KAMARAJ COLLEGE
SELF FINANCING COURSES
(Reaccredited with "A+" Grade by NAAC)
(Affiliated to Manonmaniam Sundaranar University,Tirunelveli)
THOOTHUKUDI – 628 003.

STUDY MATERIAL FOR BCA

PYTHON PROGRAMMING

IV - SEMESTER

ACADEMIC YEAR 2022 - 2023

PREPARED BY

DEPARTMENT OF COMPUTER SCIENCE
KAMARAJ COLLEGE (SF),
THOOTHUKUDI.

UNIT-I
INTRODUCTION TO PYTHON

Introduction to Python:

Python is a popular programming language. It was created by Guido van Rossum, and released in 1991.

It is used for:

- Web development (server-side),
- Software development,
- Mathematics,
- System scripting.

Features of Python:

1. **Simple**

   Python is a simple and minimalistic language. Reading a good Python program feelsalmost like reading English language.

2. **Easy to learn**

   Python uses very few keywords. Python has an extraordinarily simple syntax and simple program structure.

3. **Open Source**

   There is no need to pay for Python software. Python is FLOSS (Free/Library and Open Source Software). Its source can be read, modified and used in programs as desired by the programmers.

4. **High level language**

   When you write programs in Python, you never need to bother aboutthe low-level details such as managing the memory used by your program, etc.

5. **Dynamically typed**

   Python provides IntelliSense. IntelliSense to make writing your code easier and more error-free. IntelliSense option includes statement completion, which provides quick access to valid member function or variables, including global, via the member list. Selecting from the list inserts the member into your code.

6. **Portable**

   Due to its open-source nature, Python has been ported to (i.e. changed to make it work on) many platforms. All your Python programs can work on any of these platforms without requiring any changes at all if you are careful enough to avoid any system-dependent features.

7. **Platform independent**

   When a Python program is compiled using a Python compiler, it generates byte code. Python's byte code represents a fixed set of instructions that run on all operating systems and hardware. Using a Python Virtual Machine (PVM), anybody can run these byte code instructions on any computer system. Hence, Python programs are not dependent on any specific operating system.

8. **Procedure and Object Oriented**

   Python supports procedure-oriented programming as well as object-oriented programming. In procedure-oriented languages, the program is built around procedures or functions which are nothing but reusable pieces of programs. In object- oriented languages, the program is built around objects which combine data and functionality.

FLAVORS OF PYTHON

Flavors of Python simply refers to the different Python compilers. These flavors are useful to integrate various programming languages into Python. Let us look at some of these flavors:

i. **C Python**

   C Python is the Python compiler implemented in C programming language. In this, Python code is internally converted into the **byte code** using standard C functions. Additionally, it is possible to run and execute programs written in C/C++ using CPython compiler.

ii. **J Python**

   Earlier known as **J Python**. Jython is an implementation of the Python programming language designed to run on the Java platform. Jython is extremely useful because it provides the productivity features of a mature scripting language while running on a JVM.

iii. **PyPy**

This is the implementation using Python language. PyPy often runs faster than C Python because PyPy is a just-in-time compiler while CPython is an interpreter.

iv. **Iron Python**

Iron Python is an open-source implementation of the Python programming language which is tightly integrated with the .NET Framework.

v. **Ruby Python**

Ruby Python is a bridge between the Ruby and Python interpreters. It embeds a Python interpreter in the Ruby application's process using FFI (Foreign Function Interface).

vi. **Python xy**

Python(x,y) is a free scientific and engineering development **software** for numerical computations, data analysis and data visualization based on Python.

vii. **Stockless Python**

Stack less Python is a Python programming language interpreter. In practice, Stack less Python uses the C stack, but the stack is cleared between function calls

viii. **Anaconda Python**

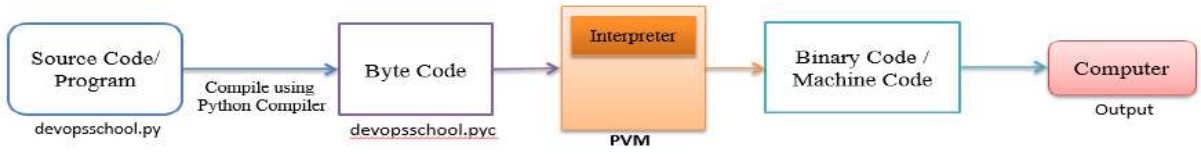
Anaconda is a free and open-source distribution of the Python and R programming languages for scientific computing, that aims to simplify package management and deployment. Package versions are managed by the package management system conda.

Python Virtual Machine

Python Virtual Machine (PVM) is a program which provides programming environment.

The role of PVM is to convert the byte code instructions into machine code so the computer can execute those machine code instructions and display the output.
Interpreter converts the byte code into machine code and sends that machine code to the computer processor for execution.

MEMORY



MANAGEMENT IN PYTHON

Garbage Collector

Garbage Collection is how the memory is freed when not use and how it can be made available for other objects. Python deletes the objects that are no longer in use. This is what we call Garbage Collection. The garbage collector initiates its execution with the program and is activated if the reference count drops to zero.

Static Memory Allocation – Stack

In static memory allocation, the memory is allocated at the compile time. The Stack data structure stores the static memory.

Static int x=2;

Dynamic Memory Allocation – Heap

In dynamic memory allocation, the memory is allocated at the run time. The Heap stores the dynamic memory. It frees up the memory space if the object is no longer needed.

x = [0]*2

As we discussed above, the garbage collector initiates its execution with the program and is activated if the reference count drops to zero.

Reference Count

The Python garbage collector initiates its execution with the program and is activated if the reference count drops to zero. Let us see when the reference count increase or decreases.

The reference count value increase when −

- When a new name is assigned or in a dictionary or tuple, the reference count increases its value.
- If we reassign the reference to an object, the reference counts decrease its value.
- The reference count value decreases when −
- The value decreases when the object's reference goes out of scope.
- The value decreases when an object is deleted.

Therefore, reference counting is actually how many times other objects reference an object. With that, The de-allocation occurs when the reference count drops to zero.

COMPARISON BETWEEN C AND PYTHON

C-Language	Python
Procedure Oriented Programming Language	Object Oriented Programming Language
Program execute faster	Program execute slower compare to C
Declaration of variable is compulsory	Type declaration is NOT required.
Type discipline is static and weak	Type discipline is dynamic and string
Pointer is available	No pointer
Does not have exception handling	Handles exceptions
It has while, for and do-while loops	It has while and for loops
It has switch-case statement	It does not have switch-case statement
The variable in for loop does not incrementedautomatically.	The variable in the for loop incremented automatically.
Memory allocation and de-allocation is not automatic	Memory allocation and de-allocation is done automatically by PVM.
It does not contain a garbage collection	Automatic garbage collection
It supports single and multi dimensional arrays	It supports only single dimensional array. Implement multi dimensional array we shouldуse third party application like

	numpy.
The array index should be positive integer.	Array index can be positive and negative integer. Negative index represents location from the end of the array.
Indentation of statements in not necessary	Indentation is required to represents a blockof statements.
A semicolon is used to terminate the Statements and comma is used to separateexpressions / variables.	New line indicates end of the statements and semicolon is used as an expression separator.
It supports in-line assignment	It does not supports in-line assignment.

