Top 25 Python Questions:

1) **What is Python? What are the benefits of using Python?**
Python is a programming language with objects, modules, threads, exceptions and automatic
memory management. The benefits of pythons are that it is simple and easy, portable, extensible, build-in data structure and it is an open source.

2) **What is PEP 8?**
PEP 8 is a coding convention, a set of recommendation, about how to write your Python code more readable.

3) **What is pickling and unpickling?**
Pickle module accepts any Python object and converts it into a string representation and dumps
it into a file by using dump function, this process is called pickling. While the process of retrieving original Python objects from the stored string representation is called unpickling.

4) **How Python is interpreted?**
Python language is an interpreted language. Python program runs directly from the source code. It converts the source code that is written by the programmer into an intermediate language, which is again translated into machine language that has to be executed.

5) **How memory is managed in Python?**
Python memory is managed by Python private heap space. All Python objects and data structures are located in a private heap. The programmer does not have an access to this private heap and interpreter takes care of this Python private heap.
The allocation of Python heap space for Python objects is done by Python memory manager. The core API gives access to some tools for the programmer to code.
Python also have an inbuilt garbage collector, which recycle all the unused memory and frees the memory and makes it available to the heap space.

6) **What are the tools that help to find bugs or perform static analysis?**
PyChecker is a static analysis tool that detects the bugs in Python source code and warns about the style and complexity of the bug. Pylint is another tool that verifies whether the module
meets the coding standard.

7) **What are Python decorators?**
A Python decorator is a specific change that we make in Python syntax to alter functions easily.

8) **What is the difference between list and tuple?**
The difference between list and tuple is that list is mutable while tuple is not. Tuple can be hashed for e.g as a key for dictionaries.

9) **How are arguments passed by value or by reference?**
Everything in Python is an object and all variables hold references to the objects. The references values are according to the functions; as a result you cannot change the value of the
references. However, you can change the objects if it is mutable.

10) **What is Dict and List comprehensions are?**
They are syntax constructions to ease the creation of a Dictionary or List based on existing

iterable.

11) **What are the built-in type does python provides?**
There are mutable and Immutable types of Pythons built in types Mutable built-in types
List
Sets
Dictionaries

Immutable built-in types
Strings
Tuples
Numbers

12) **What is namespace in Python?**
In Python, every name introduced has a place where it lives and can be hooked for. This is known as namespace. It is like a box where a variable name is mapped to the object placed. Whenever the variable is searched out, this box will be searched, to get corresponding object.

13) **What is lambda in Python?**
It is a single expression anonymous function often used as inline function.

14) **Why lambda forms in python does not have statements?**
A lambda form in python does not have statements as it is used to make new function object and then return them at runtime.

15) **What is pass in Python?**
Pass means, no-operation Python statement, or in other words it is a place holder in compound
statement, where there should be a blank left and nothing has to be written there.

16) **In Python what are iterators?**
In Python, iterators are used to iterate a group of elements, containers like list.

17) **What is unittest in Python?**
A unit testing framework in Python is known as unittest. It supports sharing of setups, automation testing, shutdown code for tests, aggregation of tests into collections etc.

18) **In Python what is slicing?**
A mechanism to select a range of items from sequence types like list, tuple, strings etc. is known as slicing.

19) **What are generators in Python?**
The way of implementing iterators are known as generators. It is a normal function except that
it yields expression in the function.

20) **What is docstring in Python?**
A Python documentation string is known as docstring, it is a way of documenting Python functions, modules and classes.

21) **How can you copy an object in Python?**
To copy an object in Python, you can try copy.copy () or copy.deepcopy() for the general case.

You cannot copy all objects but most of them.

22) **What is negative index in Python?**
Python sequences can be index in positive and negative numbers. For positive index, 0 is the first index, 1 is the second index and so forth. For negative index, (-1) is the last index and (-2)
is the second last index and so forth.

23) **How you can convert a number to a string?**
In order to convert a number into a string, use the inbuilt function str(). If you want a octal or hexadecimal representation, use the inbuilt function oct() or hex().

24) **What is the difference between Xrange and range?**
Xrange returns the xrange object while range returns the list, and uses the same memory and
no matter what the range size is.

25) **What is module and package in Python?**
In Python, module is the way to structure program. Each Python program file is a module, which
imports other modules like objects and attributes.
The folder of Python program is a package of modules. A package can have modules or subfolders.

FAQ
1. **What is Python? State some programming language features of Python**.
Python is a modern powerful interpreted language with objects, modules, threads, exceptions, and automatic memory managements.
Python was introduced to the world in the year 1991 by Guido van Rossum
Salient features of Python are
-Simple & Easy: Python is simple language & easy to learn.
-Free/open source: it means everybody can use python without purchasing license.
-High level language: when coding in Python one need not worry about low-level details.
-Portable: Python codes are Machine & platform independent.
-Extensible: Python program supports usage of C/ C++ codes.
-Embeddable Language: Python code can be embedded within C/C++ codes & can be used a scripting language.
-Standard Library: Python standard library contains prewritten tools for programming.
-Build-in Data Structure: contains lots of data structure like lists, numbers & dictionaries.

2. **What are the rules for local and global variables in Python?**
If a variable is defined outside function then it is implicitly global. If variable is assigned new value inside the function means it is local. If we want to make it global we need to explicitly define it as global. Variable referenced inside the function are implicit global. Following code snippet will explain further the difference

```
#!/usr/bin/python
# Filename: variable_localglobal.py
def fun1(a):
        print 'a:', a
        a= 33;
        print 'local a: ', a
```

```
a = 100
fun1(a)
print 'a outside fun1:', a

def fun2():
        global b
        print 'b: ', b
        b = 33
        print 'global b:', b
b =100
fun2()
print 'b outside fun2', b
```

Output

```
$ python variable_localglobal.py
a: 100
local a: 33
a outside fun1: 100
b :100
global b: 33
b outside fun2: 33
```

3. **How do we share global variables across modules in Python?**

We can create a config file & store the entire global variable to be shared across modules or script in it. By simply importing config, the entire global variable defined it will be available for use in other modules.

For example I want a, b & c to share between modules.

config.py :

```
a=0
b=0
c=0
```

module1.py:

```
import config
config.a = 1
config.b =2
config.c=3
print " a, b & resp. are : " , config.a, config.b, config.c
```

---

output of module1.py will be

```
1 2 3
```

4. **How is memory managed in python?**

Memory management in Python involves a private heap containing all Python objects and data structures. Interpreter takes care of Python heap and that the programmer has no access to it.

The allocation of heap space for Python objects is done by Python memory manager. The core API of Python provides some tools for the programmer to code reliable and more robust program.

Python also has a build-in garbage collector which recycles all the unused memory. When an object is no longer referenced by the program, the heap space it occupies can be freed. The garbage collector determines objects which are no longer referenced by the program frees the occupied memory and make it available to the heap space.
The gc module defines functions to enable /disable garbage collector:
gc.enable() -Enables automatic garbage collection.
gc.disable() - Disables automatic garbage collection.

5. **Describe how to generate random numbers in Python.**
The standard module random implements a random number generator.
There are also many other in this module, such as:
uniform(a, b) returns a floating point number in the range [a, b].
randint(a, b)returns a random integer number in the range [a, b].
random()returns a floating point number in the range [0, 1].

Following code snippet show usage of all the three functions of module random:
Note: output of this code will be different every time it is executed.

```
import random
i = random.randint(1,99)# i randomly initialized by integer between range 1 & 99
j= random.uniform(1,999)# j randomly initialized by float between range 1 & 999
k= random.random()# k randomly initialized by float between range 0 & 1
print("i :" ,i)
print("j :" ,j)
print("k :" ,k)
```

_____
Output -
('i :', 64)
('j :', 701.85008797642115)
('k :', 0.18173593240301023)

Output-
('i :', 83)
('j :', 56.817584548210945)
('k :', 0.9946957743038618)

6. **Describe how exceptions are handled in python.**
Errors detected during execution of program are called exceptions. Exceptions can be handled using the try..except statement. We basically put our usual statements within the try-block and put all our error handlers in the except-block.
try…except demo code:

```
>>> while True:
try:
        x = int(raw_input("Enter no. of your choice: "))
        break
except ValueError:
        print "Oops! Not a valid number. Attempt again"
```

Enter no. of your choice: 12ww
Oops! Not a valid number. Attempt again
Enter no. of your choice: hi there
Oops! Not a valid number. Attempt again
Enter no. of your choice: 22

>>>

7. **When to use list vs. tuple vs. dictionary vs. set?**
List is like array, it can be used to store homogeneous as well as heterogeneous data type (It can store same data type as well as different data type). List are faster compared to array. Individual element of List data can be accessed using indexing & can be manipulated.

List Code Snippet:
```
list = ["Sarah",29,30000.00]
for i in range (3):
    print list[i]
```

Output
Sarah
29
30000.0

Tuples are similar to lists, but there data can be changed once created through the execution of program. Individual element of Tuples can be accessed using indexing.

Tuples Code Snippet: The Days

```
days = ("Sunday", "Monday", "Tuesday", "Wednesday", "Thursday", "Friday", "Saturday")
print days
```

('Sunday', 'Mondays', 'Tuesday', 'Wednesday', 'Thursday', 'Friday', 'Saturday')

Sets stores unordered values & have no index. And unlike Tuples and Lists, Sets can have no duplicate data, It is similar to Mathematical sets.

add() function can be used to add element to a set.
update() function can be used to add a group of elements to a set.
Copy() function can be used to create clone of set.
Set Code Snippet:
```
disneyLand = set (['Minnie Mouse', 'Donald Duck', 'Daisy Duck', 'Goofy'])
disneyLand.add('Pluto')
print disneyLand
```

Output
set(['Goofy', 'Daisy Duck', 'Donald Duck', 'Minnie Mouse', ’Pluto’])

Dictionary are similar to what their name is. In a dictionary, In python, the word is called a 'key', and the definition a 'value'. Dictionaries consist of pairs of keys and their corresponding values.

Dictionary Code Snippet:
```
>>> dict = {'India': 'Bharat', 'Angel': ‘Mother Teresa’, 'Cartoon': 'Mickey'}
>>>print dict[India]
Bharat
>>>print dict[Angel]
Mother Teresa
```

8. **Explain the disadvantages of python.**
Disadvantages of Python are:

Python isn't the best for memory intensive tasks.
Python is interpreted language & is slow compared to C/C++ or java.
Python not a great choice for a high-graphic 3d game that takes up a lot of CPU.
Python is evolving continuously, with constant evolution there is little substantial documentation available for the language.

This set of Python Questions & Answers focuses on “Core Data Types”.

1**. Which of these in not a core datatype?**
a) Lists
b) Dictionary
c) Tuples
d) Class

Answer:d
Explanation:Class is a user defined datatype.

2. **Given a function that does not return any value, What value is thrown by it by default when executed in shell.**
a) int
b) bool
c) void
d) None

Answer:d
Explanation:Python shell throws a NoneType object back.

3. **Following set of commands are executed in shell, what will be the output?**
>>>str="hello"
>>>str[:2]
>>>str
a) he
b) lo
c) olleh
d) hello

Answer:d
Explanation: Strings are immutable and hence the answer is d.

4. **Which of the following will run without errors(multiple answers possible) ?**
a) round(45.8)
b) round(6352.898,2)
c) round()
d) round(7463.123,2,1)

Answer:a,b
Explanation:Execute help(round) in the shell to get details of the parameters that are passed into the round function.

5. **What is the return type of function id ?**
a) int
b) float
c) bool
d) dict

Answer:a
Explanation:Execute help(id) to find out details in python shell.id returns a integer value that is unique.

6. **In python we do not specify types,it is directly interpreted by the compiler, so consider the following operation to be performed.**
>>>x = 13 ? 2
objective is to make sure x has a integer value, select all that apply (python 3.xx)
a) x = 13 // 2
b) x = int(13 / 2)
c) x = 13 / 2
d) x = 13 % 2

Answer:a,b
Explanation:// is integer operation in python 3.0 and int(..) is a type cast operator.

7. **What error occurs when you execute?**
apple = mango
a) SyntaxError
b) NameError
c) ValueError
d) TypeError

Answer:b
Explanation:Banana is not defined hence name error.

8**. Carefully observe the code and give the answer.**
```
def example(a):
    a = a + '2'
     a = a*2
    return a
```
>>>example("hello")
a) indentation Error
b) cannot perform mathematical operation on strings
c) hello2
d) hello2hello2
View Answer

Answer:a
Explanation:Python codes have to be indented properly.

9. **What datatype is the object below ?**
L = [1, 23, 'hello', 1]
a) List
b) dictionary
c) array
d) tuple

Answer:a
Explanation:List datatype can store any values within it.

10**. In order to store values in terms of key and value we use what core datatype.**
a) List
b) tuple
c) class

d) dictionary

Answer:d
Explanation:Dictionary stores values in terms of keys and values.

11. **Which of the following results in a SyntaxError(Multiple answers possible) ?**
a) ‘”Once upon a time…”, she said.’
b) “He said, “Yes!”"
c) ’3\’
d) ”’That’s okay”’

Answer:b,c
Explanation:Carefully look at the colons.

12. **The following is displayed by a print function call:**
tom
dick
harry
Select all of the function calls that result in this output
a) print(”’tom
\ndick
\nharry”’)
b) print(”’tom
dick
harry”’)
c) print(‘tom\ndick\nharry’)
d) print(‘tom
dick
harry’)

Answer:b,c
Explanation:The \n adds a new line.

13. **What is the average value of the code that is executed below ?**
>>>grade1 = 80
>>>grade2 = 90
>>>average = (grade1 + grade2) / 2
a) 85
b) 85.0
c) 95
d) 95.0

Answer:b
Explanation:Cause a decimal value to appear as output.

14. **Select all options that print**
hello-how-are-you
a) print(‘hello’, ‘how’, ‘are’, ‘you’)
b) print(‘hello’, ‘how’, ‘are’, ‘you’ + ‘-’ * 4)
c) print(‘hello-’ + ‘how-are-you’)
d) print(‘hello’ + ‘-’ + ‘how’ + ‘-’ + ‘are’ + ‘-’ + ‘you’)

Answer:c,d
Explanation:Execute in the shell.

