PREVIOUS ARTICLE
SAP KM Interview Questions and Answers

NEXT ARTICLE
C Interview Questions and Answers

Python Real Time Interview Questions and Answers

BY SONIA

August 24, 2017 in **INFORMATION TECHNOLOGIES (IT)** 1 Comment 9676

Q1).What is Python?

Ans1: Python is a high-level, interpreted, interactive and object-oriented scripting language. Python is designed to be highly readable. It uses English keywords frequently where as other languages use punctuation, and it h
as fewer syntactical constructions than other languages.

Q2).Name some of the features of Python.

Ans2: Following are some of the salient features of python

- It supports functional and structured programming methods as well as OOP.
- It can be used as a scripting language or can be compiled to byte-code for building large applications.
- It provides very high-level dynamic data types and supports dynamic type checking.
- It supports automatic garbage collection.
- It can be easily integrated with C, C++, COM, ActiveX, CORBA, and Java.

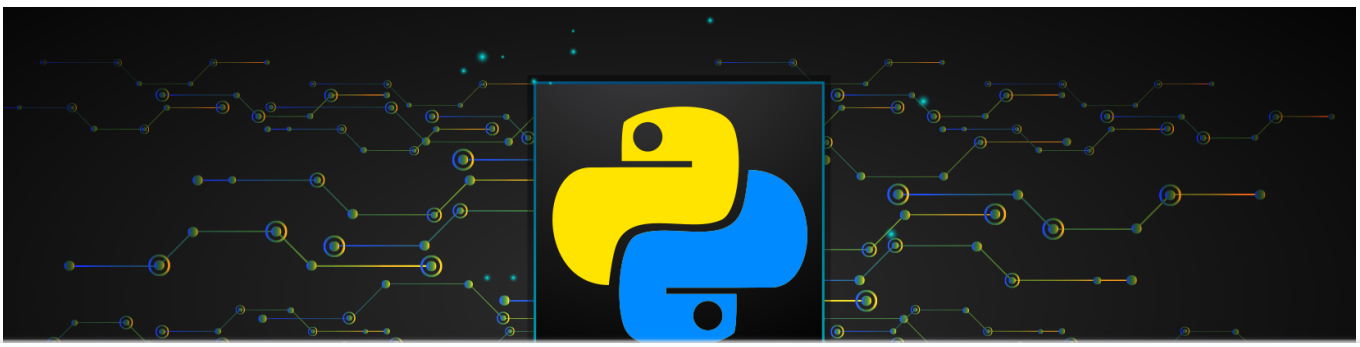
up to date. If you work on personal projects and code outside of the workplace then employers are more likely to see you as an asset that will grow. Even if they don't ask this question I find it's useful to broach the subject.

Q4).Is python a case sensitive language?
Ans4: Yes! Python is a case sensitive programming language.

What are the supported data types in Python?
Python has five standard data types –

- Numbers
- String
- List
- Tuple
- Dictionary

Q5).What is the output of print str if str = 'Hello World!'?
Ans5: It will print complete string. Output would be Hello World!.
Q6).What is the output of print str[0] if str = 'Hello World!'?
Ans6: It will print first character of the string. Output would be H.
Q7).What is the output of print str[2:5] if str = 'Hello World!'?
Ans7: It will print characters starting from 3rd to 5th. Output would be llo.
Q8).What is the output of print str[2:] if str = 'Hello World!'?
Ans8: It will print characters starting from 3rd character. Output would be llo World!.
Q9).What is the output of print str * 2 if str = 'Hello World!'?
Ans9: It will print string two times. Output would be Hello World!Hello World!.
Q10).What is the output of print str + "TEST" if str = 'Hello World!'?
Ans10: It will print concatenated string. Output would be Hello World!TEST.



Q11).What is the output of print list if list = ['abcd', 786 , 2.23, 'john', 70.2]?

Ans11: It will print concatenated lists. Output would be ['abcd', 786 , 2.23, 'john', 70.2].

Q12).What is the output of print list[0] if list = ['abcd', 786 , 2.23, 'john', 70.2]?

Ans12: It will print first element of the list. Output would be abcd.

Q13).What is the output of print list[1:3] if list = ['abcd', 786 , 2.23, 'john', 70.2]?

Ans13: It will print elements starting from 2nd till 3rd. Output would be [786, 2.23].

Q14).What is the output of print list[2:] if list = ['abcd', 786 , 2.23, 'john', 70.2]?

Ans14: It will print elements starting from 3rd element. Output would be [2.23, 'john', 70.200000000000003].

Q15).What is the output of print tinylist * 2 if tinylist = [123, 'john']?

Ans15: It will print list two times. Output would be [123, 'john', 123, 'john'].

Q16).What is the output of print list + tinylist * 2 if list = ['abcd', 786 , 2.23, 'john', 70.2] and tinylist = [123, 'john']?

Ans16: It will print concatenated lists. Output would be ['abcd', 786, 2.23, 'john', 70.2, 123, 'john', 123, 'john'].

Q17).What is tuples in Python?

Ans17: A tuple is another sequence data type that is similar to the list. A tuple consists of a number of values separated by commas. Unlike lists, however, tuples are enclosed within parentheses.

Q18).What is the difference between tuples and lists in Python?

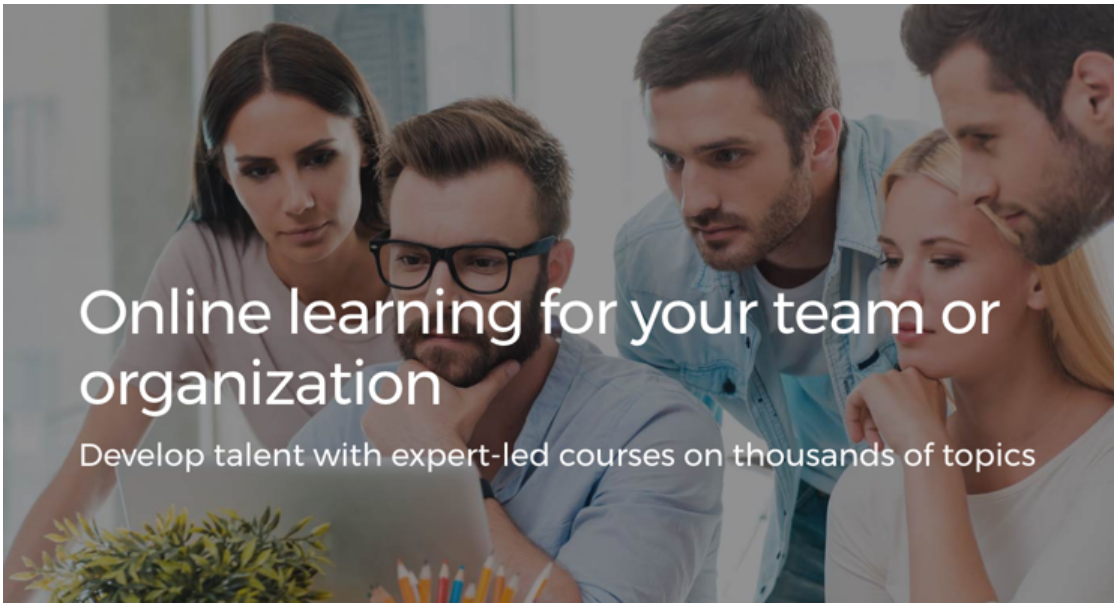
Ans18: The main differences between lists and tuples are – Lists are enclosed in brackets ([ ]) and their elements and size can be changed, while tuples are enclosed in parentheses (( )) and cannot be updated. Tuples can be thought of as read-only lists.

