(general purpose, high level programming language)

↑

13/6/18

Basics of Python Programming

Q. Difference b/w Py2 and Py3 ?

Ans In Py2, 'print' statement is not a function while in Py 3, it is a function and so can't be invoked without a parenthesis.

Q. Difference b/w Py and C?

Ans print statement automatically includes a new line in Py unlike C.

Q. What are some of the language features?

Ans i) readable and shorter codes, ease of writing

ii) supports multiple programming paradigms, like OOPS, imperative and functional programming or procedural.

iii) inbuilt functions for almost all frequently used concepts.

iv) No separate compilation & execution steps like C, C++. Program is run directly from the source code, no need to worry about linking and loading with libraries.

v) It is platform independent.

vi) High-level language, so need to worry about low-level details such as managing memory, etc.

vi) More emphasis on solution than syntax.

vi) can be used within C/C++ i.e it is embeddable.

vi) Inbuilt memory management techniques.

vi) has a rich library support.

Q. Difference b/w Python and Java?

Ans i) Python is concise and compact unlike Java.

ii) Python uses indents while Java uses braces for structuring codes.

iii) Python is dynamically typed i.e no need to declare anything unlike Java which is statically typed.

iv) No type casting required in Python when using container objects unlike in case of Java.

v) Just like Java, Python needs some form of runtime on system (JVM / python runtime)

Q. What are the cons of Python?

Ans i) Slow speed of execution compared to C, C++.

ii) Many design restrictions bcoz of being dynamically typed. It requires more testing time and errors show up when applications are finally run.

Q. What are some more differences b/w Py 2 and Py 3?

Ans i) Division operator

$\frac{7}{5} = 1$, $-\frac{7}{5} = -2$ in Py 2

$\frac{7}{5} = 1.4$, $-\frac{7}{5} = -1.4$ in Py 3.

ii) In py 2, implicit str type is ASCII, but in py 3, implicit str code is Unicode.

iii) range(3) = [0,1,2] in Py 2, xrange(3) returns iterator object which generates number when needed.

range(3) in Py 3 ≡ xrange(3) in Py 2

Note If we need to iterate over the same sequence multiple times, we prefer range() as range provides a static list while xrange()

reconstructs the sequence every time. xrange() doesn't support slices and other list methods.

iv) To get Python 3 support in our Py 2 code, we use _future_ module. Importing it, makes Py 3 applicable in Py2. e.g.

```
>> from _future_ import print_function
>> print('Hello')        # as per Py3.
   Hello.
```

Q Does Py support method overloading?

Ans No, Py doesn't. We may overload the methods but can use only the latest defined method.

Q How to check if given is a keyword?

Ans
```
import keyword
    if keyword.iskeyword(B):
        print('Yes')
```

Q. Give example of some keywords.

Ans i) assert: used for debugging purposes, to check the correctness of code. When it is false, assertion error is raised.

ii) class: to declare user defined classes.

iii) del: used to delete a reference to an object. Any variable or list value can be deleted using del

```
e.g.  a = [1, 2, 3]
      del a[1]     // this gives a = [1,3]
```

iv) pass : null statement in python. Nothing happens when this is encountered. It is used to prevent indentaⁿ errors and used as a place holder.

v) as : to create alias for the module imported.

vi) lambda : used to make inline returning functions with no statements allowed internally

vii) return v/s yield : return returns from the function while yield returns a generator.

viii) is : used to test object identity, i.e to check if both objects take same memory location or not.

ix) global : used to define a variable inside the fᴺ to be of a global scope.

x) non-local : same as global but unlike global, it declares a variable to point to variable of outside enclosing fᴺ.

Note Python programs are not compiled, rather they are interpreted.

Q What is the max possible value of an integer in Python?

Ans In Py value of an integer is not restricted by the number of bits and can expand to the limit of the available memory. In Py 3, there is only 1 type of int for all integer values. In py 2 there is 'int' and 'long int'.

Q. How to do matrix transpose in python?

Ans m = [[1,2], [3,4], [5,6]]

i) rez = [[m[j][i] for j in range(len(m))] for i in range (len(m[0]))]

ii) using zip : zip returns an iterator of tuples, with the ith tuple having ith element from each of the argument sequences or iterables. * unzips an array.

```
matrix = [(1, 2, 3), (4, 5, 6), (7, 8, 9), (10, 11, 12)]
t_matrix = zip(*matrix)
t_matrix
>> (1, 4, 7, 10)
   (2, 5, 8, 11)
   (3, 6, 9, 12)
while t_matrix = [[1,4,7,10][2,5,8,11][3,6,9,12]], when
      t_matrix = map(list, zip (*matrix))
```

iii) using numpy : numpy.transpose (matrix)

Q. Give a good example for local and global variable use.

Ans

```
a = 1
# Uses global becoz there is no local 'a'.
def f():
  print 'Inside f():', a
# variable 'a' is redefined as a local
def g():
  a = 2
  print 'Inside g():', a
```

```
# Uses global keyword to modify global 'a'.
def h():
    global a
    a = 3
    print 'Inside h():', a
# Global scope
  print 'global :', a
  f()
  print 'global :', a
  g()
  print 'global :', a
  h()
  print 'global :', a
```

Output:

```
global : 1
Inside f() : 1
global : 1
Inside g() : 2
global : 1
Inside h() : 3
global : 3
```

Q. What are partial functions?

Ans They allow us to fix a certain no. of arguments of a f^n and generate a new function.

e.g.
```
from functools import partial
 # A normal fn
 def f (a, b, c, x) :
     return 1000*a + 100*b + 10*c + x
# A partial fn that calls f with a as 3, b as 1 and c as 4.
 g = partial (f, 3, 1, 4)
# Calling g ()
 print ( g (5))
```

Output : 3145

e.g.
```
from functools import *
 # A normal function
 def add (a,b,c)
     return 100*a + 10*b + c
 # A partial fn with b=1 and c=2
  add_part = partial(add, c=2, b=1)
 # calling partial fn
  print (add_part(3))
```

Output : 312

Note Partial f^N can be used to derive specialized functions from general function and ∴ help us to reuse our code. This feature is similar to bind in C++.