15. **What is the return value of trunc() ?**
a) int
b) bool
c) float
d) None

Answer:a
Explanation:Executle help(math.trunc) to get details.

This set of Python Questions & Answers focuses on “Strings”.

1. **What is the output when following statement is executed ?**
>>>"a"+"bc"
a) a
b) bc
c) bca
d) abc

Answer:d
Explanation:+ operator is concatenation operator.

2. **What is the output when following statement is executed ?**

>>>"abcd"[2:]
a) a
b) ab
c) cd
d) dc

Answer:c
Explanation:Slice operation is performed on string.

3. **The output of executing string.ascii_letters can also be achieved by:**
a) string.ascii_lowercase_string.digits
b) string.ascii_lowercase+string.ascii_upercase
c) string.letters
d) string.lowercase_string.upercase

Answer:b
Explanation:Execute in shell and check.

4. **What is the output when following code is executed ?**
>>> str1 = 'hello'
>>> str2 = ','
>>> str3 = 'world'
>>> str1[-1:]
a) olleh
b) hello
c) h
d) o

Answer:d
Explanation:-1 corresponds to the last index.

5. **What arithmetic operators cannot be used with strings?**
a) +
b) *
c) -
d) **

Answer:c,d
Explanation:+ is used to concatenate and * is used to multiply strings.

6. **What is the output when following code is executed ?**
>>>print r"\nhello"
The output is
a) a new line and hello
b) \nhello
c) the letter r and then hello
d) Error

Answer:b
Explanation:When prefixed with the letter ‘r’ or ‘R’ a string literal becomes a raw string and the escape sequences such as \n are not converted.

7. **What is the output when following statement is executed ?**
>>>print 'new' 'line'
a) Error
b) Output equivalent to print ‘new\nline’
c) newline
d) new line

Answer:c
Explanation:String literals seperated by white space are allowed. They are concatenated.

8. **What is the output when following statement is executed ?**
>>>print '\x97\x98'
a) Error
b) 97
98
c) _~
d) \x97\x98

Answer:c
Explanation:\x is an escape sequence that means the following 2 digits are a hexadicmal number encoding a character.

9. **What is the output when following code is executed ?**
>>>str1="helloworld"
>>>str1[::-1]
a) dlrowolleh
b) hello
c) world
d) helloworld

Answer:a
Explanation:Execute in shell to verify.

10. **print 0xA + 0xB + 0xC :**

a) 0xA0xB0xC
b) Error
c) 0×22
d) 33
Answer:d
Explanation:0xA and 0xB and 0xC are hexadecimal integer literals representing the decimal values 10,11 and 12 respectively. There sum is 33.

This set of Python Questions & Answers focuses on “Strings”.
1. **What is the output of the following code ?**

```
class father:
    def __init__(self, param):
        self.o1 = param

class child(father):
    def __init__(self, param):
        self.o2 = param

>>>obj = child(22)
>>>print "%d %d" % (obj.o1, obj.o2)
```

a) None None
b) None 22
c) 22 None
d) Error is generated

Answer:d
Explanation:self.o1 was never created.

2. **What is the output of the following code ?**

```
class tester:
    def __init__(self, id):
        self.id = str(id)
        id="224"

>>>temp = tester(12)
>>>print temp.id
```

a) 224
b) Error
c) 12
d) None

Answer:c
Explanation:id in this case will be the attribute of the class.

3. **What is the output of the following code ?**

```
>>>example = "snow world"
>>>print "%s" % example[4:7]
```

a) wo
b) world
c) sn
d) rl

Answer:a
Explanation:Execute in the shell and verify.

4. **What is the output of the following code ?**
>>>example = "snow world"
>>>example[3] = 's'
>>>print example
a) snow
b) snow world
c) Error
d) snos world

Answer:c
Explanation:Strings cannot be modified.

5. **What is the output of the following code ?**
>>>max("what are you")
a) Error
b) u
c) t
d) y

Answer:d
Explanation:Max returns the character with the highest ascii value.

6**. Given a string example=”hello” what is the output of example.count(l)**
a) 2
b) 1
c) None
d) 0

Answer:a
Explanation:l occurs twice in hello.

7. **What is the output of the following code ?**
>>>example = "helle"
>>>example.find("e")
a) Error
b) -1
c) 1
d) 0
Answer:c
Explanation:returns lowest index .

8. **What is the output of the following code ?**
>>>example = "helle"
>>>example.rfind("e")
a) -1
b) 4
c) 3
d) 1
Answer:b
Explanation:returns highest index.

9. **What is the output of the following code ?**
>>>example="helloworld"
>>>example[::-1].startswith("d")
a) dlrowolleh

b) True
c) -1
d) None
Answer:b
Explanation:Starts with checks if the given string starts with the parameter that is passed.

10. **To concatenate two strings to a third what statements are applicable (multiple answers are allowed) ?**
a) s3 = s1 + s2
b) s3 = s1.add(s2)
c) s3 = s1.__add__(s2)
d) s3 = s1 * s2
Answer:a,c
Explanation:__add__ is another method that can be used for concatenation.

This set of Python Questions & Answers focuses on “Strings”.
1. **What is the output when following statement is executed ?**
>>>chr(ord('A'))
a) A
b) B
c) a
d) Error
Answer:a
Explanation:Execute in shell to verify.

2. **What is the output when following statement is executed ?**
>>>print(chr(ord('b')+1))
a) a
b) b
c) c
d) A
View Answer

Answer:c
Explanation:Execute in the shell to verify.

3. **Which of the following statement prints hello\example\test.txt ?**
a) print(“hello\example\test.txt”)
b) print(“hello\\example\\test.txt”)
c) print(“hello\”example\”test.txt”)
d) print(“hello”\example”\test.txt”)

Answer:b
Explanation:\is used to indicate that the next \ is not an escape sequence.

4. **Suppose s is “\t\tWorld\n”, what is s.strip() ?**
a) \t\tWorld\n
b) \t\tWorld\n
c) \t\tWORLD\n
d) World

Answer:d
Explanation:Execute help(string.strip) to find details.

5. **The format function returns :**

a) Error
b) int
c) bool
d) str

Answer:d
Explanation:Format function returns a string.

6**. What is the output of “hello”+1+2+3 ?**
a) hello123
b) hello
c) Error
d) hello6

Answer:c
Explanation:Cannot concantenate str and int objects.

7. **What is the output when following code is executed ?**
>>>print("D", end = ' ')
>>>print("C", end = ' ')
>>>print("B", end = ' ')
>>>print("A", end = ' ')
a) DCBA
b) A, B, C, D
c) D C B A
d) A, B, C, D will be displayed on four lines

Answer:c
Explanation:Execute in the shell.

8. **What is the output when following statement is executed ?(python 3.xx)**
>>>print(format("Welcome", "10s"), end = '#')
>>>print(format(111, "4d"), end = '#')
>>>print(format(924.656, "3.2f"))
a) Welcome# 111#924.66
b) Welcome#111#924.66
c) Welcome#111#.66
d) Welcome # 111#924.66

Answer:d
Explanation:Execute in the shell to verify.

9. **What will be displayed by print(ord(‘b’) – ord(‘a’)) ?**
a) 0
b) 1
c) -1
d) 2

Answer:b
Explanation:ascii value of b is one more than a.

10. **Say s=”hello” what will be the return value of type(s) ?**
a) int
b) bool
c) str

d) String

Answer:c
Explanation:str is used to represent strings in python.

This set of Python Questions & Answers focuses on “Strings”.
1. What is “Hello”.replace(“l”, “e”)
a) Heeeo
b) Heelo
c) Heleo
d) None
View Answer

Answer:a
Explanation:Execute in shell to verify.
2. **To retrieve the character at index 3 from string s=”Hello” what command do we execute (multiple answers allowed) ?**
a) s[3]
b) s.getitem(3)
c) s.__getitem__(3)
d) s.getItem(3)

Answer:a, c
Explanation:__getitem(..) can be used to get character at index specified as parameter.

3. **To return the length of string s what command do we execute (multiple answers allowed) ?**
a) s.__len__()
b) len(s)
c) size(s)
d) s.size()

Answer:a,b
Explanation:Execute in shell to verify.

4**. If a class defines the __str__(self) method, for an object obj for the class, you can use which command to invoke the __str__ method.(multiple answers allowed)**
a) obj.__str__()
b) str(obj)
c) print obj
d) __str__(obj)

Answer:a,b,c
Explanation:Execute in shell to verify.

5. **To check whether string s1 contains s2, use**
a) s1.__contains__(s2)
b) s1 in s2
c) s1.contains(s2)
d) si.in(s2)

Answer:a,b
Explanation:s1 in s2 works in the same way as calling the special function __contains__ .

6. **Suppose i is 5 and j is 4, i + j is same as**

a) i.__add(j)
b) i.__add__(j)
c) i.__Add(j)
d) i.__ADD(j)

Answer:b
Explanation:Execute in shell to verify.
7. **What is the output of the following code ?**

```
class Count:
    def __init__(self, count = 0):
        self.__count = count

c1 = Count(2)
c2 = Count(2)
print(id(c1) == id(c2), end = " ")

s1 = "Good"
s2 = "Good"
print(id(s1) == id(s2))
```

a) True False
b) True True
c) False True
d) False False

Answer:c
Explanation:Execute in the shell objects cannot have same id, however in the case of strings its different.

8. **What is the output of the following code ?**

```
class Name:
    def __init__(self, firstName, mi, lastName):
        self.firstName = firstName
        self.mi = mi
        self.lastName = lastName

firstName = "John"
name = Name(firstName, 'F', "Smith")
firstName = "Peter"
name.lastName = "Pan"
print(name.firstName, name.lastName)
```

a) Peter Pan.
b) John Pan.
c) Peter Smith.
d) John Smith.
Answer:b
Explanation:Execute in the shell to verify.

9. **What function do you use to read a string?**
a) input(“Enter a string”)
b) eval(input(“Enter a string”))
c) enter(“Enter a string”)
d) eval(enter(“Enter a string”))

Answer:a
Explanation:Execute in shell to verify.

10. **Suppose x is 345.3546, what is format(x, “10.3f”) (_ indicates space)**
a) __345.355
b) ___345.355
c) ____345.355
d) _____345.354
Answer:b
Explanation:Execute in the shell to verify.

This set of Python Questions & Answers focuses on “Lists”.
1. **Which of the following commands will create a list(multiple answers allowed) ?**
a) list1 = list()
b) list1 = []
c) list1 = list([1, 2, 3])
d) list1 = [1, 2, 3]

Answer:a,b,c,d
Explanation:Execute in the shell to verify

2. **What is the output when we execute list(“hello”)?**
a) ['h', 'e', 'l', 'l', 'o']
b) ['hello']
c) ['llo']
d) ['olleh']

Answer:a
Explanation:execute in the shell to verify.

3. **Suppose listExample is ['h','e','l','l','o'], what is len(listExample)?**
a) 5
b) 4
c) None
d) Error

Answer:a
Explanation:Execute in the shell and verify.

4. **Suppose list1 is [2445,133,12454,123], what is max(list1) ?**
a) 2445
b) 133
c) 12454
d) 123

Answer:c
Explanation:max returns the maximum element in the list.
5. **Suppose list1 is [3, 5, 25, 1, 3], what is min(list1) ?**
a) 3
b) 5
c) 25
d) 1

Answer:d
Explanation:min returns the minimum element in the list.

6. **Suppose list1 is [1, 5, 9], what is sum(list1) ?**

a) 1
b) 9
c) 15
d) Error

Answer:c
Explanation:Sum returns the sum of all elements in the list.

7. **To shuffle the list(say list1) what function do we use ?**
a) list1.shuffle()
b) shuffle(list1)
c) random.shuffle(list1)
d) random.shuffleList(list1)

Answer:c
Explanation:Execute in the shell to verify .

8. **Suppose list1 is [4, 2, 2, 4, 5, 2, 1, 0], Which of the following is correct (multiple answers allowed) ?**
a) print(list1[0])
b) print(list1[:2])
c) print(list1[:-2])
d) print(list1[4:6])

Answer:a, b, c, d
Explanation:Slicing is allowed in lists just as in the case of strings.

9. **Suppose list1 is [2, 33, 222, 14, 25], What is list1[-1] ?**
a) Error
b) None
c) 25
d) 2

Answer:c
Explanation:-1 corresponds to the last index in the list.

10. **Suppose list1 is [2, 33, 222, 14, 25], What is list1[:-1] ?**
a) [2, 33, 222, 14]
b) Error
c) 25
d) [25, 14, 222, 33, 2]
Answer:a
Explanation:Execute in the shell to verify.

This set of Python Questions & Answers focuses on “Lists”.
1. **What is the output when following code is executed ?**
>>>names = ['Amir', 'Bear', 'Charlton', 'Daman']
>>>print names[-1][-1]
a) A
b) Daman
c) Error
d) n

Answer:d

Explanation:Execute in the shell to verify.

2. **What is the output when following code is executed ?**

```
names1 = ['Amir', 'Bear', 'Charlton', 'Daman']
names2 = names1
names3 = names1[:]

names2[0] = 'Alice'
names3[1] = 'Bob'

sum = 0
for ls in (names1, names2, names3):
    if ls[0] == 'Alice':
        sum += 1
    if ls[1] == 'Bob':
        sum += 10

print sum
```

a) 11
b) 12
c) 21
d) 22
View Answer

Answer:b
Explanation:When assigning names1 to names2, we create a second reference to the same list. Changes to names2 affect names1. When assigning the slice of all elements in names1 to names3, we are creating a full copy of names1 which can be modified independently.

3**. Suppose list1 is [1, 3, 2], What is list1 * 2 ?**
a) [2, 6, 4]
b) [1, 3, 2, 1, 3]
c) [1, 3, 2, 1, 3, 2]
D) [1, 3, 2, 3, 2, 1]

Answer:c
Explanation:Execute in the shell and verify.

