100 Python Interview Questions and Answers

Introducing the latest Edition of Python Interview Questions with Answers to help every Python programmer.

In today's IT world, Python is a key programming skill for almost 90% of the population working on Software applications. Hence, we have formulated a list of 100 core Python interview questions to help you master this skill. So, firstly, let's begin with the most basic, yet, must-know Python programming questions.

First Set on the Basic Programming Concepts

These Python interview questions and answers for Freshers are designed to assess your basic understanding of Python concepts and coding skills. Here, you'll find a mix of questions related to data types, loops, functions, and basic problem-solving using Python.

Q-1: What is Python, what are the benefits of using it, and what do you understand of PEP 8?

Python is one of the most successful interpreted languages. When you write a Python script, it doesn't need to be compiled before execution. A few other interpreted languages are PHP and Javascript.

Benefits of Python Programming

1. Python is a dynamic-typed language that allows variables to be declared without specifying data types, supporting assignments like var1=101 and var2="You are an engineer" error-free.

2. Python supports object-oriented programming with classes, composition, and inheritance, devoid of access specifiers like public or private.

3. Python functions are treated as first-class objects, enabling assignment, return, and passing as arguments.

4. Python offers swift development but slower execution compared to compiled languages; incorporation of "C" language extensions enables script optimization.

5. Python serves a multitude of purposes such as web-based applications, test automation, data modeling, and big data analytics, and acts as a versatile “glue” layer for interoperability with other languages.

PEP 8.

PEP 8 is the latest Python coding standard, a set of coding recommendations. It assists in writing more readable Python code.

Q-2: What is the output of the following Python code fragment? Justify your answer.

```
def extendList(val, list=[]):
    list.append(val)
    return list

list1 = extendList(10)
list2 = extendList(123,[])
list3 = extendList('a')

print "list1 = %s" % list1
print "list2 = %s" % list2
print "list3 = %s" % list3
```

The result of the above Python code snippet is:

```
list1 = [10, 'a']
list2 = [123]
list3 = [10, 'a']
```

You may erroneously expect list1 to be equal to [10] and list3 to match with [‘a’], thinking that the list argument will initialize to its default value of [] every time there is a call to the extendList.

However, the flow is like a new list gets created once after the function is defined. And the same gets used whenever someone calls the extendList method without a list argument. It works like this because the calculation of expressions (in default arguments) occurs at the time of function definition, not during its invocation.

The `list1` and `list3` are hence operating on the same default list, whereas list2 is running on a separate object that it has created on its own (by passing an empty list as the value of the list parameter).

The definition of the extendList function can be changed in the following manner.

Copy

```
def extendList(val, list=None):
  if list is None:
    list = []
  list.append(val)
  return list
```

With this revised implementation, the output would be:

Copy

```
list1 = [10]
list2 = [123]
list3 = ['a']
```

Q-3: What is the statement that can be used in Python if the program requires no action but requires it syntactically?

The pass statement is a null operation. Nothing happens when it executes. You should use the “pass” keyword in lowercase. If you write “Pass,” you’ll face an error like “NameError: name Pass is not defined.” Python statements are case-sensitive.

Copy

```
letter = "hai sethuraman"
for i in letter:
    if i == "a":
        pass
        print("pass statement is execute ..............")
    else:
        print(i)
```

Q-4: What's the process to get the home directory using '~' in Python?

You need to import the OS module, and then just a single line would do the rest.

Copy

```
import os
print (os.path.expanduser('~'))
```

Output:

Copy

```
/home/runner
```

Q-5: What are the built-in types available in Python?

Here is the list of the most commonly used built-in types that Python supports:

- **Immutable built-in datatypes of Python**
  - Numbers
  - Strings
  - Tuples
- **Mutable built-in datatypes of Python**
  - List
  - Dictionaries
  - Sets

Q-6: How to find bugs or perform static analysis in a Python application?

- You can use PyChecker, which is a static analyzer. It identifies the bugs in Python projects and also reveals the style and complexity-related bugs.

- Another tool is Pylint, which checks whether the Python module satisfies the coding standard.

Q-7: When is the Python decorator used?

Python decorator is a relative change that you do in Python syntax to adjust the functions quickly.

Q-8: What is the principal difference between a list and a tuple?

The principal difference between a list and a tuple is that the former is mutable while the tuple is not.

A tuple is allowed to be hashed, for example, using it as a key for dictionaries.

Q-9: How does Python handle memory management?

- Python uses private heaps to maintain its memory. So the heap holds all the Python objects and the data structures. This area is only accessible to the Python interpreter; programmers can’t use it.
- And it’s the Python memory manager that handles the Private heap. It does the required allocation of the memory for Python objects.
- Python employs a built-in garbage collector, which salvages all the unused memory and offloads it to the heap space.

Q-10: What are the principal differences between lambda and def?

- Def can hold multiple expressions while lambda is a uni-expression function.
- Def generates a function and designates a name to call it later. Lambda forms a function object and returns it.
- Def can have a return statement. Lambda can’t have return statements.
- Lambda supports getting used inside a list and dictionary.

Q-11: Write a reg expression that confirms an email using the python reg expression module “re”?

Python has a regular expression module “re.

Check out the **“re”** expression that can check the email for .com and .co.in a subdomain.

```
import re
print(re.search(r"[0-9a-zA-Z.]+@[a-zA-Z]+\.
(com|co\.in)$","micheal.pages@mp.com"))
```

Q-12: What do you think is the output of the following code fragment? Is there any error in the code?

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

The result of the above lines of code is []. There won’t be any error like an <IndexError>.

You should know that trying to fetch a member from the list using an index larger than its length (for example, attempting to access the list[10] as given in the question) would yield the <index error>. By the way, retrieving only a slice at the starting index that surpasses the number of items in the list won’t result in an IndexError. It will just return an empty list.

Q-13: Is there a switch or case statement in Python? If not then what is the reason for the same?

No, Python does not have a Switch statement, but you can write a Switch function and then use it.

Q-14: What is a built-in function that Python uses to iterate over a number sequence?

Python range() is the built-in function that we can use to iterate over the elements of a sequence.

```
for i in range(5):
    print(i)
```

The range() function accompanies two sets of parameters.

- **range(stop)**

- Stop: It is the number of integers to generate and starts from zero. eg. range(3) == [0, 1, 2].

- **range([start], stop[, step])**
  - Start: It is the starting number of the sequence.
  - Stop: It specifies the upper limit of the sequence.
  - Step: It is the incrementing factor for generating the sequence.
- **Points to note:**
  - Only integer arguments are allowed.
  - Parameters can be positive or negative.
  - The **range()** function in Python starts from the zeroth index.

Q-15: What are the optional statements possible inside a try-except block in Python?

There are two optional clauses you can use in the **try-except** block.

- The “**else”** clause
  - It is useful if you want to run a piece of code when the try block doesn’t create an exception.
- The **“finally”** clause
  - It is useful when you want to execute some steps that run, irrespective of whether there occurs an exception or not.

Q-16: What is a string in Python?

A string in Python is a sequence of alphanumeric characters. They are immutable objects. It means that they don’t allow modification once they get assigned a value. Python provides several methods, such as join(), replace(), or split() to alter strings. But none of these change the original object.

Q-17: What is slicing in Python?

Slicing is a string operation for extracting a part of the string, or some part of a list. In Python, a string (say text) begins at index 0, and the nth character stores at position text[n-

1].