Q19).What is the output of print tuple if tuple = ('abcd', 786 , 2.23, 'john', 70.2)?

Ans19: It will print complete tuple. Output would be ('abcd', 786, 2.23, 'john', 70.200000000000003).

Q20).What is the output of print tuple[0] if tuple = ('abcd', 786 , 2.23, 'john', 70.2)?

Ans20: It will print first element of the tuple. Output would be abcd.



Q21).What is the output of print tuple[1:3] if tuple = ('abcd', 786 , 2.23, 'john', 70.2)?

Ans21: It will print elements starting from 2nd till 3rd. Output would be (786, 2.23).

Q22).What is the output of print tuple[2:] if tuple = ('abcd', 786 , 2.23, 'john', 70.2)?

Ans22: It will print elements starting from 3rd element. Output would be (2.23, 'john', 70.200000000000003).

Q23).What is the output of print tinytuple * 2 if tinytuple = (123, 'john')?

Ans23: It will print tuple two times. Output would be (123, 'john', 123, 'john').

Q24).What is the output of print tuple + tinytuple if tuple = ('abcd', 786 , 2.23, 'john', 70.2) and tinytuple = (123, 'john')?

Ans24: It will print concatenated tuples. Output would be ('abcd', 786, 2.23, 'john', 70.200000000000003, 123, 'john').

Q25).What are Python's dictionaries?

Ans25: Python's dictionaries are kind of hash table type. They work like associative arrays or hashes found in Perl and consist of key-value pairs. A dictionary key can be almost any Python type, but are usually numbers or strings. Values, on the other hand, can be any arbitrary Python object.

Q26).How will you create a dictionary in python?

Ans26: Dictionaries are enclosed by curly braces ({ }) and values can be assigned and accessed using square braces ([]).

dict = {}

dict['one'] = "This is one"

dict[2] = "This is two"

print dict.keys() # Prints all the keys

Q28).How will you get all the values from the dictionary?

Ans28: Using dictionary.values() function, we can get all the values from the dictionary object.

print dict.values() # Prints all the values

Q29).How will you convert a string to an int in python?

Ans29: int(x [,base]) – Converts x to an integer. base specifies the base if x is a string.

Q30).How will you convert a string to a long in python?

Ans30: long(x [,base]) – Converts x to a long integer. base specifies the base if x is a string.

Q31).How will you convert a string to a float in python?

Ans31: float(x) – Converts x to a floating-point number.

Q32).How will you convert a object to a string in python?

Ans32: str(x) – Converts object x to a string representation.

Q33).How will you convert a object to a regular expression in python?

Ans33: repr(x) – Converts object x to an expression string.

Q34).How will you convert a String to an object in python?

Ans34: eval(str) – Evaluates a string and returns an object.

Q35).How will you convert a string to a tuple in python?

Ans35: tuple(s) – Converts s to a tuple.

Q36).How will you convert a string to a list in python?

Ans36: list(s) – Converts s to a list.

Q38).How will you create a dictionary using tuples in python?

Ans38: dict(d) – Creates a dictionary. d must be a sequence of (key,value) tuples.

Q39).How will you convert a string to a frozen set in python?

Ans39: frozenset(s) – Converts s to a frozen set.

Q40).How will you convert an integer to a character in python?

Ans40: chr(x) – Converts an integer to a character.

Q41).How will you convert an integer to an unicode character in python?

Ans41: unichr(x) – Converts an integer to a Unicode character.

Q42).How will you convert a single character to its integer value in python?

Ans42: ord(x) – Converts a single character to its integer value.

Q43).How will you convert an integer to hexadecimal string in python?

Ans43: hex(x) – Converts an integer to a hexadecimal string.

Q44).How will you convert an integer to octal string in python?

Ans44: oct(x) – Converts an integer to an octal string.

Q45).What is the purpose of ** operator?

Ans45: ** Exponent – Performs exponential (power) calculation on operators. a**b = 10 to the power 20 if a = 10 and b = 20.

Q46).What is the purpose of // operator?

Ans46: // Floor Division – The division of operands where the result is the quotient in which the digits after the decimal point are removed.

Q48).What is the purpose of not in operator?

Ans48: not in – Evaluates to true if it does not finds a variable in the specified sequence and false otherwise. x not in y, here not in results in a 1 if x is not a member of sequence y.

Q49).What is the purpose break statement in python?

Ans49: break statement – Terminates the loop statement and transfers execution to the statement immediately following the loop.

Q50).What is the purpose continue statement in python?

Ans50: Continue statement – Causes the loop to skip the remainder of its body and immediately retest its condition prior to reiterating.

Q51).What is the purpose pass statement in python?

Ans51:pass statement – The pass statement in Python is used when a statement is required syntactically but you do not want any command or code to execute.

Q52).How can you pick a random item from a list or tuple?

Ans52: choice(seq) – Returns a random item from a list, tuple, or string.

Q53).How can you pick a random item from a range?

Ans53: randrange ([start,] stop [,step]) – returns a randomly selected element from range(start, stop, step).

Q54).How can you get a random number in python?

Ans54: random() – returns a random float r, such that 0 is less than or equal to r and r is less than 1.

Q55).How will you set the starting value in generating random numbers?

Ans55: seed([x]) – Sets the integer starting value used in generating random numbers. Call this function before calling any other random module function. Returns None.

Q57).How will you capitalizes first letter of string?

Ans57: capitalize() – Capitalizes first letter of string.

Q58).How will you check in a string that all characters are alphanumeric?

Ans58: isalnum() – Returns true if string has at least 1 character and all characters are alphanumeric and false otherwise.

Q59).How will you check in a string that all characters are digits?

Ans59: isdigit() – Returns true if string contains only digits and false otherwise.

Q60).How will you check in a string that all characters are in lowercase?

Ans60: islower() – Returns true if string has at least 1 cased character and all cased characters are in lowercase and false otherwise.

Q61).How will you check in a string that all characters are numerics?

Ans61: isnumeric() – Returns true if a unicode string contains only numeric characters and false otherwise.

Q62).How will you check in a string that all characters are whitespaces?

Ans62:isspace() – Returns true if string contains only whitespace characters and false otherwise.

Q63).How will you check in a string that it is properly titlecased?

Ans63: istitle() – Returns true if string is properly “titlecased” and false otherwise.

Q64).How will you check in a string that all characters are in uppercase?

Ans64: isupper() – Returns true if string has at least one cased character and all cased characters are in uppercase and false otherwise.

Q65).How will you merge elements in a sequence?

