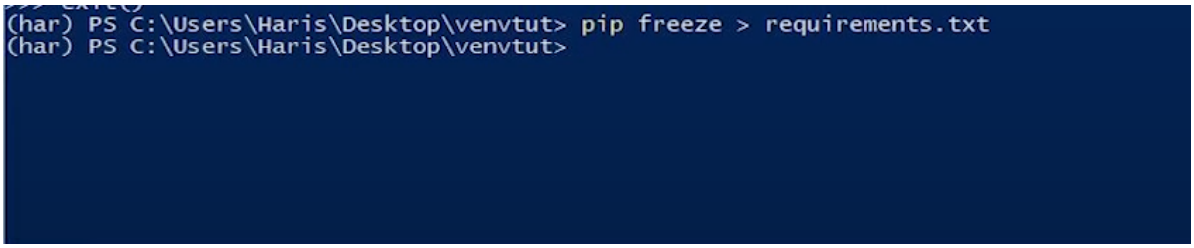
Once the virtual environment is activated, the name of your virtual environment will appear on the terminal's left side.

When you are done working in the virtual environment for the moment, you can deactivate it:

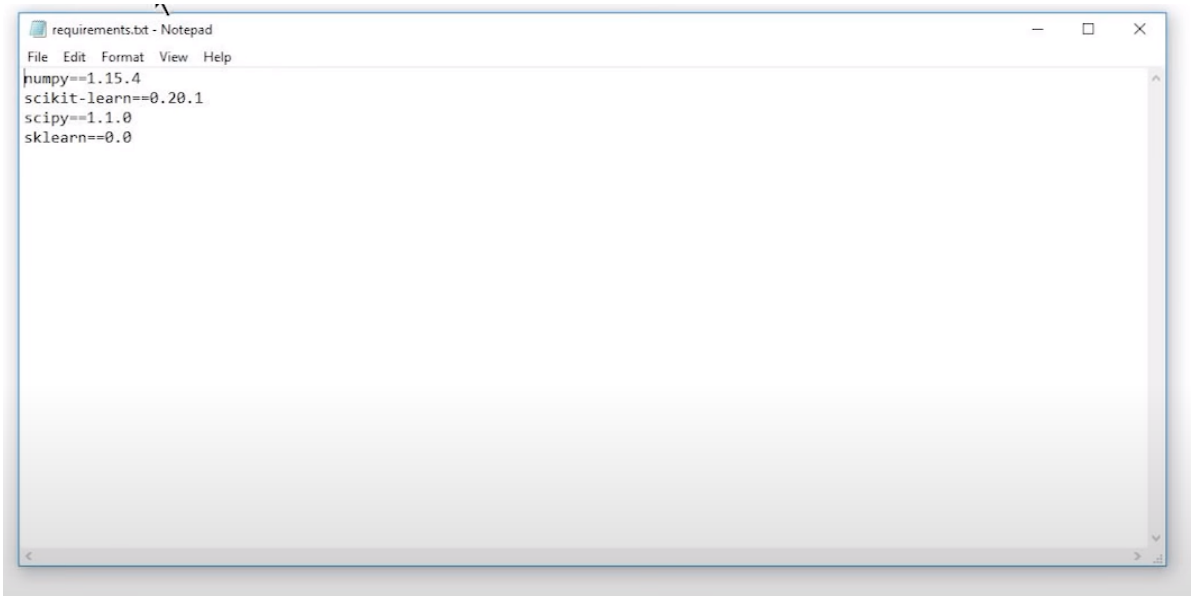
```
(virtualenv_name)$ deactivate
```

Now moving onto **requirement.txt**. When we run a certain command pip **freeze > requirement.txt,** a file will be generated in the directory where our virtual environment is based. The file will contain all the details related to the external packages that we have installed along with their versions. By having the requirement.txt file, we can create our virtual machine again easily by downloading all the same libraries, having the same versions by a simple command.

```
freeze > requirement.txt
```

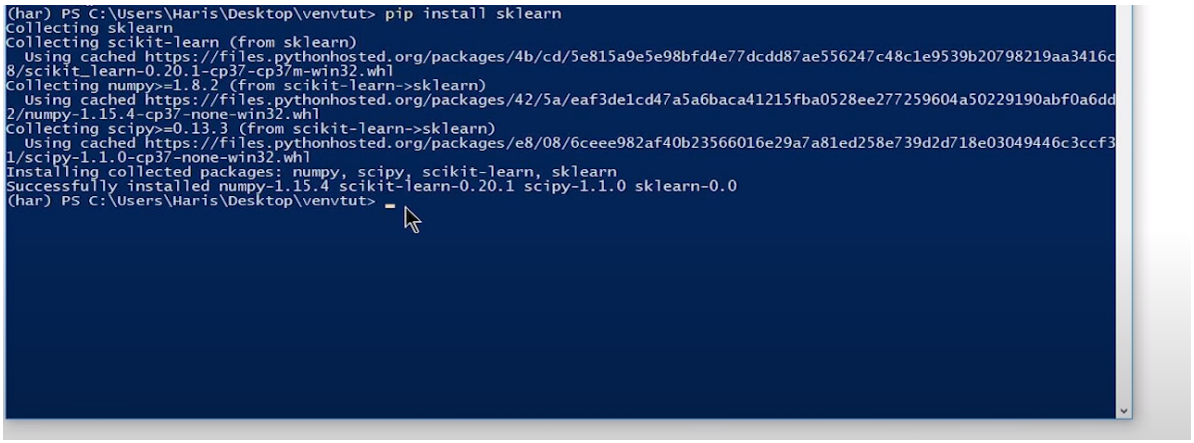


The requirement.txt folder will contain data like this:



We can install all the packages one by one by a command:

```
pip install package_name == version
```



But in case we have a large number of libraries installed, this will take a massive amount of time as we have to install each one by one so we have another method by which we can install all the packages at once by using the requirement.txt file. The syntax would be:

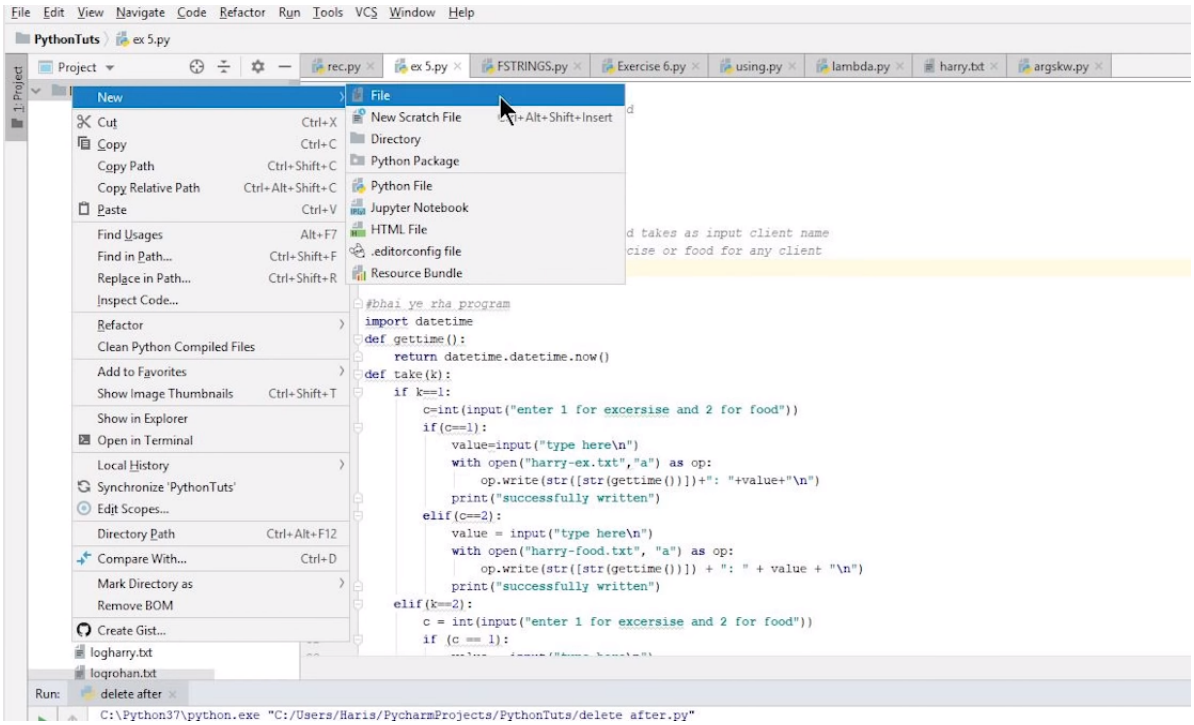
```
pip install -r .\requirements.txt
```

Code file as described in the video

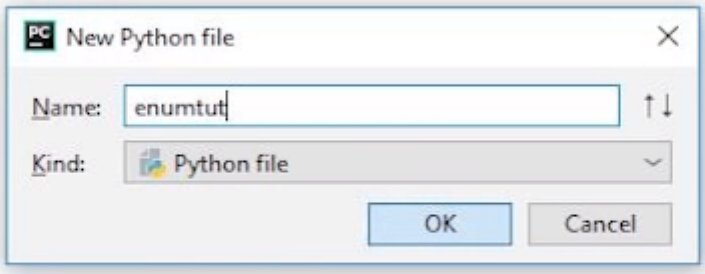
```
numpy==1.15.4
scikit-learn==0.20.1
scipy==1.1.0
sklearn==0.0
```

Enumerate Function

So guys, you must already have noticed that we are using Pycharm in all of our python tutorials. So it will be best if you use the same ide, so you can keep up with me in every way.



Note: Do not name your file similar to a module name. The reason is explained un Tutorial #45. Give it a read or watch the video for further clarification.



We are going to name our file enumtut here which means enumerate tutorial. So, let's get started without wasting time.

" An enumerate is a built-in function that provides an index to data structure elements, making iterations through them easier and simpler."

In previous tutorials, we have seen many methods that print items of different data structures onto the screen. For these kinds of operations, we need to have a for loop along with an iterator and an integer variable in which we have to increment every time the loop runs. Well, python provides us with a simple and easy solution to deal with certain situations by providing us with a

built-in function known as "**enumerate function**." Using the enumerate function, we can summarize the code and make it easier and simpler to use. It is really important to know the concept of Enumerate function so that you can apply this concept to your code.

Let us understand the working of enumerate function:

What enumerate function does is, it assigns an index to every element or value in the object that we want to iterate, so we do not have to assign a specific variable for incremental function, instead we have to apply a for loop, and our function will start working. Its syntax is a lot simpler and shorter than what we have been following till now.

Syntax

```
enumerate(iterable, start=0)
```

When calling a simple enumeration function, we have to provide two parameters:

- The data structure that we want to iterate
- The index from where we want to start our iteration

Note: The iterable must be an object that supports iteration

Example of enumerate using a python list.

We can iterate over the index and value of an item in a list by using a basic for loop with enumerate().

```
list_1=["code","with","harry"]
for index,val in enumerate(list_1):
    print(index,val)
```

Output:

```
0 code
1 with
2 harry
```

Using Enumerate() on a list with start Index:

In the below example, the starting index is given as 5. The index of the first item will start from the given starting index.

Example:

```
list_2 = ["Python", "Programming", "Is", "Fun"]
#Counter value starts from 5
result = enumerate(list_2, 5)
print(list(result))
```

We will get the following output:

```
[(5, 'Python'), (6, 'Programming'), (7, 'Is'), (8, 'Fun')]
```

If we do not provide the index, we want to start the iteration from then it automatically starts its iteration from zero index i.e., the beginning of the data structure.

Instead of returning a string, the enumerate function returns an object by adding the iterating counter value. We can also convert the enumerator object into a list(), tuple(), set(), and many more.

Advantages of using Enumerate:

- It is a built-in function
- It makes the code shorter
- We do not have to keep count of the number of iterations
- It makes the implementation of for loop simpler and cleaner
- Lesser code so lessor chances of error and bugs
- We can loop through string, tuple or objects using enumerate
- We can start the iteration from anywhere within the data structure as we have the option of providing the starting index for iteration.

Code file as described in the video

```
l1 = ["Bhindi", "Aloo", "chopsticks", "chowmein"]

# i = 1
# for item in l1:
#     if i%2 is not 0:
#         print(f"Jarvis please buy {item}")
#     i += 1

for index, item in enumerate(l1):
    if index%2==0:
        print(f"Jarvis please buy {item}")
```

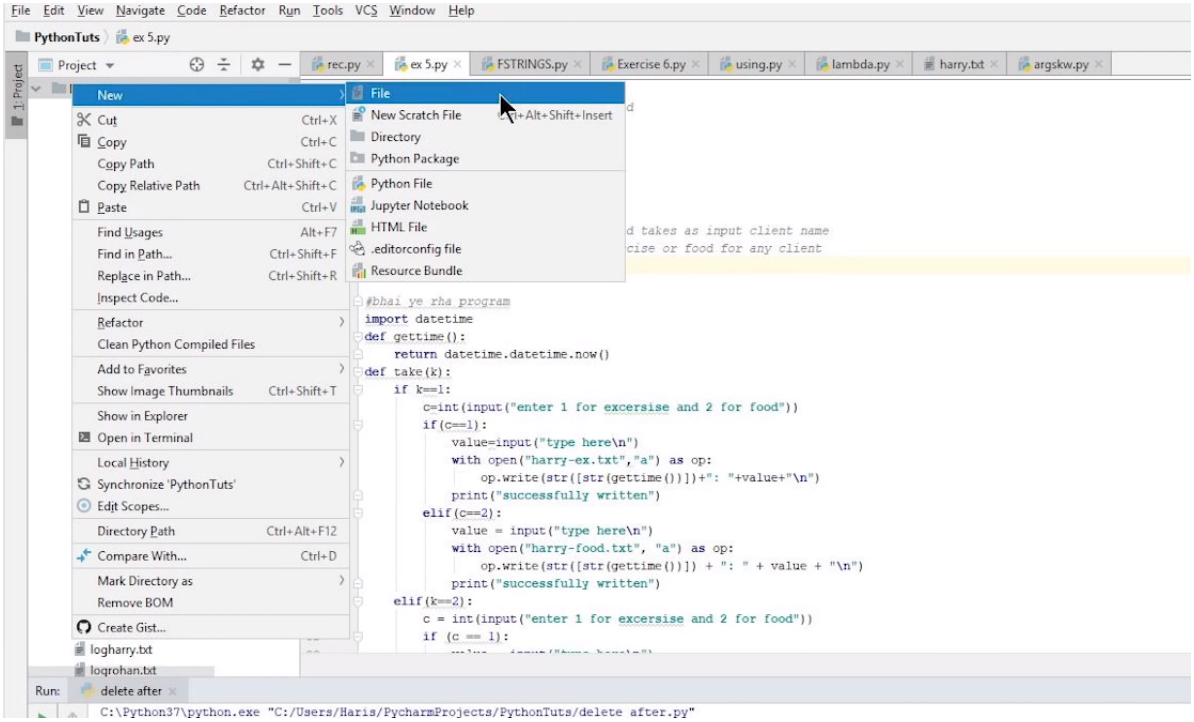
How Import Works In Python?

So let's just open our PyCharm community version so we can get started. I prefer choosing it because along with being free it also provides us with a number of features

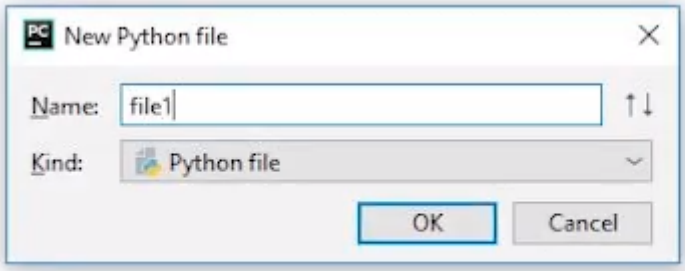
	PyCharm Professional Edition	PyCharm Community Edition
Intelligent Python editor	✓	✓
Graphical debugger and test runner	✓	✓
Navigation and Refactorings	✓	✓
Code inspections	✓	✓
VCS support	✓	✓

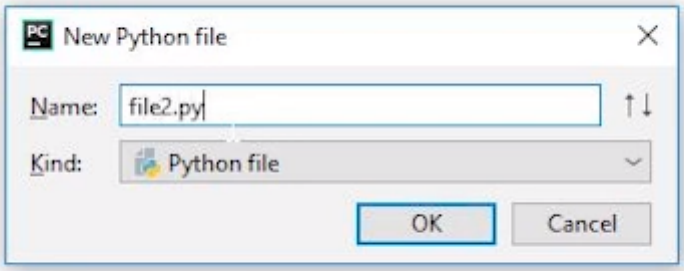
The professional edition is composed of a lot of extra features but community version will work fine for us at this level.

Moving forward let's just open our PyCharm and create a new file. Actually we are going to create two new files in this tutorial.



As always I am again going to repeat: not to name our file similar to a module name. The reason for that, I finally will be sharing with you in this particular tutorial.





We are going to name our files file1.py and file2.py.

In this tutorial we aim to understand the working of the import statement, so we can have a better grasp of the concept and resolve common importing issues. In Python, we give access to a module by using a keyword **import**. To use any package in our code, we must make it accessible by importing it first. There are many ways we can import a module in python, but what can be easier than using a single keyword, so it is also the most common way for importing.

How does the import keyword work?

When we write a certain module name along with the **import** keyword, it will start searching for a file with that name having an extension **.py**. After finding the file, it will import it into our program, which means that it will permit our program to use the functions of the certain module we imported. We can import a module named **"sys"** to see the path that our import statement takes while searching for a module.

```
1 import sys
2 print( sys.path)
```

sys.path prints out a list of directories. When we tell Python to import something, then it looks in each of the listed locations in order.

A common mistake that most of the beginners do and also the primary reason for making this tutorial is, why can't we name our file, the same as the name of a module. The reason is associated with the path. When we give our file a name same as the name of a module, then instead of importing the original module, the system will import our created file because it starts its search for the file from the directory where the file we are working on exists. So, we will not be able to use the functions of the original file.

There are two methods to use functions or variables after importing:

- The first one is to import using an object. For this, we usually import the whole module by using a simple import statement. When we use only the import keyword, we will import the resource directly, like this:

```
import sklearn
```

- When we use the second syntax, we will import the resource from another package or module. Here is an example:

```
from flask import Flask
```

We can also choose to rename an imported resource, like this:

```
import pandas as pd
```

This renames the imported resource pandas to pd.

We can not access it using pandas keyword; instead, we have to use pd or the compiler will show an error. This case comes in handy when the module name is difficult or lengthy, and we have to use a module again and again to call its functions.

Note: import module as module_name does not rename the module originally but only for a specific program where it is imported using this sort of keyword.

Disadvantages:

One of the major disadvantages of the flexibility provided by a python in the case of modules is that they can be easily modified and overridden. Along with disrupting the functionality of the program, it also poses a major security risk.

Code file1 as described in the video

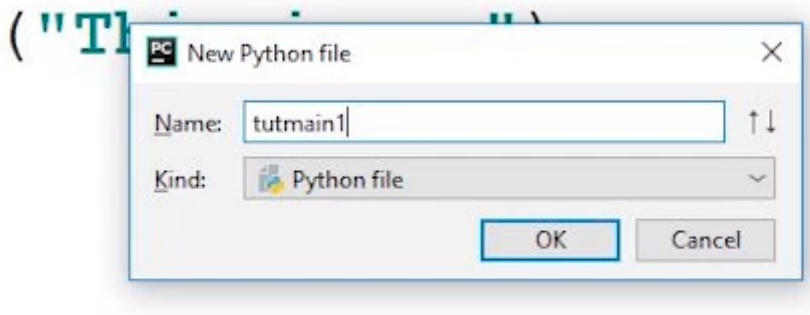
```
from sklearn.ensemble import RandomForestClassifier
print(RandomForestClassifier())


import file2
print(file2.a)

file2.printjoke("This is me")

```

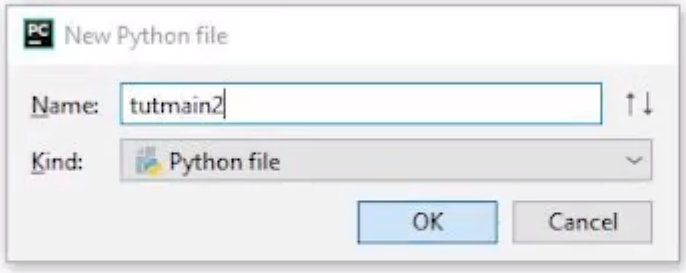
Code file2 as described in the video

```
a =7
def printjoke(str):
    print(f"this function is a joke {str}")

```

If name==main usage & necessity

This tutorial is a little different than previous one's, as we are working with multiples files here. So let's just open our iDE and start creating our files.





We are going to name our files tutmain1 and tutmain2.

Using **if name == "main"** statement is one such concept that may be difficult to grasp for beginners, but once learned, it is very helpful and used quite often afterward. However, it may seem confusing at times. This article aims to provide an understanding of the behavior of the statement and further discusses how to use it. As discussed earlier, a module is just a file containing a python code with a .py extension and can be imported to other files. That's where the keyword **name** comes in. Let's understand **name** first before moving onto if **name** == "**main**.

What is name?

"A name is a built-in variable that returns us the name of the module being used."

In simple words, by using **name**, we can check whether our module is being imported or run directly.

If we run it in the same module that it is created in, then it will print "main" onto the screen; otherwise, if it is being used elsewhere, then it will print the name of its module or file it is created in.

To fully understand what **name** is and how it is used, let us go through an example.

```
#tutmain1.py
print("__name__ in tutmain1.py is set to"+__name__)
```

Output:

```
__name__ in tutmain1.py is set to __main__
```

Let us create a new file **tutmain**2.py** in the same directory as **tutmain**1.py**

In this new file, let us import **tutmain**1.py** so that we can examine the **name** variable in **tutmain**2.py** and let us also print the **name** variable in **tutmain**2.py**

```
#tutmain2.py
import tutmain1

print("__name in tutmain2.py is set to"+__name__)
```

Output:

```
__name__ in tutmain2.py is set to tutmain1
```

Let us now move further to **"if name == "main"**. Working with Python files, when we import one file to another, along with the functions and variables, we also import all the print statements and other such data that we do not require. In such cases, we insert all the data of the module that we do not want others to import into the main, and thus it can only be executed by the file containing the main only.

Now we may have a certain confusion about "main", let us clear it out first. The main is a point of the program from where the program starts its execution. Every program has its own main function. The main function can only be executed when it is being run in the same program. If the file is being imported, then it is no longer the main function because the file that is importing it has its own "main" function.

The syntax is :

```
if__name__=="__main__":
    #Logic Statement
```

What are the Advantages of using if name == "main" statement?

Following are the advantages of using if **name** == "**main**" statement:

- Using the main in our file, we can restrict some data from exporting to other files when imported.
- We can restrict the unnecessary data, thus making the output cleaner and more readable.
- We can choose what others may import or what they may not while using our module.

To summarise the concepts discussed in this tutorial, Modules in Python has a special attribute called ***name***. The value of *the* ***name*** attribute is set to ***main*** when the module is run as the main program. Otherwise, the value of ***name*** is set to the name of the module. The if ***name ==*** *"**main**"*** block prevents the certain code from being run when the module is imported.

Code file as described in the video

```
def printhar(string):
    return f"Ye string harry ko de de thakur {string}"

def add(num1, num2):
    return num1 + num2 + 5


print("aand the name is", __name__)

if __name__ == '__main__':
    print(printhar("Harry1"))
    o = add(4, 6)
    print(o)

```

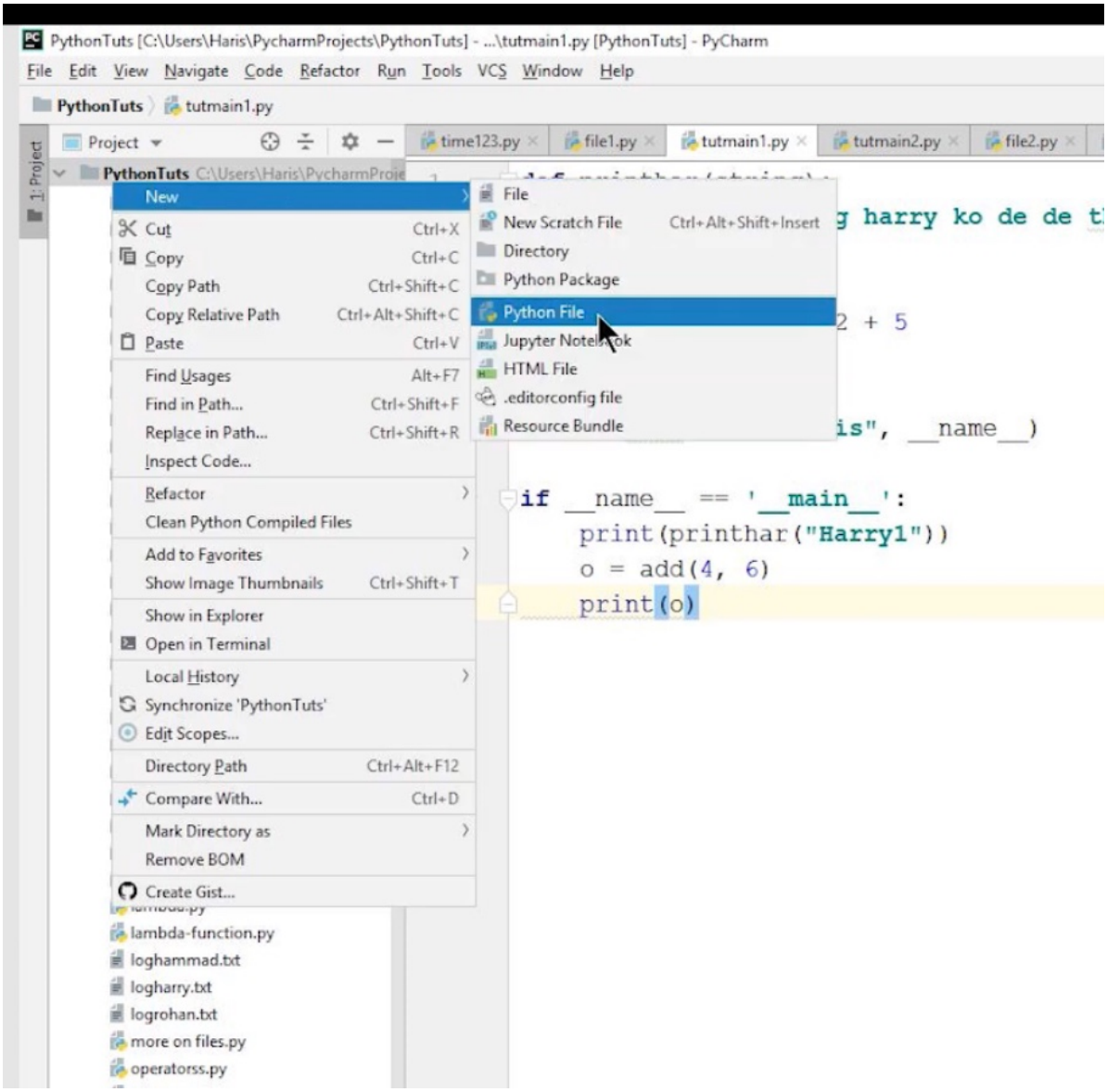
Join Function In Python

So guys, if you are working with Jupyter or nteract then its time to move onto Pycharm because we are moving to complex tutorials each day and it is best if you keep up with me in every way.