4. **Suppose list1 = [0.5 * x for x in range(0, 4)], list1 is :**
a) [0, 1, 2, 3]
b) [0, 1, 2, 3, 4]
c) [0.0, 0.5, 1.0, 1.5]
d) [0.0, 0.5, 1.0, 1.5, 2.0]

Answer:c
Explanation:Execute in the shell to verify.

5. **What is the output when following code is executed ?**

```
>>>list1 = [11, 2, 23]
>>>list2 = [11, 2, 2]
>>>list1 < list2 is
```

a) True
b) False
c) Error
d) None

Answer:b
Explanation:Elements are compared one by one.

6. **To add a new element to a list we use which command ?**
a) list1.add(5)
b) list1.append(5)
c) list1.addLast(5)
d) list1.addEnd(5)
Answer:b
Explanation:We use the function append to add an element to the list.

7. **To insert 5 to the third position in list1, we use which command ?**
a) list1.insert(3, 5)
b) list1.insert(2, 5)
c) list1.add(3, 5)
d) list1.append(3, 5)

Answer:a
Explanation:Execute in the shell to verify.
8. **To remove string “hello” from list1, we use which command ?**
a) list1.remove(“hello”)
b) list1.remove(hello)
c) list1.removeAll(“hello”)
d) list1.removeOne(“hello”)

Answer:a
Explanation:Execute in the shell to verify.

9. **Suppose list1 is [3, 4, 5, 20, 5], what is list1.index(5) ?**
a) 0
b) 1
c) 4
d) 2
Answer:d
Explanation:Execute help(list.index) to get details.

10. **Suppose list1 is [3, 4, 5, 20, 5, 25, 1, 3], what is list1.count(5) ?**
a) 0
b) 4
c) 1
d) 2
Answer:d
Explanation:Execute in the shell to verify.

This set of Python Questions & Answers focuses on “Lists”.
1. **Suppose list1 is [3, 4, 5, 20, 5, 25, 1, 3], what is list1 after list1.reverse() ?**
a) [3, 4, 5, 20, 5, 25, 1, 3]
b) [1, 3, 3, 4, 5, 5, 20, 25]
c) [25, 20, 5, 5, 4, 3, 3, 1]
d) [3, 1, 25, 5, 20, 5, 4, 3]

Answer:d
Explanation:Execute in the shell to verify.

2. **Suppose listExample is [3, 4, 5, 20, 5, 25, 1, 3], what is list1 after listExample.extend([34, 5]) ?**
a) [3, 4, 5, 20, 5, 25, 1, 3, 34, 5]
b) [1, 3, 3, 4, 5, 5, 20, 25, 34, 5]
c) [25, 20, 5, 5, 4, 3, 3, 1, 34, 5]
d) [1, 3, 4, 5, 20, 5, 25, 3, 34, 5]

Answer:a
Explanation:Execute in the shell to verify.

3. **Suppose listExample is [3, 4, 5, 20, 5, 25, 1, 3], what is list1 after listExample.pop(1) ?**
a) [3, 4, 5, 20, 5, 25, 1, 3]
b) [1, 3, 3, 4, 5, 5, 20, 25]
c) [3, 5, 20, 5, 25, 1, 3]
d) [1, 3, 4, 5, 20, 5, 25]

Answer:c
Explanation:pop() removes the element at the position specified in the parameter.

4. **Suppose listExample is [3, 4, 5, 20, 5, 25, 1, 3], what is list1 after listExample.pop()?**
a) [3, 4, 5, 20, 5, 25, 1]
b) [1, 3, 3, 4, 5, 5, 20, 25]
c) [3, 5, 20, 5, 25, 1, 3]
d) [1, 3, 4, 5, 20, 5, 25]

Answer:a
Explanation:pop() by default will remove the last element.

5. **What is the output when following code is executed ?**
>>>"Welcome to Python".split()
a) ["Welcome", "to", "Python"]
b) (“Welcome”, “to”, “Python”)
c) {“Welcome”, “to”, “Python”}
d) “Welcome”, “to”, “Python”

Answer:a
Explanation:split() function returns the elements in a list.

6. **What is the output when following code is executed ?**
>>>list("a#b#c#d".split('#'))
a) ['a', 'b', 'c', 'd']
b) ['a b c d']
c) ['a#b#c#d']
d) ['abcd']

Answer:a
Explanation:Execute in the shell to verify.

7. **What is the output when following code is executed ?**
myList = [1, 5, 5, 5, 5, 1]
max = myList[0]
indexOfMax = 0
for i in range(1, len(myList)):

```
    if myList[i] > max:
        max = myList[i]
        indexOfMax = i

>>>print(indexOfMax)
```
a) 1
b) 2
c) 3
d) 4

Answer:a
Explanation:First time the highest number is encountered is at index 1.

8**. What is the output when following code is executed ?**
```
myList = [1, 2, 3, 4, 5, 6]
for i in range(1, 6):
    myList[i - 1] = myList[i]

for i in range(0, 6):
    print(myList[i], end = " ")
```
a) 2 3 4 5 6 1
b) 6 1 2 3 4 5
c) 2 3 4 5 6 6
d) 1 1 2 3 4 5

Answer:c
Explanation:Execute in the shell to verify.

9. **What is the output when following code is executed ?**
```
>>>list1 = [1, 3]
>>>list2 = list1
>>>list1[0] = 4
>>>print(list2)
```
a) [1, 3]
b) [4, 3]
c) [1, 4]
d) [1, 3, 4]

Answer:b
Explanation:Lists should be copied by executing [:] operation.

10. **What is the output when following code is executed ?**
```
def f(values):
    values[0] = 44

v = [1, 2, 3]
f(v)
print(v)
```
a) [1, 44]
b) [1, 2, 3, 44]
c) [44, 2, 3]
d) [1, 2, 3]
View Answer

Answer:c

Explanation:Execute in the shell to verify.

This set of Python Questions & Answers focuses on “Lists”.

1. **What will be the output?**

```
def f(i, values = []):
    values.append(i)
    return values

f(1)
f(2)
v = f(3)
print(v)
```

a) [1] [2] [3]
b) [1] [1, 2] [1, 2, 3]
c) [1, 2, 3]
d) 1 2 3
Answer:c
Explanation:execute in the shell to verify

2. **What will be the output?**

```
names1 = ['Amir', 'Bala', 'Chales']
if 'amir' in names1:
    print 1
else:
    print 2
```

a) None
b) 1
c) 2
d) Error
View Answer

Answer:c
Explanation:execute in the shell to verify.

3. **What will be the output?**

```
names1 = ['Amir', 'Bala', 'Charlie']
names2 = [name.lower() for name in names1]

print names2[2][0]
```

a) None
b) a
c) b
d) c

Answer:d
Explanation:List Comprehension are a shorthand for creating new lists.

4. **What will be the output?**

```
numbers = [1, 2, 3, 4]
numbers.append([5,6,7,8])
print len(numbers)
```

a) 4
b) 5
c) 8

d) 12

Answer:b
Explanation:a list is passed in append so the length is 5.

5**. To which of the following the “in” operator can be used to check if an item is in it?**
a) Lists
b) Dictionary
c) Set
d) All of The Above

Answer:d
Explanation:in can be used in all data structures.

6. **What will be the output?**
```
list1 = [1, 2, 3, 4]
list2 = [5, 6, 7, 8]
print len(list1 + list2)
```
a) 2
b) 4
c) 5
d) 8
View Answer

Answer:d
Explanation:+ appends all the elements individually into a new list.

7. **What will be the output?**
```
def addItem(listParam):
    listParam += [1]

mylist = [1, 2, 3, 4]
addItem(mylist)
print len(mylist)
```
a) 1
b) 4
c) 5
d) 8
View Answer

Answer:c
Explanation:+ will append the element to the list.

8. **What will be the output?**
```
def increment_items(L, increment):
    i = 0
    while i < len(L):
        L[i] = L[i] + increment
        i = i + 1

values = [1, 2, 3]
print(increment_items(values, 2))
print(values)
```
a) None

[3, 4, 5]
b) None
[1, 2, 3]
c) [3, 4, 5]
[1, 2, 3]
d) [3, 4, 5]
None

Answer:a
Explanation:Execute in the shell to verify.

9. **What will be the output?**

```
def example(L):
    ''' (list) -> list
    '''
    i = 0
    result = []
    while i < len(L):
        result.append(L[i])
        i = i + 3
    return result
```

a) Return a list containing every third item from L starting at index 0.
b) Return an empty list
c) Return a list containing every third index from L starting at index 0.
d) Return a list containing the items from L starting from index 0, omitting every third item.

Answer:a
Explanation:Run the code to get a better understanding with many arguements.

10. **What will be the output?**

```
veggies = ['carrot', 'broccoli', 'potato', 'asparagus']
veggies.insert(veggies.index('broccoli'), 'celery')
print(veggies)
```

a) ['carrot', 'celery', 'broccoli', 'potato', 'asparagus'] Correct 1.00
b) ['carrot', 'celery', 'potato', 'asparagus']
c) ['carrot', 'broccoli', 'celery', 'potato', 'asparagus']
d) ['celery', 'carrot', 'broccoli', 'potato', 'asparagus']
Answer:a
Explanation:Execute in the shell to verify.

This set of Python Questions & Answers focuses on “Lists”.

1**. What will be the output?**

```
>>>m = [[x, x + 1, x + 2] for x in range(0, 3)]
```

a) [[1, 2, 3], [4, 5, 6], [7, 8, 9]]
b) [[0, 1, 2], [1, 2, 3], [2, 3, 4]]
c) [1, 2, 3, 4, 5, 6, 7, 8, 9]
d) [0, 1, 2, 1, 2, 3, 2, 3, 4]
Answer:b
Explanation:Execute in the shell to verify

2. **How many elements are in m?**

```
m = [[x, y] for x in range(0, 4) for y in range(0, 4)]
```

a) 8
b) 12

c) 16
d) 32

Answer:c
Explanation:Execute in the shell to verify.

3. **What will be the output?**

```
values = [[3, 4, 5, 1], [33, 6, 1, 2]]
v = values[0][0]
for row in range(0, len(values)):
    for column in range(0, len(values[row])):
        if v < values[row][column]:
            v = values[row][column]

print(v)
```

a) 3
b) 5
c) 6
d) 33
Answer:d
Explanation:Execute in the shell to verify.

4**. What will be the output?**

```
values = [[3, 4, 5, 1], [33, 6, 1, 2]]
v = values[0][0]
for lst in values:
    for element in lst:
        if v > element:
            v = element

print(v)
```

a) 1
b) 3
c) 5
d) 6
Answer:a
Explanation:Execute in the shell to verify.

5. **What will be the output?**

```
values = [[3, 4, 5, 1 ], [33, 6, 1, 2]]

for row in values:
    row.sort()
    for element in row:
        print(element, end = " ")
    print()
```

a) The program prints two rows 3 4 5 1 followed by 33 6 1 2
b) The program prints on row 3 4 5 1 33 6 1 2
c) The program prints two rows 3 4 5 1 followed by 33 6 1 2
d) The program prints two rows 1 3 4 5 followed by 1 2 6 33

Answer:d
Explanation:Execute in the shell to verify.

6. **What is the output?**

```
matrix = [[1, 2, 3, 4],
      [4, 5, 6, 7],
      [8, 9, 10, 11],
      [12, 13, 14, 15]]

for i in range(0, 4):
   print(matrix[i][1], end = " ")
```
a) 1 2 3 4
b) 4 5 6 7
c) 1 3 8 12
d) 2 5 9 13

Answer:d
Explanation:Execute in the shell to verify.

7. **What will be the output?**
```
def m(list):
   v = list[0]
   for e in list:
     if v < e: v = e
   return v

values = [[3, 4, 5, 1], [33, 6, 1, 2]]

for row in values:
   print(m(row), end = " ")
```
a) 3 33
b) 1 1
c) 5 6
d) 5 33
Answer:d
Explanation:Execute in the shell to verify.

8. **What will be the output?**
```
data = [[[1, 2], [3, 4]], [[5, 6], [7, 8]]]
print(data[1][0][0])
```
a) 1
b) 2
c) 4
d) 5
Answer:d
Explanation:Execute in the shell to verify.

9. **What will be the output?**
```
data = [[[1, 2], [3, 4]], [[5, 6], [7, 8]]]
def ttt(m):
   v = m[0][0]

   for row in m:
      for element in row:
         if v < element: v = element

   return v

print(ttt(data[0]))
```

a) 1
b) 2
c) 4
d) 5
Answer:c
Explanation:Execute in the shell to verify.

10. **What will be the output?**
points = [[1, 2], [3, 1.5], [0.5, 0.5]]
points.sort()
print(points)
a) [[1, 2], [3, 1.5], [0.5, 0.5]]
b) [[3, 1.5], [1, 2], [0.5, 0.5]]
c) [[0.5, 0.5], [1, 2], [3, 1.5]]
d) [[0.5, 0.5], [3, 1.5], [1, 2]]
Answer:c
Explanation:Execute in the shell to verify.

This set of Python Questions & Answers focuses on “Dictionaries”.

1. **Which of the following statements create a dictionary?(multiple answers allowed)**
a) d = {}
b) d = {“john”:40, “peter”:45}
c) d = {40:”john”, 45:”peter”}
d) d = (40:”john”, 45:”peter”)
Answer:a,b,c
Explanation:Dictionaries are created by specifying keys and values

2. What are the keys?
d = {"john":40, "peter":45}
a) “john”, 40, 45, and “peter”
b) “john” and “peter”
c) 40 and 45
d) d = (40:”john”, 45:”peter”)
Answer:b
Explanation:Dictionaries appear in the form of keys and values.

3. **What will be the output?**
d = {"john":40, "peter":45}
"john" in d
a) True
b) False
c) None
d) Error

Answer:a
Explanation:In can be used to check if the key is int dictionary.

4. **What will be the output?**
d1 = {"john":40, "peter":45}
d2 = {"john":466, "peter":45}
d1 == d2
a) True
b) False

c) None
d) Error

Answer:b
Explanation:If d2 was initialized as d2 = d1 the answer would be true.