Python can also perform reverse indexing, i.e., in the backward direction, with the help of negative numbers.

In Python, the slice() is also a constructor function that generates a slice object. The result is a set of indices mentioned by range(start, stop, step). The slice() method allows three parameters.

1. Start – The starting number for the slicing to begin.

2. Stop – The number that indicates the end of slicing.

3. Step – The value to increment after each index (default = 1).

Please keep reading all the Python interview questions to ensure you are not leaving behind any questions that you may be asked during the interview,

Q-18: What is %s in Python?

Python has support for formatting any value into a string. It may contain quite complex expressions.

One of the common usages is to push values into a string with the %s format specifier. The formatting operation in Python is quite similar to C’s <printf()> function.

Q-19: Is a string immutable or mutable in Python?

Python strings are indeed immutable.

Let’s take an example. We have an “str” variable holding a string value. We can’t mutate the container, i.e., the string, but can modify what it contains which means the value of the variable.

Q-20: What is the index in Python?

An index is an integer data type that denotes a position within an ordered list or a string.

In Python, strings are also lists of characters. We can access them using the index which begins from zero and goes to the length minus one.

For example, in the string “Program,” the indexing happens like this:

```
Program 0 1 2 3 4 5
```

Q-21: What is Docstring in Python?

A docstring is a unique text that happens to be the first statement in the following Python constructs:

Module, Function, Class, or Method definition.

A docstring gets added to the __doc__ attribute of the string object.

Now, read some of the Python interview questions and answers on functions.

Q-22: What is a function in Python programming?

A function is an object which represents a block of code and is a reusable entity. It brings modularity to a program and a higher degree of code reusability.

Python has given us many built-in functions such as print() and provides the ability to create user-defined functions.

Q-23: How many basic types of functions are available in Python?

Python gives us two basic types of functions.

1. Built-in, and

2. User-defined.

The built-in functions happen to be part of the Python language. Some of these are print(), dir(), len(), and abs() etc.

Q-24: How do we write a function in Python?

We can create a Python function in the following manner.

#1: Firstly, start a function by specifying the keyword `def` and then put the function name.

#2: Secondly, provide the argument list and enclose it using the parentheses. Next, place a colon in the end. It marks the end of the function header.

#3: Thirdly, press the enter key and add any Python statements you need.

Your first function block is now ready and it will finally look like this:

```
# You first Python function
def my_first_python_func( arg1, arg2) :
    arg1 = 1
    arg2 = "Any string"
    print("arg1 := ", arg1)
    print("arg2 := ", arg2)

# Call the function
x = 0
y = ""
my_first_python_func( x, y)

# result
# arg1 :=  1
# arg2 :=  Any string
```

Q-25: What is a function call or a callable object in Python?

A function in Python gets treated as a callable object. It can allow some arguments and also return a value or multiple values in the form of a tuple. Apart from the function, Python has other constructs, such as classes or class instances which fit in the same category.

Don’t just halt here. We have many more Python interview questions and answers lined up next. However, you may also consider attempting our basic Python quiz which has 20 questions.

Second Set of Python Interview Questions

Prepare for Python job interviews with 25 beginner-friendly Python interview questions and answers. Boost your confidence and build a strong foundation in Python fundamentals like data types, loops, and functions. Get ready to shine in your interviews!

Q-26: What is the return keyword used in Python?

The purpose of a function is to receive the inputs and return some output.

The return is a Python statement that we can use in a function for sending a value back to its caller.

Q-27: What is “Call by Value” in Python?

In call-by-value, the argument is whether an expression or a value gets bound to the respective variable in the function.

Python will treat that variable as local in the function-level scope. Any changes made to that variable will remain local and will not reflect outside the function.

Q-28: What is “Call by Reference” in Python?

We use both “call-by-reference” and “pass-by-reference” interchangeably. When we pass an argument by reference, then it is available as an implicit reference to the function, rather than a simple copy. In such a case, any modification to the argument will also be visible to the caller.

This scheme also has the advantage of bringing more time and space efficiency because it leaves the need for creating local copies.

On the contrary, the disadvantage could be that a variable can get changed accidentally during a function call. Hence, programmers need to handle the code to avoid such uncertainty.

Q-29: What is the return value of the trunc() function?

Python's <trunc()> function performs a mathematical operation to remove the decimal values from a particular expression and provides an integer value as its output.

Q-30: Is it mandatory for a Python function to return a value?

It is not at all necessary for a function to return any value. However, if needed, we can use None as a return value.

Q-31: What does the "continue" keyword do in Python?

Python's continue is a jump statement that moves the control to execute the next iteration in a loop leaving all the remaining instructions in the block unexecuted.

The continue statement is applicable for both the "while" and "for" loops.

Q-32: What is the purpose of the id() function in Python?

The id() is one of the built-in functions in Python.

```
Signature: id(object)
```

It accepts one parameter and returns a unique identifier associated with the input object.

Q-33: What does the *args do in Python?

We use *args as a parameter in the function header. It gives us the ability to pass N (variable) number of arguments.

Please note that this type of argument syntax doesn't allow passing a named argument to the function.

Example of using the *args:

Copy

```
# Python code to demonstrate
# *args for dynamic arguments
def fn(*argList):
    for argx in argList:
        print (argx)

fn('I', 'am', 'Learning', 'Python')
```

The output:

Copy

```
I
am
Learning
Python
```

Q-34: What does the `**kwargs` do in Python?

We can also use the `**kwargs` syntax in a Python function declaration. It lets us pass N (variable) number of arguments that can be named or keyworded.

Example of using the `**kwargs`:

Copy

```
# Python code to demonstrate
# **kwargs for dynamic + named arguments
def fn(**kwargs):
    for emp, age in kwargs.items():
        print ("%s's age is %s." %(emp, age))

fn(John=25, Kalley=22, Tom=32)
```

The output:

Copy

```
John's age is 25.
Kalley's age is 22.
```

```
Tom's age is 32.
```

Q-35: Does Python have a Main() method?

The main() is the entry point function which happens to be called first in most programming languages.

Since Python is interpreter-based, it sequentially executes the lines of the code one by one.

Python also has a Main() method. But it gets executed whenever we run our Python script either by directly clicking it or starting it from the command line.

We can also override the Python default main() function using the Python if statement. Please see the below code.

```
print("Welcome")
print("__name__ contains: ", __name__)
def main():
    print("Testing the main function")
if __name__ == '__main__':
    main()
```

The output:

```
Welcome
__name__ contains:  __main__
Testing the main function
```

Q-36: What does the __ Name __ do in Python?

The __name__ is a unique variable. Since Python doesn't expose the main() function, when its interpreter gets to run the script, it first executes the code which is at level 0 indentation.

To see whether the main() gets called, we can use the __name__ variable in an if clause compared with the value "__main__."

Q-37: What is the purpose of "end" in Python?

Python's print() function always prints a new line at the end. The print() function accepts an optional parameter known as the 'end.' Its value is '\n' by default. We can change the end character in a print statement with the value of our choice using this parameter.

```
# Example: Print a  instead of the new line in the end.
print("Let's learn" , end = ' ')
print("Python")

# Printing a dot in the end.
print("Learn to code from techbeamers" , end = '.')
print("com", end = ' ')
```

The output is:

```
Let's learn Python
Learn to code from techbeamers.com
```

Q-38: When should you use the "break" in Python?

