NETZWERK ACADEMY

# PYTHON

Created By: MOHAN

## 1.1 Installation of Python:

Website Link: www.Python.org (Version Recommended 3.9.4 or higher)

Launching Python.

1) Default Python editor will be IDLE.

2) Go to CMD and type python, hit enter to get details of Python installed in your system

## 1.2 Environment Setting

1) Copy the path where python installation is done.

2) Goto ‘This PC’ and right click to select Properties

3) Goto Advance system setting, select Environment variable/ click edit and paste the path you copied

## 1.3 SDK for Code Execution

Website Link: https://www.jetbrains.com/pycharm/download/

SDK stands for

How to get code Output?

1) Goto file > Select New File > Save the file in working directory.

2) Write all your codes & go to options > click Run code to get output.

If you have # before any line of code it is called as comment line, which will not give any output.

Multiline commenting can be done using Alt+3 & Alt+4 to uncomment.

## 1.4 Python Pros and Cons

Pros of Using Python:

Easy to Use Programming Language

Easy to learn

Great for Visualizing Data

Easy to Read Language

Unmatched Flexibility

Cons of Using Python

Python being an interpreted programming language is slower than other programming languages.

The Global Interpreter Lock (GIL) of Python doesn't allow executing more than one thread at a given time. This creates significant limitations for the language.

## 1.5 Advantages of Python

1) Compiler and the Interpreter are the 2 types of Coding Software's, python is a Interpreter.

2) No Data Type Declaration Needed (Pre Memory Allocation is not applicable)

3) Scripting Language (No Complier)

4) No License

5) Unstructured

6) Very Simple & less syntax complexity

7) More Packages (Library) support.

8) Dynamic Memory (Run time memory allocation)

9) Automated Garbage collection.

10) Private Heap

Data types

2.1 Mutable and Immutable Categories

Different Data types

String, List, Tuples, Dictionary, Numerical, Set and Frozen set

What is Mutable?

Mutable sequences can be changed after creation. Some of Python's mutable data types are: lists, byte arrays, sets, and dictionaries.

What is Immutable?

Immutable data types differ from their mutable counterparts in that they cannot be changed after creation. Some immutable types include numeric data types, strings, bytes, frozen sets, and tuples.

2.2 Data Type Representation

1) String

Strings in python are surrounded by either single quotation marks, or double quotation marks.

'hello' is the same as "hello".

Assign String to a Variable

Assigning a string to a variable is done with the variable name followed by an equal sign and the string:

```
a = "Hello"

print(a)
```

## Multiline Strings

You can assign a multiline string to a variable by using three quotes:

```
a = """Lorem ipsum dolor sit amet,
consectetur adipiscing elit,
sed do eiusmod tempor incididunt
ut labore et dolore magna aliqua."""
print(a)
```

Or three single quotes:

```
a = '''Lorem ipsum dolor sit amet,
consectetur adipiscing elit,
sed do eiusmod tempor incididunt
ut labore et dolore magna aliqua.'''
print(a)
```

## Escape Sequencing in Python

While printing Strings with single and double quotes in it causes Syntax Error because String already contains Single and Double Quotes and hence cannot be printed with the use of either of these. Hence, to print such a String either Triple Quotes are used or Escape sequences are used to print such Strings.

Escape sequences start with a backslash and can be interpreted differently. If single quotes are used to represent a string, then all the single quotes present in the string must be escaped and same is done for Double Quotes.

Input:

```
# Python Program for
# Escape Sequencing
# of String

# Initial String
String1 = '''I'm a "Human"'''
print("Initial String with use of Triple Quotes: ")
print(String1)

# Escaping Single Quote
String1 = 'I\'m a "Human"'
print("\nEscaping Single Quote: ")
print(String1)

# Escaping Double Quotes
String1 = "I'm a \"Human\""
print("\nEscaping Double Quotes: ")
print(String1)

# Printing Paths with the
# use of Escape Sequences
String1 = "C:\\Python\\Human\\"
print("\nEscaping Backslashes: ")
print(String1)
```

Output:

```
Initial String with use of Triple Quotes:
I'm a "Human"

Escaping Single Quote:
I'm a "Human"

Escaping Double Quotes:
I'm a "Human"

Escaping Backslashes:
C:\Python\Human\
```

Index:

There are 2 types of Indexing

1) Forward Indexing

2) Reverse Indexing

Index always starts from 0 in Python.