Q. How can you pass a list as argument in f^N with each element of list working as an argument?

Ans

e.g.
```
def fun (a, b, c, d) :
        print (a, b, c, d)
    my_list = [1, 2, 3, 4]
    fun (*my_list)   // unpacking list
```

14/6/18

Q Given an example to show both packing and unpacking

Ans

```
def fun 1 (a,b,c):
    print (a,b,c)
# below fn is an example of packing where all arguments passed to fn2
# are packed into tuple *args.
# first define fn then convert args tuple to a list so it can be modified.
def fun2 (*args):
    args = list (args)
    args [0] = 'I'
    args [1] = 'am'
# unpacking args and calling fun 1 ()
    fun1 (*args)

fun 2 ('Hello', 'Hi', 'Mahima')
```

Output: I am Mahima

Q What is done in case of packing unpacking dictionaries?

Ans. e.g.

```
def fun (a,b,c):
    print (a,b, c)

d = {'a': 2, 'b': 4, 'c': 10}
fun (**d)
```

Output: 2 4 10

So, fun(1, **d) ≡ fun(1, b=8, c=16)

e.g.
```
def fun(**kwargs):            // kwargs is a dictionary
    print (type(kwargs))

    for key in kwargs:
        print ("%s = %s" % (key, kwargs [key]))

fun (name = 'geeks', ID = '101', language = 'Python')
```

Output:
```
<class 'dict'>
language = Python
name = geeks
ID = 101
```

It is useful in sending variable number of arguments to functions. Modificaⁿ of arguments become easy this way but at the same time validation is not proper so they must be used with care.

Q How to print without a new line?

Ans
```
print ("welcome to", end = ' ')      // end = '\n' by default, to end
print ("my home", end = ' ')         // without a new line set end to
                                     // ' ' or space.
```

e.g.
```
print ("Python", end = '@')
print ('Geeks")
```

Output: Python@Geeks

How is type conversion in Python?

ns

1) int (a, base): It converts any datatype to integer. 'Base' specifies the base in which string is if data type is string.

2) float (): it converts any datatype to a floating point number.

e.g.

```
S = '10010'
c = int (S, 2)   // converting to integer base 2
print c
e = float(s)
print (e)
```

Output: 18, 10010.0

3) ord () : to convert a char to int
4) hex () : to " int to hexadecimal string
5) oct () : to " " " octal string.
6) tuple () : to convert to a tuple
7) set () : this fn returns the type after converting to a set.
8) list () : this " is used to convert any datatype to a list type.
9) dict () : to convert a tuple of order (key, value) into a dictionary.
10) str () : to convert integer into a string.
11) complex (real, imag) : converts real to complex number.

e.g.

```
ord (4)   = 52
hex (56)  = 0x38
oct (56)  = 0o70
```

```
tuple ('geeks') = ('g', 'e', 'e', 'k', 's')
set ('geeks') = {'k', 'e', 's', 'g'}
list ('geeks) = ['g', 'e', 'e', 'k', 's']

complex (1,2) = 1+2j
str (1)  = 1
// tup = (('a',1 ), ('f', 2), ('g', 3))
dict (tup) = {'a':1, 'f':2, 'g':3}
```

Q. Byte objects v/s strings in Python?

Ans i) Byte objects are sequence of bytes while strings are sequence of characters.

ii) Byte objects are in machine readable form internally, strings are only in human readable form.

iii) Byte objects can be directly stored in disk but strings need encoding as they are not machine readable.

Note PNG, JPEG, MP3, ASCII, UTF-8 are different forms of encodings. An encoding is a format to represent audio, images, text, etc in bytes. Converting strings to byte objects is termed encoding. Default encoding technique is 'UTF-8'. Encoding task is achieved using encode() which takes encoding technique as argument.

e.g.
```
a = 'Mahima'          # initializing string
c = b'Mahima'         # initializing a byte object
d = a.encode('ASCII')
if (d==c):
    print ('Yes')
```

Output : Yes.

Decoding is to convert byte object to string, implemented using decode. Encoding and decoding are inverse processes and a byte string can be decoded to character string if we know which encoding was used to encode it.

e.g.
```
a = 'Mahima'
c = b'Mahima'
d = c.decode ('ASCII')
if (d = = a):
    print ('Yes').
```

Output: Yes.

Q. Explain the logical and bitwise Not Operators on Boolean.

Ans e.g.
```
a = not True
b = not False
print a
print b
```

Output : False
True

e.g.
```
a = True
b = False
print ~a        // bitwise not operator (~)
print ~b        // returns the complement
                // of a number
                // True = 1
                // False = 0
```

Output : -2
-1

Note Java doesn't allow ~ operator to be applied on boolean values, "logical not or !" is meant for boolean values and "bitwise not or ~" is for integers.

e.g.
```
print(['a', 'b'][bool('g')])
```
Ans b

// This means print 'a' if arg
// passed to bool is zero else print
// 'b'.

