PYTHON 2.7.11

Application : Is a tool which is used to perform particular task. Applications are mainly designed to reduce human effort. Based on the usage and performance applications are into 3 parts.

- ***Windows based Application*** : The application which we work without help of internet are known as window based application.

  Ex: M.S.Office, VLC Player, PDF Reader

- ***Web based Application*** : The application which required internet to work are known as web based applications.

  Ex: Gmail, Face book, Yahoo ..

- ***Android (Mobile) based Application*** : The application which will work on mobile platform are known Mobile Applications.

Note: Every Application contain 3 major sections: Front-end, Back-end, Programming Language.

UI **LOGIC**

SOFTWARE DEVELOPMENT LIFE CYCLE (SDLC)

Front – End : HTML, CSS, JS

Back – End : SQL, MYSQL, ORACLE

Programming Language : Python, Java, .NET

Programming Language: The set of statements which controls the application behavior is known as programming language.

Every programming language should built based on specific syntactical pattern (Syntax).

Syntax: The rules and regulations to follow while working with programming language are known as syntax.

Note: Every Programming language will use a mediator to convert high level programming language to machine language (Low or Binary Language).

These mediators are divided into 2 types.

- Compiler
- Interpreter

Compiler and Interpreter both are used to convert HLL TO LLL. But compiler follows reading all the lines of a file at a time and it will return list of errors if any.

But, Interpreter will read one row at a time and it will stop the execution whenever problem occurred.

INDEX

- ✓ **LAMBDA FUNCTION**
- ✓ **LIST COMPREHENSION**
- ✓ **FILE OPERATION**
- ✓ **ITERATOR & GENERATORS**
- ✓ **MODULES**
- ✓ **DECARTORES**
- ✓ **OOPS**

INTRODUCTION & FEATURES

Python was developed by Guido van Rossum in early 20th Century. Initially python developed for working on files. Later in 2010 multiple packages were added to python that made python as powerful language.

Note: There are some important features that make Python different from others.

- Interpreter Programming Language
- Open source – we can download the code at free cost and modify the base program.
- Python having very easy syntactical pattern and looks like English phenomenal.
- Python code is easy to understand and easy to analyze.

DATA TYPES

Variable : The name contains (holds) some data it may vary in future is known as a variable.

Syntax: variable_name = Some value

Assignment Operator

Do not use capital letters in variable name.

Do not use spaces to concocinate 2 words instead use _ (Under Score)

EX: employee_name = “Hari”

Assignment operator is used to copy the right side value to left side variable.

Key Words : The name which is available in base line code (Source Code) is known as keywords. *Ex: print, pass, break, continue*

Constant : The data which will not change in future is known as constant.

Ex: Your Name, DOB etc..

Data Types : The mechanism which tells to python that what type of data we are using is known as data types. Python supports 2 data types.

- **Integer** : Integers can store only numerical data without decimal value data
  Ex: variable_name = Integer

  Age = 26
- **Float** : Floats can store numerical data along with decimal values.

  Ex: variable_name = Float

  Price = 20.06

  Distance = 2.8

Operators:

1. **Arithmetic Operators** : (a=10 ; b=20)

Operator	Description	Example
+ Addition	Adds values on either side of the operator.	a + b = 30
- Subtraction	Subtracts right hand operand from left hand operand.	a – b = -10
*	Multiplies values on either side of the operator	a * b =

Multiplication		200
/ Division	Divides left hand operand by right hand operand	b / a = 2
% Modulus	Divides left hand operand by right hand operand and returns remainder	b % a = 0
** Exponent	Performs exponential (power) calculation on operators	a**b =10 to the power 20
//	Floor Division - The division of operands where the result is the quotient in which the digits after the decimal point are removed. But if one of the operands is negative, the result is floored, i.e., rounded away from zero (towards negative infinity):	9//2 = 4 and 9.0//2.0 = 4.0, -11//3 = -4, -11.0//3 = -4.0

BODMAS (Brackets, Orders, Division, Multiplication, Addition, Subtraction)

Ex: 10*20 – (15+25) **2

200 - 40^2
200 - 1600 Ans: 1400

The division operator will always write **quotion** as output.

The Modular division operatior will always write **reminder** as output.

2. **Binary Operators :**

0 (False)

1 (True)

Python supports 2 binary operators. They are True & False.

The conditions will always returns binary operations as output.

3. **Comparison Operators :** (a=10 ; b=20)

Operator	Description	Example
==	If the values of two operands are equal, then the condition becomes true.	(a == b) is not true.
!=	If values of two operands are not equal, then condition becomes true.	
<>	If values of two operands are not equal, then condition becomes true.	(a <> b) is true. This is similar to != operator.
>	If the value of left operand is greater than the value of right operand, then condition becomes true.	(a > b) is not true.
<	If the value of left operand is less than the value of right operand, then condition becomes true.	(a < b) is true.
>=	If the value of left operand is greater than or equal to the value of right operand, then condition becomes true.	(a >= b) is not true.
<=	If the value of left operand is less than or equal to the value of right operand, then condition becomes true.	(a <= b) is true.

Comparison operators are used to create a condition. A condition will always returns Boolean operator (True or False) as output.

4. Python Assignment Operators : (a=10 ; b=20)

Operator	Description	Example
=	Assigns values from right side operands to left side operand	c = a + b assigns value of a + b into c
+= Add AND	It adds right operand to the left operand and assign the result to left operand	c += a is equivalent to c = c + a
-= Subtract AND	It subtracts right operand from the left operand and assign the result to left operand	c -= a is equivalent to c = c - a
*= Multiply AND	It multiplies right operand with the left operand and assign the result to left operand	c *= a is equivalent to c = c * a
/= Divide AND	It divides left operand with the right operand and assign the result to left operand	c /= a is equivalent to c = c / ac /= a is equivalent to c = c / a
%= Modulus AND	It takes modulus using two operands and assign the result to left operand	c %= a is equivalent to c = c % a
**= Exponent AND	Performs exponential (power) calculation on operators and assign value to the left operand	c **= a is equivalent to c = c ** a
//= Floor Division	It performs floor division on operators and assign value to the left operand	c //= a is equivalent to c = c // a

4. **Logical Operators:**

and

or

not

Logical operators are used to work with multiple conditions. Python supports 3 logical operators. They are “ and or not “. In python logical operators will follow below priority order “not, and, or”

Memory Management :

Maintaining the memory of python to make efficient management of memory.

Python contains automatic memory management. It will remove idle objects from memory in later time. If we want to remove idle objects immediately than we can use Garbage Collection process... (GC. Collect).

Whenever we declare 2 or more variables with same name than python will make latest variable data as a active and remaining variables will go into idle stage. (In Python each variable will map to one memory location number different variables uses different memory locations).

Data Structures:

Data structures are used to store multiple elements. Python supports 5 types of data structures. They are

- Strings
- List
- Tuple
- Dictonery
- Set

String: Group of characters enclosed between 2 single quotations ‘ ‘ or 2 double quotations “ “ and 2 thrible quotations ‘‘‘ ‘‘‘ or “”” “”” are known as strings in python.

A string may contain any form of data (Numbers, Characters, Special Characters)

Single quotations and Double quotations are used to create one line string but thrible quotations are used to create multiline strings.

Ex: variable_name = ‘ Hellow world ‘

“Hellow World”

‘‘‘Hellow

World’’’

Indexing : The process of accessing the data from ordered data structure is known as indexing. In ordered data structure each element holds one index position. In python we are having 2 types of indexing process.