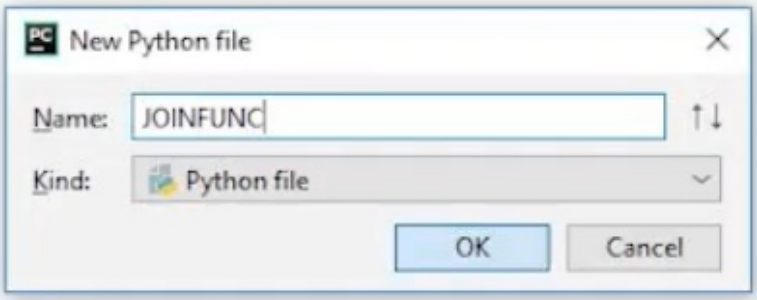
You can download the PyCharm community version from the following link:

PyCharm

Moving forward let's just open our PyCharm and create a new file,



As always I am again going to repeat: not to name our file similar to a module name. The reason for that, I have discussed in ***Tutorial #45***. You can give it a read or watch the video for further clarification.



We are going to name our file **JOINFUNC.py** here.

We are all familiar with the word join and it's meaning i.e., to combine. Join has nearly the same meaning in Python, as it is used to join the elements of a list, tuple, set, etc. In this article, we will learn about the join() method and how to use it. In Python, join() is a function that is used with iterables like list, dictionaries, and string. Examples are also mentioned below so you may learn how to use join() function. If you do not know what iterables are, then check our tutorials on ***Python Lists And List Functions***, ***Dictionary & It's Functions and String Slicing And Other Functions*** to get the basic idea about iterables.

What is the join method in Python?

"Join is a function in Python, that returns a string by joining the elements of an iterable, using a string or character of our choice."

In the case of join function, the iterable can be a list, dictionary, set, tuple, or even a string itself. The string that separates the iterations could be anything. It could just be a comma or a full-length string. We can even use a blank space or newline character (/n) instead of a string.

The syntax of the join() method is:

```
string.join(iterable)
```

the **string** is the name of string in which joined elements of iterable will be stored.

Note: If the iterable contains any non-string values, join() will raise a TypeError exception.

The implementation over the list iterable example is explained below. Here we join the elements of a list using a delimiter. A delimiter can be any character or nothing.

For Example:

```
#join() with lists
numList = ['1', '2', '3', '4']
separator = ', '
print(separator.join(numList))
```

Output:

```
1, 2, 3, 4
```

With the join function, we can join two strings together, changing it into a larger string. Along with the iterable, we also have to use a string between each iterable and a string on its own is also iterable; thus, two strings can also be joined by using the join function.
It's is a lot easier and more compact than using a loop. We use a variable for iteration along with a string or fstring to print all the elements onto the screen.

How will the join function work in case of a "dictionary"? Are there any limitations to join() function?

The join function has certain limitations. We must have a question in our mind that how join function will work in case of a "dictionary" where there are values along with the keys. In the case of the dictionary, the join function will only return the key part, separated by the string in between, leaving the value side behind.

For Example:

```
myDictionary = {"name": "Jack", "country": "America"}
separator = "_separator_"
print(separator.join(myDictionary))
```

Output:

```
name_separator_country
```

As we are on the subject, let us discuss another limitation associated with the join method. In situations where the iterable consists of a multi-data type, such as a list or tuple consisting of all integer variables and one single, double variable, the join function will not work. Instead, it will display an error. For join to function properly, all the variables should have the same sort of data type, either it is an integer, string, or any other.

For Example:

```
inputlist = ["Test1",13,"Test2",24,"Test3",100,"Test4"]
sep = '_'
out = sep.join(inputlist)
print(out)
```

Output:

```
Traceback (most recent call last): File "./prog.py", line 3, in TypeError:
sequence item 1: expected str instance, int found
```

Code file as described in the video

```
lis = ["John", "cena", "Randy", "orton",
       "Sheamus", "khali", "jinder mahal"]

# for item in lis:
#     print(item, "and", end=" ")

a = ", ".join(lis)
print(a, "other wwe superstars")
```

Map, Filter & Reduce

Python provides many built-in functions that are predefined and can be used by the programmers by just calling them. These functions not only ease our work but also create a standard coding environment. In this tutorial, we will learn about three important functions: **map(), filter,** and **reduce()** in Python. These functions are most commonly used with Lambda function. ***"Lambda functions are functions that do have any name"***. These functions are used as parameters to other functions. If you do not know what lambda functions are, then I recommend you to check ***Anonymous/Lambda Functions In Python*** tutorial first to avoid any confusion.

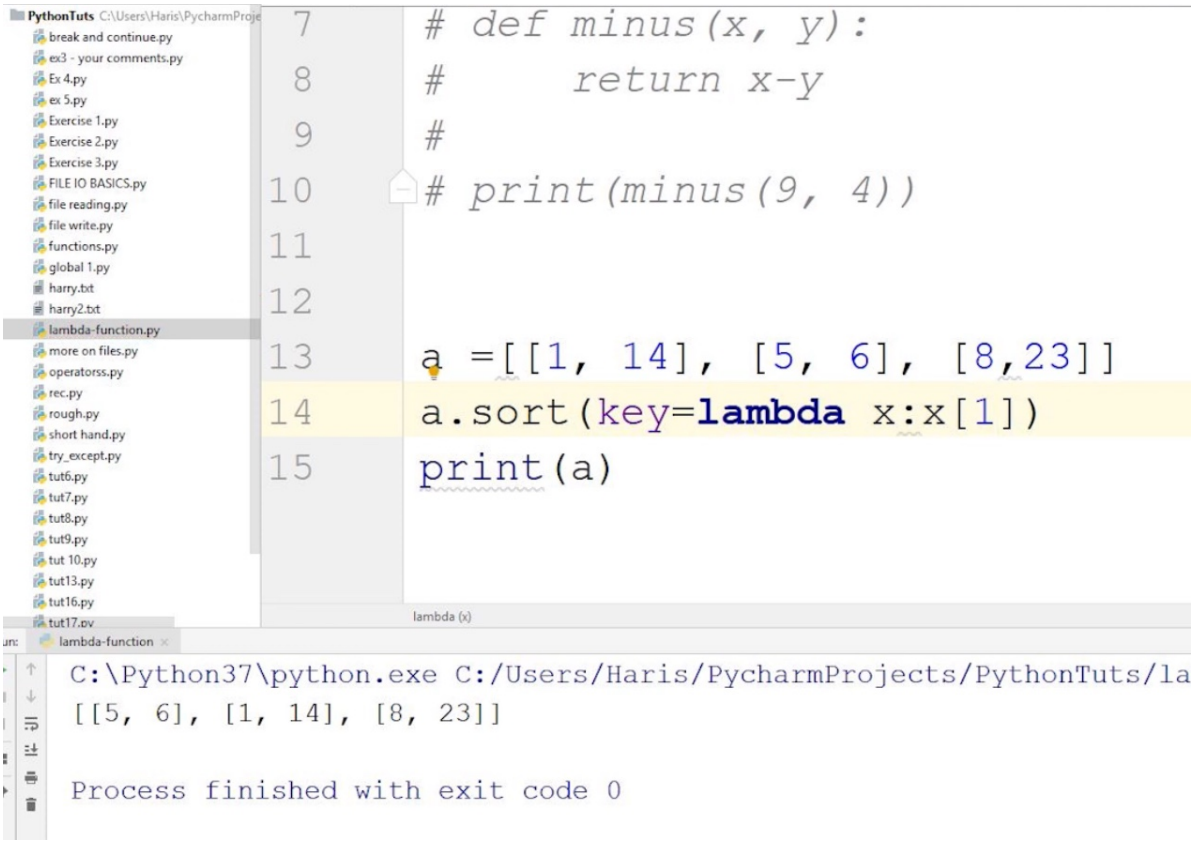


Figure 1:Anonymous/Lambda Functions In Python

Why are lambdas relevant to map(), filter() and reduce()?

These three methods expect a function object as the first argument. This function object can be a normal or predefined function. Most of the time, functions passed to map(), filter(), and reduce() are the ones we only use once, so there's often no point in defining a named function(def function).To avoid defining a new function, we write an anonymous function/lambda function that we will only use once and never again.

map():

"A map function executes certain instructions or functionality provided to it on every item of an iterable."

The iterable could be a list, tuple, set, etc. It is worth noting that the output in the case of the map is also an iterable i.e., a list. It is a built-in function, so no import statement required.

SYNTAX:

```
map(function, iterable)
```

A map function takes two parameters:

- First one is the function through which we want to pass the items/values of the iterable
- The second one is the iterable itself

Example:

```
items = [1, 2, 3, 4, 5]
a=list(map((lambda x: x **3), items))
print(a)
#Output: [1, 8, 27, 64, 125]
```

The map()function passes each element in the list to a lambda function and returns the mapped object.

filter():-

"A filter function in Python tests a specific user-defined condition for a function and returns an iterable for the elements and values that satisfy the condition or, in other words, return true."

It is also a built-in function, so no need for an import statement. All the actions we perform using the filter can also be performed by using a for loop for iteration and if-else statements to check the conditions. We can also use a Boolean that could take note of true or false, but that would make the process very lengthy and complex. So, to simplify the code, we can use the filter function.

SYNTAX:

```
filter(function, iterable)
```

It also takes two parameters:

- First one is the function for which the condition should satisfy
- The second one is the iterable

Example:

```
a = [1,2,3,4,5,6]
b = [2,5,0,7,3]
c= list(filter(lambda x: x in a, b))
print(c) # prints out [2, 5, 3]
```

reduce():

"Reduce functions apply a function to every item of an iterable and gives back a single value as a resultant".

Unlike the previous two functions (Filter and Map), we have to import the reduce function from functools module using the statement:

from functools import reduce

We can also import the whole functools module by simply writing

Import functools

But in the case of bigger projects, it is not good practice to import a whole module because of time restraint.

SYNTAX:

```
reduce(function, iterable)
```

Example:

```
from functools import reduce
a=reduce( (lambda x, y: x * y), [1, 2, 3, 4] )
print(a)
#Output: 24
```

Like the previous two, it also takes two-parameter. First one is the function and the second one is the iterable

Its working is very interesting as it takes the first two elements of the iterable and performs the function on them and converts them into a single element. It proceeds further, taking another element and performing the function of that one and the new element. For example, if we have four digits and the function wants to multiply them, then we can first multiply the first two and then multiply the third one in their resultant and then the forth and so on. The **reduce** is in the **functools** in Python 3.0. It is more complex. It accepts an iterator to process, but it is not an iterator itself. It returns a single result.

Code file as described in the video

```
#--------------------------MAP------------------------------
# numbers = ["3", "34", "64"]
# numbers = list(map(int, numbers))

# for i in range(len(numbers)):
#     numbers[i] = int(numbers[i])

# numbers[2] = numbers[2] + 1
# print(numbers[2])

# def sq(a):
#     return a*a
#
# num = [2,3,5,6,76,3,3,2]
# square = list(map(sq, num))
# print(square)
# num = [2,3,5,6,76,3,3,2]
# square = list(map(lambda x: x*x, num))
# print(square)


# def square(a):
#     return a*a
#
# def cube(a):
```

```
#     return a*a*a

# func = [square, cube]
# num = [2,3,5,6,76,3,3,2]
# for i in range(5):
#     val = list(map(lambda x:x(i), func))
#     print(val)

#-------------------------FILTER-----------------------------
# list_1 = [1,2,3,4,5,6,7,8,9]
#
# def is_greater_5(num):
#     return num>5
#
# gr_than_5 = list(filter(is_greater_5, list_1))
# print(gr_than_5)
#-------------------------REDUCE-----------------------------
from functools import reduce

list1 = [1,2,3,4,2]
num = reduce(lambda x,y:x*y, list1)
# num = 0
# for i in list1:
#     num = num + i
print(num)


```

Decorators In Python

In this tutorial, you will learn how you can create a decorator and why you should use it. Before you understand decorators, make sure you know about how Python functions work. If you don't know about python functions, then check our ***Functions And Docstrings*** tutorial because functions are the fundamental concept in understanding Python decorators. Functions in Python can be defined as lines of codes built to create a specific task and can be used again and again in a program when called.

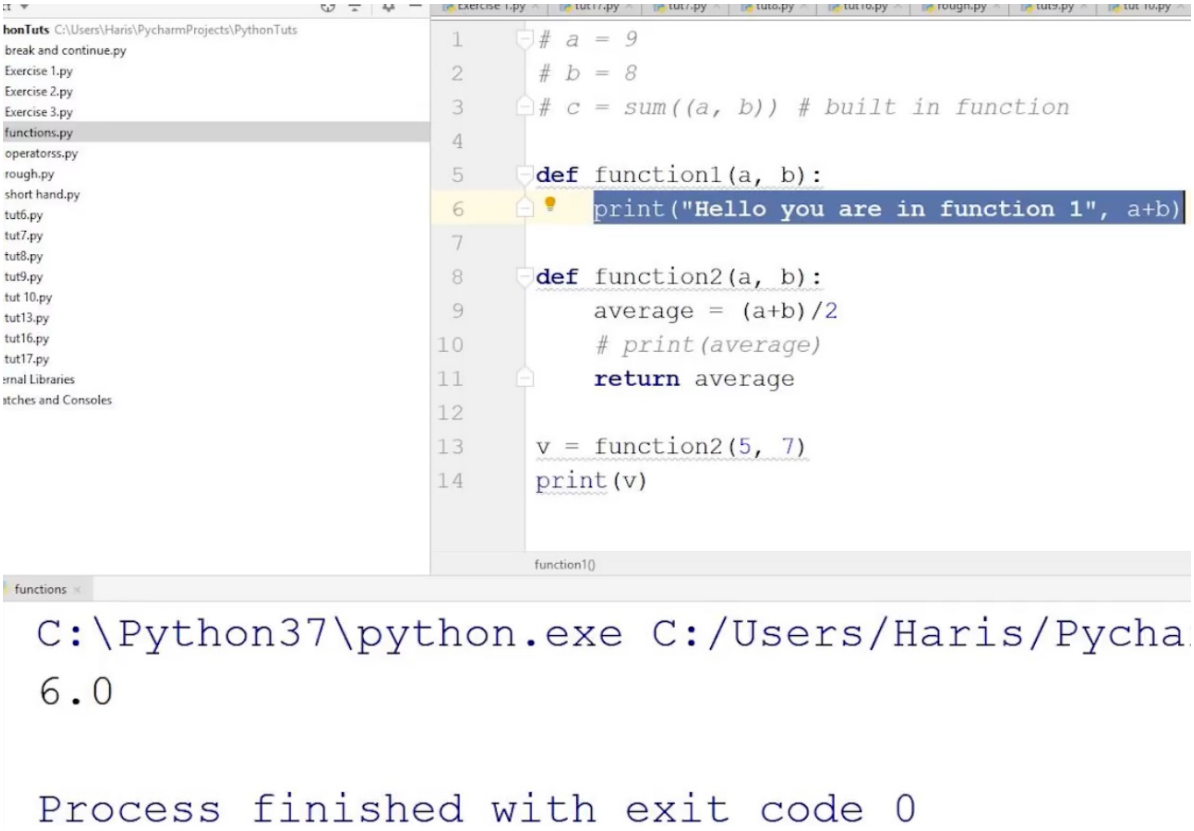


Figure1: Functions in Python

What is Python Decorator?

Decorator, as can be noticed by the name, is like a designer that helps to modify a function. The decorator can be said to be a modification to the external layer of function, as it does not change its structure. A decorator takes a function and inserts some new functionality in it without changing the function itself. A reference to a function is passed to a decorator, and the decorator returns a modified function. The modified functions usually contain calls to the original function. This is also known as **metaprogramming** because a part of the program tries to modify and add functionality to another part of the program at compile time. Understanding the definition could be difficult, but you can easily grasp the concept through the video section example. In terms of Python, the other function is also called a wrapper.

A **wrapper** is a function that provides a wrap-around another function. While using decorator, all the code executed before our function that we passed as a parameter and the code after it is executed belongs to the wrapper function. The purpose of the wrapper function is to assist us. Like if we are dealing with a number of similar statements, the wrapper can provide us with some code that all the functions have in common, and we can use a decorator to call our function along with the wrapper. A function can be decorated many times.

Note that a decorator is called before defining a function.

There are two ways to write a Python decorator:

- We can pass our function to the decorator as an argument, thus defining a function and passing it to our decorator.
- We can simply use the @ symbol before the function we'd like to decorate.

```
def inner1(func):
    def inner2():
        print("Before function execution");
        func()
        print("After function execution")
    return inner2

@inner1
def function_to_be_used():
    print("This is inside the function")

function_to_be_used()
```

Output:

```
Before function execution
This is inside the function
After function execution
```

Advantages:

• Decorator function can make our work compact because we can pass all the functions to a decorator that requires the same sort of code that the wrapper provides.
• We can get our work done without any alteration in the original code of our function.
• We can apply multiple decorators to a single function.
• We can use decorators in authorization in Python frameworks such as Flask and Django, Logging, and measuring execution time.

We can do a lot with decorators, like Multiple decorators that can be applied to a single function. I hope this tutorial serves as a good introduction to decorators in Python. After understanding the basics of Python decorator, learn more advanced use cases of decorators and how to apply them to classes.

Code file as described in the video

```
# def function1():
#     print("Subscribe now")
#
# func2 = function1
# del function1
# func2()

# def funcret(num):
#     if num==0:
#         return print
#     if num==1:
#         return sum
#
# a = funcret(1)
# print(a)
```

```python

# def executor(func):
#     func("this")
#
#
# executor(print)

def dec1(func1):
    def nowexec():
        print("Executing now")
        func1()
        print("Executed")
    return nowexec

@dec1
def who_is_harry():
    print("Harry is a good boy")

# who_is_harry = dec1(who_is_harry)

who_is_harry()
```

Class OOPS

Classes & Objects (OOPS)

Python is a powerful programming language that supports the **object-oriented programming** paradigm. In object-oriented programming, the program splits into self-contained objects. Each object represents a different part of the application which can communicate among themselves. We will be discussing classes and objects in more detail in the next tutorial, i.e., ***Creating Our First Class In Python***. The primary focus of this tutorial is to give you an understanding of Object-Oriented Programming or, in short form, **OOP**. A programming technique that requires the use of objects and classes is known as OOP. Object-Oriented Programming is based on the principle of writing reusable code that the user can access multiple times.

What is Python Class And Object?

A class is a collection of **objects**, and an object is defined as an instance of a class possessing attributes. The object is an entity that has a state and behavior. A class has all the similar attributes, like if we have a class students, then it will only consist of students' related data, such as subjects, names, attendance ratio, etc.

Along with classes and objects, you will learn many new terminologies related to OOP in further tutorials. Some of these terminologies are:

- Instances
- Constructor
- Methods
- Abstraction
- Inheritance

By using oop, we can divide our code into many sections known as **classes**. Each class holds a distinct purpose or usage. For example, if we have created a class named "Books," then all the attributes it possesses should be related to books, such as the number of pages, publishing date or price, etc.
There is no limit to the number of classes we can create in a program. Also, one class can be easily accessible by another, and we can also restrict the access of a class so other classes can not use its functions. This concept comes in handy while working on bigger projects. All the employees are given separate tasks to work on the classes they have been assigned. And after they are done with their contribution, the classes can be combined as a whole to form a complete project. So, now you can understand that to become a successful programmer, you must master the concept of OOP.

Object-oriented vs. Procedure-oriented Programming

Index	Object-oriented programming	Procedure Oriented Programming
1	Object-oriented programming is the problem-solving approach. The computation is done by using objects.	It is Structure oriented. Procedural programming uses a list of instructions. It performs the computation step by step.
2	OOP makes development and maintenance easier.	When the project becomes lengthy, it is not easy to maintain the code.
3	OOP provides a proper way for data hiding. It is more secure than procedural programming. You cannot access private data from anywhere.	Procedural programming does not provide any proper way for data binding, so it is less secure. In Procedural programming, we can access the private data.
4	Program is divided into objects	The program is divided into functions.

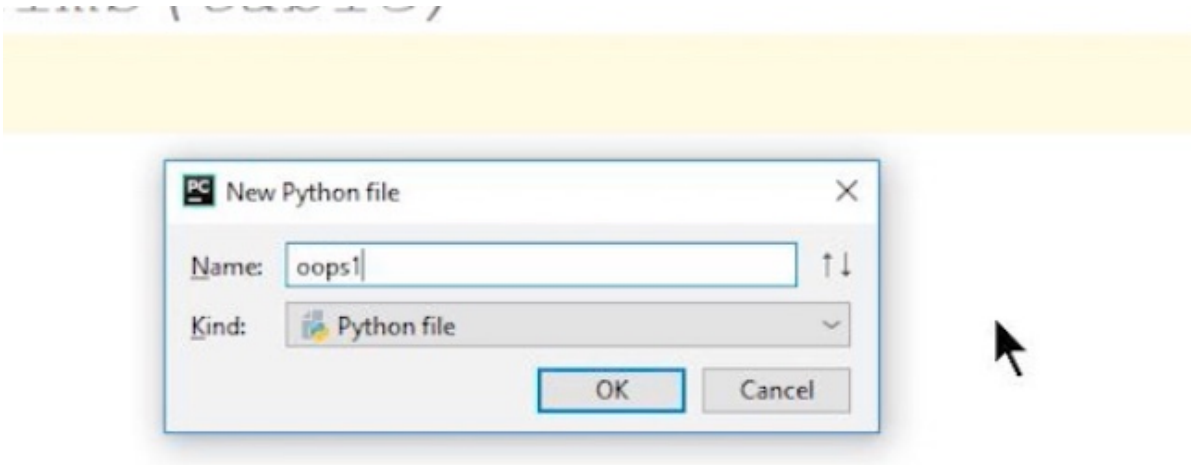
So this was a quick introduction to object-oriented programming. We will learn how to create a class in the next tutorial. Till then, keep coding and keep learning.

Code file as described in the video

```
# Classes - Template
# Object - Instance Of the Class

# DRY - Do not repeat Yourself

# get_no_of_films(table)

```

Creating Our First Class In Python

In the previous tutorial, we learned about OOP. In this one, we will make our first class using the concept of oop. Let's open our IDE and create a new file named oops1.



Let's begin our Learning:

As procedure-oriented programming focuses on functions, object-oriented programming stresses on objects. An object is simply a collection of data and methods, and a class is a blueprint for that object. We have already discussed the OOP in Python in the previous ***Classes & Objects(OOPS)*** tutorial. This tutorial is based on creating objects and classes.

Defining a Class in Python:

As in function, definitions begin with the keyword def, class begins with a **class** keyword.

```
1  class MyClass:
2      '''This is a docstring.'''
3      pass
```

A class is a blueprint from which objects are created. Creating a new class creates a new type of object, which allows the new instances of that type to be made. Each class instance has attributes attached so that it can maintain its state. Class instances can also have methods that are defined by their class for modifying their state.

Let us understand the concept of class through an example. Suppose we have to create a program that requires the data of all the individuals in a school. We will create three different classes, one for students, one for teaching staff, and one for accounting officers and others. The separation of the class is based on attributes because a teacher's attributes are different from students', and both have different attributes from the members of account officers. Although many attributes are the same, such as name, age, address, etc. but the teacher also has an attribute salary that the student does not or an attribute, number of classes that the accounts officers do not possess. So, now we have an understanding of how and on what basis we form different classes.

Classes are not like functions, so we do not have to use the keyword def to create a class; instead, we use the keyword **Class** along with the name of the class. Also, we do not call a class as a whole; instead, we use an object to access its different attributes. We can assign new values and can also overwrite the previous values with the help of an object. In short, an object gives us permission to access the whole class. We can access variables in a class, like:

```
Object_name.variable_name = “abc”
```

Here we are setting a variable equal to abc. By doing this, its previous value will be overwritten.

Creating Object:

Creating an object of a class is rather easy and simple. Suppose we have a class named Student. We can create an object of it by these certain lines of code:

```
Stu1 = Student()
Stu2 = Student()
```

Here we have created two objects of class Student. We can access every item in the Student class using these objects. There is no restriction on the number of objects a class may have, and also, there is no limit to the number of classes a program may have.

An object consists of :

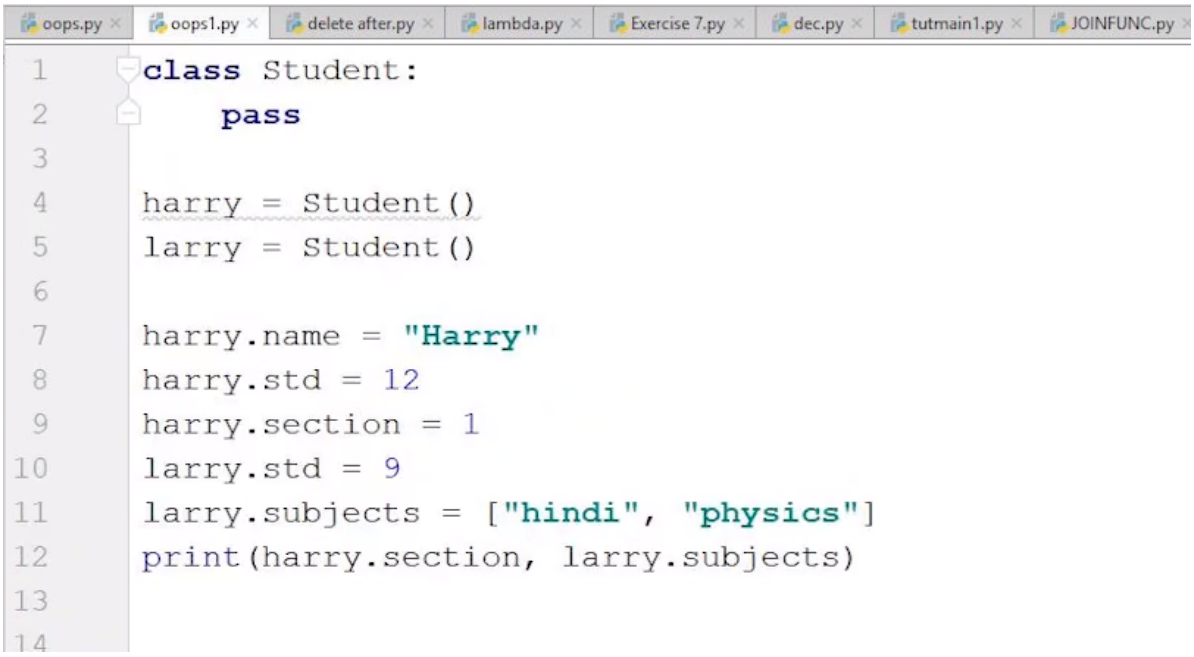
- The **State**, which is represented by attributes of an object which reflect the properties of an object.
- Methods of an object represent the object’s behavior and the response of an object with other objects.
- **Identity**, which gives a unique name to an object so that one object can interact with other objects.

Code file as described in the video:

```
class Student:
    pass

harry = Student()
larry = Student()

harry.name = "Harry"
harry.std = 12
harry.section = 1
larry.std = 9
larry.subjects = ["hindi", "physics"]
print(harry.section, larry.subjects)
```

Instance & Class Variables

In the ***previous tutorial***, we learned about the object and their working with variables in detail. Here we have created a class Student. We also created its two objects (harry and larry) and a few of their instance variables.



