D. Y. Patil Pratishthan's

Institute for Advanced Computing and Software Development

IACSD

Python

Dr. D.Y. Patil Educational Complex Sector 29,Near Akurdi Railway Station, Nigdi Pradhikaran,Akurdi,Pune-44

Q. How can you improve the following code?

```
import string

i = 0
for letter in string.letters:
    print("The letter at index %i is %s" % (i, letter))
    i = i + 1
```

Bonus points for mentioning `enumerate` and use of `str.format`.

Q. What is Python particularly good for? When is using Python the "right choice" for a project?

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 has fewer syntactical constructions than other languages.

Python is a high-level general-purpose programming language that can be applied to many different classes of problems.

The language comes with a large standard library that covers areas such as string processing like regular expressions, Unicode, calculating differences between files, Internet protocols like HTTP, FTP, SMTP, XML-RPC, POP, IMAP, CGI programming, software engineering like unit testing, logging, profiling, parsing Python code, and operating system interfaces like system calls, file systems, TCP/IP sockets.

Although likes and dislikes are highly personal, a developer who is "worth his or her salt" will highlight features of the Python language that are generally considered advantageous (which also helps answer the question of what Python is "particularly good for". Some of the more common valid answers to this question include:

- Ease of use and ease of refactoring, thanks to the flexibility of Python's syntax, which makes it especially useful for rapid prototyping.
- More compact code, thanks again to Python's syntax, along with a wealth of functionally-rich Python libraries (distributed freely with most Python language implementations).
- A dynamically-typed and strongly-typed language, offering the rare combination of code flexibility while at the same time avoiding pesky implicit-type-conversion bugs.
- It's free and open source! Need we say more?

With regard to the question of when using Python is the "right choice" for a project, the complete answer also depends on a number of issues orthogonal to the language itself, such as prior technology investment, skill set of the team, and so on. Although the question as stated above implies interest in a strictly technical answer, a developer who will raise these additional issues in an interview will always "score more points" with me since it indicates an awareness of, and sensitivity to, the "bigger picture" (i.e., beyond just the technology being employed). Conversely, a response that Python is always the right choice is a clear sign of an unsophisticated developer.

Q. What are some drawbacks of the Python language?

For starters, if you know a language well, you know its drawbacks, so responses such as "there's nothing I don't like about it" or "it has no drawbacks" are very telling indeed.

The two most common valid answers to this question (by no means intended as an exhaustive list) are:

- The Global Interpreter Lock (GIL). CPython (the most common Python implementation) is not fully thread safe. In order to support multi-threaded Python programs, CPython provides a global lock that must be held by the current thread before it can safely access Python objects. As a result, no matter how many threads or processors are present, only one thread is ever being executed at any given time. In comparison, it is worth noting that the PyPy implementation discussed earlier in this article provides a stackless mode that supports micro-threads for massive concurrency.
- Execution speed. Python can be slower than compiled languages since it is interpreted. (Well, sort of. See our earlier discussion on this topic.)

Q. We know Python is all the rage these days. But to be truly accepting of a great technology, you must know its pitfalls as well.

Of course. To be truly yourself, you must be accepting of your flaws. Only then can you move forward to work on them. Python has its flaws too:

Python's interpreted nature imposes a speed penalty on it. While Python is great for a lot of things, it is weak in mobile computing, and in browsers.

Being dynamically-typed, Python uses duck-typing (If it looks like a duck, it must be a duck). This can raise runtime errors.

Python has underdeveloped database access layers. This renders it a less-than-perfect choice for huge database applications.

And even after these pitfalls, of course. Being easy makes it addictive. Once a Python-coder, always a Python coder.

Q. What are the key differences between Python 2 and 3?

```
Division operator
`print` function
Unicode
xrange
Error Handling
`_future_` module
```

Although Python 2 is formally considered legacy at this point,its use is still widespread enough that is important for a developer to recognize the differences between Python 2 and 3.

- Here are some of the key differences that a developer should be aware of:
    - Text and Data instead of Unicode and 8-bit strings. Python 3.0 uses the concepts of text and (binary) data instead of Unicode strings and 8-bit strings. The biggest ramification of this is that any attempt to mix text and data in Python 3.0 raises a TypeError (to combine the two safely, you must decode bytes or encode Unicode, but you need to know the proper encoding, e.g. UTF-8)
    - This addresses a longstanding pitfall for naïve Python programmers. In Python 2, mixing Unicode and 8-bit data would work if the string happened to contain only 7-bit (ASCII) bytes, but you would get UnicodeDecodeError if it contained non-ASCII values. Moreover, the exception would happen at the combination point, not at the point at which the non-ASCII characters were put into the str object. This behavior was a common source of confusion and consternation for neophyte Python programmers.
    - `print` function. The print statement has been replaced with a print() function
    - `xrange` – buh-bye. xrange() no longer exists (range() now behaves like xrange() used to behave, except it works with values of arbitrary size)
- API changes:
    - `zip()`, `map()` and `filter()` all now return iterators instead of lists.
    - `dict.keys()`, `dict.items()` and `dict.values()` now return 'views' instead of lists.
    - `dict.iterkeys()`, `dict.iteritems()` and `dict.itervalues()` are no longer supported.
    - Comparison operators. The ordering comparison operators (<, <=, >=, >) now raise a TypeError exception when the operands don't have a meaningful natural ordering. Some examples of the ramifications of this include:
    - Expressions like 1 < '', 0 > None or len <= len are no longer valid
    - None < None now raises a TypeError instead of returning False
    - Sorting a heterogeneous list no longer makes sense.
    - All the elements must be comparable to each other

Q. What are some key differences to bear in mind when coding in Python vs. Java?

Disclaimer #1. The differences between Java and Python are numerous and would likely be a topic worthy of its own (lengthy) post. Below is just a brief sampling of some key differences between the two languages.

Disclaimer #2. The intent here is not to launch into a religious battle over the merits of Python vs. Java (as much fun as that might be!). Rather, the question is really just geared at seeing how well the developer understands some practical differences between the two languages. The list below therefore deliberately avoids discussing the arguable advantages of Python over Java from a programming productivity perspective.

- With the above two disclaimers in mind, here is a sampling of some key differences to bear in mind when coding in Python vs. Java:
    - Dynamic vs static typing: One of the biggest differences between the two languages is that Java is restricted to static typing whereas Python supports dynamic typing of variables.
    - Static vs. class methods: A static method in Java does not translate to a Python class method.
    - In Python, calling a class method involves an additional memory allocation that calling a static method or function does not.
    - In Java, dotted names (e.g., foo.bar.method) are looked up by the compiler, so at runtime it really doesn't matter how many of them you have. In Python, however, the lookups occur at runtime, so "each dot counts".
    - Method overloading: Whereas Java requires explicit specification of multiple same-named functions with different signatures, the same can be accomplished in Python with a single function that includes optional arguments with default values if not specified by the caller.
    - Single vs. double quotes. Whereas the use of single quotes vs. double quotes has significance in Java, they can be used interchangeably in Python (but no, it won't allow beginnning the same string with a double quote and trying to end it with a single quote, or vice versa!).
    - Getters and setters (not!). Getters and setters in Python are superfluous; rather, you should use the 'property' built-in (that's what it's for!). In Python, getters and setters are a waste of both CPU and programmer time.
    - Classes are optional. Whereas Java requires every function to be defined in the context of an enclosing class definition, Python has no such requirement.
    - Indentation matters… in Python. This bites many a newbie Python programmer.

  The Big Picture

    - An expert knowledge of Python extends well beyond the technical minutia of the language. A Python expert will have an in-depth understanding and appreciation of Python's benefits as well as its limitations. Accordingly, here are some sample questions that can help assess this dimension of a candidate's expertise:

Q. What will be the output of the code below in Python 2?

```
def div1(x,y):
    print "%s/%s = %s" % (x, y, x/y)
def div2(x,y):
    print "%s//%s = %s" % (x, y, x//y)

div1(5,2)
div1(5.,2)
div2(5,2)
div2(5.,2.)
```

Also, how would the answer differ in Python 3 (assuming, of course, that the above [print] statements were converted to Python 3 syntax)?

```
-  kjalfkjaslf
```

Q. What is the difference between range and xrange? How has this changed over time?

- As follows:
  - `xrange` returns the xrange object while range returns the list, and uses the same memory and no matter what the range size is.
  - For the most part, xrange and range are the exact same in terms of functionality. They both provide a way to generate a list of integers for you to use, however you please.
  - The only difference is that range returns a Python list object and x range returns an xrange object. This means that xrange doesn't actually generate a static list at run-time like range does. It creates the values as you need them with a special technique called yielding. This technique is used with a type of object known as generators. That means that if you have a really gigantic range you'd like to generate a list for, say one billion, xrange is the function to use.
  - This is especially true if you have a really memory sensitive system such as a cell phone that you are working with, as range will use as much memory as it can to create your array of integers, which can result in a Memory Error and crash your program. It's a memory hungry beast.

Q. What will be the output of the code below?

```
List = ['a', 'b', 'c', 'd', 'e']
print(list[10:])
```

- TypeError: 'type' object is not subscriptable if proper name given,it will print [].

Q. What is a method?

A method is a function on some object x that you normally call as x.name(arguments...). Methods are defined as functions inside the class definition:

```
class C:
    def meth (self, arg):
        return arg*2 + self.attribute
```

Q. How do I call a method defined in a base class from a derived class that overrides it?

If you're using new-style classes, use the built-in `super()` function:

```
class Derived(Base):
    def meth (self):
       super(Derived, self).meth()
```

If you're using classic classes: For a class definition such as `class Derived(Base):` ... you can call method meth() defined in Base (or one of Base's base classes) as Base.meth(self,arguments). Here, Base.meth is an unbound method, so you need to provide the self argument.

Q. How can I organize my code to make it easier to change the base class?

You could define an alias for the base class, assign the real base class to it before your class definition, and use the alias throughout your class. Then all you have to change is the value assigned to the alias. Incidentally, this trick is also handy if you want to decide dynamically (e.g. depending on availability of resources) which base class to use.

Example: BaseAlias = class Derived(BaseAlias): def meth(self): BaseAlias.meth(self).

Q. How do I find the current module name?

A module can find out its own module name by looking at the predefined global variable `__name__`. If this has the value `'__main__'`, the program is running as a script. Many modules that are usually used by importing them also provide a command-line interface or a self-test, and only execute this code after checking `__name__`:

```
def main():
    print('Running test...')
if __name__ == '__main__':
    main()
```

`__import__('x.y.z')` returns Try: `__import__('x.y.z').y.z`

```
# For more realistic situations, you may have to do something like:
m = __import__(s)
        for i in s.split(".")[1:]: m = getattr(m, i)
```

Q. How do I access a module written in Python from C?

You can get a pointer to the module object as follows:

```
module = PyImport_ImportModule("");
```

If the module hasn't been imported yet (i.e. it is not yet present in sys.modules), this initializes the module; otherwise it simply returns the value of `sys.modules[""]`. Note that it doesn't enter the module into any namespace -- it only ensures it has been initialized and is stored in sys.modules. You can then access the module's attributes (i.e. any name defined in the module) as follows: attr = PyObject_GetAttrString(module, ""); Calling PyObject_SetAttrString() to assign to variables in the module also works.

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

To convert, e.g., the number 144 to the string '144', use the built-in function str(). If you want a hexadecimal or octal representation, use the built-in functions hex() or oct(). For fancy formatting, use the % operator on strings, e.g. "%04d" % 144 yields '0144' and "%.3f" % (1/3.0) yields '0.333'. See the library reference manual for details.

Q. How is the Implementation of Python's dictionaries done?

Python dictionary needs to be declared first:
```
dict = {}
```

Key value pair can be added as:
```
dict[key] = value
```
or
```
objDict.update({key:value})
```

Remove element by:
```
dict.pop(key)
```

Remove all:
```
objDict.clear()
```

A hash value of the key is computed using a hash function, The hash value addresses a location in an array of "buckets" or "collision lists" which contains the (key , value) pair.

Q. What is used to create Unicode string in Python?

"u" should be added before the string

```
a = (u'Python')
type(a) #will give you unicode
```

Add unicode before the string. Ex: unicode(text) resulting in text.

Q. What is the built-in function used in Python to iterate over a sequence of numbers?

Syntax: `range(start,end,step count)`

Ex:

```
a = range(1,10,2)
print (a)
```

Output: `[1, 3, 5, 7, 9]`

If using to iterate

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

Output:

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

Q. Does Python have a switch-case statement?

Ans. In languages like C++, we have something like this:

```
switch(name)
{
    case 'Ram':
        cout<<"Monday";
        break;
    case 'Shiv':
        cout<<"Tuesday";
        break;
    default:
        cout<<"Hi, user";
}
```

But in Python, we do not have a switch-case statement. Here, you may write a switch function to use. Else, you may use a set of if-elif-else statements. To implement a function for this, we may use a dictionary.

```
def switch(choice):
    switcher={
        'Ram':'Monday',
        'Shiv':'Tuesday',
        print(switcher.get(choice,'Hi, user'))
    return
    switch('Shiv')
    Tuesday
    switch('Ram')
    Monday
    switch('Raghav')
    Hi, user
    }
```

Here, the get() method returns the value of the key. When no key matches, the default value (the second argument) is returned.

Q. Does python support switch or case statement in Python? If not what is the reason for the same?

Dictionary can be used as case/switch. Actually there is no switch statement in the Python programming language but the is a similar construct that can do justice to switch that is the exception handling using try and except1,except2,except3.... and so on.

Q. What is the statement that can be used in Python if a statement is required syntactically but the program requires no action?

`pass` keyword is used to do nothing but it fulfill the syntactical requirements.

```
try x[10]:
    print(x)
except:
    pass
```

Use `pass` keyword over there like:

```
if a > 0:
    print("Hello")
else:
    pass
```

Q. Does Python support strongly for regular expressions?

Yes, Python Supports Regular Expressions Well. `re` is an in-buit library for the same. There is a lot of other languages that have good support to RegEx- Perl, Awk, Sed, Java etc.