Python provides a break statement to exit from a loop. Whenever the break hits the code, the control of the program immediately exits from the body of the loop.

The break statement in a nested loop causes the control to exit from the inner iterative block.

Q-39: What is the difference between pass and continue in Python?

The continue statement makes the loop resume from the next iteration.

On the contrary, the pass statement instructs to do nothing, and the remainder of the code executes as usual.

Q-40: What does the len() function do in Python?

In Python, the len() is a primary string function. It determines the length of an input string.

Copy

```
>>> some_string = 'techbeamers'
>>> len(some_string)
11
```

Q-41: What does the chr() function do in Python?

The chr() function was re-added in Python 3.2. In version 3.0, it was removed.

It returns the string denoting a character whose Unicode code point is an integer.

For example, the chr(122) returns the string 'z' whereas the chr(1212) returns the string 'Ҽ'.

Q-42: What does the ord() function do in Python?

The ord(char) in Python takes a string of size one and returns an integer denoting the Unicode code format of the character in case of a Unicode type object, or the value of the byte if the argument is of 8-bit string type.

Copy

```
>>> ord("z")
122
```

Q-43: What is Rstrip() in Python?

Python provides the <rstrip()> method which duplicates the string but leaves out the whitespace characters from the end.

This method escapes the characters from the right end based on the argument value, i.e., a string mentioning the group of characters to get excluded.

The method has the following signature:

Copy

```
str.rstrip([char sequence])
```

Copy

```
#Example
test_str = 'Programming    '
# The trailing whitespaces are excluded
print(test_str.rstrip())
```

Q-44: What is whitespace in Python?

Whitespace represents the characters that we use for spacing and separation.

They possess an “empty” representation. In Python, it could be a tab or space.

Q-45: What does the <isalpha()> function in Python?

The <isalpha()> is a built-in function in Python and is used for string-handling purposes.

It returns True if all characters in the string are of alphabet type, and returns False otherwise.

Q-46: How do you use the split() function in Python?

Python’s split() function works on strings to cut a large piece into smaller chunks, or sub-strings. We can specify a separator to start splitting, or it uses the space as one by default.

Copy

```
#Example
str = 'pdf csv json'
print(str.split(" "))
print(str.split())
```

The output:

Copy

```
['pdf', 'csv', 'json']
['pdf', 'csv', 'json']
```

Q-47: What does the join method do in Python?

Python provides the join() method which works on strings, lists, and tuples. It combines them and returns a united value.

Q-48: What does the Title() method do in Python?

Python provides the title() method to convert the first letter in each word to capital format while the rest turns to Lowercase.

```
#Example
str = 'lEaRn pYtHoN'
print(str.title())
```

The output:

```
Learn Python
```

Now, check out some general-purpose Python interview questions and answers.

Q-49: What makes CPython different from Python?

CPython has its core developed in C. The prefix 'C' represents this fact. It runs an interpreter loop used for translating the Python-ish code to C language.

Q-50: Which package is the fastest form of Python?

PyPy provides maximum compatibility while utilizing CPython implementation to improve its performance.

The tests confirmed that PyPy is nearly five times faster than CPython. It currently supports Python 2.7.

Great! It's nice to see you complete the halfway mark to our 100+ Python interview questions and answers. Consider spending 5-10 minutes on Python function quiz part 1 and Python function quiz part 2.

Third Set of Python Interview Questions

Elevate your Python expertise with 25 intermediate-level interview questions with answers. Covering topics like OOP, file handling, and advanced data structures, these questions will challenge and enhance your Python coding skills. Excel in interviews and stand out with confidence!

Q-51: What is GIL in Python language?

Python supports GIL (the global interpreter lock) which is a mutex used to secure access to Python objects, synchronizing multiple threads from running the Python bytecodes at the same time.

Q-52: How is the Python thread safe?

Python ensures safe access to threads. It uses the GIL mutex to set synchronization. If a thread loses the GIL lock at any time, then you have to make the code thread-safe.

For example, many of the Python operations execute as atomic such as calling the sort() method on a list.

Q-53: How does Python manage memory?

Python implements a heap manager internally which holds all of its objects and data structures.

This heap manager does the allocation/de-allocation of heap space for objects.

Q-54: What is a tuple in Python?

A tuple is a collection-type data structure in Python that is immutable.

They are similar to sequences, just like the lists. However, there are some differences between a tuple and a list; the former doesn’t allow modifications whereas the list does.

Also, the tuples use parentheses for enclosing, but the lists have square brackets in their syntax.

Q-55: What is a dictionary in Python programming?

A dictionary is a data structure known as an associative array in Python that stores a collection of objects.

The collection is a set of keys having a single associated value. We can call it a hash, a map, or a hashmap as it gets called in other programming languages.

Q-56: What is the set object in Python?

Sets are unordered collection objects in Python. They store unique and immutable objects. Python has its implementation derived from mathematics.

Q-57: What is the use of the dictionary in Python?

A dictionary has a group of objects (the keys) mapped to another group of objects (the values). A Python dictionary represents a mapping of unique Keys to Values.

They are mutable and hence will not change. The values associated with the keys can be of any Python type.

Q-58: Is Python list a linked list?

A Python list is a variable-length array that is different from C-style linked lists.

Internally, it has a contiguous array for referencing other objects and stores a pointer to the array variable and its length in the list head structure.

Here are some Python interview questions and answers on classes and objects.

Q-59: What is Class in Python?

Python supports object-oriented programming and provides almost all OOP features to use in programs.

A Python class is a blueprint for creating objects. It defines member variables and gets their behavior associated with them.

We can make it by using the keyword "class." An object gets created from the constructor. This object represents the instance of the class.

In Python, we generate classes and instances in the following way.

```
>>>class Human:  # Create the class
...     pass
>>>man = Human()  # Create the instance
>>>print(man)
<__main__.Human object at 0x0000000003559E10>
```

Q-60: What are Attributes and Methods in a Python class?

A class is useless if it has not defined any functionality. We can do so by adding attributes. They work as containers for data and functions. We can add an attribute directly specifying inside the class body.

```
>>> class Human:
...     profession = "programmer" # specify the attribute 'profession' of
the class
>>> man = Human()
>>> print(man.profession)
programmer
```

After we add the attributes, we can go on to define the functions. Generally, we call them methods. In the method signature, we always have to provide the first argument with a self-keyword.

```
>>> class Human:
    profession = "programmer"
    def set_profession(self, new_profession):
        self.profession = new_profession
>>> man = Human()
>>> man.set_profession("Manager")
>>> print(man.profession)
Manager
```

Q-61: How to assign values for the Class attributes at runtime?

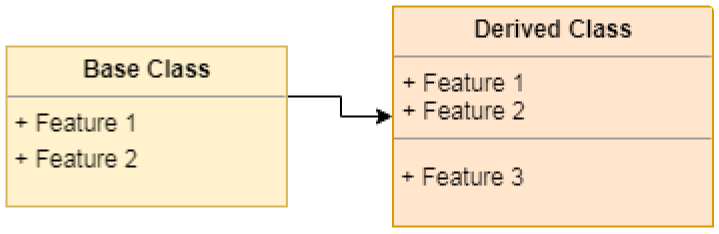
We can specify the values for the attributes at runtime. We need to add an init method and pass input to the object constructor. See the following example demonstrating this.

```
>>> class Human:
    def __init__(self, profession):
        self.profession = profession
    def set_profession(self, new_profession):
        self.profession = new_profession

>>> man = Human("Manager")
>>> print(man.profession)
Manager
```

Q-62: What is Inheritance in Python programming?