When working with objects in Python, we have to work with two types of variables, i.e., instance variables and class variables. But what do these types of variables mean, and how do they work? OOP allows the variables to be used at the class level or the instance level. In this tutorial, we will learn about the two different kinds of variables associated with a class and the difference between them. The variables are:

- Instance variable
- Class variable

Instance variable:

"Instance variables are the variables for which the value of the variable is different for every instance."

We can also say that the value is different for every object that we create. Let us dive into some in-depth explanations. When we create a class, we define a few variables along with it. For example, we have created a class of Students, and we have defined a variable age. All the students cannot have the same age in a class, so we have assigned the variable an average age of 16. Now, whenever we use an object to print the value of age, it will show 16. We can change the value of age, but it will create a new instance variable for the specific object that we are updating it for, hence defining the value to it.

The code for changing age for a particular object will be something like this:

Std1.age = 18

Class variable:

"Class attributes are owned by the class directly, which means that they are not tied to any object or instance."

Same as in the above example, if we want to change the age for every instance from 16 to 17, then we can do it by using the class variable, which in this case is Student.

"It is worth noting that updating the value of the class variable will not change it for the instance variables of the objects, such as in the case above."

The code for changing age using a class variable will be something like this:

Students.age = 18

The following are the notable differences between Class (static) and instance variables.

Following are the differences between Class and instance variables.

Instance variables	Class variables
When an object is created with the use of the new keyword, instance variables are created. They are destroyed when the object is destroyed.	When the program starts, static variables are created and destroyed when the program stops.
Instance variables can be accessed by calling the variable name inside the class. ***ObjectReference.VariableName***.**	Static variables can be accessed by calling using a class name. ***ClassName.VariableName***.**
Every instance of the class has its own copy of that variable. Changes made to the variable don't affect the other instances of that class.	There is only one copy of that variable that is shared with all instances of the class. If changes are made to that variable, all other instances will be affected.

The dict attribute

Every object in Python has an attribute that is denoted by **dict**. It maps the attribute name to its value. This dictionary stores all the attributes defined for the object itself. Following is the syntax of using **dict**:

```
object.__dict__
```

A quick review:

Instance variables are created only for a specific object. The object can change, create, or update only its instance variables. While in the case of class variables, the variables and values we create or define are set as default for all the objects. The objects cannot change the class value or variable by updating it using **object_name.variableName**.However, it can change the values of their particular instance variables. Making use of class and instance variables can ensure that our code adheres to the DRY (don't repeat yourself) principle to reduce repetition within code.

Code as described/written in the video

```
class Employee:
    no_of_leaves = 8
    pass

harry = Employee()
rohan = Employee()

harry.name = "Harry"
harry.salary = 455
harry.role = "Instructor"

rohan.name = "Rohan"
rohan.salary = 4554
rohan.role = "Student"

print(Employee.no_of_leaves)
print(Employee.__dict__)
Employee.no_of_leaves = 9
print(Employee.__dict__)
print(Employee.no_of_leaves)
```

Self & init() (Constructors)

In this tutorial, we will be discussing methods and constructors in detail. If you are familiar with any other object-oriented programming language, then the concept will be easy for you to grasp. So, let us begin with the method.

Method:

A method is just like a function, with a **def** keyword and a single parameter in which the object's name has to be passed. The method's purpose is to show all the details related to the object in a single go. We choose variables that we want the method to take but do not have to pass them as parameters. Instead, we have to set the parameters we want to include in the method during its creation. Using methods makes the process simpler and a lot faster.

Self keyword:

The self keyword is used in the method to refer to the instance of the current class we are using. The self keyword is passed as a parameter explicitly every time we define a method.

```
def read_number(self):
        print(self.num)
```

init method:-

"**init**" is also called a constructor in object-oriented terminology. Whereas a constructor is defined as:

"Constructor in Python is used to assign values to the variables or data members of a class when an object is created."

Python treats the constructor differently as compared to C++ and Java. The constructor is a method with a def keyword and parameters, but the purpose of a constructor is to assign values to the instance variables of different objects. We can give the values by accessing each of the variables one by one, but in the case of the constructor, we pass all the values directly as parameters. Self keyword is used to assign value to a constructor too.

As there can be only one constructor for a specific class, so the name of the constructor is a constant, i.e., **init.**

We declare a constructor in Python using the def keyword,

```
def __init__(self):
    # body of the constructor
```

Here,

- The def keyword is used to define the function.
- The first argument refers to the current object which binds the instance to the init() method.
- In init() method ,arguments are optional. Constructors can be defined with any number of arguments or with no arguments.

For Example:

```python
class Person:
  def __init__(self, name, age):
    self.name = name
    self.age = age

p1 = Person("John", 36)
print(p1.name)
#Output: John
```

Types of constructors in Python

We have two types of constructors in Python.

1. The default constructor is the one that does not take any arguments.
2. Constructor with parameters is known as parameterized constructor.

Method vs. Function:

Methods and functions are very similar, yet there are some differences:

- Methods are explicitly for Object-Oriented programming.
- The method can only be used by the object that it is called for. In simple terms, for a method, the parameter must be an object.
- The method can only access the data that is initialized in the class the method is formed in.

Code as described/written in the video

```python
class Employee:
    no_of_leaves = 8

    def __init__(self, aname, asalary, arole):
        self.name = aname
        self.salary = asalary
        self.role = arole

    def printdetails(self):
        return f"The Name is {self.name}. Salary is {self.salary} and role is {self.role}"


harry = Employee("Harry", 255, "Instructor")

# rohan = Employee()
# harry.name = "Harry"
# harry.salary = 455
# harry.role = "Instructor"
#
# rohan.name = "Rohan"
# rohan.salary = 4554
# rohan.role = "Student"

print(harry.salary)
```

Class Methods In Python

Before moving forward to our topic, I would like to share the link to all the exercises we have done so far. If you have been following this course since the beginning, then they are an excellent way to test your skills.

- Exercise 1
- Exercise 2
- Exercise 3
- Exercise 4
- Exercise 5
- Exercise 6
- Exercise 7

We have been dealing with static methods until now. In Object-Oriented programming, there is a concept of a **class method**, which we will see today in this lecture. They are very different from static methods as they are limited in their functionality to the built-in class. They can be called by using the class name and also can be accessed by using the object.

As we have observed in the previous tutorials, we cannot change the value of a variable defined in the class from outside using an object. Instead, if we try that, a new instance variable will be created for the class having the value we assigned. But no change will occur in the original value of the variable.

We saw the use of the self keyword in the previous tutorial. In this tutorial, we are going to know the working of a new keyword, i.e., cls. Class methods take **cls** parameter that points to the class and not the object instance when the method is called.

Syntax:

```
class myClass:
    @classmethod
    def myfunc (cls, arg1, arg2, ...):
                            ....
```

myfunc defines the function that needs to be converted into a class method

returns: @classmethod returns a class method for function.

Because the class method only has access to the **cls** argument, it cannot modify the object instance state. However, class methods can still modify the class state that applies to all instances of the class. So a class method automatically recognizes a class, so the only parameter that remains to be passed is the function that needs conversion.

@classmethod Decorator:

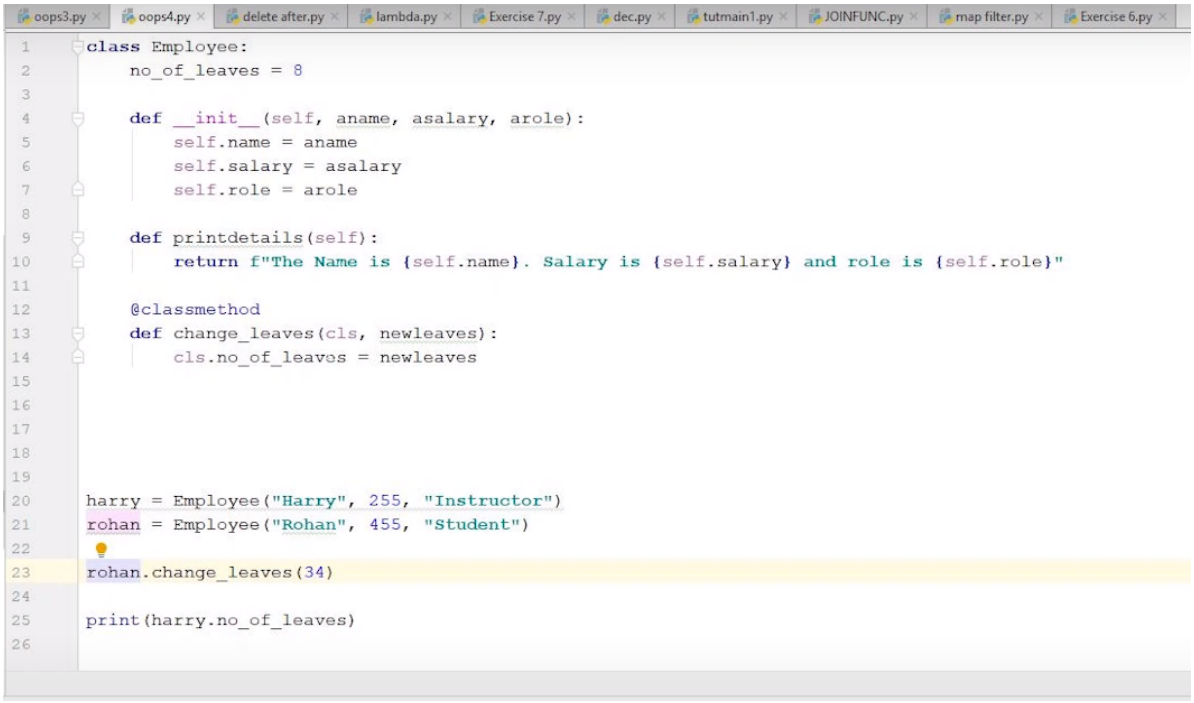
We have covered decorators in detail in Tutorial #51, so here we are just doing to define the functionality of decorators instead of its working. A **@classmethod** Decorator is a built-in function in Python. It can be applied to any method of the class. We can change the value of variables using this method too.

Differences between Class Method and Static Method:

Class method	Static Method
Taking a class or, in short, form cls as an argument is a must for a class method.	There is no such restriction of any specific parameter related to class in the Static method.
With the help of class methods, we can change and alter the variables of the class.	With a static method, we can not change or alter the class state.
Class methods are restricted to OOPs, so we can only use them if a class exists.	The static method is not restricted to a class.
We generally use class methods to create factory methods. Factory methods return a class object which is similar to a constructor for different use cases.	Static methods are used to create utility functions.

A quick overview:

The Python class method is a way to define a function for the Python class. It receives the class as an implicit first argument. Using **@classmethod** decorator, creating as many constructors for a class that behaves like a factory constructor is possible. Hopefully, now you can apply this concept to your projects and use it to improve your code's organization and quality.



Class Methods As Alternative Constructors

Before moving forward to our topic, I would like to share with you the link to all the exercises we have done so far. If you are following this course since the beginning then they are a great way for you to test your skills.

- Exercise 1
- Exercise 2
- Exercise 3
- Exercise 4
- Exercise 5
- Exercise 6
- Exercise 7

In this tutorial, we will learn how we can convert a class method into an alternating constructor. This tutorial is not about "what," as we have seen in previous tutorials, where we learn about new concepts or functionality. But this one is about" how." We will focus more on implementation as we are already familiar with all the concepts we are going to use. In this tutorial, we are going to learn some new skills or techniques other than a new concept.

We learned about constructor and its functionality in tutorial #55, where we had to set all the values as parameters to a constructor. Now we will learn how to use a method as a constructor. It has its own advantages. By using a method as a constructor, we would be able to pass the values to it using a string.

Note that we are talking about a class method, not a static method.

The parameters that we have to pass to our constructor would be the class i.e., cls and the string containing the parameters. Moving on towards the working, we have to use a function "**split()**," that will divide the string into parts. And the parts as results will be stored in a list. We can now pass the parameters to the constructor using the index numbers of the list or by the concept of *args (discussed in tutorial#43).

split():

Let us have a brief overview of the split() function. What split() does is, it takes a separator as a parameter. If we do not provide any, then the default separator is any whitespace it encounters. Else we can provide any separator to it such as full stop, hash, dash, colon, etc. After separating the string into parts, the split() function stores it into a list in a sequence. For example:

```
text = "Python tutorial for absolute beginners."
 t = text.split()
 print(t)
```

Here, we are not providing it any separator as a parameter, so it will automatically divide, taking whitespace as a separator.

The output will be a list, such as:

['Python', 'tutorial', 'for', 'absolute', 'beginners.']

Example of Class methods - alternative constructor:

```
class Date:
    def __init__(self, year, month, day):
        self.year = year
        self.month = month
        self.day = day

@classmethod
    def from_dash(cls,string):
          return cls(*string.split("-"))

date1=Date.from_dash("2008-12-5")
print(date1.year)
#Output: 2008
```

If we want multiple and independent "constructors", we can use class methods. They are usually called factory methods. It does not invoke the default constructor **init**.In the above example, we split the string based on the "-" operator. We first create a class method as a constructor that takes the string and split it based on the specified operator. For this purpose, we use a split() function, which takes the separator as a parameter. This alternative constructor approach is useful when we have to deal with files containing string data separated by a separator.

Code as described/written in the video

```
class Employee:
    no_of_leaves = 8

    def __init__(self, aname, asalary, arole):
        self.name = aname
        self.salary = asalary
        self.role = arole

    def printdetails(self):
        return f"The Name is {self.name}. Salary is {self.salary} and role
is {self.role}"

    @classmethod
    def change_leaves(cls, newleaves):
        cls.no_of_leaves = newleaves

    @classmethod
    def from_dash(cls, string):
        # params = string.split("-")
        # print(params)
        # return cls(params[0], params[1], params[2])
        return cls(*string.split("-"))


harry = Employee("Harry", 255, "Instructor")
rohan = Employee("Rohan", 455, "Student")
karan = Employee.from_dash("Karan-480-Student")

print(karan.no_of_leaves)
# rohan.change_leaves(34)
#
```

```
31 # print(harry.no_of_leaves)
```

Static Methods In Python

As discussed in ***Tutorial#56***, there are two types of methods, i.e., static and class. In that tutorial, our primary focus was on class methods. From this tutorial, you will get to learn about the **Static method**. It is one of the important concepts to use while you are learning OOPs programming with Python. A static method is very much easy to understand if we are familiar with class methods. It is also easier to implement than a class method because it can be accessed without any object. However, we can also access it using a class or any instance.

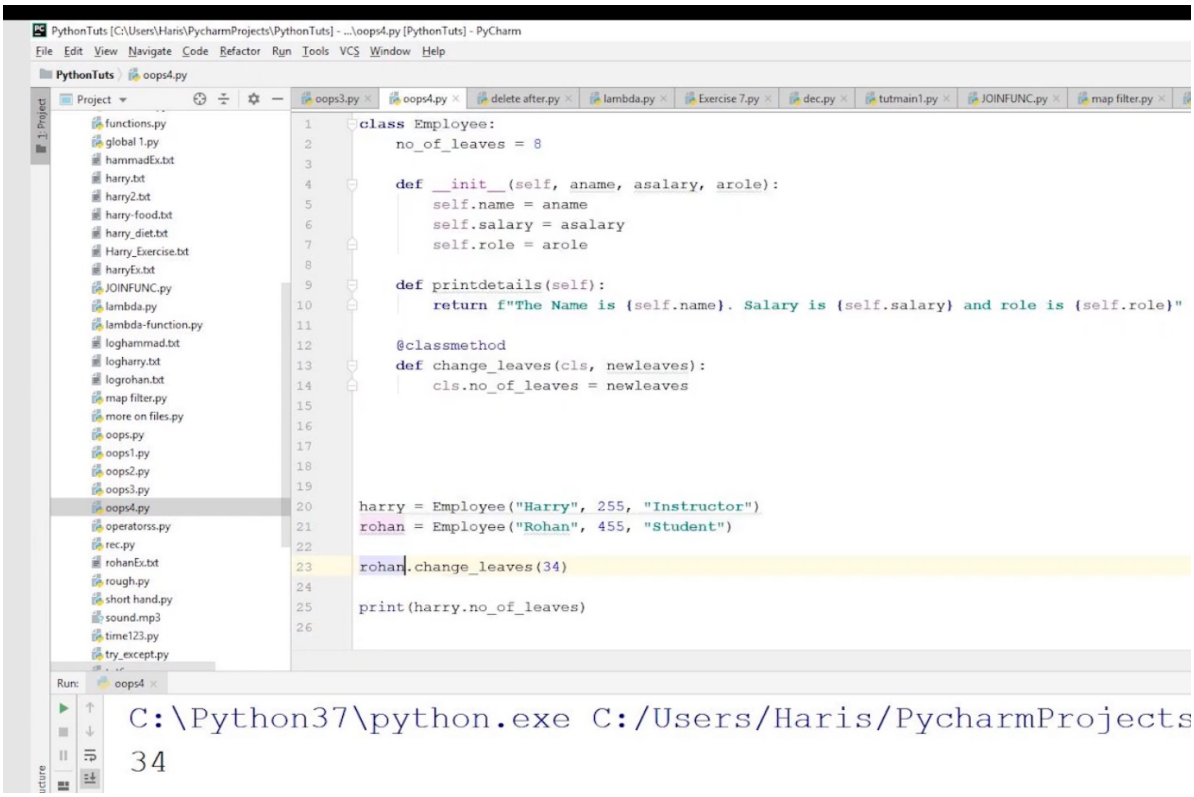


Figure 1: class method in Python

Static Method in Python:

We know that a static method in Python is treated differently than in other languages. Static methods in Python are extremely similar to python class methods, but the difference is that a static method is bound to a class rather than the objects for that class.
To define a static method, we use the **@staticmethod** decorator, which is a built-in decorator. Also, there is no need to import any module to use decorators. Using a static method in a class, we permit it to be accessed only by the class objects or inside the class.

There are few limitations related to static methods, such as:

- Unlike, class method, a static method cannot alter or change any variable value or state of the class.
- Static methods do not have any knowledge related to the class.

Static methods do not require any knowledge related to the class that they are built-in. They are only formed in a class so that only the class instances can access them. We can use a static method for simple functionality that is mostly not related to the class. For example, we want to add two numbers together, but we do not want our method to be used elsewhere, other than the

class or through its instances, so we will create a static method. It will not require any information related to class as it only requires the numbers it has to add, but it will still fulfill its purpose as it is like a personal method that only the class has access to.
Most of the functionalities of the static methods can be performed using a class method. However, we prefer the static method, where it could work to make our program more efficient and faster as we do not have to pass self as a parameter, so the program's efficiency increases.
In Python, static methods can be created using @staticmethod.

```
class Student:
@staticmethod
    def myfunc():
        //Code to be executed
```

Alternatively, we can follow the below syntax as well:

```
staticmethod(class_name.method())
Using @staticmethod annotation is actually a better way to create a static
method in Python as the intention of keeping the method static is clear.
```

Advantages of Python static method

Static methods have a very clear use case. When we need some functionality not for an Object but with the complete class, we make a method static. This is advantageous when we need to create utility methods.
Finally, note that we do not need the self or cls to be passed as the first argument in a static method.

Code as described/written in the video

```
class Employee:
    no_of_leaves = 8

    def __init__(self, aname, asalary, arole):
        self.name = aname
        self.salary = asalary
        self.role = arole

    def printdetails(self):
        return f"The Name is {self.name}. Salary is {self.salary} and role
is {self.role}"

    @classmethod
    def change_leaves(cls, newleaves):
        cls.no_of_leaves = newleaves

    @classmethod
    def from_dash(cls, string):
        return cls(*string.split("-"))

    @staticmethod
    def printgood(string):
        print("This is good " + string)

harry = Employee("Harry", 255, "Instructor")
rohan = Employee("Rohan", 455, "Student")
```

```
karan = Employee.from_dash("Karan-480-Student")

Employee.printgood("Rohan")
```

Abstraction & Encapsulation

Our today's tutorial is based mainly on theory because coding is not just about learning new concepts but also about understanding them. Let us give our ***PyCharm*** some rest and try to learn some theory about Abstraction and Encapsulation, which are fundamental concepts for our tutorials ahead.

What is Abstraction?

Abstraction refers to hiding unnecessary details to focus on the whole product instead of parts of the project separately. It is a mechanism that represents the important features without including implementation details. Abstraction helps us in partitioning the program into many independent concepts so we may hide the irrelevant information in the code. It offers the greatest flexibility when using abstract data-type objects in different situations.

Example of Abstraction:

Let us take the example of a car. It has an engine, tires, windows, steering wheel, etc. All these things combine to form a car, which is an abstraction, and all the different parts are its layers of abstraction. Now an engine is composed of various parts such as camshaft, valves, oil pan, etc. these flayers the engine is an abstraction. In simple words, abstraction can be achieved by hiding the background details and showing only the necessary ones. In programming, abstraction can not be achieved without Encapsulation.

What is Encapsulation?

Encapsulation means hiding under layers. When working with classes and handling sensitive data, global access to all the variables used in the program is not secure. In Encapsulation, the internal representation of an object is generally hidden from the outside to secure the data. It improves the maintainability of an application and helps the developers to organize the code better.

Example of Encapsulation

We can take an example of a capsule in which the medicine is encapsulated. We have often used examples of bigger projects in which many programmers contribute according to their tasks. In the end, the whole project is done by joining the contribution of each participant. Well, this is what Encapsulation aims to achieve.

Abstraction and Encapsulation are fundamental concepts of OOP. Encapsulation takes all the worry away from the user, providing him with just the product that he requires, irrespective of the way it is formed. Abstraction focuses on the working of the object instead of the how part, while Encapsulation is all about hiding the way or method of working and just providing the working model.

Classes can be a perfect example of abstraction as each team member is given a separate class to work on, to develop a more significant project. A person working in a class only knows his job. While Encapsulation can be said to hide the code from normal users by making a front end through which the user can interact through the software without direct access to the code.

Abstraction	Encapsulation
Abstraction is used to solve the problem and issues that arise at the design stage.	Encapsulation is used to solve the problem and issue that arise at the implementation stage.
Abstraction focuses on what the object does instead of how the details are implemented.	Encapsulation focuses on hiding the code and data into a single unit to secure the data from the outside world.
Abstraction can be implemented by using Interface and Abstract Class.	Encapsulation can be implemented using Access Modifiers (Public, Protected, and Private.)
Its application is during the design level.	Its application is during the Implementation level.

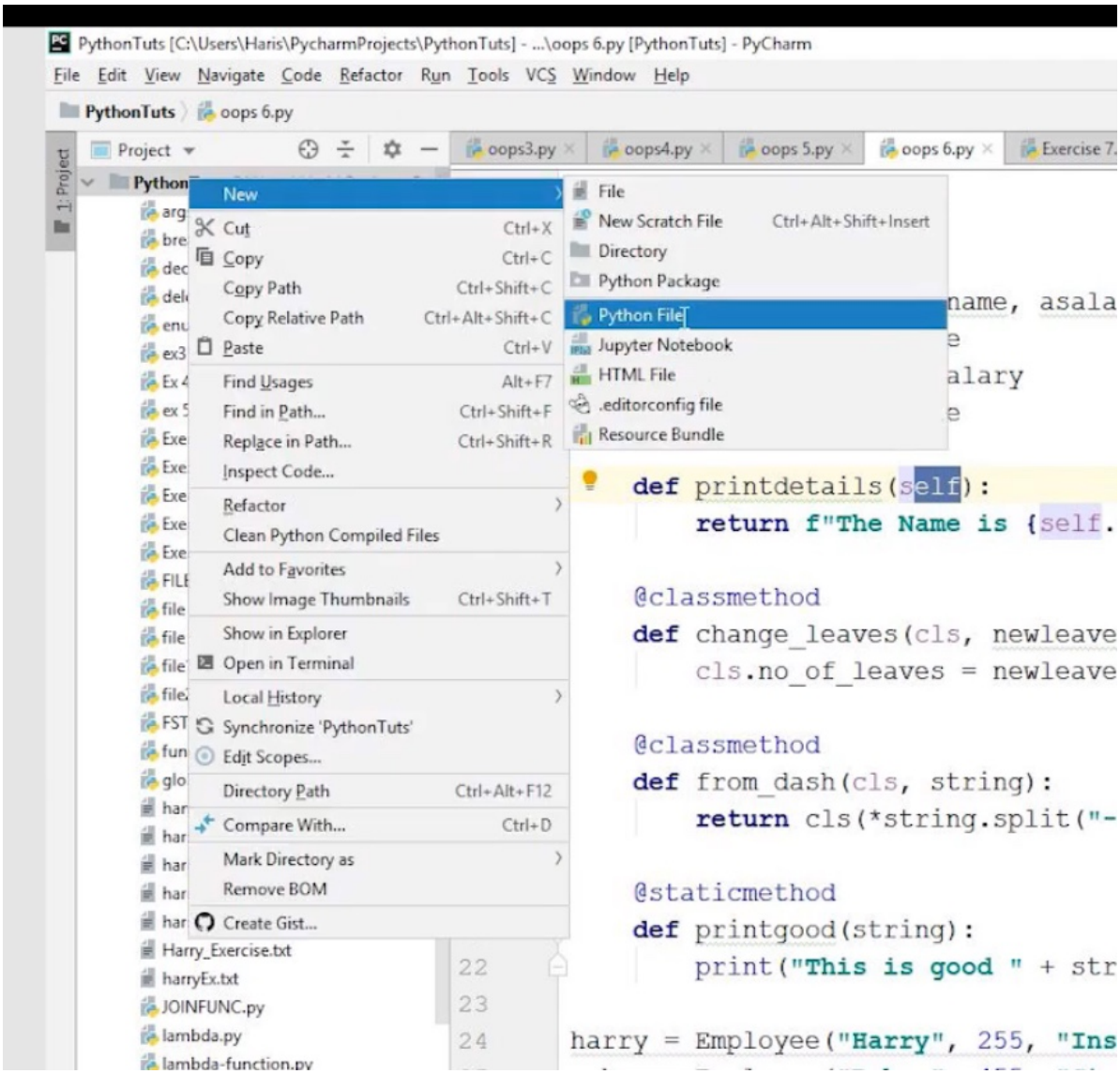
This tutorial is all about understanding the concept of Abstraction and Encapsulation. Check the next tutorials to learn how to implement more OOP concepts in Python.