Q. What are ternary operator in Python?

Ans Ternary operators also known as conditional expressions are operators that evaluate something based on a condition being T or F. It simply allows to test a condition in a single line replacing the multi line if-else making the code compact.

e.g.
```
a, b = 10, 20
min = a if a<b else b
print (min)

Output: 10
```

Note Conditional operators have lowest priority amongst all Python operations.

e.g.
```
a, b = 10, 20
# Use tuple for selecting an item
print ([b, a) [a<b])
# Use dictionary for selecting an item
print ({True: a, False: b} [a<b])
# lambda is more efficient than above
# two methods bcoz in lambda we are
# assure that only one expression will
# be evaluated unlike in tuple & dict
print((lambda: b, lambda: a) [a<b] ())

Output: 10
        10
        10
```

```
print ("Both a and b are equal" if a==b else "a is greater than b" if a>b else
        "b is greater than a").
```

b, a = 20, 10

e.g. $\text{min} = \underbrace{a<b}_{\text{expression}} \text{ and } \underbrace{a}_{\text{on true}} \text{ or } \underbrace{b}_{\text{on-false}}$

// If a<b, a is assigned else b is assigned.
// It doesn't work if a is 0 or False, bcoz
// if so happen then on-false is evaluated
// always.

Q. What about ++ and -- operators in python?

Ans. They don't exist in python. Instead of for (i= 0; i<5; ++i)
we write for i in range (0,5) :
print (i).

Q. How does division operator works in python?

Ans e.g. print 5/2 // 2
print -5/2 // -3

'/' operator works as a floor division for integer arguments. However it returns a float value if one of the arguments is a float.
e.g. print 5.0/2 // 2.5
print -5.0/2 // -2.5

Note '//' is real floor division operator which returns floor value for both integer and floating point arguments.

print 5/2 // 2
print -5/2 // -3
" 5.0/2 // 2.0
" -5.0/2 // -3.0

* In Py3, '/' operator does floating point division for both int and float arguments.

Q. Explain the Any / All built ins provided by python?

Ans They are used for successive And/OR.

Any : returns T if any of the items is true, returns F if empty or all are F. It works as a sequence of OR and short circuits the execution as soon as the result is known.

All : returns T if all items are T (or if iterable is empty). It works as a sequence of AND on the provided iterables & short circuits the execu^n as soon as result is known.

e.g.
```
print(all([True, True, True, True]))   // True
print(all([F, F, F]))          // False
print(all([F, T, F]))          // False
print(any([F, F, F]))          // False
print(any([F, T, F, F]))       // True
```

add : normal operator
iadd : inplace operator

Q. Give example of inplace v/s standard operators.

Ans
```
import operator
x = 5
y = 6
a = 5
b = 6
z = operator.add(a,b)
p = operator.iadd(x,y)
```

Output:
```
z = 11
p = 11
a = 5
x = 5
```

Case I : immutable targets

```
import operator
a = [1,2,4,5]
z = operator.add(a, [1,2,3])
print(z,a)
p = operator.iadd(a, [1,2,3])
print(p,a)
```

Output :
```
z = [1,2,4,5,1,2,3]
a = [1,2,4,5]
p = [1,2,4,5,1,2,3]
a = [1,2,4,5,1,2,3]
```

Case II : mutable targets.

15/6/18

Q. What are some operator functions in python?

Ans. i) add (a,b) : a+b

ii) sub (a,b) : a-b

iii) mul (a,b) : a*b

iv) truediv (a,b) : a/b

v) floordiv (a,b) : a//b

vi) pow (a,b) : a**b

vii) mod (a,b) : a % b

viii) lt (a,b) : returns True if a<b else false.

ix) le (a,b) : " " " a<=b " "

x) eq (a,b) : " " " a==b " "

xi) gt (a,b) : T if a>b else false.

xii) ge (a,b) : T if a>=b " "

xiii) ne (a,b) : T if a!=b " "

} operations under the module operator.

Q. What are some other operators under the operator module?

Ans

i) setitem (ob, pos, val) : to assign the value at a particular posi^{n} in container.
ob[pos] = val

ii) delitem (ob, pos) : del ob[pos]

iii) getitem (ob, pos) : to access the value at ob[pos]

iv) setitem (ob, slice(a,b), vals) : to set the values in a particular range in the container. obj[a:b] = vals

v) delitem (ob, slice(a,b)) : del obj[a:b]

vi) getitem (ob, slice (a,b)) : obj [a:b]

vii) concat (obj1, obj2) : to concatenate two containers. obj1 + obj2

viii) contains (obj1, obj2) : checks if obj2 is +nt in obj1 i.e obj2 in obj1

ix) and_(a,b) : to compute bitwise and. a & b

x) or_(a,b) : to compute bitwise or. a|b

xi) xor (a,b) : to compute bitwise xor. a^b

xii) invert (a) : to " " inversion. ~a. eg ~1 = 1

Q Give example of chaining comparison operator in python? {x = 5)

Ans

1 < x < 10	True
10 < x < 20	False
x < 10 < x*10 < 100	True
10 > x < = 9	True
5 == x > 4	True
x=5 < 12 > 0 is not 15 is 15	True
0 is 0 > 15 is not 12	False

All these expressions are equivalent to an AND between them and yield boolean values.