5. **What will be the output?**
d1 = {"john":40, "peter":45}
d2 = {"john":466, "peter":45}
d1 > d2
a) True
b) False
c) Error
d) None

Answer:c
Explanation:Arithmetic > operator cannot be used with dictionaries.

6**. What is the output?**
d = {"john":40, "peter":45}
d["john"]
a) 40
b) 45
c) “john”
d) “peter”

Answer:a
Explanation:Execute in the shell to verify.

7. **Suppose d = {“john”:40, “peter”:45}, to delete the entry for “john” what command do we use**
a) d.delete(“john”:40)
b) d.delete(“john”)
c) del d["john"]
d) del d(“john”:40)
View Answer

Answer:c
Explanation:Execute in the shell to verify.

8. **Suppose d = {“john”:40, “peter”:45}, to obtain the number of entries in dictionary what command do we use**
a) d.size()
b) len(d)
c) size(d)
d) d.len()

Answer:b
Explanation:Execute in the shell to verify.

9. **What will be the output?**
d = {"john":40, "peter":45}
print(list(d.keys()))
a) ["john", "peter"]
b) ["john":40, "peter":45]

c) (“john”, “peter”)
d) (“john”:40, “peter”:45)
Answer:a
Explanation:Execute in the shell to verify.

10. **Suppose d = {“john”:40, “peter”:45}, what happens when retieving a value using d["susan"]?**
a) Since “susan” is not a value in the set, Python raises a KeyError exception.
b) It is executed fine and no exception is raised, and it returns None.
c) Since “susan” is not a key in the set, Python raises a KeyError exception.
d) Since “susan” is not a key in the set, Python raises a syntax error.
Answer:c
Explanation:Execute in the shell to verify.

This set of Python Questions & Answers focuses on “tuples”.
1. Which of the following is a Python tuple?
a) [1, 2, 3]
b) (1, 2, 3)
c) {1, 2, 3}
d) {}

Answer:b
Explanation:Tuples are characterised by their round brackets.

2. **Suppose t = (1, 2, 4, 3), which of the following is incorrect?**
a) print(t[3])
b) t[3] = 45
c) print(max(t))
d) print(len(t))
View Answer

Answer:b
Explanation:Values cannot be modified in the case of tuple.

3. **What will be the output?**
>>>t=(1,2,4,3)
>>>t[1:3]
a) (1, 2)
b) (1, 2, 4)
c) (2, 4)
d) (2, 4, 3)

Answer:c
Explanation:Slicing just as in the case of strings takes place in tuples.

4. **What will be the output?**
>>>t=(1,2,4,3)
>>>t[1:-1]
a) (1, 2)
b) (1, 2, 4)
c) (2, 4)
d) (2, 4, 3)

Answer:c
Explanation:Slicing just as in the case of strings takes place in tuples.

5. What will be the output?
>>>t = (1, 2, 4, 3, 8, 9)
>>>[t[i] for i in range(0, len(t), 2)]
a) [2, 3, 9]
b) [1, 2, 4, 3, 8, 9]
c) [1, 4, 8]
d) (1, 4, 8)
Answer:c
Explanation:Execute in the shell to verify.

6. **What will be the output?**
d = {"john":40, "peter":45}
d["john"]
a) 40
b) 45
c) “john”
d) “peter”
Answer:a
Explanation:Execute in the shell to verify.

7. **What will be the output?**
>>>t = (1, 2)
>>>2 * t
a) (1, 2, 1, 2)
b) [1, 2, 1, 2]
c) (1, 1, 2, 2)
d) [1, 1, 2, 2]
Answer:a
Explanation:* operator concatenates tuple.

8**. What will be the output?**
>>>t1 = (1, 2, 4, 3)
>>>t2 = (1, 2, 3, 4)
>>>t1 < t2
a) True
b) False
c) Error
d) None

Answer:b
Explanation:Elements are compared one by one in this case.

9. **What will be the output?**
>>>my_tuple = (1, 2, 3, 4)
>>>my_tuple.append((5, 6, 7))
>>>print len(my_tuple)
a) 1
b) 2
c) 5
d) Error

Answer:d
Explanation:Tuples are immutable and don’t have an append method. An exception is thrown in this case..

10. **What will be the output?**

```
numberGames = {}
numberGames[(1,2,4)] = 8
numberGames[(4,2,1)] = 10
numberGames[(1,2)] = 12

sum = 0
for k in numberGames:
   sum += numberGames[k]

print len(numberGames) + sum
```

a) 30
b) 24
c) 33
d) 12
Answer:c
Explanation:Tuples can be used for keys into dictionary. The tuples can have mixed length and the order of the items in the tuple is considered when comparing the equality of the keys.

This set of Python Questions & Answers focuses on “files”.

1. **To open a file c:\scores.txt for reading, we use**
a) infile = open(“c:\scores.txt”, “r”)
b) infile = open(“c:\\scores.txt”, “r”)
c) infile = open(file = “c:\scores.txt”, “r”)
d) infile = open(file = “c:\\scores.txt”, “r”)

Answer:b
Explanation:Execute help(open) to get more details.

2. **To open a file c:\scores.txt for writing, we use**
a) outfile = open(“c:\scores.txt”, “w”)
b) outfile = open(“c:\\scores.txt”, “w”)
c) outfile = open(file = “c:\scores.txt”, “w”)
d) outfile = open(file = “c:\\scores.txt”, “w”)

Answer:b
Explanation:w is used to indicate that file is to be written to.

3. **To open a file c:\scores.txt for appending data, we use**
a) outfile = open(“c:\\scores.txt”, “a”)
b) outfile = open(“c:\\scores.txt”, “rw”)
c) outfile = open(file = “c:\scores.txt”, “w”)
d) outfile = open(file = “c:\\scores.txt”, “w”)

Answer:a
Explanation:a is used to indicate that data is to be apended.

4. **Which of the following statements are true? (multiple answers allowed)**
a) A. When you open a file for reading, if the file does not exist, an error occurs.
b) When you open a file for reading, if the file does not exist, the program will open an empty file.

c) When you open a file for writing, if the file does not exist, a new file is created.
d) When you open a file for writing, if the file exists, the existing file is overwritten with the new file.
Answer:a,c,d.
Explanation:The program will throw an error.

5. **To read two characters from a file object infile, we use**
a) infile.read(2)
b) infile.read()
c) infile.readline()
d) infile.readlines()

Answer:a
Explanation:Execute in the shell to verify.

6. **To read the entire remaining contents of the file as a string from a file object infile, we use**
a) infile.read(2)
b) infile.read()
c) infile.readline()
d) infile.readlines()

Answer:b
Explanation:read function is used to read all the lines in a file.

7. **What is the output?**

```
f = None

for i in range (5):
    with open("data.txt", "w") as f:
        if i > 2:
            break

print f.closed
```

a) True
b) False
c) None
d) Error
Answer:a
Explanation:The WITH statement when used with open file guarantees that the file object is closed when the with block exits..

8. **To read the next line of the file from a file object infile, we use**
a) infile.read(2)
b) infile.read()
c) infile.readline()
d) infile.readlines()
Answer:c
Explanation:Execute in the shell to verify.

9. **To read the remaining lines of the file from a file object infile, we use**
a) infile.read(2)
b) infile.read()
C) infile.readline()

d) infile.readlines()

Answer:d
Explanation:Execute in the shell to verify.

10. **The readlines() method returns**
a) str
b) a list of lines
c) a list of single characters
d) a list of integers
Answer:b
Explanation:Every line is stored in a list and returned.

This section on Python aptitude questions and answers focuses on “Function – 2″.

1. **Which are the advantages of functions in python?**
a) Reducing duplication of code
b) Decomposing complex problems into simpler pieces
c) Improving clarity of the code
d) Reuse of code
e) Information hiding
f) All of the mentioned
Answer: f
Explanation: None.

2. **What are the two types of functions?**
a) Custom function
b) Built-in function
c) User-Defined function
d) System function
Answer: b and c
Explanation: Built-in functions and user defined ones. The built-in functions are part of the Python language. Examples are: dir(), len() or abs(). The user defined functions are functions created with the def keyword.

3. **Where is function defined?**
a) Module
b) Class
c) Another function
d) None of the mentioned

Answer: a, b and c
Explanation: Functions can be defined inside a module, a class or another function.

4. **What is called when a function is defined inside a class?**
a) Module
b) Class
c) Another function
d) Method

Answer: d
Explanation: None.

5. **Which of the following is the use of id() function in python?**

a) Id returns the identity of the object
b) Every object doesn’t have a unique id
c) All of the mentioned
d) None of the mentioned

Answer: a
Explanation: Each object in Python has a unique id. The id() function returns the object’s id.

6. **Which of the following refers to mathematical function?**
a) sqrt
b) rhombus
c) add
d) rhombus

Answer: a
Explanation: Functions that are always available for usage, functions that are contained within external modules, which must be imported and functions defined by a programmer with the def keyword.
Eg: math import sqrt
The sqrt() function is imported from the math module.

7. **What is the output of below program?**

```
def cube(x):
    return x * x * x

x = cube(3)
print x
```

a) 9
b) 3
c) 27
d) 30
View Answer

Answer: c
Explanation: A function is created to do a specific task. Often there is a result from such a task. The return keyword is used to return values from a function. A function may or may not return a value. If a function does not

have a return keyword, it will send a none value.

8. **What is the output of the below program?**

```
def C2F(c):
    return c * 9/5 + 32
print C2F(100)
print C2F(0)
```

a) 212
32
b) 314
24
c) 567
98
d) None of the mentioned

Answer: a
Explanation: None.

9. **What is the output of the below program?**

```
def power(x, y=2):
    r = 1
    for i in range(y):
       r = r * x
    return r
print power(3)
print power(3, 3)
```

a) 212
32
b) 9
27
c) 567
98
d) None of the mentioned
View Answer

Answer: b
Explanation: The arguments in Python functions may have implicit values. An implicit value is used, if no value is provided. Here we created a power function. The function has one argument with an implicit value. We can call
the function with one or two arguments.

10. **What is the output of the below program?**

```
def sum(*args):
   '''Function returns the sum
   of all values'''
   r = 0
   for i in args:
      r += i
   return r
print sum.__doc__
print sum(1, 2, 3)
print sum(1, 2, 3, 4, 5)
```

a) 6
15
b) 6
100
c) 123
12345
d) None of the mentioned
View Answer

Answer: a
Explanation: We use the * operator to indicate, that the function will accept arbitrary number of arguments. The sum() function will return the sum of all arguments. The first string in the function body is called the
function documentation string. It is used to document the function. The string must be in triple quotes.

This section on Python aptitude questions and answers focuses on “Function – 3″.

1. **Python supports the creation of anonymous functions at runtime, using a construct called _____?**
a) Lambda
b) pi
c) anonymous
d) None of the mentioned

Answer: a
Python supports the creation of anonymous functions (i.e. functions that are not bound to a name) at runtime, using a construct called lambda. Lambda functions are restricted to a single expression. They can be used wherever
normal functions can be used.

2. **What is the output of this program?**
y = 6
z = lambda x: x * y
print z(8)
a) 48
b) 14
c) 64
d) None of the mentioned

Answer: a
Explanation: The lambda keyword creates an anonymous function. The x is a parameter, that is passed to the lambda function. The parameter is followed by a colon character. The code next to the colon is the expression that is executed, when the lambda function is called. The lambda function is assigned to the z variable.
The lambda function is executed. The number 8 is passed to the anonymous function and it returns 48 as the result. Note that z is not a name for this function. It is only a variable to which the anonymous function was
assigned.

3. **What is the output of below program?**
lamb = lambda x: x ** 3
print(lamb(5))
a) 15
b) 555
c) 125
d) None of the mentioned

Answer: c
Explanation: None.

4. **Is Lambda contains return statements**
a) True
b) False

Answer: b
Explanation: lambda definition does not include a return statement — it always contains an expression which is returned. Also note that we can put a lambda definition anywhere a function is expected, and we don’t have to
assign it to a variable at all.

5**. Lambda is a statement.**
a) True
b) False

Answer: b
Explanation: None.

6. **Lambda contains block of statements**
a) True
b) False
View Answer

Answer: b
Explanation: None.

7. **What is the output of below program?**
```
def f(x, y, z): return x + y + z
f(2, 30, 400)
```
a) 432
b) 24000
c) 430
d) None of the mentioned

Answer: a
Explanation: None.

8.**What is the output of below program?**
```
def writer():
        title = 'Sir'
        name = (lambda x:title + ' ' + x)
        return name

who = writer()
who('Arthur')
```
a) Arthur Sir
b) Sir Arthur
c) Arthur
d) None of the mentioned

Answer: b
Explanation: None.

9. **What is the output of this program?**
```
L = [lambda x: x ** 2,
      lambda x: x ** 3,
      lambda x: x ** 4]

for f in L:
        print(f(3))
```
a) 27
81
343
b) 6
9

12
c) 9
27
81
d) None of the mentioned

Answer: c
Explanation: None.

10. **What is the output of this program?**
min = (lambda x, y: x if x < y else y)
 min(101*99, 102*98)
a) 9997
b) 9999
c) 9996
d) None of the mentioned

Answer: c
Explanation: None.

Below are top 20 Python interview questions, frequently asked to fresher and amateur Python programmers.

1. **Is python compiled based or interpretive based language?**
Python mostly used in scripting, is general purpose programming language which supports OOP Object oriented programming principles as supported by C++, Java, etc.

Python programs are written to files with extension .py . These python source code files are compiled to byte code (python specific representation and not binary code), platform independent form stored in .pyc files. These

byte code helps in startup speed optimization. These byte code are then subjected to Python Virtual Machine PVM where one by one instructions are read and executed. This is interpreter.

2. **What built-in type does python provide?**
Following are the most commonly used built-in types provided by Python:
Immutable built-in types of python
Numbers
Strings
Tuples

Mutable built-in types of python
List
Dictionaries
Sets

3. **What is module in python?**
Modules are way to structure python program. A module would have set of related functionalities. Each python program file (.py file) is a module, which imports other modules (object) to use names (attributes) they define

using object.attribute notation. All the top level names of a module are attributes, exported for use by the importer of this module.