1) Forward Indexing:

```
H    E    L    L    O
0    1    2    3    4
```

2) Reverse Indexing

| H | E | L | L | O |
|---|---|---|---|---|
| 0 | -4 | -3 | -2 | -1 |

String Slicing in Python

Syntax:

slice(stop)

Slice (start, stop, step)

Parameters:

start: Starting index where the slicing of object starts.

stop: Ending index where the slicing of object stops.

step: It is an optional argument that determines the increment between each index for slicing.

Return Type: Returns a sliced object containing elements in the given range only.

Interview Questions:

What are Data Types?

Name four of the main data types in Python

What does immutable mean and what three types of Python core data types are considered immutable?

What does mapping mean and what kind of data type is based on mapping?

What does sequence mean and which three types of data fall into this category?

Conditional statements

3.1 if elif and else

## if condition

syntax of ifif(condition/s):    statement1    statement2    ..........

Code Input:

num1 = 10num2 = 10**if**(num1 == num2):    print("Both the numbers are same")

Code Output:

Both the numbers are same

## Indentation

whitespace at the beginning of a line

## If & else Statement.

syntax of if elseif(condition/s):    statement1    statement2 .............else:    statement1    statement2    ..........

If & else Statement

if(condition/s):    statement1    statement2    ..........elif(condition/s):    statement1    statement2    ..........else:    statement1    statement2    ..........

## For loops

- For loops are basically used to loop over a sequence.
- Sequence can be anything like strings,Lists,Tuples,Dictionaries,Sets etc

Syntax:for item in seq:    statements....

Code Input:

str1 = "abcde"**for** item **in** str1:    print(item)

Code Output:

abcde

## 3.2 While Statement

syntaxwhile(condition/s):    statement1    statement2    ..........

## while - else

if you want to know your while loop has finished its execution and you want to do some operations then you can make use of "else"

## Break:

break keyword is used to break out of the loop or come out of the loop

Interview Questions:

What is pass in Python?

What Does The Continue Do In Python?

When Should You Use The "Break" In Python?

What Is The Difference Between Pass And Continue In Python?

What are the comparison operators in Python?

Loops and control statements

4.1 __iter__and__next__

At many instances, we get a need to access an object like an iterator. One way is to form a generator loop but that extends the task and time taken by the programmer. Python eases this task by providing a built-in method__iter__() for this task.
The__iter__() function returns an iterator for the given object (array, set, tuple, etc. or custom objects). It creates an object that can be accessed one element at a time using__next__() function, which generally comes in handy when dealing with loops.

Code Input:

```
listA = ['a','e','i','o','u']


iter_listA = iter(listA)


try:
    print( next(iter_listA))
    print( next(iter_listA))
    print( next(iter_listA))
    print( next(iter_listA))
    print( next(iter_listA))
    print( next(iter_listA)) #StopIteration error
```

except:

    pass

Code Output:

aeiou

Code Input:

# define a listmy_list = [4, 7, 0, 3]# get an iterator using iter()my_iter = iter(my_list)# iterate through it using next()# Output: 4print(next(my_iter))# Output: 7print(next(my_iter))# next(obj) is same as obj.__next__()# Output: 0print(my_iter.__next__())# Output: 3print(my_iter.__next__())# This will raise error, no items leftnext(my_iter)

Code Output:

4703Traceback (most recent call last): File "<string>", line 24, in <module> next(my_iter)StopIteration

4.2 break and continue statement

Break

The break statement takes care of terminating the loop in which it is used. If the break statement is used inside nested loops, the current loop is terminated, and the flow will continue with the code followed that comes after the loop.