Inheritance is an OOP concept that allows a child class object to access its parent class features. It carries forward the base class functionality to the child.



Inheritance In Python

It helps to cut down the same code from related classes.

The common code rests with the base class, and the child class objects can access it via inheritance. Here is an example of inheritance in Python.

Copy

```
class PC: # Base class
    processor = "Xeon" # Common attribute
    def set_processor(self, new_processor):
        processor = new_processor

class Desktop(PC): # Derived class
    os = "Mac OS High Sierra" # Personalized attribute
    ram = "32 GB"

class Laptop(PC): # Derived class
    os = "Windows 10 Pro 64" # Personalized attribute
    ram = "16 GB"

desk = Desktop()
print(desk.processor, desk.os, desk.ram)

lap = Laptop()
print(lap.processor, lap.os, lap.ram)
```

The output:

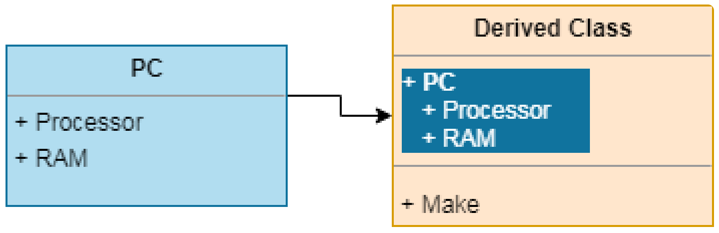
Copy

```
Xeon Mac OS High Sierra 32 GB
Xeon Windows 10 Pro 64 16 GB
```

Q-63: What is Composition in Python?

The composition is also a type of inheritance in Python. It intends to inherit from the base class but a little differently, i.e., by using an instance variable of the base class acting as a member of the derived class.

See the below diagram.



Composition In Python

To demonstrate composition, we need to instantiate other objects in the class and then make use of those instances.

```
class PC: # Base class
    processor = "Xeon" # Common attribute
    def __init__(self, processor, ram):
        self.processor = processor
        self.ram = ram

    def set_processor(self, new_processor):
        processor = new_processor

    def get_PC(self):
        return "%s cpu & %s ram" % (self.processor, self.ram)

class Tablet():
    make = "Intel"
    def __init__(self, processor, ram, make):
        self.PC = PC(processor, ram) # Composition
        self.make = make

    def get_Tablet(self):
        return "Tablet with %s CPU & %s ram by %s" % (self.PC.processor,
self.PC.ram, self.make)

if __name__ == "__main__":
    tab = Tablet("i7", "16 GB", "Intel")
    print(tab.get_Tablet())
```

The output is:

```
Tablet with i7 CPU & 16 GB ram by Intel
```

Q-64: What are Errors and Exceptions in Python programs?

Errors are coding issues in a program that may cause it to exit abnormally.

On the contrary, exceptions happen due to the occurrence of an external event that interrupts the normal flow of the program.

Q-65: How do you handle exceptions with Try/Except/Finally in Python?

Python try-except, and Finally blocks are exception-handling constructs. We can wrap the unsafe code under the try block. And we can keep our fall-back code inside the except block. Any instructions intended for execution in the end should come under the final block.

```
try:
    print("Executing code in the try block")
    print(exception)
except:
    print("Entering in the except block")
finally:
    print("Reached to the final block")
```

The output is:

```
Executing code in the try block
Entering in the except block
Reached to the final block
```

Q-66: How do you raise exceptions for a predefined condition in Python?

We can raise an exception based on some conditions.

For example, if we want the user to enter only odd numbers, else will raise an exception.

```
# Example - Raise an exception
while True:
    try:
        value = int(input("Enter an odd number- "))
        if value%2 == 0:
            raise ValueError("Exited due to invalid input!!!")
        else:
            print("Value entered is : %s" % value)
    except ValueError as ex:
        print(ex)
        break
```

The output is:

```
Enter an odd number- 2
Exited due to invalid input!!!
```

```
Enter an odd number- 1
Value entered is : 1
Enter an odd number-
```

Q-67: What are Python Iterators?

Iterators in Python are array-like objects which allow moving to the next element. We use them in traversing a loop, for example, in a "for" loop.

Python has several types of iterators. For example, a list is also an iterator and we can start a for loop over it.

Q-68: What is the difference between an Iterator and an Iterable?

The collection types like a list, tuple, dictionary, and set are all iterable objects whereas they are also iterable containers that return an iterator while traversing.

Here are some advanced-level Python interview questions and answers.

Q-69: What are Python Generators?

A Generator is a kind of function that lets us specify a function that acts like an iterator and hence can be used in a “for” loop.

In a generator function, the yield keyword substitutes the return statement.

```
# Simple Python function
def fn():
    return "Simple Python function."

# Python Generator function
def generate():
    yield "Python Generator function."

print(next(generate()))
```

The output is:

```
Python Generator function.
```

Q-70: What are Closures in Python?

Python closures are function objects returned by another function. We use them to eliminate code redundancy.

In the example below, we’ve written a simple closure for multiplying numbers.

```
def multiply_number(num):
    def product(number):
        'product() here is a closure'
        return num * number
    return product

num_2 = multiply_number(2)
print(num_2(11))
print(num_2(24))

num_6 = multiply_number(6)
print(num_6(1))
```

The output is:

```
22
48
6
```

Q-71: What are Decorators in Python?

Python decorator gives us the ability to add new behavior to an object at runtime. In the example below, you can check out to see how a decorator works.

Here, we are utilizing the decorator to display a message before the entry and after the exit call of a function.

```
def decorator_sample(func):
    def decorator_hook(*args, **kwargs):
        print("Before the function call")
        result = func(*args, **kwargs)
        print("After the function call")
        return result
    return decorator_hook

@decorator_sample
def product(x, y):
    "Function to multiply two numbers."
    return x * y

print(product(3, 3))
```

The output is:

```
Before the function call
After the function call
9
```

Q-72: How do you create a dictionary in Python?

Let’s take the example of building a website statistics. For this, we first need to break up the key-value pairs using a colon(“:”).

The keys should be of an immutable type. We’ll use the data types that don’t allow changes at runtime. Hence, we’ll choose from an int, string, or tuple.

However, we can take values of any kind. For distinguishing the data pairs, we can use a comma(“,”) and keep the whole stuff inside curly braces({…}).

```
>>> site_stats = {'site': 'tecbeamers.com', 'traffic': 10000, "type":
"organic"}
>>> type(site_stats)
<class 'dict'>
>>> print(site_stats)
{'type': 'organic', 'site': 'tecbeamers.com', 'traffic': 10000}
```

Q-73: How do you read from a dictionary in Python?

To fetch data from a dictionary, we can directly access using the keys. We can enclose a “key” using brackets […] after mentioning the variable name corresponding to the dictionary.

```
>>> site_stats = {'site': 'tecbeamers.com', 'traffic': 10000, "type":
"organic"}
>>> print(site_stats["traffic"])
```

We can even call the get method to fetch the values from a dictionary. It also lets us set a default value. If the key is missing, then the KeyError would occur.