Code as described/written in the video

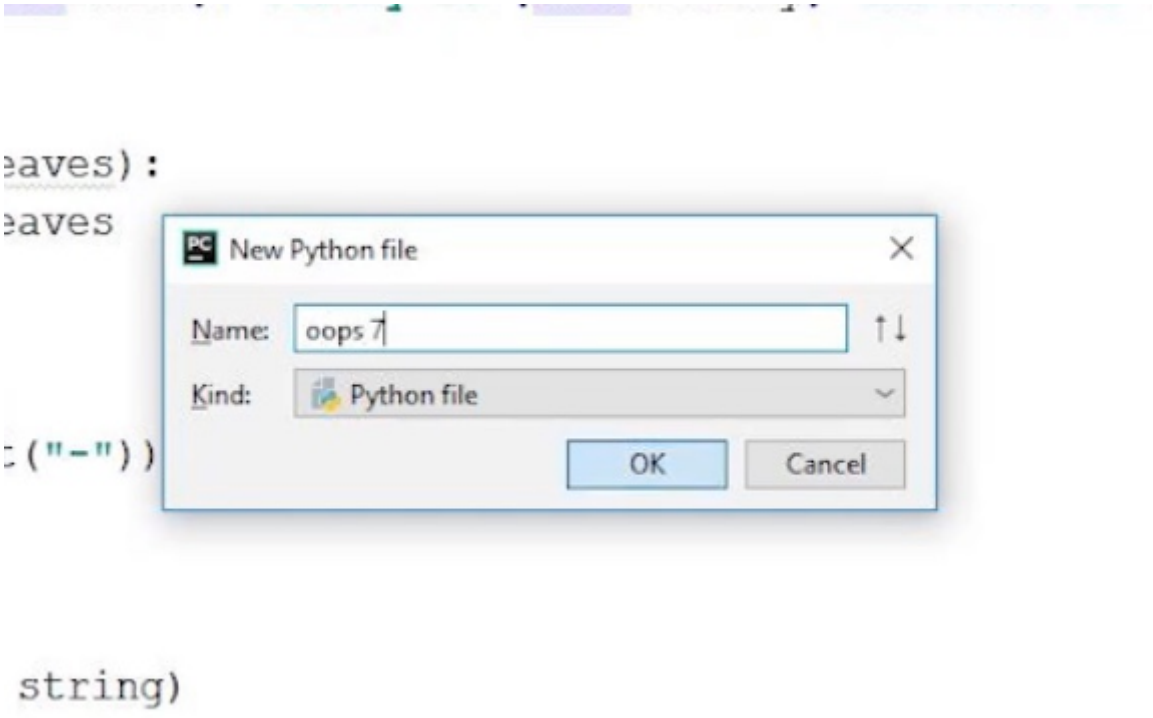
```
class Employee:
    no_of_leaves = 8

    def __init__(self, aname, asalary, arole):
        self.name = aname
        self.salary = asalary
        self.role = arole

    def printdetails(self):
        return f"The Name is {self.name}. Salary is {self.salary} and role is {self.role}"

    @classmethod
    def change_leaves(cls, newleaves):
        cls.no_of_leaves = newleaves

    @classmethod
    def from_dash(cls, string):
        return cls(*string.split("-"))

    @staticmethod
    def printgood(string):
        print("This is good " + string)

harry = Employee("Harry", 255, "Instructor")
rohan = Employee("Rohan", 455, "Student")
karan = Employee.from_dash("Karan-480-Student")

Employee.printgood("Rohan")
```

Single Inheritance

Let's just begin this tutorial by opening our PyCharm and creating a new file.



We are going to name our file **oops7.py** here.



In today's tutorial, we are going to learn about Single inheritance in Python. Before discussing the concept of inheritance in Python, we should have some knowledge about the word itself. Inheritance means to receive something from one's parents or ancestors. The concept of inheritance is very similar in cases of classes where a class inherits all the properties and methods of its previous class that it is inheriting from.

"Inheritance is the ability to define a new class(child class) that is a modified version of an existing class(parent class)"

Syntax:

```
class Parent_class_Name:
#Parent_class code block
class Child_class_Name(Parent_class_name):
#Child_class code block
```

The above syntax consists of two classes declared. One is the parent class or by other means, the base class, and the other one is the child class which acts as the derived class.

Two common terms related to inheritance are as follows:

- Parent: The parent class is the one that is giving access to its methods or properties to the child class or derived class.
- Child: Child class is the one that is inheriting methods and properties from the parent class.

The class that is inheriting, i.e., the child class, can inherit all the functionality of the parent class and add its functionalities also. As we have already discussed that each class can have its constructors and methods, so in case of inheritance the child class can make and use its constructor and also can use the constructor of the parent class. We can simply construct it as we did for the parent class but OOP has provided us with a simple and more useful solution known as Super().

We will be discussing super() and overriding in our ***Super() and Overriding In Classes*** tutorial of the course.

Single inheritance exists when a class is only derived from a single base class. Or in other words when a child class is using the methods and properties of only a single parent class then single inheritance exists. Single inheritance and Multiple inheritance are very similar concepts, the only major difference is the number of classes. We will see ***Multiple Inheritance*** in our next tutorial.

Different forms of Inheritance:

1. **Single inheritance:** When a child class inherits from only one parent class then it is called single inheritance.
2. **Multiple inheritance:** When a child class inherits from multiple parent classes then it is called multiple inheritance

Below is an example of single inheritance in Python.

```
class Parent():
    def first(self):
        print('Parent function')

class Child(Parent):
    def second(self):
        print('Child function')

object1 = Child()
object1.first()
object1.second()
```

Output:

```
Parent function
Child function
```

Advantages of Inheritance:

- It increases the code's reusability as we do not have to copy the same code again to our new class.
- It makes the program more efficient.
- We can add more features to our already built class without modifying it or changing its functionality.
- It provides a representation of real-world relationships.

In this tutorial, we have discussed the Inheritance concept. Inheritance is among the most significant concepts in object-oriented programming technique which provides code reusability, readability and helps in building optimized and efficient code.

Code as described/written in the video

```
class Employee:
    no_of_leaves = 8

    def __init__(self, aname, asalary, arole):
        self.name = aname
        self.salary = asalary
        self.role = arole

    def printdetails(self):
```

```
        return f"The Name is {self.name}. Salary is {self.salary} and role is {self.role}"

    @classmethod
    def change_leaves(cls, newleaves):
        cls.no_of_leaves = newleaves

    @classmethod
    def from_dash(cls, string):
        return cls(*string.split("-"))

    @staticmethod
    def printgood(string):
        print("This is good " + string)


class Programmer(Employee):
    no_of_holiday = 56
    def __init__(self, aname, asalary, arole, languages):
        self.name = aname
        self.salary = asalary
        self.role = arole
        self.languages = languages


    def printprog(self):
        return f"The Programmer's Name is {self.name}. Salary is {self.salary} and role is {self.role}.The languages are {self.languages}"



harry = Employee("Harry", 255, "Instructor")
rohan = Employee("Rohan", 455, "Student")

shubham = Programmer("Shubham", 555, "Programmer", ["python"])
karan = Programmer("Karan", 777, "Programmer", ["python", "Cpp"])
print(karan.no_of_holiday)
```

Multiple Inheritance

In the previous Tutorial i.e., ***tutorial#60***, we learned the concept of inheritance and did some examples related to single inheritance. Single inheritance exists when a class is only derived from a single base class. Or in other words, when a child class uses the methods and properties of only a single parent class, then single inheritance is exhibited.

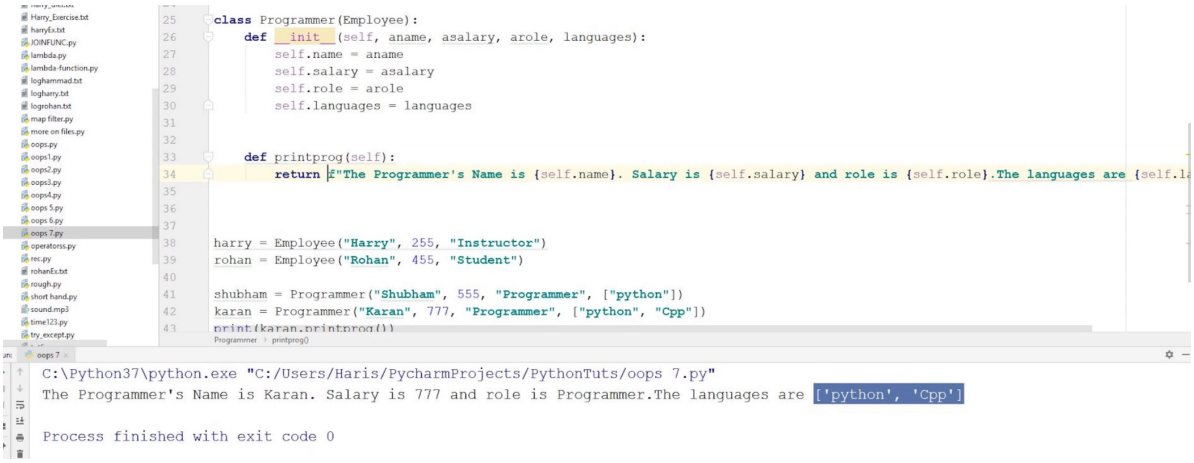


Figure1: Single Inheritance

In this tutorial, we are going to learn about Multiple inheritance. Let us start with a definition:

"In multiple inheritance, a class is derived from more than one class i.e. multiple base classes. The child class, in this case, has features of both the parent classes."

As the name implies, python's multiple inheritance is when a class inherits from more than one class. This concept is very similar to multilevel inheritance, which also is our next topic of this course. It is also nearly the same as a single-level inheritance because it contains all of the same functionality, except for the number of base classes.

While using the concept of multiple inheritance, the order of placing the base classes is very important. Let us clear the concept using an example. Suppose we have a child class named Child, and it has two base classes, named Base1 and Base2.

Example:

```
class Base1:
      def func1(self):
            print("this is Base1 class")
class Base2:
      def func2(self):
            print("this is Base2 class")

class Child(Base1 , Base2):
      def func3(self):
            print("this is Base3 class")

obj = Child()
obj.func1()
obj.func2()
obj.func3()
```

Output:

this is Base1 class

this is Base2 class

this is Base3 class

Now, when we are looking for some attribute, let it be a constructor. Then the program will search the current class i.e., the Child1 class first. If it does not find it in the Child1, it will look in the base class that is present at the leftmost side, which is Base1. After that, the program will start moving from left to right in a sequential manner, hence searching the Base2 class at the end. We should always give attention to the ordering of the base classes because it helps us a lot when multiple classes contain the same methods and also in method overriding.

Method Overriding:

Override means having two methods that have the same name. They may perform same tasks or different tasks. In python, when the same method defined in the parent class is also defined in the child class, the process is known as Method overriding. This is also true when multiple classes have the same method and are linked together somehow.

There are few rules for Method overriding that should be followed:

- The name of the child method should be the same as parents.
- **Inheritance** should be there, and we need to derive a child class from a parent class
- Both of their parameters should be the same.

In this case, the child method will run, and the reason for which, we have discussed in the paragraph above, related to ordering. Multiple inheritance is based on the same concept on which the single inheritance is based on i.e., DRY (do not repeat yourself). **Multiple inheritance** makes it easier to inherit methods and attributes from base classes that implement the functionality. When done right, we can reuse the code without having to copy-and-paste a similar code to implement interfaces.

Code as described/written in the video

```
class Employee:
    no_of_leaves = 8
    var = 8

    def __init__(self, aname, asalary, arole):
        self.name = aname
        self.salary = asalary
        self.role = arole

    def printdetails(self):
        return f"The Name is {self.name}. Salary is {self.salary} and role
is {self.role}"

    @classmethod
    def change_leaves(cls, newleaves):
        cls.no_of_leaves = newleaves

    @classmethod
    def from_dash(cls, string):
```

```
        return cls(*string.split("-"))

    @staticmethod
    def printgood(string):
        print("This is good " + string)

class Player:
    var = 9
    no_of_games = 4
    def __init__(self, name, game):
        self.name = name
        self.game =game

    def printdetails(self):
        return f"The Name is {self.name}. Game is {self.game}"

class CoolProgramer(Player, Employee):

    language = "C++"
    def printlanguage(self):
        print(self.language)

harry = Employee("Harry", 255, "Instructor")
rohan = Employee("Rohan", 455, "Student")

shubham = Player("Shubham", ["Cricket"])
karan = CoolProgramer("Karan",["Cricket"])
# det = karan.printdetails()
# karan.printlanguage()
# print(det)
print(karan.var)
```

Multilevel Inheritance

In ***Tutorial#60***, we learned inheritance, and in the previous tutorial of this course i.e., ***tutorial#61*** , we leaned multiple inheritance. These topics are very important and crucial for us to understand today's topic, which is Multilevel inheritance.

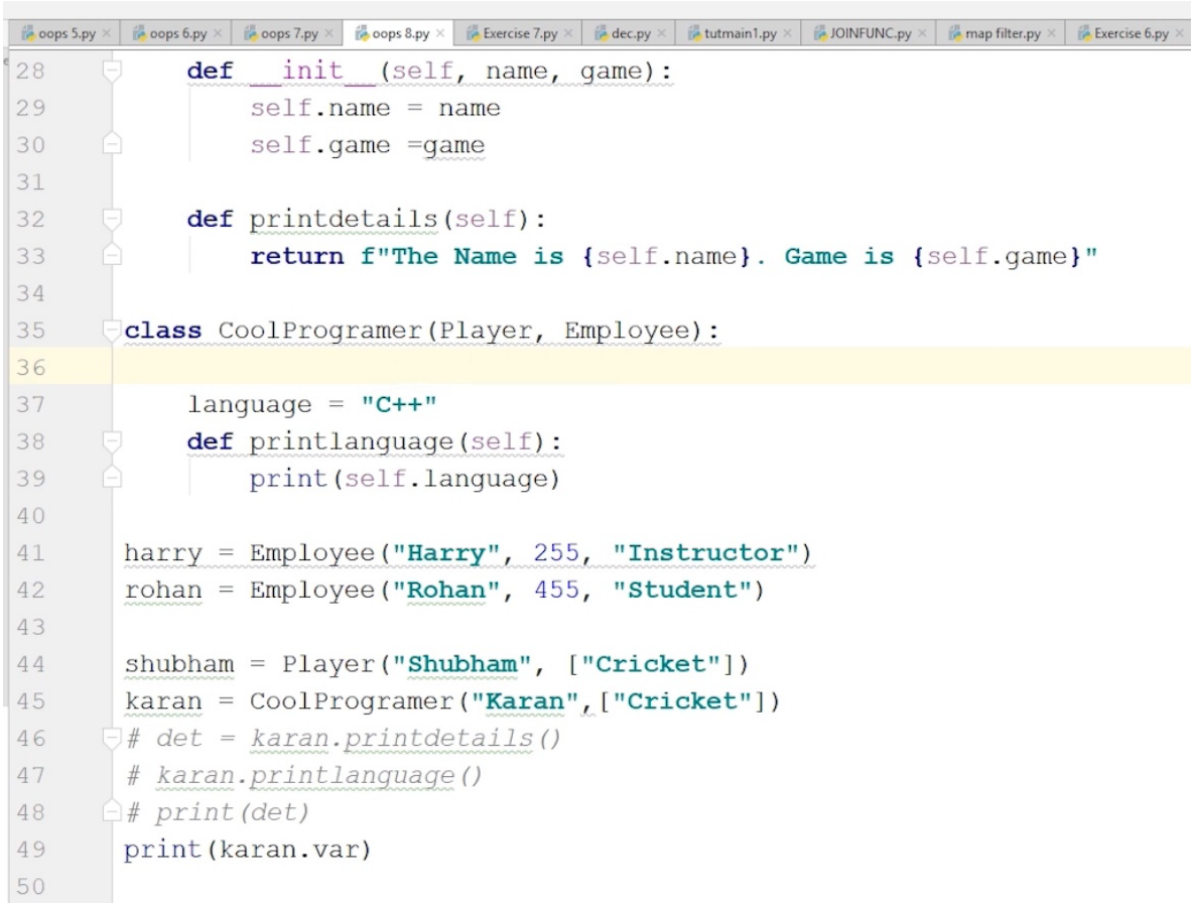


Figure 1: Multiple Inheritance

Multiple inheritance and Multilevel inheritance are very similar concepts. If you have a complete understanding of multiple inheritance then understanding, multilevel will take no time. The minimum number of classes for Multilevel inheritance is the same as for multiple inheritance i.e., three.

What is Multilevel Inheritance in Python?

In multilevel inheritance, a class that is already derived from another class is derived by a third class. So in this way, the third class has all the other two former classes' features and functionalities. The syntax looks something like this:

```
class Parent1:
    pass
class Derived1(Parent1):
    pass
class Derived2(Derived1):
    pass
```

Now let us dive into the priority brought by the ordering of the class. Suppose that we are looking for a constructor or a value for any variable. Our program will first check our current class i.e., Derived2, for it. If nothing is found, then it will move towards Derived1 and in order at last towards Parent1 until it finds the constructor or variable in the way.

If we have the same method or variable in the base and derived class, then the order we discussed above will be followed, and the method will be overridden. Else, if the child class does not contain the same method, then the derived1 class method will be followed by the sequence defined in the paragraph above.

For Example:

```
class level1:
      def first(self):
            print ('code')

class level2(level1): #inherit level1
      def second(self):
             print ('with')

class level3(level2): #inherit level2
      def third(self):
            print ('harry')

obj=level3()
obj.first()
obj.second()
obj.third()
```

Rules for Method overriding:-

There are few rules for Method overriding that should be followed:

- The name of the child method should be the same as parents.
- **Inheritance** should be there, and we need to derive a child class from a parent class
- Both of their parameters should be the same.

Multiple inheritance VS. Multilevel inheritance

Multiple inheritance	Multilevel inheritance
Multiple Inheritance is where a class inherits from more than one base class. Sometimes, multiple Inheritance makes the system more complex, that's why it is not widely used. Multiple Inheritance has two class levels; these are base class and derived class.	In multilevel inheritance, we inherit from a derived class, making that derived class a base class for a new class. Multilevel Inheritance is widely used. It is easier to handle code when using multilevel inheritance. Multilevel Inheritance has three class levels, which are base class, intermediate class, and derived class.

Advantages of Inheritance

1. It reduces code redundancy.
2. Multilevel inheritance provides code reusability.
3. Using multilevel inheritance, code is easy to manage, and it supports code extensibility by overriding the base class functionality within child classes

Code as described/written in the video

```
class Dad:
    basketball =6

class Son(Dad):
    dance =1
    basketball = 9
    def isdance(self):
        return f"Yes I dance {self.dance} no of times"

class Grandson(Son):
    dance =6
    guitar = 1

    def isdance(self):
        return f"Jackson yeah!" \
            f"Yes I dance very awesomely {self.dance} no of times"

darry = Dad()
larry = Son()
harry = Grandson()

# print(darry.guitar)

# electronic device
# pocket gadget
# phone
```

Public, Private & Protected Access Specifiers

In high-level programming languages like C++, Java, etc., private, protected, and public keywords are used to control the access of class members or variables. However, Python has no such keywords. Python uses a convention of prefixing the name of the variable or method with a single underscore(_) or double underscore(__) to emulate the behavior of protected and private access specifiers.

Access modifiers are used for the restrictions of access any other class has on the particular class and its variables and methods. In other words, access modifiers decide whether other classes can use the variables or functions of a specific class or not. The arrangement of private and protected access variables or methods ensures the principle of data encapsulation. In Python, there are three types of access modifiers.

- Public Access Modifier
- Protected Access Modifier
- Private Access Modifier

Public Access Modifier:

In public, all the functions, variables, methods can be used publicly. Meaning, every other class can access them easily without any restriction. Public members are generally methods declared in a class that is accessible from outside the class. Any ordinary class is, by default, a public class. So, all the classes we had made till now in the previous tutorials were all public by default.

Example of public access modifier:

```
class employee:
      def __init__(self, name, age):
            self.name=name
            self.age=age
```

Protected Access Modifier:

In the case of a protected class, its members and functions can only be accessed by the classes derived from it, i.e., its child class or classes. No other environment is permitted to access it. To declare the data members as protected, we use a single underscore “_” sign before the data members of the class.

Example of protected access modifier:

```
class employee:
      def __init__(self, name, age):
            self._name=name # protected attribute
            self._age=age # protected attribute
```

Private Access Modifier:

In the case of private access modifiers, the variables and functions can only be accessed within the class. The private restriction level is the highest for any class. To declare the data members as private, we use a double underscore “” sign before the data members of the class. Here is a suggestion not to try to access private variables from outside the class because it will result in an AttributeError.

Example of private access modifier:

```
class employee:
      def __init__(self, name, age):
            self.__name=name # private attribute
            self.__age=age # private attribute
```

Name mangling in Python:

Python does not have any strict rules when it comes to public, protected, or private, like java. So, to protect us from using the private attribute in any other class, Python does name mangling, which means that every member with a double underscore will be changed to *object.*class__variable when trying to call using an object. The purpose of this is to warn a user so he does not use any private class variable or function by mistake without realizing its states.

The use of single underscore and double underscore is just a way of name mangling because Python does not take the public, private and protected terms much seriously so we have to use our naming conventions by putting single or double underscore to let the fellow programmers know which class they can access or which they can't.

Code as described/written in the video

```
# Public -
# Protected -
# Private -

class Employee:
    no_of_leaves = 8
    var = 8
    _protec = 9
    __pr = 98

    def __init__(self, aname, asalary, arole):
        self.name = aname
        self.salary = asalary
        self.role = arole

    def printdetails(self):
        return f"The Name is {self.name}. Salary is {self.salary} and role
is {self.role}"

    @classmethod
    def change_leaves(cls, newleaves):
        cls.no_of_leaves = newleaves

    @classmethod
    def from_dash(cls, string):
        return cls(*string.split("-"))

    @staticmethod
    def printgood(string):
        print("This is good " + string)

emp = Employee("harry", 343, "Programmer")
print(emp._Employee__pr)
```

Polymorphism In Python

Today's tutorial is mainly based on theory, although we will see the implementation of Polymorphism in most of our next oop-based tutorials. To have a firm grasp of the theoretical concepts is very important. Polymorphism is more of a technique rather than a skill that is based on just syntax. Python is an object-oriented programming language. Previously we have studied some essential concepts of OOP, which include ***Inheritance*** and ***Abstraction & Encapsulation***. In this tutorial, we will learn about Polymorphism and how we can implement it in Python.

What Is Polymorphism?

In the basic English language, Polymorphism means to exist in different states. The same object or thing changing its state from one form to another is known as polymorphic. The same function or method, being used differently in different scenarios, can perfectly describe Polymorphism. It occurs mostly with base and derived classes.

Understanding Polymorphism in Python:

The concept of Polymorphism has very strong ties with the method overriding concept that we will learn in the next tutorial, i.e., ***tutorial#65*** of this course, along with the super() function. In Python, it is mostly related to objects or the values of variables that are assigned in different classes. For example, suppose a method in the child class has the same name as the methods in the parent class, and also they take the same number of variables as parameters. In that case, the child class will inherit the methods from the parent class and will override the method too. Meaning that the compiler will execute the method in the child because it will be the first place it looks while searching for the method when called. By overriding a method, we can also add some more functionalities in it, so in a way modifying the method in the child class but letting it remain the same in the parent class.

For example:

```
len("Python") # returns 6 as result
len([1,2,3,4,5,6,7,8,9]) # returns 9 as result
```

Python also implements Polymorphism using methods. The len() method returns the length of an object. In this case, the function len() is polymorphic as it is taking **a string** as input in the first case, which returns the total length/characters of the string, and in the second case, it is taking **list** as input.

Polymorphism is a very important concept. Although being a theoretical concept, it is of great importance as it teaches us to use one entity, let it be a method or variable, differently at different places. By applying the concept of Polymorphism, we can save our time and make our code more efficient and compact by using the DRY (Don't Repeat Yourself) concept, which is the basis of Oop.

We can apply the concept of Polymorphism to the methods, objects, functions, and inheritance. Even though the syntax and rules differ, but the concept remains the same.

Polymorphism in '+' operator:-

Take a look at the below example:

```
print(5+6)
print("5" + "6")
```

In the above example, we have used the '+' arithmetic python operator two times in our programs. This is an example of the implementation of Polymorphism in Python. We can use the same + operator to add two integers, concatenate two strings, or extend two lists. The + arithmetic operator acts differently depending on the type of objects it is operating upon.

Code as described/written in the video

```
print(5+6)
print("5" + "6")

# Abstraction
# Encapsulation
# Inheritance
# Polymorphism
```

Super() and Overriding In Classes | Python Tutorials For Absolute Beginners In Hindi #65

As you people might have noticed that we came across super and overriding keywords a lot of time in a few of our previous tutorials but did not get into its detailed working. The reason for that was to give you an idea of concepts leading to the use of **super() and overriding** first. In this tutorial, we are going to discuss it in detail, **using the single inheritance** as an example.

Yet being different concepts, they are often used together. The need for superclass arises when overriding is being done at a certain point in our code. Overriding is an essential part of object-oriented programming since it makes the inheritance utilize its full power. Overriding avoids duplication of code, and at the same time, enhances or customizes the part of it. It is a part of the inheritance mechanism. Let us get a description of overriding first so that we can dive into the concept of super() later.

How to override class methods in Python?

Overriding occurs when a derived class or child class has the same method that has already been defined in the base or parent class. When called, the same methods with the same name and number of parameters, the interpreter checks for the method first in a child class and runs it ignoring the method in the parent class because it is already overridden. In the case of instance variables, the case is a little different. When the method is called, the program will look for any instance variable having the same name as the one that is called in the child, then in the parent, and after that, it comes again into child class if not found.

Where does super() fit in all this?

When we want to call an already overridden method, then the use of the super function comes in. It is a built-in function, so no requirement of any module import statement. What super does is it allows us to use the method of our superclass, which in the case of inheritance is the parent class. Syntax of using super() is given below:

```
class Parent_Class(object):
      def __init__(self):
            pass

class Child_Class(Parent_Class):
     def __init__(self):
           super().__init__()
```

super() returns a temporary object of the superclass that then allows you to call that superclass's methods. The primary use case of super() is to extend the functionality of the inherited method.

We have discussed earlier that in the case of method overriding, the previous method could not be called, but super makes an exception, and thus we can partially or completely use the method of the parent class too. We can even use super() to call only a specific variable we used in our overridden method. Calling the superclass-built methods with super() saves us from rewriting those methods in our subclass and allows us to swap out superclasses with minimal code changes.