Code Input:

my_list = ['Siya', 'Tiya', 'Guru', 'Daksh', 'Riya', 'Guru'] for i in range(len(my_list)):    print(my_list[i])    if my_list[i] == 'Guru':

print('Found the name Guru')        break        print('After break statement')print('Loop is Terminated')

Code Output:

SiyaTiyaGuruFound the name GuruLoop is Terminated

<u>Continue:</u>

The continue statement skips the code that comes after it, and the control is passed back to the start for the next iteration.

Code Input:

for i in range(10):        if i == 7:        continue       print("The Number is :" , i)

Code Output:

The Number is : 0The Number is : 1The Number is : 2The Number is : 3The Number is : 4The Number is : 5The Number is : 6The Number is : 8The Number is : 9

Pass:

Python pass is a null statement. When the Python interpreter comes across the across pass statement, it does nothing and is ignored.

Code Input:

def my_func():    print('pass inside function')    passmy_func()

Code Output:

pass inside function

4.3 enumerate

When using the iterators, we need to keep track of the number of items in the iterator. This is achieved by an in-built method called enumerate(). The enumerate() method adds counter to the iterable. The returned object is a enumerate object. Its syntax and parameters are described below.

Syntax:

enumerate(iterable, start=0)iterable - a sequence, an iterator, or objects that supports iterationstart – is the position in the iterator from where the counting starts.Default is 0.

Code Input:

ays= { 'Mon', 'Tue', 'Wed','Thu'}enum_days = enumerate(days)print(type(enum_days))# converting it to alistprint(list(enum_days))# changing the default counter to 5enum_days = enumerate(days, 5)print(list(enum_days))

Code Output:

[(0, 'Tue'), (1, 'Thu'), (2, 'Mon'), (3, 'Wed')][(5, 'Tue'), (6, 'Thu'), (7, 'Mon'), (8, 'Wed')]

4.4 range

Python range() function generates the immutable sequence of numbers starting from the given start integer to the stop integer. The range() is a built-in function that returns a range object that consists series of integer numbers, which we can iterate using a for loop.

Syntax:

range(start, stop, step)

start, Optional. An integer number specifying at which position to start. Default is 0 stop, Required. An integer number specifying at which position to stop (not included).

step, Optional. An integer number specifying the incrementation. Default is 1

Code Inpu:

```
# Generate numbers between 0 to 6
for i in range(6):
	print(i)
```

Code Output:

```
012345
```

4.5 else in for loop

In most of the programming languages (C/C++, Java, etc), the use of else statement has been restricted with the if conditional statements. But Python also allows us to use the else condition with for loops.

Code Input:

```
#Else block is executed in below Python 3.x program:
for i in range(1, 4):
	print(i)
else: # Executed because no break in for
	print("No Break")
```

Code Output:

```
123No Break
```

Else block is NOT executed in Python 3.x or below:

Interview Questions:

What function can generate a list of numbers?

What module is used for generating random values?

What keyword is used to skip back to the beginning of a loop?

What keyword is used to end looping completely?

What does the break keyword do?

Strings

5.1 Inbuilt Function Manipulation

Python Document link: https://docs.python.org/3/library/functions.html

Inbuilt functions are like Readymade materials available, We can take them and use them for our requirements

but we can’t change them since they are immutable every built-in function will have their own operations.

There are over 68 built-in functions available in python.

Some of the commmonly used inbuilt functions are

Inbuilt Functions, Explanation

type() = To check Data type.

len() = To check lengthof the string

Print(), prints the particularly given object abs(), gives the absolute value of a number

max(), largest of the given parameters

min(), smallest in the given parameters

pow(), gives the power of the given parameters

range(), the sequence of numbers between the given start integer to the ending integer.

round(), floating point into a specified number of decimal points

slice(), it is used to slice a list, tuple, etc

sum(), adds the given parameters and returns the added value

type(), it will return what type of object or function it is with the parameters passed to it

str(), it will return the converted string of the given object

reversed(), it reverses the sequence of the given sequence

property(), it will return the property of the attribute

id(), returns the unique integer address of the object

help(), Python has an inbuilt help system by using this help() function we can clear our doubts

hex(), converts an integer into a respective hexadecimal number hash(), returns the hash value of the object

bin(), converts given type into binary format and return value

input(), gets input from the user and returns it

open(), it is used to open a file and returns the file objects

Interview Questions:

Given a string with leading and trailing spaces, illustrate how to return a string without the leading and trailing spaces.