Q What are the basic operators in python?

Ans Arithmetic operators : +, -, *, /, //, %.

Relational operators : >, <, ==, !=, >=, <=

logical operators : and, or, not

Bitwise right shift

Bitwise operator : &, |, ~, ^, >>, <<

Bitwise AND | Bitwise XOR Bitwise left shift

Bitwise OR

Assignments operator : =, +=, -=, *=, /=, %=, //=, **=, &=, |=, ^=, >>=, <<=.

Special operators: Identity operators < is / is not (True if value is found in sequence)

e.g. 3 is 3 True

'yes' is 'Yes' True

bcoz lists are mutable. ← [1,2,3] is [1,2,3] False

Membership operators < in / not in (True if operands are identical)

e.g 'b' in y = {3:'a', 4:'b'} False

16/6/18

Q What does bool() in python return?

Ans bool([x])

In general, bool() takes only 1 parameter on which the standard truth testing procedure is applied.

Conditions when bool() returns false:

i) If a False value is passed.

ii) If None is passed

iii) If an empty sequence is passed, such as (), [], '', etc.

iv) If zero is passed in any numeric type, such as 0, 0.0, etc.

v) If an empty mapping is passed, such as {}.

vi) If objects of classes having _bool_() or _len()_, returning 0 or False.

Q Give an example of pass statement.

Ans
```
for letter in 'geekforgeeks':
    pass
print 'last letter' ;letter
```

Output : s

Q. Diff b/w while and if?

Ans

e.g.
```
cars = ['A', 'B', 'C']
i=0
while (i<len(cars)):
    print cars[i]
    i+=1
```

Output :
```
A
B
C
```

e.g.
```
cars = ['A', 'B', 'C']
for x in cars
    print x
```

Output :
```
A
B
C
```

Note use of while loop is not appreciated in python as
i) it creates no compactness
ii) prone to errors in large scale programs or designs
iii) No automatic ↑ in i
iv) length of iterable should be know in priori.

e.g.
```
cars = ['A', 'B', 'C']
for i in range (len(cars)):
    print cars [i]
```

e.g.
```
for i, x in enumerate(cars):
    print x
```

e.g.
```
for x in enumerate (cars):
    print (x[0], x[1])
```

enumerate takes i/p as iterator, list, etc and returns a tuple containing index and data at that index in the iterator sequence

All e.g. give same output but the last e.g. with print (x[0], x[1]),

<u>Output</u>:
```
(0, 'A')
(1, 'B')
(2, 'C')
```

<u>e.g.</u>
```
print enumerate (cars)
```

<u>Output</u>:
```
[(0, 'A'), (1, 'B'), (2, 'C')]
```

<u>e.g.</u>
```
for x in enumerate (cars, start = 1):
        print (x[0], x[1])
```

<u>Output</u>:
```
(1, 'A')
(2, 'B')
(3, 'C')
```

<u>Q.</u> Give e.g. by using zip function.

<u>Ans</u> It is useful in combining similar type iterators (list-list or dict-dict) data items at ith position. It uses shortest length of these i/p iterators. Other items of larger length iterators are skipped. In case of empty iterators, it returns No output.

<u>e.g.</u>
```
cars = ['A', 'B', 'C']
access = ['P', 'Q', 'R']
for c, a in zip (cars, access):
    print ("car : %S, Accessory required: %s" % (c, a))
```

Output:

```
car : A , Accessory required : P
car : B , Accessory required : Q
car : C , Accessory required : R
```

Q. e.g. of unzipping?

Ans

```
l1, l2 = zip (*[('Aston', 'GPS'), ('Audi', 'Car Repair'),
               ('BMW', 'Gear')])
print (l1)
print (l2)
```

output:

```
('Aston', 'Audi', 'BMW')
('GPS', 'Car Repair', 'Gear')
```

Q What are counters in Python?

Ans Counter is a container included in the collections module. Containers are objects that hold objects. They provide a way to access the contained objects and iterate over them. Examples of built in containers are tuple, list and dictionary. Others are included in collection module.

A Counter is a subclass of dict. ∴ it is an unordered collection where elements and their respective count are stored as dictionary.

```
class collections. Counter ([iterable-or-mapping])
```

e.g.
```
from collections import Counter
print Counter (['B', 'B', 'A', 'B', 'C', 'A', 'B', 'B', 'A', 'C']) // c̄ sequence of items
print Counter ({'A': 3, 'B': 5, 'C': 2}) // with dictionary
print Counter (A=3, B=5, C=2)  // with keyword arguments
```

Output:
```
Counter ({'B': 5, 'A': 3, 'C': 2})
Counter ({   "    "    "  })
Counter ({   "    "    "  })
```

e.g.
```
from collections import Counter
coun = Counter()
coun.update ([1, 2, 3, 1, 2, 1, 1, 2])
print (coun)
coun.update ([1, 2, 4])
print (coun)
```

Output:
```
Counter ({1:4, 2:3, 3:1})
Counter ({1:5, 2:4, 3:1, 4:1})   // data is ↑ed & not replaced.
```

e.g.
```
from collections import Counter
c1 = Counter (A=4, B=3, C=10)
c2 = Counter (A=10, B=3, C=4)
c1.subtract (c2)
print (c1)
```

Output:
```
Counter ({'C': 6, 'B': 0, 'A': -6}
```

e.g.

```
coun = Counter (a=1, b=2, c=3)
print (coun)
print ( list (coun. elements ()))
```

Output:

```
Counter ({'c': 3, 'b': 2, 'a': 1})
['a', 'b', 'b', 'c', 'c', 'c']
```

e.g.

```
coun = Counter (a=1, b=2, c=3, d=120, e=1, f=219)
for letter, count in coun. most_ common (3):
        print ('%s : %d' % (letter, count))
```

Output:

```
f : 219
d : 120
c : 3
```

Note Iterator in python is any python type that can be used with a 'for in loop'. Python lists, tuples, dicts and sets are all examples of inbuilt iterators.