COMPARISON BETWEEN JAVA AND PYTHON

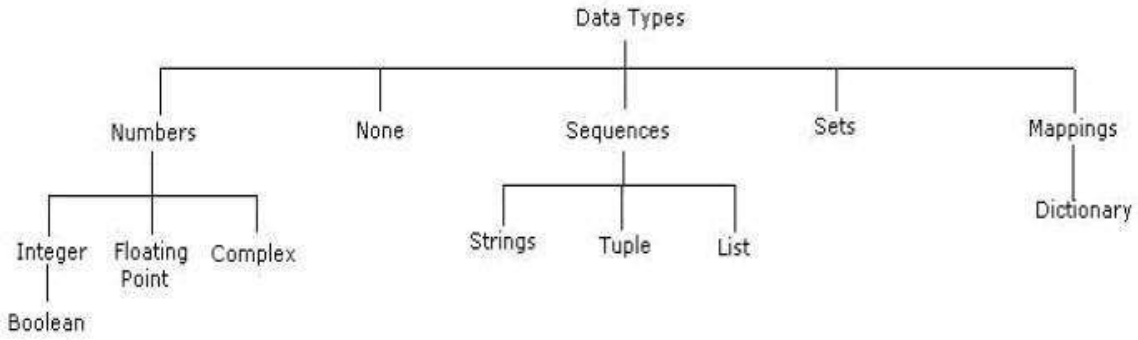
Java	Python
Pure Object-Oriented Programming Language	Both Object-Oriented and Procedure-Orientedprogramming language
Java programs are verbose.	Python programs are concise and compact.
Declaration of variable is compulsory	Type declaration is NOT required.
Type discipline is static and weak	Type discipline is dynamic and string
It has while, for and do-while loops	It has while and for loops
It has switch-case statement	It does not have switch-case statement
The variable in for loop does not incrementedautomatically.	The variable in the for loop incremented automatically.
Memory allocation and de-allocation is automatically by JVM	Memory allocation and de-allocation is doneautomatically by PVM.
It supports single and multi-dimensional arrays	It supports only single dimensional array. Implement multi-dimensional array we should use third party application like numpy.
The array index should be positive integer.	Array index can be positive and negative integer. Negative index represents

	location from the end of the array.
Indentation of statements in not necessary	Indentation is required to represents a blockof statements.
A semicolon is used to terminate the statements and comma is used to separate expressions / variables.	New line indicates end of the statements and semicolon is used as an expression separator.
The collection objects like stack, linked list orvector but not primitive data types like int, float, char etc.,	The collection objects like lists and dictionariescan store objects of any type including numbers and lists.

DATA TYPES IN PYTHON

A data type represents the type of data stored into a variableor memory. There are 5 different data types are:

* None type
* Numeric type
* Sequences
* Sets
* Dictionary



None data type: The none data type represents an object that does not contain any value. In java language it is called "NULL" object. But in Python it is called as "none". In Python maximum of only one 'none'object is provided. If no value is passed to the function, then the default value will be taken as 'none'.

Numeric data type

The numeric type represents numbers. There are 3 subtypes:

* int
* float
* complex

Int data type

The int data type represents integer number (Whole number). An integer number is number without fraction. Integers can be of any length, itis only limited by the memory available.

E.g. a=10 b=-29

Float data type

The float data type represents floating point number. Afloating point number is a number with fraction. Floating point numbers can alsobe written in scientific notation using exponentiation format.

A floating point number is accurate up to 15 decimal places. Integer and floating points are separated by decimal points.

Complex data type:

A complex number is number is written in the form ofx +yj or x+yJ. Here x is the real part and y is the imaginary part.

We can use the **type()** function to know which class a variable or a value belongs to and the **isinstance()** function to check if an object belongs to a particular class.

E.g.

a = 5

print(a, "is of type", type(a))b = 2.0

print(a, "is of type", type(b))

SEQUENCES

A sequence represents a group of items or elements. There aresix types of sequences in Python. Important sequences as follows,

* **str**

* **list**
* **tuple**

str data type : The str represents string data type. A string is a collection of character enclosed in single or double quotes. Both are valid.

E.g. str=”kvn” # str is name of string variable

str=’vedish’ # str is name of string variable

Triple double quote or triple single quotes are used to embed a string in a another string (Nested string).

str=”””This is ‘str data type’ example”””

print(str) # output is : This is ‘str data type’ example

The [] operator used to retrieve specified character from the string. Thestring index starts from 0. Hence, str[0] indicates the 0^{th} character in the string.

e.g str=” SSCASC Tumkur”

print(str) # it display - SSCASC Tumkur

print(str[0]) # it display - G

list data type

A List is a collection which is ordered and changeable. It allows duplicate members. A list is similar to array. Lists are represented by square brackets [] and the elements are separated by comma.

The main difference between a list and an array is that a list can store different data type elements, but an array can store only one type ofelements. List can grow dynamically in memory but the size of array is fixed and they cannot grow dynamically.

e.g. list=[10,3.5,-20, “SSCASCT”,’TUMKUR’] # create a list

print(list) # it display all elements in the list : 10,3.5,-20,

“SSCASCT”,’TUMKUR’

Tuple data type

A tuple is similar to list. A tuple contains group of elements which can be different types. The elements in the tuple are separated by commas and enclosed in parentheses (). The only difference is that tuples are immutable. Tuples once created cannot bemodified. The tuple cannot change dynamically. That means a tuple can be treated as read-only list.

e.g. tpl=(10,3.5,-20, “SSCASCT”,’TUMKUR’) # create a tuple

print(tpl) # it display all elements in the tuple : 10,3.5,-20,

“SSCASCT”,’TUMKUR’

Sets

Set is an unordered collection of unique items and un-indexed. The order of elements is not maintained in the sets. A set does not accept duplicate elements. Set is defined by values separated by comma inside braces { }.

There are two sub types in sets:

* Set data type
* Frozen Set data type

Set data type: To create a set, we should enter the elements separated bycomma inside a curly brace.

e.g. s = {10,30, 5, 30,50}

print(s) # it display : {10,5,30,50}

In the above example, it displays un-orderly and repeated elements onlyonce, because set is unordered collection and unique items.

We can use set() to create a set asK=set(“kvn”)

Print(K) # it display : “kvn”

Frozen set data type

Frozen set is just an immutable version of a Python set object. While elements of a set can be modified at any time, an elementof frozen set remains the same after creation. Due to this, frozen sets can beused as key in Dictionary or as element of another set.

Dictionary: A dictionary is an unordered collection, changeable and indexed. In Python dictionaries are written with curly brackets, and theyhave keys and values. String operations:

OPERATORS

* Operators are special symbols which represents computation. They are applied on operand(s), which can be values or variables.
* Same operator can behave differentlyon different data types. Operators when applied on operands form an expression.
* Operators are categorized as Arithmetic, Relational, Logical and Assignment. Value and variables when used with operator are known as operands.