What is the difference between the find( ) and index( ) string methods?

Given a multiple-line string, how is a list of individual lines obtained?

What method is used to determine if a string contains any special characters or punctuation?

What method is used to determine if a string contains only numbers?

List

6.1 Inbuilt Function Manipulation

Python includes the following list functions −

1, cmp(list1, list2)
Compares elements of both lists.

2, len(list)
Gives the total length of the list.p>

3, max(list)
Returns item from the list with max value.

4, min(list)
Returns item from the list with min value.

5, list(seq)
Converts a tuple into list.

Python includes following list methods

1, list.append(obj)
Appends object obj to list

2, list.count(obj)
Returns count of how many times obj occurs in list

3, list.extend(seq)
Appends the contents of seq to list

4, list.index(obj)
Returns the lowest index in list that obj appears

5, list.insert(index, obj)
Inserts object obj into list at offset index

6, list.pop(obj=list[-1])
Removes and returns last object or obj from list

7, list.remove(obj)
Removes object obj from list

8, list.reverse()
Reverses objects of list in place

9, list.sort([func])
Sorts objects of list, use compare func if given

Interview Questions:

How To Concatenate or Join two Lists in Python ?

How To Sort a List in Python ?

What is Pop Method in List ?

What is Pop Method in List ?

How to count the number of occurrence of specific element in List ?

Tuples

7.1 Inbuilt Function Manipulation

1, cmp(tuple1, tuple2) Compares elements of both tuples.

2, len(tuple) Gives the total length of the tuple.

3, max(tuple) Returns item from the tuple with max value.

4, min(tuple) Returns item from the tuple with min value.

5, tuple(seq) Converts a list into tuple.

Code Input:

```
tuple1, tuple2 = (123, ‘xyz’), (456, ‘abc’)

print cmp(tuple1, tuple2);
print cmp(tuple2, tuple1);
tuple3 = tuple2 + (786,);
print cmp(tuple2, tuple3)
```

Code Output:

```
-1

1

-1
```

Interview Questions:

How to Sort List Of Tuples By The First Element

How to Sort List Of Tuples By Second Element

How to remove duplicate from list of tuples

How do we reverse a string in Python?

Example to convert list to tuple.

Dictionaries

8.1 Inbuilt Function Manipulation

Python includes the following dictionary functions −

1, cmp(dict1, dict2)
Compares elements of both dict.

2, len(dict)
Gives the total length of the dictionary. This would be equal to the number of items in the dictionary.

3, str(dict)
Produces a printable string representation of a dictionary

4, type(variable)
Returns the type of the passed variable. If passed variable is dictionary, then it would return a dictionary type.

Python includes following dictionary methods −

1, dict.clear()
Removes all elements of dictionary dict

2, dict.copy()
Returns a shallow copy of dictionary dict

3, dict.fromkeys()
Create a new dictionary with keys from seq and values set to value.

4, dict.get(key, default=None)
For key key, returns value or default if key not in dictionary

5, dict.has_key(key)
Returns true if key in dictionary dict, false otherwise

6, dict.items()
Returns a list of dict's (key, value) tuple pairs

7, dict.keys()
Returns list of dictionary dict's keys

8, dict.setdefault(key, default=None)
Similar to get(), but will set dict[key]=default if key is not already in dict

9, dict.update(dict2)
Adds dictionary dict2's key-values pairs to dict

10, dict.values()
Returns list of dictionary dict's values

11 clear(), Removes all the elements from the dictionary

12 copy(), Returns a copy of the dictionary

13 fromkeys(), Returns a dictionary with the specified keys and value

14 get(), Returns the value of the specified key

15 items(), Returns a list containing a tuple for each key value pair

16 keys(), Returns a list containing the dictionary's keys

17 pop(), Removes the element with the specified key

18 popitem(), Removes the last inserted key-value pair

19 setdefault(), Returns the value of the specified key. If the key does not exist: insert the key, with the specified value

20 update(), Updates the dictionary with the specified key-value pairs

21 values(), Returns a list of all the values in the dictionary

Interview Question's & Assignment:

Program to delete set of keys from Dictionary

How to change the name of key in dictionary.