- ***Positive Indexing***: In positive indexing the first element starts with 0 index position and incremented by 1 while moving to its right.
- ***Negative Indexing*** : In Negative indexing the first element starts with -1 index position and decremented by 1 for each left indent.

S= “PYTHON”

-6	-5	-4	-3	-2	-1
P	**Y**	**T**	**H**	**O**	**N**
0	1	2	3	4	5

print s[4] (or) print s[-3]

Ans = O H

Inbuilt Functions

print : This method is used to display some date in output window. Print can be used as a method or a key word in 2.x versions and it can be used as only method in 3.x versions.

EX: s = ”Hellow”

Print (s)

Print s

Id : This method will write the memory location number in given variable. This method accepts one parameter.

Id (s)

Id s

type : This method will write the data type of given variable.

type (s)

type s

Ans: type ‘str’

Input : This method is used to get the data from the user. This method will accet any type of data (integer, float, string etc...). this method accepts one parameter that is display message to the user in string format. Ex: “Enter your name :”

Syntax: variable_name = input(“Enter your name :”)

Raw_input : This method is also used to get the data from user but it will store the given data in string format.

Syntax: variable_name = raw_input(“Enter your name :”)

Slicing : The process of getting required data from group of data is known as slicing in python. Slicing is devided into 2 types.

- Positive slicing
- Negative negative slicing

We can write string scenario by using square brackets [ ] , slicing accepts 3 parameters and syntax looks like below.

Syntax: string_name[p1:p2:p3]

P1: Here the first parameter represents starting index position of required string. The default value for P1 is 0 in positive slicing. And the default value for P1 in negative slicing is -1.

P2: represents ending index position of required string. The default value for P2 in positive slicing is last element index position +1 (length of data structure)

The default value for P2 in negative direction is the left most element index position -1 (length of data structure).

P3: represents positive or negative slicing and the number of alternate values. The default value for P3 is always +1

String Functions

dir: dir python inbuilt function. This method will written list of properties or functions for given objects.

This method accepts one parameter that is object name or variable name.

Syntax : dir(variable_name) Ex: s = ”Hellow World”

dir(s) Ans: it gives list of available functions for string

String functions are:

1. **Count** s.count(‘l’) Ans: 3

s.count(‘he’) Ans: 1

2.**upper** s.upper() Ans: HELLOW WORLD

3. **lower** s.lower() Ans: hellow world

4. **isupper** s.isupper() Ans: Flase (it writes Boolean value)

5. **islower** s.islower()

6. **capitalize** s.capitalize() Ans: it writes first letter in capital rest are small

7.**startswith** s.startswith('He') Ans: True (it writes Boolean value)

8.**endswith** s.endswith('jk') Ans: False (it writes Boolean value)

9. **isalpha / isdigit / isalnum** Ans: it writes Boolean value True / False

If the given value is in alpha / digit / alpha numeric

Syn : s.isalpha() / s.isnum() /s.alnum()

Alpha : a - z (or) A - Z

Digit :0 - 9

Special characters : !@#$$^%&": }{&^ .... Etc..

12. **index** : This method will written the index position of given substring. This method accepts atleast 1 parameter or atmost 2 parametors.

First parameter represents substring to find the index position.

Second parameter represents the index position from where to start search.

Ex: s="hellow worlds"

```
print s.index('l')       #     2

print s.index('ow')    #     4

print s.index('o',s.index('o')+1)       #8
```

13. **find** : find method will also written the index position of substring. Index method and find method will contain common properties and setup parameters. But index method will return “substring not found” error if the given string is not available. Where as find method will write -1 if substring not found.

```
s="hellow worlds"

print s.find('l')  #     2

print s.find('ox')       #     -1

print s.find('o',s.find('o')+1)     #8
```

14. **replace** : This method will return the replaced string as output. This method accepts 2 parameters.

First parameter is element to replace.

Second parameter is element to place.

Ex:
```
s="hellow worlds"

print s.replace('o',"@") # hell@w w@rlds

print s.replace('world','nagi')    # hellow nagis
```

Note: As replace method written a string as output, we can perform concurrent replace operators.

Ex:
```
print s.replace('e','A').replace('d','D')       # hAllow worlDs
```

15. **format** : This method will pass the values to dynamic strings and it will return the replaced string as output.

The string which contains variable data in it are known as dynamic strings. Python follows 2 mechanisms to pass the data to dynamic strings using format method.

Using variable names in dynamic strings:

In this mechanism we will place the variable in string. So that we can pass the data to the variables.

Use flower brackets { } to place variable name

In this mechanism the order of passing parameter is optional.

```
Ex:    s=""" Hi {name}
       your bill amount is {amt}
       and bill No. {bill_no}"""


print s.format(name="Hari", amt='120',bill_no='004')
```

Using index positions in Dynamic string:

As is the index position always starts from 0 we should place 0 index position for first variable and increment the position by +1for each variable.

The order of passing parameter is mandatory.

```
Ex:          s=""" Hi {0}
             your bill amount is {1}
             and bill No. {2}"""
             print s.format('Hari','120','004')
```

16. **Split :** This method is used to split the given strings based on substring. The method accepts atleast 0 parameters and atmost 1 parameter.

The default value for split method is always “space” ‘ ’

The output format is always in List.

Ex: s='python programming language is easy'

print s.split() # ['python', 'programming', 'language', 'is', 'easy']

t='''Hi

Hellow

Hari'''

print t.split() # ['Hi', 'Hellow', 'Hari']

17. **encode / decode** : These methods are mainly used to provide the security to the data. Encode method is used to **enscript** the tata and decode method is used to **descript** the data.

These methods accepts only one parameter that is security pattern.

Security patterns : base64, base32, base16, base8

Utf64, utf32, utf16, utf8

18. **Join** : Join method is used to join group of strings together. This method accepts one parameter that is string name. and it requires another element that is element to join with.

Ex: s='python programming language is easy'

s_sp= s.split() # ['python', 'programming', 'language', 'is', 'easy']

print s_sp

print '__'.join(s_sp) #
python__programming__language__is__easy

LIST

List: The group of elements are enclosed between 2 square brackets [ ] is known are list. A list may contain any type of data in it. We can access the data from a list using indexing process. So we can name it as ordered data structures.

We can perform CRUD [Create, Read, Update Delete] operations. So list are mutable data structures in python.

Syn: list_name = [element 1, element 2, element 3, …]

s=[1,2,3,4,”Hellow”]

s[0] = 1

s[4] = Hellow

s[4][3] = l

s[4][3::-1]

List Functions – dir(s)

1. count: This method accepts parameter that is element to count. This method writes no of occurrences of the given element.

Ex: s=[1,'hi',2.0, [4,5,6,],1,2,0]

print s.count(2.0) Ans: 2

print s.count(4) Ans: 0

print s[3].count(4) Ans: 1

2. **Index**: print s.index(2.0)

print s.index([4,5,6])

3. append: This method is used to insert 1 element at the end of the list. This method accepts only one parameter to add.

Here the element can be any type.

Ex: s.append(23) Ans: [1, 'hi', 2.0, [4, 5, 6], 1, 2, 0, 23]

4. extend: This method is used to insert group of elements at end of the list. This method accepts only one parameter that is **data structure** to insert.

Note: we can pass only data structure as input to extend method.