1. Arithmetic Operators:

Symbol	Description	Example-1	Example-2
+	Addition	>>> 5 + 6 11	>>>'SSCASCT'+'BCA' SSCASCTBCA
-	Subtraction	>>>10-5 5	>>>5 – 6 -1
*	Multiplication	>>> 5*630	>>>'SSCASCT' * 2 SSCASCTSSCASCT
/	Division	>>> 10 / 5 2	>>>5 /2.0 2.5
%	Remainder / Modulo	>>> 5 % 2 1	>>>15%5 0
**	Exponentiation	>>> 2**38	>>>2**8256
//	Integer Division	>>> 7.0 // 2 3.0	>>>3//21

2. Relational Operators:

Symbol	Description	Example-1	Example-2
<	Less than	>>> 7<10 True	>>> 'SSCASCT' <'BCA' False
>	Greater Than	>>> 7 >10 False	>>>'SSCASCT' > 'BCA' True
<=	Less than or equal to	>>> 7<=10 True	>>>'SSCASCT' <='BCA' False
>=	Greater than or equal to	>>> 7>=10 False	>>>'SSCASCT'>='BCA' True
!= , <>	Not equal to	>>> 7!=10 True	>>>'SSCASCT'!= 'sscasct' True
==	Equal to	>>> 7==10 False	>>>'SSCASC' =='SSCASC' True

3. Logical Operators:

Symbol	Description	Example-2
or	If any one of the operand is true, then condition becomes TRUE	>>> 7<=10 or 7 ==10 True
and	If both the operands are true, then the condition becomes TRUE	>>>7<10 and 7 >20 False

not	Reverse the state of operand / condition	>>> not 7<10False

4. Assignment Operator:

Symbol	Description	Example-1
=	Assigned values from right side operands to left variable. the left operand	>>> x=10 10

Bitwise Operator: a bit is the smallest unit of data storage and it can have only one of the two values, 0 and 1. Bitwise operators works on bits and perform bit-by-bit operation.

Symbol	Description	Example
|	Performs binary OR operation	5 | 3 gives 7
&	Performs binary AND operation	5 & 3 gives 1
~	Performs binary XOR operation	5 ^ 3 gives 6
^	Performs binary one's complement operation	~5 gives -6
<<	**Left shift operator**: The left-hand side operand bit is moved left by the number specified on the right-handside (Multiply by 2)	0010 << 2 gives 8
>>	**Left shift operator**: The left-hand side operand bit ismoved left by the number specified on the right-hand side (Divided by 2)	100 << 2 gives 1

Membership operators

Python has membership operators, which test for membership in a sequence, such as strings, lists or tuples. There are two membership operators are:

Symbol	Description	Example
in	Returns True if the specified operand is found in thesequence	>>> x = [1,2,4,6,8] >>> 3 in xfalse
Not in	Returns True if the specified operand is found in thesequence	>>> x = [1,2,4,6,8] >>> 3 not in xtrue

1. Identity operator: Identity operators compare the memory locations of two objects. There are two Identity operators are:

Symbol	Description	Example	Example

is	Returns True if two variables point to the same object and False, otherwise	>>>X=10
		>>>Y=10
		>>> X is Y
		true
			>>> x=[1,2,3]
			>>> y=[1,2,3]
			>>> x is yfalse

is not	Returns False if two variables point to thesame object and True, otherwise	>>>X=10
		>>>Y=10
		>>> X is not Yfalse
			>>> x=[1,2,3]
			>>> y=[1,2,3]
			>>> x is not y true

UNIT-II

INPUT AND OUTPUT:

CONTROL STATEMENTS

If Statements

One-way if statement executes the statements if the condition is true. The syntax fora one-way if statement is:

if boolean-expression:

Statement #Note that the statement(s) must be indented

- The reserved word if begins a if statement.
- The condition is a Boolean expression that determines whether or not the body will beexecuted. A colon (:) must follow the condition.
- The block is a block of one or more statements to be executed if the condition is true.

The statements within the block must all be indented the same number of spaces from the left. The block within an Example: To demonstrate simple if

#Get two integers from the user

Dividend = int(input('Please enter the number to divide: '))

Divisor = int(input('Please enter dividend: ')) # If possible, divide them and report the resultif divisor != 0:

Quotient = dividend/divisor

Print(dividend, '/', divisor, "=", quotient)print('Program finished')

Output

Please enter the number to divide: 4Please enter dividend: 5

4 / 5 = 0.8

Program finished

>>>

If-else statements

A two-way **if-else** statement decides which statements to execute based on whetherthe condition is true or false.

The syntax for a two-way **if-else** statement:

if boolean-expression:

Statement(s) #for-the-true-case ,the statement(s) must be indented

Else: statement(s) #for-the-false-case

Example: to demonstrate if else

Percent=float(input("enter percentage"))if percent >= 90.0:

Print ("congratulations, you got an a") print ("you are doing well in this class")

Else:

Print ("you did not get an a")

Print ("see you in class next week")

Nested if statements.

A series of tests can written using nested if statements.

Example: Nestedif percent=float(input("Enter Percentage")) if (percent >= 90.00):

Print ('congratuations, you got an A')else:

If (percent >= 80.0): print ('you got a B')

Else:

If (percent >= 70.0): print ('you got a C')

Else:

Print ('your grade is less than a C')

If_elif_else Statement

In Python we can define a series of conditionals (multiple alternatives) using if for the first one, elif for the rest, up until the final (optional) else for anything not caught by the other conditionals.

Example:If_elif_else score=int(input("Enter Score"))if score >= 90.0:

grade = 'A' elif score >= 80.0:

grade = 'B' elif score >= 70.0:

grade = 'C' elif score >= 60.0:

grade = 'D'else:

grade = 'F' print("Grade=",grade)

Using **else if** instead of **elif** will trigger a syntax error and is not allowed.

Loops

It is one of the most basic functions in programming; loops are an important in every programming language. Loops enable is to execute a statement repeatedly which are referred to as iterations. (A loop is used to tell a program to execute statements repeatedly).

The simplest type of loop is a **while** loop.

The syntax for the while loop is:

while loop-continuation-condition:# Loop body

Statement(s)# Note that the statement(s) must be indented

```
i = initialValue # Initialize loop-control
                 variable
while i <
  endValue:#
   Loop body
```

Example1: To demonstrate while count = 0#Program to print "Programming is fun!" for 10 times

while count < 10: print("Programming is fun!")count = count + 1

The for Loop

A for loop iterates through each statements in a sequence for exactly know many times the loop body needs to be executed, so a control variable can be used to count the executions. A loop of this type is called a counter-controlled loop. In general, the loop can be written

as follows:

for i in range(initialValue, endValue):

Loop body #Note that the statement(s) must be indented

In general, the syntax of a for loop is:

for var in sequence:# Loop body

The function **range(a, b)** returns the sequence of integers **a**, **a + 1**, …, **b-2**, and **b- 1.** The **range** function has two more versions. You can also use **range(a)** or **range(a, b,k)**. **range(a)** is the same as **range(0, a). k** is used as **step value** in **range(a, b, k)**. The first number in the sequence is **a**. Each successive number in the sequence will increase by the step value **k**. **b** is the limit. The last number in the sequence must be less than **b**.