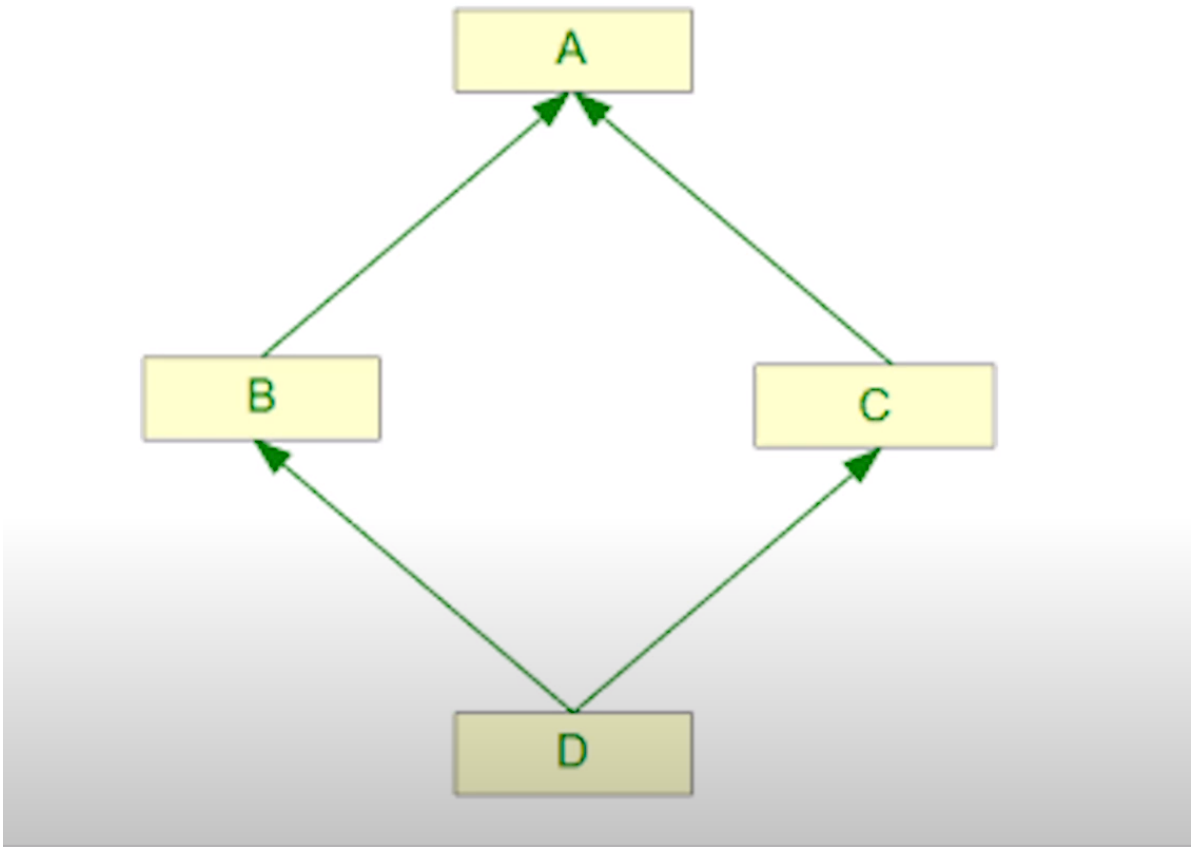
Code as described/written in the video:

```
class A:
    classvar1 = "I am a class variable in class A"
    def __init__(self):
        self.var1 = "I am inside class A's constructor"
        self.classvar1 = "Instance var in class A"
        self.special = "Special"

class B(A):
    classvar1 = "I am in class B"

    def __init__(self):
        self.var1 = "I am inside class B's constructor"
        self.classvar1 = "Instance var in class B"
        # super().__init__()
        # print(super().classvar1)


a = A()
b = B()

print(b.special, b.var1, b.classvar1)
```

Diamond Shape Problem In Multiple Inheritance

From the start of this course, you may have noticed that I am not just teaching you the syntax so that you may learn just the practical approach to programming and could create a few programs that have no real-world value. Instead, I am trying to teach you all the theoretical and practical concepts together so you may become a successful programmer. You can do the programming by just learning the syntax, but without proper conceptual knowledge, you won't be able to develop proper logic while writing code. Our today's tutorial is also based on a theoretical concept.

In previous tutorials, we have seen a lot of concepts related to Object-Oriented programming, such as ***Single inheritance***, ***Multiple Inheritance***, ***Multilevel Inheritance***, etc. Today we are going to discuss a problem or, more like a confusion associated with multiple inheritance. The problem is commonly known as the ***"Diamond Shape Problem."***. It is about priority related confusion, which arises when four classes are related to each other by an inheritance relationship, as shown in the image below:



Explanation:-

In the above image, we can see that class C and class B are inheriting from class A, or it can be said as class A is a parent to class B and C. And class D is inheriting from both class C and B. So, in a way, they are all in relation to one and other somehow. Let us write down the relation in code format so it will be easier to understand.

```
class A:
    pass
class B(A):
    pass
class C(A):
    pass
class D( B, C ):
    pass
```

As discussed earlier that it creates priority-related confusion, so let's clear that out here.

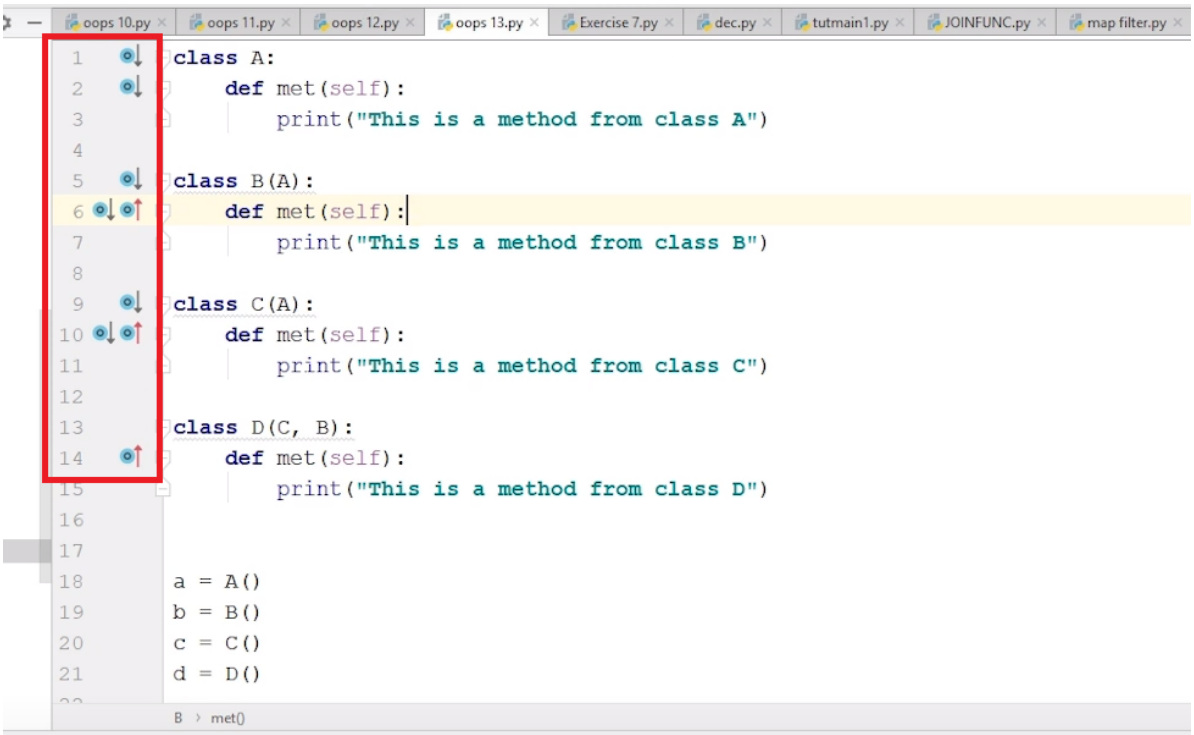
- If we have a method that is only present in class A and we use the class D object to call the method, it will go to class A while checking for the method name in all the classes in between and run the method in class A.
- However, if the same method is also present in class B, then it will run the B class method because, for class D, class B holds more priority than class A. The reason is that class D is derived from class B, which is further derived from A. So, a closer relation exists with B than A.
- If the same method is present in classes C and B, it may create a little bit of confusion. But as we have already discussed in Tutorial #61, that in such cases, our priority is based from left to right, meaning whichever class is on the left side will be given more priority, and then we will move towards the right one. In this case, the left class is B, so the method in B will de be executed first.

If the C class would be on the left, such as

```
class D( C,B ):
      pass
```

Then priority would be given to C.

Sometimes, when we are working with too many classes, the concept of multiple inheritance could make our code more confusing and difficult to understand, such as:



Talk is cheap. Show me the code.

You can see in the highlighted area that it is a little difficult to understand which method is overriding which one, so multiple inheritance is discouraged in such situations.

Code as described/written in the video

```
class A:
    def met(self):
        print("This is a method from class A")

class B(A):
    def met(self):
        print("This is a method from class B")

class C(A):
    def met(self):
        print("This is a method from class C")

class D(C, B):
    def met(self):
        print("This is a method from class D")


a = A()
b = B()
c = C()
d = D()

d.met()
```

Operator Overloading & Dunder Methods

Operator overloading and Dunder Methods may be new concepts for some of you. However, we have already seen similar concepts in which different methods act differently on different occasions and places. Let us understand the Operator overloading first.

Operator Overloading In Python

Operator overloading means giving new meanings to an operator. In simple words, it means to assign new functionality to an operator beyond its normal functioning. We will go with the most common and easiest that we could find related to the concept, i.e., the + sign. For numbers, it is used for addition between them, but in the case of a string, it is used to join or combine two strings, working differently in two different scenarios. The operators are methods defined in respective classes. Defining methods for operators is known as operator overloading.

Python Dunder Methods Or Special Functions

Dunder methods in Python are special methods. In Python, we sometimes see method names with a double underscore (**), such as the** init__ method that every class has. These methods are called **"dunder"** methods. In Python, Dunder methods are used for operator overloading and customizing some other function's behavior.

Python usually calls dunder methods under the hood. Suppose we want to join a string with a number using the + sign. Now joining between two different data types is not possible in Python, and the resultant in such a case will be an error. So for this purpose, we can use a function provided to us by Python, named as dunder function. We will write such code in it so that it may first convert the number to a string and then join them, or any other logic will be fine too until it does what we require. We can even return 85; regardless of what string or number is given to us, it is all up to us.

Methods starting with a double underscore (__) and ending with a double underscore (__) represent dunder methods.

Check https://docs.python.org/2/library/operator.html to explore more about operator overloading.

str and repr functions

Both of these built-in methods are used to return a presentable description of any object rather than the default one. The difference in them is the way of writing them. The **str** method is mainly written for the end-user, while **repr** is written for a developer. It is overridden to return a printable string representation of any user-defined class. An interesting thing to note here is that the priority of **str** is greater than **repr**. This means that if we pass an object into a print statement, it will return us the **str** string even if **repr** is also present there. In such cases, if we want to print **repr**, we have to call it exclusively with the object name in the print statement.

Difference between str and repr functions

1. If the implementation of **str** is missing, then **repr** function is used as a fallback. If the implementation of **repr** is missing, then there will be no fallback.
2. If **repr** function is returning the object's String representation, we can skip the implementation of **str** function.
3. The priority of **str** is higher than **repr**.

Code as described/written in the video

```
class Employee:
    no_of_leaves = 8

    def __init__(self, aname, asalary, arole):
        self.name = aname
        self.salary = asalary
        self.role = arole

    def printdetails(self):
        return f"The Name is {self.name}. Salary is {self.salary} and role is {self.role}"

    @classmethod
    def change_leaves(cls, newleaves):
        cls.no_of_leaves = newleaves

    def __add__(self, other):
        return self.salary + other.salary

    def __truediv__(self, other):
        return self.salary / other.salary

    def __repr__(self):
        return f"Employee('{self.name}', {self.salary}, '{self.role}')"

    def __str__(self):
        return f"The Name is {self.name}. Salary is {self.salary} and role is {self.role}"

emp1 =Employee("Harry", 345, "Programmer")
# emp2 =Employee("Rohan", 55, "Cleaner")
print(str(emp1))
```

Abstract Base Class & @abstractmethod

Today we are going to learn the concept of abstract class and abstract method. Let us begin our tutorial with some definitions. First, we will define the abstract class:

"An abstract class is a class that holds an abstract method."

And

"An abstract method is a method defined inside an abstract class."

The definition may appear too simple, but the abstract method holds all the cards because it is necessary for all classes derived by the abstract class to have the same method even though their functionality and code may differ. However, the name of the method should be the same as the abstract method. The abstract method inside the abstract class could even be empty because we can not implement it anywhere. It is just so that all the other classes define a method by the same name.

It is important to remember that we can not make an object for an abstract class.

Following is the syntax for defining an abstract method in an abstract class in Python:

```
from abc import ABC, abstractmethod
Class MyClass(ABC):
      @abstractmethod
      def mymethod(self):
            #empty body
            pass
```

To implement an abstract class, we have to import the abc module by using an import statement. Along with it, we have to import the abstract method too. **If we are using a Python version older than 3.4, then we have to write:**

```
from abc import ABCMeta, abstractmethod
```

Moreover, we have to pass ABCMeta into the class, which we are defining as abstract.

Although if our version is newer, then we can import by the statement:

```
from abc import ABC, abstractmethod
```

And we have to pass ABC to the class we are defining as abstract.

To make a method specifically abstract in a class, we use the abstract method decorator denoted as **@absttractmethod**, which is defined in the abc module.

To understand the abstract class and method better, let us take an example. Suppose we have a few classes related to different items in a bookstore. Now our classes are books, stationery, and magazines. All these products are dealt with separately, so all of these classes must have a method named sale that could return the amount achieved by selling these items. So to ensure that each of these classes has a sale method, we will make an abstract class named bookstore with an abstract method Sale and will derive all the other three classes from it. Until we have made a method Sale in each of the derived classes, we would not be able to make its object, or the compiler will report an error.

Important points about abstract class in Python:

1. Abstract methods are defined in the abstract class. They mostly do not have the body, but it is possible to implement abstract methods in the abstract class. Any subclass deriving from such an abstract class still needs to provide an implementation for that abstract method.
2. An abstract class can have both abstract methods as well as concrete methods.
3. The abstract class works as a template for other classes.
4. Using the abstract class, we can define a structure without properly implementing every method.
5. It is not possible to create objects of an abstract class because Abstract class cannot be instantiated.
6. An error will occur if the abstract method has not been implemented in the derived class.

Code as described/written in the video

```python
# from abc import ABCMeta, abstractmethod
from abc import ABC, abstractmethod

class Shape(ABC):
    @abstractmethod
    def printarea(self):
        return 0

class Rectangle(Shape):
    type = "Rectangle"
    sides = 4
    def __init__(self):
        self.length = 6
        self.breadth = 7

    def printarea(self):
        return self.length * self.breadth

rect1 = Rectangle()
print(rect1.printarea())
```

Setters & Property Decorators

In today's tutorial, we are going over four main topics, i.e., Getter, Setter, Deleter, and Property decorator. Let us start with the property decorator. Before discussing property decorators, we must have an understanding of decorators themselves. ***Decorators*** are functions that take another function as an argument, and their purpose is to modify the other function without changing it.

Note: For more information, visit Tutorial #51, which is solely based on decorators.

A ***property decorator*** is a built-in function in Python. Property decorator is a pythonic way to use getters and setters in object-oriented programming, which comes from the Python property class. Theoretically speaking, Python property decorator is composed of four things, i.e., getter, setter, deleted, and Doc. The first three are methods, and the fourth one is a docstring or comment.

```
@property
#def getter method
```

Use @property along with the getter method to access the value of the attribute.

A property decorator is used for setting the parameters. In OOP, the setter is an important part of the program as we can easily get the values passed in parameters. Without a setter, it is impossible to update the values passed as parameters during object creation. Setters are usually used in Oop to set the value of private attributes in a class.

Setters are a great way of performing encapsulation that we discussed in Tutorial #59 of our Python Tutorials for beginners course. So by using setter, our interaction gets limited to the decorator, making the use of encapsulation concept, which is the basis of Oop. We can set new values for a newer object or update values for an older one.

```
@function_name.setter
#def function
```

@function_name.setter is a setter method with which we can set the value of the attribute

Deleter is used to delete the values passed as a parameter before. We can use a setter if we want to update or change the value, but we can not use it to delete the value. This is where deleter comes in; it removes the previous value and sets the variable equal to none. As in OOP, we do not completely erase the existence of some variable but sets it equal to none.

```
 # Deleter method
@function_name.deleter
```

***@function_name.deleter* is a deleter method which can delete the assigned value by the setter method**

Advantages of @property in Python:

Following are some advantages of using @property in Python:

- The syntax of defining @property is very concise and readable.
- We can access instance attributes while using the getters and setter to validate new values. This will avoid accessing or modifying the data directly.

- By using @property, we can reuse the name of a property. This will prevent us from creating new names for the getters, setters, and deleters.

Summary:

In this tutorial, we have learned about property decorators, setters, getters, and deleters. Property decorator(@property) is a pythonic way of defining getters, setters, and deleters. Properties defined with the @property syntax is more compact and readable. When we define the properties, we can change the class's internal implementation without changing the program.

Code as described/written in the video

```
class Employee:
    def __init__(self, fname, lname):
        self.fname = fname
        self.lname = lname
        # self.email = f"{fname}.{lname}@codewithharry.com"

    def explain(self):
        return f"This employee is {self.fname} {self.lname}"

    @property
    def email(self):
        if self.fname==None or self.lname == None:
            return "Email is not set. Please set it using setter"
        return f"{self.fname}.{self.lname}@codewithharry.com"

    @email.setter
    def email(self, string):
        print("Setting now...")
        names = string.split("@")[0]
        self.fname = names.split(".")[0]
        self.lname = names.split(".")[1]

    @email.deleter
    def email(self):
        self.fname = None
        self.lname = None


hindustani_supporter = Employee("Hindustani", "Supporter")
# nikhil_raj_pandey = Employee("Nikhil", "Raj")

print(hindustani_supporter.email)

hindustani_supporter.fname = "US"

print(hindustani_supporter.email)
hindustani_supporter.email = "this.that@codewithharry.com"
print(hindustani_supporter.fname)

del hindustani_supporter.email
print(hindustani_supporter.email)
hindustani_supporter.email = "Harry.Perry@codewithharry.com"
print(hindustani_supporter.email)
```

Object Introspection

In today's tutorial, we are going to learn about object introspection. We have used it a bit in our previous tutorial but never discussed it in depth. As we have discussed earlier that everything in Python is an object. All the functions we use regularly are predefined in some built-in class. For example, while printing any string, we are using the object of an str class that is predefined for the usage of string.

Object Introspection in Python:

Introspection can be said as the ability to recognize the object along with all its details, such as id or location at runtime. One of the most basic introspects we came across many times earlier is **type()**.

```
type(object)
```

We used it to see the type of our object, that whether it is int, float, or string. We have to pass the object in the parenthesis, and the compiler will return the type. Introspection gives us useful information about the program's objects. Python provides tremendous introspection support. Introspection is the ability to determine the type of an object at runtime. Hence, by using introspection, we can inspect the Python objects dynamically.

There are many types of introspections. In this tutorial, we will focus on three of them to get a brief idea about their working. You may search the internet for more, but we will be focusing on three for conceptual learning. We have already discussed type( ), now let's move onto **id( )**. Id provides us with the id allocated to the particular object. The id of each object is unique, meaning it is different, and no two objects can have the same id.

```
id(object)
```

Now the most important introspection function is dir(). It returns us a list of attributes and methods associated with an object. By using **dir()**, we can check the attributes that our object is composed of. It is mostly executed before and after updating our object by inserting more attributes or methods.

```
o = MyClass()
print(dir(o))
```

Types of introspects:-

Some of the other common Introspects:

Functions	Working
hasattr()	It checks if an object has an attribute.
getattr()	It returns the contents of an attribute if there are some.
repr()	It returns the string representation of an object
vars()	It checks all the instance variables of an object
issubclass()	This function checks that if a specific class is a derived class of another class.
isinstance()	It checks if an object is an instance of a specific class.
doc	This attribute gives some documentation about an object
name	This attribute holds the name of the object
callable()	This function checks if an object is a function
help()	It checks what other functions do

So, this was all about object introspection in Python. I hope you are enjoying this course. If yes, then please share this course with your friends and family also. I will see you in the next tutorial. Till then, keep coding and keep learning.

Code as described/written in the video:

```
class Employee:
    def __init__(self, fname, lname):
        self.fname = fname
        self.lname = lname
        # self.email = f"{fname}.{lname}@codewithharry.com"

    def explain(self):
        return f"This employee is {self.fname} {self.lname}"

    @property
    def email(self):
        if self.fname==None or self.lname == None:
            return "Email is not set. Please set it using setter"
        return f"{self.fname}.{self.lname}@codewithharry.com"

    @email.setter
    def email(self, string):
        print("Setting now...")
        names = string.split("@")[0]
        self.fname = names.split(".")[0]
        self.lname = names.split(".")[1]

    @email.deleter
    def email(self):
        self.fname = None
        self.lname = None


skillf = Employee("Skill", "F")
# print(skillf.email)
```

```
o = "this is a string"
# print(dir(skillf))
# print(id("that that"))

import inspect
print(inspect.getmembers(skillf))
```

Advanced Python

Generators In Python

In this tutorial, we are going to learn about ***generators in Python***. Before starting our main topic, we should know about **iterables, iterator, and iterations**. Although we have covered these concepts before in our previous tutorials many times, for the sake of revision, let's go through them one more time by getting a brief introduction about these three concepts before moving to a deep understanding of generators.

Iterables are objects that can be placed inside a loop and can return one variable at a time. In simple terms, we can say that iterables are objects capable of iteration. Examples of iterable include list, string, tuple, etc.

```
for c in a:
      print (a)
#Here a is an iterable.
```

Now, moving on to iterator. An ***iteration*** can be defined as an object that does iterations on iterable. Meaning that it will move from character to character doing iteration. Every iterable, either it is a string or tuple, has a built-in method **iter()** that creates an object when called. The object moves from character to character of iterable using the **next()** method. The **next**() method is what's really behind the working of the loop.

The phenomenon that occurs by the combination of the two concepts defined above is known as iteration. We can define iteration as the repetition of the same commands again and again. Now, moving on towards our main topic, i.e., Generators.

What are the Generators in Python?

Generators concept is also very similar as it is used to make an iterator. The only difference comes in the return statement. The generator does not use a return statement. Instead, it uses a yield keyword. Yield functionality is very similar to return as it returns a value to the caller, but the difference is that it also saves the state of the iterator. Meaning that when we use the function again, the yield will resume the value from the place it left off.

Generators in Python are created just like the normal functions using the **'def'** keyword. Generator functions do not run by their name, and they are run when the **next**() function is called. A generator is very helpful in projects relating to memory issues because, like a simple iterator, it does not return all the values at a time; instead, it produces, series of values over time. So a generator is generally used when we want to iterate over a series of values but do not want to store them completely in memory.

For Example:

```
def getNum (x):
    for i in range(x):
        yield i

seq = getNum (2)
print(seq.__next__())
print(seq.__next__())
print(seq.__next__())
```

Output:

```
0
1
Traceback (most recent call last):
File "<stdin>", line 7, in <module>
 print(seq.__next__())
StopIteration
```

When we run print(seq.**next**()) for the third time, StopIteration is raised. This is because a for loop takes an iterator and iterates over it using **next**() function, which automatically ends when StopIteration is raised.

Advantages of using Generators:

- Producing iterables is extremely difficult and lengthy without Generators in Python.
- Generators automatically implement **iter**(), **next**(), and StopIteration which otherwise, need to be explicitly specified.
- The most significant advantage of generators is that the memory is saved as the items are produced when required.
- Generators are also used to pipeline a series of operations, for example, Generate Fibonacci Series.

Code as described/written in the video:

```
"""
Iterable - __iter__() or __getitem__()
Iterator - __next__()
Iteration -

"""

def gen(n):
    for i in range(n):
        yield i

g = gen(3)
# print(g.__next__())
# print(g.__next__())
# print(g.__next__())
# print(g.__next__())


# for i in g:
#     print(i)

h = 546546
ier = iter(h)
print(ier.__next__())
print(ier.__next__())
print(ier.__next__())
# for c in h:
#     print(c)
```

Python Comprehensions

Comprehensions in Python can be defined as the Pythonic way of writing code. Using comprehension, we compress the code so it takes less space. Comprehension in Python converts the four to five lines of code into a one-liner. In this tutorial, we will see the ways to write the code we used to write earlier in four to five lines, in just one line.

Comprehension's importance comes in the scenarios when the project is too big, for example, Google is made up of 2 billion lines of code, Facebook is made from 62 million lines of code, Windows 10 has roughly 50 million lines of code. So in such scenarios, comprehension has to be implemented as much as we can so that the lines of code decreases and the efficiency increases.

In Python 2.0, we only had the option of list comprehensions, but now after 3.0 version, we can also apply comprehension on dictionary and sets. We will discuss all three of them in this tutorial along with its implementation on generators. We have already discussed all of these topics before in our previous tutorials. I will provide you with the links of them so that if you have somehow missed any of those, you may watch them or give the description a read.

- Sets – ***Tutorial #12***
- List – ***Tutorial #9***
- Dictionary – ***Tutorial #10***
- Generators – ***Tutorial #72***

We will now learn ways to create a comprehensive way of implementing the functions. The concept is the same; the only difference comes in writing them i.e. in a more compact form. To understand the working, you can refer to the tutorial links given above as our focus will be more towards syntax in this part.

List as ordinarily are written as such:

```
listA = []
for a in range(50):
    if a%5==0:
        listA.append(a)
```

While it can be written in a one liner format using comprehension as such:

```
listA = [a for a in range(50) if i%5==0]
```

The compressed code works exactly like the one above but with more precision.

Set comprehension works exactly the same way as List comprehension. The syntax is almost the same two, except for the brackets i.e. set uses curly brackets. The main difference arrives while printing the items as a set will only print the same items once.

```
alpha = {alpha for alpha in ["a", "a", "b", "c", "d", "d"]}
```

The output will be: {'a', 'b', 'c', 'd'}

In the case of the dictionary, it has more benefits as we can alter the sequence of the dictionary by printing the values before the keys. We can also write conditional statements, that in the case of dictionary consumes 8-9 lines in just a single one. The syntax for a dictionary using ordinary syntax is:

```
Normaldict = {
  0 : "item0",
  1 : "item1",
  2 : "item2",
  3 : "item3",
  4 : "item4",
}
```

And the more compact one is:

```
Compdict = {i:f"Item {i}" for i in range(5)}
```

We can implement comprehension on generators too. We discussed Generators in the previous tutorial, i.e. Tutorial #72. We will again see its syntax here:

```
def gener(n):
    for i in range(n):
        yield i

a = gener(5)
print(a.__next__())
```

We can also create a one liner for generators too by following the syntax below.

```
gener = (i for i in range(n))
a = gener(5)
print(a.__next__())
```

Code as described/written in the video

```
# ls = []
# for i in range(100):
#     if i%3==0:
#         ls.append(i)

# ls = [i for i in range(100) if i%3==0]
#
# print(ls)


# dict1 = {i:f"item {i}" for i in range(1, 10001) if i%100==0}
# dict1 = {i:f"Item {i}" for i in range(5)}
#
# dict2 = {value:key for key,value in dict1.items()}
# print(dict1,"\n", dict2)

# dresses = [dress for dress in ["dress1", "dress2","dress1",
#                                "dress2","dress1", "dress2"]]
# print(type(dresses))

evens = (i for i in range(100) if i%2==0)
# print(evens.__next__())

# for item in evens:
```

```
25 #     print(item)
```

Using Else With For Loops

In this tutorial, we are going to see the implementation of for loop with an else statement. We will divide the tutorial into a few sections, including how, why, and when to use it for else statement. So, let us start with the basics. In most programming languages, an else statement is directly linked to an if statement and cannot be used separately. The use of "if" is a must for else in them. As we know that Python is more flexible in every aspect than the other ones, so in Python, we can use else without an if. Python allows us to implement the else functionality with for loops.

How can we implement for-else in Python?

We have to write an else statement using the Else keyword, followed by a colon after the for loop block of code.

Syntax:

```
for x in n:
   #statements
 else:
   #statements
```

When we use else with for loop, the compiler will only go into the else block of code when two conditions are satisfied:

- The loop is normally executed without any error.
- We are trying to find an item that is not present inside the list or data structure on which we are implementing our loop.

Except for these conditions, the program will ignore the else part of the program. For example, if we are just partially executing the loop using a break statement, then the else part will not execute. So, a break statement is a must if we do not want our compiler to execute the else part of the code.

For Example:

```
for i in ['C','O','D','E']:
 print(i)
else:
print("Statement successfully executed")
```

Output:

```
C
O
D
E
Statement successfully executed
```

In the code above, we iterate over the list and print each element. Since we let the loop complete normally, the else statement is also executed, and "statement successfully executed" is printed. Conversely, if we stop the loop by using the break statement, then the else block will not execute.

Why do we implement the else functionality with for loops?

We mostly use such implementations in our program when we want to encounter less logical errors and want to know that our program is functioning correctly. In such cases, we will most probably send a print statement inside the code, and based on its execution, we could see if our code is working correctly. In bigger-level projects, there are more chances of bugs occurrence, so we try to implement as many such techniques as possible so we do not have to spend too much time debugging. This will help us to know if our loop is functioning properly or not.