Ex: s.extend("hai") Ans: [1, 'hi', 2.0, [4, 5, 6], 1, 2, 0, 23, 'h', 'a', 'i']

5. Insert: This method is used to insert one element at the given index position. This method accepts exactly 2 parameters, they are index position & element to insert.

If the index position is not available in given list then the given element will append to the last.

Ex: x=[0,1,2,3,[10,20,30],"%$","hI","Hi"]

x.insert(3,"Hellow") Ans: [0, 1, 2, 'Hellow', 3, [10, 20, 30], '%$', 'hI', 'Hi']

6. pop: This method will return the element at given index position and removes it. This method accept at most 1 parameter that is index position. The default value for pop method is always -1.

Ex: x=[0,1,2,3,[10,20,30],"%$","hI","Hi"]

x.pop(4) Ans: [0, 1, 2, 3, '%$', 'hI', 'Hi']

x=[0,1,2,3,[10,20,30],"%$","hI","Hi"]

x[4].pop(1) Ans: [0, 1, 2, 3, [10, 30], '%$', 'hI', 'Hi']

7.remove: This method is used to remove the given element. This method accepts exactly 1 parameter that is element to delete.

If the element is not available in the list then it will return an error.

Ex: x=[0,1,2,3,[10,20,30],"%$","hI","Hi"]

x.remove([10,20,30])

x=[0,1,2,3,[10,20,30],"%$","hI","Hi"]

x[4].remove(10)

8. sort : This method is used to arrange the elements in a proper order by using sort method. We can do ascending order as well as desending order.

x.sort() - Ascending

x.sort(reverse=True) - Decending

9. reverse: This method is used to reverse the given list. This method does not require any parameters.

x.reverse()

Note: in python mutable data structures we can update data by using assignment operator.

Ex: x=[0,1,2,3,[10,20,30],"%$","hI","Hi"]

x[3]=30 Ans: [0, 1, 2, 30, [10, 20, 30], '%$', 'hI', 'Hi']

x=[0,1,2,3,[10,20,30],"%$","hI","Hi"]

x[4][0]=100 Ans: [0, 1, 2, 3, [100, 20, 30], '%$', 'hI', 'Hi']

Membership Operators:

Membership Operators are used to test whether the given element is available in group of elements or not. Member operations will always return Boolean operator as output.

Python supports 2 types of membership operators. They are **in, not in**

Ex: a=10

L=[1,2,3,4,5,10,”hi”]

Print a in l Ans: True

Print a not in l Ans: False

Print ‘H’ in ‘python’ Ans: False

Print l[-1] in ‘Hi Hellow’ Ans: True

Tuples

The group of elements are enclosed between 2 parenthesis () is known as known as Tuple in python.

The group of elements will separated by commas , . here the element can be of any type. Tuples are ordered and immutable data structures in nature. That means we can access the data from a tuple by using indexing process and the elements of tuple do not change.

Note: In some special situations like placing mutable data structure in tuple we can alter the elements of mutable data structure.

Syn: Tuple_name(element 1, element 2, element 3, [element 1, element 2], 3.0.4.5,4)

Tuples support only 2 functions. They are count and index. These methods will work just like index and count methods of list.

X=(1,2,[3.0,4.0])

X[1]=5 : it will write error

X[2][1]=10 : it will change 3.0 to 10 inside the list

Del x[2].pop(1) : it will delete 4.0 from the list inside the tuple.

Block

Indentation: The rules and regulations to write block of statements is known as indentation. We should follow the below rules while writing block of statements.

- Every block should starts with ‘ : ‘ every statement of block should contain particular no. of spaces.
- The block will close automatically when no statements contains spaces.
- Syn : Block Name:
  ---- Statement 1
  ----Statement 2
  ----Statement 3

Writing from here is out side the block (means block is closed)

Conditional Blocks : Python supports 3 types of conditional blocks they are if, else, ifelse (elif) ladder.

If: if block can handle the successful execution of condition. (only for True output)

Syn: if (condition):

Statement 1
Statement 2
Statement 3

} What to do if **condition** is **True**

Ex: a=10, b=20

if (a>b)

Print “ a is smaller than b”

Note: If can be used as method or key word.

else: These conditional blocks used for handling both successful and failure condition to write false block statements we will use **else** key word.

Only one block will execute based on the condition statements.

Syn: if (condition):

----Statement 1
__Statement 2 } True Block

Else:

__Staement 1
__statement 2 } False Block

elif: elif ladder contains multiple conditional blocks. We can write negative conditional blocks by using **elif** keyword.

elif block should contain **condition.**

Every elif ladder should starts with if block & ends with else block.

Ex: X=”Nagireddy_8@yahoo.com”

Syn: if (condition): if x.isalpha():

----Statement 1 } Print “string contains only alpha”
__Statement 2 } elif x.isdigit():

Elif (Condition): Print “string contains only numbers”

__Staement 1 } elif x.isalnum():
__statement 2 } Print “string contains both alpha & num”

Elif (Condition):

__Staement 1 } else:
__statement 2 } print “ string contains special charcters”

Else:

__Staement 1

__statement 2

Loop: Loop are iteration. Is a process of working some operation for multiple times.

Compare to other looping techniques python for loops are very efficient in nature because it does not require variable assignment, conditional checks and increment/decrement operation.

We should mention **in** operation in “for loop”. Because this operation will take care of item assignment (variable assignment) for each iteration.

Python interpreter will exit from the loop when there are no element left in group of element.

Note: Python follows indentation process according to this every block should contain one statement. To overcome this problem python developers introduced “**pass**” key word.

The **pass** key word is used to create unimplemented blocks.

Syn: for **variable** in **Group Element:**

Ex:1 x=[0,1,3,4,5,6,7,8,9,10]

For I in x:

Print i

Ans: it will print all the numbers one by one.

Ex:2 emp_name=[“Hari”, “Kiran”, “Raju”, “Mahi”]

For name in emp_name:

If “I” in name

Print name

Ans: Hari, Kiran, Mahi

TASKS

 # TASK 1 --- FIND THE MAX VALUE

```
a=50;b=80;c=30
if a>b and a>c:
        print a
elif b>a and b>c:
        print b
elif c>a and c>b:
        print c
```

 # TASK 2 --- FIND ANY KEYWORD

```
x="PYTHON"
if "PY" in x or "ON" in x:
        print "Yes 'PY' is available in 'PYTHON'"
```

 # TASK 3 --- FIND EVEN/ODD NUMBERS

```
numbers = [46,9,55,32,67,24,10,80,90]
for i in numbers:
        if i%2==0:
                print "even number is: " ,i
for x in numbers:
        if x%2>0:
                print "Odd number is: " ,x
```

 # TASK 4 --- FIND SUM/AVG/FAILED MARKS

```
marks = [46,9,55,32,67,24,10,80,90.0]
total_marks=0
for i in marks:
        total_marks=total_marks+i
        print total_marks
        if i<35:
                print i
print "Total Marks : ", total_marks
print "Average Marks : ", total_marks/len(marks)
print "Failed Marks are :", i
```

 # TASK 5 --- FIND FILTERS

```
emp=[("hari","Bangalore","Ram"),("Raju",          "Bangalore","Wipro"),
("Ramu","Bangalore","Ram"),("Mahi","Chennai","Wipro")]
for city in emp:
        if city[1]=="Bangalore":
                print city[0]+" works in "+city[1]
print "employess woking in Ram"
for company in emp:
        if company[2]=="Ram":
                print company[0]+ " placed in "+company[2]
for company in emp:
        if "Bangalore" in company[1] and "Ram" in company[2]:
```

```
        print company[0]
```