Regular expressions (called REs, or regexes, or regex patterns) are essentially a tiny, highly specialized programming language embedded inside Python and made available through the re module. Using this little language, you specify the rules for the set of possible strings that you want to match; this set might contain English sentences, or e-mail addresses, or TeX commands, or anything you like. You can then ask questions such as "Does this string match the pattern?", or "Is there a match for the pattern anywhere in this string?". You can also use REs to modify a string or to split it apart in various ways.

Regular expression patterns are compiled into a series of bytecodes which are then executed by a matching engine written in C. For advanced use, it may be necessary to pay careful attention to how the engine will execute a given RE, and write the RE in a certain way in order to produce bytecode that runs faster. Optimization isn't covered in this document, because it requires that you have a good understanding of the matching engine's internals.

Q. How do you perform pattern matching in Python? Explain.

Regular Expressions/REs/ regexes enable us to specify expressions that can match specific "parts" of a given string. For instance, we can define a regular expression to match a single character or a digit, a telephone number, or an email address, etc. The Python's "re" module provides regular expression patterns and was introduce from later versions of Python 2.5. "re" module is providing methods for search text strings, or replacing text strings along with methods for splitting text strings based on the pattern defined.

Q. Write a regular expression that will accept an email id. Use the `re` module.

Ans.

```
import re
e = re.search(r'[0-9a-zA-Z.]+@[a-zA-Z]+\.(com|co\.in)$' 'JaiRameshwar@gmail.com')
e.group()
```

Q. What is Garbage Collection?

The concept of removing unused or unreferenced objects from the memory location is known as a Garbage Collection. While executing the program, if garbage collection takes place then more memory space is available for the program and rest of the program execution becomes faster.

Garbage collector is a predefined program, which removes the unused or unreferenced objects from the memory location.

Any object reference count becomes zero then we call that object as a unused or unreferenced object Then no.of reference variables which are pointing the object is known as a reference count of the object.

While executing the python program if any object reference count becomes zero, then internally python interpreter calls the garbage collector and garbage collector will remove that object from memory location.

Q. How is memory 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. Like other programming language python also has garbage collector which will take care of memory management in python.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. 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 has a private heap space to hold all objects and data structures. Being programmers, we cannot access it; it is the interpreter that manages it. But with the core API, we can access some tools. The Python memory manager controls the allocation.

Q. Why isn't all memory freed when Python exits?

Objects referenced from the global namespaces of Python modules are not always deallocated when Python exits. This may happen if there are circular references. There are also certain bits of memory ...

Q. Whenever you exit Python, is all memory de-allocated? State why is it so.

The answer here is no. The modules with circular references to other objects, or to objects referenced from global namespaces, aren't always freed on exiting Python. Plus, it is impossible to de-allocate portions of memory reserved by the C library.

Whenever Python exits, especially those Python modules which are having circular references to other objects or the objects that are referenced from the global namespaces are not always de-allocated or freed.It is impossible to de-allocate those portions of memory that are reserved by the C library.On exit, because of having its own efficient clean up mechanism, Python would try to de-allocate/destroy every other object.

Q. Is it possible to assign multiple var to values in list?

The multiple assignment trick is a shortcut that lets you assign multiple variables with the values in a list in one line of code. So instead of doing this:

```
cat = ['fat', 'orange', 'loud']
size = cat[0]
color = cat[1]
disposition = cat[2]
```

Do this:

```
cat = ['fat', 'orange', 'loud']
size, color, disposition = cat
```

Q. What is `__slots__` and when is it useful?

In Python, every class can have instance attributes. By default Python uses a `dict` to store an object's instance attributes. This is really helpful as it allows setting arbitrary new attributes at runtime.

However, for small classes with known attributes it might be a bottleneck. The `dict` wastes a lot of RAM. Python can't just allocate a static amount of memory at object creation to store all the attributes. Therefore it sucks a lot of RAM if you create a lot of objects. The usage of `__slots__` to tell Python not to use a dict, and only allocate space for a fixed set of attributes.

Example:

1. Object without slots

```
class MyClass(object):
      def __init__(self, *args, **kwargs):
                self.a = 1
                self.b = 2

if __name__ == "__main__":
     instance = MyClass()
     print(instance.__dict__)
```

2. Object with slots

```
class MyClass(object):
      __slots__=['a', 'b']
      def __init__(self, *args, **kwargs):
                self.a = 1
                self.b = 2

if __name__ == "__main__":
     instance = MyClass()
     print(instance.__slots__)
```

Q. What Are The Types of Objects Support in Python Language?

Python supports are two types are of objects. They are:

Immutable built-in types:

```
Strings
Tuples
Numbers
```

Mutable built-in types:

```
List
Sets
Dictionaries
```

- Immutable Objects

  The objects which doesn't allow to modify the contents of those objects are known as 'Immutable Objects'

  Before creating immutable objects with some content python interpreter verifies is already any object is available. In memory location with same content or not.

  If already object is not available then python interpreter creates new objects with that content and store that object address two reference variable.

  If already object is present in memory location with the same content creating new objects already existing object address will be given to the reference variable.

  Program:

```
i=1000
print(i)
print(type(i))
print(id(i))
j=2000
print(j)
print(type(j))
print(id(j))
x=3000
print(x)
print(type(x))
print(id(x))
y=3000
print(y)
print(type(y))
print(id(y))
```

  `int, float, complex, bool, str, tuple` are immutable objects

  Immutable objects performance is high.

  Applying iterations on Immutable objects takes less time.

  All fundamentals types represented classes objects and tuple class objects are immutable objects.

- Mutable Objects:
    1. The Objects which allows to modify the contents of those objects are known as 'Mutable Objects'
    2. We can create two different mutable objects with same content

  Program:

```
x=[10,20,30]
print(x)
print(type(x))
print(id(x))
y=[10,20,30]
print(y)
print(type(y))
print(id(y))
```

  Output:

  `List, set, dict` classes objects are mutable objects

  Mutable objects performance is low when compared to immutable objects
  Applying Iterations mutable objects takes huge time

Q. Python is Call by Value or Call by Reference? 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.

Q. Explain Python's parameter-passing mechanism.

- To pass its parameters to a function, Python uses pass-by-reference. If you change a parameter within a function, the change reflects in the calling function. This is its default behavior.
- However, when we pass literal arguments like strings, numbers, or tuples, they pass by value. This is because they are immutable.

Q. What are `*args`, `**kwargs` ?

In cases when we don't know how many arguments will be passed to a function, like when we want to pass a list or a tuple of values, we use `*args`.

```
def func(*args):
    for i in args:
        print(i)

func(3,2,1,4,7)
```

```
3
2
1
4
7
```

`**kwargs` takes keyword arguments when we don't know how many there will be:

```
def func(**kwargs):
    for i in kwargs:
        print(i,kwargs[i])

func(a=1,b=2,c=7)
```

```
a.1
b.2
c.7
```

The words `args` and `kwargs` are a convention, and we can use anything in their place.

Q. How can I pass optional or keyword parameters from one function to another?

Collect the arguments using the * and ** specifier in the function's parameter list; this gives you the positional arguments as a tuple and the keyword arguments as a dictionary. You can then pass these arguments when calling another function by using `*` and `**` :

```
def f(x, *tup, **kwargs):
    kwargs['width']='14.3c'
    g(x, *tup, **kwargs)
```

In the unlikely case that you care about Python versions older than 2.0, use 'apply':

```
def f(x, *tup, **kwargs):
    kwargs['width']='14.3c'
    apply(g, (x,)+tup, kwargs)
```

Q. What is lambda? What are Lambda Functions ?

A function which doesn't contain any name is known as a anonymous function lambda function, Lambda function we can assign to the variable & we can call the lambda function through the variable.

Syntax: `Lambda arguments:expression`

It is a single expression anonymous function often used as inline function. A lambda form in python does not have statements as it is used to make new function object and then return them at runtime.

The lambda operator is used to create anonymous functions. It is mostly used in cases where one wishes to pass functions as parameters. or assign them to variable names.

When we want a function with a single expression, we can define it anonymously. A lambda expression may take input and returns a value. To define the above function as a lambda expression, we type the following code in the interpreter:

```
(lambda a,b:a if a>b else b)(3,3.5)
```

```
3.5
```

Here, a and b are the inputs.
`a if a > b else b` is the expression to return. The arguments are 3 and 3.5.

It is possible to not have any inputs here.

```
(lambda :print("Hi"))()
```

Hi

Example:

```
myfunction = lambda x:x*x
a = myfunction(10)
print(a)
```

Output: 100

- Why can't lambda forms in Python contain statements?

Lambdas evaluates at run time and these do not need statements Lambda is a anonymous function, which does not have a name and no fixed number of arguments. Represented by keyword lambda followed by statement. Ex:

```
add = lambda a,b: a+b
add(2,3)
```

output:

```
5
```

Q. How do you create your own package in Python?

It overrides the any initialization from an inherited class and is called when the class is instantiated.

We know that a package may contain sub-packages and modules. A module is nothing but Python code.
To create a package of our own, we create a directory and create a file `__init__.py` in it. We leave it empty. Then, in that package, we create a module(s) with whatever code we want. For a detailed explanation with pictures, refer to Python Packages.

Q. Explain the use "with" statement in python?

- In python generally "with" statement is used to open a file, process the data present in the file, and also to close the file without calling a close() method. "with" statement makes the exception handling simpler by providing cleanup activities.

General form of with:

```
with open("filename", "mode") as file-var:
```

processing statements
Note: no need to close the file by calling close() upon file-var.close()

Q. What is Monkey patching ? Give example ?

Dynamically modifying a class or module at run-time.

```
class A:
    def func(self):
        print("Hi")
    def monkey(self):
        print "Hi, monkey"
    m.A.func = monkey
    a = m.A()
    a.func()
```

Hi, monkey

Q. Explain serialization and deserialization / Pickling and unpicking.

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.

To create portable serialized representations of Python objects, we have the module 'pickle'. It accepts a Python object (remember, everything in Python is an object). It then converts it into a string representation and uses the dump() function to dump it into a file. We call this pickling. In contrast, retrieving objects from this stored string representation is termed 'unpickling'.

The pickle module implements binary protocols for serializing and deserializing a Python object structure. "Pickling" is the process whereby a Python object hierarchy is converted into a byte stream, and "unpickling" is the inverse operation, whereby a byte stream (from a binary file or bytes-like object) is converted back into an object hierarchy. Pickling (and unpickling) is alternatively known as `serialization`, `marshalling`, or `flattening`; however, to avoid confusion, the terms used here are `pickling` and `unpickling`.

```
import json
json_string = json.dumps([1, 2, 3, "a", "b"])
print(json_string)

import pickle
pickled_string = pickle.dumps([1, 2, 3, "a", "b"])
print(pickle.loads(pickled_string))
```

Q. What are higher ordered functions?

You have two choices: you can use nested scopes or you can use callable objects. For example, suppose you wanted to define linear(a,b) which returns a function f(x) that computes the value a*x+b.

```
Using nested scopes:

def linear(a,b):
    def result(x):
        return a*x + b
            return result
```

Or

```
using a callable object:

class linear:
    def __init__(self, a, b):
        self.a, self.b = a,b
        def __call__(self, x):
            return self.a * x + self.b
```

In both cases:

`taxes = linear(0.3,2)` gives a callable object where

```
taxes(10e6) == 0.3 * 10e6 + 2.
```

The callable object approach has the disadvantage that it is a bit slower and results in slightly longer code. However, note that a collection of callables can share their signature via inheritance:

```
class exponential(linear):
    __init__ inherited
    def __call__(self, x):
        return self.a * (x ** self.b)
```

Object can encapsulate state for several methods:

```
class counter:
    value = 0
    def set(self, x):
        self.value = x
    def up(self):
        self.value=self.value+1
    def down(self):
        self.value=self.value-1
        count = counter()

inc, dec, reset = count.up, count.down, count.set
```

Here inc(), dec() and reset() act like functions which share the same counting variable.

Q. How do I copy a file? How to copy object in Python? Diff between shallow copy and deep copy?

The shutil module contains a `copyfile()` function.

A deep copy copies an object into another. This means that if you make a change to a copy of an object, it won't affect the original object. In Python, we use the function deepcopy() for this, and we import the module copy. We use it like:

```
import copy
b = copy.deepcopy (a)
```

A shallow copy, however, copies one object's reference to another. So, if we make a change in the copy, it will affect the original object. For this, we have the function `copy()`, we use it like:

```
b = copy.copy(a)
```

- Differentiate between lists and tuples.

The major difference is that a list is mutable, but a tuple is immutable. Examples:

Traceback (most recent call last): File "<pyshell#97>", line 1, in mytuple[1]=2
TypeError: 'tuple' object does not support item assignment

Q. What is the purpose of PYTHONSTARTUP, PYTHONCASEOK, PYTHONHOME & PYTHONPATH environment variables?

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

Q. Explain Inheritance in Python with an example.

When one class inherits from another, it is said to be the child/ derived/sub class inheriting from the parent/base/super class. It inherits/gains all members (attributes and methods). Inheritance lets us reuse our code, and also makes it easier to create and maintain applications.

Inheritance allows One class to gain all the members(say attributes and methods) of another class. Inheritance provides code reusability,makes it easier to create and maintain an application. The class from which we are inheriting is called super-class and the class that is inherited is called a derived/child class.

- They are different types of inheritance supported by Python:
  - Single Inheritance – where a derived class acquires the members of a single super class.

    OR

  - Single Inheritance- A class inherits from a single base class.
  - Multi-level inheritance – a derived class d1 in inherited from base class base1, and d2 is inherited from base2.

OR

- Multilevel Inheritance- A class inherits from a base class, which in turn, inherits from another base class.
- Hierarchical inheritance – from one base class you can inherit any number of child classes

OR

- Hierarchical Inheritance- Multiple classes inherit from a single base class.
- Multiple inheritance – a derived class is inherited from more than one base class.

OR

- Multiple Inheritance- A class inherits from multiple base classes.
- Hybrid Inheritance- Hybrid inheritance is a combination of two or more types of inheritance.