Break and Continue in Loops

break statement:

When a break statement executes inside a loop, control flow comes out of the loop immediately:

Example:to demonstrate **break**

i = 0

while i < 7:

print(i) if i == 4:

print("Breaking from loop")

break

i += 1

The loop conditional will not be evaluated after the break statement is executed. Note that break statements are only allowed inside loops. A break statement inside a function cannot be used to terminate loops that called that function.

Executing the following prints every digit until number 4 when the break statement is met and the loop stops:

Output

01234

Breaking from loop

Break statements can also be used inside for loops, the other looping constructprovided by Python:

Example:

for i in (0, 1, 2, 3, 4):print(i)

if i == 2:

break

Executing this loop now prints:

012

Note that 3 and 4 are not printed since the loop has ended.

Continue statement

A continue statement will skip to the next iteration of the loop bypassing the rest of the current block but continuing the loop. Continue can only used inside loops:

Example to demonstrate continue

for i in (0, 1, 2, 3, 4, 5):if i == 2 or i == 4:

continue

print(i)

Note that 2 and 4 aren't printed, this is because continue goes to the nextiteration instead of continuing on to print(i) when i == 2 or i == 4.

ARRAYS

An array is a data structure that stores values of same data type. In Python, this is the main difference between arrays and lists.

While python lists can contain values corresponding to different data types, arrays in python can only contain values corresponding to same data type.

To use arrays in python language, you need to import the standard array module. This is because array is not a fundamental data type like strings, integer etc. Here is how you can import array module in python:

From array import *

Once you have imported the array module, you can declare an array. Here is how you do it:

Array Identifier Name = array(type code, [Initializers])

Type code	Details
B	Represents signed integer of size 1 byte
B	Represents unsigned integer of size 1 byte
C	Represents character of size 1 byte
u	Represents unicode character of size 2 bytes
h	Represents signed integer of size 2 bytes
H	Represents unsigned integer of size 2 bytes
i	Represents signed integer of size 2 bytes
I	Represents unsigned integer of size 2 bytes
w	Represents unicode character of size 4 bytes
1	Represents signed integer of size 4 bytes
L	Represents unsigned integer of size 4 bytes
f	Represents floating point of size 4 bytes
D	Represents floating point of size 8 bytes

Example of an array containing 5 integers:

from array import *

my_array = array('i', [1,2,3,4,5])for i in my_array:

print(i) #output:1,2,3,4,5

Some built-in array methods:

Append any value to the array using append() method my_array = array('i', [1,2,3,4,5]) my_array.append(6)

array('i', [1, 2, 3, 4, 5, 6])

Note that the value 6 was appended to the existing array values.

Insert value in an array using insert() method my_array = array('i', [1,2,3,4,5])my_array.insert(0,0)

#array('i', [0, 1, 2, 3, 4, 5])

In the above example, the value 0 was inserted at index 0. Note that the first argumentis the index while second argument is the value.

Extend python array using extend() method my_array = array('i', [1,2,3,4,5]) my_extnd_array = array('i', [7,8,9,10])my_array.extend(my_extnd_array)

array('i', [1, 2, 3, 4, 5, 7, 8, 9, 10])

We see that the array my_array was extended with values from my_extnd_array.

Remove any array element using remove() my_array = array('i', [1,2,3,4,5]) my_array.remove(4)

array('i', [1, 2, 3, 5])

We see that the element 4 was removed from the array.

Remove last array element using pop() method

pop removes the last element from the array.

my_array = array('i', [1,2,3,4,5])

my_array.pop()

array('i', [1, 2, 3, 4])

So we see that the last element (5) was popped out of array.

Fetch any element through its index using index()

index() returns first index of the matching value. Remember that arrays are zero- indexed.

my_array = array('i', [1,2,3,4,5])print(my_array.index(5))

#output: 5

my_array = array('i', [1,2,3,3,5])print(my_array.index(3))

#output: 3

Note in that second example that only one index was returned, even though the value exists twice in the array

Reverse a python array using reverse() method

The reverse() method reverses the array. my_array = array('i', [1,2,3,4,5]) my_array.reverse()

array('i', [5, 4, 3, 2, 1])

Sort a python array using sort() method

from array import * my_array = [1,20,13,4,5]my_array.sort() print(my_array) #output:1,4,5,13,20

Multi-Dimensional Array

An array containing more than one row and column is called multidimensional array. It is also called combination of several 1D arrays.2D array is also considered as matrix.

A=array([1,2,3,4])# create 1D array with 1 row B=array([1,2,3,4],[5,6,7,8]) create 2D array with 2 row

Example:2D_array

from numpy import* a=array([[1,2,3],[4,5,6],[7,8,9]])

print(a)#Prints 2D array as rows print("2D Array Element wise Printing")for i in range(len(a)):

for j in range(len(a[i])):

print(a[i][j],end=' ')#Prints array element wiseprint(end='\n')

print(end='\n')

#2D array As matrix by using matrix funprint("Matrix printing")

a=matrix('1 2 3; 4 5 6 ; 7 8 9')print(a)

Output

[[1 2 3]

[4 5 6]

[7 8 9]]

2D Array Element wise Printing1 2 3

4 5 6

7 8 9

Matrix printing[[1 2 3]

[4 5 6]

[7 8 9]]

>>>

Matrix in Numpy

In python we can show matrices as 2D array. In numpy, a matrix is considered as specialized 2D array. It has lot of built in operators on 2D matrices. In numpy, matrix is created using the following syntax.

Matrix_name=matrix(2D array or string)

Eg. a=matrix('1 2 3;4 5 6;7 8 8')

Matrix addition, multiplication and division.

We can use arithmetic operators like +, -,* ,/ to perform different operations on matrices.

Example: Matrix_Operation

from numpy import* a=matrix('4 4 4;4 4 4;4 4 4')

b=matrix('2 2 2;2 2 2;2 2 2')print("Printing A matrix") print(a)

print("Printing B matrix")print(b)

print("Printing Addition of two matrix")c=a+b #matrix addition

print(c)

print("Printing Multplication of two matrix")c=a*b #matrix addition

print(c)

print("Printing Division of two matrix")c=a/b #matrix addition

print(c)

Output

Printing A matrix[[4 4 4]

[4 4 4]

[4 4 4]]

Printing B matrix[[2 2 2]

[2 2 2]

[2 2 2]]

Printing Addition of two matrix

[[6 6 6]

[6 6 6]

[6 6 6]]

Printing Multplication of two matrix[[24 24 24]

[24 24 24]

[24 24 24]]

Printing Division of two matrix[[2. 2. 2.]

[2. 2. 2.]

[2. 2. 2.]]

>>>

Enter rows,col: 2 2

Enter matrix elements:1 2 3 4The original matrix

[[1 2]

[3 4]]

Printing Transpose of matrix[[1 3]

[2 4]]

>>>

UNIT-III

STRING AND CHARACTERS

Slicing String

Python slicing is about obtaining a sub-string from the given string by slicing it respectively from start to end.