Copy

```
>>> site_stats = {'site': 'tecbeamers.com', 'traffic': 10000, "type":
"organic"}
>>> print(site_stats.get('site'))
tecbeamers.com
```

Also Read: Python OrderedDict Tutorial

Q-74: How do you traverse through a dictionary object in Python?

We can use the “for” and “in” loops for traversing the dictionary object.

Copy

```
>>> site_stats = {'site': 'tecbeamers.com', 'traffic': 10000, "type":
"organic"}
>>> for k, v in site_stats.items():
    print("The key is: %s" % k)
    print("The value is: %s" % v)
    print("++++++++++++++++++++++++")
```

The output is:

Copy

```
The key is: type
The value is: organic
++++++++++++++++++++++++
The key is: site
The value is: tecbeamers.com
++++++++++++++++++++++++
The key is: traffic
The value is: 10000
++++++++++++++++++++++++
```

Q-75: How do you add elements to a dictionary in Python?

To add elements to a dictionary, we specify a key as an array index and specify a value to it as shown below.

Copy

```
>>> # Setup a blank dictionary
>>> site_stats = {}
>>> site_stats['site'] = 'google.com'
>>> site_stats['traffic'] = 10000000000
>>> site_stats['type'] = 'Referral'
>>> print(site_stats)
{'type': 'Referral', 'site': 'google.com', 'traffic': 10000000000}
```

We can even join two dictionaries to get a bigger dictionary with the help of the update() method.

Copy

```
>>> site_stats['site'] = 'google.co.in'
>>> print(site_stats)
{'site': 'google.co.in'}
>>> site_stats_new = {'traffic': 1000000, "type": "social media"}
>>> site_stats.update(site_stats_new)
>>> print(site_stats)
{'type': 'social media', 'site': 'google.co.in', 'traffic': 1000000}
```

If you wish to further strengthen or test your OOP skills, try attempting our Python classes and objects quiz.

Fourth Set of Python Interview Questions

Master Python with 25 advanced-level interview questions. Covering dictionaries in Python, decorators, generators, multithreading, and more, these questions challenge your expertise. Excel in interviews and showcase your advanced Python skills with confidence!

Q-76: How do you delete elements of a dictionary in Python?

We can delete a key in a dictionary by using the del() method.

Copy

```
>>> site_stats = {'site': 'tecbeamers.com', 'traffic': 10000, "type":
"organic"}
>>> del site_stats["type"]
>>> print(site_stats)
{'site': 'google.co.in', 'traffic': 1000000}
```

Another method, we can use is the pop() function. It accepts the key as the parameter. Also, as a second parameter, we can pass a default value if the key doesn't exist.

Copy

```
>>> site_stats = {'site': 'tecbeamers.com', 'traffic': 10000, "type":
"organic"}
>>> print(site_stats.pop("type", None))
organic
>>> print(site_stats)
{'site': 'tecbeamers.com', 'traffic': 10000}
```

Q-77: How do you check the presence of a key in a dictionary?

We can use Python's "in" operator to test the presence of a key inside a dictionary object.

Copy

```
>>> site_stats = {'site': 'tecbeamers.com', 'traffic': 10000, "type":
"organic"}
>>> 'site' in site_stats
True
>>> 'traffic' in site_stats
True
>>> "type" in site_stats
True
```

Earlier, Python also provided the has_key() method which was deprecated.

Q-78: What is the syntax for List comprehension in Python?

The signature for the list comprehension is as follows:

Copy

```
[ expression(var) for var in iterable ]
```

For example, the below code will return all the numbers from 10 to 20 and store them in a list.

```
>>> alist = [var for var in range(10, 20)]
>>> print(alist)
```

Q-79: What is the syntax for Dictionary comprehension in Python?

A dictionary has the same syntax as for list comprehension but the difference is that it uses curly braces:

```
{ aKey, itsValue for aKey in iterable }
```

For example, the below code will return all the numbers 10 to 20 as the keys and will store the respective squares of those numbers as the values.

```
>>> adict = {var:var**2 for var in range(10, 20)}
>>> print(adict)
```

Q-80: What is the syntax for Generator expression in Python?

The syntax for generator expression matches with the list comprehension, but the difference is that it uses parenthesis:

```
( expression(var) for var in iterable )
```

For example, the below code will create a generator object that generates the values from 10 to 20 upon using it.

```
>>> (var for var in range(10, 20))
 at 0x0000000003668728>
>>> list((var for var in range(10, 20)))
```

Now, see more Python interview questions and answers for practice.

Q-81: How do you write a conditional expression in Python?

We can utilize the following single statement as a conditional expression. default_statment if Condition else another_statement

```
>>> no_of_days = 366
>>> is_leap_year = "Yes" if no_of_days == 366 else "No"
>>> print(is_leap_year)
Yes
```

Q-82: What do you know about the Python enumerate?

While using the iterators, sometimes we might have a use case to store the count of iterations. Python makes this task quite easy for us by giving a built-in method known as the enumerate().

The enumerate() function attaches a counter variable to an iterable and returns it as the "enumerated" object.

We can use this object directly in the "for" loops or transform it into a list of tuples by calling the list() method. It has the following signature:

```
enumerate(iterable, to_begin=0)
```

```
Arguments:
iterable: array type object which enables iteration
to_begin: the base index for the counter is to get started, its default
value is 0
```

```
# Example - enumerate function
alist = ["apple","mango", "orange"]
astr = "banana"
```

```
# Let's set the enumerate objects
list_obj = enumerate(alist)
str_obj = enumerate(astr)

print("list_obj type:", type(list_obj))
print("str_obj type:", type(str_obj))

print(list(enumerate(alist)) )
# Move the starting index to two from zero
print(list(enumerate(astr, 2)))
```

The output is:

```
list_obj type: <class 'enumerate'>
str_obj type: <class 'enumerate'>
[(0, 'apple'), (1, 'mango'), (2, 'orange')]
[(2, 'b'), (3, 'a'), (4, 'n'), (5, 'a'), (6, 'n'), (7, 'a')]
```

Q-83: What is the use of the globals() function in Python?

The globals() function in Python returns the current global symbol table as a dictionary object.

Python maintains a symbol table to keep all necessary information about a program. This info includes the names of variables, methods, and classes used by the program.

All the information in this table remains in the global scope of the program and Python allows us to retrieve it using the globals() method.

```
Signature: globals()

Arguments: None
```

```
# Example: globals() function
x = 9
def fn():
    y = 3
    z = y + x
    # Calling the globals() method
    z = globals()['x'] = z
    return z

# Test Code
ret = fn()
print(ret)
```

The output is:

```
12
```

Q-84: Why do you use the zip() in Python?

Python zip function combines the values of the corresponding indexes of multiple containers e.g. lists into a tuple. It returns a zip object containing tuples.

```
Signature:
 zip(*iterators)
Arguments:
 Python iterables or collections (e.g., list, string, etc.)
Returns:
 A single iterator object with combined mapped values
```

```
# Example: zip() function

emp = [ "tom", "john", "jerry", "jake" ]
age = [ 32, 28, 33, 44 ]
dept = [ 'HR', 'Accounts', 'R&D', 'IT' ]
```

```
# call zip() to map values
out = zip(emp, age, dept)

# print the tuples from the zip object
for ele in out:
    print(ele)
```

The output is:

```
The output of zip() is : {('jerry', 33, 'R&D'), ('jake', 44, 'IT'),
('john', 28, 'Accounts'), ('tom', 32, 'HR')}
```

Q-85: What is a class-level or static variable in Python Programming?