How to change the value of any key in nested dictionary

How to concatenate multiple dictionary in one

How to sum all elements of Dictionary

Functions

9.1 Types of Functions & 9.2 return statement

In Python programming, as per our requirement, We can define the User defined functions in multiple ways. The following are the list of available types of functions in Python.

1. Python Function with no argument and no return value.
2. Function with no argument and with a Return value.
3. Python Function with argument and No Return value.
4. Function with argument and return value.

1. Python Function with No argument and No Return value

In this type of function in Python, While defining, declaring, or calling the function, We won't pass any arguments to them. This type of Python function won't return any value when we call them.

Whenever we are not expecting any return value, we might need some print statements as output. In such a scenario, we can use this type of functions in Python.

Code Input:def Adding():    a = 20    b = 30    Sum = a + b    print("After Calling :", Sum)Adding()

Code Output:

After Calling : 50>>> Adding()After Calling : 50

2. Function with no argument and with a Return value.

In this type of function in the Python program, we are going to calculate the multiplication of 2 integer values using the user defined function without arguments and return keyword.

Code Input:

```
# Python Function with No Arguments, and with Return Valuedef
Multiplication():    a = 10    b = 25    Multi = a * b    return Multiprint("After
Calling the Multiplication : ", Multiplication())
```

Code Output:

```
After Calling the Multiplication : 250>>> Multiplication()250
```

3. Python Function with argument and No Return value

This program for the type of function in Python allows the user to enter 2 integer values and then, We are going to pass those values to the user defined function to Multiply them.

Code Input:

```
# Python Function with Arguments, and NO Return Valuedef Multiplications(a,
b):    Multi = a * b    print("After Calling the Function:",
Multi)Multiplications(10, 20)
```

4. Python Function with argument and Return value

This type of function in the Python program allows the user to enter 2 integer values. Then, we pass those values to the user-defined function to add those values and return the value using the return keyword.

Code Input:

# Python Function with Arguments, and NO Return Valuedef Addition(a, b): Sum = a + b    return Sum# We are calling the Function Outside the Definitionprint("After Calling the Function:", Addition(25, 45))

9.3 Function Scoping

Inside of a function in Python, the scope changes.

Think about it this way: scoping in Python happens with whitespace. When we delineate the code a function contains by indenting it under a function definition, it's scope changes to a new internal scope. It has access to the variables defined outside of it, but it can't change them.

Once the function is done running, its scope goes away, as do its defined variables.

Generally, we want to be careful when using variables defined outside of our function.

Code Input:

```
>>> name = "Nina"

>>> print(f"Name outside of function: {name}")

 def try_change_name():

        name = "Max"

        print(f"Name inside of function: {name}")

try_change_name()

print(f"Name outside of function: {name}")
```

Code Output:

```
Name outside of function: Nina

Name inside of function: Max

Name outside of function: Nina
```

## 9.4 Execution of Nested Function

Scope of variables in nested function

We already know how to access a global variable form a function.
In this section, we will learn about accessing and changing the variables of an outer function from the inner one. So, first let's try to print a variable of an outer function from the inner one.

Code Input:

```
def f1(): #outer function    a = 1    def f2(): #outer function        nonlocal a        a
= 2        print (a) #prints 2    f2()    print (a) #prints 2f1()
```

Code Output:

```
2
2
```

Interview Question's & Assignments:

Name different types of Functions.

Name few usages of return statement in function.

Name few usages of function scoping.

Explain the Execution of Nested Function

Name the advantages of function.

## Exception handling

## 10.1 Types of Exception and Exception Base class

There are several exceptions available as part of Python language that are called built-in exceptions. In the same way, the programmer can also create his own exceptions called user-defined exceptions.