Q. What is Hierarchical Inheritance?

The concept of inheriting the properties from one class into multiple classes separately is known as hierarchical inheritance.

Example:

```
class x(object):
    def m1(self):
        print("in m1 of x")

class y(x):
    def m2(self):
        print("in m2 of y")

class z(x):
    def m3(self):
        print("in m3 of z")
y1=y()
y1.m1()
y1.m2()
a=y1.--hash--()
print(a)
z1=z()
```

```
z1.m1()
z1.m3()
b=z1.hash--()
print(b)
```

Output:

```
M m1 of X
In m2 of Y
2337815
In m1 of X
In m3 of Z
2099735
```

Q. Suppose class C inherits from classes A and B as class C(A,B).Classes A and B both have their own versions of method func(). If we call func() from an object of class C, which version gets invoked?

Ans. In our article on Multiple Inheritance in Python, we discussed Method Resolution Order (MRO). C does not contain its own version of func(). Since the interpreter searches in a left-to-right fashion, it finds the method in A, and does not go to look for it in B.

Q. Which methods/functions do we use to determine the type of instance and inheritance?

Ans. Here, we talk about three methods/functions- `type(), isinstance() and issubclass().`

a. `type()`: This tells us the type of object we're working with.

```
type(3)
```

<class 'int'>

```
type(False)
```

<class 'bool'>

```
type(lambda :print("Hi"))
```

<class 'function'>

```
type(type)
```

<class 'type'>

b. isinstance()

This takes in two arguments- a value and a type. If the value is of the kind of the specified type, it returns True. Else, it returns False.

```
 isinstance(3,int)
```

True

```
 isinstance((1),tuple)
```

False

```
 isinstance((1,),tuple)
```

True

c. issubclass()

This takes two classes as arguments. If the first one inherits from the second, it returns True. Else, it returns False.

```
 class A: pass
 class B(A): pass
 issubclass(B,A)
```

True

```
 issubclass(A,B)
```

False

Q. Write a one-liner that will count the number of capital letters in a file. Your code should work even if the file is too big to fit in memory.

Let us first write a multiple line solution and then convert it to one liner code.

```py
with open(SOME_LARGE_FILE) as fh:
    count = 0
    text = fh.read()
for character in text:
    if character.isupper():
        count += 1
```

Q. Write a sorting algorithm for a numerical dataset in Python.

The following code can be used to sort a list in Python:

```py
list = ["1", "4", "0", "6", "9"]
list = [int(i) for i in list]
list.sort()
print(list)
```

Q. How will you remove last object from a list?

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

Q. What are negative indexes and why are they used?

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.

The sequences in Python are indexed and it consists of the positive as well as negative numbers. The numbers that are positive uses '0' that is uses as first index and '1' as the second index and the process goes on like that.

The index for the negative number starts from '-1' that represents the last index in the sequence and '-2' as the penultimate index and the sequence carries forward like the positive number.

The negative index is used to remove any new-line spaces from the string and allow the string to except the last character that is given as S[:-1]. The negative index is also used to show the index to represent the string in correct order.

Let's take a list for this.

```
mylist=[0,1,2,3,4,5,6,7,8]
```

A negative index, unlike a positive one, begins searching from the right.

```
mylist[-3]
```

6

This also helps with slicing from the back:

```
mylist[-6:-1]
```

[3, 4, 5, 6, 7]

Q. Explain split(), sub(), subn() methods of `re` module in Python.

- To modify the strings, Python's "re" module is providing 3 methods. They are:
  - split() – uses a regex pattern to "split" a given string into a list.
  - sub() – finds all substrings where the regex pattern matches and then replace them with a different string.
  - subn() – it is similar to sub() and also returns the new string along with the no. of replacements.

Q. What is map function in Python?

Map function executes the function given as the first argument on all the elements of the iterable given as the second argument. If the function given takes in more than 1 arguments, then many iterables are given. #Follow the link to know more similar functions

Q. How to get indices of N maximum values in a NumPy array?

We can get the indices of N maximum values in a NumPy array using the below code:

```
import numpy as np
arr = np.array([1, 3, 2, 4, 5])
print(arr.argsort()[-3:][::-1])
```

Q. What is a Python module?

A module is a Python script that generally contains import statements, functions, classes and variable definitions, and Python runnable code and it "lives" file with a '.py' extension. zip files and DLL files can also be modules.Inside the module, you can refer to the module name as a string that is stored in the global variable name .

Q. Name the File-related modules in Python?

Python provides libraries / modules with functions that enable you to manipulate text files and binary fileson file system. Using them you can create files, update their contents, copy, and delete files. The librariesare : os, os.path, and shutil.

Here,
`os` and `os.path` – modules include functions for accessing the filesystem
`shutil` – module enables you to copy and delete the files.

Q. How many kinds of sequences are supported by Python? What are they?

Python supports 7 sequence types. They are str, list, tuple, unicode, byte array, xrange, and buffer. where xrange is deprecated in python 3.5.X.

Q. How to display the contents of text file in reverse order?How will you reverse a list?

`list.reverse()` – Reverses objects of list in place, convert the given file into a list. Reverse the list by using `reversed()`.
E.g.:

```
for line in reversed(list(open("file-name","r"))):
     print(line)
```

Q. What is the difference between NumPy and SciPy?

In an ideal world, NumPy would contain nothing but the array data type and the most basic operations: indexing, sorting, reshaping, basic element wise functions, et cetera. All numerical code would reside in SciPy. However, one of NumPy's important goals is compatibility, so NumPy tries to retain all features supported by either of its predecessors. Thus NumPy contains some linear algebra functions, even though these more properly belong in SciPy. In any case, SciPy contains more fully-featured versions of the linear algebra modules, as well as many other numerical algorithms. If you are doing scientific computing with python, you should probably install both NumPy and SciPy. Most new features belong in SciPy rather than NumPy.

Q. Which of the following is an invalid statement?

a) abc = 1,000,000
b) a b c = 1000 2000 3000
c) a,b,c = 1000, 2000, 3000
d) a_b_c = 1,000,000 Answer: b

Q. What is the output of the following?

~~~
```py
try:
    if '1' != 1:
        raise
```
a) some Error has occured
b) some Error has not occured
c) invalid code
d) none of the above

Answer: C
~~~

Q. Suppose list1 is [2, 33, 222, 14, 25], What is list1[-1] ?

~~~
25
~~~

Q. How to open a file c:\scores.txt for writing?

~~~
``fileWriter = open("c:\\scores.txt", "w")``
~~~

Q. Name few Python modules for Statistical, Numerical and scientific computations ?

```
`numPy` - this module provides an array/matrix type, and it is useful for doing computations on arrays.
`scipy` - this module provides methods for doing numeric integrals, solving differential equations, etc
`pylab` - is a module for generating and saving plots
```

Q. What is TkInter?

TkInter is Python library. It is a toolkit for GUI development. It provides support for various GUI tools or widgets (such as buttons, labels, text boxes, radio buttons, etc) that are used in GUI applications. The common attributes of them include Dimensions, Colors, Fonts, Cursors, etc.

Q. Is Python object oriented? what is object oriented programming?

Yes. Python is Object Oriented Programming language. OOP is the programming paradigm based on classes and instances of those classes called objects. The features of OOP are: Encapsulation, Data Abstraction, Inheritance, Polymorphism.

Q. Does Python supports interfaces like in Java? Discuss.

Python does not provide interfaces like in Java. Abstract Base Class (ABC) and its feature are provided by the Python's "abc" module. Abstract Base Class is a mechanism for specifying what methods must be implemented by its implementation subclasses. The use of ABC'c provides a sort of "understanding" about methods and their expected behaviour. This module was made available from Python 2.7 version onwards.

Q. What are Accessors, mutators, @property?

Accessors and mutators are often called getters and setters in languages like "Java". For example, if x is a property of a user-defined class, then the class would have methods called setX() and getX(). Python has an @property 'decorator' that allows you to ad getters and setters in order to access the attribute of the class.

Q. Differentiate between append() and extend() methods.?

Both append() and extend() methods are the methods of list. These methods are used to add the elements at the end of the list.

`append(element)` – adds the given element at the end of the list which has called this method.
`extend(another-list)` – adds the elements of another-list at the end of the list which is called the extend method.

Q. Name few methods that are used to implement Functionally Oriented Programming in Python?

Python supports methods (called iterators in Python3), such as filter(), map(), and reduce(), that are very useful when you need to iterate over the items in a list, create a dictionary, or extract a subset of a list.

- `filter()` – enables you to extract a subset of values based on conditional logic.
- `map()` – it is a built-in function that applies the function to each item in an iterable.
- `reduce()` – repeatedly performs a pair-wise reduction on a sequence until a single value is computed.

Q. What is the output of the following?

```
x = ['ab', 'cd']
print(len(map(list, x)))
```

A TypeError occurs as map has no len().

Q. What is the output of the following?

```
x = ['ab', 'cd']
print(len(list(map(list, x))))
```

The length of each string is 2.

Q. Which of the following is not the correct syntax for creating a set?

a) set([[1,2],[3,4]]) b) set([1,2,2,3,4]) c) set((1,2,3,4)) d) {1,2,3,4}

A. Explanation : The argument given for the set must be an iterable.

Q. Explain a few methods to implement Functionally Oriented Programming in Python.

Sometimes, when we want to iterate over a list, a few methods come in handy.

`filter()`: Filter lets us filter in some values based on conditional logic.

```
list(filter(lambda x:x>5,range(8)))
```

Ans: [6, 7] `map()`: Map applies a function to every element in an iterable.

```
list(map(lambda x:x**2,range(8)))
```

Ans: [0, 1, 4, 9, 16, 25, 36, 49]

`reduce()`: Reduce repeatedly reduces a sequence pair-wise until we reach a single value

```
from functools import reduce
reduce(lambda x,y:x-y,[1,2,3,4,5])
```

Ans: -13

Q. Write a Python function that checks whether a passed string is palindrome Or not?

Note: A palindrome is a word, phrase, or sequence that reads the same backward as forward, e.g., madam , saas, nun.

```
def isPalindrome(string):
    left_pos = 0
    right_pos = len(string) - 1

    while right_pos >= left_pos:
        if not string[left_pos] == string[right_pos]:
            return False

    left_pos += 1
    right_pos -= 1
    return True
print(isPalindrome('aza'))
```

Q. Write a Python program to calculate the sum of a list of numbers.

```
def list_sum(num_List):
    if len(num_List) == 1:
        return num_List[0]
    else:
        return num_List[0] + list_sum(num_List[1:])
print(list_sum([2, 4, 5, 6, 7]))
```

Sample Output: 24

Q. How to retrieve data from a table in MySQL database through Python code? Explain.

```
#import MySQLdb module as :
import MySQLdb

#establish a connection to the database.
db = MySQLdb.connect("host"="local host", "database-user"="user-name", "password"="password","database-
name"="database")

#initialize the cursor variable upon the established connection:
c1 = db.cursor()

#retrieve the information by defining a required query string.
s = "Select * from dept"

#fetch the data using fetch() methods and print it.
data = c1.fetch(s)

#close the database connection.
db.close()
```

Q. Write a Python program to read a random line from a file.

```
import random
def random_line(fname):
    lines = open(fname).read().splitlines()
    return random.choice(lines)
    print(random_line('test.txt'))
```

Q. Write a Python program to count the number of lines in a text file.

```
def file_lengthy(fname):
    with open(fname) as f:
        for i, l in enumerate(f):
            pass
        return i + 1

print("Number of lines in the file: ",file_lengthy("test.txt"))
```

Q. What are the key features of Python?

If it makes for an introductory language to programming, Python must mean something. These are its qualities:

- Interpreted.
- Dynamically-typed.
- Object-oriented
- Concise and simple
- Free
- Has a large community

Q. Explain the ternary operator in Python.

Unlike C++, we don't have ?: in Python, but we have this:

[on true] if [expression] else [on false]

If the expression is True, the statement under [on true] is executed. Else, that under [on false] is executed.

Below is how you would use it: ex 1.

```
a,b=2,3
min=a if a<b else b
print(min)
```

Ans: 2

ex 2.

```
print("Hi") if a<b else print("Bye")
```

Ans: Hi

Q. What is multithreading? Give an example.

It means running several different programs at the same time concurrently by invoking multiple threads. Multiple threads within a process refer the data space with main thread and they can communicate with each other to share information more easily.Threads are light-weight processes and have less memory overhead. Threads can be used just for quick task like calculating results and also running other processes in the background while the main program is running.

Thread Is a functionality or logic which can execute simultaneously along with the other part of the program. Thread is a light weight process. Any program which is under execution is known as process. We can define the threads in python by overwriting run method of thread class.
Thread class is a predefined class which is defined in threading module.
Thread in module is a predefined module.
If we call the run method directly the logic of the run method will be executed as a normal method logic. In order to execute the logic of the run method as a we use start method of thread class. Example

```
import threading
class x (threading.Thread):
      def run(self):
         for p in range(1, 101):
              print(p)
class y (threading.Thread):
      def run(self):
           for q in range(1, 101):
              print(q)
x1=x()
y1=y()
x1.start()
y1.start()
```

A thread is a lightweight process and multithreading allows us to execute multiple threads at once. As you know, Python is a multithreaded language.

It has a multithreading package. The GIL (Global Interpreter Lock) ensures that a single thread executes at a time. A thread holds the GIL and does a little work before passing it on to the next thread. This makes for an illusion of parallel execution. But in reality, it is just threaded taking turns at the CPU. Of course, all the passing around adds overhead to the execution.