In Python programming, a class-level or static variable is a variable that is associated with a class rather than with instances or objects of that class. It is defined within the class scope but outside any class method.

Unlike instance variables, which have unique values for each object of the class, class-level variables have the same value for all instances of the class. They are shared among all instances and can be accessed using either the class name or an instance of the class.

Class-level variables are typically used to store data that is common to all instances of a class or to maintain state information that needs to be shared across instances. They can be accessed and modified directly from the class or through instances of the class.

To define a class-level variable in Python, it is typically declared within the class block, outside of any class methods, using the syntax: variable_name = value.

```
# Example
class Test:
    aclass = 'programming' # A class variable
    def __init__(self, ainst):
        self.ainst = ainst # An instance variable
```

```
# Objects of CSStudent class
test1 = Test(1)
test2 = Test(2)

print(test1.aclass)
print(test2.aclass)
print(test1.ainst)
print(test2.ainst)

# A class variable is also accessible using the class name
print(Test.aclass)
```

The output is:

Copy
```
programming
programming
1
2
programming
```

Let's face some more Python interview questions and answers.

Q-86: How does the ternary operator work in Python?

The ternary operator is an alternative for conditional statements. It combines true or false values with a statement that you need to test.

The syntax would look like the one given below.

[onTrue] if [Condition] else [onFalse]

Copy
```
x, y = 35, 75
smaller = x if x < y else y
print(smaller)
```

Q-87: What does the "self" keyword do?

The **self** is a Python keyword that represents a variable that holds the instance of an object.

In almost, all object-oriented languages, it is passed to the methods as a hidden parameter.

Q-88: What are the different methods to copy an object in Python?

There are two ways to copy objects in Python.

- **Copy() function**
  - It makes a copy of the file from source to destination.
  - It'll return a shallow copy of the parameter.
- **Deepcopy() function**
  - It also produces a copy of an object from the source to the destination.
  - It'll return a deep copy of the parameter that you can pass to the function.

Q-89: What is the purpose of docstrings in Python?

In Python, the docstring is what we call the docstring. It sets a process for recording Python functions, modules, and classes.

Q-90: Which Python function will you use to convert a number to a string?

For converting a number into a string, you can use the built-in function **str()**. If you want an octal or hexadecimal representation, use the inbuilt function **oct()** or **hex()**.

Q-91: How do you debug a program in Python? Is it possible to step through the Python code?

Yes, we can use the Python debugger `pdb` to debug any Python program. And if we start a program using the, then it lets us even step through the code.

Q-92: List down some of the PDB commands for debugging Python programs.

Here are a few PDB commands to start debugging Python code.

- Add breakpoint **(b)**

- Resume execution **(c)**
- Step-by-step debugging **(s)**
- Move to the next line **(n)**
- List source code **(l)**
- Print an expression **(p)**

Q-93: What is the command to debug a Python program?

The following command helps run a Python program in debug mode.

```
$ python -m pdb python-script.py
```

Q-94: How do you monitor the code flow of a program in Python?

In Python, we can use **the sys** module's `settrace()` method to set up trace hooks and monitor the functions inside a program.

You need to define a trace callback method and pass it to the `settrace()` function. The callback should specify three arguments as shown below.

```
import sys

def trace_calls(frame, event, arg):
    # The 'call' event occurs before a function gets executed.
    if event != 'call':
        return
    # Next, inspect the frame data and print information.
    print 'Function name=%s, line num=%s' % (frame.f_code.co_name, 
frame.f_lineno)
    return

def demo2():
    print 'in demo2()'

def demo1():
    print 'in demo1()'
    demo2()

sys.settrace(trace_calls)
demo1()
```

Q-95: Why and when do you use generators in Python?

A generator in Python is a function that returns an iterable object. We can iterate on the generator object using the **yield** keyword. But we can only do that once because their values don't persist in memory, they get the values on the fly.

Generators give us the ability to hold the execution of a function or a step as long as we want to keep it. However, here are a few examples where it is beneficial to use generators.

- We can replace loops with generators for efficiently calculating results involving large data sets.
- Generators are useful when we don't want all the results and wish to hold back for some time.
- Instead of using a callback function, we can replace it with a generator. We can write a loop inside the function doing the same thing as the callback and turn it into a generator.

Q-96: What does the yield keyword do in Python?

The **yield** keyword can turn any function into a generator. It works like a standard return keyword. But it'll always return a generator object. Also, a method can have multiple calls to the **yield** keyword.

See the example below.

Copy

```
def testgen(index):
  weekdays = ['sun','mon','tue','wed','thu','fri','sat']
  yield weekdays[index]
  yield weekdays[index+1]

day = testgen(0)
print next(day), next(day)
```

```
#output: sun mon
```

Q-97: How to convert a list into other data types?

Sometimes, we don’t use lists as is. Instead, we have to convert them to other types.

Turn a list into a string.

We can use the **”.join()** method which combines all elements into one and returns as a string.

```
weekdays = ['sun','mon','tue','wed','thu','fri','sat']
listAsString = ' '.join(weekdays)
print(listAsString)

#output: sun mon tue wed thu fri sat
```

Turn a list into a tuple.

Call Python’s **tuple()** function for converting a list into a tuple.

This function takes the list as its argument.

But remember, we can’t change the list after turning it into a tuple because it becomes immutable.

```
weekdays = ['sun','mon','tue','wed','thu','fri','sat']
listAsTuple = tuple(weekdays)
print(listAsTuple)

#output: ('sun', 'mon', 'tue', 'wed', 'thu', 'fri', 'sat')
```

Turn a list into a set.

Converting a list to a set poses two side effects.

- The set doesn't allow duplicate entries so the conversion will remove any such item.
- A set is an ordered collection, so the order of list items would also change.

However, we can use the **set()** function to convert a list into a Set.

Copy

```
weekdays = ['sun','mon','tue','wed','thu','fri','sat','sun','tue']
listAsSet = set(weekdays)
print(listAsSet)

#output: set(['wed', 'sun', 'thu', 'tue', 'mon', 'fri', 'sat'])
```

Turn a list into a dictionary.

In a dictionary, each item represents a key-value pair. So converting a list isn't as straightforward as it was for other data types.

However, we can achieve the conversion by breaking the list into a set of pairs and then calling the **zip()** function to return them as tuples.

Passing the tuples into the `dict()` function would finally turn them into a dictionary.

Copy

```
weekdays = ['sun','mon','tue','wed','thu','fri']
listAsDict = dict(zip(weekdays[0::2], weekdays[1::2]))
print(listAsDict)

#output: {'sun': 'mon', 'thu': 'fri', 'tue': 'wed'}
```

Q-98: How do you count the occurrences of each item present in the list without explicitly mentioning them?

Unlike sets, lists can have items with the same values.

In Python, the list has a **count()** function which returns the occurrences of a particular item.

Count the occurrences of an individual item.

Copy

```
weekdays = ['sun','mon','tue','wed','thu','fri','sun','mon','mon']
print(weekdays.count('mon'))

#output: 3
```

Count the occurrences of each item in the list.