Some Important Built-in Exceptions:

Exception, Represents any type of exception. All exceptions are sub classes of this class.

ArithmeticError, Represents the base class for arithmetic errors like OverflowError,ZeroDivisionError,FloatingPointError.

AssertionError, Raised when an assert statement gives error.

AttributeError, Raised when an attribute reference or assignment fails.

EOFError, Raised when input() function reaches end of file condition without reading any data.

FloatingPointError, Raised when a floating point operation fails.

GeneratorExit, Raised when generator's close() method is called.

IOError, Raised when an input or output operation failed. It raises when the file opened is not found or when writing data disk is full.

ImportError, Raised when an import statement fails to find the module being imported.

IndexError, Raised when a sequence index or subscript is out of range.

KeyError, Raised when a mapping (dictionary) key is not found in the set of existing keys.

KeyboardInterrupt, Raised when the user hits the interrupt key(normally control-C or Delete).

NameError, Raised when an identifier is not found locally or globally.

NotImplementedError, Derived from 'RuntimeError'. In user defined base classes,abstract methods should raise this exception when they require derived classes to override the method.

OverflowError, Raised when the result of an arithmetic operation is to large too be represented. This cannot occur for long integers (which would rather raise 'MemoryError').

RuntimeError, Raised when an error is detected that doesn't fall in any of the other categories.

StopIteration, Raised by an iterator's next() method to signal that there are no more elements.

SyntaxError, Raised when the compiler encounters a syntax error.Import or exec statements and input() and eval() functions may raise this exception.

IndentationError, Raised when indentation is not specified properly.

SystemExit, Raised by the sys.exit() function. When it is not handled, the Python interpreter exits.

TypeError, Raised when an operation or function is applied to an object of inappropriate datatype.

UnboundLocalError, Raised when a reference is made to a local variable in a function or method, but no value has been bound to that variable.

ValueError, Raised when a built-in operation or function receives an argument that has right datatype but wrong value.

ZeroDivisionError, Raised when the denominator is zero in adivision or modulus operation.

Avoid Exceptions with Try..Except..

In order to avoid the errors coming up and stop the flow of the program, we make use of the try-except statements. The whole code logic is put inside the try block and the except block handles the cases where an exception/error occurs.

Code Input:

```
try:
    #block of code

except <Name of Exception>:
    #block of code

#Rest of the code
```

Exception handling with try, except, finally block.

Code Input:

```
print("start of the program")try:    num1 = int(input("Enter the numerator : "))
num2 = int(input("Enter the denominator : "))    print('Numerator = {} ,
Denominator = {}'.format(num1,num2))    result = num1/num2    print("result =
",result)    except ZeroDivisionError:    print("Enter a non-zero denominator")
except ValueError:    print("Please enter the numbers only")    except:
print("Something went wrong")    finally:    print("End of program")
```

Code Output:

```
start of the programEnter the numerator : 100Enter the denominator :
0Numerator = 100 , Denominator = 0Enter a non-zero denominatorEnd of
program
```

Interview Question's & Assignments:

What Are Errors and Exceptions in Python Programs?

What is Index Out of Range Error?

How Do You Handle Exceptions with Try/Except/Finally in Python?

Name some standard Python errors you know?

Name few Built-in Exceptions.

# OOP concepts

## 11.1 document string

Python documentation strings make it easy to link documentation to modules, functions, classes, and methods in Python.
There are two types of document strings.

- one-line docstrings
- multi-line docstrings

**One-line Docstrings**

One-line docstrings are used to give single-line documentation.one-line docstrings are represented by triple double-quotes.

Example:

```
def sum(a, b):
        """Returns sum of a and b."""
        return a+b
print(sum.__doc__)
```

output:

```
Returns sum of a and b.
```

**Multi-line Docstrings**

If we want to give more summary and details about function, method, or class we use multi-line docstrings.multi-line doctrings are represented by triple single quotes.