Q. Explain help() and dir() functions in Python.

The help() function displays the documentation string and help for its argument.

```
import copy
help(copy.copy)
```

Help on function copy in module copy: copy(x) Shallow copy operation on arbitrary Python objects. See the module's **doc** string for more info. The dir() function displays all the members of an object(any kind).

```
dir(copy.copy)
```

```
['__annotations__', '__call__', '__class__', '__closure__', '__code__', '__defaults__', '__delattr__', '__dict__',
'__dir__', '__doc__', '__eq__', '__format__', '__ge__', '__get__', '__getattribute__', '__globals__', '__gt__',
'__hash__', '__init__', '__init_subclass__', '__kwdefaults__', '__le__', '__lt__', '__module__', '__name__', '__ne__',
'__new__', '__qualname__', '__reduce__', '__reduce_ex__', '__repr__', '__setattr__', '__sizeof__', '__str__',
'__subclasshook__']
```

Q. What is a dictionary in Python?

The built-in datatypes in Python is called dictionary. It defines one-to-one relationship between keys and values. Dictionaries contain pair of keys and their corresponding values. Dictionaries are indexed by keys.

Let's take an example:

The following example contains some keys. Country, Capital & PM. Their corresponding values are India, Delhi and Modi respectively.

```
dict={'Country':'India','Capital':'Delhi','PM':'Modi'}
```

```
print dict[Country]
```

```
roots={25:5,16:4,9:3,4:2,1:1}
type(roots)
```

<class 'dict;>

```
roots[9]
```

3

A dictionary is mutable, and we can also use a comprehension to create it.

```
roots={x**2:x for x in range(5,0,-1)}
roots
```

{25: 5, 16: 4, 9: 3, 4: 2, 1: 1}

Q. How do you get a list of all the keys in a dictionary?

Be specific in these type of Python Interview Questions and Answers.

For this, we use the function keys().

```
mydict={'a':1,'b':2,'c':3,'e':5}
mydict.keys()
print(dict_keys)
```

(['a', 'b', 'c', 'e'])

Q. Write Python logic to count the number of capital letters in a file.

```
import os
os.chdir('C:\\Users\\lifei\\Desktop')
with open('Today.txt') as today:
    count = 0
    for i in today.read():
        if i.isupper():
            count+=1
    print(count)
```

26

Q. How would you randomize the contents of a list in-place?

For this, we'll import the function `shuffle()` from the module `random`.

```
from random import shuffle
shuffle(mylist)
mylist
```

[3, 4, 8, 0, 5, 7, 6, 2, 1]

Q. Explain join() and split() in Python.

`.join([])` It takes any iterables into this method. Join method is used to concatenate the elements of any list. `join()` lets us join characters from a string together by a character we specify.

```
','.join('12345')
```

'1,2,3,4,5'

`split()` lets us split a string around the character we specify.

```
'1,2,3,4,5'.split(',')
```

['1', '2', '3', '4', '5']

Q. Is Python case-sensitive?

A language is case-sensitive if it distinguishes between identifiers like myname and Myname. In other words, it cares about case- lowercase or uppercase. Let's try this with Python.

```
myname='Ramayan'
Myname
```

Traceback (most recent call last): File "<pyshell#3>", line 1, in Myname NameError: name 'Myname' is not defined

As you can see, this raised a NameError. This means that Python is indeed case-sensitive.

- How long can an identifier be in Python?

In Python, an identifier can be of any length. Apart from that, there are certain rules we must follow to name one:

```
- It can only begin with an underscore or a character from A-Z or a-z.
- The rest of it can contain anything from the following: A-Z/a-z/_/0-9.
- Python is case-sensitive, as we discussed in the previous question.
- Keywords cannot be used as identifiers.
```

Python has the following keywords:

```
and def False import not True
as del finally in or try
assert elif for is pass while
break else from lambda print with
class except global None raise yield
continue exec if nonlocal return
```

Q. How do you remove the leading whitespace in a string?

Leading whitespace in a string is the whitespace in a string before the first non-whitespace character. To remove it from a string, we use the method `lstrip()`.

```
'   Ram '.lstrip()
```

'Ram '

As you can see, this string had both leading and trailing whitespaces. lstrip() stripped the string of the leading whitespace. If we want to strip the trailing whitespace instead, we use rstrip().

```
'   Ram '.rstrip()
```

' Ram'

- How would you convert a string into lowercase?

We use the lower() method for this.

```
'Ramayan'.lower()
```

'ramayan'

To convert it into uppercase, then, we use upper().

```
'Ramayan'.upper()
```

'RAMAYAN'

Also, to check if a string is in all uppercase or all lowercase, we use the methods isupper() and islower().

```
'Ramayan'.isupper()
```

False

```
'Ramayan'.isupper()
```

True

```
'Ramayan'.islower()
```

True

```
'$hrir@m'.islower()
```

True

```
'$HRIR@M'.isupper()
```

True

So, characters like @ and $ will suffice for both cases.

Also, istitle() will tell us if a string is in title case.

```
'Arrested Development'.istitle()
```

```
True
```

Q. What is the pass statement in Python?

There may be times in our code when we haven't decided what to do yet, but we must type something for it to be syntactically correct. In such a case, we use the pass statement.

```
def func(*args):
    pass
```

Similarly, the break statement breaks out of a loop.

```
for i in range(7):
    if i==3: break
        print(i)
```

```
0
1
2
```

Finally, the continue statement skips to the next iteration.

```
for i in range(7):
    if i==3: continue
        print(i)
```

```
0
1
2
4
5
6
```

Q. What is a closure in Python?

A closure is said to occur when a nested function references a value in its enclosing scope. The whole point here is that it remembers the value.

```
def A(x):
    def B():
        print(x)
    return B

A(7)()
```

7

Q. Explain the //, %, and ** operators in Python.

The // operator performs floor division. It will return the integer part of the result on division.

```
7//2

3
```

Normal division would return 3.5 here.

Similarly, ** performs exponentiation. a**b returns the value of a raised to the power b.

```
2**10
```

1024

Finally, % is for modulus. This gives us the value left after the highest achievable division.

```
13 % 7
```

6

```
3.5 % 1.5
```

0.5

Q. How many kinds of operators do we have in Python? Explain arithmetic operators.

This type of Python Interview Questions and Answers can decide your knowledge in Python. Answer the Python Interview Questions with some good Examples.

Here in Python, we have 7 kinds of operators: arithmetic, relational assignment, logical, membership, identity, and bitwise.

We have seven arithmetic operators. These allow us to perform arithmetic operations on values:

Addition (+) This adds two values.

```
7+8
7-8
7*8
7/8  # 0.875
7//8 # 0
```

Q. Explain relational operators in Python.

Relational operators compare values.

- Less than (<) If the value on the left is lesser, it returns True.

```
'hi'<'Hi'
```

False

```
Greater than (>) If the value on the left is greater, it returns True.
```

```
 1.1 + 2.2 > 3.3
```

True

This is because of the flawed floating-point arithmetic in Python, due to hardware dependencies.

```
Less than or equal to (<=) If the value on the left is lesser than or equal to, it returns True.
```

```
3.0 <= 3
```

True

`Greater than or equal to (>=) If the value on the left is greater than or equal to, it returns True.`

```
True >= False
```

True

`Equal to (==) If the two values are equal, it returns True.`

```
{1,3,2,2} == {1,2,3}
```

True

`Not equal to (!=) If the two values are unequal, it returns True.`

```
True!=0.1
```

True

```
False!=0.1
```

True

Q. What are assignment operators in Python?

This one is an Important Interview question in Python Interview.

We can combine all arithmetic operators with the assignment symbol.

```
a = 7
a += 1
a
```

8

```
a -= 1
a
```

7

```
a*=2
a
```

14

```
a/=2
a
```

7.0

```
a**=2
a
```

49.0

```
a//=3
a
```

16.0

```
a%=4
a
```

0.0

Q. Explain logical operators in Python.

We have three logical operators- and, or, not.

```
False and True
```

False

```
 7<7 or True
```

True

```
 not 2==2
```

False

Q. What are membership, operators?

With the operators 'in' and 'not in', we can confirm if a value is a member in another.

```
'me' in 'disappointment'
```

True

```
'us' not in 'disappointment'
```

True

Q. Explain identity operators in Python.

This is one of the very commonly asked Python Interview Questions and answers it with examples. The operators 'is' and 'is not' tell us if two values have the same identity.

```
10 is '10'
```

False

```
True is not False
```

True

Q. Finally, tell us about bitwise operators in Python.

These operate on values bit by bit.

```
AND (&) This performs & on each bit pair.
```

```
 0b110 & 0b010
```

2

```
OR (|) This performs | on each bit pair.
```

```
 3|2
```

3

```
XOR (^) This performs an exclusive-OR operation on each bit pair.
```

```
 3^2
```

1

```
Binary One's Complement (~) This returns the one's complement of a value.
```

```
 ~2
```

-3

```
Binary Left-Shift (<<) This shifts the bits to the left by the specified amount.
```

```
 1<<2
```

4

Here, 001 was shifted to the left by two places to get 100, which is binary for 4.

```
Binary Right-Shift (>>)
```

```
 4>>2
```

1

Q. How would you work with numbers other than those in the decimal number system?

With Python, it is possible to type numbers in binary, octal, and hexadecimal.

```
Binary numbers are made of 0 and 1. To type in binary, we use the prefix 0b or 0B.
```

```
 int(0b1010)
```

10

To convert a number into its binary form, we use bin().

```
 bin(0xf)
```

'0b1111'

```
Octal numbers may have digits from 0 to 7. We use the prefix 0o or 0O.
```

```
 oct(8)
```

'0o10'

```
Hexadecimal numbers may have digits from 0 to 15. We use the prefix 0x or 0X.
```

```
 hex(16)
```

'0x10'

```
 hex(15)
```

'0xf'

Q. Why are identifier names with a leading underscore disparaged?

Since Python does not have a concept of private variables, it is a convention to use leading underscores to declare a variable private. This is why we mustn't do that to variables we do not want to make private.

Q. How can you declare multiple assignments in one statement?

There are two ways to do this:

```
     a,b,c=3,4,5     #This assigns 3, 4, and 5 to a, b, and c resp.
     a = b = c =3         #This assigns 3 to a, b, and c
```

Q. What is tuple unpacking?

First, let's discuss tuple packing. It is a way to pack a set of values into a tuple.

```
mytuple=3,4,5
mytuple
```

(3, 4, 5)

This packs 3, 4, and 5 into mytuple.

Now, we will unpack the values from the tuple into variables x, y, and z.

```
x,y,z=mytuple
x+y+z
```

Q. What data types does Python support?

```
Python provides us with five kinds of data types:
     a=7.0
     title="Ramayan's Book"
     colors=['red','green','blue']
     type(colors)
        <class 'list'>
     name=('Ramayan','Sharma')
     name[0]='Avery'
        Traceback (most recent call last):
        File "<pyshell#129>, line 1, in <module> name[0]='Avery'
        TypeError: 'tuple' object does not support item assignment
     squares={1:1,2:4,3:9,4:16,5:25}
     type(squares)
        <class 'dict'>
     type({})
```

```
        <class 'dict'>
     squares={x:x**2 for x in range(1,6)}
     squares
        {1: 1, 2: 4, 3: 9, 4: 16, 5: 25}
```

Q. What is a docstring?

A docstring is a documentation string that we use to explain what a construct does. We place it as the first thing under a function, class, or a method, to describe what it does. We declare a docstring using three sets of single or double quotes.

```
def sayhi():
    """
    The function prints Hi
    """
    print("Hi")

sayhi()
```

Hi

To get a function's docstring, we use its `__doc__` attribute.

```
sayhi.__doc__
```

'\n\tThis function prints Hi\n\t'

A docstring, unlike a comment, is retained at runtime.

Q. What is the PYTHONPATH variable?

PYTHONPATH is the variable that tells the interpreter where to locate the module files imported into a program. Hence, it must include the Python source library directory and the directories containing Python source code. You can manually set PYTHONPATH, but usually, the Python installer will preset it.

Q. What is slicing?

These are the types of basic Python interview questions for freshers.

Slicing is a technique that allows us to retrieve only a part of a list, tuple, or string. For this, we use the slicing operator [].

```
(1,2,3,4,5)[2:4]
   (3, 4)
[7,6,8,5,9][2:]
   [8, 5, 9]
'Hello'[:-1]
   'Hell'
```

Q. What is a namedtuple?

A namedtuple will let us access a tuple's elements using a name/label. We use the function namedtuple() for this, and import it from collections.

```
from collections import namedtuple
result=namedtuple('result','Physics Chemistry Maths') #format
Ramayan=result(Physics=86,Chemistry=95,Maths=86) #declaring the tuple
Ramayan.Chemistry
```

```
95
```

As you can see, it let us access the marks in Chemistry using the Chemistry attribute of object Ramayan.

Q. How would you declare a comment in Python?

Unlike languages like C++, Python does not have multiline comments. All it has is octothorpe (#). Anything following a hash is considered a comment, and the interpreter ignores it.

```
#line 1 of comment
#line 2 of comment
```

In fact, you can place a comment anywhere in your code. You can use it to explain your code.

Q. How would you convert a string into an int in Python?

If a string contains only numerical characters, you can convert it into an integer using the int() function.

```
int('227')
```

227

Let's check the types:

```
type('227')
```

<class 'str'>

```
type(int('227'))
```

<class 'int'>

Q. How do you take input in Python?

For taking input from user, we have the function input(). In Python 2, we had another function raw_input().

The input() function takes, as an argument, the text to be displayed for the task:

```
a=input('Enter a number')
```

Enter a number7

But if you have paid attention, you know that it takes input in the form of a string.

```
type(a)
```

<class 'str'>

Multiplying this by 2 gives us this:

```
a*=2
a
```

'77'

So, what if we need to work on an integer instead?

We use the int() function for this.

```
 a=int(input('Enter a number'))
```

Enter a number7

Now when we multiply it by 2, we get this:

```
 a*=2
 a
```

```
14
```

Q. What is a frozen set in Python?

Answer these type of Python Interview Questions with Examples.

First, let's discuss what a set is. A set is a collection of items, where there cannot be any duplicates. A set is also unordered.

```
 myset={1,3,2,2}
 myset
```

```
{1, 2, 3}
```

This means that we cannot index it.

```
 myset[0]
```

```
Traceback (most recent call last):
File "<pyshell#197>", line 1, in myset[0]
TypeError: 'set' object does not support indexing
```

However, a set is mutable. A frozen set is immutable. This means we cannot change its values. This also makes it eligible to be used as a key for a dictionary.

```
myset=frozenset([1,3,2,2])
myset
```

frozenset({1, 2, 3})

```
type(myset)
```

<class 'frozenset'>

Q. How would you generate a random number in Python?

This kind of Python interview Questions and Answers can Prove your depth of knowledge.