Filename of a module turns out to be an object name where it is imported.
import re; statement imports all the names defined in module re.
from statements can be used to import specific names from a module.
Both the above statements finds, compiles and loads (if not yet loaded) the module.
Python by default imports modules only once per file per process, when the very first import statement is encountered.
With from statement, names can be directly used. module name is not required.

4. **What is package in python?**
A folder of python programs (modules) is a package of modules. A package can have subfolders and modules.
A import statement can import packages and each import package introduces a namespace.
import folder1.subfolder2.module1
OR
from folder1.subfolder2.module1 import names
To import a package, __init__.py file must be present in each of the folders, subfolders.

5. **What is namespace in python?**
Every name introduced in a python program has a place where it lives and can be looked for. This is its namespace. Consider it as a box where a variable name mapped to object is placed. Whenever the variable name is referenced, this box is looked out to get corresponding object.
For example, functions defined using def have namespace of the module in which it is defined. And so 2 modules can define function with same name.
Modules and namespace go hand in hand. Each module introduces a namespace.
Namespace helps in reusing name by avoiding name collision. Classes and functions are other namespace constructs.

6. **What is scope in python?**
Scope for names introduced in a python program is a region where it could be used, without any qualification. That is, scope is region where the unqalified reference to a name can be looked out in the namespace to find the

object.
During execution, when a name is referred, python uses LEGB rule to find out the object. It starts looking out first into the local namespace. Then it searches name in enclosed namespace created by nested def and lambda. Then
into global namespace which is the module in which it is defined and then finally into built-in namespace.

Example 1:
```
>>> def addxy(x,y):        # x, y are local. addxy is global
...     temp=x+y           # temp is local
...     print temp
...     return temp
...
>>> addxy(1,2)
3
3
```

Example 2:

```
>>> total = 0          # total is global
>>> def addxy(x,y):
...     global total
...     total = x+y
...
>>> x
100
>>> y
200
>>> addxy(x,y)
>>> total
300
```

7. **What are the different ways of passing arguments to a function in python?**

Different forms of calling function and passing arguments to functions in python:

Function definition Function Caller Function call mapping to function definition

def func(x,y) func(a,b) Positional matching of argument

func(y=b, x=a) Argument name matching

def func(x,y=10) func(a)

func(a,b) Default value for argument

def func(x,y, *tuple_arg) func(a,b)

func(a,b,c) and many other arguments can be passed as positional arguments

Function with varying positional arguments stored in a tuple

Example:

```
def add(x,y, *tup):
    temp = x+y
    for elem in tup:
        temp = temp + elem
    return temp

print add(1,2) # prints 3
print add(1,2,3,4) # prints 10
```

def func(x,y, **dict_arg) func(a,b)

func(a,b, c=10)

func(a,b, c=10, name='abcd') and many other arguments can be passed as keyword argument

Function with varying keyword arguments stored in dictionary.

Example:

```
def percentage(mks1, mks2, **dict):
    total_mks = mks1 + mks2
    return total_mks / float( dict['out_of'] )

print percentage(65, 50, out_of=150)
```

8. **What is lambda in python?**

lamda is a single expression anonymous function often used as inline function. It takes general form as:

lambda arg1 arg2 ... : expression where args can be used

Example of lambda in python:

```
>>> triangle_perimeter = lambda a,b,c:a+b+c
>>> triangle_perimeter(2,2,2)
6
```

Difference between lamda and def :

a. def can contain multiple expressions whereas lamda is a single expression function
b. def creates a function and assigns a name so as to call it later. lambda creates a function and returns the function itself
c. def can have return statement. lambda cannot have return statements
d. lambda can be used inside list, dictionary.

9. **What is shallow copy and deep copy in python?**

Object assignment does not copy object, it gets shared. All names point to same object.
For mutable object types, modifying object using one name, reflects changes when accessed with other name.
Example :

```
>>> l=[1,2,3]
>>> l2 = l
>>> l2.pop(0)
1
>>> l2
[2, 3]
>>> l
[2, 3]
```

A copy module overcomes above problem by providing copy() and deepcopy(). copy() creates a copy of an object, creating a separate entity.
Example of shallow copy copy():

```
>>> import copy
>>> copied_l = copy.copy(l)  # performs shallow copy
>>> copied_l.pop(0)
2
>>> copied_l
[3]
>>> l
[2, 3]
```

copy() does not perform recursive copy of object. It fails for compound object types.
Example program for shallow copy problems:

```
>>> l
[[1, 2, 3], ['a', 'b', 'c']]
>>> s_list=copy.copy(l)       # performs shallow copy
>>> s_list
[[1, 2, 3], ['a', 'b', 'c']]
>>> s_list[0].pop(0)
1
>>> s_list
[[2, 3], ['a', 'b', 'c']]
>>> l
[[2, 3], ['a', 'b', 'c']]          # problem of shallow copy on compund object types
```

To overcome this problem, copy module provides deepcopy(). deepcopy() creates and returns deep copy of compound object (object containing other objects)
Example for deep copy deepcopy():

```
>>> l
[[1, 2, 3], ['a', 'b', 'c']]
>>> deep_l = copy.deepcopy(l)
>>> deep_l
[[1, 2, 3], ['a', 'b', 'c']]
>>> deep_l[0].pop(0)
1
>>> deep_l
[[2, 3], ['a', 'b', 'c']]
>>> l
[[1, 2, 3], ['a', 'b', 'c']]
```

10. **How exceptions are handle in python?**
Exceptions are raised by Python when some error is detected at run time. Exceptions can be caught in the program using try and except statments. Once the exceptions is caught, it can be corrected in the program to avoid abnormal termination. Exceptions caught inside a function can be transferred to the caller to handle it. This is done by rethrowing exception using raise. Python also provide statements to be grouped inside finally which are executed irrespective of exception thrown from within try .

Example of handling exception and rethrowing exception:

```
def func2(a,b):
    try:
        temp = a/float(b)
    except ZeroDivisionError:
        print "Exception caught. Why is b = 0? Rethrowing. Please handle"
        raise ZeroDivisionError
    finally:
        print "Always executed"


def func1():
    a=1
    b=1

    print "Attempt 1: a="+str(a)+", b="+str(b)
    func2(a,b)

    b=a-b
    print "Attempt 2: a="+str(a)+", b="+str(b)
    try:
        func2(a,b)
    except ZeroDivisionError:
        print "Caller handling exception"

func1()
```

Output:

```
Attempt 1: a=1, b=1
Always executed
```

Attempt 2: a=1, b=0
Exception caught. Why is b = 0? Rethrowing. Please handle
Always executed
Caller handling exception

11. **Give a regular expression that validates email id using python regular expression module re**
Python provides a regular expression module re
Here is the re that validates a email id of .com and .co.in subdomain:
re.search(r"[0-9a-zA-Z.]+@[a-zA-Z]+\.(com|co\.in)$","micheal.pages@mp.com")

12. **Explain file opertaions in python**
Python provides open() to open a file and open() returns a built-in type file object. The default mode is read.
fread = open("1.txt") is equivalent to fread = open("1.txt", "r"), where fread is where the file object returned by open() is stored.
Python provides read(), readline() and readlines() functions to read a file. read() reads entire file at once. readline() reads next line from the open file. readlines() returns a list where each element is a line of a file.
The file can also be open in write mode. Python provides write() to write a string in a file, writelines() to write a sequence of lines at once.
The built-in type file object has close() to which is called for all open files.

13. **What standard do you follow for Python coding guidlines?**
PEP 8 provides coding conventions for the Python code. It describes rules to adhere while coding in Python. This helps in better readability of code and thereby better understanding and easy maintainability. It covers from
code indentation, amount of space to use for indentation, spaces v/s tabs for indentation, commenting, blank lines, maximum line length, way of importing files, etc.

14. **What is pass in Python ?**
pass is no-operation Python statement. It indicates nothing is to be done. It is just a place holder used in compund statements as they cannot be left blank.

Example of using pass statement in Python:
```
>>> if x==0:
...     pass
... else:
...     print "x!=0"
```

15. **What are iterators in Python?**
Iterators in Python are used to iterate over a group of elements, containers, like list. For a container to support iterator, it must provide __iter__().
container.__iter__() :
This returns an iterator object.
Iterator protocol:
The iterator object is required to support the iterator protocol. Iterator protocol is implemented by an iterator object by providing definition of the following 2 functions:

1. iterator.__iter__() :

It returns the iterator object itself. This is required to allow both containers and iterators to be used with the for and in statements.

2. iterator.__next__() :
It returns the next item from the container. If there are no further items, raise the StopIteration exception.

Example of iterator on list:

```
>>> a=[1,2,3]
>>> i1= a.__iter__()   # creating an iterator using __iter__() on container
>>> i1
<listiterator object at 0x7f5192ccbd10>
>>> i2= iter(a)        # creating another iterator using iter() which calls __iter__() on container
>>> i2
<listiterator object at 0x7f5192ccbcd0>
>>> i1.next()
1
>>> next(i1)          # calls i1.next()
2
>>> next(i2)
1
```

Iterators are required to implement __iter__ which returns the iterator (self) . Hence it can be used with for in

```
>>> for x in i1:
...   print x
...
3
```

16. **What are generators in Python?**
Generators are way of implementing iterators. Generator function is a normal function except that it contains yield expression in the function definition making it a generator function. This function returns a generator

iterator known as generator. To get the next value from a generator, we use the same built-in function as for iterators: next() . next() takes care of calling the generator's __next__() method.

When a generator function calls yield, the "state" of the generator function is frozen; the values of all variables are saved and the next line of code to be executed is recorded until next() is called again. Once it is, the

generator function simply resumes where it left off. If next() is never called again, the state recorded during the yield call is (eventually) discarded.

Example of generators:

```
def gen_func_odd_nums():
    odd_num = 1
    while True:
```

```
        yield odd_num         # saves context and return from function
        odd_num = odd_num + 2

generator_obj = gen_func_odd_nums();
print "First 10 odd numbers:"
for i in range(10):
    print next(generator_obj) # calls generator_obj.__next__()
```

Output:
First 10 odd numbers:
1
3
5
7
9
11
13
15
17
19

17. **How do you perform unit testing in Python?**
Python provides a unit tesing framework called unittest . unittest module supports automation testing, sharing of setup and shutdown code for tests, aggregation of tests into collections, and independence of the tests from the reporting framework.

18. **What is slicing in Python?**
Slicing in Python is a mechanism to select a range of items from Sequence types like strings, list, tuple, etc.
Example of slicing:

```
>>> l=[1,2,3,4,5]
>>> l[1:3]
[2, 3]
>>> l[1:-2]
[2, 3]
>>> l[-3:-1]      # negative indexes in slicing
[3, 4]

>>> s="Hello World"
>>> s[1:3]
'el'
>>> s[:-5]
'Hello '
>>> s[-5:]
'World'
```

19. **Explain OOP principle inheritance in Python**
Classes can derive attributes from other classes via inheritance. The syntax goes:

```
class DeriveClass( BaseClass):
    <statement 1>
    <statement 2>
```

```
  ...
  <last statement >
```

If the base class is present in other module, the syntax for derivation changes to:
class DeriveClass(module.BaseClass):
Python supports overriding of base class functions by derive class. This helps in adding more functionality to be added in overriden function in derived class if required.

Python supports limited form of multiple inheritance as:

```
class DeriveClass( BaseClass1, BaseClass2, ...):
  <statement 1>
  <statement 2>
   ...
  <last statement >
```

where an attribute if not found in DeriveClass are then searched in BaseClass1 and their parent then BaseClass2 and their parents and so on. With new style, Python avoids diamond problem of reaching common base class from
multiple paths by lineraly searching base classes in left to right order.

20. **What is docstring in Python?**
docstring or Python documentation string is a way of documenting Python modules, functions, classes. PEP 257 standardize the high-level structure of docstrings. __doc__ attribute can be used to print the docstring.
Example of defining docstring:

```
>>> def test_doc_string():
...    """ this is a docstring for function test_doc_string """
...
>>> test_doc_string.__doc__
' this is a docstring for function test_doc_string '
```

PYTHON INTERVIEW QUESTIONS & ANSWERS

1. **Define python?**
Python is simple and easy to learn language compared to other programming languages. Python was introduced to the world in the year 1991 by Guido van Rossum. It is a dynamic object oriented language used for developing
software. It supports various programming languages and have a massive library support for many other languages. It is a modern powerful interpreted language with objects, modules, threads, exceptions, and automatic memory managements.

Salient features of Python are
-Simple & Easy: Python is simple language & easy to learn.
-Free/open source: it means everybody can use python without purchasing license.
-High level language: when coding in Python one need not worry about low-level details.
-Portable: Python codes are Machine & platform independent.
-Extensible: Python program supports usage of C/ C++ codes.
-Embeddable Language: Python code can be embedded within C/C++ codes & can be used a scripting language.
-Standard Library: Python standard library contains prewritten tools for programming.

-Build-in Data Structure: contains lots of data structure like lists, numbers & dictionaries.

2. **Define a method in Python?**
A function on object x is a method which is called as x.name(arguments...). Inside the definition of class, methods are defined as functions:
class C:
def meth(self, atg):
return arg*2+self.attribute

3. **Define self?**
'self' is a conventional name of method's first argument. A method which is defined as meth(self, x ,y ,z) is called as a.meth(x, y, z) for an instance of a class in which definition occurs and is called as meth(a, x ,y, z).

4. **Describe python usage in web programming?**
Python is used perfectly for web programming and have many special features to make it easy to use. Web frame works, content management systems, WebServers, CGI scripts, Webclient programming, Webservices, etc are the
features supported by python. Python language is used to create various high end applications because of its flexibility.

5. **Is there any tool used to find bugs or carrying out static analysis?**
Yes. PyChecker is the static analysis tool used in python to find bugs in source code, warns about code style and complexity etc. Pylint is a tool that verifies whether a module satisfies standards of coding and makes it
possible to add custom feature and write plug-ins.