In this tutorial, we learned the implementation of the loop-else statement and the concept behind it. There is no such case where the use of else statement with for loop is mandatory. It is completely optional. Without using else, we can also set a flag that checks if any of the values met the condition or not. We use the else with a loop because it makes the code more elegant and readable.

Code as described/written in the video

```
khana = ["roti", "Sabzi", "chawal"]

for item in khana:
    if item == "rotiroll":
        break

else:
    print("Your item was not found")
```

Function Caching In Python

In today's tutorial, we are going to learn about ***function caching***. If you have some background related to computer science, you must have come across this term in relation to the operating system. Or you may have seen the caching term in your browser history settings. Before discussing function caching, we must know what caching is.

What does the term caching mean?

Caching means storing the data in a place from where it can be served faster. In the case of data that has been frequently used, the computer assigns a cache memory, so it does not have to load it again and again from the main memory. The purpose of the cache is to make the tasks more efficient and quicker. The same is true for web browsers; the pages we load again and again are stored in the cache for faster retrieval. In Python, however, we have to do it all manually, as the program will not store anything in the cache itself.

How to use function caching in Python?

Function caching is a way to improve code's performance by storing the function's return values. Before the 3.2 updates of Python, we had to create a cache ourselves by storing the value in a variable or by other such means. But in **Python 3.2**, there is a new update in the **functools** module of Python. To use this module, we have to import it first.

```
1 import functools
```

We have been facilitated with the help of a decorator known as **lru_cache. Decorators** are an essential part of Python. Decorators in Python can be used for a variety of different purposes.

If you are not familiar with the concept of a decorator, go and watch ***Tutorial #51*** on decorators or give the description a read.

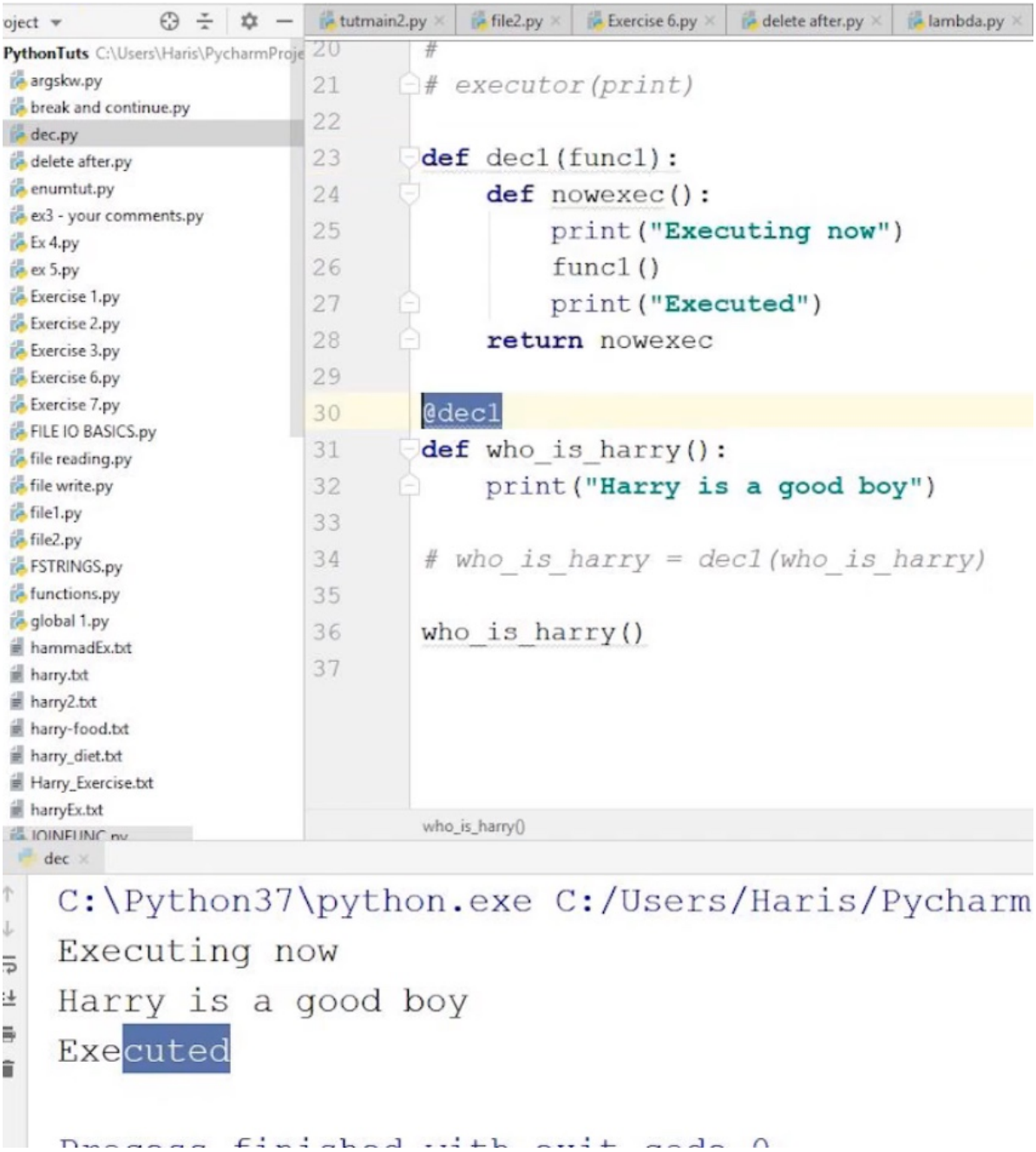


Figure1: Decorators in Python

We have to pass maxsize as a parameter with the decorator. maxsize is defined to tell the program how many values we want to store in the cache. It automatically stores the values for the latest number of calls.

For example

```
@functools.lru_cache(maxsize=4)
def myfunc(x):
    time.sleep(2)
    return x

myfunc(1)
# myfunc(1) takes 2 seconds and results for myfunc(1) are now cached
myfunc(1)
myfunc(2)
myfunc(3)
myfunc(4)
myfunc(5)
```

Talk is cheap. Show me the code.

We set the maxsize equal to 4, and the program uses the same call five times. Then, the program will only be able to retrieve the data faster for the last five calls because caching is only storing data for them. It is important to define the maxsize as per our requirements because it takes up memory accordingly, so for a better program, it should be precisely according to our needs.

We are using the **time module** as an example in this tutorial to understand better function caching working. We are using its function is known as **time.sleep(),** which delays the execution of further commands for given specific seconds of time. The number of seconds is sent as a parameter within parenthesis. If you are not familiar with Python's time module, visit ***tutorial #42,*** i.e., **Time Module In Python**, for a better understanding.

Code as described/written in the video

```
import time
from functools import lru_cache

@lru_cache(maxsize=32)
def some_work(n):
    #Some task taking n seconds
    time.sleep(n)
    return n

if __name__ == '__main__':
    print("Now running some work")
    some_work(3)
    some_work(1)
    some_work(6)
    some_work(2)
    print("Done... Calling again")
    input()
    some_work(3)
    print("Called again")
```

Else & Finally In Try Except

In this tutorial, we are going to learn about four keywords. Two of them we have already covered in the previous tutorial, and the other two are new to us. So, let's get to our actual topic. We are going to learn about try, except, else and finally. We are already familiar with try and expect, but for the sake of revision, we will briefly go through them one more time.

try and except block:

In the **try block**, we write the code in which an exception might occur, and in **except block**, we write the code as a result if an exception occurs. This could either be a print statement or the error itself. If no exception occurs, the except block will not execute. The purpose of try and except is to keep the program running either an error or exception occurs. In simple programs, if an error occurs, the program stops its execution and displays the error onto the screen. We can show the error as a string onto the screen by using try and except, and the program could continue its execution after that. For further explanation, you can give Tutorial #24 a watch or read the description given below.

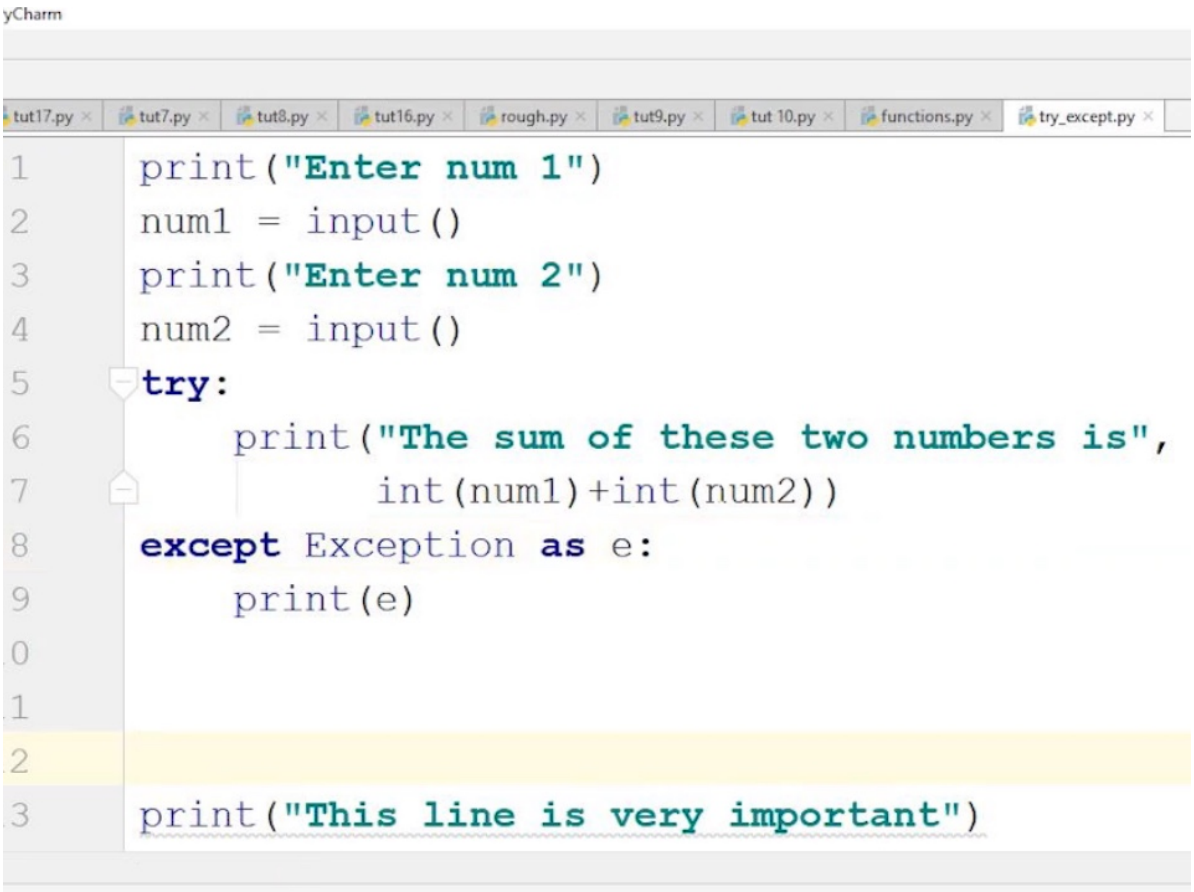


Figure1: try and except in Python

else keyword:-

Now moving on to the **else keyword**. We use an else keyword to print something in cases where no exception occurs. Suppose that an exception occurred, and the except block is printing the error; likewise, if the exception does not occur, we can print a statement that no error occurred, using an else keyword. Syntax of using else with try and except block is:

```
try:
    #Run this code
except Exception as error:
    #Execute this code when there is an exception
else:
    #No Exception. Run this code
```

Note: An else will only run in the case where no exception occurs.

For Example:-

```
def divide(a, b):
    try:
        print(f'{a}/{b} is {a / b}')
    except ZeroDivisionError as e:
        print(e)
    else:
        print("No Exception")
divide(1, 2)
```

Output:-

```
 1/2 is 0.5
No Exception
```

finally:-

Now the last keyword for this tutorial, i.e., **finally,** will run in either case. It is also known as code cleaner because it will perform its action, either an exception occurs or not. We write such commands in the finally part of the code that we want to execute, even an exception occurs or not. It is mostly used to clean resources or close files.

```
try:
    #Run this code
except Exception as error:
    #Execute this code when there is an exception
else:
   #No Exception. Run this code
finally:
   #Always run this code
```

Now, in cases where all the four keywords are being used simultaneously, which will run and which will not, can easily be understood by the table below:

Try	Not running	Running
Except	Will run	Will not run
Else	Will not run	Will run
Finally	Will run	Will run

Summing Up

After seeing the difference between these four keywords, we learned about various ways to handle exceptions in Python. In this tutorial, so studied that:

- In the try block, all the statements are executed until an exception occurs.
- Except block is used to catch and handle the exception(s) that occurs during the execution of the try block.
- Else block runs only when no exceptions occur in the execution of the try block.
- Finally block always runs; either an exception occurs or not.

This tutorial ends here, and I will see you in the next tutorial. Till then, keep coding and keep learning.

Code as described/written in the video

```
f1 = open("harry.txt")

try:
    f = open("does2.txt")

except EOFError as e:
    print("Print eof error aa gaya hai", e)

except IOError as e:
    print("Print IO error aa gaya hai", e)

else:
    print("This will run only if except is not running")

finally:
    print("Run this anyway...")
    # f.close()
    f1.close()

print("Important stuff")
```

Coroutines In Python

Coroutines are mostly used in cases of time-consuming programs, such as tasks related to machine learning or deep learning algorithms, or in cases where the program has to read a file containing a large number of data. In such situations, we do not want the program to read the file or data again and again, so we use coroutines to make the program more efficient and faster. Coroutines run endlessly in a program because they use a while loop with a true or 1 condition, so it may run until infinite time. Even after yielding the value to the caller, it still awaits further instruction or calls. We have to stop the execution of the coroutine by calling the coroutine.close() function. It is crucial to close a coroutine because its continuous running can take up memory space, as we have discussed in ***Tutorial #75***, related to function caching. We can define a coroutine using the following statements.

```
def myfunc():
    while True:
        value = (yield)
```

The while block continues the execution of the coroutine for as long as it receives values. The value is collected through the yield statement.

Coroutine Execution:-

Execution is the same as of a generator. When you call a coroutine, nothing happens. They only run in response to the next() and send() methods. Coroutine requires the use of the next statement first so it may start its execution. Without a next(), it will not start executing. We can search a coroutine by sending it the keywords as input using object name along with send(). The keywords to be searched are send inside the parenthesis.

When we run the **next() function** the first time, the coroutine executes and waits for new input. After the input is sent to it using the send() function, it executes it and again waits for next input, and the process goes on like this because we have set the while loop as true, so it will never exit its execution. In order to make the execution stop, we have to close the coroutine using coroutine.close() function.

- **send() — used to send data to coroutine**
- **close() — to close the coroutine**

Example:

```
def myfunc():
    print("Code With Harry")
    while True:
        value = (yield)
        print(value)

coroutine =myfunc()
next(coroutine)
coroutine.send("Python")
coroutine.send(" Tutorial ")
coroutine.close()
```

Output:-

```
Code With Harry
 Python
 Tutorial
```

After closing the coroutine, if we send values, it will raise the **StopIteration** exception. Coroutines are used for data processing mechanisms. **Coroutines** are similar to generators, except they wait for information to be sent to it using send() function.

Code as described/written in the video

```
def searcher():
    import time
    # Some 4 seconds time consuming task
    book = "This is a book on harry and code with harry and good"
    time.sleep(4)

    while True:
        text = (yield)
        if text in book:
            print("Your text is in the book")
        else:
            print("Text is not in the book")

search = searcher()
print("search started")
next(search)
print("Next method run")
search.send("harry")

search.close()
search.send("harry")
# input("press any key")
# search.send("harry and")
# input("press any key")
# search.send("thi si")
# input("press any key")
# search.send("joker")
# input("press any key")
# search.send("like this video")
```

Os Module

OS module provides our code with a direct connection to the operating system. We can use its different functions to interact and do activities on our operating system. For example, if we want to create such software that needs to store or access files from a directory or folder, we can use the OS module to perform the task for us. To use OS Module in Python, we have to import it.

```
1 import os
```

Build-in Function in OS Module:-

OS modules have a lot of built-in functions. You can see the list of built-in functions we can run through it by running the following code, also known as object introspection(For more details, visit **tutorial #70**):

We will discuss some of these main functions here in this tutorial.

Built-in Functions	Working
print(dir(os))	It gives us a list of all the functions the OS module is composed of.
os.getcwd( )	Here cwd is a short form for the current working directory. The function returns us the path of the directory we are currently in. It is important to know about our directory because when we are trying to import a file in python, the compiler searches for it in our current directory.
os.chdir( )	It is used in case we want to change our directory. The new path is sent inside the parenthesis. If we want to access some files or folders from some other directory, we can use it.
os.listdir( )	If we want to output the names of all the directories at a certain location, we can use this function.
os.mkdir( )	To create a new directory or folder. The name is sent as a parameter inside the parenthesis.
os.makedirs( )	To make more than on directory simultaneously.
os.rename( )	To rename an already existing directory, we use this. We send the current name and new name of our directory as parameters.
os.rmdir( )	It deletes an empty directory.
os.removedirs( )	We can remove all directories within a directory by using removedirs().
os.environ.get( )	It will return us the environment variable. The environment variable must be set, or the function will return null.
os.path.join( )	It joins one or more path components. We can join the paths by simply using a + sign, but the benefit of using this function is that we do not have to worry about extra slashes between the components. So less accuracy still provides us with the same result.
os.path.exists( )	It returns us a Boolean value, i.e., either true or false. It is used to check whether a path exists or not. If it does, then the output will be true, otherwise false.
os.path.isfile( )	It returns true if the path is an existing regular file.
os.path.isdir( )	It returns true if the path is an existing directory

Summing Up:-

In this tutorial, we learned about the OS module with Python 3. The main purpose of the OS module is to interact with the operating system. The primary use of the OS module is to create folders, remove folders, move folders, and sometimes change the working directory. OS module has a lot of built-in functions, some of which we've already discussed in this tutorial. You can explore more OS module build-in functions by reading its python documentation.

Code as described/written in the video:

```
import os
# print(dir(os))
# print(os.getcwd())
# os.chdir("C://")
# print(os.getcwd())
# f = open("harry.txt")
# print(os.listdir("C://"))
# os.makedirs("This/that")
# os.rename("harry.txt", "codewithharry.txt")
# print(os.environ.get('Path'))
# print(os.path.join("C:/", "/harry.txt"))

# print(os.path.exists("C://Program Files2"))
print(os.path.isfile("C://Program Files"))
```

Requests Module For HTTP Requests

In today's tutorial, we will learn about the requests module and how we can send HTTP requests through our program directly. By sending HTTP requests, we get a response object as a return. The request library in Python is the standard for making HTTP requests. Let us first understand what HTTP means.

What is HTTP?

HTTP stands for the ***'Hypertext Transfer Protocol,'***. It is a set of protocols that are designed to enable communication between clients and servers. Between clients and servers, it works as a request-response protocol. To request a response from the server, we can request data from the server or submit data to be processed to the server.

What Is Requests Module?

The response data depends on our type of request. The requests module is not a built-in Python module; instead, we have to download it manually. Requests module is used to send all kinds of HTTP requests. It is also one of the most downloaded modules in Python because all the web-related software and programs must have it in them.

Install Python Requests:-

To work with the requests module, the first step is to install the module in Python. To download a requests module, we can run the code:

```
pip install requests
```

After that, we have to import our module into the program, the same as we import all other modules.

```
import requests
```

Built-in methods in the request module:-

There are a lot of built-in methods in request module, such as delete(), get(), Head(), put(), etc. We will see the working of the get() function in this tutorial in detail.

get():

From the name, we can detect that the get function returns us some information about the site we requested. All the information is stored in the object we used to send the request. We can access different kinds of information through it, such as status, header, cookies, etc. To make a GET request, invoke

The basic syntax is:

```
requests.get(URL, params={key: value}, args)
```

***URL*:** this is the URL of the website where we want to send the request.

***Params*:** this is a dictionary or a list of tuples used to send a query string.

Args: *It is optional. It* can be any named arguments offered by the get method.

We can also fetch all the data from the homepage of a website into an HTML format using the request module. Few important types of methods defined in the request module are as follows:

Methods	Working
put( ):	It is used to send a put request to a variable. It is usually used to update or completely change the resources of a specific URL.
delete( ):	Delete is used to delete the specific resource specified by URL.
head( ):	The head method returns a header for a specific resource.
post( ):	Post requests take with it the data to the server to update it with.
patch( ):	The patch is used to send patch requests. It is used to apply partial modifications to a resource. It carries with it the instructions for the modification.

For more detail, you can check the Python documentation about the requests module. This tutorial ends here, and I will see you in the next tutorial.

Code as described/written in the video

```
import requests
r = requests.get("https://financialmodelingprep.com/api/company/price/AAPL")
print(r.text)
print(r.status_code)

# url = "www.something.com"
# data = {
#     "p1":4,
#     "p2":8
# }
# r2 = requests.post(url=url, data=data)
```

Json Module

Our today's topic for this course is the ***JSON module in Python***. Before learning about the module, we should be familiar with what Json is actually. JSON stands for JavaScript Object Notation. JSON is a data-interchange format derived from JavaScript. It is mostly used for storing or transferring data. So, if we want our program to interact with the internet, we must be familiar with this module, even only to send or receive data through the internet. It is one of Python's most important modules because if we want it to interact only a bit through the internet, the Json module must be imported first. A JSON is an unordered collection of key and value pairs similar to a python dictionary. The following are some important points about JSON.

- Keys are unique strings that cannot be null.
- Values can be anything from a String, Number, Tuple, Boolean, list, or null.
- A JSON is represented by a string which is enclosed within curly braces { } with key-value pairs which are separated by a colon (:), and pairs separated by a comma(,).

Below is an example of JSON data. We can notice that the data representation is similar to Python dictionaries.

```
{"name": "harry", "salary": 9000, "language": "Python"}
```

JSON in Python:

JSON is already built-in in Python, so no need for an installation command. We can import it so we may start working on it. JSON module in Python helps us in converting the data structures to JSON strings. Use the import function to import the JSON module in your Python program.

```
import json
```

If we convert a JSON string to Python, the resultant will be a dictionary. The conversion is also known as parsing, and that is the keyword we use professionally for the conversion. We can convert from JSON to Python or Python to JSON by using json.loads() or json.dumps() methods. Methods:

- **load():** This method is used to load data from a JSON file into a python dictionary.
- **Loads( ):** This method is used to load data from a JSON variable into a python dictionary.
- **dump()**: This method is used to load data from the python dictionary to the JSON file.
- **dumps():** This method is used to load data from the python dictionary to the JSON variable.

Following is the program showing the use of JSON package in Python

```
import json
a ={"name":"harry","salary":9000,"language":"Python"}
# conversion to JSON done by dumps() function
b = json.dumps(a)
print(b) # printing the output
```

Output:-

```
{"name": "harry", "salary": 9000, "language": "Python"}
```

What parsing actually does?

Parsing converts the code from one form to another, making it compatible with running on the other platform by changing all the little syntax differences and making it perfect for running on the other platform. The following table shows how Python objects are converted to JSON objects.

JSON	PYTHON
true	true
false	false
string	string
number	number
array	list, tuple
object	dict
null	none

Summing Up:

This tutorial has taught us how to create, manipulate, and parse JSON in Python using the JSON module. JSON is most commonly used for client-server communication because it is human-readable and easy to read/write. JSON is mainly based on key-value pairs, similar to a dictionary in Python. To use the JSON module in Python, we have to import it. While using json.dumps(), we can send separators in parameters to make the code more readable and well defined. The separators could be full stops, commas, blank spaces, etc.

Code as described/written in the video

```
import json

data = '{"var1":"harry", "var2":56}'
print(data)

parsed = json.loads(data)
print(type(parsed))

#Task 1 - json.load?


data2 = {
    "channel_name": "CodeWithHarry",
    "cars": ['bmw', 'audi a8', 'ferrari'],
    "fridge": ('roti', 540),
    "isbad": False
}

jscomp = json.dumps(data2)
print(jscomp)

# Task 2 = what is sort_keys parameter in dumps
```

Pickle Module

In this tutorial, we are going to learn about the pickle module in Python. Pickle means to preserve, and in Python, we use it for the same purpose. Pickle comes in handy while saving complicated data. They are easy to use, lighter, and do not require several lines of code. The pickled file generated is not easily readable and thus provides some level of security. We will divide this tutorial into two parts i.e.

- Pickling
- Unpickling

First, we will learn about the basics of pickling and how to achieve it.

What is pickling?

Pickling is the process of serializing an object. Serializing means storing the object in the form of binary representation so it can be saved in our main memory. The object could be of any type. It could be a string, tuple, or any other sort of object that Python supports. The data is stored in the main memory in a file. While writing the code for pickling, we open the file in ***"wb"*** mode, also known as writing binary mode. So, to use the pickle module, we have to make a file with the .pkl extension and send it in a **dump()** function along with the object. dump() is a built-in function in the Pickle module, made for pickling.

Pickling files:-

To use pickle, we have to import it in Python.

```
import pickle
```

In this example, we will pickle a dictionary. We will save it to a file and then load it again.

```
py_dict = { 'name': 'harry', 'salary':9000, 'language': 'Hindi' }
myfile = open('filename','wb')
pickle.dump(py_dict,myfile)
myfile.close()
```

What is unpickling?

The file format is binary, so we cannot directly open and read it; instead, we have to open it using a python program, and the process is known as unpickling. We have to open the file in ***"rb"*** mode for unpickling, also known as a read binary mode. The function we use this time is also a built-in function, named **load()**. Unlike dump(), we have to send the file name as a parameter, and it automatically retrieves the data in the object type it was inserted in. For example, if we send a list while pickling, the return result will also be a list.

```
myfile = open(filename,'rb')
py_dict = pickle.load(myfile)
myfile.close()
```

To make sure that you successfully unpickled it, you can print the dictionary, compare it to the previous dictionary and check its type with type().d")

We can face some of the common pickle exceptions raised while dealing with the pickle module.

- **Pickle.PicklingError:** If the pickle object does not support pickling, then Pickle.PicklingError exception is raised.
- **Pickle.UnpicklingError:** This exception will raise if the file contains bad or corrupted data.
- **EOF Error:** This exception will be raised if the end of the file is detected.

Disadvantages:

- There are some situations in which pickling is discouraged. For example, when we are working with multiple programming languages, pickle is discouraged.
- Pickle has been found slower than its alternatives.
- In some cases, it has also shown a few security vulnerabilities.
- When we update our program to a newer version, pickled data through the previous version can cause issues.

Conclusion:-

In this tutorial, we learned about the Pickle module. In examples, we pickled a Python dictionary, but we can also use the same method to save a large spectrum of Python data types, as long as we make sure our objects contain only other pickleable objects. There are some disadvantages of using the pickle module. When you need a cross-language solution, JSON is a better option.