Example:

```
def sum(a, b):
        ‘’’ function name: sum
        Parameters:2
         Returning sum of two numbers’’’
        return a+b
print(sum.__doc__)
```

output:

```
function name: sum
Parameters:2
Returning sum of two numbers
```

## 11.2 Inheritance and Types

Inheritance is the capability of one class to derive or inherit the properties from another class.

There are different types of inheritance

- Single inheritance
- Multiple-level inheritance
- Multiple inheritance

## Single inheritance

When a child class inherits from only one parent class, it is called single inheritance.

Example:

```
class A:
    def __init__(self, n ):
            self.n = n
    def display(self):
        print(self.n*2)
class B(A):
    def __init__(self, roll):
            self.roll = roll
            A.__init__(self,roll)
object = B(23)
object.display()
```

output:

```
46
```

Here we are calling the parent constructor in the child constructor. If you forget to invoke the __init__() of the parent class then its instance variables would not be available to the child class.

## Multi-level inheritance

when we have more than two levels of inheritance then we call it multi-level inheritance.

Example:

```
class A(object):
    def __init__(self, name):
        self.name = name
    def getName(self):
        return self.name
class B(A):
    def __init__(self, name, age):
        A.__init__(self, name)
        self.age = age
    def getAge(self):
        return self.age
class C(B):
    def __init__(self, name, age, address):
        B.__init__(self, name, age)
        self.address = address
    def getAddress(self):
        return self.address
g = C("name", 23, "banglore")
print(g.getName(), g.getAge(), g.getAddress())
output:
name 23 banglore
```

Here class b is inheriting class a and class c is inheriting class b.

**Multiple inheritance**

When one child class inherits from multiple parent classes then we call it as Mutiple inheritance.

Example:

```
class A(object):
    def __init__(self):
        self.st1 = "A"
        print("1st class")

class B(object):
    def __init__(self):
        self.st2= "B"
        print("2nd class")
class C(A, B):
    def __init__(self):
        A .__init__(self)
        B.__init__(self)
        print("Derived class")

    def printStrs(self):
        print(self.st1, self.st2)
ob = C()
ob.printStrs()
output:
1st class
2nd class
Derived class
A B
```

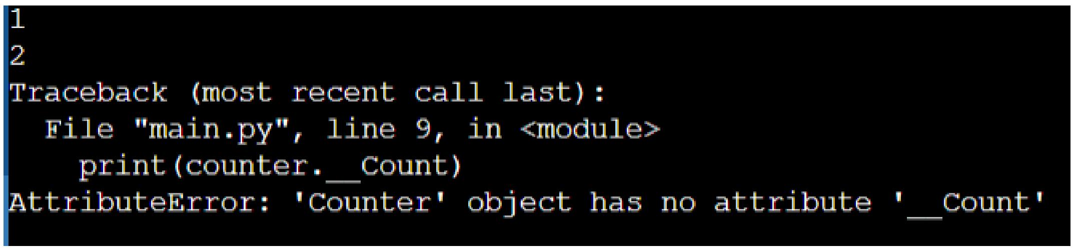
## 11.3 Data Hiding(Access Specifier)

Data hiding isolates the client from a part of the program implementation. We can perform data hiding in Python using the __ double underscore before the prefix.

This makes the class members private and inaccessible to the other classes.
Example:

```
class Counter:
  __Count = 0
  def count(self):
    self.__Count += 1
    print(self.__Count)
counter = Counter()
counter.count()
counter.count()
print(counter.__Count)
```

output:



```
1
2
Traceback (most recent call last):
  File "main.py", line 9, in <module>
    print(counter.__Count)
AttributeError: 'Counter' object has no attribute '__Count'
```

## 11.4 Decorators

Using decorators we can add additional functionality to function or method without modifying it.
Example:

```
def check(func):
	def inside(a,b):
		if b==0:
			print("can not divide by 0")
			return
		func(a,b)
	return inside
```

```
@check
def div(a,b):
   return a/b
print(div(10,0))
```

output:

```
can not divide by 0
None
```

There are some decorators for methods in the class. they are @classmethod,@staticmethod, and @instancemethod.