To generate a random number, we import the function random() from the module random.

```
from random import random
random()
```

```
0.7931961644126482
```

Let's call for help on this.

```
help(random)
```

Help on built-in function random:

`random(…)` method of random.Random instance

`random() -> x` in the interval [0, 1).

This means that it will return a random number equal to or greater than 0, and less than 1.

We can also use the function randint(). It takes two arguments to indicate a range from which to return a random integer.

```
    from random import randint
    randint(2,7)
```

```
6
```

```
    randint(2,7)
```

```
5
```

```
randint(2,7)
```

```
7
```

```
randint(2,7)
```

```
6
```

```
randint(2,7)
```

```
2
```

Q. How will you capitalize the first letter of a string?

Simply using the method capitalize().

```
'Ramayan'.capitalize()
```

```
'Ramayan'
```

```
type(str.capitalize)
```

```
<class 'method_descriptor'>
```

However, it will let other characters be.

```
'$hrir@m'.capitalize()
```

'$HRIR@M'

Q. How will you check if all characters in a string are alphanumeric?

For this, we use the method isalnum().

```
'Ramayan123'.isalnum()
```

True

```
'Ramayan123!'.isalnum()
```

False

Other methods that we have include:

```
'123.3'.isdigit()
```

False

```
'123'.isnumeric()
```

True

```
'Ramayan'.islower()
```

True

```
'Ramayan'.isupper()
```

False

```
'Ramayan'.istitle()
```

True

```
 '   '.isspace()
```

True

```
 '123F'.isdecimal()
```

False

Q. What is the concatenation?

This is very basic Python Interview Question, try not to make any mistake in this.

Concatenation is joining two sequences. We use the + operator for this.

```
 '32'+'32'
```

'3232'

```
 [1,2,3]+[4,5,6]
```

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

```
 (2,3)+(4)
```

```
Traceback (most recent call last):
File "<pyshell#256>", line 1, in <module> (2,3)+(4)
TypeError: can only concatenate tuple (not "int") to tuple
```

Here, 4 is considered an int. Let's do this again.

```
 (2,3)+(4,)  # (obj,) is way to declare single empty
(2, 3, 4)
```

Q. What is a function?

When we want to execute a sequence of statements, we can give it a name. Let's define a function to take two numbers and return the greater number.

```
def greater(a,b):
    return a is a>b else b
greater(3,3.5)
```

```
3.5
```

You can create your own function or use one of Python's many built-in functions.

Q. What is recursion?

When a function makes a call to itself, it is termed recursion. But then, in order for it to avoid forming an infinite loop, we must have a base condition. Let's take an example.

```
def facto(n):
    if n==1: return 1
        return n*facto(n-1)

facto(4)
```

```
24
```

Q. What does the function zip() do?

One of the less common functions with beginners, zip() returns an iterator of tuples.

```
 list(zip(['a','b','c'],[1,2,3]))
```

[('a', 1), ('b', 2), ('c', 3)]

Here, it pairs items from the two lists, and creates tuples with those. But it doesn't have to be lists.

```
list(zip(('a','b','c'),(1,2,3)))
```

```
[('a', 1), ('b', 2), ('c', 3)]
```

Q. If you are ever stuck in an infinite loop, how will you break out of it?

For this, we press Ctrl+C. This interrupts the execution. Let's create an infinite loop to demonstrate this.

```
def counterfunc(n):
    while(n==7):print(n)
counterfunc(7)
```

```
7
7
7
7
.
.
.
.
.
Traceback (most recent call last):
File "<pyshell#332>", line 1, in counterfunc(7)
File "<pyshell#331>", line 2, in counterfunc
while(n==7):print(n)
KeyboardInterrupt
```

Q. With Python, how do you find out which directory you are currently in?

To find this, we use the function/method getcwd(). We import it from the module os.

```
 import os
 os.getcwd()
```

```
'C:\Users\lifei\AppData\Local\Programs\Python\Python36-32'
```

```
type(os.getcwd)
```

<class 'builtin_function_or_method'>

We can also change the current working directory with chdir().

```
os.chdir('C:\\Users\\lifei\\Desktop')
os.getcwd()
```

'C:\Users\lifei\Desktop'

Q. How will you find, in a string, the first word that rhymes with 'cake'?

For our purpose, we will use the function search(), and then use group() to get the output.

```
import re
rhyme=re.search('.ake','I would make a cake, but I hate to bake')
rhyme.group()
```

'make'

And as we know, the function search() stops at the first match. Hence, we have our first rhyme to 'cake'.

Q. What is Tkinter?

Tkinter is a famous Python library with which you can craft a GUI. It provides support for different GUI tools and widgets like buttons, labels, text boxes, radio buttons, and more. These tools and widgets have attributes like dimensions, colors, fonts, colors, and more.

You can also import the tkinter module.

```
import tkinter
top=tkinter.Tk()
```

This will create a new window for you:

This creates a window with the title 'My Game'. You can position your widgets on this.

Follow this link to know more about Python Libraries

Q. How is a .pyc file different from a .py file?

While both files hold bytecode, .pyc is the compiled version of a Python file. It has platform-independent bytecode. Hence, we can execute it on any platform that supports the .pyc format. Python automatically generates it to improve performance(in terms of load time, not speed).

Q. How do you calculate the length of a string?

This is simple. We call the function len() on the string we want to calculate the length of.

```
len('Adi Shakara')
```

Q. What does the following code output?

```
def extendList(val, list=[]):
     list.append(val)
     return list
list1 = extendList(10)
list2 = extendList(123,[])
list3 = extendList('a')
list1,list2,list3
```

Ans. ([10, 'a'], [123], [10, 'a'])

You'd expect the output to be something like this:

([10],[123],['a'])

Well, this is because the list argument does not initialize to its default value ([]) every time we make a call to the function. Once we define the function, it creates a new list. Then, whenever we call it again without a list argument, it uses the same list. This is because it calculates the expressions in the default arguments when we define the function, not when we call it.

Q. What is a decorator? How do I define my own?

Ans. A decorator is a function that adds functionality to another function without modifying it. It wraps another function to add functionality to it. A Python decorator is a specific change that we make in Python syntax to alter functions easily.

```
def decor(func):
    def wrap():
        print("$$$$$$$$$$$$$$$$$")
        func()
            print("$$$$$$$$$$$$$$$$$")
    return wrap

@decor
def sayhi():
    print("Hi")

sayhi()
```

$$$$$$$$$$$$$$$$$ Hi
$$$$$$$$$$$$$$$$$

Decorators are an example of metaprogramming, where one part of the code tries to change another. For more on decorators, read Python Decorators.

Q. Why use function decorators? Give an example.

A decorator is essentially a callable Python object that is used to modify or extend a function or class definition.

One of the beauties of decorators is that a single decorator definition can be applied to multiple functions (or classes). Much can thereby be accomplished with decorators that would otherwise require lots of boilerplate (or even worse redundant!) code.

Flask, for example, uses decorators as the mechanism for adding new endpoints to a web application. Examples of some of the more common uses of decorators include adding synchronization, type enforcement,logging, or pre/post conditions to a class or function.

2. Basic Python Programming Interview Questions

Q. How many arguments can the range() function take?

Ans. The range() function in Python can take up to 3 arguments. Let's see this one by one.

a. One argument

When we pass only one argument, it takes it as the stop value. Here, the start value is 0, and the step value is +1.

```
list(range(5))
```

[0, 1, 2, 3, 4]

```
list(range(-5))
```

[]

```
list(range(0))
```

[]

b. Two arguments

When we pass two arguments, the first one is the start value, and the second is the stop value.

```
list(range(2,7))
```

[2, 3, 4, 5, 6]

```
list(range(7,2))
```

[]

```
list(range(-3,4))
```

[-3, -2, -1, 0, 1, 2, 3]

c. Three arguments

Here, the first argument is the start value, the second is the stop value, and the third is the step value.

```
 list(range(2,9,2))
```

[2, 4, 6, 8]

```
 list(range(9,2,-1))
```

[9, 8, 7, 6, 5, 4, 3]

Q. How do you debug a program in Python? Answer in brief.

Ans. To debug a Python program, we use the module. This is the Python debugger; we will discuss it in a tutorial soon. If we start a program using pdb, it will let us step through the code.

Q. List some pdb commands.

Some pdb commands include-

```
<b> — Add breakpoint
<c> — Resume execution
<s> — Debug step by step
<n> — Move to next line
<l> — List source code
<p> — Print an expression
```

Q. What command do we use to debug a Python program?

Ans. To start debugging, we first open the command prompt, and get to the location the file is at.

Microsoft Windows [Version 10.0.16299.248]

```
(c) 2017 Microsoft Corporation. All rights reserved.

C:\Users\lifei> cd Desktop

C:\Users\lifei\Desktop>
```

Then, we run the following command (for file try.py):

```
C:\Users\lifei\Desktop>python -m pdb try.py

c:\users\lifei\desktop\try.py(1)()
```

Q. What is a Counter in Python?

Ans. The function Counter() from the module 'collections'. It counts the number of occurrences of the elements of a container.

```
from collections import Counter
Counter([1,3,2,1,4,2,1,3,1])
```

```
Counter({1: 4, 3: 2, 2: 2, 4: 1})
```

Python provides us with a range of ways and methods to work with a Counter. Read Python Counter.

Q. What is NumPy? Is it better than a list?

Python Programming Interview Questions - Numpy vs List

Python Programming Interview Questions – Numpy vs List

Ans. NumPy, a Python package, has made its place in the world of scientific computing. It can deal with large data sizes, and also has a powerful N-dimensional array object along with a set of advanced functions.

Yes, a NumPy array is better than a Python list. This is in the following ways:

```
It is more compact.
It is more convenient.
It Is more efficiently.
It is easier to read and write items with NumPy.
```

Q. How would you create an empty NumPy array?

Ans. To create an empty array with NumPy, we have two options:

a. Option 1

```
import numpy
numpy.array([])
```

array([], dtype=float64)

b. Option 2

```
numpy.empty(shape=(0,0))
```

array([], shape=(0, 0), dtype=float64)

Q. Explain the use of the 'nonlocal' keyword in Python.

Ans. First, let's discuss the local and global scope. By example, a variable defined inside a function is local to that function. Another variable defined outside any other scope is global to the function.

Suppose we have nested functions. We can read a variable in an enclosing scope from inside he inner function, but cannot make a change to it. For that, we must declare it nonlocal inside the function. First, let's see this without the nonlocal keyword.

```
def outer():
    a=7
    def inner():
        print(a)
    inner()
outer()
```

7

```
def outer():
   a=7
   def inner():
       print(a)
       a+=1
       print(a)
   inner()

outer()
```

Traceback (most recent call last):
File "<pyshell#462>", line 1, in outer() File "<pyshell#461>", line 7, in outer inner() File "<pyshell#461>", line 4, in inner print(a)
UnboundLocalError: local variable 'a' referenced before assignment

So now, let's try doing this with the 'nonlocal' keyword:

```
def outer():
   a=7
   def inner():
       nonlocal a
       print(a)
       a+=1
       print(a)
```

inner()

```
outer()
```

Q. What is the global keyword?

Ans. Like we saw in the previous question, the global keyword lets us deal with, inside any scope, the global version of a variable.

The problem:

```
a=7
def func():
   print(a)
   a+=1
```

```
    print(a)
```

The solution:

```
a=7
def func():
    global a
    print(a)
    a+=1
    print(a)
func()
```

Q. How would you make a Python script executable on Unix?

Ans. For this to happen, two conditions must be met:

The script file's mode must be executable The first line must begin with a hash(#). An example of this will be: #!/usr/local/bin/python

Q. What functions or methods will you use to delete a file in Python?

Ans. For this, we may use remove() or unlink().

```
import os
os.chdir('C:\\Users\\lifei\\Desktop')
os.remove('try.py')
```

When we go and check our Desktop, the file is gone. Let's go make it again so we can delete it again using unlink().

```
os.unlink('try.py')
```

Both functions are the same, but unlink is the traditional Unix name for it.

Q. What are accessors, mutators, and @property?

Ans. What we call getters and setters in languages like Java, we term accessors and mutators in Python. In Java, if we have a user-defined class with a property 'x', we have methods like getX() and setX(). In Python, we have @property, which is syntactic sugar for property(). This lets us get and set variables without compromising on the conventions. For a detailed explanation on property, refer to Python property.

Q. Differentiate between the append() and extend() methods of a list.

Ans. The methods append() and extend() work on lists. While append()adds an element to the end of the list, extend adds another list to the end of a list.

Let's take two lists.

```
list1,list2=[1,2,3],[5,6,7,8]
```

This is how append() works:

```
list1.append(4)
list1
```

[1, 2, 3, 4]

And this is how extend() works:

```
list1.extend(list2)
list1
```

[1, 2, 3, 4, 5, 6, 7, 8]

Q. What do you mean by overriding methods?

Ans. Suppose class B inherits from class A. Both have the method sayhello()- to each, their own version. B overrides the sayhello() of class A. So, when we create an object of class B, it calls the version that class B has.

```
class A:
    def sayhello(self):
        print("Hello, I'm A")
class B(A):
    def sayhello(self):
        print("Hello, I'm B")
a=A()
b=B()
a.sayhello()
```

Hello, I'm A

```
b.sayhello()
```

Hello, I'm B

Q. What is JSON? Describe in brief how you'd convert JSON data into Python data?

Ans. JSON stands for JavaScript Object Notation. It is a highly popular data format, and it stores data into NoSQL databases. JSON is generally built on the following two structures:

```
A collection of <name,value> pairs
An ordered list of values.
```

Python supports JSON parsers. In fact, JSON-based data is internally represented as a dictionary in Python. To convert JSON data into Python data, we use the load() function from the JSON module.