How String slicing in Python works

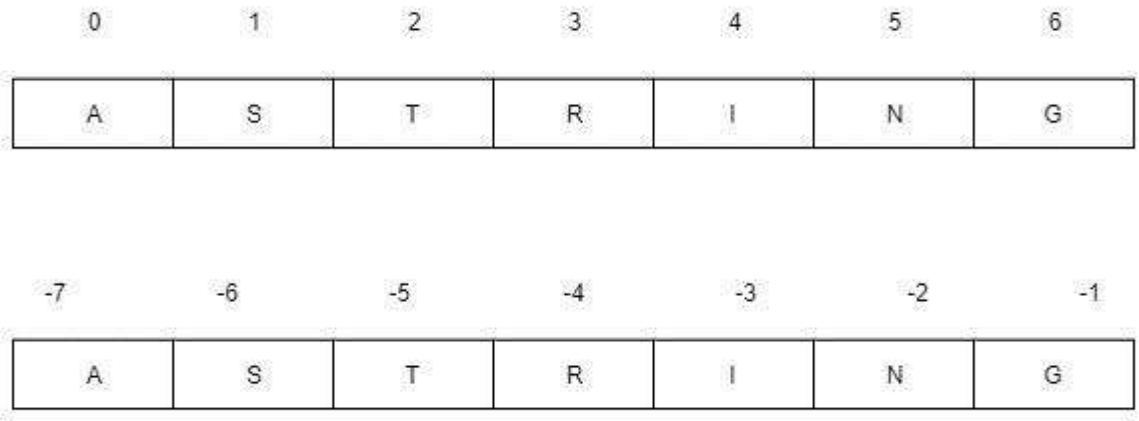
For **understanding slicing** we will use different methods, here we will cover 2 methods of string slicing, the one is using the in-build slice() method and another using the **[:] array slice**. String slicing in Python is about obtaining a sub-string from the given string by slicing it respectively from start to end.

Python slicing can be done in two ways:

* Using a slice() method
* Using array slicing [: :] method

Index tracker for positive and negative index

String indexing and slicing in python. Here, the Negative comes into consideration when tracking the string in reverse.



Method 1: Using slice() method

The slice() constructor creates a slice object representing the set of indices specified by range(start, stop, step).

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.

Example:

```
# Python program to demonstrate
# string slicing
 # String slicing
String = 'ASTRING'
# Using slice constructor
s1 = slice(3)
s2 = slice(1, 5, 2)
s3 = slice(-1, -12, -2)
 print(& quot  String slicing & quot  )
print(String[s1])
print(String[s2])
print(String[s3])
```

Output:

String slicing

AST

SR

GITA

Python String Functions

The Python String Functions which we are going to discuss in this article are as follows:

- Capitalize( ) function
- Lower( ) function
- Title( ) function
- Casefold( ) function
- Upper( ) function
- Count( ) function
- Find( ) function
- Replace( ) function
- Swapcase( ) function
- Join( ) function

FUNCTIONS

- A function is a collection of statements grouped together that performs an operation.
- A function is a way of packaging a group of statements for later execution.

The function is given a name. The name then becomes a short-hand to describe the process. Once defined, the user can use it by the name, and not by the steps involved. Once again, we have separated the "what" from the"how", i.e. abstraction.

Functions in any programming language can fall into two broad categories:

- Built-in functions

They are predefined and customized, by programming languages and each serves aspecific purpose.

- User-defined functions

They are defined by users as per their programming requirement.

There are two sides to every Python function:

Function definition. The definition of a function contains the code that determines the function's behaviour.

Function call. A function is used within a program via a function invocation.

Defining a Function

A function definition consists of the function's name, parameters, and body.

The syntax for defining a function is as follows:

def function Name(list of parameters):

Statements # Note that the statement(s) must be indentedreturn

- A function contains a header and body. The header begins with the def keyword, followed by the function's name known as the identifier of the function and parameters, and ends with a colon.
- The variables in the function header are known as formal parameters or simply parameters. When a function is invoked, you pass a value to the parameter. This value is referred to as an actual parameter or argument. Parameters are optional; that is, a function may not have any parameters.
- Statement(s) – also known as the function body – are a nonempty sequence of statements executed each time the function is called. This means a function body cannot be empty, just like any indented block.
- Some functions return a value, while other functions perform desired operations without returning a value. If a function returns a value, it is called a value- returning function.

Calling a Function

Calling a function executes the code in the function. In a function's definition, you define what it is to do. To use a function, you have to call or invoke it. The program that calls the function is called a caller. There are two ways to call a function, depending on whether or not it returns a value. If the function returns a value, a call to that function is usually treated as a value.

For example,

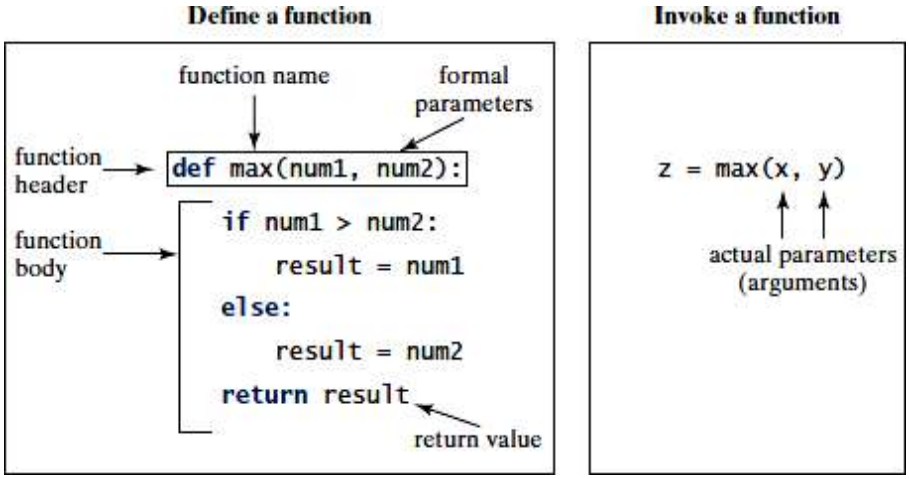
larger = max(3, 4)

Calls max(3, 4) and assigns the result of the function to the variable larger.

Another example of a call that is treated as a value is

print(max(3, 4))

This prints the return value of the function call max (3, 4).



RECURSION

A recursive function is one that invokes itself. Or A recursive function is a function that calls itself in its definition.

For example the mathematical function, factorial, defined by factorial (n) = n*(n-1)*(n-2)*...*3*2*1. can be programmed as

```
def factorial(n):
#n here should be an integer
if n == 0:return 1 else:
return n*factorial(n-1)
```

Any **recursive function can be divided into two parts.**

First, there must be one or more **base cases**, to solve the simplest case, which is referred to as the base case or the stopping condition

Next, **recursive cases**, here function is called with different arguments, which are referred to as a recursive call. These are values that

are handled by "reducing" the problem to a "simpler" problem of the same form.

Example: To find factorial using Recursion

```
def main():
n=int(input("Enter a nonnegative integer: ")) print("Factorial of", n, "is",factorial(n)) print(" 0! = ", factorial(0))
print(" 1! = ", factorial(1))
print(" 5! = ", factorial(6))
# Return the factorial for the specified numberdef factorial(n):
if n == 0: # Base casereturn 1
else:
return n*factorial(n-1) # Recursive callmain()# call the main
```

Output:

```
Enter a nonnegative integer: 5
Factorial of 5 is 120
0! =  1
1! =  1
5! =  720
```

Python Tuples

A collection of ordered and immutable objects is known as a tuple. Tuples and lists are similar as they both are sequences. Though, tuples and lists are different because we cannot modify tuples, although we can modify lists after creating them, and also because we use parentheses to create tuples while we use square brackets to create lists.