We'll use the list comprehension along with the **count()** method. It'll print the frequency of each of the items.

Copy

```
weekdays = ['sun','mon','tue','wed','thu','fri','sun','mon','mon']
print([[x,weekdays.count(x)] for x in set(weekdays)])

#output: [['wed', 1], ['sun', 2], ['thu', 1], ['tue', 1], ['mon', 3],
['fri', 1]]
```

Q-99: What is NumPy and how is it better than a list in Python?

NumPy is a Python package for scientific computing that can deal with large data sizes. It includes a powerful N-dimensional array object and a set of advanced functions.

Also, the NumPy arrays are superior to the built-in lists. There are several reasons for this.

- NumPy arrays are more compact than lists.
- Reading and writing items is faster with NumPy.
- Using NumPy is more convenient than the standard list.
- NumPy arrays are more efficient as they augment the functionality of lists in Python.

Q-100: What are the different ways to create an empty NumPy array in Python?

There are two methods that we can apply to create empty NumPy arrays.

The first method is to create an empty array.

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

The second method is to create an empty array.

```
# Make an empty NumPy array
numpy.empty(shape=(0,0))
```

Fifth Set of Python Interview Questions

Looking for quick help in preparing for a Python interview? Then, read the top 10 Python interview questions and answers that are essential for every developer.

Q-1. What are the different environment variables in Python? And what's the use of these variables?

1.1- PYTHONPATH- It is the same as the PATH variable. Python interpreter uses it to search the module files.

1.2- PYTHONSTARTUP- It stores the path of an initialization script containing Python code. It gets to run every time the Python interpreter starts.

1.3- PYTHONCASEOK– In Windows, it instructs Python to find the first case-insensitive match in an import statement. You need to set it for activation.

1.4- PYTHONHOME– It's an extra PATH variable to search modules.

Q-2. What are the commands to copy an object in Python?

2.1- `<copy.copy()>` – It makes a copy of a file from the source to the target. Its output is a shallow copy of the parameter passed.

2.2- `<copy.deepcopy()>` – It produces a copy of an object from the source to the target. Its output is a deep copy of the parameter passed.

Q-3. What is the result of the below lines of code?

Copy
```
str = 'Learn Python!'
print str[3:8]
```

It’ll return the part of the string from the 3rd to the 5th index. The result would be <rn Py>.

Q-4. What could be the result of <print str * 2> if str = ‘Learn Python!’?

It’ll repeat the string two times. e.g. <Learn Python! Learn Python!>.

Q-5. What is a <tuple>in Python? Why is it used?

A <tuple> is a kind of sequence data type. It is the same as a list. A tuple stores a list of values furcated by commas. Unlike lists, you use parentheses to confine tuples.

Q-6. How do you change a string to an int or long in Python?

6.1- <int(X [,base])>- Changes X to an integer. The default value of the base is 10.

6.2- <long(Y [,base])>- Changes Y to a long value. The default value of the base is 10.

Q-7. What is a negative index? Why is it used?

7.1- <-1> is the first -ve index. It acts as the last index.

7.2- <-2> is the second -ve index. It acts as the next-to-last index.

Q-8. How do you start a thread in Python?

To run a thread in Python, you need to call the following method of the thread module.

Copy
```
thread.start_new_thread ( function, args[, kwargs] )
```

Read more about multithreading in Python.

Q-9. How do you create a socket in Python?

9.1- <socket.socket()> creates a new socket using the below syntax.

```
# create a socket object
simplesocket = socket.socket(
                socket.AF_INET, socket.SOCK_STREAM)
```

9.2- It takes the address family, socket type, and protocol number as input.

Read more about socket programming in Python.

Q-10. How do you create a multi-dimensional list in Python?

You can build it by creating a 1-D list. And, then fill each of its elements with a new list.

So, we've now come to the end of this list of the top 10 Python interview questions and answers. And, we hope you can use this knowledge in the right place. Stay tuned for more updates.

Sixth Set of Python Programming Interview Questions

Even the best Python programmers need to prepare when it comes to facing an interview. However, they might quickly catch up with the competition than those who recently started studying Python programming.

So to fill this gap, we are laying down the top 20 Python programming interview questions for both beginners and experienced.

Q-1. What are the core default modules available in Python? List down a few of them.

Ans. Following are a few of the default modules available in Python.

- **email –** Help to parse, handle, and generate email messages.
- **string –** Contains functions to process standard Python strings.
- **SQLite3 –** Provides methods to work with the SQLite database.
- **XML –** Enables XML support.
- **logging –** Adds support for log classes and methods.

- **traceback –** Allows to extract and print stack trace details.

Q-2. Why is <__init__.py> module used in Python?

Ans.

The **<__init__.py>** module does the following:

1. It makes Python interpret directories as containing packages by excluding the ones with a common name such as string.

2. It grants a programmer the control to decide which directory is a package and which is not.

3. However, the **<__init__.py>** can also be an empty file. It can then help in executing the initialization code for a package or setting the **<__all__>** variable.

Q-3. What is a negative index in Python?

Ans.
In Python, we can access both arrays & lists using positive or negative numbers (aka index). A negative index reads the list of elements from the end counting in the backward direction. Check out the example given below.

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

print(a[-3])
print(a[-2])
print(a[-1])
```

Q-4. What is Pickling and how does it differ from Unpickling?

Ans.

Pickling is a process by which a Python object gets converted into a string via a pickle module. The process then puts it into a file by calling the dump() method.

Whereas unpickling does the reverse of the above-said process. It retrieves the stored string and turns it back into an object.

Q-5. What does slicing do in Python? Explain it with an example.

Ans.
Slicing in Python lets you take a part of a list, tuple, or string by specifying the starting and ending positions. It's like cutting a pizza into smaller slices, where you choose which part of the pizza you want to eat.

In Python, you use square brackets `[]` with a colon `:` to do slicing. The format is `sequence[start_index:end_index]`, where you get elements from the `start_index` up to, but not including, the `end_index`.

Here's an example:

```
# Create a list
my_list = [1, 2, 3, 4, 5, 6, 7, 8, 9]

# Slice the list to get a new list from index 2 to index 5 (exclusive)
sliced_list = my_list[2:5]

# Output: [3, 4, 5]
print(sliced_list)
```

Q-6. What are the different ways to generate random numbers in Python?

Ans.

```
#1. random() - returns a floating point number, between 0 and 1.

#2. uniform(X, Y) - returns a floating point number between the values
given as X and Y.