Ans65: join(seq) – Merges (concatenates) the string representations of elements in sequence seq into a string, with separator string.

Q67).How will you get a space-padded string with the original string left-justified to a total of width columns?

Ans67: just(width[, fillchar]) – Returns a space-padded string with the original string left-justified to a total of width columns.

Q68).How will you convert a string to all lowercase?

Ans68: lower() – Converts all uppercase letters in string to lowercase.

Q69).How will you remove all leading whitespace in string?

Ans69: strip() – Removes all leading whitespace in string.

Q70).How will you get the max alphabetical character from the string?

Ans70: max(str) – Returns the max alphabetical character from the string str.

0Q71).How will you get the min alphabetical character from the string?

Ans71: min(str) – Returns the min alphabetical character from the string str.

Q72).How will you replaces all occurrences of old substring in string with new string?

Ans72: replace(old, new [, max]) – Replaces all occurrences of old in string with new or at most max occurrences if max given.

Q73).How will you remove all leading and trailing whitespace in string?

Ans73:strip([chars]) – Performs both lstrip() and rstrip() on string.

Q74).How will you change case for all letters in string?

Ans74: swapcase() – Inverts case for all letters in string.

Q75).How will you get titlecased version of string?

Ans75: title() – Returns "titlecased" version of string, that is, all words begin with uppercase and the rest

Q76).How will you convert a string to all uppercase?

Ans76:upper() – Converts all lowercase letters in string to uppercase.

Q77).How will you check in a string that all characters are decimal?

Ans77: isdecimal() – Returns true if a unicode string contains only decimal characters and false otherwise.

Q78).What is the difference between del() and remove() methods of list?

Ans78: To remove a list element, you can use either the del statement if you know exactly which element(s) you are deleting or the remove() method if you do not know.

Q79).What is the output of len([1, 2, 3])?

Ans79: 3.

Q80).What is the output of [1, 2, 3] + [4, 5, 6]?

Ans80: [1, 2, 3, 4, 5, 6]

Q81).What is the output of ['Hi!'] * 4?

Ans81: ['Hi!', 'Hi!', 'Hi!', 'Hi!']

Q82).What is the output of 3 in [1, 2, 3]?

Ans82: True

Q83).What is the output of for x in [1, 2, 3]: print x?

Ans83: 1 2 3

Q84).What is the output of L[2] if L = [1,2,3]?

Ans84: 3, Offsets start at zero.

Q85).What is the output of L[-2] if L = [1,2,3]?

Ans85: L[-1] = 3, L[-2]=2, L[-3]=1

Q87).How will you compare two lists?

Ans87: cmp(list1, list2) – Compares elements of both lists.

Q88).How will you get the length of a list?

Ans88: len(list) – Gives the total length of the list.

Q89).How will you get the max valued item of a list?

Ans89: max(list) – Returns item from the list with max value.

Q90).How will you get the min valued item of a list?

Ans90: min(list) – Returns item from the list with min value.

Q91).How will you get the index of an object in a list?

Ans91: list.index(obj) – Returns the lowest index in list that obj appears.

Q92).How will you insert an object at given index in a list?

Ans92: list.insert(index, obj) – Inserts object obj into list at offset index.

Q93).How will you remove last object from a list?

Ans93: list.pop(obj=list[-1]) – Removes and returns last object or obj from list.

Q94).How will you remove an object from a list?

Ans94: list.remove(obj) – Removes object obj from list.

Q95).How will you reverse a list?

Ans95: list.reverse() – Reverses objects of list in place.

Q96).How will you sort a list?

Ans96: list.sort([func]) – Sorts objects of list, use compare func if given

Ans97:

datetime (used to manipulate date and time)
re (regular expressions)
urllib, urllib2 (handles many HTTP things)
string (a collection of different groups of strings for example all lower_case letters etc)
itertools (permutations, combinations and other useful iterables)
ctypes (from python docs: create and manipulate C data types in Python)
email (from python docs: A package for parsing, handling, and generating email messages)
__future__ (Record of incompatible language changes. like division operator is different and much better when imported from __future__)
sqlite3 (handles database of SQLite type)
unittest (from python docs: Python unit testing framework, based on Erich Gamma's JUnit and Kent Beck's Smalltalk testing framework)
xml (xml support)
logging (defines logger classes. enables python to log details on severity level basis)
os (operating system support)
pickle (similar to json. can put any data structure to external files)
subprocess (from docs: This module allows you to spawn processes, connect to their input/output/error pipes, and obtain their return codes)
webbrowser (from docs: Interfaces for launching and remotely controlling Web browsers.)
traceback (Extract, format and print Python stack traces)

Q98).Name a module that is not included in python by default

Ans98:mechanize
django
gtk
A lot of other can be found at pypi.

Q99).What is __init__.py used for?

Ans99:It declares that the given directory is a package. #Python Docs (From Endophage's comment)

Q100).When is pass used for?

Ans100:pass does nothing. It is used for completing the code where we need something. For eg:

```
1  class abc():
2  pass
```

Q101).What is a docstring?

Ans101: docstring is the documentation string for a function. It can be accessed by

function_name.__doc__

it is declared as:

```
1  def function_name():
2  """your docstring"""
```

Writing documentation for your progams is a good habit and makes the code more understandable and reusable.

Q102).What is list comprehension?

Ans102: Creating a list by doing some operation over data that can be accessed using an iterator. For eg:

```
1  >>>[ord(i) for i in string.ascii_uppercase]
2  [65, 66, 67, 68, 69, 70, 71, 72, 73, 74, 75, 76, 77, 78, 79, 80, 8
3  >>>
```

Q103).What is map?

Ans103: map executes the function given as the first argument on all the elements of the iterable given as

```
1
2
3
4
5
```

```
>>>a='ayush'

>>>map(ord,a)

....  [97, 121, 117, 115, 104]

>>> print map(lambda x, y: x*y**2, [1, 2, 3],
[2, 4, 1])

....  [4, 32, 3]
```

```
1
2
3
4
5
6
7
8
9
10
11
```

```
Help on built-in function map in module
__builtin__:



map(...)

map(function, sequence[, sequence, ...]) -> list



Return a list of the results of applying the
function to the items of

the argument sequence(s).  If more than one
sequence is given, the

function is called with an argument list
consisting of the corresponding

item of each sequence, substituting None for
missing values when not all

sequences have the same length.  If the function
is None, return a list of

the items of the sequence (or a list of tuples if
more than one sequence).
```

Q104).What is the difference between a tuple and a list?

Ans104:A tuple is immutable i.e. can not be changed. It can be operated on only. But a list is mutable. Changes can be done internally to it.

tuple initialization: a = (2,4,5)
list initialization: a = [2,4,5]

The methods/functions provided with each types are also different. Check them out yourself.

Q105).Using various python modules convert the list a to generate the output 'one, two, three'

Ans105:

1 2	a = ['one', 'two', 'three'] Ans: ", ".join(a)

1 2 3 4 5 6	>>>help(str.join) Help on method_descriptor: join(...) S.join(iterable) -> string Return a string which is the concatenation of the strings in the iterable. The separator between elements is S.