Placing different values separated by commas and enclosed in parentheses forms a tuple. For instance,

Example

```
1. tuple_1 = ("Python", "tuples", "immutable", "object")
2. tuple_2 = (23, 42, 12, 53, 64)
```

3. tuple_3 = "Python", "Tuples", "Ordered", "Collection"

We represent an empty tuple by two parentheses enclosing nothing. Empty tuple = ()

We need to add a comma after the element to create a tuple of a single element

Tuple_1 = (50,)

Tuple indices begin at 0, and similar to strings, we can slice them, concatenate them, and perform other operations.

Creating a Tuple

All the objects (elements) must be enclosed in parenthesis (), each separated by a comma, to form a tuple. Although using parenthesis is not required, it is recommended to do so.

Whatever the number of objects, even of various data types, can be included in a tuple (dictionary, string, float, list, etc.).

```
# Python program to show how to create a tuple
# Creating an empty tuple
empty_tuple = ()
print("Empty tuple: ", empty_tuple)
 # Creating tuple having integers
int_tuple = (4, 6, 8, 10, 12, 14)
print("Tuple with integers: ", int_tuple)
 # Creating a tuple having objects of different data types
mixed_tuple = (4, "Python", 9.3)
print("Tuple with different data types: ", mixed_tuple)
 # Creating a nested tuple
nested_tuple = ("Python", {4: 5, 6: 2, 8:2}, (5, 3, 5, 6))
print("A nested tuple: ", nested_tuple)
```

Output:

```
Empty tuple: ()
Tuple with integers: (4, 6, 8, 10, 12, 14)
Tuple with different data types: (4, 'Python', 9.3)
A nested tuple: ('Python', {4: 5, 6: 2, 8: 2}, (5, 3, 5, 6))
```

Parentheses are not mandated to build tuples. Tuple packing is the term for this.

Tuple Operations

Like string, tuple objects are also a sequence. Hence, the operators used with strings are also available for the tuple.

Operator	Example

The + operator returns a tuple containing all the elements of the first and the second tuple object.	>>> t1=(1,2,3)
	>>> t2=(4,5,6)
	>>> t1+t2
	(1, 2, 3, 4, 5, 6)
	>>> t2+(7,)
	(4, 5, 6, 7)

The * operator Concatenates multiple copies of the same tuple.	>>> t1=(1,2,3)
	>>> t1*4
	(1, 2, 3, 1, 2, 3, 1, 2, 3, 1, 2, 3)

The [] operator Returns the item at the given index. A negative index counts the position from the right side.	>>> t1=(1,2,3,4,5,6)
	>>> t1[3]
	4
	>>> t1[-2]
	5

Operator	Example

The **[:]** operator returns the items in the range specified by two index operands separated by the : symbol. If the first operand is omitted, the range starts from zero. If the second operand is omitted, the range goes up to the end of the tuple.	>>> t1=(1,2,3,4,5,6)
	>>> t1[1:3]
	(2, 3)
	>>> t1[3:]
	(4, 5, 6)
	>>> t1[:3]
	(1, 2, 3)

The **in** operator returns true if an item exists in the given tuple.	>>> t1=(1,2,3,4,5,6)
	>>> 5 in t1
	True
	>>> 10 in t1
	False

The **not in** operator returns true if an item does not exist in the given tuple.	>>> t1=(1,2,3,4,5,6)
	>>> 4 not in t1
	False
	>>> 10 not in t1
	True

UNIT-IV

DICTIONARIES:

A dictionary is a collection which is unordered, changeable and indexed. In pythondictionaries are written with curly brackets, and they have keys and values.

* Key-value pairs
* Unordered

We can construct or create dictionary like:

X={1:'A',2:'B',3:'c'}

X=dict([('a',3) ('b',4)]X=dict('A'=1,'B' =2)

Example:

```
>>> dict1 = {"brand":"mrcet","model":"college","year":2004}
>>> dict1
{'brand': 'mrcet', 'model': 'college', 'year': 2004}
```

Operations and methods:

Methods that are available with dictionary are tabulated below. Some of them have alreadybeen used in the above examples.

Method	Description
Clear()	Remove all items form the dictionary.
Copy()	Return a shallow copy of the dictionary.
Fromkeys(seq[, v])	Return a new dictionary with keys from seq and value equal to v (defaults to None).
Get(key[,d])	Return the value of key. If key doesnot exit,return d (defaults to None).

Items()	Return a new view of the dictionary's items(key, value).
Keys()	Return a new view of the dictionary's keys.
Pop(key[,d])	Remove the item with key and return its valueor d if key is not found. If d is not provided and key is not found, raises KeyError.
Popitem()	Remove and return an arbitary item (key, value). Raises KeyError if the dictionary isempty.
Setdefault(key[,d])	If key is in the dictionary, return its value. Ifnot, insert key with a value of d and return d (defaults to None).
Update([other])	Update the dictionary with the key/value pairsfrom other, overwriting existing keys.
Values()	Return a new view of the dictionary's values
Copy()	Return a shallow copy of the dictionary.
Fromkeys(seq[, v])	Return a new dictionary with keys from seq and value equal to v (defaults to None).
Get(key[,d])	Return the value of key. If key doesnot exit,return d (defaults to None).
Items()	Return a new view of the dictionary's items(key, value).
Keys()	Return a new view of the dictionary's keys.

Pop(key[,d])	Remove the item with key and return its valueor d if key is not found. If d is not provided and key is not found, raises KeyError.
Popitem()	Remove and return an arbitary item (key, value). Raises KeyError if the dictionary isempty.
Setdefault(key[,d])	If key is in the dictionary, return its value. Ifnot, insert key with a value of d and return d (defaults to None).
Update([other])	Update the dictionary with the key/value pairsfrom other, overwriting existing keys.
Values()	Return a new view of the dictionary's values

PASSING DICTIONARY TO FUNCTIONS

A dictionary in Python is a collection of data which is unordered and mutable. Unlike, numeric indices used by lists, a dictionary uses the key as an index for a specific value. It can be used to store unrelated data types but data that is related as a real-world entity. The keys themselves are employed for using a specific value.

Passing Dictionary as an argument

In Python, everything is an object, so the dictionary can be passed as an argument to a function like other variables are passed.

Example:

```
# Python program to demonstrate
# passing dictionary as argument
 # A function that takes dictionary
# as an argument
def func(d):

 for key in d:
 print("key:", key, "Value:", d[key])


# Driver's code
D = {'a':1, 'b':2, 'c':3}
func(D)
```

Output:

key: b Value: 2

key: a Value: 1

key: c Value: 3

Passing Dictionary as kwargs

"kwargs" stands for keyword arguments. It is used for passing advanced data objects like dictionaries to a function because in such functions one doesn't have a clue about the number of arguments, hence data passed is be dealt properly by adding "**" to the passing type.