 # TASK 6 --- COUNT REPEATED WORDS

```
slogan='''python is simple
python is beautiful'''
marks=slogan.split()
total_marks=''
for i in marks:
    total_marks=i
    print i, " is Repeated" ,marks.count(total_marks),"times"
```

 # FIND Emp_1 TOTAL PERCENTAGE AND AVG MARKS IN SCIENCE FOR ALL CANDIDATES AND LIST MARKS WHO GOT LESS THAN 70

```
s='''Emp_1 Maths 71
Emp_2 Maths 81
Emp_3 Maths 91
Emp_1 Science 42
Emp_2 Science 52
Emp_3 Science 62
Emp_1 English 73
Emp_2 English 43
Emp_3 English 93'''
x= s.splitlines()
```

```
y=[]
for i in x:
        y.append(i.split())
#print y
Emp_1_marks=0
Emp_1_Total=0
science_total=0
science_marks=0

for i in y:

        if "Emp_1" in i[0]:
                Emp_1_Total=Emp_1_Total+100
                Emp_1_marks=Emp_1_marks+int(i[2])
        if "Science" in i[1]:
                science_total=science_total+1
                science_marks=science_marks+int(i[2])
        if int(i[2])<70:
                print i[0],i[1], i[2]
Emp1_percentage= (float(Emp_1_marks)/Emp_1_Total)*100
print y[0][0], Emp1_percentage
print "Average marks in Science : ", science_marks/science_total
```

Type Conversions

The process of converting from one data type to another data type is known as type conversion. We cannot convert the data type (int, float) to data structures like (list, tuple) and vice versa.

In type conversions python supports 7 types of conversion methods

1. int(): This method is used to convert given data to integer. This method accept only one parameter that is the data to convert into integer.

Supported convertions:

1. Int(data) > float, string
2. Float > int, string
3. String (data) > int, string
4. List(data) > Tuple, string
5. Tuple > List, string

Range : This method accept list of sequential numbers. This method accept atleast one parameter or atmost 3 parameters as input.

Syn : Range(p1,p2p3)

Ex: range(0,50,1)

P1 : Starting element value (lower value)

Default value is: 0

Not mandatory

P2 : Ending element value (Higher value)

Mandatory

P3 : number of alternate value

Default value is: 1

Not mandatory

Note: range method always write output in list. It will cause higher memory consumption. To avoid this memory consumption python developers introduces a method named as **xrange** which will write an object as output.. the memory consumption is low.

Syn: xrange(P1,P2,P3)

Ex: xrange(10) Ans: xrange(10) --- but inside it is 0---9

DICTONORIES

The group of items are enclosed between 2 flower brackets { } are known as Dictionaries in python. Here items are separated with commas,

Syn: Dict_name = { Item1, Item2, Item3,...}

Dict_name = {key1:value1, key2:value2, key3:value3}

Here **item** is combination of **key and value** pair. Keys and values are separated by colon **:**

Keys are immutable data elements, which can accept only strings, tuple, int, float.

Values can be of any type in dictionary.

As the combination of immutable and mutable data elements dictionaries were treated as mutable data structures and we can perform CRUD operations on dictionaries.

Dictionaries are unordered data structures in python and which uses **Heap** as memory type. Heap will store the data by using hash key mapping mechanism. In Heap, key located in outer level and values mapped to the keys as inner level.

As dictionaries uses hash key mechanism the data retrieval process can be done by using keying process. The performance of dictionaries is very high as it does not contain ordered data.

Dictionary - Functions

Keys() : This method will retrieve list of keys which are available in dictionary. This method does not require any parameter. The output format will always returns list as output.

Syn: dict_name = {k1:v1,k2:v2,k3:v3}

Ex: x={'a':1,'b':2,'c':3}

print x.keys() Ans: ['a', 'c', 'b']

Values() : This method will retrieve list of values which are available in dictionary. This method does not require any parameter. The output format will always returns list as output.

Syn: dict_name = {k1:v1,k2:v2,k3:v3}

Ex: x={'a':1,'b':2,'c':3}

print x.values() Ans: [1, 3, 2]

items() : This method will retrieve list of items which are available in dictionary. This method does not require any parameter. The output format will always returns tuple as output.

Syn: dict_name = {k1:v1,k2:v2,k3:v3}

Ex: x={'a':1,'b':2,'c':3}

print x.items() Ans: [('a', 1), ('c', 3), ('b', 2)]

viewkeys() : This method will return list of keys which are enclosed in "**dict_keys**" method. This method does not required any parameters. Output format is always a method.

Ex: x={'a':1,'b':2,'c':3}

print x.viewkeys() Ans: dict_keys(['a', 'c', 'b'])

viewvalues() : This method will return list of keys which are enclosed in "**dict_values**" method. This method does not required any parameters. Output format is always a method.

Ex: x={'a':1,'b':2,'c':3}

print x.viewvalues() Ans: dict_values([1, 3, 2])

viewitems() : This method will return list of keys which are enclosed in "**dict_items**" method. This method does not required any parameters. Output format is always a method.

Ex: x={'a':1,'b':2,'c':3}

print x.viewitems() Ans: dict_items([('a', 1), ('c', 3), ('b', 2)])

iterkeys() : This method will return dictionary key **iterator object** which contains keys of given dictionary. This method does not require any parameter. The output format is always an iterator object.

Ex: x={'a':1,'b':2,'c':3}

print x.iterkeys() Ans: <dictionary-keyiterator object at 0x01DEF8A0>

itervalues() : This method will return dictionary values **iterator object** which contains values of given dictionary. This method does not require any parameter. The output format is always an iterator object.

Ex: x={'a':1,'b':2,'c':3}

print x.itervalues() Ans: <dictionary-valueiterator object at 0x01DEF8A0>

iteritmes() : This method will return dictionary items **iterator object** which contains items of given dictionary. This method does not require any parameter. The output format is always an iterator object.

Ex: x={'a':1,'b':2,'c':3}

print x.iteritems() Ans: <dictionary-itemiterator object at 0x01DEF8A0>

Dictionary - Methods

Get(p1,p2) : This method will return the value at given key. This method accepts at least 1 parameter and at most 2 parameters. If the key is not available in the dictionary get method will return the default value (None) as output.

Syn: get (p1,p2)

Ex: x={'a':1,'b':2,'c':3}

print x.get("a") Ans: 1

print x.get("a","key not available") Ans: 1

print x.get("x","key not available") Ans: key not available

Setdefault(p1,p2) : This method will return the value at given key, it will add an item to dictionary it the key is not available. If the key is available, then it will return the value and will not affect the dictionary.

This method accepts at least 1 parameter and at most 2 parameters.

Syn: d.setdefault(P1,P2) P1= key, P2 = Value

Ex: d={'a':1,'b':2}

print d.setdefault('c',10) Ans: 10

print d.setdefault('b',25) Ans: 2

print d.setdefault('x') Ans: None

print d Ans: {'a': 1, 'x': None, 'c': 10, 'b': 2}

pop(p1,p2) : This method is used to delete particular item from dictionary, it will return the value of given key before deleting it. This method accepts at least 1 parameter and at most 2 parameters.