6. **Rules for local and global variables in python?**
In python, the variables referenced inside a function are global. When a variable is assigned new value anywhere in the body of a function then it is assumed as local. In a function, if a variable ever assigned new value then
the variable is implicitly local and explicitly it should be declared as global. If all global references require global then you will be using global at anytime. You'd declare as global each reference to built-in function or to component of module which is imported. The usefulness of global declaration in identifying side-effects is defeated by this clutter.

7. **How to find methods or attributes of an object?**
Built-in dir() function of Python ,on an instance shows the instance variables as well as the methods and class attributes defined by the instance's class and all its base classes alphabetically. So by any object as argument
to dir() we can find all the methods & attributes of the object's class.
Following code snippet shows dir() at work :
class Employee:
def __init__(self,name,empCode,pay):
self.name=name
self.empCode=empCode
self.pay=pay

print("dir() listing all the Methods & attributes of class Employee")
print dir(e)

Output
dir() listing all the Methods & attributes of class Employee

['__init__', 'empCode', 'name', 'pay']

8. **Is there any equivalent to scanf() or sscanf()?**
No. Usually, the easy way to divide line into whitespace-delimited words for simple input parsing use split() method of string objects. Then, decimal strings are converted to numeric values using float() or int(). An
optional “sep” parameter is supported by split() which is useful if something is used in the place of whitespace as separator. For complex input parsing, regular expressions are powerful then sscanf() of C and perfectly suits
for the task.

9. **Define class?**
Class is a specific object type created when class statement is executed. To create instances objects, class objects can be used as templates which represent both code and data specific to datatype. In general, a class is
based on one or many classes known as base classes. It inherits methods and attributes of base classes. An object model is now permitted to redefine successively using inheritance. Basic accessor methods are provided by
generic Mailbox for subclasses and mailbox like MaildirMailbox, MboxMailbox, OutlookMailbox which handle many specific formats of mailbox.

10. **How to prevent blocking in content() method of socket?**
Commonly, select module is used to help asynchronous I/O.

11. **In python, are there any databases to DB packages?**
Yes. Bsddb package is present in Python 2.3 which offers an interface to BerkeleyDatabase library. It Interface to hashes based on disk such as GDBM and DBM are included in standard python.

12. **How do we share global variables across modules in Python?**
We can create a config file & store the entire global variable to be shared across modules or script in it. By simply importing config, the entire global variable defined it will be available for use in other modules.

For example I want a, b & c to share between modules.
config.py :
a=0
b=0
c=0

module1.py:
import config
config.a = 1
config.b =2
config.c=3
print “ a, b & resp. are : “ , config.a, config.b, config.c
---
output of module1.py will be
1 2 3

13. **How can we pass optional or keyword parameters from one function to another in Python?**

Gather the arguments using the * and ** specifiers in the function's parameter list. This gives us positional arguments as a tuple and the keyword arguments as a dictionary. Then we can pass these arguments while calling

another function by using * and **:
```
def fun1(a, *tup, **keywordArg):
...
keywordArg['width']='23.3c'
...
Fun2(a, *tup, **keywordArg)
```

14**. Explain pickling and unpickling.**
Pickle is a standard module which serializes & de-serializes a python object structure. Pickle module accepts any python object converts it into a string representation & dumps it into a file(by using dump() function) which
can be used later, process is called pickling. Whereas unpickling is process of retrieving original python object from the stored string representation for use.

15. **Explain how python is interpreted.**

Python program runs directly from the source code. Each type Python programs are executed code is required. Python converts source code written by the programmer into intermediate language which is again translated it into the native language / machine language that is executed. So Python is an Interpreted language.

16. **How is memory managed in python?**
Memory management in Python involves a private heap containing all Python objects and data structures. Interpreter takes care of Python heap and that the programmer has no access to it. The allocation of heap space for Python objects is done by Python memory manager. The core API of Python provides some tools for the programmer to code reliable and more robust program. Python also has a build-in garbage collector which recycles all the unused memory. When an object is no longer referenced by the program, the heap space it occupies can be freed. The garbage collector determines objects which are no longer referenced by the sprogram frees the occupied memory and make it available to the heap space. The gc module defines functions to enable /disable garbage collector:
gc.enable() -Enables automatic garbage collection.
gc.disable() - Disables automatic garbage collection.

17. **Explain indexing and slicing operation in sequences**
Different types of sequences in python are strings, Unicode strings, lists, tuples, buffers, and xrange objects. Slicing & indexing operations are salient features of sequence. indexing operation allows to access a particular
item in the sequence directly (similar to the array/list indexing) and the slicing operation allows to retrieve a part of the sequence. The slicing operation is used by specifying the name of the sequence followed by an
optional pair of numbers separated by a colon within square brackets say S[startno.:stopno]. The startno in the slicing operation indicates the position from where the slice starts and the stopno indicates where the slice
will stop at. If the startno is ommited, Python will start at the beginning of the sequence. If the stopno is ommited, Python will stop at the end of the sequence..

Following code will further explain indexing & slicing operation:

>>> cosmeticList =[‘lipsstick’,’facepowder’,eyeliner’,’blusher’,kajal’]
>>> print “Slicing operation :”,cosmeticList[2:]
Slicing operation :[‘eyeliner’,’blusher’,kajal’]
>>>print “Indexing operation :”,cosmeticList[0]
“Indexing operation :lipsstick

18. **Explain how to make Forms in python.**
As python is scripting language forms processing is done by Python. We need to import cgi module to access form fields using FieldStorage class.
Every instance of class FieldStorage (for ‘form’) has the following attributes:

form.name: The name of the field, if specified.
form.filename: If an FTP transaction, the client-side filename.
form.value: The value of the field as a string.
form.file: file object from which data can be read.
form.type: The content type, if applicable.
form.type_options: The options of the ‘content-type’ line of the HTTP request, returned as a dictionary.
form.disposition: The field ‘content-disposition'; None if unspecified.
form.disposition_options: The options for ‘content-disposition’.
form.headers: All of the HTTP headers returned as a dictionary.

A code snippet of form handling in python:

```
import cgi
form = cgi.FieldStorage()
if not (form.has_key("name") and form.has_key("age")):
print "<H1>Name & Age not Entered</H1>"
print "Fill the Name & Age accurately."
return
print "<p>name:", form["name"].value
print "<p>Age:", form["age"].value
```

29. **Describe how to implement Cookies for Web python.**
A cookie is an arbitrary string of characters that uniquely identify a session. Each cookie is specific to one Web site and one user.
The Cookie module defines classes for abstracting the concept of cookies. It contains following method to creates cookie
Cookie.SimpleCookie([input])
Cookie.SerialCookie([input]
Cookie.SmartCookie([input])
for instance following code creates a new cookie ck-

```
import Cookie
ck= Cookie.SimpleCookie ( x )
```

20.**What are uses of lambda?**
It used to create small anonymous functions at run time. Like e.g.

```
def fun1(x):
return x**2
print fun1(2)
```

it gives you answer 4
the same thing can be done using

```
sq=lambda x: x**2
print sq(2)
```

it gives the answer 4

21. **When do you use list vs. tuple vs. dictionary vs. set?**
List and Tuple are both ordered containers. If you want an ordered container of constant elements use tuple as tuples are immutable objects.

22. **When you need ordered container of things, which will be manipulated, use lists.**
Dictionary is key, value pair container and hence is not ordered. Use it when you need fast access to elements, not in ordered fashion. Lists are indexed and index of the list cannot be “string” e.g. list [‘myelement’] is not a valid statement in python.

23. **Do they know a tuple/list/dict when they see it?**
Dictionaries are consisting of pair of keys and values.like {’key’:’value’}.
book={’cprog’:'1024',’c++’:'4512'}
Keys are unique but values can be same. The main difference between list and tuple is you can change the list but you cannot change the tuple. Tuple can be used as keys in mapping where list is not.

24. **Why was the language called as Python?**
At the same time he began implementing Python, Guido van Rossum was also reading the published scripts from “Monty Python’s Flying Circus” (a BBC comedy series from the seventies, in the unlikely case you didn’t know). It
occurred to him that he needed a name that was short, unique, and slightly mysterious, so he decided to call the language Python.

25. **What is used to represent Strings in Python? Is double quotes used for String representation or single quotes used for String representation in Python?**

Using Single Quotes (‘)
You can specify strings using single quotes such as ‘Quote me on this’ . All white space i.e. spaces and tabs are preserved as-is.

Using Double Quotes (“)
Strings in double quotes work exactly the same way as strings in single quotes. An example is “What’s your name?”

Using Triple Quotes (”’ or “””)
You can specify multi-line strings using triple quotes. You can use single quotes and double quotes freely within the triple quotes. An example is

”’This is a multi-line string. This is the first line.
This is the second line.
“What’s your name?,” I asked.
He said “Bond, James Bond.”

26. **Why cannot lambda forms in Python contain statements?**
A lambda statement is used to create new function objects and then return them at runtime that is why lambda forms in Python did not contain statement.

27. **Which of the languages does Python resemble in its class syntax?**
C++ is the appropriate language that Python resemble in its class syntax.

28**. Does Python support strongly for regular expressions? What are the other languages that support strongly for regular expressions?**

Yes, python strongly support regular expression. Other languages supporting regular expressions are: Delphi, Java, Java script, .NET, Perl, Php, Posix, python, Ruby, Tcl, Visual Basic, XML schema, VB script, Visual Basic 6.

29. **Why is not all memory freed when Python exits?**

Objects referenced from the global namespaces of Python modules are not always de-allocated when Python exits. This may happen if there are circular references. There are also certain bits of memory that are allocated by the
C library that are impossible to free (e.g. a tool like the one Purify will complain about these). Python is, however, aggressive about cleaning up memory on exit and does try to destroy every single object.
If you want to force Python to delete certain things on de-allocation, you can use the at exit module to register one or more exit functions to handle those deletions.

30. **What is a Lambda form? Explain about assert statement?**

The lambda form:
Using lambda keyword tiny anonymous functions can be created. It is a very powerful feature of Python which declares a one-line unknown small function on the fly. The lambda is used to create new function objects and then
return them at run-time. The general format for lambda form is:
lambda parameter(s): expression using the parameter(s)

For instance k is lambda function-

```
>>> k= lambda y: y + y
>>> k(30)
60
>>> k(40)
80
```

The assert statement:
The build-in assert statement of python introduced in version 1.5 is used to assert that something is true. Programmers often place assertions at the beginning of a function to check for valid input, and after function call
to check for valid output. Assert statement can be removed after testing of program is over. If assert evaluates to be false, an AssertionError exception is raised. AssertionError exceptions can be handled with the try-except

statement.

The general syntax for assert statement is:
assert Expression[, Arguments]

31. **Explain the role of repr function.**

Python can convert any value to a string by making use of two functions repr() or str(). The str() function returns representations of values which are human-readable, while repr() generates representations which can be read
by the interpreter. repr() returns a machine-readable representation of values, suitable for an exec command. Following code sniipets shows working of repr() & str() :

```
def fun():
```

```
y=2333.3
x=str(y)
z=repr(y)
print " y :",y
print "str(y) :",x
print "repr(y):",z
fun()
-------------
output
y : 2333.3
str(y) : 2333.3
repr(y) : 2333.3000000000002
```

32. **What is LIST comprehensions features of Python used for?**

LIST comprehensions features were introduced in Python version 2.0, it creates a new list based on existing list. It maps a list into another list by applying a function to each of the elements of the existing list. List comprehensions creates lists without using map() , filter() or lambda form.

33. **How do you make a higher order function in Python?**

A higher-order function accepts one or more functions as input and returns a new function. Sometimes it is required to use function as data. To make high order function , we need to import functools module The functools.partial() function is used often for high order function.

34. **Explain how to copy an object in Python.**

There are two ways in which objects can be copied in python. Shallow copy & Deep copy. Shallow copies duplicate as minute as possible whereas Deep copies duplicate everything. If a is object to be copied then

-copy.copy(a) returns a shallow copy of a.
-copy.deepcopy(a) returns a deep copy of a.

35. How do I convert a string to a number?

Python contains several built-in functions to convert values from one data type to another data type.

The int function takes string and coverts it to an integer.

```
s = "1234" # s is string
i = int(s) # string converted to int
print i+2
------------------------
1236
```

The float function converts strings into float number.

```
s = "1234.22" # s is string
i = float(s) # string converted to float
print i
-------------------------
1234.22
```

36. **What is a negative index in python?**

Python arrays & list items can be accessed with positive or negative numbers (also known as index). For instance our array/list is of size n, then for positive index 0 is the first index, 1 second, last index will be n-1. For negative index, -n is the first index, -(n-1) second, last negative index will be – 1. A negative index accesses elements from the end of the list counting backwards.

An example to show negative index in python.

```
>>> import array
>>> a= [1, 2, 3]
>>> print a[-3]
1
>>> print a[-2]
2
>>> print a[-1]
3
```

37 .**How do you make an array in Python?**

The array module contains methods for creating arrays of fixed types with homogeneous data types. Arrays are slower then list. Array of characters, integers, floating point numbers can be created using array module. array
(typecode[, intializer]) Returns a new array whose items are constrained by typecode, and initialized from the optional initialized value. Where the typecode can be for instance ‘c’ for character value, ‘d’ for double, ‘f’ for float.

38. **Explain how to create a multidimensional list**.

There are two ways in which Multidimensional list can be created:
By direct initializing the list as shown below to create multidimlist below

```
>>>multidimlist = [ [227, 122, 223],[222, 321, 192],[21, 122, 444]]
>>>print multidimlist[0]
>>>print multidimlist[1][2]
```

_____

Output

```
[227, 122, 223]
192
```

The second approach is to create a list of the desired length first and then fill in each element with a newly created lists demonstrated below :

```
>>>list=[0]*3
>>>for i in range(3):
>>> list[i]=[0]*2
>>>for i in range (3):
>>> for j in range(2):
>>> list[i][j] = i+j
>>>print list
```

_____

Output

```
[[0, 1], [1, 2], [2, 3]]
```

39. **Explain how to overload constructors (or methods) in Python**.