Q Mention some iterator functions in Python.

Ans

```
import itertools
import operator
li1 = [1,4,5,7]
li2 = [1,6,5,9]
li3 = [8,10,5,4]
print ('sum after each iteraⁿ is:', end = "")
print (list (itertools. accumulate (li1)))
print ('product after each iteraⁿ is :", end = " ")
```

Note accumulate (iter, func) : default func is add.

```
print (list (itertools. accumulate (li1, operator. mul)))
print ("All values in mentioned chain are:", end = " ")
print ( list (itertools. chain (li1, li2, li3 )))
```

Output: Sum after each itera^n is : [1,5,10,17]
Product " " " " : [1,4,20,140]
All values in mentioned chain are: [1,4,5,7,1,6,5,9,8,10,5,4]

e.g. # code to demonstrate the working of islice() & starmap().

```
import itertools
li = [2,4,5,7, 8, 10, 20]
li1 = [(1,10,5), (8,4,1), (5,4,9), (11,10,1)]
# using islice() to slice the list acc. to need starting from
# 2nd till 6th index skipping 2.
print (list ( itertools. islice (li, 1, 6, 2)))
# using starmap() for selec^n value acc. to f^n
# selects min of all tuple values.
print ("The values acc. to func are :", end = " ")
print (list (itertools. starmap (min, li1)))
```

Output: The sliced list values are : [4,7,10]
The values acc. to f^n are : [1,1,4,1]

e.g.

```
li = [2,4,6,7,8,10,20]
li = iter ( li)
# using takewhile () to print values till condi^n is false.
print ( list (itertools. takewhile (lambda x: x%2 ==0, li)))
```

Note <u>tee</u> (iterator, count) splits the containter into a number of iterators mentioned in argument.

```
# using tee () to make a list of iterators
 it = itertools . tee (iti , 3)
print (" The iterators are:")
for i in range (0,3) :
    print ( list (it [i]))
```

Output:

```
[2,4,6]
The iterators are:
[2,4, 6 , 7, 8, 10, 20]
   "    "    "
   "    "    "
```

Q Explain Generators in Python.

Ans

1) Generator f^N: If the body of a f^N contains <u>yield</u>, the f^N automatically becomes a generator f^N.

e.g.

```
def f^N():
    yield 1
    yield 2
    yield 3

for value in f^N():
    print (value).
```

Output:

```
1
2
3
```

e.g.

```
def f^N():
    yield 1
    yield 2
    yield 3

x = f^N()
# iterating over x (generator object) using next.
print (x.next());
  .   .   .   ;
  .   .   .   ;
```

Output:

```
1
2
3
```

2) Generator Object : Generator fn returns a generator object which can be used either by calling the next method on the generator object or using the generator object in a 'for in' loop. This object is thus iterable.

Q. Make a generator for fibonacci Numbers

Ans

```
def fib(limit):
    a, b = 0, 1
    while a < limit:
        yield a
        a, b = b, a+b
x = fib(5)
# iterating over the generator object using 'for in' loop:
for i in fib(5):
    print(i)
# iterating over generator object using next.
print(x.next());
   "        "
   "        "
   .        .
   "        .
```

Note Generators provide a space efficient method for data processing (of huge data) as only parts of file are handled at one given point of time.

Output:

```
0
1
1
2
3
0
1
1
2
3
```

Q. Give enample for looping techniques.

Ans e.g.

```
d = {'geeks' : 'for', 'only' : 'geeks'}
print ("using iteritems")
for i, j in d.iteritems ():
    print i, j
print ("using items is")
for i, j in d.items ( ):
    print i, j
```

Output:

```
using iteritems
geeks for
only geeks
using items is
geeks for
only geeks
```

e.g.

```
lis = [1, 3, 5, 6, 2, 1, 3]
for i in sorted ( lis ):
    print(i, end = " ")
for i in sorted ( set ( lis ) ):
    print (i, end = " ")
```

Note sorted ( ) prints the container in sorted order.
set ( ) can be combined to remove duplicates.

Output:

```
1 1 2 3 3 5 6
1 2 3 5 6
```

e.g.

```
print ("list in reversed order is:")
for i in reversed(lis):
    print(i, end=" ")
```

Output: 6 5 3 3 2 1 1

e.g.

```
for i in reversed(range(1,10,3)):
    print(i)
```

Output:

```
7
4
1
```

Note: The above techniques are quick to use & reduce coding effort. for, while loop needs the entire structure of container to be changed. They have keywords which tell the purpose at just one glance thus making the code more concise.

Q. Differentiate b/w range() and xrange()?

Ans

range(): returns a list of number created using range() fn.

xrange(): returns generator object that can be used to display numbers only by looping. Only particular range is displayed on demand, so called lazy evaluation.