Q106).What would the following code yield?

1 2	word = 'abcdefghij' print word[:3] + word[3:]

and 'defghij'. The '+' operator on strings concatenates them. Thus, the two slices formed are concatenated to give the answer 'abcdefghij'.

Q107).Optimize these statements as a python programmer.

```
1  word = 'word'
2  print word.__len__()
```

Ans107:

```
1  word = 'word'
2  print len(word)
```

Q108).Write a program to print all the contents of a file

Ans108:

```
1  try:
2  with open('filename','r') as f:
3  print f.read()
4  except IOError:
5  print "no such file exists"
```

Q109). What will be the output of the following code

```
1  a = 1
2  a, b = a+1, a+1
```

Ans109:

2

2

The second line is a simultaneous declaration i.e. value of new a is not used when doing b=a+1.

This is why, exchanging numbers is as easy as:

```
1    a,b = b,a
```

Q110). Given the list below remove the repetition of an element.

All the elements should be unique

words = ['one', 'one', 'two', 'three', 'three', 'two']

Ans110:

A bad solution would be to iterate over the list and checking for copies somehow and then remove them!

One of the best solutions I can think of right now:

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

2    list(set(a))
```

set is another type available in python, where copies are not allowed. It also has some good functions available used in set operations (like union, difference).

Q111).Iterate over a list of words and use a dictionary to keep track of the frequency(count) of each word. for example

```
>>> def dic(words):
    a = {}
    for i in words:
        try:
            a[i] += 1
        except KeyError: ## the famous pythonic way:
            a[i] = 1       ## Halt and catch fire
    return a

>>> a='1,3,2,4,5,3,2,1,4,3,2'.split(',')
>>> a
['1', '3', '2', '4', '5', '3', '2', '1', '4', '3', '2']
>>> dic(a)
{'1': 2, '3': 3, '2': 3, '5': 1, '4': 2}
```

Without using try-catch block:

```
>>> def dic(words):
    data = {}
    for i in words:
        data[i] = data.get(i, 0) + 1
```

```
7   >>> a
8   ['1', '3', '2', '4', '5', '3', '2', '1', '4', '3', '2']
9   >>> dic(a)
10  {'1': 2, '3': 3, '2': 3, '5': 1, '4': 2}
```

PS: Since the collections module (which gives you the defaultdict) is written in python, I would not recommend using it. The normal dict implementation is in C, it should be much faster. You can use timeit module to check for comparing the two.
So, David and I have saved you the work to check it. Check the files on github. Change the data file to test different data.

Q112).Write the following logic in Python:
If a list of words is empty, then let the user know it's empty, otherwise let the user know it's not empty.

Ans112: Can be checked by a single statement (pythonic beauty):

```
1  print "The list is empty" if len(a)==0 else "The list is not empty"
2
3  >>> a=''
4  >>> print "'The list is
5  empty'" if len(a)==0 else "'The list is not empty'"
6  'The list is empty'
7  >>> a='asd'
8  >>> print "'The list is
   empty'" if len(a)==0 else "'The list is not empty'"
   'The list is not empty'
```

Ans113:

```
1  try:
2  import mechanize as me
3  except ImportError:
4  import urllib as me
```

here you have atleast 1 module imported as me.
This is used to check if the users computer has third party libraries that we need. If not, we work with a default library of python. Quite useful in updating softwares.
PS: This is just one of the uses of try-except blocks. You can note a good use of these in API's.
Also note that if we do not define the error to be matched, the except block would catch any error raised in try block.

Q114).Print the length of each line in the file 'file.txt' not including any whitespaces at the end of the lines.

Ans114:

```
1  with open("filename.txt", "r") as f1:
2  print len(f1.readline().rstrip())
```

rstrip() is an inbuilt function which strips the string from the right end of spaces or tabs (whitespace characters).

Q115). Print the sum of digits of numbers starting from 1 to 100 (inclusive of both)

Ans115:

```
1  print sum(range(1,101))
```

```
1  print sum(xrange(1, 101))
```

xrange() returns an iterator rather than a list which is less heavy on the memory.

Q116).Create a new list that converts the following list of number strings to a list of numbers.

num_strings = ['1','21','53','84','50','66','7','38','9']

Ans116:
use a list comprehension

```
1  >>> [int(i) for i in num_strings]
2  [1, 21, 53, 84, 50, 66, 7, 38, 9]
```

#num_strings should not contain any non-integer character else ValueError would be raised. A try-catch block can be used to notify the user of this.

Another one suggested by David using maps:

```
1  >>> map(int, num_strings)
2  [1, 21, 53, 84, 50, 66, 7, 38, 9]
```

Q117).Create two new lists one with odd numbers and other with even numbers
num_strings = [1,21,53,84,50,66,7,38,9]

Ans117:

```
1  >>> odd=[]
2  >>> even=[]
3  >>> for i in n:
4  even.append(i) if i%2==0 else odd.append(i)
```

```
7    ## all even numbers in list even
```

Though if only one of the lists were requires, using list comprehension we could make:

```
1    even = [i for i in num_strings if i%2==0]
2    odd = [i for i in num_strings if i%2==1]
```

But using this approach if both lists are required would not be efficient since this would iterate the list two times.!

Q118).Write a program to sort the following intergers in list

nums = [1,5,2,10,3,45,23,1,4,7,9]

Ans118:nums.sort() # The lists have an inbuilt function, sort()
sorted(nums) # sorted() is one of the inbuilt functions)

Python uses TimSort for applying this function. Check the link to know more.

Q119).Write a for loop that prints all elements of a list and their position in the list.
Printing using String formatting

Ans119:

```
1    >>> for index, data in enumerate(asd):
2    ....    print “{0} -> {1}”.format(index, data)
3
4    0 -> 4
5    1 -> 7
6    2 -> 3
```

9	5 -> 9

#OR

```
1   >>> asd = [4,7,3,2,5,9]
2
3   >>> for i in range(len(asd)):
4   ....    print i+1,'–>',asd[i]
5
6   1 –> 4
7   2 –> 7
8   3 –> 3
9   4 –> 2
10  5 –> 5
11  6 –> 9
```

Q120).The following code is supposed to remove numbers less than 5 from list n, but there is a bug. Fix the bug.

Ans120:

```
1   n = [1,2,5,10,3,100,9,24]
2
3   for e in n:
```

```
6    print n
```

after e is removed, the index position gets disturbed. Instead it should be:

```
1    a=[]
2    for e in n:
3    if e >= 5:
4    a.append(e)
5    n = a
```

OR again a list comprehension:

```
1    return [i for i in n if i >= 5]
```

OR use filter

```
1    return filter(lambda x: x >= 5, n)
```


Q121).What will be the output of the following

3	
4	func(1,2,3)

Ans121: Here the output is :

{} #Empty Dictionay

x is a normal value, so it takes 1..
y is a list of numbers, so it takes 2,3..
z wants named parameters, so it can not take any value here.
Thus the given answer.

Q122).Write a program to swap two numbers.

Ans122:

a = 5
b = 9

as i told earlier too, just use:
a,b = b,a

Q123).What will be the output of the following code

1	class C(object):
2	…. def__init__(self):
3	…. self.x =1
4	
5	c=C()
6	print c.x
7	print c.x
8	c.x=2

Ans123: All the outputs will be 1, since the value of the the object's attribute(x) is never changed.

1
1
1
1

x is now a part of the public members of the class C.
Thus it can be accessed directly..

Q124). What is wrong with the code