Syn: Dict_name.pop(key, default value)

Note: if the key is available in dictionary (only key given as per) if the key is available in dictionary it will delete the item, if the key is not available then pop method will return ‘key error’. To avoid this we need to pass some value as second parameter.

Ex:
```
d={'a':1,'b':2, 'c':3}
print d.pop('a')
print d.pop('x',98)
print d
```

popitem(): This method is used to delete the first item of the dictionary, it will return the first item and then delete it. This method accepts no parameters.

Syn: Dict_name.popitem()

Ex:
```
d={'a':1,'b':2, 'c':3}
print d.popitem()          Ans: ('a', 1)
print d                    Ans: {'c': 3, 'b': 2}
```

has_key(p1): This method is used to check the availability of given key. This method will always returns a Boolean operator as output.

This metod required only 1 parameter that is **key** to check.

Syn: dict_name.has_key(key)

Ex:
```
d={'a':1,'b':2, 'c':3}
print d.has_key('a')       Ans: True
print d
```

clear(): This method is used to remove complete data from dictionary. No parameters required.

Syn: dict_name.clear()

Ex: d={'a':1,'b':2, 'c':3}

print d.clear()

print d Ans: { }

fromkeys(P1,P2): This method will return a dictionary which contain the keys taken from given data structure and value from given default value. This method will not affect the base dictionary. This method accepts at least one parameter at most two parameters. First parameter represents data structures and second parameter represents default value (none as defalt)

Syn: dict_name.fromkeys(datastrucure, default value)

Ex: d={'a':1,'b':2, 'c':3}

print d.fromkeys('abc',10) Ans: {'a': 10, 'c': 10, 'b': 10}

Shallow Copy : The process of copying the object (Mapping Memory Location) from one data to another data is known as shallow copying.

Shallow copy will behave differently in both mutable and immutable data types.

Shallow copy on immutable data types

a=10

b=a

print b Ans: 10

a=20

print b Ans: 10

Shallow copy on mutable data types

Ex: x1=[1,2,3]

Deep Copy : The process of copying only the data from one location to other is known as deep copy.

Ex:
```
x1=[1,2,3]
x2=x1[::]
x1[1]=15
print x1
print x2
```

Note:

Deep copy will only support in mutable data structures.

On immutable objects only shallow copy will work.

Shallow copy:

Mutable Data Types : Share single objects

We can change values

Immutable Data Types : Share single objects

We cannot change values

Deep Copy

Mutable Data Types : Share different objects

We cannot change data in both at a time.

Copy() **:** This method is used to perform deep copy on dictionaries (This method will written the data of given dictionary)

Ex: d1={'a':1,'b':2,'c':3}

d2=d1.copy()

print d2 Ans: {'a': 1, 'c': 3, 'b': 2}

FUNCTIONS

The block of statements which are used to perform particular operation is known as a **function** in python. The main purpose of functions is reusability.

Python functions are devided into 2 types they are **called functions** and **calling functions**

called functions: a functions which contains definition (“def”), set of parameters / arguments, signature and return statement is known as called function.

Syn: def function_name(P1,P2):

Statement 1

Statement 2

Return

Ex: def add(a,b): >>> Called Function

c=a+b

return c

print add(10,20)>>> Calling Function

Without return statement we cannot traverse the data from called function to calling function.

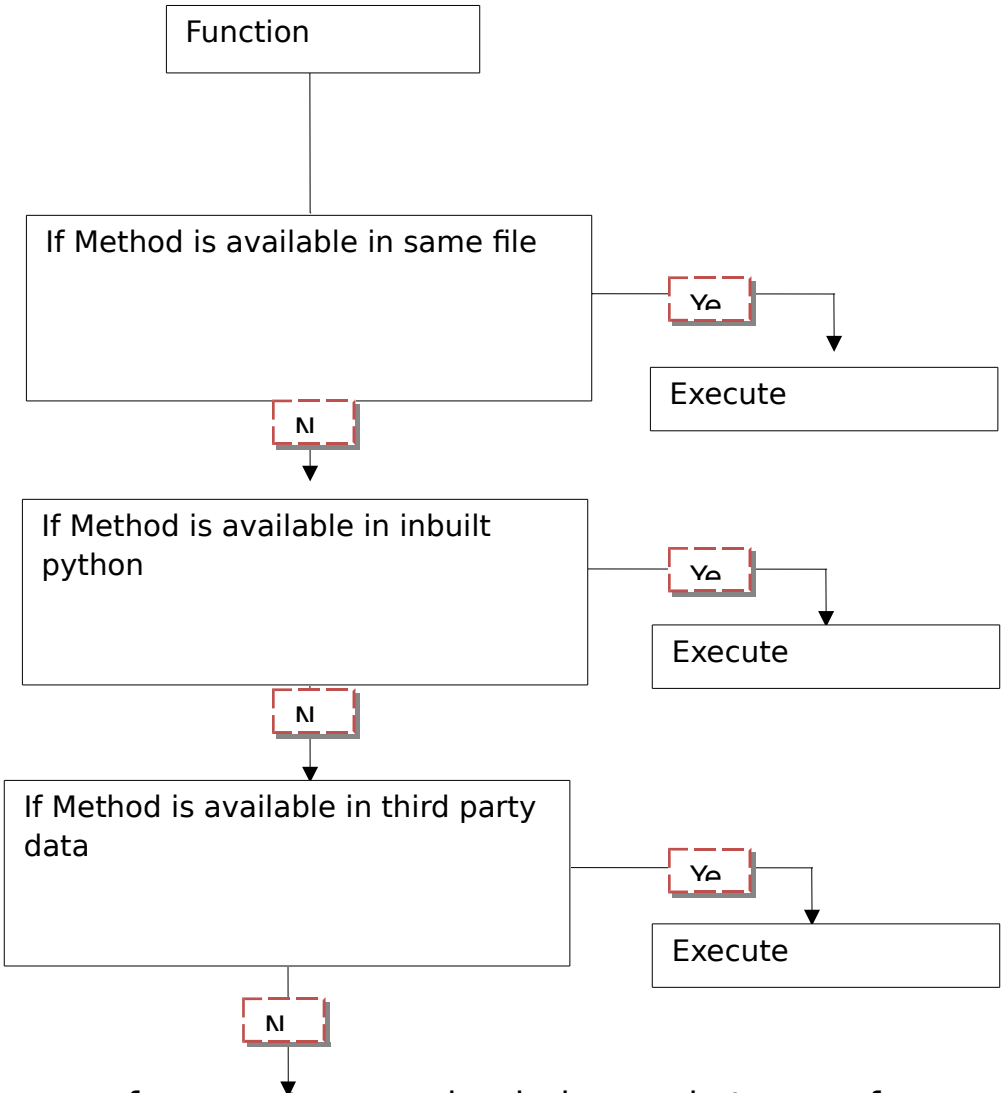
Without calling function no called function will execute.

Calling function: The function which contain only function name along with signature is known as calling function.

Syn : function_name(P1)

Int (“123”)

Function resolution order: The order of searching the given function in entire mission is known as function resolution order.



Note: the no. of parameters passing in is equals to no. of parameters receiving. We ca no. of parameters or lesser no. of parameters to a function. To overcome this limitations python is uses default parameters methodology.

According to this we should define the default parameters only at function definition level (called function) and default parameters should declare after mandatory parameters.

Note: The life time of the function variable is upto the function. We cannot use outside parameters in the function.