Code as described/written in the video

```python
import pickle

# Pickling a python object
# cars = ["Audi", "BMW", "Maruti Suzuki", "Harryti Tuzuki"]
# file = "mycar.pkl"
# fileobj = open(file, 'wb')
# pickle.dump(cars, fileobj)
# fileobj.close()

file = "mycar.pkl"
fileobj = open(file, 'rb')
mycar = pickle.load(fileobj)
print(mycar)
print(type(mycar))


# pickle.loads = ?
```

Regular Expressions

Regular expressions are used to perform search-related tasks in Python. In this tutorial, our primary focus should be on understanding because we are going to cover a concept that has a wide range of uses. To work with regular expressions, we have to import a built-in module in Python called **'re'.**

```
1 import re
```

The module defines several functions and constants to work with RegEx. The "re" module is composed of five functions known as:

- **findall**: It finds all searches for matches and prints resultant in the form of a list.
- **search**: It works the same as a findall, but the resultant is a matched object if any is found.
- **split**: The split function splits the string from every matched into two new strings.
- **sub**: The sub-function works exactly like a replace function in notepad or MS Word. It replaces the original word with a word of our choice.
- **finditer**: The finditer yields an iterator as a resultant with all the objects that match the one we sent it) finditer supports more attributes than any other function defined above. It also provides more details related to the matched object. So, most of the examples we are going to see next will contain a finditer function in them.

So, you must be wondering that all of the searchings can easily be done using a simple loop with some conditions so, what is the purpose of the "re" module. Well "re" module is used for complex searching, using Metacharacters and special sequences.

Metacharacters have special meaning in Python, and they are used with "re" modules to search for keywords and objects more technically and efficiently. We will see the working of a few Meta Characters in this tutorial so you can get an idea. I will provide you with a list of these characters and their working at the end of this tutorial.

Use of "^":-

We use the" ^" symbol to check whether the string is starting from the keyword we wrote after ^ or not. For example, if a string starts from CodeWithHarry and we are searching the keyword using ^CodeWithHarry with finditer, it will return us whether our string is starting from the searched keyword or not. The same is the case for $ sign. It will check whether our string is ending with the specific keyword or not.

Use of "|":-

We can also use a unique character "|" to use more than one condition, so if we use it for the above case, then it will check whether the string starts or ends with CodeWithHarry.Now we will move on to special sequences. We will see a few special sequences in this tutorial, and you can have a look at the list of these sequences at the end of the tutorial description for further practice.

- **\A:** the resultant is a match if the input characters are at the beginning of the string
- **\b** the resultant is a match whether the input characters are at the beginning or the end of a word
- **\d** the resultant is a match if the string contains any digits
- **\s** the resultant is a match if the string contains a white space character

There are many metacharacters supported by the re module. Some characters with their working are the following:

- **‘.’:** Matches any single character except newline
- **‘$’:** Anchors a match at the end of a string
- **‘*’:** Matches zero or more repetitions
- **‘+’:**Matches one or more repetitions
- **‘{}’:** Matches an explicitly specified number of repetitions
- **‘[]’:** Specifies a character class

To explore more about the re module, check the https://docs.python.org/3/library/re.html python documentation.

Meta Characters:-

- **[]** A set of characters
- \ Signals a special sequence (can also be used to escape special characters)
- **.** Any character (except newline character)
- **^** Starts with
- $ Ends with
- * Zero or more occurrences
- + One or more occurrences
- **{}** Exactly the specified number of occurrences
- **|** Either or
- **()** Capture and group

Special Sequences:-

- \A Returns a match if the specified characters are at the beginning of the string
- \b Returns a match where the specified characters are at the beginning or at the end of a word r” ain\b.”
- \B Returns a match where the specified characters are present, but NOT at the beginning (or at the end) of a word
- \d Returns a match where the string contains digits (numbers from 0-9)
- \D Returns a match where the string DOES NOT contain digits
- \s Returns a match where the string contains a white space character
- \S Returns a match where the string DOES NOT contain a white space character
- \w Returns a match where the string contains any word characters (characters from a to Z, digits from 0-9, and the underscore _ character)
- \W Returns a match where the string DOES NOT contain any word characters
- \Z Returns a match if the specified characters are at the end of the string

Code as described/written in the video

```
import re
mystr = '''Tata Limited
Dr. David Landsman, executive director
18, Grosvenor Place
London SW1X 7HSc
Phone: +44 (20) 7235 8281
Fax: +44 (20) 7235 8727
```

```
Email: tata@tata.co.uk
Website: www.europe.tata.com
Directions: View map

Tata Sons, North America
1700 North Moore St, Suite 1520
Arlington, VA 22209-1911
USA
Phone: +1 (703) 243 9787
Fax: +1 (703) 243 9791
66-66
455-4545
Email: northamerica@tata.com
Website: www.northamerica.tata.com
Directions: View map fass
harry bhai lekin
bahut hi badia aadmi haiaiinaiiiiiiiiiiii'''
```

Converting .py to .exe

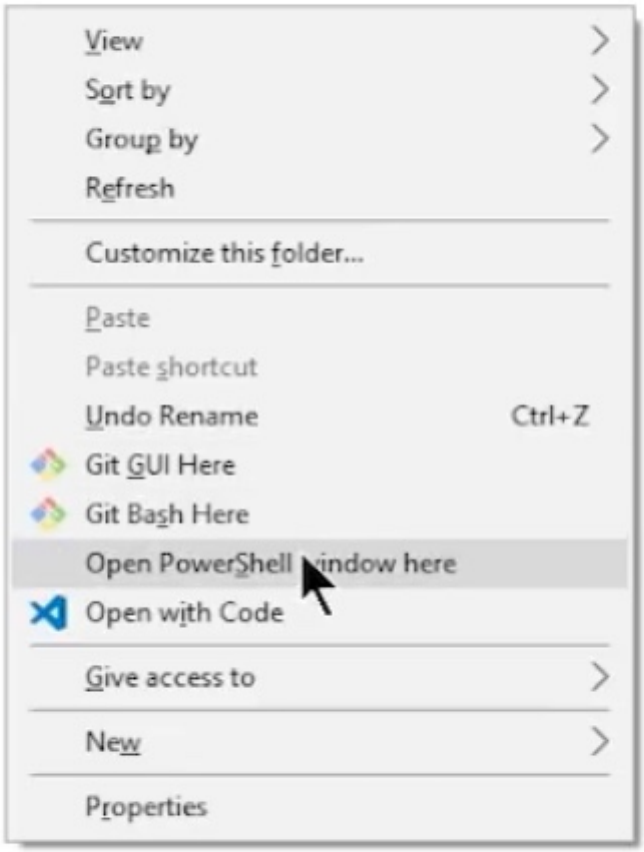
Today we will discuss the process of converting a python, i.e., the .py file, into an executable or .exe file. Let us start with the **introduction of the .exe file**. In windows, **.exe** is an extension used for executable files. It is one of the most common file extensions used for almost all kinds of software and applications programs and setups.

Now let's come to the purpose of the conversion. We create lots of Python programs and want to share them with the world. If we want our python program to run on any computer without the IDE or even installing Python itself, we must convert it to .exe. All the apps and programs we use on our computer are written using some language or code, but we do not have to install the particular language for the program to run.

Steps to Create an Executable from Python Script using Pyinstaller

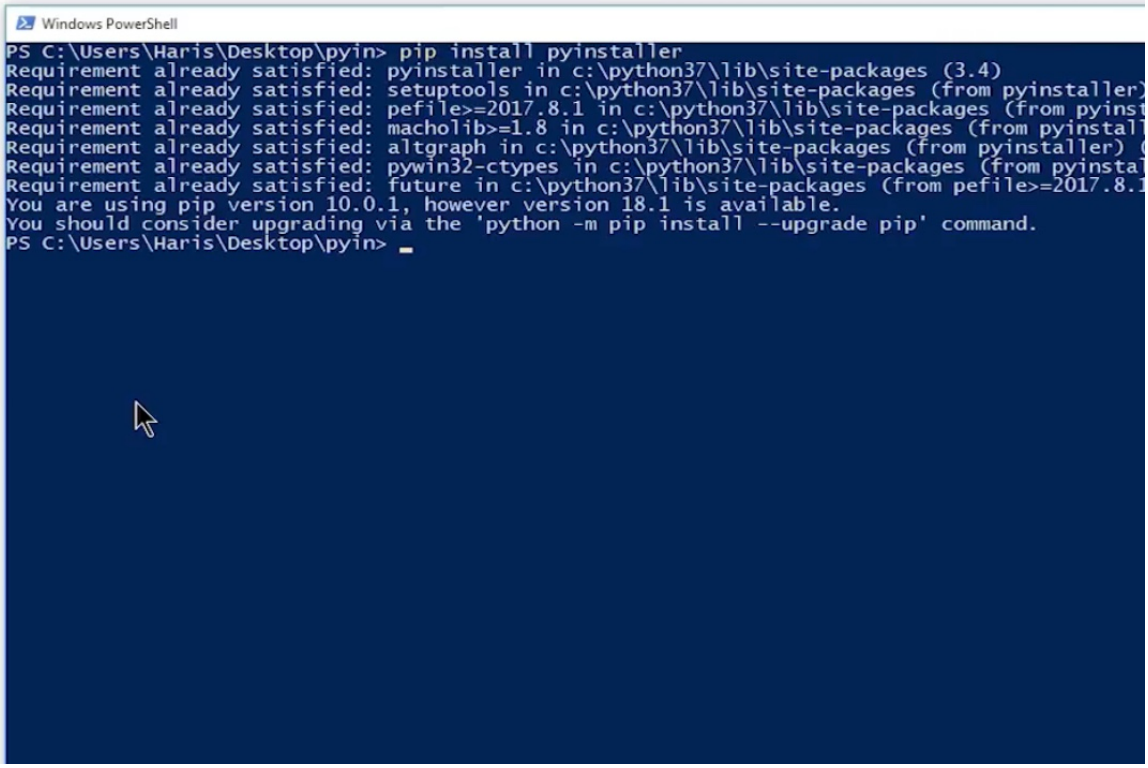
Step 1: Install the Pyinstaller Package

The first step is to install the module pyinstaller. For this, we will create or destinate a folder and open our power shell window there using shift + mouse right-click.



Now we will run the following command for our module installation.

```
pip install pyinstaller
```

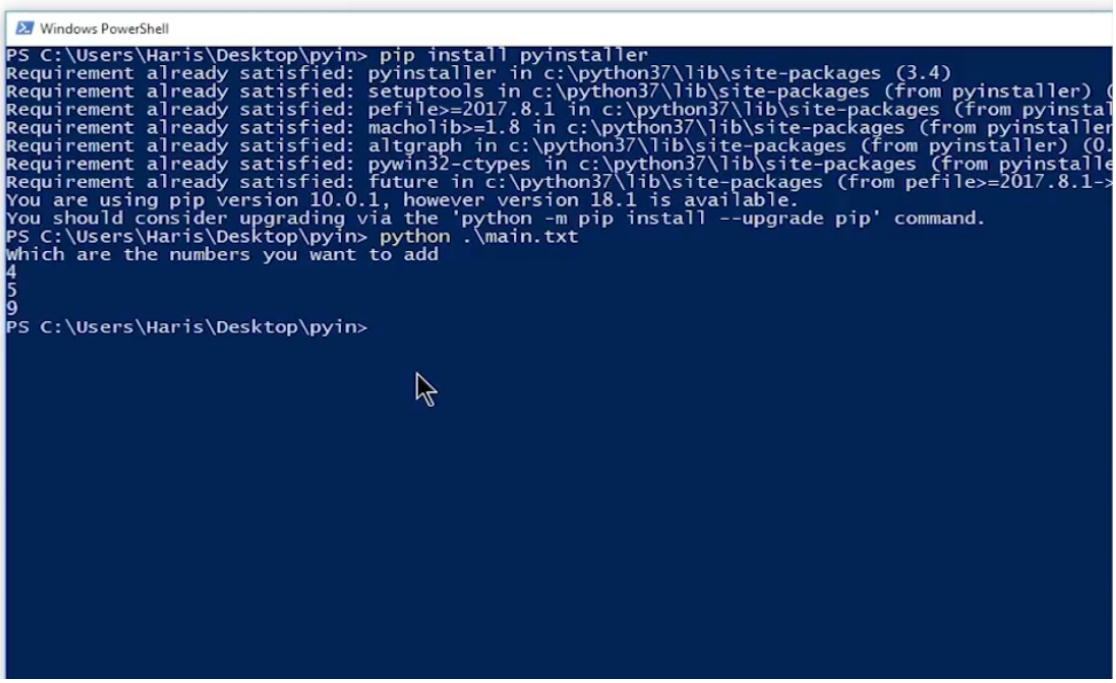


Step 2: Save your Python Script

Right-click on the screen and then new and text document to create a .txt file and write the python script. We can save our python script in a text file, too. For the conversion, it does not necessarily have to be a .py file, but the code should be Python. Now open the power shell window and write the following commands.

```
python .\main.txt
```

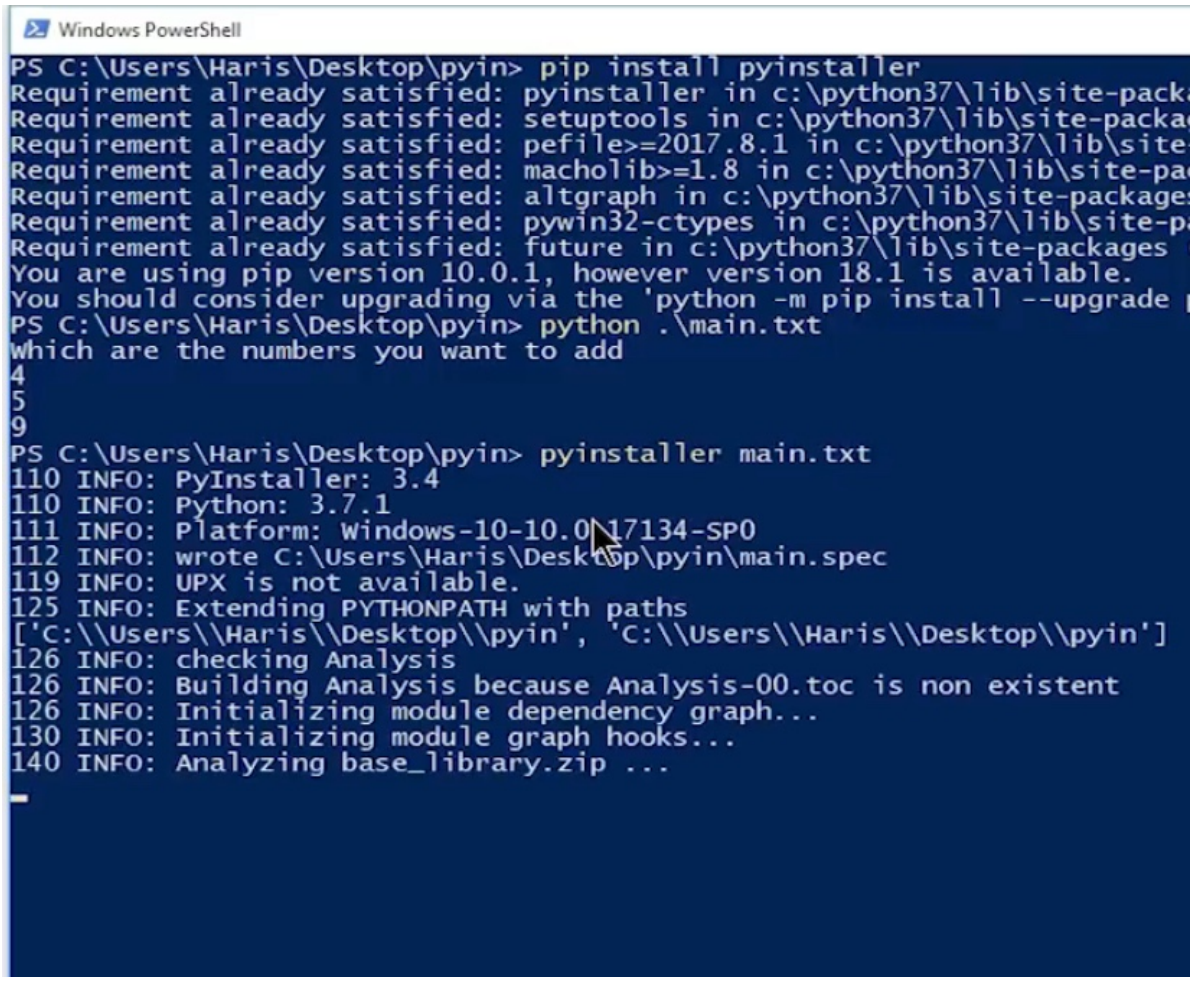
The code written in the main.txt file will execute.



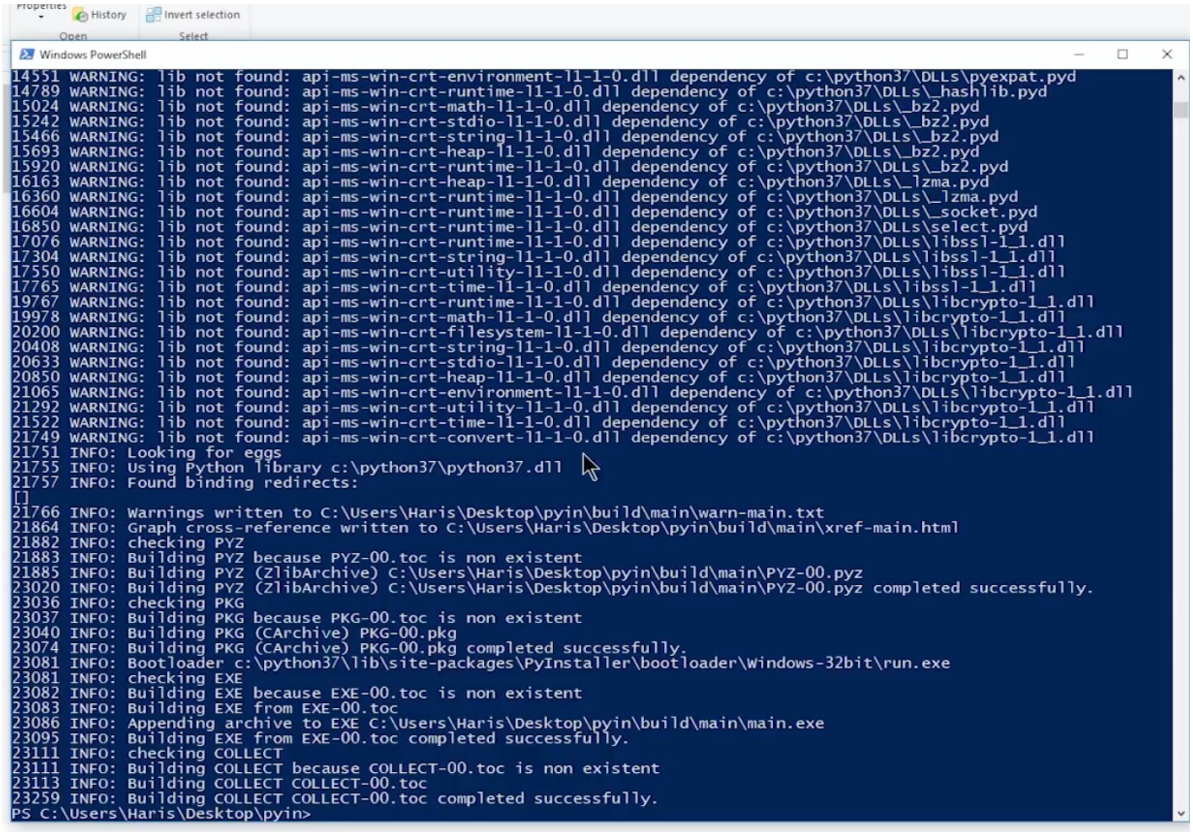
Step 3: Create the Executable using Pyinstaller

As we want a .exe file. So we are going to run the following command.

```
pyinstaller main.txt
```

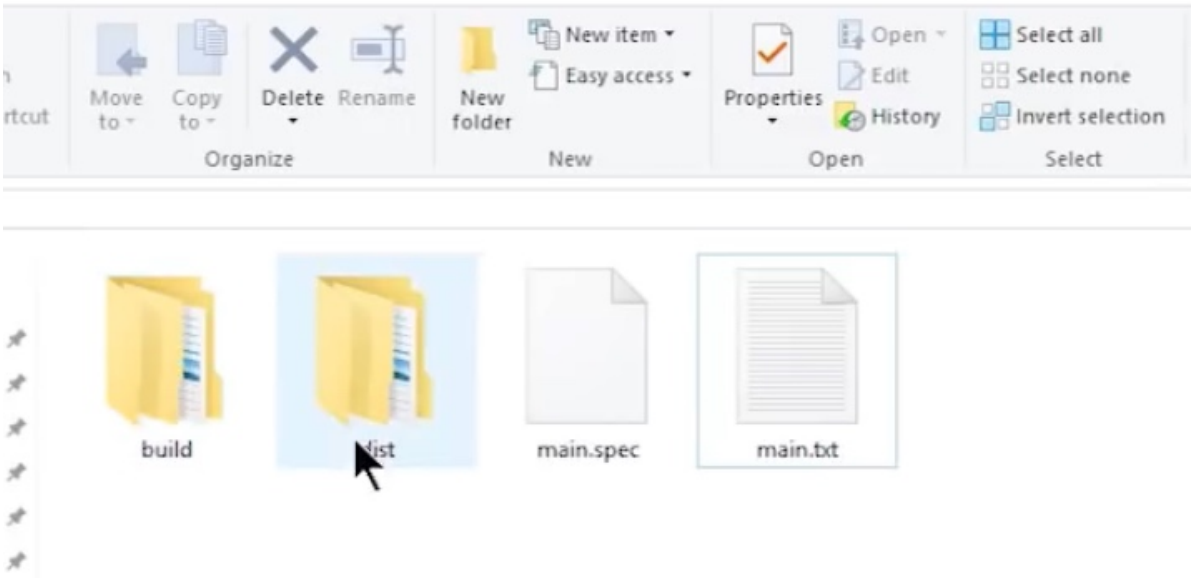


It will show us a few warnings, as shown in the image below. Along with that, it could be a little time-consuming in the case of bigger programs. We should not avoid the warnings as they can be a security threat later on. You can search for the meaning of the given warning by searching it on the internet. Sometimes software causes problems while installing the pyinstaller module. The reason for this might be the incomplete installation of pip. So for that check, pip is not recognized error solution video for the solution.



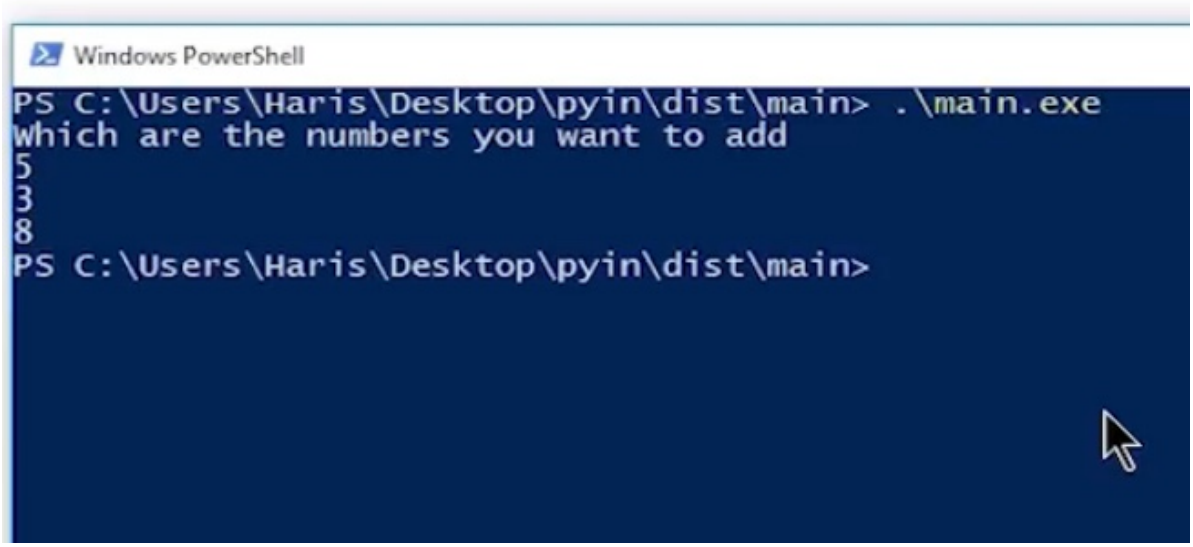
Step 4: Run the Executable

After running the pyinstaller main.txt command, it will create some folders. Click on the dist folder and then click on the main.



In that folder, we can find our .exe file. We can open it through the PowerShell window by running the command.

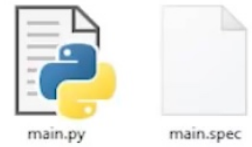
```
.\main.exe
```

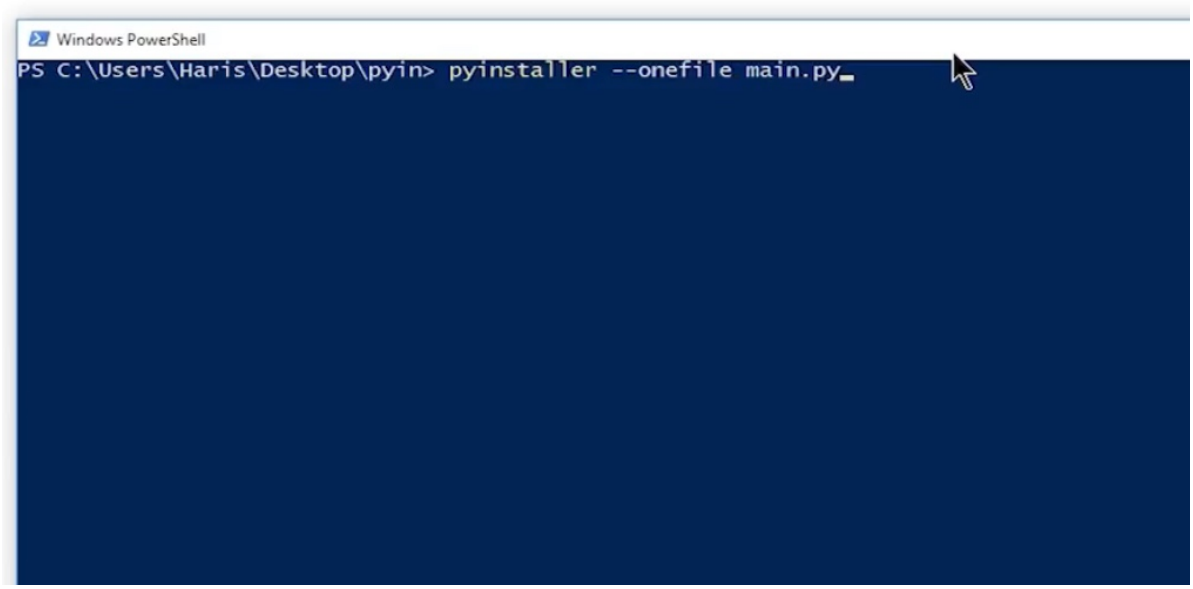


Now, as you saw, by converting the file to .exe, we got several files in the folder, but we can also run a command that will provide us with only one file as a resultant. It will take more time in creation, and later on, it will extract too, but for compatibility, we can use the following command:

```
pyinstaller --onefile file.py
```

Here the file extension depends on your file that whether you created it using .py or .txt.


main.py
main.spec


Windows PowerShell
PS C:\Users\Haris\Desktop\pyin> pyinstaller --onefile main.py

Raise In Python + Examples

In this tutorial, we are going to learn the benefits and uses of Raise In Python. You all must remember that in ***tutorial#24*** and ***tutorial#76***, we learned a few ways to handle the exception. In ***tutorial #24***, we learned about try and catch. In the try block, we write the code in which an exception might occur, and in except block, we write the code as a result if an exception occurs.