Example 1:

```
# Python program to demonstrate
# passing dictionary as kwargs
def display(**name):
 print (name["fname"]+" "+name["mname"]+" "+name["lname"])
```

```
def main():
 # passing dictionary key-value
 # pair as arguments
 display(fname ="John",
 mname ="F.",
 lname ="Kennedy")
# Driver's code
main()
```

Output:

John F. Kennedy

Errors and Exceptions:

Python Errors and Built-in Exceptions: Python (interpreter) raises exceptions when it encounters **errors.** When writing a program, we, more often than not, willencounter errors. Error caused by not following the proper structure (syntax) of the languageis called syntax error or parsing error

Zero Division Error:

Zero Division Error in Python indicates that the second argument used in a division (or modulo) operation was zero**.**

Overflow Error:

Overflow Error in Python indicates that an arithmetic operation has exceeded the limits ofthe current Python runtime. This is typically due to excessively large float values, as integer values that are too big will opt to raise memory errors instead.

Import Error:

It is raised when you try to import a module which does not exist. This may happen if you made a typing mistake in the module name or the module doesn't exist in its standard path. In the example below, a module named "non_existing_module" is being imported but it doesn't exist, hence an import error exception is raised.

Index Error:

An Index Error exception is raised when you refer a sequence which is out of range. In the example below, the list abc contains only 3 entries, but the 4th index is being accessed,which will result an Index Error exception.

Type Error:

When two unrelated type of objects are combined, Type Error exception is raised. In example below, an int and a string is added, which will result in Type Error exception.

Indentation Error:

Unexpected indent. As mentioned in the "expected an indented block" section, Python not only insists on indentation, it insists on consistent indentation. You are free to choose the number of spaces of indentation to use, but you then need to stick with it.

Syntax errors:

These are the most basic type of error. They arise when the Python parser is unable to understand a line of code. Syntax errors are almost always fatal, i.e. there is almost never a way to successfully execute a piece of code containing syntax errors.

Run-time error:

A run-time error happens when Python understands what you are saying, but runs into trouble when following your instructions.

Key Error:

Python raises a KeyError whenever a dict() object is requested (using the format a = adict[key]) and the key is not in the dictionary.

Value Error:

In Python, a value is the information that is stored within a certain object. To encounter a ValueError in Python means that is a problem with the content of the object you tried to assign the value to.

Python has many built-in exceptions

Which forces your program to output an error when something in it goes wrong. In Python, users can define such exceptions by creating a new class. This exception class has to be derived, either directly or indirectly, from Exception class.

Different types of exceptions:

* Array Index Out Of Bound Exception.

- Class Not Found Exception.
- File Not Found Exception.
- IO Exception.
- Interrupted Exception.
- No Such Field Exception.
- No Such Method Exception

Handling Exceptions:

The cause of an exception is often external to the program itself. For example, an incorrect input, a malfunctioning IO device etc. Because the program abruptly terminates on encountering an exception, it may cause damage to system resources, such as files. Hence, the exceptions should be properly handled so that an abrupt termination of the program is prevented.

Python uses try and except keywords to handle exceptions. Both keywords are followed by indented blocks.

Syntax:

try:

#statements in try blockexcept :

#executed when error in try blockTypically we see, most of the times

- Syntactical errors (wrong spelling, colon (:) missing ….), At developer level and compile level it gives errors.
- Logical errors (2+2=4, instead if we get output as 3 i.e., wrong output …..,),

As a developer we test the application, during that time logical error may obtained.

Run time error (In this case, if the user doesn't know to give input, 5/6 is ok but ifthe user say 6 and 0 i.e.,6/0 (shows error a number cannot be divided by zero))

This is not easy compared to the above two errors because it is not done by thesystem, it is (mistake) done by the user.

The things we need to observe are:

- You should be able to understand the mistakes; the error might be done by user, DB connection or server.

* Whenever there is an error execution should not stop. Ex: Banking Transaction
* The aim is execution should not stop even though an error occur

FILES

A file is some information or data which stays in the computer storage devices. Python givesyou easy ways to manipulate these files. Generally files divide in two categories, text file and binary file. Text files are simple text where as the binary files contain binarydata which is only readable by computer.

Text files: In this type of file, Each line of text is terminated with a special character called EOL (End of Line), which is the new line character (‘\n’) in python by default.

Binary files: In this type of file, there is no terminator for a line and the data is stored after converting it into machine understandable binary language.

Text files:

We can create the text files by using the syntax:

Variable name=open (“file.txt”, file mode)For ex: f= open ("hello.txt","w+")

* We declared the variable f to open a file named hello.txt. **Open** takes 2 arguments, thefile that we want to open and a string that represents the kinds of permission or operation we want to do on the file
* Here we used "w" letter in our argument, which indicates write and the plus sign that means it will create a file if it does not exist in library
* The available option beside "w" are "r" for read and "a" for append and plus signmeans if it is not there then create it

File Modes in Python:

Mode	Description

'r'	This is the default mode. It Opens file for reading.
'w'	This Mode Opens file for writing. If file does not exist, it creates a new file.If file exists it truncates the file.
'x'	Creates a new file. If file already exists, the operation fails.
'a'	Open file in append mode. If file does not exist, it creates a new file.
't'	This is the default mode. It opens in text mode.
'b'	This opens in binary mode.
'+'	This will open a file for reading and writing (updating)

Zipping and unzipping a file

In Python, the module zipfile contains **ZipFile** class that helps us to zip or unzip a file contents. For example, to zip the files, we should first pass the zip file name in write mode with an attribute ZIP_DEFLATED to the ZipFile class object as:

f = ZipFile('test.zip', 'w', ZIP_DEFLATED)

Here, 'f' is the ZipFile class object to which test.zip file name is passed. This is the zip file that is created finally. The next step is to add the filenames that are to be zipped, using write() method as:

f.write('file1.txt')

f.write('file2.txt')

Python program to compress the contents of files

* from zipfile import *
* f = ZipFile('test.zip', 'w', ZIP_DEFLATED)
* f.write('file1.txt')
* f.write('file2.txt')
* f.write('file3.txt')
* print('test.zip file created...')
* f.close()

A Python program to unzip the contents of the files that are available in a zip file

```
#to view contents of zipped files
from zipfile import *
#Open the zip file
z = ZipFile('test.zip', 'r')
#extract all the file names which are in the zip file
z.extractall()
```

UNIT-V

CONSTRUCTOR

Constructors are generally used for instantiating an object. The task of constructors is to initialize(assign values) to the data members of the class when an object of the class is created. In Python the __init__() method is called the constructor and is always called when an object is created.