_init__ () is a first method defined in a class. when an instance of a class is created, python calls __init__() to initialize the attribute of the object.

Following example demonstrate further:

```
class Employee:

def __init__(self, name, empCode,pay):
self.name=name
self.empCode=empCode
```

```
self.pay=pay

e1 = Employee("Sarah",99,30000.00)

e2 = Employee("Asrar",100,60000.00)
print("Employee Details:")

print(" Name:",e1.name,"Code:", e1.empCode,"Pay:", e1.pay)
print(" Name:",e2.name,"Code:", e2.empCode,"Pay:", e2.pay)
```

Output

```
Employee Details:
(' Name:', 'Sarah', 'Code:', 99, 'Pay:', 30000.0)
(' Name:', 'Asrar', 'Code:', 100, 'Pay:', 60000.0)
```

40. **Describe how to send mail from a Python script**.
The smtplib module defines an SMTP client session object that can be used to send mail to any Internet machine.
A sample email is demonstrated below.

```
import smtplib
SERVER = smtplib.SMTP('smtp.server.domain')
FROM = sender@mail.com
TO = ["user@mail.com"] # must be a list
SUBJECT = "Hello!"
TEXT = "This message was sent with Python's smtplib."
# Main message
message = """
From: Sarah Naaz < sender@mail.com >
To: CarreerRide user@mail.com
Subject: SMTP email msg
This is a test email. Acknowledge the email by responding.
""" % (FROM, ", ".join(TO), SUBJECT, TEXT)
server = smtplib.SMTP(SERVER)
server.sendmail(FROM, TO, message)
server.quit()
```

41. **Describe how to generate random numbers in Python.**
Thee standard module random implements a random number generator.\
There are also many other in this module, such as:
uniform(a, b) returns a floating point number in the range [a, b].
randint(a, b)returns a random integer number in the range [a, b].
random()returns a floating point number in the range [0, 1].

Following code snippet show usage of all the three functions of module random:
Note: output of this code will be different evertime it is executed.

```
import random
i = random.randint(1,99)# i randomly initialized by integer between range 1 & 99
j= random.uniform(1,999)# j randomly initialized by float between range 1 & 999
k= random.random()# k randomly initialized by float between range 0 & 1
print("i :" ,i)
print("j :" ,j)
print("k :" ,k)
```

_____

Output -
('i :', 64)
('j :', 701.85008797642115)
('k :', 0.18173593240301023)

Output-
('i :', 83)
('j :', 56.817584548210945)
('k :', 0.9946957743038618)

42. **What is the optional statement used in a try except statement in Python?**
There are two optional clauses used in try except statements:
1. Else clause: It is useful for code that must be executed when the try block does not create any exception
2. Finally clause: It is useful for code that must be executed irrespective of whether an exception is generated or not.

43. **What is used to create Unicode string in Python?**
Add u before the string
>>> u 'test'

44. **What are the uses of List Comprehensions feature of Python?**
List comprehensions help to create and manage lists in a simpler and clearer way than using map(), filter() and lambda. Each list comprehension consists of an expression followed by a clause, then zero or more for or if
clauses.

45. **Which all are the operating system that Python can run on?**
Python can run of every operating system like UNIX/LINUX, Mac, Windows, and others.

46. **What is the statement that can be used in Python if a statement is required syntactically but the program requires no action?**
Pass is a no-operation/action statement in python
If we want to load a module and if it does not exist, let us not bother, let us try to do other task. The following example demonstrates that.

Try:
Import module1
Except:
Pass

47. **What is the Java implementation of Python popularly known as?**
Jython

48. **What is the method does join() in python belong?**
String method

49. **Does python support switch or case statement in Python? If not what is the reason for the same?**
No. You can use multiple if-else, as there is no need for this.

50. **How is the Implementation of Pythons dictionaries done?**
Using curly brackets -> {}
E.g.: {'a':'123', 'b':'456'}

51. **What is the language from which Python has got its features or derived its features?**
Most of the object oriented programming languages to name a few are C++, CLISP and Java is the language from which Python has got its features or derived its features.

52. **What are the disadvantages of the Python programming language?**
One of the disadvantages of the Python programming language is it is not suited for fast and memory intensive tasks.

53. **Why is not all memory freed when Python exits?**
Objects referenced from the global namespaces of Python modules are not always de-allocated when Python exits. This may happen if there are circular references. There are also certain bits of memory that are allocated by the C library that are impossible to free (e.g. a tool like the one Purify will complain about these). Python is, however, aggressive about cleaning up memory on exit and does try to destroy every single object.
If you want to force Python to delete certain things on de-allocation, you can use the at exit module to register one or more exit functions to handle those deletions.

54. **Which of the languages does Python resemble in its class syntax?**
C++ is the appropriate language that Python resemble in its class syntax.

55. **Who created the Python programming language?**
Python programming language was created by Guido van Rossum.
==========

1 line: Output
```
print 'Hello, world!'
```

2 lines: Input, assignment
```
name = raw_input('What is your name?\n')
print 'Hi, %s.' % name
```

3 lines: For loop, built-in enumerate function, new style formatting
```
friends = ['john', 'pat', 'gary', 'michael']
for i, name in enumerate(friends):
    print "iteration {iteration} is {name}".format(iteration=i, name=name)
```

4 lines: Fibonacci, tuple assignment
```
parents, babies = (1, 1)
while babies < 100:
    print 'This generation has {0} babies'.format(babies)
    parents, babies = (babies, parents + babies)
```

5 lines: Functions
```
def greet(name):
    print 'Hello', name
greet('Jack')
greet('Jill')
greet('Bob')
```

6 lines: Import, regular expressions
```
import re
```

```
for test_string in ['555-1212', 'ILL-EGAL']:
    if re.match(r'^\d{3}-\d{4}$', test_string):
        print test_string, 'is a valid US local phone number'
    else:
        print test_string, 'rejected'

7 lines: Dictionaries, generator expressions
prices = {'apple': 0.40, 'banana': 0.50}
my_purchase = {
    'apple': 1,
    'banana': 6}
grocery_bill = sum(prices[fruit] * my_purchase[fruit]
                   for fruit in my_purchase)
print 'I owe the grocer $%.2f' % grocery_bill

8 lines: Command line arguments, exception handling
# This program adds up integers in the command line
import sys
try:
    total = sum(int(arg) for arg in sys.argv[1:])
    print 'sum =', total
except ValueError:
    print 'Please supply integer arguments'

9 lines: Opening files
# indent your Python code to put into an email
import glob
# glob supports Unix style pathname extensions
python_files = glob.glob('*.py')
for file_name in sorted(python_files):
    print '    ------' + file_name

    with open(file_name) as f:
        for line in f:
            print '    ' + line.rstrip()

    print

10 lines: Time, conditionals, from..import, for..else
from time import localtime
activities = {8: 'Sleeping',
              9: 'Commuting',
              17: 'Working',
              18: 'Commuting',
              20: 'Eating',
              22: 'Resting' }

time_now = localtime()
hour = time_now.tm_hour

for activity_time in sorted(activities.keys()):
    if hour < activity_time:
        print activities[activity_time]
        break
else:
```

```
    print 'Unknown, AFK or sleeping!'

11 lines: Triple-quoted strings, while loop
REFRAIN = '''
%d bottles of beer on the wall,
%d bottles of beer,
take one down, pass it around,
%d bottles of beer on the wall!
'''
bottles_of_beer = 99
while bottles_of_beer > 1:
    print REFRAIN % (bottles_of_beer, bottles_of_beer,
        bottles_of_beer - 1)
    bottles_of_beer -= 1

12 lines: Classes
class BankAccount(object):
    def __init__(self, initial_balance=0):
        self.balance = initial_balance
    def deposit(self, amount):
        self.balance += amount
    def withdraw(self, amount):
        self.balance -= amount
    def overdrawn(self):
        return self.balance < 0
my_account = BankAccount(15)
my_account.withdraw(5)
print my_account.balance

13 lines: Unit testing with unittest
import unittest
def median(pool):
    copy = sorted(pool)
    size = len(copy)
    if size % 2 == 1:
        return copy[(size - 1) / 2]
    else:
        return (copy[size/2 - 1] + copy[size/2]) / 2
class TestMedian(unittest.TestCase):
    def testMedian(self):
        self.failUnlessEqual(median([2, 9, 9, 7, 9, 2, 4, 5, 8]), 7)
if __name__ == '__main__':
    unittest.main()

14 lines: Doctest-based testing
def median(pool):
    '''Statistical median to demonstrate doctest.
    >>> median([2, 9, 9, 7, 9, 2, 4, 5, 8])
    7
    '''
    copy = sorted(pool)
    size = len(copy)
    if size % 2 == 1:
        return copy[(size - 1) / 2]
    else:
```

```
        return (copy[size/2 - 1] + copy[size/2]) / 2
if __name__ == '__main__':
    import doctest
    doctest.testmod()

15 lines: itertools
from itertools import groupby
lines = '''
This is the
first paragraph.

This is the second.
'''.splitlines()
# Use itertools.groupby and bool to return groups of
# consecutive lines that either have content or don't.
for has_chars, frags in groupby(lines, bool):
    if has_chars:
        print ' '.join(frags)
# PRINTS:
# This is the first paragraph.
# This is the second.

16 lines: csv module, tuple unpacking, cmp() built-in
import csv
# write stocks data as comma-separated values
writer = csv.writer(open('stocks.csv', 'wb', buffering=0))
writer.writerows([
    ('GOOG', 'Google, Inc.', 505.24, 0.47, 0.09),
    ('YHOO', 'Yahoo! Inc.', 27.38, 0.33, 1.22),
    ('CNET', 'CNET Networks, Inc.', 8.62, -0.13, -1.49)
])

# read stocks data, print status messages
stocks = csv.reader(open('stocks.csv', 'rb'))
status_labels = {-1: 'down', 0: 'unchanged', 1: 'up'}
for ticker, name, price, change, pct in stocks:
    status = status_labels[cmp(float(change), 0.0)]
    print '%s is %s (%s%%)' % (name, status, pct)

18 lines: 8-Queens Problem (recursion)
BOARD_SIZE = 8
def under_attack(col, queens):
    left = right = col

    for r, c in reversed(queens):
        left, right = left - 1, right + 1

        if c in (left, col, right):
            return True
    return False

def solve(n):
    if n == 0:
        return [[]]
```

```
    smaller_solutions = solve(n - 1)

    return [solution+[(n,i+1)]
        for i in xrange(BOARD_SIZE)
            for solution in smaller_solutions
                if not under_attack(i+1, solution)]
for answer in solve(BOARD_SIZE):
    print answer

20 lines: Prime numbers sieve w/fancy generators
import itertools
def iter_primes():
     # an iterator of all numbers between 2 and +infinity
     numbers = itertools.count(2)

     # generate primes forever
     while True:
         # get the first number from the iterator (always a prime)
         prime = numbers.next()
         yield prime

         # this code iteratively builds up a chain of
         # filters...slightly tricky, but ponder it a bit
         numbers = itertools.ifilter(prime.__rmod__, numbers)

for p in iter_primes():
    if p > 1000:
        break
    print p

21 lines: XML/HTML parsing (using Python 2.5 or third-party library)
dinner_recipe = '''<html><body><table>
<tr><th>amt</th><th>unit</th><th>item</th></tr>
<tr><td>24</td><td>slices</td><td>baguette</td></tr>
<tr><td>2+</td><td>tbsp</td><td>olive oil</td></tr>
<tr><td>1</td><td>cup</td><td>tomatoes</td></tr>
<tr><td>1</td><td>jar</td><td>pesto</td></tr>
</table></body></html>'''

# In Python 2.5 or from http://effbot.org/zone/element-index.htm
import xml.etree.ElementTree as etree
tree = etree.fromstring(dinner_recipe)

# For invalid HTML use http://effbot.org/zone/element-soup.htm
# import ElementSoup, StringIO
# tree = ElementSoup.parse(StringIO.StringIO(dinner_recipe))

pantry = set(['olive oil', 'pesto'])
for ingredient in tree.getiterator('tr'):
    amt, unit, item = ingredient
    if item.tag == "td" and item.text not in pantry:
        print "%s: %s %s" % (item.text, amt.text, unit.text)

28 lines: 8-Queens Problem (define your own exceptions)
BOARD_SIZE = 8
```

```
class BailOut(Exception):
    pass

def validate(queens):
    left = right = col = queens[-1]
    for r in reversed(queens[:-1]):
        left, right = left-1, right+1
        if r in (left, col, right):
            raise BailOut

def add_queen(queens):
    for i in range(BOARD_SIZE):
        test_queens = queens + [i]
        try:
            validate(test_queens)
            if len(test_queens) == BOARD_SIZE:
                return test_queens
            else:
                return add_queen(test_queens)
        except BailOut:
            pass
    raise BailOut

queens = add_queen([])
print queens
print "\n".join(". "*q + "Q " + ". "*(BOARD_SIZE-q-1) for q in queens)
```

33 lines: "Guess the Number" Game (edited) from http://inventwithpython.com

```
import random
guesses_made = 0
name = raw_input('Hello! What is your name?\n')
number = random.randint(1, 20)
print 'Well, {0}, I am thinking of a number between 1 and 20.'.format(name)

while guesses_made < 6:

    guess = int(raw_input('Take a guess: '))

    guesses_made += 1

    if guess < number:
        print 'Your guess is too low.'

    if guess > number:
        print 'Your guess is too high.'

    if guess == number:
        break

if guess == number:
    print 'Good job, {0}! You guessed my number in {1} guesses!'.format(name,
guesses_made)
else:
    print 'Nope. The number I was thinking of was {0}'.format(number)
```