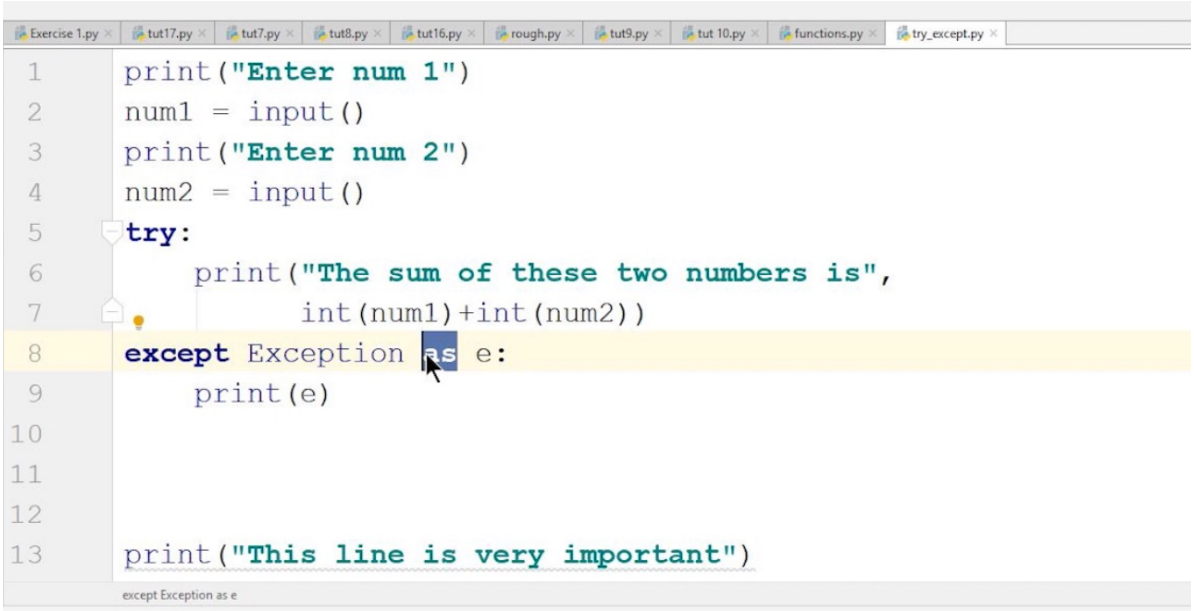


Figure1: Try and Except in Python

Moreover, in ***tutorial #76***, we learned about else, and finally. We use an else keyword to print something in cases where no exception occurs. It is also known as code cleaner because it will perform its action, either an exception occurs or not.

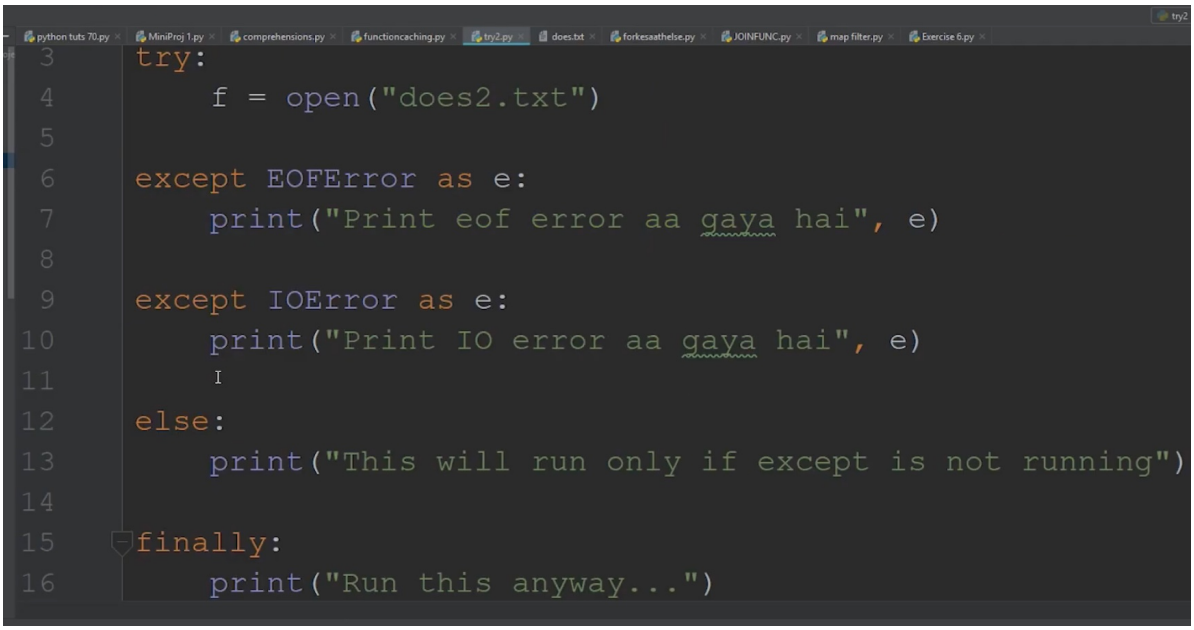


Figure2: Else and Finally in Python

If you have not watched them, go and watch them first or give their descriptions a read because they are important prerequisites to this tutorial.

First, let us briefly go over the meaning of the word exception. The ***exception*** is an error that halts the program's normal functioning and displays an error onto the screen. While the try and except block are for handling exceptions, the raise keyword, on the contrary, is to raise an exception.

Following is the syntax:

Syntax of raise keyword is:

```
if test_condition:
raise EXCEPTION_CLASS_NAME
```

Taking a simple usage example:

```
raise ZeroDivisionError
```

Python has a range of built-in exceptions that we can use for our benefit. We can learn and read about the exceptions by visiting https://docs.python.org/3/library/exceptions.html Python documentation of python site. A few of these exceptions include:

- **KeyError:** Raised when a mapping key is not found in the set of existing keys.
- **ValueError**: Raised when a function receives an argument with the right type but an inappropriate value.
- **EOFError (End Of File Error):** Raised when the input() function hits an end-of-file condition without reading any data.
- **ImportError:** Raised when the import statement has trouble trying to load a module.
- **NameError:** Raised when a local or global name is not found.
- **ZeroDivisionError:** Raised when the second argument of a division is zero.

These are only a few names. We will discuss few build-in exceptions in our code today, but you can search for more on the internet according to your requirement as nearly every possible exception is already out there. Covering so many of them in a single tutorial is impossible, so I will give you an idea about them with examples of the most frequently used exceptions. We can also make custom or user-defined exceptions that fulfills our purpose.

Example:-

Before moving on to their detailed work, let us cover the reason for their use. Suppose we have made a program in which we want a number that is greater than 10. Now the user is giving the input (x), 5. So in such a case, we can raise ValueError, returning an error to the user that the input is wrong. By doing this, we can save the program running time and prevent the program from storing the wrong input.

You can use the raise keyword to signal that the situation is exceptional to the normal flow. For example:

```
x = 5
if x < 10:
   raise ValueError('x should not be less than 10!')
```

Notice how you can write the error message with more information inside the parentheses. The example above gives the following output (by default, the interpreter will print a traceback and the error message):

```
>>>
Traceback (most recent call last):
File "C:/Python34/Scripts/raise1.py", line 3, in <module>
raise ValueError('x should not be less than 10!')
ValueError: x should not be less than 10!
>>>
```

Code as described/written in the video

```
# a = input("What is your name")
# b = input("How much do you earn")
# if int(b)==0:
#     raise ZeroDivisionError("b is 0 so stopping the program")
# if a.isnumeric():
#     raise Exception("Numbers are not allowed")
#
# print(f"Hello {a}")
# 1000 lines taking 1 hour

# Task - Write about 2 built in exception

c = input("Enter your name")
try:
    print(a)

except Exception as e:

    if c =="harry":
        raise ValueError("Harry is blocked he is not allowed")

    print("Exception handled")
```

Python 'is' vs '==': What's The Difference?

Our today's tutorial is related to a very basic yet very important concept. We are going to learn about the difference between 'is' and '==.' We have been using them both in our programs from the start. Sometimes one of them also works in place of the other. However, there is a huge difference in the working between the Python identity operator (is) and the equality operator (==) that we will cover in this tutorial.

Equality operator (==) in Python:

'==' is used to represent value equality. Value equality means two variables or objects or even data structures such as a list composed of the same value. Suppose we have two variables, a and b. We assign the value 2 to both of them. Now, as we know that they do not have any direct link with each other. The only similarity is that they have been given the same value. So, if we place an '==' sign between them, the output will be **True**. However, when we change the value of one of the variables, it will return false.

For Example:-

```
x = [1, 2, 3, 4]
y= [1, 2, 3, 4]
x == y
#True
```

In this example, '**x == y**' returns **true** because what **x** is referencing contains the same things that **y** is referencing.

Identity operator (is) in Python:

'is' is generally used to denote reference equality. The data structure or variables in the case has to be the same. In the case of the object, the objects must be referring to the same kind of data. In case we use a copy of our variable or data structure or even make a similar one with the same values, it will still return **False** as their reference is not the same. For example, if we assign a list to two different objects, the 'is' keyword will return true as they refer to the same list. If we change an entry in the list, it will also be changed for the other one, so no change in output.

For Example:-

```
c = [1, 2, 3]
d = [1, 2, 3]
c == d #True
c is d #False
```

When we assign a list to a variable, Python allocates memory for that list, but the actual list is not stored in our variable. Instead, Python creates a list object and stores a reference to that object in the variable. However, in the above example, c = [1, 2, 3, 4] and d = [1, 2, 3, 4]

This creates a list object and stores a reference to it in c; then, it creates a second list object and stores a reference to it in d.

'c == d' is still **true**. However, **'c is d'** is now **false**. This is because both **c** and **d** refer to **different objects**.

So, to recap the difference between "is" and "==" into short definitions:

- An identity operator(is) expression evaluates to True if two variables point to the same object.
- An equality operator (==)expression evaluates to True if the objects referred to by the variables have the same contents.

The identity operator 'is' tracks the object back to its identity while the equality operator '==' only compares the values.

Code as described/written in the video:

```
# == - value equality - Two objects have the same value
# is - reference equality - Two references refer to the same object


# Task:
a =[6, 4 , "34"]
b = [6, 4 , "34"]
print(b is a)
```

Python 2.x Vs Python 3.x

Today's tutorial is linked to an issue or a question that most of the viewers who are new to programming or the viewers who are programming in the older version of Python have in mind. The purpose of this tutorial is to provide them with a clear understanding of where they should start learning from and help them decide whether they should start learning from ***python 2.x or 3.x***.

As we know that with every new update, the bugs in the previous ones are resolved, and also new features are included, or previous ones are improved for the benefit of the users. However, along with all these advantages, sometimes a few functions that were not that good are replaced or are completely erased, and sometimes the implementation techniques change a little bit. All these scenarios that create problems for users need to be upgraded from one version to another.

What are the differences between Python 2 and Python 3?

The most common example we can take of such a scenario is the print statement. In Python 2.x, the print statement does not need a pair of parentheses. It just needed two single quotes to implement. While in 3.x, we have to write the quotes in double quotations inside the parenthesis.

```
#Python 2:
print "Hello world!"

#Python 3:
print ("Hello world!")
```

Another major syntax difference is the raw_input() function has changed. This is a common function that takes input from the user.

```
#Python 2:
user_input1 = raw_input(“entered_value”)

#Python 3:
user_input1 = input(“entered_value”)
```

Python 3.x offers a **range () function** to perform iterations whereas, In Python 2.x, the **xrange()** is used for iterations.

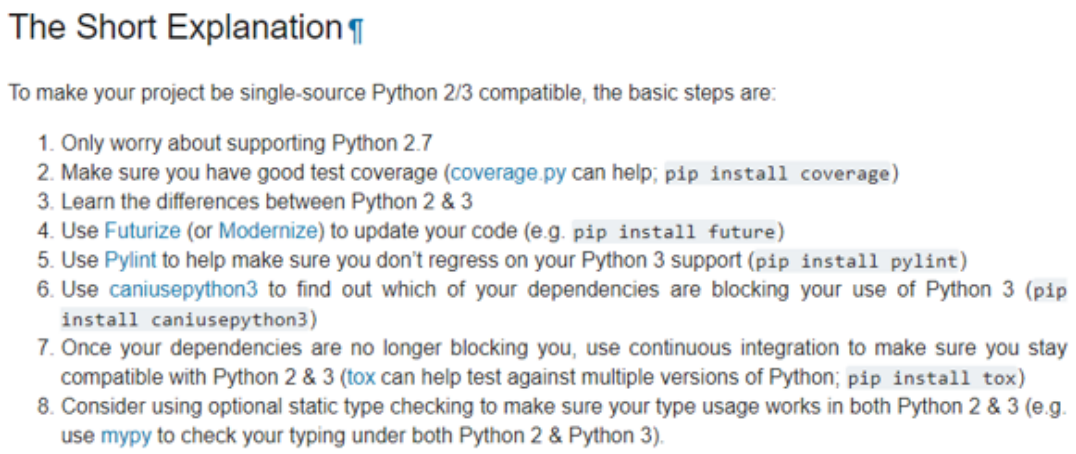
There are also some other differences like Python 3 default storing of strings is Unicode. In contrast, Python 2 stores need to define Unicode string value with "u.", and Python 3 exceptions should be enclosed in parenthesis while Python 2 exceptions should be enclosed in notations.

So, concluding the first purpose of this lecture, I would recommend new users to start learning from the latest version. It contains all the newly updated functions and methods, along with new modules or ways of programming. And after some time, most of the applications and software also update themselves according to the latest version.

How to convert code from Python 2.x to Python 3.x?

In the case of developers that are associated with the previous versions, the scenario poses some questions. When a new update arrives, all the applications, software are not updated readily according to the new update, but it takes some time. Also, developers sometimes have to choose where they should keep working with the same update or set their code according to the newer

update. Now converting all the code by writing it again can be very hectic and time-consuming in the case of programs with a larger number of the code. For such a scenario, Python has introduced a guide by following which we can make our code compatible for both 2.x or 3.x or convert it from 2.x to 3.x. The guidelines provided by Python can be seen in the screenshot below. You can visit the Python official documentation site for further explanation and guidelines.



The Short Explanation ¶

To make your project be single-source Python 2/3 compatible, the basic steps are:

1. Only worry about supporting Python 2.7
2. Make sure you have good test coverage (coverage.py can help; `pip install coverage`)
3. Learn the differences between Python 2 & 3
4. Use Futurize (or Modernize) to update your code (e.g. `pip install future`)
5. Use Pylint to help make sure you don't regress on your Python 3 support (`pip install pylint`)
6. Use caniusepython3 to find out which of your dependencies are blocking your use of Python 3 (`pip install caniusepython3`)
7. Once your dependencies are no longer blocking you, use continuous integration to make sure you stay compatible with Python 2 & 3 (tox can help test against multiple versions of Python; `pip install tox`)
8. Consider using optional static type checking to make sure your type usage works in both Python 2 & 3 (e.g. use mypy to check your typing under both Python 2 & Python 3).

Before converting your code, you should read all the guidelines carefully and interact with an expert for consultation prior to the conversion. If not done properly, the conversion can cause many bugs and security breaches in your program, so it is best to be safe than sorry. Many computers come with Python 2.7 pre-installed by default, but it's worth it to learn how to install and use Python 3. Check my ***download python*** tutorial to download python 3.x.

Creating a Command Line Utility In Python

What Is a Command Line Utility?

A command-line utility is a way of giving operating system instructions using lines of text. Command-line programs operate via the command line or PowerShell. It will interact with a command-line script.

Now let us come to why we should use the command-line utility in our program. We can easily call a command line program in Python or any other language into a different language program. Each program has calling support in it for calling the command lines program. So in cases, where we are writing a program in some other language, but we want to perform a task in Python and call it in our program, then the command line can help us do that.

Now we are going to discuss how part of this tutorial. For creating a Command Line Utility In Python, first import two modules, i.e., argsparse and sys. argsparse helps us to get command-line arguments in our program, and the sys module helps us to import the code we wrote using argparse onto the console. For more details and descriptions about these modules, you can read the python documentation for these modules.

```
import argparse
import sys
```

What is argparse?

Python comes with several different libraries that allow us to write a command-line interface for our scripts, but the standard way for creating a CLI in Python is by using the Python ***argparse module***. The argparse module helps us to parse the arguments passed with the script and process them more conveniently. One of the advantages of using the argparse module is that it makes it easy to write user-friendly command-line interfaces.

We can use the Python argparse module to create a command-line interface by following these steps:

1. Import the Python argparse module
2. Create the parser
3. Add optional and positional arguments to the parser
4. Execute .parse_args()

When we execute **.parse_args()**, we will get the Namespace object that contains a simple property for each input argument received from the command line. In this tutorial, we are going to use the Argumentparser class available in the argparse module. We fill **ArgumentParser** with information about program arguments by making calls to the **add_argument()** method.

What is the sys module?

Python provides the ***sys module*** that gives us independence from the host machine Operating System and allows us to operate on an underlying interpreter, irrespective of its being a Linux or Windows Platform. With the help of the sys module, we can access system-specific parameters and functions. It provides different functions used to manipulate different parts of the Python Runtime Environment. To use the sys module, we have to import it so that it brings required sys module dependencies into our scope.

Code as described/written in the video

```python
import argparse
import sys

def calc(args):
    if args.o == 'add':
        return args.x + args.y

    elif args.o == 'mul':
        return args.x * args.y

    elif args.o == 'sub':
        return args.x - args.y

    elif args.o == 'div':
        return args.x / args.y

    else:
        return "Something went wrong"

if __name__ == '__main__':
    parser = argparse.ArgumentParser()
    parser.add_argument('--x', type=float, default=1.0,
                        help="Enter first number. This is a utility for calculation. Please contact harry bhai")

    parser.add_argument('--y', type=float, default=3.0,
                        help="Enter second number. This is a utility for calculation. Please contact harry bhai")

    parser.add_argument('--o', type=str, default="add",
                        help="This is a utility for calculation. Please contact harry bhai for more")

    args = parser.parse_args()
    sys.stdout.write(str(calc(args)))
```

Creating a Python Package Using Setuptools

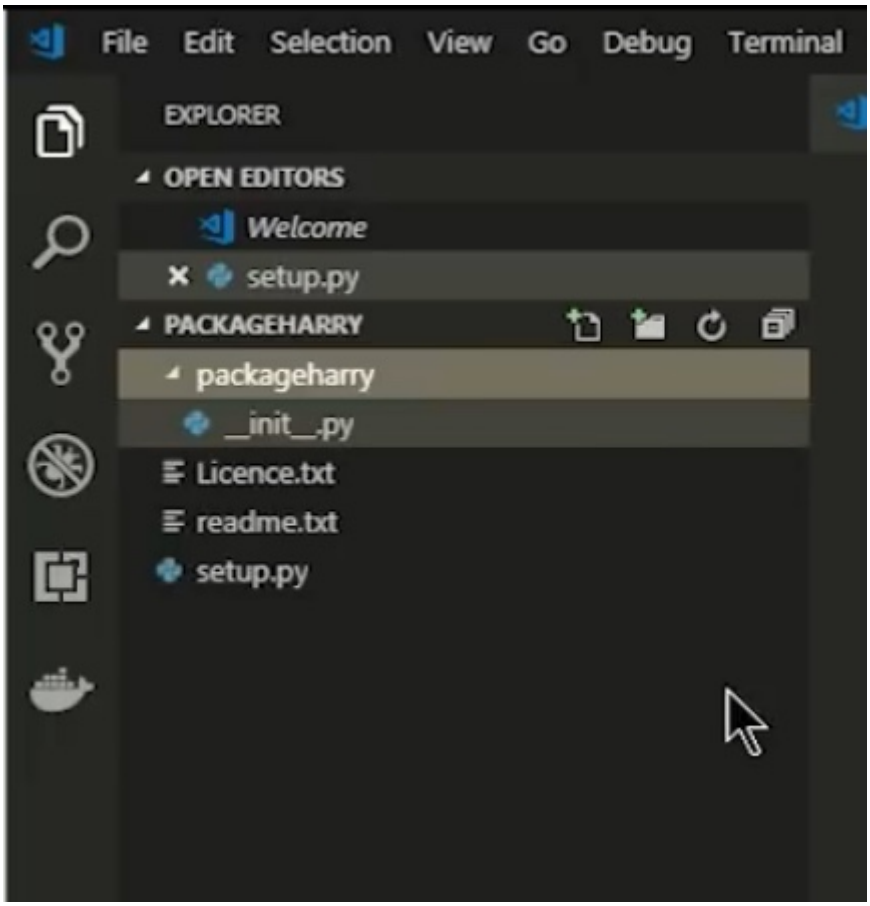
In this tutorial, we will learn how to create a Python Package Using **Setuptools** so that we can share or distribute it among others. I will walk you through all the steps so you may make your own package.

Note: We are going to use the visual studio in this tutorial. For more understanding about setting and using Visual Studio, you may refer to my ***tutorial #121*** and watch the video to learn the basics of the visual studio and its installation and use.

Python Package Using Setuptools:-

First, we are going to make a folder, and inside it, we are going to make a license and readme named text files (we can also use the .md extension to create a markdown file instead of text, which is also a sort of text-based file with better editing). These license files contain the copyright details, and the readme will contain instructions that you want to convey. Along with the text files, we are also going to make a package folder.

Now we will open our visual studio so we can make our work more simple and easier. We can open our visual studio with the same directory address by writing the command “code .” in our PowerShell. Now we will create a setup.py file, and along with it, we will also create an **init**.py file in our package folder.



The package code is present in **init.py**, and **setup.py** is used for its identification. In simple words, setup.py tells setuptools about our package.

In **init.py**, we will write all the functions we want our package to have. We can use classes too for this cause. In setup.py, we have to import the setuptools module, and we are going to use its setup function here.

```
from setuptools import setup
```

We will provide it with some basic details about our package, such as:

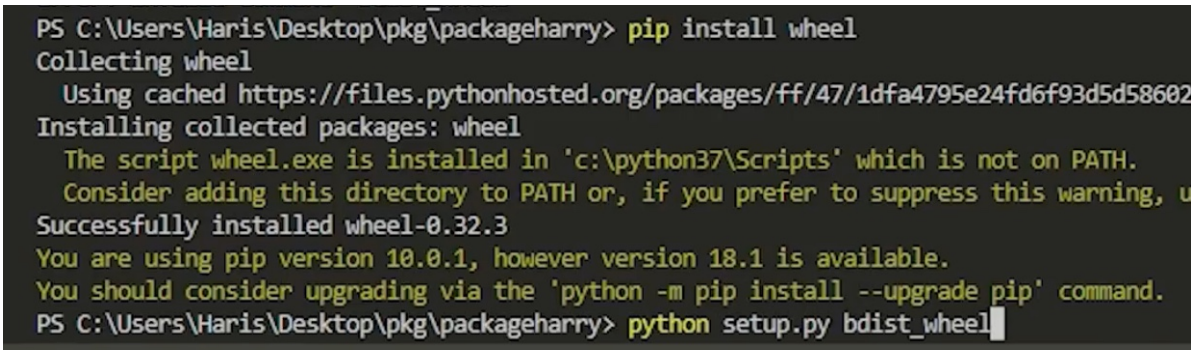
- **name:** The name of the package. We can give our package any name of our choice.
- **version:** The starting version should be 0.1 because, with any update, it automatically increases it to one decimal place.
- **description:** Here, we give a brief description of our package and its functionalities and uses.
- **long_description:** In the long description we give an explanatory description of our package
- **author:** We can specify the creator of the package here
- **packages:** Here we give the name by which we want our package to be called or imported
- **install_requires:** If our package has a prerequisite package, then we have to specify that here so both of them can work simultaneously and can perform better.

Now we are going to build our package. We are going to open our terminal window or PowerShell in the same directory, and we are going to install the wheel.

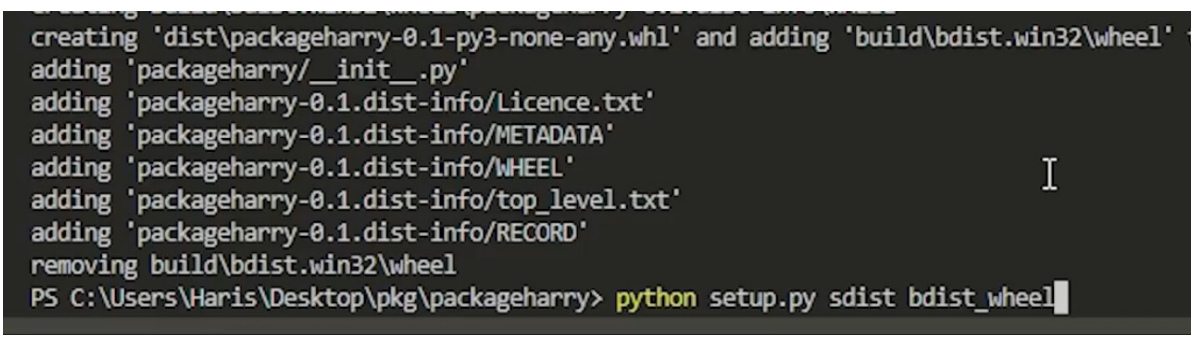
```
pip install wheel
```

and after that, we are going to make our package using the command

```
python setup.py bdist_wheel
```

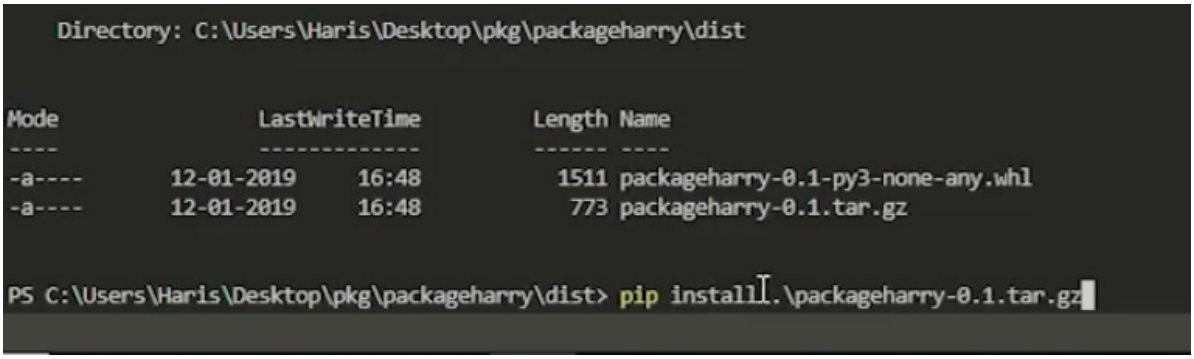



```
python setup.py sdist bdist_wheel
```



Now our package is created, and we can install it using.

```
pip install package_name
```



And we can also import it in the same way

```
Import package_name
```

Code setup.py as described/written in the video

```
from setuptools import setup
setup(name="packageharry",
version="0.3",
description="This is code with harry package",
long_description = "This is a very very long description",
author="Harry",
packages=['packageharry'],
install_requires=[])
```

Code init.py as described/written in the video

```
class Achha:
    def __init__(self):
        print("Constructor ban gaya")

    def achhafunc(self, number):
        print("This is a function")
        return number
```

####