```
1  func([1,2,3]) # explicitly passing in a list
2  func()        # using a default empty list
3
4  def func(n = []):
5  #do something with n
6
7  print n
```

Ans124. This would result in a NameError. The variable n is local to function func and can't be accessesd outside. So, printing it won't be possible.

Edit: An extra point for interviews given by Shane Green and Peter: """Another thing is that mutable types should never be used as default parameter values. Default parameter value expressions are only evaluated once, meaning every invocation of that method shares the same default value. If one invocation that ends up using the default value modifies that value–a list, in this case–it will forever be modified for all future invocations. So default parameter values should limited to primitives, strings, and tuples; no lists, dictionaries, or complex object instances."""
Reference: Default argument values

1. n = 1
   print n++ ## no such operator in python (++)
2. n = 1
   print ++n ## no such operator in python (++)
3. n = 1
   print n += 1 ## will work
4. int n = 1
   print n = n+1 ##will not work as assignment can not be done in print command like this
5. n =1
   n = n+1 ## will work

Q126).In Python function parameters are passed by value or by reference?

Ans126: It is somewhat more complicated than I have written here (Thanks David for pointing). Explaining all here won't be possible. Some good links that would really make you understand how things are:

Stackoverflow

Python memory management

Viewing the memory

Q127).Remove the whitespaces from the string.

s = 'aaa bbb ccc ddd eee'

Ans127:

```
1  ''.join(s.split())
2  ## join without spaces the string after splitting it
```

OR

```
1  filter(lambda x: x != ' ', s)
```

s = a + '[' + b + ':' + c + ']'

Ans128: seems like a string is being concatenated. Nothing much can be said without knowing types of variables a, b, c. Also, if all of the a, b, c are not of type string, TypeError would be raised. This is because of the string constants ('[' , ']') used in the statement.

Q129). Optimize the below code

```
1  def append_s(words):
2      new_words=[]
3      for word in words:
4          new_words.append(word + 's')
5      return new_words
6
7  for word in append_s(['a','b','c']):
8      print word
```

Ans129: The above code adds a trailing s after each element of the list.

```
def append_s(words):
return [i+'s' for i in words] ## another list comprehension

for word in append_s(['a','b','c']):
print word
```

Q130).If given the first and last names of bunch of employees how would you store it and what datatype?

Ans130: best stored in a list of dictionaries..
dictionary format: {'first_name':'Ayush','last_name':'Goel'}

Q131). What is Python really? You can (and are encouraged) make comparisons to other technologies in your answer

Ans131: Here are a few key points:

- Python is an interpreted language. That means that, unlike languages like *C*and its variants, Python does not need to be compiled before it is run. Other interpreted languages include *PHP* and *Ruby*.
- Python is dynamically typed, this means that you don't need to state the types of variables when you declare them or anything like that. You can do things like x=111and then x="I'm a string"without error
- Python is well suited to object orientated programming in that it allows the definition of classes along with composition and inheritance. Python does not have access specifiers (like C++'s public, private), the justification for this point is given as "we are all adults here"
- In Python, functions are first-class objects. This means that they can be assigned to variables, returned from other functions and passed into functions. Classes are also first class objects
- Writing Python code is quick but running it is often slower than compiled languages. Fortunately， Python allows the inclusion of C based extensions so bottlenecks can be optimised away and often are.
The numpypackage is a good example of this, it's really quite quick because a lot of the number crunching it does isn't actually done by Python
- Python finds use in many spheres – web applications, automation, scientific modelling, big data applications and many more. It's also often used as "glue" code to get other languages and components to play nice.
- Python makes difficult things easy so programmers can focus on overriding algorithms and structures rather than nitty-gritty low level details.

Why This Matters:

If you are applying for a Python position, you should know what it is and why it is so gosh-darn cool. And why it isn't o.O

"""

This function takes the name of a directory

and prints out the paths files within that

directory as well as any files contained in

contained directories.

This function is similar to os.walk. Please don't

use os.walk in your answer. We are interested in your

ability to work with nested structures.

"""

fill_this_in

Ans132: def print_directory_contents(sPath):

import os

for sChild **in** os.listdir(sPath):

sChildPath = os.path.join(sPath,sChild)

if os.path.isdir(sChildPath):

print_directory_contents(sChildPath)

else:

print(sChildPath)

- Recursive functions need to recurse *and* Make sure you understand how this happens so that you avoid bottomless callstacks
- We use the osmodule for interacting with the operating system in a way that is cross platform. You could say sChildPath = sPath + '/' + sChild but that wouldn't work on windows
- Familiarity with base packages is really worthwhile, but don't break your head trying to memorize everything, Google is your friend in the workplace!
- Ask questions if you don't understand what the code is supposed to do
- KISS! Keep it Simple, Stupid!

Why This Matters:

- Displays knowledge of basic operating system interaction stuff
- Recursion is hella useful

Q133).

Looking at the below code, write down the final values of A0, A1, ...An.

A0 = dict(zip(('a','b','c','d','e'),(1,2,3,4,5)))

A1 = range(10)

A2 = sorted([i **for** i **in** A1 **if** i **in** A0])

A3 = sorted([A0[s] **for** s **in** A0])

A4 = [i **for** i **in** A1 **if** i **in** A3]

A5 = {i:i*i **for** i **in** A1}

A6 = [[i,i*i] **for** i **in** A1]

If you dont know what zip is don't stress out. No sane employer will expect you to memorize the standard library. Here is the output of help(zip).

zip(...)

Return **a** list **of** tuples, where **each** tuple **contains the** i-th **element**

from **each of the** argument sequences. The returned list is truncated

in length to **the** length **of the** shortest argument sequence.

If that doesn't make sense then take a few minutes to figure it out however you choose to.

Ans133: A0 = {'a': 1, 'c': 3, 'b': 2, 'e': 5, 'd': 4} *# the order may vary*

A1 = range(0, 10) *# or [0, 1, 2, 3, 4, 5, 6, 7, 8, 9] in python 2*

A2 = []

A3 = [1, 2, 3, 4, 5]

A4 = [1, 2, 3, 4, 5]

A5 = {0: 0, 1: 1, 2: 4, 3: 9, 4: 16, 5: 25, 6: 36, 7: 49, 8: 64, 9: 81}

A6 = [[0, 0], [1, 1], [2, 4], [3, 9], [4, 16], [5, 25], [6, 36], [7, 49], [8, 64], [9, 81]]

Why This Matters

1. List comprehension is a wonderful time saver and a big stumbling block for a lot of people
2. If you can read them, you can probably write them down
3. Some of this code was made to be deliberately weird. You may need to work with some weird people

Q134). Python and multi-threading. Is it a good idea? List some ways to get some Python code to run in a parallel way.

Ans134: Python doesn't allow multi-threading in the truest sense of the word. It has a multi-threading package but if you want to multi-thread to speed your code up, then it's usually not a good idea to use it. Python has a construct called the Global Interpreter Lock (GIL). The GIL makes sure that only one of your 'threads' can execute at any one time. A thread acquires the GIL, does a little work, then passes the GIL onto the next thread. This happens very quickly so to the human eye it may seem like your threads are executing in parallel, but they are really just taking turns using the same CPU core. All this GIL passing adds

There are reasons to use Python's threading package. If you want to run some things simultaneously, and efficiency is not a concern, then it's totally fine and convenient. Or if you are running code that needs to wait for something (like some IO) then it could make a lot of sense. But the threading library won't let you use extra CPU cores.

Multi-threading can be outsourced to the operating system (by doing multi-processing), some external application that calls your Python code (eg, Spark or Hadoop), or some code that your Python code calls (eg: you could have your Python code call a C function that does the expensive multi-threaded stuff).

Why This Matters

Because the GIL is an A-hole. Lots of people spend a lot of time trying to find bottlenecks in their fancy Python multi-threaded code before they learn what the GIL is.

Q135). How do you keep track of different versions of your code?

Ans135: Version control! At this point, you should act excited and tell them how you even use Git (or whatever is your favorite) to keep track of correspondence with Granny. Git is my preferred version control system, but there are others, for example subversion.

Why This Matters:

Because code without version control is like coffee without a cup. Sometimes we need to write once-off throw away scripts and that's ok, but if you are dealing with any significant amount of code, a version control system will be a benefit. Version Control helps with keeping track of who made what change to the code base; finding out when bugs were introduced to the code; keeping track of versions and releases of your software; distributing the source code amongst team members; deployment and certain automations. It allows you to roll your code back to before you broke it which is great on its own. Lots of stuff. It's just great.

Q136). What does this code output:

```
def f(x,l=[]):

for i in range(x):
```

f(2)

f(3,[3,2,1])

f(3)

Ans136:

[0, 1]

[3, 2, 1, 0, 1, 4]

[0, 1, 0, 1, 4]

Hu?

The first function call should be fairly obvious, the loop appends 0 and then 1 to the empty list, l. l is a name for a variable that points to a list stored in memory.
The second call starts off by creating a new list in a new block of memory. l then refers to this new list. It then appends 0, 1 and 4 to this new list. So that's great.
The third function call is the weird one. It uses the original list stored in the original memory block. That is why it starts off with 0 and 1.

Try this out if you don't understand:

```
l_mem = []


l = l_mem           # the first call

for i in range(2):

l.append(i*i)


print(l)           # [0, 1]
```

```
for i in range(3):

l.append(i*i)



print(l)          # [3, 2, 1, 0, 1, 4]



l = l_mem         # the third call

for i in range(3):

l.append(i*i)



print(l)          # [0, 1, 0, 1, 4]
```

Q137). What is monkey patching and is it ever a good idea?

Ans137: Monkey patching is changing the behaviour of a function or object after it has already been defined. For example:

```
import datetime

datetime.datetime.now = lambda: datetime.datetime(2012, 12, 12)
```

Most of the time it's a pretty terrible idea – it is usually best if things act in a well-defined way. One reason to monkey patch would be in testing. The mock package is very useful to this end.

Why This Matters

It shows that you understand a bit about methodologies in unit testing. Your mention of monkey avoidance will show that you aren't one of those coders who favor fancy code over maintainable code (they are out there, and they suck to work with). Remember the principle of KISS? And it shows that you know a little bit about how Python works on a lower level, how functions are actually stored and called and

Q138). What does this stuff mean: *args, **kwargs? And why would we use it?

Ans138: Use *args when we aren't sure how many arguments are going to be passed to a function, or if we want to pass a stored list or tuple of arguments to a function. **kwargsis used when we dont know how many keyword arguments will be passed to a function, or it can be used to pass the values of a dictionary as keyword arguments. The identifiers args and kwargs are a convention, you could also use *bob and **billy but that would not be wise.

Here is a little illustration:

```
def f(*args,**kwargs): print(args, kwargs)

l = [1,2,3]

t = (4,5,6)

d = {'a':7,'b':8,'c':9}

f()

f(1,2,3)                    # (1, 2, 3) {}

f(1,2,3,"groovy")           # (1, 2, 3, 'groovy') {}

f(a=1,b=2,c=3)              # () {'a': 1, 'c': 3, 'b': 2}

f(a=1,b=2,c=3,zzz="hi")     # () {'a': 1, 'c': 3, 'b': 2, 'zzz': 'hi'}

f(1,2,3,a=1,b=2,c=3)        # (1, 2, 3) {'a': 1, 'c': 3, 'b': 2}
```

f(1,2,*t) *# (1, 2, 4, 5, 6) {}*

f(q="winning",**d) *# () {'a': 7, 'q': 'winning', 'c': 9, 'b': 8}*

f(1,2,*t,q="winning",**d) *# (1, 2, 4, 5, 6) {'a': 7, 'q': 'winning', 'c': 9, 'b': 8}*

def f2(arg1,arg2,*args,**kwargs): print(arg1,arg2, args, kwargs)

f2(1,2,3) *# 1 2 (3,) {}*

f2(1,2,3,"groovy") *# 1 2 (3, 'groovy') {}*

f2(arg1=1,arg2=2,c=3) *# 1 2 () {'c': 3}*

f2(arg1=1,arg2=2,c=3,zzz="hi") *# 1 2 () {'c': 3, 'zzz': 'hi'}*

f2(1,2,3,a=1,b=2,c=3) *# 1 2 (3,) {'a': 1, 'c': 3, 'b': 2}*

f2(*l,**d) *# 1 2 (3,) {'a': 7, 'c': 9, 'b': 8}*

f2(*t,**d) *# 4 5 (6,) {'a': 7, 'c': 9, 'b': 8}*

f2(1,2,*t) *# 1 2 (4, 5, 6) {}*

f2(1,1,q="winning",**d) *# 1 1 () {'a': 7, 'q': 'winning', 'c': 9, 'b': 8}*

f2(1,2,*t,q="winning",**d) *# 1 2 (4, 5, 6) {'a': 7, 'q': 'winning', 'c': 9, 'b': 8}*

Why Care?

Sometimes we will need to pass an unknown number of arguments or keyword arguments into a function. Sometimes we will want to store arguments or keyword arguments for later use. Sometimes it's just a time saver.

Ans139: Answer Background Knowledge:

These are decorators. A decorator is a special kind of function that either takes a function and returns a function, or takes a class and returns a class. The @ symbol is just syntactic sugar that allows you to decorate something in a way that's easy to read.

@my_decorator

def my_func(stuff):

do_things

Is equivalent to

def my_func(stuff):

do_things

my_func = my_decorator(my_func)

Click Here To learn More On Python

Actual Answer:

The decorators @classmethod, @staticmethod and @property are used on functions defined within classes. Here is how they behave:

class MyClass(object):

def __init__(self):

self._some_property = "properties are nice"

self._some_other_property = "VERY nice"

def normal_method(*args,**kwargs):

print("calling normal_method({0},{1})".format(args,kwargs))