==========
Here is a set of small scripts, which demonstrate some features of
Python programming.

```
# this is the first comment
#! python
# integer variables
SPAM = 1
#! python
print "Hello, Python"


#! python
# string variable
STRING = "# This is not a comment."
print STRING

#! python
# integer arith
a=4
print a

b=12+5
print b

c=b%a
print c


#! python

# trailing comma
i = 256*256
print 'The value of i is', i

#! python
# Fibonacci series:
# the sum of two elements defines the next
a, b = 0, 1
while b < 200:
      print b,
      a, b = b, a+b


#! python
# input and operator if
x = int(raw_input("Please enter an integer: "))
if x < 0:
     x = 0
     print 'Negative changed to zero'
elif x == 0:
     print 'Zero'
elif x == 1:
     print 'Single'
else:
```

```
      print 'More'
```

```
#! python
# operator for:
# Measure some strings:
a = ['cat', 'window', 'defenestrate']
for x in a:
    print x, len(x)
```

```
#! python
# range function
print range(10)
print range(5, 10)
print range(0, 10, 3)
```

```
a = ['Mary', 'had', 'a', 'little', 'lamb']
for i in range(len(a)):
    print i, a[i]
```

```
#! python
# break operator
# prime numbers

for n in range(2, 1000):
    for x in range(2, n):
        if n % x == 0:
            print n, 'equals', x, '*', n/x
            break
    else:
        # loop fell through without finding a factor
        print n, 'is a prime number'
```

```
#! python
#pass statement does nothing.
#It can be used when a statement is required syntactically but the program requires no
action. For example:

while True:
      pass # Busy-wait for keyboard interrupt
```

```
#! python
# Defining Functions
def fib(n):    # write Fibonacci series up to n
    """Print a Fibonacci series up to n."""
    a, b = 0, 1
    while b < n:
        print b,
        a, b = b, a+b
```

```
# Now call the function we just defined:
fib(2000)
! python

# function that returns a list of the numbers of the Fibonacci series
def fib2(n): # return Fibonacci series up to n
   """Return a list containing the Fibonacci series up to n."""
   result = []
   a, b = 0, 1
   while b < n:
      result.append(b)    # see below
      a, b = b, a+b
   return result

#====================================
f100 = fib2(100)    # call it
print f100          # write the result

#! python
# work with strings
# Strings can be concatenated (glued together) with the + operator, and repeated with *:
word = 'Help' + 'A'
print word
print '<' + word*5 + '>'

# Two string literals next to each other are automatically concatenated;
# the first line above could also have been written "word = 'Help' 'A'";
# this only works with two literals, not with arbitrary string expressions:

st='str' 'ing'             #  <-  This is ok
print st
st='str'.strip() + 'ing'   #  <-  This is ok
print st

# Strings can be subscripted (indexed); like in C, the first character of a string
# has subscript (index) 0. There is no separate character type; a character is
# simply a string of size one. Like in Icon, substrings can be specified with
# the slice notation: two indices separated by a colon.

print word[4]
print word[0:2]
print word[2:4]

# Slice indices have useful defaults; an omitted first index defaults to zero,
# an omitted second index defaults to the size of the string being sliced.

print word[:2]    # The first two characters
print word[2:]    # All but the first two characters

# Python strings cannot be changed. Assigning to an indexed position in the string results in
an error:
# However, creating a new string with the combined content is easy and efficient:

print 'x' + word[1:]
print 'Splat' + word[4]
```

```
# Here's a useful invariant of slice operations: s[:i] + s[i:] equals s.

print word[:2] + word[2:]
print word[:3] + word[3:]


# Degenerate slice indices are handled gracefully: an index that is too large is replaced
# by the string size, an upper bound smaller than the lower bound returns an empty string.

print word[1:100]
print word[10:]
print word[2:1]


# Indices may be negative numbers, to start counting from the right. For example:
print word[-1]     # The last character
print word[-2]     # The last-but-one character
print word[-2:]    # The last two characters
print word[:-2]    # All but the last two characters

# But note that -0 is really the same as 0, so it does not count from the right!
print word[-0]     # (since -0 equals 0)

# Out-of-range negative slice indices are truncated, but don't try this for single-element
(non-slice) indices:
print word[-100:]

# print word[-10]    # error

#The best way to remember how slices work is to think of the indices as pointing between
characters,
#with the left edge of the first character numbered 0. Then the right edge of the last
character
#of a string of n characters has index n, for example:
# +---+---+---+---+---+
# | H | e | l | p | A |
# +---+---+---+---+---+
# 0   1   2   3   4   5
#-5  -4  -3  -2  -1


s = 'supercalifragilisticexpialidocious'
print s
print len(s)

#! python
# Default Argument Values
def ask_ok(prompt, retries=4, complaint='Yes or no, please!'):
    while True:
        ok = raw_input(prompt)
        if ok in ('y', 'ye', 'yes'): return True
        if ok in ('n', 'no', 'nop', 'nope'): return False
        retries = retries - 1
        if retries < 0: raise IOError, 'refusenik user'
```

```
        print complaint

#=============================================================
========

i = 5

def f(arg=i):
    print arg

i = 6
f()
#=============================================================
========

z=ask_ok('really quit???')

if z==False :
          print "bad"



#! python
# Lambda Forms

def make_incrementor(n):
    return lambda x: x + n

#====================================
f = make_incrementor(42)

print f(0)
print f(1)
print f(15)

//=============================================================
============================
//
//=============================================================
============================
#! python

# speed test
nn=10000000
i=0;
s=0;

print "beginning..."

while i
#! python

# raw input of strings only!
st = raw_input("")
```

```
print st
st=st*3  # triple the string
print st

#! python
# math
import math
print math.cos(math.pi / 4.0)
print math.log(1024, 2)

#! python
# random
import random
print random.choice(['apple', 'pear', 'banana'])
print random.sample(xrange(100), 10)   # sampling without replacement
print random.random()    # random float
print random.randrange(6)    # random integer chosen from range(6)

#! python
def perm(l):
        # Compute the list of all permutations of l
    if len(l) <= 1:
                  return [l]
    r = []  # here is new list with all permutations!
    for i in range(len(l)):
             s = l[:i] + l[i+1:]
             p = perm(s)
             for x in p:
              r.append(l[i:i+1] + x)
    return r

#=============================================
a=[1,2,3]

print perm(a)


#! python
a=2+3j
b=2-3j

print a*a
print a*b
print a.real
print b.imag


#! python

while True:
    try:
        x = int(raw_input("Please enter a number: "))
        break
    except ValueError:
        print "Oops!  That was no valid number.  Try again..."
```

```
#! python
import string, sys

try:
    f = open('myfile.txt')
    s = f.readline()
    i = int(string.strip(s))
except IOError, (errno, strerror):
    print "I/O error(%s): %s" % (errno, strerror)
except ValueError:
    print "Could not convert data to an integer."
except:
    print "Unexpected error:", sys.exc_info()[0]
    raise


#! python
# work with lists
a = ['spam', 'eggs', 100, 1234]
print " list a=",a

# list indices start at 0,
print 'a[0]=', a[0]
print 'a[3]=', a[3]
print 'a[-2]=', a[-2]

# lists can be sliced, concatenated and so on:
print  "a[1:-1]=", a[1:-1]
print a[:2] + ['bacon', 2*2]

print 3*a[:3] + ['Boe!']

# possible to change individual elements of a list:
a[2] = a[2] + 23
print "changing a[2]=", a

#Assignment to slices is also possible, and this can even change the size of the list:
# Replace some items:
a[0:2] = [1, 12]
print a

# Remove some:
a[0:2] = []
print a

# Insert some:
a[1:1] = ['bletch', 'xyzzy']
print a

a[:0] = a     # Insert (a copy of) itself at the beginning
print a

print "length=", len(a)
```

```
# possible to nest lists (create lists containing other lists)
q = [2, 3]
p = [1, q, 4]
print " nest list=", p
print 'length =', len(p)
print p[1]

print p[1][0]
p[1].append('xtra')
print p
print q


#! python
# more work with lists
a = [66.6, 333, 333, 1, 1234.5]
print a.count(333), a.count(66.6), a.count('x')
a.insert(2, -1)
print a

a.append(333)
print a

print a.index(333)

a.remove(333)
print a

a.reverse()
print a

a.sort()
print a

#! python
# huge list making
nn=1000000
a = []
i=0

while i
#! python
# Using Lists as Stacks

stack = [3, 4, 5]
stack.append(6)
stack.append(7)
print stack

x=stack.pop()
print "popped ",x
print stack

x=stack.pop()
```

```
print "popped ",x
x=stack.pop()
print "popped ",x
print stack

#! python
# Using Lists as Queues

queue = ["Eric", "John", "Michael"]
queue.append("Terry")           # Terry arrives
queue.append("Graham")          # Graham arrives
print queue

s=queue.pop(0)
print s
s=queue.pop(0)
print s
print queue

#! python
# The del statement
a = [-1, 1, 66.6, 333, 333, 1234.5]
del a[0]
print a

del a[2:4]
print a

#! python
# filter of sequence

def f(x): return x % 2 != 0 and x % 3 != 0
res=filter(f, range(2, 25))
print res


#! python
# map of sequence
def cube(x): return x*x*x
res=map(cube, range(1, 11))
print res


#! python
# reduce(func, sequence)" returns a single value constructed by
# calling the binary function func on the first two items of the sequence,
# then on the result and the next item, and so on
def add(x,y): return x+y
r=reduce(add, range(1, 11))
print r # 55

#! python
# A tuple consists of a number of values separated by commas
t = 12345, 54321, 'hello!' # tuple packing
print t[0]
```

```
print t
(12345, 54321, 'hello!')

# Tuples may be nested:
u = t, (1, 2, 3, 4, 5)
print u     #  ((12345, 54321, 'hello!'), (1, 2, 3, 4, 5))


#! python
# Dictionaries are sometimes as ``associative memories'' or ``associative arrays''
tel = {'jack': 4098, 'sape': 4139}
tel['guido'] = 4127
print tel
print tel['jack']
del tel['sape']
tel['irv'] = 4127
print tel
print tel.keys()
x=tel.has_key('guido')
print x

# The dict() constructor builds dictionaries directly from lists
# of key-value pairs stored as tuples. When the pairs form a pattern,
# list comprehensions can compactly specify the key-value list.

d=dict([('sape', 4139), ('guido', 4127), ('jack', 4098)])
print d

vec=[1,2,3,4,5]
dd=dict([(x, x**2) for x in vec])     # use a list comprehension
print dd



#! python
# Standard Module sys
import sys
print sys.path
sys.path.append('c:\temp')
print sys.path
print sys.version
print sys.platform
print sys.maxint

#! python
#=============================================================
=
# dir() is used to find out which names a module defines
import sys
print dir(sys)

# Without arguments, dir() lists the names you have defined currentl

#! python
```

```
#  convert any value to a string: pass it to the repr() or str()
s = 'Hello, world.'
print str(s)
print repr(s)
print str(0.1)
print repr(0.1)
x = 10 * 3.25
y = 200 * 200
s = 'The value of x is ' + repr(x) + ', and y is ' + repr(y) + '...'
print s

# The repr() of a string adds string quotes and backslashes:
hello = 'hello, world\n'
hellos = repr(hello)
print hellos   # 'hello, world\n'

# The argument to repr() may be any Python object:
print repr((x, y, ('spam', 'eggs')))

# reverse quotes are convenient in interactive sessions:
print `x, y, ('spam', 'eggs')`

#! python
#  two ways to write a table of squares and cubes:

for x in range(1, 11):
    print repr(x).rjust(2), repr(x*x).rjust(3),
    # Note trailing comma on previous line
    print repr(x*x*x).rjust(4)


print '=================================================='
for x in range(1,11):
    print '%2d %3d %4d' % (x, x*x, x*x*x)

#! python
# output results from running "python demo.py one two three"
# at the command line:

import sys
print sys.argv[]  #  ['demo.py', 'one', 'two', 'three']




#! python
# String Pattern Matching - regular expression
import re
r=re.findall(r'\bf[a-z]*', 'which foot or hand fell fastest')
print r # ['foot', 'fell', 'fastest']

s=re.sub(r'(\b[a-z]+) \1', r'\1', 'cat in the the hat')
print s # 'cat in the hat'

#! python
```

```
# dates are easily constructed and formatted
from datetime import date
now = date.today()
print now
datetime.date(2003, 12, 2)
print now.strftime("%m-%d-%y or %d%b %Y is a %A on the %d day of %B")

# dates support calendar arithmetic
birthday = date(1964, 7, 31)
age = now - birthday
print age.days # 14368



#! python
# Internet Access
import urllib2
for line in urllib2.urlopen('http://tycho.usno.navy.mil/cgi-bin/timer.pl'):
          if 'EST' in line:      # look for Eastern Standard Time
              print line
import smtplib
server = smtplib.SMTP('localhost')
server.sendmail('soothsayer@tmp.org', 'jceasar@tmp.org',
"""To: jceasar@tmp.org
From: soothsayer@tmp.org
Beware the Ides of March.
""")
server.quit()


# work with files
#open file for write

f=open('c:/TEMP/workpy.txt','w')
print f
f.write("aaaaaaaaaaaaaaaaaaa\n")
f.write("bbbbbbbbbbbbbb");

# work with files
#open file for read
f=open('c:/TEMP/workpy.txt','r')
# line reading
s=f.readline()
print s

f.close()

# work with files
#open file for read
f=open('c:/TEMP/workpy.txt','r')

# pieces reading
s1=f.read(5)
print s1
```

```
s2=f.read(19)
print s2
s2=f.read(25)
print s2
f.close()


# work with files
#open file for read
f=open('c:/TEMP/workpy.txt','r')

# pieces reading
s1=f.read(5)
print s1
print f.tell()

s2=f.read(19)
print s2
print f.tell()

s2=f.read(25)
print s2
print f.tell()

f.close()


# work with files
# seek
f=open('c:/TEMP/workpy.txt','r+')
f.write('0123456789abcdef')
f.seek(5)     # Go to the 6th byte in the file
print f.read(1)
f.seek(-3, 2) # Go to the 3rd byte before the end
print f.read(1)

#! python
# The glob module provides a function for making file lists from
# directory wildcard searches:
import glob
s=glob.glob('*.*')
print s  #  ['primes.py', 'random.py', 'quote.py']
```