Differences:

i) range() returns list
xrange() " object.

e.g.
```
a= range (1,10000)
x= xrange (1,10000)

print (type (a))
print (type (x))
```
Output:
```
<type 'list'>
<type 'xrange'>
```

e.g.
```
print (sys.getsizeof (a))
print (sys.getsizeof (x))
```
Output:
```
80064
40
```

ii) variable created by range() takes more memory than the variable storing the range using xrange(). This is bcoz of the diff return types.

iii) bcoz range () returns list, all list opera^n can be applied on range () but list opera^n can't be applied on xrange () being an object.

e.g.
```
a=range (1,6)
x= xrange (1,6)
print (a [2:5])
// print (x[2:5])    // returns error if uncommented.
```
Output:
```
[3,4,5]
```

iv) xrange() is faster bcoz it evaluates only the generator object containing only the values that are required by lazy evaluation.

Use xrange() reconstructs integer object everytime unlike range () which has real integer objects.

Use range () as xrange() is deprecated in Python 3.

range() is faster if iterating over the same sequence multiple times.

Q. find output of

```
n = 123
for i in n:
    print(i)
```

Ans Error!

bcoz objects of type int are not iterable instead a list, dict or a tuple should be used.

Note [::-1] besides a list reverses the list.

e.g. [1, 2, 3] [::-1] = [3, 2, 1]

Q. find o/p of:

```
n = ['ab', 'cd']
for i in n:
    n.append (i.upper())
print(n)
```

Output: Error / ∞ looping

// bcoz n is going on ring

Q. find o/p of:

```
n = ['ab', 'cd']
for i in n:
    i.upper()
print(n)
```

Output: ['ab', 'cd']

// bcoz upper() is not modifying string in place, New string is not stored anywhere.

Q. find o/p:

```
True = False
print (True)
```

Ans Error Syntax

bcoz True is a keyword. So, its value can't be changed.

Python Glossary :

i) block : section of code which is grouped together

ii) class : template for creating user defined objects.

iii) compiler : translates program written in high level to low level language

iv) dictionary : a mutable associative array of key & value pairs. can contain mixed types (keys & values). Keys must be a hashable type.

v) docstring : string literal that occurs as the first statement in a module, fn, class or method definition.

vi) _future_ : pseudo module that enables new language features which are not compatible with current interpreter.

vii) evaluation order : python evaluates expressions from left to right, but while evaluating an assignment, RHS is evaluated before LHS.

viii) expression : python code that produces a value.

ix) filter (function, sequence) : it returns a sequence consisting of those items for which function (item) is true in given sequence.

x) float : an immutable floating point number.

xi) generator : fn which returns an iterator.

xii) immutable : can't be changed after its created.

xiii) int : immutable integer of unlimited magnitude.

xiv) interpret : to execute a program by translating it one line at a time

xv) lambda : shorthand to create anonymous functions.

xvi) list : mutable list, can contain mixed types.

xvii) literals : notations for constant values of some built in types

xviii) map (f^{n}, iterable, ...) : applies f^{n} to every item of iterable and returns a list of the results.

xix) module : basic unit of code reusability in python. A block of code imported by some other code.

xx) object : any data c̄ state (attributes or value) and defined behaviour (methods).

xxi) Python Package Index : official repository of 3rd party s/w for python.

→ method : f^{n} defined inside a class.

xxii) set : unordered set, contains no duplicates.

xxiii) string : a character string : an immutable sequence of Unicode codepoints.

xxiv) statements : part of a "block" of code.

xxv) tuple : immutable, can contain mixed types.

xxvi) variables : placeholders for text & numbers.

xxvii) yield : returns a value from a generator f^{n}.

Q. o/p for type(type(int))
Ans type 'type'.

Q. print "".join(['a','b','c','d'])
Ans abcd

Q. Chr(ord('A'))
Ans A
bcoz ord converts to ASCII nota^{2} & Chr() converts ASCII to character

Q. z = lambda x : x * 8
z(6)

Ans Output : 48.

Note Each object in Python has a unique id returned by the id() function.

Q. time.time() returns?

Ans current time in ms since midnight, January 1, 1970 GMT (the Unix time).

Note rshift() overloads the >> operator, | overloads or () function.

Note No ++ operator or -- in python.

Q Given a func^n that does not return any value, what value is shown when executed at the shell?

Ans Python explicitly defines the None object that is returned if no value is specified.

Q. 0.1 + 0.2 == 0.3

Ans False bcoz neither of 0.1, 0.2 and 0.3 can be represented accurately in binary. The round off errors from 0.1 and 0.2 accumulate and hence 0.1 + 0.2 ≠ 0.3.

Note $\boxed{\sim x \equiv -(x+1)}$

Note Stub is a simple but incomplete version of a function. It is a placeholder class or f^N that doesn't do anything yet, but needs to be there so that the class or f^N in question is defined. The idea is that we can already use certain aspects of it (such as put it in a collection or pass it as a callback), even without writing its implementation yet.

Q. 3*1**3 = 3 bcoz ** has higher priority than *.

Q. print '{0:-2%.}'.format(1.0/3)

Ans 33.33%

bcoz % converts the 0.33 to percentage w.r.t 1.0.

Q. 'abcefd'.replace('cd','12')

Ans abcefd bcoz no substring 'cd' exists in 'abcefd'.