***args:** *args itself is a variable used to receive/accept multiple parameters or unknown no.of parameters. Here * indicates a tuple declaration which can accept multiple parameters definition.

Args indicates the variable name of tuple. According to python we should define default parameters fallowed by mandatory parameters.

Syn: def fun_name(*t)

Ex: def function1(*t):

print type(t)

return sum(t)

print function1(1,2,3)

Do not define multiple *args parameters in the single function.

****kwargs or **kvargs:** This default parameter is used to accept/receive multiple items in a function definition.

** indicates a dictionary definition and kwargs indicates the name of the dictionary.

We should send the data (items to the called function like below)

Function_name(var1=value1, var2=value2)

Ex: def fun_1(a,*t,**d):

```
        return a,t,d
print fun_1(1,2,3,4,5,"hi",x="hari",y="mani")
```

Ans: (1, (2, 3, 4, 5, 'hi'), {'y': 'mani', 'x': 'hari'})

Note: follow the below order while defining function signature.

1. Mandatory parameters
2. Default parameters (with variable assignment (a=10,b=20))
3. *args
4. **kwargs / **kvargs

Note: in dictionaries we can update an item or insert an item by using keying process along with assignment operator.

Syn: dict_name[key]=value

The above process will update the value at given key if key is already available, if not available it will insert as a item.

Interview - Task:

```
def fun1(a,l=[]):
        for ele in range(a):
            l.append(ele*ele)
        return l
print fun1(5)
print fun1(2)
```

Tasks

{“hari={“Telugu”:68, “Eng”:75, “science”:43},

1. Write a program to display percentage of marks for each student

2. write a program to update a dictionary as below.

* Replace marks with ‘Distinction’ if marks>70

*Replace marks with ‘Ist class’ if marks >60

*Replace marks with ‘2nd class’ if marks >50

*Replace marks with ‘Fail’ if marks <35

```
Def get_percentage(d)

Def get_finalresult(d)

def percentage(stu_marks):
    for key,values in stu_marks.items():
        percentage=sum(stu_marks[key].values())/
((len(stu_marks[key])*100)/100)
        print key, percentage


        for i in xrange(len(stu_marks[key])):
            if stu_marks[key].values()[i]>=75:
                stu_marks[key][stu_marks[key].keys()
[i]]="Distinction"
            elif stu_marks[key].values()[i]>=60:
                stu_marks[key][stu_marks[key].keys()[i]]="First
class"
            elif stu_marks[key].values()[i]>=50:
                stu_marks[key][stu_marks[key].keys()
[i]]="Second class"
            elif stu_marks[key].values()[i]<35:
```

```
                        stu_marks[key][stu_marks[key].keys()
[i]]="Failed"
            return stu_marks

stu_marks={'Hari':{'Telugu':50,'English':61,'Hindi':75},
'Raju':{'Telugu':40,'English':20,'Hindi':59},
'Mahi':{'Telugu':49,'English':70,'Hindi':80},
'Nani':{'Telugu':80,'English':90,'Hindi':75}}

print percentage(stu_marks)
```

2. emp_data = “” hari Bangalore IT 10000
Raju Hyderabad Infra 10000
Balu Pune Civil 10000
Kavita Bangalore IT 10000
Srujana Hyderabad Infra 10000
Srikanth Pune Civil 10000

Output: {‘hari’={‘loc’=’Bangalore’,’domain’:’IT’,’sal’:10000}

Ans)..

```
def convert_dict(emp_data,*key):
        emp_Data= emp_data.split('\n')
        item={}
        for i in xrange(len(emp_Data)):
```

```
            v= emp_Data[i].split()
            value= v[1:]
            item.setdefault(v[0],dict(zip(key, value)))
        print item


emp_data='''Hari Bangalore IT 10000 Old
Raju Hyderabad Infra 20000 Young
Balu Pune Civil 30000 Boy
Kavita Bangalore IT 40000
Srujana Hyderabad Infra
Srikanth_1 Pune Civil 60000'''

key=["Loc","Domain","Salary","Age"]
convert_dict(emp_data,*key)
```

FUNCTIONAL PROGRAMMING TOOLS

Functional programming tools are used to perform particular task on each and every element of given data structure with less complexity.

These tools will contains virtual looping mechanism on memory system.

Python supports 4 types of functional programming tools they are (**filter, map, reduce, zip**)

Filter: ‘filter(*function*, *sequence*)’ returns a sequence consisting of those items from the sequence for which *function*(*item*) is true. If *sequence* is a `string` or `tuple`, the result will be of the same type; otherwise, it is always a `list`.

Syn: Filter(function_name, group of elements as parameters)

The calling function is different for both normal and functional programming tools.

Ex 1:
```
def get_even(a):
    if a%2==0:
        return a
```

print filter(get_even,range(10)) Ans: [2, 4, 6, 8]

Ex 2:
```
emp_names='hari kiran ravi raju manoj'

def emp(name):
    return name.endswith('i')
```

print filter(emp,emp_names.split()) Ans: ['hari', 'ravi']

map: ‘map(*function*, *sequence*)’ calls *function*(*item*) for each of the items in the sequence and returns a list of the return values.

More than one sequence may be passed; the function must then have as many arguments as there are sequences and is called with the corresponding item from each sequence (or `None` if some sequence is shorter than another).

This method accepts a list as input and returns a list as output.

emp_names='hari kiran ravi raju manoj'

Ex: def emp(name):

return name.replace('i','@')

print map(emp,emp_names.split()) Ans: ['har@', 'k@ran', 'rav@', 'raju', 'manoj']

Ex: def sal_increment(sal):

return sal+(sal/100*30)

print map(sal_increment,[15000,30000,60000]) Ans: [19500, 39000, 78000]

Ex: def add(x, y): return x+y

seq = range(8)

print map(add, seq, seq)

reduce: ‘`reduce(`*`function`*`,` *`sequence`*`)`’ returns a single value constructed by calling the binary function *function* on the first two items of the sequence, then on the result and the next item, and so on. For example, to compute the sum of the numbers 1 through 10:

Ex: def add(x,y): return x+y

print reduce(add, range(1, 11)) Ans: 55

If there's only one item in the sequence, its value is returned; if the sequence is empty, an exception is raised.

A third argument can be passed to indicate the starting value. In this case the starting value is returned for an empty sequence, and the function is first applied to the starting value and the first sequence item, then to the result and the next item, and so on. For example,

Ex:
```
def sum(seq):
    def add(x,y): return x+y
    return reduce(add, seq, 0)
print sum(range(1, 11))      Ans: 55
print sum([])      Ans: 0
```

Zip: This method will return list of tuples (list of items) from given 2 lists.

To convert list of tuples into dict then use **dict** method.

Syn: zip(list1, list2)

Ex:
```
x=['hari','raju','kiran']
y=[1,2,3]
print dict(zip(x,y))  Ans: {'raju': 2, 'kiran': 3, 'hari': 1}
```

Tasks:

LAMBDA FUNCTIONS

The function which contains one line statements and does't contain return key word are know as lambda functions. Lambda functions can be created by using lambda key word.