## 11.5 Polymorphism

The word polymorphism means having many forms. In programming, polymorphism means the same function name (but different signatures) being used for different types.

**Example of inbuilt polymorphic functions :**

```
print(len("python"))
print(len([1,2,3]))
```

**Examples of user-defined polymorphic functions :**

```
def add(x, y, z = 0):
        return x + y+z
print(add(2, 3))
print(add(2, 3, 4))
```

In python, method overloading is not possible only method overriding is possible. This process of re-implementing a method in the child class is known as **Method Overriding**. It is useful in cases where the method inherited from the parent class doesn’t quite fit the child class. In such cases, we re-implement the method in the child class.

Example:

```
class A:
    def fun1(self):
      print(‘A1’)
   def fun(self):
      print('A')
```

```
class B(A):
    def fun(self):
        print('B')
a=A()
b=B()
a.fun()
b.fun()
b.fun1()
```

output:

```
A
B
A1
```

## 11.6 Method Resolution Order

The method resolution order controls how the base classes are searched when a method is called in Python. The method or attribute is initially looked for within a class, and then the inheritance order is followed from bottom to top. Class linearization is another name for this structure, and MRO stands for a set of rules (Method Resolution Order). While inheriting from another class, the interpreter needs the means to resolve the methods that are invoked via an instance.

Example:

```
class A:
    def fun(self):
        print('A')
class B():
    def fun(self):
        print('B')
class C(A,B):
    pass
c=C()
c.fun()
```

output:

```
A
```

In the above example, we got A as output because we inherited class A from class C.

Example:
```
class A:
    def fun(self):
        print('A')
class B():
    def fun(self):
        print('B')
class C(B,A):
    pass
c=C()
c.fun()
```

output:
```
B
```

The difference between the first code and the second code is class B is inherited first in the second code that is why we got output B here.

# File Operation (txt file)

## 16.1 Context Manager

Context managers allow you to allocate and release resources precisely when you want to. The most widely used example of context managers is the with a statement. Suppose you have two related operations which you'd like to execute as a pair, with a block of code in between. Context managers allow you to do specifically that. For example:

```
with open('some_file', 'w') as opened_file:
    opened_file.write('Hello!')
```

**opening a file:**

syntax:

```
File_object = open(r"File_Name","Access_Mode")
```

**Access modes for different functions:**

**Read Only (‘r’):** Open text file for reading. The handle is positioned at the beginning of the file. If the file does not exists, raises the I/O error. This is also the default mode in which a file is opened.

**Write Only (‘w’):** Open the file for writing. For the existing file, the data is truncated and over-written. The handle is positioned at the beginning of the file. Creates the file if the file does not exist

**Append Only (‘a’)**: Open the file for writing. The file is created if it does not exist. The handle is positioned at the end of the file. The data being written will be inserted at the end, after the existing data.

For reading data from a text file we use three functions

**read:**
**syntax:**
fileobject.read([n])

for n reads at most n bytes. If we did not provide n it will read an entire text file

**readline:**
**syntax:**
fileobject.readline([n])

for n it reads at most n bytes. However, does not reads more than one line, even if n exceeds the length of the line.

**readlines:**
**syntax:**
it reads all line

the syntax for writing data to a text file:

fileobject.write(“text here”)

tell() is used to return the current file position after reading the first line.

Seek() is used to change the current file position to 4, and return the rest of the line.

# Date Time Manipulation

## 17.1 import datetime and time

For importing datetime we use “import datetime”
For importing time we use “import time”

## 17.2 change time format

datetime.strftime(format) is syntax to change format of date and time.
Example:

```
from datetime import datetime
now = datetime.now()
date_time = now.strftime("%Y-%m-%d %H:%M:%S")
print(date_time)
```

```
2022-05-19 09:38:58
```

## 17.3 Adding Hours, minutes and Seconds

We use hour for adding hours and minute for minute and second for adding seconds.

Example:

```
from datetime import datetime
now = datetime.now()
print(now.hour)
print(now.minute)
print(now.second)
```

output:

```
9
45
39
```