Q. How do you execute a Python Script?

From the command line, type python .py or pythonx.y .py where the x.y is the version of the Python interpreter desired. Learn how to use Python, from beginner basics to advanced techniques, with online video tutorials taught by industry experts. Enroll for Free Python Training Demo!

Q. Explain the use of try: except: raise, and finally:

Try, except and finally blocks are used in Python error handling. Code is executed in the try block until an error occurs. One can use a generic except block, which will receive control after all errors, or one can use specific exception handling blocks for various error types. Control is transferred to the appropriate except block. In all cases, the finally block is executed. Raise may be used to raise your own exceptions.

Q. Illustrate the proper use of Python error handling.

Code Example:

```
try:
    ….#This can be any code
except:
    …# error handling code goes here
finally:
    …# code that will be executed regardless of exception handling goes here.
```

Q. What is a 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.

A namespace is a collection of names. It maps names to corresponding objects. When different namespaces contain objects with the same names, this avoids any name collisions. Internally, a namespace is implemented as a Python dictionary.

On starting the interpreter, it creates a namespace for as long as we don't exit. We have local namespaces, global namespaces, and a built-in namespace.

Q. Explain the differences between local and global namespaces.

Local namespaces are created within a function. when that function is called. Global name spaces are created when the program starts.

Q. Name the four main types of namespaces in Python?

Global, Local, Module and Class namespaces.

Q. When would you use triple quotes as a delimiter?

Triple quotes """ or ''' are string delimiters that can span multiple lines in Python. Triple quotes are usually used when spanning multiple lines, or enclosing a string that has a mix of single and double quotes contained therein.

Q. How to use GUI that comes with Python to test your code?

That is just an editor and a graphical version of the interactive shell. You write or load code and run it, or type it into the shell. There is no automated testing.

Q. How does the Python version numbering scheme work?

Python versions are numbered A.B.C or A.B.

A is the major version number. It is only incremented for major changes in the language.

B is the minor version number, incremented for less earth-shattering changes.

C is the micro-level. It is incremented for each bug fix release. Not all releases are bug fix releases.

In the run-up to a new major release, 'A' series of development releases are made denoted as alpha, beta, or release candidate.

Alphas are early releases in which interfaces aren't finalized yet; it's not unexpected to see an interface change between two alpha releases.

Betas are more stable, preserving existing interfaces but possibly adding new modules, and release candidates are frozen, making no changes except as needed to fix critical bugs.

Alpha, beta and release candidate versions have an additional suffix.

The suffix for an alpha version is "aN" for some small number N,

The suffix for a beta version is "bN" for some small number N, And the suffix for a release candidate version is "cN" for some small number N.

In other words, all versions labeled 2.0aN precede the versions labeled 2.0bN, which precede versions labeled 2.0cN, and those precede 2.0.

You may also find version numbers with a "+" suffix, e.g. "2.2+". These are unreleased versions, built directly from the subversion trunk. In practice, after a final minor release is made, the subversion trunk is incremented to the next minor version, which becomes the "a0" version, e.g. "2.4a0".

Q. Where is math.py (socket.py, regex.py, etc.) source file?

If you can't find a source file for a module, it may be a built-in or dynamically loaded module implemented in C, C++ or other compiled language. In this case you may not have the source file or it may be something like mathmodule.c, somewhere in a C source directory (not on the Python Path). There are (at least) three kinds of modules in Python:

- Modules written in Python (.py);
- Modules written in C and dynamically loaded (.dll, .pyd, .so, .sl, etc);
- Modules written in C and linked with the interpreter; to get a list of these, type; Import sys print sys.builtin_module_names;

Q. How do I make a Python script executable on UNIX?

You need to do two things: The script file's mode must be executable and the first line must begin with "#!" followed by the path of the Python interpreter.

- The first is done by executing chmod +x scriptfile or perhaps chmod 755 'script' file.
- The second can be done in a number of ways.

The most straightforward way is to write:
```
#!/usr/local/bin/python
```

as the very first line of your file, using the pathname for where the Python interpreter is installed on your platform. If you would like the script to be independent of where the Python interpreter lives, you can use the "env" program. Almost all UNIX variants support the following, assuming the python interpreter is in a directory on the users $PATH:

```
#! /usr/bin/env python
```

Don't do this for CGI scripts. The **$PATH** variable for CGI scripts is often minimal, so you need to use the actual absolute pathname of the interpreter. Occasionally, a user's environment is so full that the /usr/bin/env program fails; or there's no env program at all. In that case, you can try the following hack (due to Alex Rezinsky):

```
#! /bin/sh
""":"
exec python $0 ${1+"$@"}
"""
```

The minor disadvantage is that this defines the script's `__doc__`string. However, you can fix that by adding:
`__doc__ = """…Whatever…"""`

Q. Why do not my signal handlers work?

The most common problem is that the signal handler is declared with the wrong argument list. It is called as: handler (signum, frame) So it should be declared with two arguments: def handler(signum, frame):

Q. How do I find undefined g++ symbols __builtin_new or __pure_virtual?

To dynamically load g++ extension modules, you must: Recompile Python Re-link it using g++ (change LINKCC in the python Modules Makefile) Link your extension module using g++ (e.g., "g++ -shared -o mymodule.so mymodule.o").

Q. How do I send mail from a Python script?

Use the standard library module smtplib. Here's a very simple interactive mail sender that uses it. This method will work on any host that supports an SMTP listener.

```
import sys, smtplib
fromaddr = raw_input("From: ")
toaddrs = raw_input("To: ").split(',')
print "Enter message, end with ^D:"
msg = "
while 1:
    line = sys.stdin.readline()
    if not line:
        break

msg = msg + line

# The actual mail send
server = smtplib.SMTP('localhost')
server.sendmail(fromaddr, toaddrs, msg)
server.quit()
```

A UNIX-only alternative uses send mail. The location of the send mail program varies between systems; sometimes it is /usr/lib/sendmail, sometime /usr/sbin/sendmail. The send mail manual page will help you out. Here's some sample code:

```
SENDMAIL = "/usr/sbin/sendmail" # sendmail location
import os
p = os.popen("%s -t -i" % SENDMAIL, "w")
p.write("To: receiver@example.comn")
p.write("Subject: testn")
p.write("n") # blank line separating headers from body
p.write("Some textn")
p.write("some more textn")
sts = p.close()
if sts != 0:
    print ("Sendmail exit status", sts)
```

Q. How can I mimic CGI form submission (METHOD=POST)? I would like to retrieve web pages that are the result of posting a form.

Yes. Here is a simple example that uses httplib:

```
#!/usr/local/bin/python
import httplib, sys, time

### build the query string
qs = "First=Josephine&MI=Q&Last=Public"

### connect and send the server a path
httpobj = httplib.HTTP('www.some-server.out-there', 80)
httpobj.putrequest('POST', '/cgi-bin/some-cgi-script')

### now generate the rest of the HTTP headers…
httpobj.putheader('Accept', '*/*')
httpobj.putheader('Connection', 'Keep-Alive')
httpobj.putheader('Content-type', 'application/x-www-form-urlencoded')
httpobj.putheader('Content-length', '%d' % len(qs))
httpobj.endheaders()
httpobj.send(qs)

### find out what the server said in response…
reply, msg, hdrs = httpobj.getreply()
if reply != 200:
sys.stdout.write(httpobj.getfile().read())
```

Note that in general for URL-encoded POST operations, query strings must be quoted by using urllib.quote(). For example to send name="Guy Steele, Jr.":

```
from urllib import quote
x = quote("Guy Steele, Jr.")
print(x)
'Guy%20Steele,%20Jr.'
 query_string = "name="+x
 query_string
'name=Guy%20Steele,%20Jr.'
```

Q. Why is that none of my threads are not running? How can I make it work?

As soon as the main thread exits, all threads are killed. Your main thread is running too quickly, giving the threads no time to do any work. A simple fix is to add a sleep to the end of the program that's long enough for all the threads to finish:

```
import threading, time
def thread_task(name, n):
for i in range(n): print name, i
for i in range(10)
```

Q. What Are The Implementation In Python Program?

Python program can be implemented by two ways

```
1. Interactive Mode (Submit statement by statement explicitly)
2. Batch Mode (Writing all statements and submit all statements)
```

In Interactive mode python command shell is required. It is available in installation of python cell.

In Interactive mode is not suitable for developing the projects & Applications

Interactive mode is used for predefined function and programs. Example:

```
X=1000
Y=2000
X+Y
3000
Quit(X+Y)
```

X, Y is not find. Interactive mode is unfit for looping purpose.

Interactive Mode: The concept of submitting one by one python statements explicitly in the python interpreter is known as "Interactive Mode" In Order to submit the one by one python statements explicitly to the python interpreter we use python command line shell. Python command line shell is present in python software We can open the python command line shell by executing python command on command prompt or terminal Example:
c:/users/mindmajix>python
```
3+4
```
7

```
'mindmajix'*3
```
'mindmajix mindmajix mindmajix'

```
x=1000
y=2000
x+y
```
3000
Quit

x+y
c:/users/sailu>python Error: name 'X' not defined

Batch Mode:
In the concept of writing the group of python statements in a file, save the file with extension .py and submit that entire file to the python interpreter is known as Batch Mode.

In Order to develop the python files we use editors or IDE's Different editors are notepad, notepad++, edit+,nano, VI, gedil and so on.
Open the notepad and write the following code:

Example:

```
X=1000
Y=2000
print(x+y, x-y, x*y)
```

Save the file in D drive mindmajix python folder with the demo.py Open command prompt and execute following commands:

Python D:/mindmajix python/Demo.py

3000

-1000

2000.000

Save another method if we correctly set the path

D:

D:/>cd "mindmajix python"

D:/mindmajix Python>python Demo.py
```
3000
-1000
2000.000
```

Q. What are The Data Types Supports in Python Language?

- Numbers- Numbers use to hold numerical values.
- Strings- A string is a sequence of characters. We declare it using single or double quotes.
- Lists- A list is an ordered collection of values, and we declare it using square brackets.
- Tuples- A tuple, like a list, is an ordered collection of values. The difference. However, is that a tupleis immutable. This means that we cannot change a value in it.
- Dictionary- A dictionary is a data structure that holds key-value pairs. We declare it using curly braces.
- We can also use a dictionary comprehension:

Numbers: Int Float Complex Boolean Operational: Strings List Tuple Set Dictionary

Every data type in python language is internally implemented as a class. Python language data types are categorized into two types.

They are:

Fundamental Types, Collection Types

Q. Explain Control flow statements.

By default python program execution starts from first line, execute each and every statements only once and transactions the program if the last statement of the program execution is over. Control flow statements are used to disturb the normal flow of the execution of the program.

Q. What are the two major loop statements?

`for and while`

Q. Under what circumstances would one use a `while` statement rather than `for`?

The while statement is used for simple repetitive looping and the for statement is used when one wishes to iterate through a list of items, such as database records, characters in a string, etc.

Q. What happens if output an `else` statement after after block?

The code in the else block is executed after the for loop completes, unless a break is encountered in the for loop execution. in which case the else block is not executed.

Q. Explain the use of break and continue in Python looping.

The break statement stops execution of the current loop. and transfers control to the next block. The continue statement ends the current block's execution and jumps to the next iteration of the loop.

Q. When would you use a continue statement in a for loop?

When processing a particular item was complete; to move on to the next, without executing further processing in the block. The continue statement says, "I'm done processing this item, move on to the next item."

Q. When would you use a break statement in a for loop?

When the loop has served its purpose. As an example. after finding the item in a list searched for, there is no need to keep looping. The break statement says, I'm done in this loop; move on to the next block of code."

Q. What is the structure of a for loop?

for in : … The ellipsis represents a code block to be executed, once for each item in the sequence. Within the block the item is available as the current item from the entire list.

Q. What is the structure of a while loop?

while : … The ellipsis represents a code block to be executed. until the condition becomes false. The condition is an expression that is considered true unless it evaluates to o, null or false.

Q. Use a for loop and illustrate how you would define and print the characters in a string out, one per line.

```
myString = "I Love Python"
```

```
for myChar hi myString:
    print(myChar)
```

Q. Given the string "I LoveQPython" use afor loop and illustrate printing each character tip to, but not including the Q.

```
inyString = "I Love Pijtlzon"
for myCizar in myString:
    fmyC'har ==
    break
print(myChar)
```

Q. Given the string "I Love Python" print out each character except for the spaces, using a for loop.

```
inyString = I Love Python"
for myCizar in myString:
fmyChar == '' '':
continue
print myChar
```

Q. What is a Tuple?

- Tuple Objects can be created by using parenthesis or by calling tuple function or by assigning multiple values to a single variable
- Tuple objects are immutable objects
- Incision order is preserved
- Duplicate elements are allowed
- Heterogeneous elements are allowed
- Tuple supports both positive and negative indexing
- The elements of the tuple can be mutable or immutable

```
    #Example:
    x=()
    print(x)
    print(type(x))
```

```
print(len(x))
y-tuple()
print(y)
print(type(y))
print(len(y))
z=10,20
print(z)
print(type(z))
print(len(z))
p=(10,20,30,40,50,10,20,10) Insertion & duplicate
print(p)
q=(100, 123.123, True, "mindmajix") Heterogeneous
print(q)
Output:
```

Q. What is the Dictionary?

- Dictionary objects can be created by using curly braces{} or by calling dictionary function.
- Dictionary objects are mutable objects.
- Dictionary represents key value base.
- Each key value pair of Dictionary is known as a item.
- Dictionary keys must be immutable.
- Dictionary values can be mutable or immutable.
- Duplicate keys are not allowed but values can be duplicate.
- Insertion order is not preserved.
- Heterogeneous keys and heterogeneous values are allowed.

Q. How to Search Path of Modules?

By default python interpreter search for the imported modules in the following locations:

- Current directory (main module location)
- Environment variable path
- Installation dependent directory
- If the imported module is not found in the any one of the above locations. Then python interpreter giveserror.
- Built-in attributes of a module:

- By default for each and every python module some properties are added internally and we call thoseproperties as a built-in-attribute of a module

Q. What are the Packages?

Package is nothing but a folder or dictionary which represents collection of modules.