Syntax of constructor declaration:

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

Types of constructors:

- **Default constructor:** The default constructor is a simple constructor which doesn't accept any arguments. Its definition has only one argument which is a reference to the instance being constructed.

```
class Student:
roll_num = 101
name = "Joseph"
def display(self):
print(self.roll_num,self.name)
st = Student()
st.display()
```

Output:

```
101 Joseph
```

- Parameterized constructor: constructor with parameters is known as parameterized constructor. The parameterized constructor takes its first argument as a reference to the instance being constructed known as self and the rest of the arguments are provided by the programmer.

```
class Student:
 # Constructor - parameterized
 def __init__(self, name):
 print("This is parametrized constructor")
 self.name = name
 def show(self):
 print("Hello",self.name)
student = Student("John")
```

* student.show()

Output:

```
This is parametrized constructor
Hello John
```

INHERITANCE

Types of Inheritance in Python Programming

Types of inheritance: There are five types of inheritance in python programming:

* Single inheritance
* Multiple inheritances
* Multilevel inheritance
* Hierarchical inheritance
* Hybrid inheritance

i. **Single inheritance**

When child class is derived from only one parent class. This is called single inheritance. The example we did above is the best example for single inheritance in python programming.

ii. **Multiple Inheritance**

When child class is derived or inherited from more than one parent class. This is called multiple inheritance. In multiple inheritance, we have two parent classes/base classes and one child class that inherits both parent classes properties.

iii. **Multilevel Inheritance**:

In multilevel inheritance, we have one parent class and child class that is derived or inherited from that parent class. We have a grand-child class that is derived from the child class. See the below-given flow diagram to understand more clearly.

iv. Hierarchical inheritance

When we derive or inherit more than one child class from one(same) parent class. Then this type of inheritance is called hierarchical inheritance.

v. Hybrid Inheritance

Hybrid inheritance satisfies more than one form of inheritance ie. It may be consists of all types of inheritance that we have done above. It is not wrong if we say Hybrid Inheritance is the combinations of simple, multiple, multilevel and hierarchical inheritance. This type of inheritance is very helpful if we want to use concepts of inheritance without any limitations according to our requirements.

POLYMORPHISM

DUCK TYPING PHILOSOPHY OF PYTHON

Duck Typing is a type system used in dynamic languages. For example, Python, Perl, Ruby, PHP, Javascript, etc. where the type or the class of an object is less important than the method it defines. Using Duck Typing, we do not check types at all. Instead, we check for the presence of a given method or attribute.

The name Duck Typing comes from the phrase:

"If it looks like a duck and quacks like a duck, it's a duck"

Example:

```
# Python program to demonstrate
# duck typing
class Specialstring:
 def __len__(self):
 return 21
# Driver's code
if __name__ == "__main__":
 string = Specialstring()
 print(len(string))
```

Output:

In this case, we call method len() gives the return value from __len__ method. Here __len__ method defines the property of the class Special string The object's type itself is not significant in this we do not declare the argument in method prototypes. This means that compilers can not do type-checking. Therefore, what really matters is if the object has particular attributes at run time. Duck typing is hence implemented by dynamic languages.

But now some of the static languages like Haskell also supports it. But, Java/C# doesn't have this ability yet.

Example: Now, lets look demonstrates how an object be used in any other circumstances until it is not supported.

```
# Python program to demonstrate
# duck typing
class Bird:
 def fly(self):
 print("fly with wings")
class Airplane:
 def fly(self):
 print("fly with fuel")
class Fish:
 def swim(self):
 print("fish swim in sea")
# Attributes having same name are
# considered as duck typing
for obj in Bird(), Airplane(), Fish():
 obj.fly()
```

Output:

```
fly with wings
fly with fuel
Traceback (most recent call last):
 File "/home/854855e5570b9ce4a9e984209b6a1c21.py", line 20, in
 obj.fly()
AttributeError: 'Fish' object has no attribute 'fly'
```

In this example, we can see a class supports some method we can modify it or give them new functionality. Duck-typing emphasis what the object can really do, rather than what the object is.

OPERATOR OVERLOADING

Operators work for user-defined types.

For example, the + operator will perform arithmetic addition on two numbers, merge two lists, or concatenate two strings.

This feature in Python that allows the same operator to have different meaning according to the context is called **operator overloading**.

Example: + Operator Overloading

Suppose if we have a class called Complex that represents complex numbers, we could overload the + operator to add two Complex objects together. For example,

```
class Complex:
 def __init__(self, real, imag):
 self.real = real
 self.imag = imag

 # add two objects
 def __add__(self, other):
 return self.real + other.real, self.imag + other.imag

obj1 = Complex(1, 2)
obj2 = Complex(3, 4)
obj3 = obj1 + obj2
print(obj3)

# Output: (4, 6)
```

In the above example, we have used the + operator to add two Complex objects a and b together.

The `__add__()` method overloads the `+` operator to add the real and imaginary parts of the two complex numbers together and returns a new `Complex` object with the resulting values.

The `__str__()` method returns a string representation of the complex number in the form `a + bj`.

Overloading Comparison Operators

Python does not limit operator overloading to arithmetic operators only. We can overload comparison operators as well.

Here's an example of how we can overload the `<` operator to compare two objects the `Person` class based on their `age`:

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

 # overload < operator
 def __lt__(self, other):
 return self.age < other.age

p1 = Person("Alice", 20)
p2 = Person("Bob", 30)

print(p1 < p2) # prints True
print(p2 < p1) # prints False
```

Output

```
True
False
```

Here, `__lt__()` overloads the `<` operator to compare the `age` attribute of two objects.

The `__lt__()` method returns,

* `True` - if the first object's `age` is less than the second object's `age`
* `False` - if the first object's `age` is greater than the second object's `age` We can define similar methods to overload the other comparison operators. For example, `__gt__()` to overload `>` operator, `__eq__()` to overload `==` operator and so on.

Python Special Functions

Class functions that begin with double underscore `__` are called special functions in Python.

The special functions are defined by the Python interpreter and used to implement certain features or behaviors.

They are called **"double underscore"** functions because they have a double underscore prefix and suffix, such as `__init__()` or `__add__()`.

Here are some of the special functions available in Python,

Function	Description
`__init__()`	initialize the attributes of the object
`__str__()`	returns a string representation of the object
`__len__()`	returns the length of the object
`__add__()`	adds two objects
`__call__()`	call objects of the class like a normal function

Advantages of Operator Overloading

* Improves code readability by allowing the use of familiar operators.
* Ensures that objects of a class behave consistently with built-in types and other user-defined types.
* Makes it simpler to write code, especially for complex data types.
* Allows for code reuse by implementing one operator method and using it for other operators.

METHOD OVERRIDING

Method Overriding in Python is an OOPs concept closely related to inheritance. When a child class method overrides (or, provides it's own implementation) the parent class method of the same name, parameters and return type, it is known as method overriding.

In this case, the child class's method is called the **overriding method** and the parent class's method is called the overridden method.

Method overriding is completely different from the concept of method overloading. Method overloading occurs when there are two functions with the same name but different parameters. And, method overloading is not directly supported in Python.

Parent class: The class being **inherited** is called the Parent or Superclass.

Child class: The class that **inherits** the properties and methods of the parent class is called the Child or Subclass.

Key features of Method Overriding in Python

These are some of the key features and advantages of method overriding in Python --

* Method Overriding is derived from the concept of object oriented programming
* Method Overriding allows us to change the implementation of a function in the child class which is defined in the parent class.
* Method Overriding is a part of the inheritance mechanism
* Method Overriding avoids duplication of code

* Method Overriding also enhances the code adding some additional properties.

Prerequisites for method overriding

There are certain prerequisites for method overriding in Python. They're discussed below --

* Method overriding cannot be done within a class. So,we need to derive a child class from a parent class. Hence **Inheritance** is mandatory.
* The method must have the **same name** as in the parent class
* The method must have the **same number of parameters** as in the parent class.