```
print("calling class_method({0},{1})".format(args,kwargs))

    @staticmethod

def static_method(*args,**kwargs):

print("calling static_method({0},{1})".format(args,kwargs))

    @property

def some_property(self,*args,**kwargs):

print("calling some_property getter({0},{1},{2})".format(self,args,kwargs))

return self._some_property

    @some_property.setter

def some_property(self,*args,**kwargs):

print("calling some_property setter({0},{1},{2})".format(self,args,kwargs))

self._some_property = args[0]

    @property

def some_other_property(self,*args,**kwargs):

print("calling some_other_property getter({0},{1},{2})".format(self,args,kwargs))

return self._some_other_property



o = MyClass()

# undecorated methods work like normal, they get the current instance (self) as the first argument
```

```
o.normal_method()

# normal_method((<__main__.MyClass instance at 0x7fdd2537ea28>,),{})


o.normal_method(1,2,x=3,y=4)

# normal_method((<__main__.MyClass instance at 0x7fdd2537ea28>, 1, 2),{'y': 4, 'x': 3})


# class methods always get the class as the first argument


o.class_method

# <bound method classobj.class_method of <class __main__.MyClass at 0x7fdd2536a390>>


o.class_method()

# class_method((<class __main__.MyClass at 0x7fdd2536a390>,),{})


o.class_method(1,2,x=3,y=4)

# class_method((<class __main__.MyClass at 0x7fdd2536a390>, 1, 2),{'y': 4, 'x': 3})


# static methods have no arguments except the ones you pass in when you call them
```

```
o.static_method()

# static_method((),{})


o.static_method(1,2,x=3,y=4)

# static_method((1, 2),{'y': 4, 'x': 3})


# properties are a way of implementing getters and setters. It's an error to explicitly call them

# "read only" attributes can be specified by creating a getter without a setter (as in some_other_property)


o.some_property

# calling some_property getter(<__main__.MyClass instance at 0x7fb2b70877e8>,(),{})

# 'properties are nice'


o.some_property()

# calling some_property getter(<__main__.MyClass instance at 0x7fb2b70877e8>,(),{})

# Traceback (most recent call last):

#   File "<stdin>", line 1, in <module>

# TypeError: 'str' object is not callable
```

```
# 'VERY nice'


# o.some_other_property()

# calling some_other_property getter(<__main__.MyClass instance at 0x7fb2b70877e8>,(),{})

# Traceback (most recent call last):

#   File "<stdin>", line 1, in <module>

# TypeError: 'str' object is not callable


o.some_property = "groovy"

# calling some_property setter(<__main__.MyClass object at 0x7fb2b7077890>,('groovy',),{})


o.some_property

# calling some_property getter(<__main__.MyClass object at 0x7fb2b7077890>,(),{})

# 'groovy'


o.some_other_property = "very groovy"

# Traceback (most recent call last):

#   File "<stdin>", line 1, in <module>

# AttributeError: can't set attribute
```

 # 'VERY nice'

Q140). Consider the following code, what will it output?

class A(object):

def go(self):

print("go A go!")

def stop(self):

print("stop A stop!")

def pause(self):

raise Exception("Not Implemented")

class B(A):

def go(self):

super(B, self).go()

print("go B go!")

class C(A):

def go(self):

super(C, self).go()

print("go C go!")

def stop(self):

```
class D(B,C):

def go(self):

super(D, self).go()

print("go D go!")

def stop(self):

super(D, self).stop()

print("stop D stop!")

def pause(self):

print("wait D wait!")


class E(B,C): pass


a = A()

b = B()

c = C()

d = D()

e = E()


# specify output from here onwards
```

b.go()

c.go()

d.go()

e.go()

a.stop()

b.stop()

c.stop()

d.stop()

e.stop()

a.pause()

b.pause()

c.pause()

d.pause()

e.pause()

Ans140:

The output is specified in the comments in the segment below:

a.go()

 # go A go!

```
# go A go!

# go B go!


c.go()

# go A go!

# go C go!


d.go()

# go A go!

# go C go!

# go B go!

# go D go!


e.go()

# go A go!

# go C go!

# go B go!


a.stop()

# stop A stop!
```

```
# stop A stop!


c.stop()

# stop A stop!

# stop C stop!


d.stop()

# stop A stop!

# stop C stop!

# stop D stop!


e.stop()

# stop A stop!


a.pause()

# ... Exception: Not Implemented


b.pause()

# ... Exception: Not Implemented
```

d.pause()

 # wait D wait!

e.pause()

 # ...Exception: Not Implemented

Why do we care?

Because OO programming is really, really important. Really. Answering this question shows your understanding of inheritance and the use of Python's super function. Most of the time the order of resolution doesn't matter. Sometimes it does, it depends on your application.

Q141).

Consider the following code, what will it output?

class Node(object):

def __init__(self,sName):

self._lChildren = []

self.sName = sName

```
def append(self,*args,**kwargs):

self._lChildren.append(*args,**kwargs)

def print_all_1(self):

print(self)

for oChild in self._lChildren:

oChild.print_all_1()

def print_all_2(self):

def gen(o):

lAll = [o,]

while lAll:

oNext = lAll.pop(0)

lAll.extend(oNext._lChildren)

yield oNext

for oNode in gen(self):

print(oNode)



oRoot = Node("root")

oChild1 = Node("child1")

oChild2 = Node("child2")

oChild3 = Node("child3")
```

```
oChild6 = Node("child6")

oChild7 = Node("child7")

oChild8 = Node("child8")

oChild9 = Node("child9")

oChild10 = Node("child10")



oRoot.append(oChild1)

oRoot.append(oChild2)

oRoot.append(oChild3)

oChild1.append(oChild4)

oChild1.append(oChild5)

oChild2.append(oChild6)

oChild4.append(oChild7)

oChild3.append(oChild8)

oChild3.append(oChild9)

oChild6.append(oChild10)



# specify output from here onwards



oRoot.print_all_1()
```

oRoot.print_all_1() prints:

<Node ‘root’>

<Node ‘child1’>

<Node ‘child4’>

<Node ‘child7’>

<Node ‘child5’>

<Node ‘child2’>

<Node ‘child6’>

<Node ‘child10’>

<Node ‘child3’>

<Node ‘child8’>

<Node ‘child9’>

oRoot.print_all_2() prints:

<Node ‘root’>

<Node ‘child1’>

<Node ‘child2’>

<Node ‘child3’>

<Node ‘child4’>

<Node ‘child5’>

<Node ‘child6’>

<Node 'child7'>

<Node 'child10'>

Why do we care?

Because composition and object construction is what objects are all about. Objects are composed of stuff and they need to be initialised somehow. This also ties up some stuff about recursion and use of generators.

Generators are great. You could have achieved similar functionality to print_all_2 by just constructing a big long list and then printing it's contents. One of the nice things about generators is that they don't need to take up much space in memory.

It is also worth pointing out that print_all_1 traverses the tree in a depth-first manner, while print_all_2 is width-first. Make sure you understand those terms. Sometimes one kind of traversal is more appropriate than the other. But that depends very much on your application.

Q142). Describe Python's garbage collection mechanism in brief.

Ans142: A lot can be said here. There are a few main points that you should mention:

- Python maintains a count of the number of references to each object in memory. If a reference count goes to zero then the associated object is no longer live and the memory allocated to that object can be freed up for something else
- occasionally things called "reference cycles" happen. The garbage collector periodically looks for these and cleans them up. An example would be if you have two objects o1and o2 such that x == o2 and o2.x == o1.
If o1 and o2 are not referenced by anything else then they shouldn't be live. But each of them has a reference count of 1.
- Certain heuristics are used to speed up garbage collection. For example, recently created objects are more likely to be dead. As objects are created, the garbage collector assigns them to generations. Each object gets one generation, and younger generations are dealt with first.

This explanation is CPython specific.

Q143)

How would you prove that your answer is correct?