A package can also contain sub packages.

We can import the modules of the package by using package name.module name or name.subpackage name.module name

Q. What is File Handling?

File is a named location on the disk, which stores the data in permanent manner. Python language provides various functions and methods to provide the communication between python programs and files. Python programs can open the file, perform the read or write operations on the file and close the file We can open the files by calling open function of built-in-modules At the time of opening the file, we have to specify the mode of the file Mode of the file indicates for what purpose the file is going to be opened(r,w,a,b)

Q. What are the Runtime Errors?

The errors which occurs after starting the execution of the programs are known as runtime errors. Runtime errors can occur because of:

```
Invalid Input
Invalid Logic
Memory issues
Hardware failures and so on
```

With respect to every reason which causes to runtime error correspoing runtime error representation class is available Runtime error representation classes technically we call as a exception classes. While executing the program if any runtime error is occur corresponding runtime error representation class object is created Creating runtime error representation class object is technically known as a rising exception While executing the program if any exception is raised, then internally python interpreter verify any code is implemented to handle raised exception or not If code is not implemented to handle raised exception then program will be terminated abnormally

Q. What is Abnormal Termination?

The concept of terminating the program in the middle of its execution without executing last statement of the main module is known as a abnormal termination Abnormal termination is undesirable situation in programming languages.

Q. What is `try` Block?

A block which is preceded by the try keyword is known as a try block Syntax: try{ //statements that may cause an exception }

The statements which causes to run time errors and other statements which depends on the execution of run time errors statements are recommended to represent in try block While executing try block statement if any exception is raised then immediately try block identifies that exception, receive that exception and forward that exception to except block without executing remaining statements to try block.

Q. What is the Difference Between Methods & Constructors?

Methods

- Method name can be any name.
- With respect to one object one method can be called for 'n' members of lines
- Methods are used to represent business logic to perform the operations

Constructor

- Constructor will be executed automatically whenever we create a object.
- With respect to one object one constructor can be executed only once
- Constructors are used to define & initialize the non static variable

Q. What is the Encapsulation?

The concept of binding or grouping related data members along with its related functionalities is known as a Encapsulation.

Q. Executing DML Commands Through Python Programs?

DML (Data Modification Language) Commands are used to modify the data of the database objects Whenever we execute DML Commands the records are going to be modified temporarily.

Whenever we run "rollback" command the modified records will come back to its original state.

To modify the records of the database objects permanently we use "commit" command

After executing the commit command even though we execute "rollback" command, the modified records will not come back to its original state.

Create the emp1 table in the database by using following command

```
Create table emp1 as select * from emp;
```

Whenever we run the DML commands through the python program, then the no.of records which are modified because of that command will be stored into the rowcount attribute of cursor object.
After executing the DML Command through the python program we have to call commit method of cursor object.

Q. What is Threads Life Cycle?

Threads Life Cycle

- Creating the object of a class which is overwriting run method of thread class is known as a creating thread.
- Whenever thread is created then we call thread is in new state or birth state thread.
- Whenever we call the start method on the new state threads then those threads will be forwarded for scheduling.
- The threads which are forwarded for scheduling are known as ready state threads
- Whenever scheduling time occurs, ready state thread starts execution.
- The threads which are executing are known as running state threads Whenever sleep fun or join methods are called on the running state threads then immediately those threads will wait.

- The threads which are waiting are known as waiting state threads Whenever waiting time is over or specified thread execution is over - then immediately waiting state threads are forwarded for scheduling.
- If running state threads execution is over then immediately those threads execution will be terminated -The threads which execution is terminated are known as dead state threads.

Q. What is scheduling?

Among multiple threads:

- which thread as to start the execution first,
- How much time the thread as to execute after allocated time is over,
- which thread as to continue the execution next this comes under scheduling. Scheduling is highly dynamic

Q. for loop is implemented in python language as follows:

```
for element in iterable:
    iter-obj=iter(iterable)
    while true:
        try:
            element=next(iter_obj)
        except(slop iteration)
            break
```

For loop takes the given object, convert that object in the form of iterable object & gets the one by one element form the iterable object.

While getting the one by value element from the iterable object if stop iteration exception is raised then for loop internally handle that exception

Q. OS Module

OS Module is a predefined module and which provides various functions and methods to perform the operating system related activities, such as creating the files, removing the files, creating the directories removing the directories, executing the operating system related commands, etc.

Example:

```
import os
cwd=os.getwd()
print("1", cwd)
os.chdir("samples")
print("2", os.getcwd())
os.chdir(os.pardir)
print("3",os.getcwd())
```

Output:

Q. What Are Applications of Python?

Applications of Python

- Automation App
- Data Analytics
- Scientific App
- Web App
- Web Scrapping
- Test Cases
- Network with IOT
- Admin Script
- GUI
- Gaming
- Animation

Q. 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.

Q. 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.

Q. 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.

Q. In Python what are iterators?

In Python, iterators are used to iterate a group of elements, containers like list.

Q. 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.

Q. 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.
Python generator produces a sequence of values to iterate on. This way, it is kind of an iterable. We define a function that 'yields' values one by one, and then use a for loop to iterate on it.

```
def squares(n):
    i=1
    while(i<=n):
        yield i**2
        i+=1
for i in squares(7):
    print(i)
```

```
1
4
9
16
25
36
49
```

Q. So, what is an iterator, then?

An iterator returns one object at a time to iterate on. To create an iterator, we use the iter() function.

```
odds=iter([1,3,5])
```

Then, we call the next() function on it every time we want an object.

```
 next(odds)
```

```
1
```

```
 next(odds)
```

```
3
```

```
 next(odds)
```

```
5
```

And now, when we call it again, it raises a StopIteration exception. This is because it has reached the end of the values to iterate on.

```
 next(odds)
```

```
Traceback (most recent call last):
File "<pyshell#295>", line 1, in <module> next(odds)
StopIteration
```

Q. Explain generators and iterators in python?

- They do, but there are subtle differences:
- For a generator, we create a function. For an iterator, we use in-built functions `iter()` and `next()`.
- For a generator, we use the keyword 'yield' to yield/return an object at a time.
- A generator may have as many 'yield' statements as you want.
- A generator will save the states of the local variables every time 'yield' will pause the loop.
- An iterator does not use local variables; it only needs an iterable to iterate on.
- Using a class, you can implement your own iterator, but not a generator.
- Generators are fast, compact, and simpler.
- Iterators are more memory-efficient.

Q. 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.

Q. 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().

Q. 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.

Q. Mention what are the rules for local and global variables in Python?

Local variables: If a variable is assigned a new value anywhere within the function's body, it's assumed to be local.

Global variables: Those variables that are only referenced inside a function are implicitly global.

Q. How can you share global variables across modules?

To share global variables across modules within a single program, create a special module. Import the config module in all modules of your application. The module will be available as a global variable across modules.

Q. Explain how can you make a Python Script executable on Unix?

To make a Python Script executable on Unix, you need to do two things,

```
Script file's mode must be executable and
the first line must begin with # ( #!/usr/local/bin/python)
```

Q. Explain how to delete a file in Python?

By using a command os.remove (filename) or os.unlink(filename)

Q. Explain how can you generate random numbers in Python?

To generate random numbers in Python, you need to import command as

```
import random
random.random()
```

This returns a random floating point number in the range (0,1)

- Explain how can you access a module written in Python from C?

You can access a module written in Python from C by following method,

Module = =PyImport_ImportModule("");

Q. Mention the use of // operator in Python?

It is a Floor Divisionoperator , which is used for dividing two operands with the result as quotient showing only digits before the decimal point. For instance, 10//5 = 2 and 10.0//5.0 = 2.0.

Q. Mention five benefits of using Python?

- Python comprises of a huge standard library for most Internet platforms like Email, HTML, etc.
- Python does not require explicit memory management as the interpreter itself allocates the memory to newvariables and free them automatically
- Provide easy readability due to use of square brackets
- Easy-to-learn for beginners
- Having the built-in data types saves programming time and effort from declaring variables

Q. Mention the use of the split function in Python?

The use of the split function in Python is that it breaks a string into shorter strings using the defined separator. It gives a list of all words present in the string.

Q. Mention what is the difference between Django, Pyramid, and Flask?

Flask is a "microframework" primarily build for a small application with simpler requirements. In flask, you have to use external libraries. Flask is ready to use.

Pyramid are build for larger applications. It provides flexibility and lets the developer use the right tools for their project. The developer can choose the database, URL structure, templating style and more. Pyramid is heavy configurable.

Like Pyramid, Django can also used for larger applications. It includes an ORM.

- You are having multiple Memcache servers running Python, in which one of the memcacher server fails, and it has your data, will it ever try to get key data from that one failed server?

The data in the failed server won't get removed, but there is a provision for auto-failure, which you can configure for multiple nodes. Fail-over can be triggered during any kind of socket or Memcached server level errors and not during normal client errors like adding an existing key, etc.

Q. Explain how you can minimize the Memcached server outages in your Python Development?

- When one instance fails, several of them goes down, this will put larger load on the database server when lost data is reloaded as client make a request. To avoid this, if your code has been written to minimize cache stampedes then it will leave a minimal impact
- Another way is to bring up an instance of Memcached on a new machine using the lost machines IP address
- Code is another option to minimize server outages as it gives you the liberty to change the Memcached server list with minimal work
- Setting timeout value is another option that some Memcached clients implement for Memcached server outage. When your Memcached server goes down, the client will keep trying to send a request till the time-out limit is reached

Q. Explain what is Dogpile effect? How can you prevent this effect?

Dogpile effect is referred to the event when cache expires, and websites are hit by the multiple requests made by the client at the same time.
This effect can be prevented by using semaphore lock. In this system when value expires, first process acquires the lock and starts generating new value.

Q. Explain how Memcached should not be used in your Python project?

Memcached common misuse is to use it as a data store, and not as a cache. Never use Memcached as the only source of the information you need to run your application. Data should always be available through another source as well. Memcached is just a key or value store and cannot perform query over the data or iterate over the contents to extract information.

Memcached does not offer any form of security either in encryption or authentication

Q. What is List Comprehensions feature of Python used for?

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 for clause, then zero or more for or if clauses.

Q. What are lambda expressions, list comprehensions and generator expressions?

Lambda expressions

are a shorthand technique for creating single line, anonymous functions. Their simple, inline nature often – though not always – leads to more readable and concise code than the alternative of formal function declarations. On the other hand, their terse inline nature, by definition, very much limits what they are capable of doing and their applicability. Being anonymous and inline, the only way to use the same lambda function in multiple locations in your code is to specify it redundantly.

List comprehensions

provide a concise syntax for creating lists. List comprehensions are commonly used to make lists where each element is the result of some operation(s) applied to each member of another sequence or iterable. They can also be used to create a subsequence of those elements whose members satisfy a certain condition. In Python, list comprehensions provide an alternative to using the built-in map() and filter() functions.

As the applied usage of lambda expressions and list comprehensions can overlap, opinions vary widely as to when and where to use one vs. the other. One point to bear in mind, though, is that a list comprehension executes somewhat faster than a comparable solution using map and lambda (some quick tests yielded a performance difference of roughly 10%). This is because calling a lambda function creates a new stack frame while the expression in the list comprehension is evaluated without doing so.

Generator expressions

are syntactically and functionally similar to list comprehensions but there are some fairly significant differences between the ways the two operate and, accordingly, when each should be used. In a nutshell, iterating over a generator expression or list comprehension will essentially do the same thing, but the list comprehension will create the entire list in memory first while the generator expression will create the items on the fly as needed. Generator expressions can therefore be used for very large (and even infinite) sequences and their lazy (i.e., on demand) generation of values results in improved performance and lower memory usage. It is worth noting, though, that the standard Python list methods can be used on the result of a list comprehension, but not directly on that of a generator expression.

- Consider the two approaches below for initializing an array and the arrays that will result. How will the resulting arrays differ and why should you use one initialization approach vs. the other?

```
 # INITIALIZING AN ARRAY -- METHOD 1
...
 x = [[1,2,3,4]] * 3
 x
[[1, 2, 3, 4], [1, 2, 3, 4], [1, 2, 3, 4]]


 # INITIALIZING AN ARRAY -- METHOD 2
...
 y = [[1,2,3,4] for _ in range(3)]
 y
[[1, 2, 3, 4], [1, 2, 3, 4], [1, 2, 3, 4]]

 # WHICH METHOD SHOULD YOU USE AND WHY?
```

Ans:

While both methods appear at first blush to produce the same result, there is an extremely significant difference between the two. Method 2 produces, as you would expect, an array of 3 elements, each of which is itself an independent 4-element array. In method 1, however, the members of the array all point to the same object. This can lead to what is most likely unanticipated and undesired behavior as shown below.

```
# MODIFYING THE x ARRAY FROM THE PRIOR CODE SNIPPET:
x[0][3] = 99
x
[[1, 2, 3, 99], [1, 2, 3, 99], [1, 2, 3, 99]]

# UH-OH, DON'T THINK YOU WANTED THAT TO HAPPEN!
...
# MODIFYING THE y ARRAY FROM THE PRIOR CODE SNIPPET:
y[0][3] = 99
y
[[1, 2, 3, 99], [1, 2, 3, 4], [1, 2, 3, 4]]
# THAT'S MORE LIKE WHAT YOU EXPECTED!
...
```

Q. What will be printed out by the second append() statement below?

```
def append(list=[]):
    # append the length of a list to the list
    list.append(len(list))
    return list

append(['a','b'])
['a', 'b', 2]
```

append() # calling with no arg uses default list value of [][0]

append() # but what happens when we AGAIN call append with no arg?

Ans:

When the default value for a function argument is an expression, the expression is evaluated only once, not every time the function is called. Thus, once the list argument has been initialized to an empty array, subsequent calls to append without any argument specified will continue to use the same array to which list was originally initialized. This will therefore yield the following, presumably unexpected, behavior:

append() # first call with no arg uses default list value of [][0] append() # but then look what happens...[0, 1] append() # successive calls keep extending the same default list! [0, 1, 2] append() # and so on, and so on, and so on... [0, 1, 2, 3]

- How might one modify the implementation of the `append` method in the previous question to avoid the undesirable behavior described there?