Q.
```
def f(value, values):
    v = 1
    values[0] = 44
t = 3
v = [1, 2, 3]
f(t, v)
print (t, v[0])
```
Ans 3 44

Q. 'abef'.partition('cd')

Ans ('abef', '', '')

bcoz the separator is not +nt in the string hence the second and 3rd elements of the tuple are null strings.

Q. 'cd'.partion('c')

Ans ('', 'c', 'd')

Q. 'cd'.partition('cd')

Ans ('', 'cd', '')

17/6/18

Q. 'cdbhg'. partion ('db')

Ans ('c', 'db', 'hg')

Note AND has more precedence than OR. NOT has more precedence than AND.

Q.
```
class Geeks :
    def __init__(self, id):
        self.id = id
    manager = Geeks (100)
    manager.__dict__ ['life'] = 49
    print manager.life + len (manger.__dict__)
```

Ans 51

bcoz manager.__dict__ has 2 items 'item' & 'life'.

Q.
```
dict = {1: '1', 2: '2', 3: '3'}
del dict[1]
dict [1] = '10'
del dict [2]
print len (dict)
```

Ans 2

Q.
```
name = [ 'Mahi', 'Gargi', 'Ananya']
pos = name. inden ('Geeks')
```

Ans ↳ Error : 'Geeks' is not in list

Q.
```
a = "Geeks"
b = 13
print a+b
```

Ans Type Error

bcoz pyhon is a strongly typed language, we can't simply concatenate an integer with a string.

Q.
```
List = ['Harsh', 'Mahi']
print List [1] [-1]
```

Ans i

last element in list or last char in string.

Q.
```
geek codes = [1, 2, 3, 4]
geek codes. append ([5, 6, 7])
print gockcodes.
```

Ans [1, 2, 3, 4, [5, 6, 7]]

Q.
```
mycontainer = [10, 20, 30]
mycontainer + = [10]
print mycontainer
```

Ans [10, 20, 30, 10]

Q.
```
class A(object):
   val = 1
class B(A):
  pass
class C(A):
  pass
print A.val, B.val, C.val
B.val = 2        // overwritten
print A.val, B.val, C.val
A.val = 3
print A.val, B.val, C.val
```

Output:
```
111
121
323
```

Q.
```
def gfg(n, l = []):
    for i in range(n):
       l.append(i*i)
    print(l)
    gfg(2)
    gfg(3, [3, 2, 1]
    gfg(3)
```

→ Output:
```
[0,1]
[3, 2, 1, 0, 1, 4]
[0, 1, 0, 1, 4]
```
← Surp

Q.
```
def gfg():
    "Geeks for Geeks"
    return1
print gfg.__doc__[07:11]
```

Ans re Geek,

```
# docstring is defined for gfg()
#method by putting a string on the
#first line after the start of the
# definition.
```

Note ★
```
check 1 = ['A', 'B', 'C', 'D']
check 2 = check 1
check 3 = check 1 [:]
```

Any changes in check2 will reflect in check 1 but changes in check3 will not reflect in either check 1 or check2 bcoz unlike check2 which is a 2nd reference to check1, check 3 is a full copy of check1 which can be modified independently.

Q. l_1 = [19, 20, 21, 22]
l_2 = [14, 16, 96, 25]
$l_1 + l_2$
$l_1 * 2$ // Imp

Ans
[19, 20, 21, 22, 14, 16, 96, 25]
[19, 20, 21, 22, 19, 20, 21, 22] // Imp

Q print 'C:\\ inside C'
print r'C:\\ inside C'

Ans C:\ inside C # consider 'backslash' as special char
C:\\ inside C # it is a raw string and so treats '\' as normal char

Q print '\x25\x26'

Ans %& # \x is an escape sequence meaning that following 2 digits are a hexadecimal no. encoding a char.

Q. list = ['A', 'B', 'C', 'D', 'E', 'F', 'G']

Ans
list [::] = ['A', 'B', 'C', 'D', 'E', 'F', 'G']
list [0:6:2] = ['A', 'C', 'E']
list [:6:] = ['A', 'B', 'C', 'D', 'E', 'F', 'G']
list [:6:2] = ['A', 'C', 'E']
list [::3] = ['A', 'D', 'G']
list [::-2] = ['G', 'E', 'C', 'A']

Note list [x:y:z] : default x = 0 , y = len (list) , z = 1

Note list = ['a', 'b', 'c', 'd', 'e']

print list [10:]

// print list [10] // give Index Error if uncommented

Output: [] // bcoz if slice is accessed with starting index going
// out of bound of length of list, an empty list is
// returned.

e.g. list = ['a', 'b'] * -3

Ans [] // list * N {N = 0 or -ve} returns empty list
// If N is +ve, list is repeated.

Q. dict = {'G':1, 'F':2, 'G':3}

print (dict ['G'])

Ans 3 // G is duplicate key. Duplicate keys are not allowed in python. If
// there are same keys in a dictionary, then the value assigned
// most recently is assigned to that key.

Note Dictionaries are unordered. Any key value pairs can be added at any loc^n within a dictionary. e.g.

temp = {'A': 1, 'B': 2, 'C': 3}

print (key, values, end = " ")

Output: A1 B2 C3
B2 A1 C3
C3 B2 A1
} Any one output is possible.

Note else block following a finally block is not allowed in python. Python throws Syntax error when such format is used.