This keyword contains 2 sections:

1. parameter section

2. operation section

These sections were separated by ' **:** '

Lambda functions were reusable upto a file and it does not support modularity mechanism

Syn: function_name lambda parameters : operation

Ex 1 : add=lambda a,b:a+b

print add(10,20) Ans: 30

Ex 2 : add=lambda a,b:dict(zip(a,b))

a=range(10)

b=range(11,20)

print add(a,b) Ans:{0: 11, 1: 12, 2: 13, 3: 14, 4: 15, 5: 16, 6: 17, 7: 18, 8: 19}

Ex 3: print reduce(lambda a,b:a+b,range(10)) Ans: 45

Ex 4: print filter(lambda a:a%2==0, range(10)) Ans: [0, 2, 4, 6, 8]

Note: We can use lambda functions along with functional programming tools to reduce number of lines of code.

Find average of 100 odd numbers

Ans: print sum(range(1,101,2))/len(range(1,101,2))

LIST COMPREHENSIONS:

List comprehensions are concise way to create complex data list in the single line of code. A list comprehension will always enclosed between 2 square brackets [ ].

It should contain atleast 2 sections and atmost 3 sections.

1. Looping Section

2. Condition Section (Optional)

3. operational section

Syn: `[EXPRESSION FOR_LOOPS CONDITIONALS]`

Ex: print [ele for ele in range(1,50) if ele%2==0]

Ans: [2, 4, 6, 8, 10, 12, 14, 16, 18, 20, 22, 24, 26, 28, 30, 32, 34, 36, 38, 40, 42, 44, 46, 48]

Ex: print [ele for ele in range(1,50) if ele%3==0 and ele%5==0]

Ans: [15, 30, 45]

Multiple conditions

If else in List Comprehensions:

Syn: [output if true if condition else o/p of false Looping]

Ex: print [ele if ele%2==0 else None for ele in range(10)]

Ans: [0, None, 2, None, 4, None, 6, None, 8, None]

Ex: print[(ele, 'sc') if not ele.isalpha() else (ele,'vowel') if ele in 'aeiou' else (ele,'con') for ele in s]

Ans: [('h', 'con'), ('e', 'vowel'), ('l', 'con'), ('l', 'con'), ('o', 'vowel'), ('w', 'con'), (' ', 'sc'), ('w', 'con'), ('o', 'vowel'), ('r', 'con'), ('l', 'con'), ('d', 'con')]

FILE OPERATIONS:

Python is able to work with different types of files like [**text files, excel files, csv files, word documents, pdf docs etc..**].

Especially to work with text files we don't require any special modules rather we can use inbuilt methods like **open**.

Open Methods: This method is used to open any text files in a particular operation mode.

Note: python supports mainly 3 file operating modes they are:

- Read operation mode – 'r'
- Write operation mode – 'w'
- Append operation mode – 'a'

Open method accepts at least 1 parameter and at most 2 parameters. P1 – is file name and P2 is operating mode. Default operation mode is always read 'r'.

Read Operation Mode: *To work in* read operation mode use **open** method along with **file name** and '**r**' as operation mode. 'r' is optional.

Open method will return a file object as output which holds the data in givenfile.

Syn: open("file name.txt",'r')

Ex: open("file_op.txt")

Read: read method will return the complete file data in the form of string. This method accepts at least 0 and at most 1 parameter. If we are not passing any parameter it will return all the characters of the file.

If we are passing an integer as parameter then it will return the given no. of character.

Syn: file_pointer_variable.read(integer)

Ex: print fp.read()

Print fp.read(6)

Readline: This method will return the first line data of given file. This method accepts at most 1 parameter that is integer. If we are passing an integer then it will return the given no. of characters in 1st line of the file. If we are not passing any parameter it will return the complete line.

Syn: file_pointer_variable.readline(integer)

Ex: print fp.readlines()

Print fp.readlines(2)

Readlines: This method will return list of all the lines of given file. Each and every element in a list ends with '\n'

Syn: file_pointer_variable.readlines()

Ex: print fp.readlines()

Note: To work with a file in read operation mode the file should exist in local machine.

Close: This method will remove file-pointer object. (used to close the file)

Syn: file_pointer_variable.close()

Task:

Data.txt – file name

Company	month	year	open Bal	close Bal
IBM	Jan	2016	1011	1008
CSC	Jun	2014	8649	7333
TCS	Aug	2012	6484	9473
WIPRO	Mar	2010	9733	9476

Q) write a program which will return the open and closing balance of given company for given date (mm/yy)

input: (C.name, month, year) – **Ans**: output(name, month, year, open Bal, Close Bal)

Q) write a program to list all the company names which work in loss.

Ans)

```
# DATA WITH PROFIT OR LOSS FOR SELECTED COMPANY,MM,YY

data=open('sensex.txt')

x=data.readlines()

y= [ele.split() for ele in x]

def required(a=raw_input("Enter Company Name : "),b=raw_input("Enter Month : "),c=raw_input("Enter Year : ")):
    z=[a,b,c]
    return z

z=required()

a,b,c=z

ur_data= filter(lambda y:y[0]==a and y[1]==b and y[2]==c, y)
```

```
ur_data=ur_data[0]

ur_data.append(int(ur_data[-1])-int(ur_data[-2]))

print ur_data[0],": O/B:" ,ur_data[-3],": C/B:",ur_data[-2],":
N/P:",ur_data[-1]

print '-'*170

# ALL DATA WITH PROFIT OR LOSS

y=y[1:]

for i in y:

        pl=(int(i[4]))-(int(i[3]))

        i.append(pl)

        print i
```

Write operation mode: to open any file in write operation mode then use below syntax.

Syn: open('fileanme.txt','w')

Ex: fp= open("file_op.txt",'w')

In write operation mode we can perform 2 operations like adding single string to the file and adding multiple strings to the file.

Write: this method is used to add a single string into the file. This method takes exactly 1 parameter. That is string to add.

This method will not return anything.

Syn: file_pointer.write("any string"")

Ex: fp.write("hellow\n")

Write lines: This method is used to add multiple strings into the file. This method takes exactly 1 argument/parameter that is group of elements (data structures).

Syn: file_pointer.write([data strcture])

Ex: fp.writelines(['line 1\n','line 2'])

Ex: fp.writelines(['hi\n',str(123),'world'])

Append mode: To work with files in append operation mode then use ‘a’ as second parameter in open method.

We can perform all the operations which are available in write operation mode but the data will be added to file at the end.

Write method in append mode : this method is used to append a single string to the file.

Syn: open(“file_name.txt”,’a’)

Ex: fp= open("file_op.txt",'a')

fp.write("hellow\n")

fp.writelines(['hi\n',str(123),'world\n'])

Note: other file operation modes:

- r+
- rb+
- w+
- wb+

Seek : This method is used to reposition the file pointer to the given index no. this method takes exactly 1 parameter that is index no.

Syn: file_pointer.seek(index no.)

Ex: zp.seek(0)

Iterators: Iterators or containers which holds group of elements in an object format.

The object name will contain iterator key word.

To create an iterator we may use **iter** method:

Syn: iterator_name = iter([data structure])

Use **next** method to access the data from iterator.

Ex: x=iter([1,2,3,4,5])

print x.next() : 1

print x.next() : 2

print x.next() : 3

Note: **next** method will return " strop iteration " error if no elements are left in iterator.

We cannot control iteration behavior as no conditions will apply on iterable data.

Generators: to avoid the problems caused by iterators use generator every generator is an iterator but not vice versa. Generators are an object which holds conditional formatted data. Use a function along with yield key word to create a generator. The generators name will be the function name.