The following alternative implementation of the append method would be one of a number of ways to avoid the undesirable behavior described in the answer to the previous question:

```
def append(list=None):
    if list is None:
        list = []
        # append the length of a list to the list
        list.append(len(list))
        return list
append()
[0]
append()
[0]
```

Q: How can you swap the values of two variables with a single line of Python code?

Consider this simple example:

```
 x = 'X'
 y = 'Y'
```

In many other languages, swapping the values of x and y requires that you to do the following:

```
tmp = x
x = y
y = tmp
x, y
('Y', 'X')
```

But in Python, makes it possible to do the swap with a single line of code (thanks to implicit tuple packing and unpacking) as follows:

```
x,y = y,x
x,y
('Y', 'X')
```

Q. What will be printed out by the last statement below?

```
flist = []
for i in range(3):
    flist.append(lambda: i)

[f() for f in flist]   # what will this print out?
```

In any closure in Python, variables are bound by name. Thus, the above line of code will print out the following:
[2, 2, 2]
Presumably not what the author of the above code intended?

A workaround is to either create a separate function or to pass the args by name; e.g.

```
flist = []
for i in range(3):
    flist.append(lambda i = i : i)

[f() for f in flist]
[0, 1, 2]
```

Q. Is Python interpreted or compiled?

As noted in Why Are There So Many Pythons?, this is, frankly, a bit of a trick question in that it is malformed. Python itself is nothing more than an interface definition (as is true with any language specification) of which there are multiple implementations. Accordingly, the question of whether "Python" is interpreted or compiled does not apply to the Python language itself; rather, it applies to each specific implementation of the Python specification.

Further complicating the answer to this question is the fact that, in the case of CPython (the most common Python implementation), the answer really is "sort of both". Specifically, with CPython, code is first compiled and then interpreted. More precisely, it is not precompiled to native machine code, but rather to bytecode. While machine code is certainly faster, bytecode is more portable and secure. The bytecode is then interpreted in the case of CPython (or both interpreted and compiled to optimized machine code at runtime in the case of PyPy).

Q. What are some alternative implementations to CPython? When and why might you use them?

One of the more prominent alternative implementations is Jython, a Python implementation written in Java that utilizes the Java Virtual Machine (JVM). While CPython produces bytecode to run on the CPython VM, Jython produces Java bytecode to run on the JVM.

Another is IronPython, written in C# and targeting the .NET stack. IronPython runs on Microsoft's Common Language Runtime (CLR).

As also pointed out in Why Are There So Many Pythons?, it is entirely possible to survive without ever touching a non-CPython implementation of Python, but there are advantages to be had from switching, most of which are dependent on your technology stack.

Another noteworthy alternative implementation is PyPy whose key features include:

- Speed. Thanks to its Just-in-Time (JIT) compiler, Python programs often run faster on PyPy.
- Memory usage. Large, memory-hungry Python programs might end up taking less space with PyPy than they do in CPython.
- Compatibility. PyPy is highly compatible with existing python code. It supports cffi and can run popular Python libraries like Twisted and Django.
- Sandboxing. PyPy provides the ability to run untrusted code in a fully secure way.
- Stackless mode. PyPy comes by default with support for stackless mode, providing micro-threads for massive concurrency.

Q. What is unittest in Python? What's your approach to unit testing 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.

The most fundamental answer to this question centers around Python's unittest testing framework. Basically, if a candidate doesn't mention unittest when answering this question, that should be a huge red flag.

`unittest` supports test automation, sharing of setup and shutdown code for tests, aggregation of tests into collections, and independence of the tests from the reporting framework. The unittest module provides classes that make it easy to support these qualities for a set of tests.

Assuming that the candidate does mention unittest (if they don't, you may just want to end the interview right then and there!), you should also ask them to describe the key elements of the unittest framework; namely, test fixtures, test cases, test suites and test runners.

A more recent addition to the unittest framework is mock. mock allows you to replace parts of your system under test with mock objects and make assertions about how they are to be used. mock is now part of the Python standard library, available as unittest.mock in Python 3.3 onwards.

The value and power of mock are well explained in An Introduction to Mocking in Python. As noted therein, system calls are prime candidates for mocking: whether writing a script to eject a CD drive, a web server which removes antiquated cache files from /tmp, or a socket server which binds to a TCP port, these calls all feature undesired side-effects in the context of unit tests. Similarly, keeping your unit-tests efficient and performant means keeping as much "slow code" as possible out of the automated test runs, namely filesystem and network access.

Q. How would you perform unit-testing on your Python code?

Ans. For this purpose, we have the module unittest in Python. It has the following members:

- FunctionTestCase
- SkipTest
- TestCase
- TestLoader
- TestResult
- TestSuite

- TextTestResult
- TextTestRunner
- defaultTestLoader
- expectedFailure
- findTestCases
- getTestCaseNames
- installHandler
- main
- makeSuite
- registerResult
- removeHandler
- removeResult
- skip
- skipIf
- skipUnless

Below are some Advanced Python Programming Interview Questions For Experienced. I recommend freshers to also refer these interview questions for advanced knowledge.

Q. How do I test a Python program or component?

Python comes with two testing frameworks: The documentation test module finds examples in the documentation strings for a module and runs them, comparing the output with the expected output given in the documentation string.

The unittest moduleis a fancier testing framework modeled on Java and Smalltalk testing frameworks.

For testing, it helps to write the program so that it may be easily tested by using good modular design. Your program should have almost all functionality encapsulated in either functions or class methods. And this sometimes has the surprising and delightful effect of making the program run faster because local variable accesses are faster than global accesses.

Furthermore the program should avoid depending on mutating global variables, since this makes testing much more difficult to do. The "global main logic" of your program may be as simple as:

if **name**=="**main**":
main_logic()

at the bottom of the main module of your program. Once your program is organized as a tractable collection of functions and class behaviors, you should write test functions that exercise the behaviors.

A test suite can be associated with each module which automates a sequence of tests.

You can make coding much more pleasant by writing your test functions in parallel with the "production code", since this makes it easy to find bugs and even design flaws earlier.

"Support modules" that are not intended to be the main module of a program may include a self-test of the module.

if **name** == "**main**": self_test()

Even programs that interact with complex external interfaces may be tested when the external interfaces are unavailable by using "fake" interfaces implemented in Python.

Q. What is Flask & its benefits?

Python Flask, as we've previously discussed, is a web microframework for Python. It is based on the '`Werkzeug`, `Jinja 2` and good intentions' BSD license. Two of its dependencies are Werkzeug and Jinja2. This means that it has around no dependencies on external libraries. Due to this, we can call it a light framework. A session uses a signed cookie to allow the user to look at and modify session contents. It will remember information from one request to another. However, to modify a session, the user must have the secret key `Flask.secret_key`.

Flask is a web micro framework for Python based on "Werkzeug, Jinja 2 and good intentions" BSD licensed. Werkzeug and jingja are two of its dependencies.

Flask is part of the micro-framework. Which means it will have little to no dependencies on external libraries. It makes the framework light while there is little dependency to update and less security bugs.

Q. Mention what is Flask-WTF and what are their features?

Flask-WTF offers simple integration with WTForms. Features include for Flask WTF are

```
Integration with wtforms
Secure form with csrf token
Global csrf protection
```

```
Internationalization integration
Recaptcha supporting
File upload that works with Flask Uploads
```

Q. Explain what is the common way for the Flask script to work?

The common way for the flask script to work is

```
Either it should be the import path for your application
Or the path to a Python file
```

Q. Explain how you can access sessions in Flask?

A session basically allows you to remember information from one request to another. In a flask, it uses a signed cookie so the user can look at the session contents and modify. The user can modify the session if only it has the secret key Flask.secret_key.

Q. Is Flask an MVC model and if yes give an example showing MVC pattern for your application?

Basically, Flask is a minimalistic framework which behaves same as MVC framework. So MVC is a perfect fit for Flask, and the pattern for MVC we will consider for the following example

from flask import Flask app = Flask(*name*) @app.route("/") def hello(): return "Hello World" app.run(debug = True)

In this code your,

Configuration part will be

```
from flask import Flask
app = Flask(_name_)
```

View part will be

```
@app.route("/")
```

```
def hello():
    return "Hello World"
```

While your model or main part will be

```
app.run(debug = True)
```

Q. Explain database connection in Python Flask?

Best database for flask is MySQL. Flask supports database powered application (RDBS). Such system requires creating a schema, which requires piping the shema.sql file into a sqlite3 command. So you need to install `sqlite3` command in order to create or initiate the database in Flask. Flask allows to request database in three ways

- `before_request()`: They are called before a request and pass no arguments.
- `after_request()`: They are called after a request and pass the response that will be sent to the client.
- `teardown_request()`: They are called in situation when exception is raised, and response are not guaranteed. They are called after the response been constructed. They are not allowed to modify the request, and their values are ignored.

How will you sort result of student whose marks are unknown to you based on their roll numbers?

Using bubble sort.

Q. How will you check memory leak on Linux?

- valgrind along with gcc.

Q. How can you return multiple values from a function/method?

Q. What's the fastest way to swap the values bound to two variables?

Q. What is the importance of reference counting?

Q. Do functions (or methods) return something even if there isn't a `return statement? If so, what do they return?

Q. How do you reverse a list? Can you come up with at least three ways?

Q. How would you merge two sorted lists? They can be any length, or empty.

Q. How would you count the lines in a file? How would you do it if the file was too big to hold in memory?

Q. Is python compile language?

Q. Standard python libraries used?

Q. What is size of integer in python?

Q. What are serialization formats(in python)?

Q. How python manages memory?

Q. Is tuple mutable or immutable?

Q. Tell me some datastructures in Python?

Q. Why python instead of scala on spark when scala has better performance?

Q. How to access file on linux server using python?

Q. What is List Comprehension ? Show with example

Q. What version of Azure DataFactory you are using>?

Q. How to execute a java code?

Q. How to find PID of process and How to find how much resources a process is taking on linux?

Q. What is Amazon's version of Hadoop?

Q. What are tools for ingestion?

Q. What is Python?

Q. Describe some features of Python.

Q. How does Python execute code?

Q. What are some built-in types in Python?

Q. What are bindings, i.e., what does it mean for a value to be bound to a variable?

Q. How do you create a list?

Q. How do you create a dictionary?

Q. What is a list comprehension? Why would you use one?

Q. What is a generator? What can it be used for?

Q. What is inheritance?

Q. What happens if you have an error in an init statement?

Q. What happens in python if you try to divide by zero?

Q. How instance variables are different from class variables?

Q. Differentiate between ".*py" file and "*.pyc" file?

Q. Explain difference between Map vs Reduce Vs Filter ?

Q. What are Generators ?

Q. What are Iterators ?

Q. Can generator be used to create Iterators ? Give example

Q. Can iterators be used to create generator ?

Q. What are iterators and generators?

Q. What is Method Resolution Order ?

Q. Differentiate between append() and extend() methods ?

Q. How can you implement functional programming and why would you?

Q. Explain ctypes and why you would use them?

Q. What is multiple inheritence and when should you use it?

Q. What is a meta class?

Q. What are properties and what's the point?

Q. What's the difference between 2.7+ and 3?

Q. What is a unicode string?

Q. What does the yield statement do?

Q. What is a generator?

Q. Why would I use one?

Q. What is polymorphism, when would I use it?

Q. How do you go about packaging python code?

Q. Is Python compiled?, If yes how so, if not how so.

Q. What does some-variable mean?

Q. Should I import an entire module?

Q. What does dynamicly/duck typed mean?

Q. When would I not use Python?

Q. What is DRY, how can I apply it through OOP or FP?

Q. When would I use Python?

Q. Give examples of Python Framework ?

Q. How Python is interpreted.

Q. Explain dict.

Q. How to pass optional or keyword arguments.

Q. Explain indexing and slicing.

Q. Difference between str() and repr().

Q. What is Dynamic Typing ?

Q. Justify this statement : Everything is object in Python?

Q. What are Middlewares?

Q. What is the use of enumerate() in Python?

Q. What is list / dict compression.

Q. How to make array in python?

Q. How to generate random numbers.

Q. How to handle exceptions.

Q. When to use list/tuple/set/dict?

Q. What is virtualenv

Q. `with` statement and its usage.

Q. What is class and what is self.

Q. Explain isinstance()

Q. What is static method, class method and instance method?

Q. Explain map, filter,reduce and lambda.

Q. Difference between new styled classed and old styled classes.

Q. What is diff between Python and Java?

Q. What is context processor?

Q. What is exec() and eval ()?

Q. How to pass command line argument.

Q. What is yield ?

Q. What is ord() and chr()?

Q. What are Metaclasses?

Q. What is descriptor ?

Q. namespace vs scope ?

Q. What is MRO ?

Q. When to use list comprehensions and when to avoid list comprehensions ?

Q. What are Map, filter and reduce functions ?

Q. What are the different types of exceptions generated in python?

Q. How to write your own custom exception handling ?

Q. Difference between input and raw_input ?

Q. Why do we write `__name__ == "__main__"` in a python script ?

Q. Why does the exception handling have a finally block ?

Q. Does python provide thread safe multi-threading ?

Q. What do you mean by non blocking IO ?

Q. What happens if an error occurs that is not handled in the except block?

Q. How are modules used in a Python program?

Q. How do you create a Python function?

Q. How is a Python class created?

Q. How is a Python class instantiated?

Q. How does a function return values?

Q. What happens when a function doesn't have a return statement? Is this valid?

Q. How do you create a dictionary which can preserve the order of pairs?

Q. Can you use mutable Data Structure as key in Dictionaries?

Q. What is difference between tuple and list? Where will you use tuple and where will you use list?

Q. Explain all the file processing modes supported by Python?

Q. What is PEP 8?

Q. What are parameters to consider for checking when server is down?

Q. How will you measure memory on Linux server?