```
def f1(lIn):
l1 = sorted(lIn)
l2 = [i for i in l1 if i<0.5]
return [i*i for i in l2]

def f2(lIn):
l1 = [i for i in lIn if i<0.5]
l2 = sorted(l1)
return [i*i for i in l2]

def f3(lIn):
l1 = [i*i for i in lIn]
l2 = sorted(l1)
return [i for i in l1 if i<(0.5*0.5)]
```

Ans143:

Most to least efficient: f2, f1, f3. To prove that this is the case, you would want to profile your code. Python has a lovely profiling package that should do the trick.

```
import cProfile
lIn = [random.random() for i in range(100000)]
```

cProfile.run('f3(lln)')

For completion's sake, here is what the above profile outputs:

>>> cProfile.run('f1(lln)')

4 function calls **in** 0.045 seconds

Ordered by: standard name

ncalls tottime percall cumtime percall filename:lineno(function)

1 0.009 0.009 0.044 0.044 <stdin>:1(f1)

1 0.001 0.001 0.045 0.045 <string>:1(<module>)

1 0.000 0.000 0.000 0.000 {method 'disable' of '_lsprof.Profiler' objects}

1 0.035 0.035 0.035 0.035 {sorted}

>>> cProfile.run('f2(lln)')

4 function calls **in** 0.024 seconds

Ordered by: standard name

ncalls tottime percall cumtime percall filename:lineno(function)

1 0.000 0.000 0.000 0.000 {method 'disable' of '_lsprof.Profiler' objects}

1 0.016 0.016 0.016 0.016 {sorted}

>>> cProfile.run('f3(lln)')

4 function calls **in** 0.055 seconds

Ordered by: standard name

ncalls tottime percall cumtime percall filename:lineno(function)

1 0.016 0.016 0.054 0.054 <stdin>:1(f3)

1 0.001 0.001 0.055 0.055 <string>:1(<module>)

1 0.000 0.000 0.000 0.000 {method 'disable' of '_lsprof.Profiler' objects}

1 0.038 0.038 0.038 0.038 {sorted}

Why Care?

Locating and avoiding bottlenecks is often pretty worthwhile. A lot of coding for efficiency comes down to common sense – in the example above it's obviously quicker to sort a list if it's a smaller list, so if you have the choice of filtering before a sort it's often a good idea. The less obvious stuff can still be located through use of the proper tools. It's good to know about these tools.

Q144).What is the purpose of PYTHONPATH environment variable?

Ans144: PYTHONPATH – It has a role similar to PATH. This variable tells the Python interpreter where to locate the module files imported into a program. It should include the Python source library directory and the directories containing Python source code. PYTHONPATH is sometimes preset by the Python installer

Ans145: PYTHONSTARTUP – It contains the path of an initialization file containing Python source code. It is executed every time you start the interpreter. It is named as .pythonrc.py in Unix and it contains commands that load utilities or modify PYTHONPATH.

Q146).What is the purpose of PYTHONCASEOK environment variable?

Ans146: PYTHONCASEOK – It is used in Windows to instruct Python to find the first case-insensitive match in an import statement. Set this variable to any value to activate it.

Q147).What is the purpose of PYTHONHOME environment variable?

Ans147: PYTHONHOME – It is an alternative module search path. It is usually embedded in the PYTHONSTARTUP or PYTHONPATH directories to make switching module libraries easy.

Q148).What does a python do?

Ans148: Python is a general-purpose programming language typically used for web development. ... **SQLite** is one free lightweight database commonly used by **Python** programmers to store data. Many highly trafficked websites, such as YouTube, are created using **Python**.

Q149).What is the interpreter in Python?

Ans149: An **interpreter** is a program that reads and executes code. This includes source code, pre-compiled code, and scripts. Common **interpreters** include Perl, **Python**, and Ruby **interpreters**, which execute Perl, **Python**, and Ruby code respectively.

Q150).Why is it called Python?

Ans150: When 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 1970s. Van Rossum thought he needed a name that was short, unique, and slightly mysterious, so he decided to call the language **Python**.

Click here to read more...

Summary	
Reviewer	Meghana
Review Date	2017-08-24
Reviewed Item	Excellent!! Collection of Python Interview Questions and Answers! Thank you For sharing!!
Author Rating	★★★★★

150 PYTHON INTERVIEW QUESTIONS AND ANSWERS | PYTHON INTERVIEW QUESTIONS

RECOMMENDED POSTS

Pega Interview Questions And Answers

May 21, 2018

Mechanical Engineering Interview Questions and Answers

May 21, 2018

› Finance and Accounting

› Information Technologies (IT)

Popular | Recent | Comments

OOPS ABAP Interview Questions and Answers
June 12, 2017

Basic Accounting and Financial Accounting Interview Questions and Answers
May 17, 2017

Interview Questions for Adobe Experience Manager AEM
May 23, 2017

IOT Interview Questions and Answers
May 20, 2017

ITIL Service Transition Interview Questions
May 27, 2017

SOCIAL

TAGS

angularjs interview questions | automation testing with python | aws | aws interview questions | Data Science Interview Question | Digital Marketing | Digital Marketing Interview questions | django interview questions | ext js | ext js interview questions | hadoop | hadoop interview questions | hbase

mongo db interview questions node js node js interview questions oracle scm

oracle scm interview questions pentaho bi pentaho bi interview questions

PHP interview questions Python Interview Question ruby cucumber

ruby cucumber interview questions sap Bpc interview questions SEO

SEO Interview Question swift programming swift programming interview questions tableau

tableau interview questions TCL Interview Questions and Answers-2017 testing with python Typo3 cms

Typo3 cms interview questions web design web design interview questions zend framework

zend framework interview questions

10P, IWWA Building, 2nd Floor,7th Main Road, BTM Layout 2nd Stage,Bangalore-560076,Karnataka, India

info@mytectra.com
support@mytectra.com

+91 9019191856
+1 (209) 222-4733 (USA)

ABOUT US

myTectra a global learning solutions company helps transform people and organization to gain real, lasting benefits.Join Today.Ready to Unlock your Learning Potential !

Terms and Conditions

Privacy and Policy