Syn: def generator_name(data):

Yield ----

Ex: l=[1,2,3,4,5,6,7,8,9,10]

y= gen_rator(l)

print y.next()

Ex2: def even(l):

```
        yield [i for i in l if i%2==0]
        o=even(range(10))
print o.next()                    Ans:  [0, 2, 4, 6, 8]
```

Decorators : decorators are wrapper functions in python which holds another function.

Decorator function will take the another **function name** (function object) as **parameter.**

A decorator function should declare on top of any function by declaring the decorator function name on top of main function which processed by @ symbol

```
Syn:  @decorator_name                ----        (decorator function)
    Def function_name(Parameters):   -----       (Main function)
        ----------------------------------
```

Note: if we are calling main function, the decorator function will execute first then based on the output of the decorator function the main function will execute.

Syntax for defining decorator function:

```
Syn:  def decorator_name(function_name)
        Def wrapper(parameter)
            #operation
            # return some data
            # if success
```

```
        # call main function
    # call main function
    Function_name(parameters)
```

Examples of decorators: class method, static method , login required

MODULES: A module is a container which holds required data in the form of python objects (everything in python is object) every python file will be treated as a module.

Modules mainly implemented for achieving modularity mechanism (accessing the existing data in some other file. The main purpose of modules is reusability.

Types of modules: Python supports 3 types of modules they are:

- User defined modules
- System in built modules – (internal / default modules)
- Third party modules - (Exteral Modules)

User defined Modules : The modules which are created by users are known as user defined modules. To access the module data In another files python provides 2 mechanisms they are **import** and **from-import**

Ex: test.py --- Higher level

```
def add(a,b):
    return a+b
def mul(a,b):
    return a*b
def power(a,b):
    return a**b
```

Sublevel or lower level data

Import : This keyword is used to access higher or outer level data available in a module. Using import statement all the sub functionality or sub level data occupies the memory in python.

Syn: **import module** name

To access sub-level functionality use ‘ . ‘ symbol.

Ex: import test

print test.add(10,20)

print test.mul(20,30)

print test.power(2,2)

Note: import statement on higher level data will cause higher memory consumption as python will allocate memory for all sub functionalities.

To overcome these scenario we can use **from-import** statement.

From-import :

Syn: from **outerleve file** import function_name, function_name ...

Or from **outerleve file** import * (import every thing)

System in-built modules :

- **Date time** : to work with date and time
- Os : to work operating system commands
- Sys : system driver commands
- **Json** : serialization or deserialization
- **Xml** :
- Math : to work with mathometial functions
- Hine :
- Setup tools : filter tools
- Collections: for data manupulations
- **Csv** : to work with comma seperated files
- Requests
- Urllib } web crawling
- Urllib2 }

- Threading : for simultaneous crawling
- Socket : to transfer the data
- Subprocess : for simultaneous crawiling
- **Html** : to work with html files
- random
- Smptlib:
- Email
- **Numpy** : advanced data structures,
- **Unittest** : Testing

Third party modules: the modules which are not available in system and not created by user are known as third party modules (created neither by user nor by python version)

Third party modules are available in dynamic places and can be downloaded from internet.

- Xlrd : to read excel file
- Xlwt : to write excel file
- Paypal : payment gateway
- Twilio : messaging tool
- **Dijango** : web application create
- Request
- Opencafe : for aumati0n testing
- Mysqldb : to connect database
- Pycong2 :
- Way2sms :
- Oruth
- Oracle :
- Sql :

Task:

How to send email

Write a program to sort dectonaries based on keys

OOPS: OOPS stands for Object Oriented Programming Structure, which denotes a particular structure which is created by grouping multiple objects. The advantages of oops are: Extendibility, Security and Reusability.

The major concepts in oops are class and Objects (instance)

Class: a class is an entity which holds multiple objects (group of data). A class can contain any kind of data.

Syn: class class_name:

Ex: class A:

    x=10
    y=20
    l=[10,20,30]

Object / Instance : Object or Instance is the reference of the class. Which is used to access the class data.

Syn: instance_name = class_name()

Ex: obj=A()

Self: self is keyword in python which is equal to class reference (instance).

Self keyword can declare and utilize only in class functions. By using self we can access all the class data.

Ex: class A:

```
    x=10
    y=20
    l=[1,2,3]
    def add(self,a,b):
        return self.x+self.y
```

obj=A()

print obj.add(20,30)

Constructor : Constructor is a function which will invoke automatically when an object created for given class.

Constructor method will not return any data (return type is 'None').

Constructor is mainly designed for initializing the pre-required data.

Ex: class A:

    Def __init__(self,x,y):

        Print " I am constructor"

        Self.x=x

        Self.y=y

    Def add(self):

        Return self.x+self.y

Obj=A(10,20)

Obj.add() Ans: 30

Method Overloading: The process of declaring 2 or more functions with same function name and with different signature(arguments set) is known as method overloading.

As python does not allow to create multiple functions with same function name we cannot achieve method overloading directly.

Reather we can use default parameter mechanism to achieve method overloading.

Ex: class A():

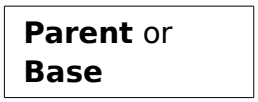
```
    def add(self,*a):
        return sum(a)

obj=A()
a=[10,20,30,40]
#print obj.add(10,20,30,40)
print obj.add(*a)
```

Method Overriding (MOR): The process of creating 2 or more functions with same function name and same signature is known as method overriding. Method overriding is also not possible in python. Rather place the functions in different classes to achieve method over riding.

Inheritance: The process of acquiring the properties from one class to another is known as inheritance. Inheritance is used for achieving extendibility without changing existing data.

In python we can derive inheritance into 3 types they are: **single inheritance, multiple inheritance, multilevel inheritance.**

single inheritance: The inheritance which contains one base class and one derived class is known as single inheritance.



Note: To map inheritance process declare the base class name in derived class definition with in paranthasis.

Syn: Class Base:

Class Derived(Base)

```
class A:
        x=10
        def xxx(self):
                return 'I AM IN CLASS A'


class B(A):
        y=20
        def yyy(self):
                return 'I AM IN CLASS B'


obj=B()
print obj.y
print obj.x
print obj.xxx()
print obj.yyy()
```

multiple inheritance: Acquiring the properties of multiple base classes in derieved class is known as multiple inheritance.

To define multiple inheritance declare multiple class names in derived class definition within parentheses.

Syn: Class Base1:

C – Derived

Class Base2:

Class Derived(Base1, Base2)

Note: Multiple will work based on MRO.

MRO: Stands for Method Resolution Order, it will take care of executing the methods in an particular order.

According to MRO Python will search required data in the class for which we have created an object. If the data is not available then python will search In inherited classes in the direction of left to right.

Multilevel Inheritance:

Class A:

Class B(a):

Class c(B)

Class d(c)

Obj=d()

Print obj.x

The process of acquiring the properties from base class to derive class and derived class to another derieved class and so on.. is known as multilevel inheritance.

The combination of multiple and multilevel inheritance in python is not achievable.

Super: Super method is used to change method resolution order.

Super method will work only on **new style classes**(The class which inherit ‘object’).

This function accepts 2 parameters **P1**: represents class name **P2**: class instance(self)

We can call the required function by calling the function name.

Ex:
```
class A(object):
        def xyz(self,a,b):
            return a*b
class B(A):
    def xyz(self,a,b):
        return super(B,self).xyz(a,b)
obj=B()
print obj.xyz(10,20)
```