FREQUENTLY ASKED QUESTIONS: PYTHON

(PART 1)

1) What are the rules for local and global variables in Python?

In Python, variables that are only referenced inside a function are implicitly global. If a variable is assigned a value anywhere within the function's body, it's assumed to be a local unless explicitly declared as global.

Requiring global for assigned variables provides a bar against unintended side-effects.

2) What is negative index in Python?

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

3) Does Python have a switch-case statement?

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_demo(argument):
    switcher = {
        1: "January",
        2: "February",
        3: "March",
        4: "April",
        5: "May",
        6: "June",
        7: "July",
        8: "August",
        9: "September",
        10: "October",
        11: "November",
        12: "December"
    }
    print switcher.get(argument, "Invalid month")
```

FAQ

4) What are the built-in types available In Python?

Immutable built-in datatypes of Python:

-Numbers
-Strings
-Tuples

Mutable built-in datatypes of Python

-List
-Dictionaries
-Sets

5) Why would you use the "pass" statement?

Python has the syntactical requirement that code blocks cannot be empty. Empty code blocks are however useful in a variety of different contexts, for example if you are designing a new class with some methods that you don't want to implement:

```
class MyClass(object):
def meth_a(self):
pass
def meth_b(self):
print "I'm meth_b"
```

If you were to leave out the pass, the code wouldn't run and you'll get an error:

```
IndentationError: expected an indented block
```

Other examples when we could use pass :

- Ignoring (all or) a certain type of Exception.
- Deriving an exception class that does not add new behaviour.
- Testing that code runs properly for a few test values, without caring about the results.

FAQ

6) How do you list the functions in a module?

Use the *dir()* method to list the functions in a module:

```
import some_module
print dir(some_module)
```

7) When to use a tuple vs list vs dictionary in Python?

- Use a tuple to store a sequence of items that will not change.
- Use a list to store a sequence of items that may change.
- Use a dictionary when you want to associate pairs of two items.

8) What are local variables and global variables in Python?

Global Variables: Variables declared outside a function or in global space are called global variables. These variables can be accessed by any function in the program.

Local Variables: Any variable declared inside a function is known as a local variable. This variable is presentin the local space and not in the global space.

9) What is pickling and unpickling?

The pickle module implements a fundamental, but powerful algorithm for serializing and de-serializing a Python object structure.

-Pickling: is the process whereby a Python object hierarchy is converted into a byte stream,

-Unpickling: is the inverse operation, whereby a byte stream is converted back into an object hierarchy.

FAQ

10) Suppose lst is [2, 33, 222, 14, 25], What is lst[-1]?

Problem:

Suppose lst is [2, 33, 222, 14, 25] , What is lst[-1] ?

Solution:

It's 25 . Negative numbers mean that you count from the right instead of the left. So, lst[-1] refers to the last element, lst[-2] is the second-last, and so on.

11) What's the difference between lists and tuples?

The key difference is that tuples are immutable. This means that you cannot change the values in a tuple once you have created it. So if you're going to need to change the values use a List .

Apart from tuples being immutable there is also a semantic distinction that should guide their usage. Tuples are heterogeneous data structures (i.e., their entries have different meanings), while lists are homogeneous
sequences. Tuples have structure, lists have order.

One example of tuple be pairs of page and line number to reference locations in a book, e.g.:

```
my_location = (42, 11)  # page number, line number
```

You can then use this as a key in a dictionary to store notes on locations. A list on the other hand could be used to store multiple locations. Naturally one might want to add or remove locations from the list, so it makes sense that lists are mutable. On the other hand it doesn't make sense to add or remove items from an existing location - hence tuples are immutable.

FAQ

12) What is the function of “self”?

Self is a variable that represents the instance of the object to itself. In most of the object oriented programming language, this is passed to the methods as a hidden parameters that is defined by an object. But, in python, it is declared and passed explicitly. It is the first argument that gets created in the instance of the class A and the parameters to the methods are passed automatically. It refers to separate instance of the variable for individual objects.

Let's say you have a class ClassA which contains a method methodA defined as:

```
def methodA(self, arg1, arg2): #do something
```

and ObjectA is an instance of this class.

Now when ObjectA.methodA(arg1, arg2) is called, python internally converts it for you as:

```
ClassA.methodA(ObjectA, arg1, arg2)
```

The self variable refers to the object itself.

13) What is a "callable"?

A callable is an object we can call - function or an object implementing the __call__ special method. Any object can be made callable

FAQ

14) What are decorators in Python?

In Python, functions are the first class objects, which means that:

- Functions are objects; they can be referenced to, passed to a variable and returned from other functions as well.

- Functions can be defined inside another function and can also be passed as argument to another function.

Decorators are very powerful and useful tool in Python since it allows programmers to modify the behavior of function or class. Decorators allow us to wrap another function in order to extend the behavior of wrapped function, without permanently modifying it.

```
@gfg_decorator
def hello_decorator():
    print("Gfg")

'''Above code is equivalent to:

def hello_decorator():
    print("Gfg")

hello_decorator = gfg_decorator(hello_decorator)'''
```

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

- Use *args when we aren't sure how many arguments are going to be passed to a function, or if we want to pass a stored list or tuple of arguments to a function.

- **kwargs is used when we dont know how many keyword arguments will be passed to a function, or it can be used to pass the values of a dictionary as keyword arguments.

FAQ

16) What is a None value?

None is just a value that commonly is used to signify 'empty', or 'no value here'.

If you write a function, and that function doesn't use an explicit return statement, None is returned instead, for example.

Another example is to use None for default values. it is good programming practice to not use mutable objects as default values. Instead, use None as the default value and inside the function, check if the parameter is None and create a new list/dictionary/whatever if it is.

Consider:

```
def foo(mydict=None):
    if mydict is None:
        mydict = {}  # create a new dict for local namespace
```

17) How can I create a copy of an object in Python?

- To get a fully independent copy (deep copy) of an object you can use the copy.deepcopy() function. Deep copies are recursive copies of each interior object.

- For shallow copy use copy.copy() . Shallow copies are just copies of the outermost container.

18) Is it possible to have static methods in Python?

Use the @staticmethod decorator:

```
class MyClass(object):
@staticmethod
def the_static_method(x):
print x
MyClass.the_static_method(2) # outputs 2
```

FAQ

19) What is the difference between range and xrange functions in Python 2.X?

In Python 2.x:

- range creates a list, so if you do range(1, 10000000) it creates a list in memory with 9999999 elements.

- xrange is a generator object that evaluates lazily.

In Python 3, range does the equivalent of python's xrange , and to get the list, you have to use list(range(...)) .

xrange brings you two advantages:

- You can iterate longer lists without getting a MemoryError.
- As it resolves each number lazily, if you stop iteration early, you won't waste time creating the whole list.

20) What's the difference between the list methods append() and extend()?

append adds an element to a list, and extend concatenates the first list with another list (or another iterable, not necessarily a list).

```
x = [1, 2, 3]
x.append([4, 5])
print (x)
# [1, 2, 3, [4, 5]]

x = [1, 2, 3]
x.extend([4, 5])
print (x)
# [1, 2, 3, 4, 5]
```