Top 20 Python Interview Questions & Answers

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 (http://docs.python.org/2.4/lib/types.html) provided by Python:

Immutable built-in types of python

1. Numbers (http://docs.python.org/2.4/lib/typesnumeric.html)
2. Strings (http://docs.python.org/2.4/lib/typesseq.html)
3. Tuples

Mutable built-in types of python

1. List (http://docs.python.org/2.4/lib/typesseq-mutable.html)
2. Dictionaries (http://docs.python.org/2.4/lib/typesmapping.html)
3. Sets (http://docs.python.org/2.4/lib/types-set.html)

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 (http://www.python.org/dev/peps/pep-0008/) 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 (http://docs.python.org/2/library/unittest.html) . *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 (http://www.python.org/dev/peps/pep-0257/) 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 '
```

INTERVIEW QUESTIONS

C++ (/INTERVIEW-QUESTIONS/CPP/1)

JAVA (/INTERVIEW-QUESTIONS/JAVA/1)

PYTHON

PYTHON INTERVIEW QUESTIONS (/PYTHON-INTERVIEW-QUESTIONS.HTML)

CLASSES IN PYTHON (/CLASSES-IN-PYTHON.HTML)

PYTHON CTYPES TUTORIAL (/PYTHON-CTYPES.HTML)

PYTHON OBJECTS (/PYTHON-OBJECT.HTML)

PYTHON PACKAGE INSTALLATIONS (/HOW-TO-INSTALL-PACKAGES-IN-PYTHON.HTML)

C/C++ QUICK REFERENCE

C++ STL - VECTOR LISTS DEQUE (/CPP-STL-LIST-VECTOR-DEQUE.HTML)

C++ STL - MAP, SET (/CPP-STL-MAP-SET.HTML)

C++ STL - ITERATORS (/CPP-STL-ITERATORS.HTML)

C++ - INLINE FUNCTIONS (/INLINE-FUNCTIONS.HTML)

C++ - 2D DYNAMIC ARRAY (/CPP-2D-ARRAY-CREATION-DYNAMICALLY.HTML)

STRING FUNCTIONS (/STRING-FUNCTIONS-IN-C.HTML)

FILE HANDLING IN C (/C-FILE-OPERATIONS.HTML)

BINARY PROGRAMS IN C

BINARY PRINT A INTEGER (/C-PROGRAM-TO-PRINT-INTEGER-IN-BINARY.HTML)

REVERSE BINARY PRINT A INTEGER (/C-PROGRAM-TO-REVERSE-BINARY-PRINT-A-INTEGER.HTML)

FTP (/FTP-UNIX-COMMAND.HTML)

GDB DEBUGGING (/GDB-TUTORIAL.HTML)

GLUSTERFS INSTALLATION & PERFORMANCE (/GLUSTERFS-INSTALLATION-PERFORMANCE.HTML)

IPV6 SOCKET APIS (/IPV6-SOCKET-API.HTML)

IPV6 TUTORIALS (/IPV6-TUTORIALS.HTML)

JAVA EXECUTORSERVICE (/JAVA-CONCURRENCY-EXECUTORSERVICE.HTML)

JAVA PROGRAM FOR FILE APPEND (/JAVA-FILE-APPEND-PROGRAM.HTML)

ESSENTIAL LINUX COMMANDS (/LINUX-COMMANDS.HTML)

LOOP DEVICE IN LINUX (/LOOP-DEVICE-IN-LINUX.HTML)

POSTGRESQL SETUP (/POSTGRESQL.HTML)

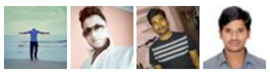
RESUME TIPS (/RESUME.HTML)

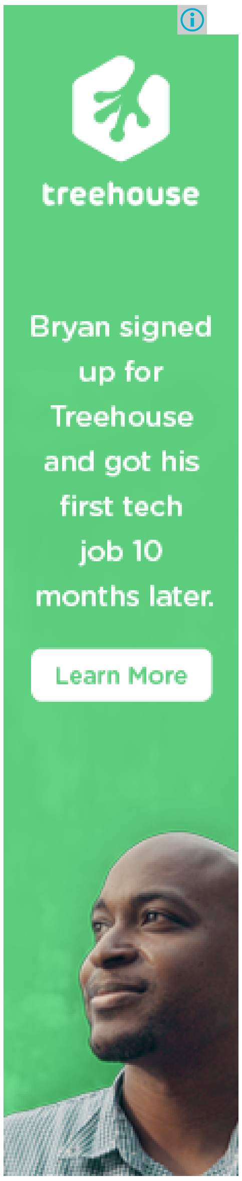
TAYGA (/TAYGA.HTML)

Questions ...

Like Page

3 friends like this




treehouse
Bryan signed up for Treehouse and got his first tech job 10 months later.
Learn More