Q. temp = 'Geeks 22536 for 445 Geeks'

```
data = [x for x in (int(x) for x in temp if x.isdigit()) if x%2==0]
print (data)
```

Ans [2, 2, 6, 4, 4] || example of nested list comprehension. The
|| inner list created contains a list of integers
|| in temp. The outer list only procures those x
|| which are a multiple of 2.

```
print([x for x in (x for x in temp) if (x in ([x for x in range(10)]
)))]
```

Output: []
|| This is becz x has not been converted to int, the condi^n in if
|| statement fails and the list remains empty.

Q. what is shallow and deep copy?

Ans

```
l1 = [1,2,3]
l2 = l1            || shallow
l3 = l1.copy()     || deep
l4 = list(l1)      || deep
```

changes in l1 will reflect in shallow copies & vice versa.

Note An empty tuple has 48 Bytes as overhead size and each additional element requires 8 bytes.

```
t1 = tuple()       || size = 48
t2 = (1, 2)        || size = 48 + 2(8)
```

18/6/18

```
t3 = (1,3,(4,5))   // size = 48 + 3(8)
t4 = (1, 2,3, 4, 5, [3,4], 'p', '8', 9.777, (1,3))   // size = 48 + 10(08)
```

e.g.
```
T1 = (1)
T2 = (3,4)
T1 + =5
print (T1)
print (T1 + T2)
```

Ans 6 Type Error   // T1 is int while T2 is tuple and can't be added.

e.g.
```
List = [True, 50, 10]
List.insert (2,5)
print (list)
print (sum(list))
```

Ans
```
[True, 50, 5, 10]
66   // bcoz boolean also has int value 1
```

e.g.
```
List = [1,2,3]
print List [-1]
```

Ans 3

e.g.
```
L = [1, 3, 5, 7, 9]
print (L.pop(-3), end = ' ')
print (L.remove (L[0] ), end = ' ')
print (L)
```

Ans 5 None [3,7,9]

Note
```
List = [1,2,3,4]
List.pop
```

Output : 4   // Last element is popped.

Note randrange(0, 100, 2) // range(0, 100) with stepsize = 2

Note fabs() returns the modulus of the number.

E.g.
```
p = re.compile('\d+')
print(p.findall('I met 11 AM on 4th Jun 1886'), end = " ")
print(re.compile('\d').findall('I went at 11')
```

Output: ['11', '4', '1886'] ['1', '1']
// bcoz '\d' is equivalent to [0-9] and '\d+' will group on [0-9]
// and match a group of one or greater size.

E.g.
```
print(re.sub('ge', '**', 'Geeks forgeeks', flags = re.IGNORECASE))
print(re.sub('ge', '**', 'Geeks forgeeks')
```

Ans
```
** eksfor ** eks
Geeks for **eks
```

Q Give e.g. of random-choices() available in Python 3.6.1 only.

Ans
```
string = random-choices(['apple', 'carrot', 'grape'], [0.4, 1, 8], k=1)
                                    ↓                      ↓        ↓
                             choices available       weights of   choices
                                                      choices    to be made.
print string.
```

Output: apple
OR Carrot
OR grape // most probable o/p bcoz of max weight.

*

eg.
```
D = dict ()
for x in enumerate (range(2)):
    D[x[0]] = x[1]
    D [x[1]+7] = x[0]
print (D)
```

Ans {0:0, 7:0, 1:1, 8:1}

// enumerate returns a tuple, the loop will have x = (0,0), (1,1).
So, D[0] = 0, D[1] = 1, D[0+7] = D[7] = 0, D[1+7] = D[8] = 1

Q.
```
set1 = {1, 2, 3}
set2 = set1.add (4)
print (set2)
```

Ans None // bcoz add method doesn't return anything

// set1 = {1,2,3,4}

Q.
```
set 1 = {1, 2}
set 2 = {3, 4}
set 3 = set1 + set 2
```

Ans Error // unsupported operand type for + : 'set' and 'set'.

Q.
```
for i in range ("Geeks")
    print i
```

Ans Error // range(str) not allowed.

eg. T = tuple('geeks')

```
a,b,c,d,e = T          // unpacked into a,b,c,d,e
b = c = 'x'
T = (a,b,c,d,e)        // packed again
print(T)
```

Ans ('g','x','x','k','s')

eg: int(n) throws Error when n = 'A' or character bcoz of type conversion.

eg: str1 = '{0:.4f} {0:3d} {2} {1}'.format(2, '3.77', -6)

Ans 2.0000 2 -6 3.77   // 3d converts to decimal i.e formatted // into an integer

Q. chr(ord(i)+3)   // converts i to integer, adds 3, converts // back to character.

Q.

```
line = "what will have so will"
L = line.split('a')        // creates partition at a.
for i in L:
    print(i, end=' ')
```

Ans wh t will h ve so will

Note tuple.append((5,6,7))   // ERROR bcz tuples are immutable

Q. print((1,2,3,4) < (2,2,5,4))   // compares element by element

Ans False

tuple = (1,2,3)

Q print (2*tuple)

Ans (1,2,3,1,2,3)   // '*' operator is used to concatenate tuples

tuple = ("check") *3

Q print (tuple)

Ans (CheckCheckCheck)   // ("Check" is treated as a string not as tuple as there is no comma after the element")

Q. print (list (filter (bool, mylist)))   // list = [0, 5, 2, 'gfg', '', []]

Ans [5, 2, 'gfg'] // returns only those elements of list which are ≠ 0.

Q How can you iterate over a list and also retrieve element indices at the same time?

Ans by using enumerate.