#3. randint(X, Y) - returns a random integer between the values given as
X and Y.
```

Q-7. What is the purpose of the "pass" keyword in Python?

Ans.
In Python, the "pass" keyword is used as a placeholder or a do-nothing statement. It is like an empty command that allows you to create a syntactically correct block of code without actually doing anything.

The "pass" keyword is often used in situations where Python requires an indented block, like in loops, functions, or conditional statements, but you don't want to perform any specific action at that moment.

Here's an example to illustrate the use of the "pass" keyword:

Copy

```
# A loop where you want to skip some iterations for now
for i in range(5):
    if i == 2:
        pass  # We don't want to do anything for i = 2, but we may add
code later
    else:
        print(i)  # Print other values
```

Q-8. What are iterators in Python?

Ans.
Iterators in Python are objects that can be used to iterate over a sequence of data. They are powerful tools that can be used to read and process data more efficiently.

To create an iterator, you need to use the `iter()` function. The `iter()` function takes an iterable object as an argument and returns an iterator object. An iterable object is an object that can be iterated over, such as a list, string, or tuple.

For example, the following code creates an iterator for the list `[11, 22, 33]`:

Copy

```
list = [11, 22, 33]
iterator = iter(list)
```

Q-9. What are the generators in Python?

Ans.

Generators are a way of implementing iterators. A generator function is a normal function except that it contains yield expression in the function definition making it a generator function.

This function returns a generator iterator known as a generator.

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

Q-10. How will you run a subprocess or an external program with arguments in Python?

Ans.

Python provides two ways to run a subprocess or external programs. The first is to use the subprocess module in the `stdlib`.

Copy
```
from subprocess import call
call(["ls", "-l"])
```

The advantage of subprocess vs. system is that it is more flexible. You can get the stdout, stderr, the "real" status code, and better error handling. The second approach for running a program with arguments is as follows.

Copy
```
subprocess.Popen(arglist,stdout=outputfile)
```

Q-11. How will you remove the duplicate elements from the given list?

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

Ans.

A simple solution is to iterate over the list, identify duplicates, and remove them.

But the best solution which we can recommend is as follows.

Copy

```
a = [1,2,2,3]
list(set(a))
```

The set is another type available in Python. It doesn't allow copies and provides some good functions to perform set operations like union, difference, etc.

Q-12. How will you print the sum of numbers starting from 1 to 100 (inclusive of both)?

Ans.

We'll be calculating the sum for numbers from 1-100 using the Python range() function.

Copy

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

#range() returns a list to the sum function containing
#all the numbers from 1 to 100. Please see that
#the range function does not include the end given (101 here).

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

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

Q-13. What is the best approach to storing a list of an employee's first and last names?

Ans.
A list of first and last names is best stored as a list of dictionaries. The following format can be used.

Copy

```
{'first_name':'Example','last_name':'TechBeamers'}
```

Q-14: Does Python allow arguments to Pass by Value or Pass by Reference?

Ans.
Neither the arguments are Pass by Value nor does Python support Pass by reference.

Instead, they are “Pass by assignments”.

The parameter which you pass is originally a reference to the object not a reference to a fixed memory location. But the reference is passed by value. Additionally, some data types like strings and tuples are immutable whereas others are mutable.

Q-15. What are the different methods Python provides for copying an object?

Ans. We can either use a **“Shallow Copy”** or follow a **“Deep Copy**” approach.

Shallow Copy method.

The content of an object (say dictionary) doesn’t get copied by value but by creating a new reference.

Copy

```
>>> a = {1: [1,2,3]}
>>> b = a.copy()
>>> a, b
({1: [1, 2, 3]}, {1: [1, 2, 3]})
>>> a[1].append(4)
>>> a, b
({1: [1, 2, 3, 4]}, {1: [1, 2, 3, 4]})
```

Deep Copy method.

It copies all the contents by value.

Copy

```
>>> c = copy.deepcopy(a)
>>> a, c
({1: [1, 2, 3, 4]}, {1: [1, 2, 3, 4]})
>>> a[1].append(5)
>>> a, c
({1: [1, 2, 3, 4, 5]}, {1: [1, 2, 3, 4]})
```

Q-16. How will you convert a string to a number in Python?

Ans.
Python provides the **<int()>** method, a standard built-in function to convert a string into an integer value.

You can call it with a string containing a number as the argument, and it returns the number converted to an actual integer.

```
print int("1") + 1
The above prints 2.
```

Q-17. How will you set a global variable inside a function?

Ans.
You can use a global variable in other functions by declaring it as global in each function that is assigned to it:

```
globvar = 0
def set_globvar_to_one():
    global globvar    # Needed to modify global copy of globvar
    globvar = 1
def print_globvar():
    print globvar     # No need for global declaration to read value of
globvar
set_globvar_to_one()
print_globvar()       # Prints 1
```

I imagine the reason for it is that, since global variables are so dangerous, Python wants to make sure that you know that's what you're playing with by explicitly requiring the global keyword.

Q-18. How will you share global variables across modules?

Ans.
If you add a variable to the **<__builtin__>** module, it will be accessible as if a global from any other module that includes **<__builtin__>** — which is all of them, by default.

```
a.py contains
print foo
b.py contains
import __builtin__
__builtin__.foo = 1
import a
The result is that "1" is printed.
```

Note: Python 3 introduced the builtins keyword as a replacement for the **<__builtin__>**.

Q-19. Is there a tool to help find bugs or perform the static analysis?

Ans.

Yes. **PyChecker** is a static analysis tool. It finds bugs in the source code and raises alerts for issues in code complexity or style.

Pylint is another tool that checks if a module meets the coding standard. It also supports additional plug-ins to enable custom features.

Q-20. How can you perform unit testing in Python?

Ans.
Python packages a unit testing framework called **<Unittest>**. It supports the following features.

- Automation testing.
- Sharing of setup and shutdown code for tests.
- Aggregation of tests into collections.
- Independence of the tests from the reporting framework.

Don’t leave now. Continue your practice with our selected picks of Python coding interview questions.

Seventh Set of Python Coding Interview Questions

We've compiled some of the trickiest Python coding interview questions and answers. Take your time to read them carefully and become familiar with the latest questions often asked in Python interviews.

Additionally, remember to focus more on the practical application of Python while preparing for interviews, as it will undoubtedly be beneficial to you.

Q-1. What is the function to randomize the items of a list in place?

Ans. Python has a built-in module called <random>. It exports a public method <shuffle(<list>)> which can randomize any input sequence.

Copy

```
import random
list = [2, 18, 8, 4]
print("Prior Shuffling - 0", list)
random.shuffle(list)
print("After Shuffling - 1", list)
random.shuffle(list)
print("After Shuffling - 2", list)
```

Q-2. What is the best way to split a string in Python?

Ans. We can use the Python <split()> function to break a string into substrings based on the defined separator. It returns the list of all words present in the input string.

Copy

```
test = "I am learning Python."
print(test.split(" "))
```

Program Output.

Copy

```
Python 2.7.10 (default, Jul 14 2015, 19:46:27)
[GCC 4.8.2] on linux

['I', 'am', 'learning', 'Python.']
```

Q-3. What is the right way to transform a Python string into a list?

Ans. In Python, strings are just like lists. And it is easy to convert a string into a list. Simply passing the string as an argument to the list would result in a string-to-list conversion.

```
list("I am learning Python.")
```

Program Output.

```
Python 2.7.10 (default, Jul 14 2015, 19:46:27)
[GCC 4.8.2] on linux

=> ['I', ' ', 'a', 'm', ' ', 'l', 'e', 'a', 'r', 'n', 'i', 'n', 'g', ' ',
'P', 'y', 't', 'h', 'o', 'n', '.']
```

Q-4. How does exception handling in Python differ from Java? Also, list the optional clauses for a <try-except> block in Python.

Ans. Unlike Java, exception handling in Python is implemented differently. It provides an option of using a <try-except> block where the programmer can see the error details without terminating the program. Sometimes, along with the problem, this <try-except> statement offers a solution to deal with the error.

The following error-handling blocks are available in Python language:

1. try-except-finally
2. try-except-else

Q-5. What do you know about the <list> and <dict> comprehensions? Explain with an example.

Ans. The <List/Dict> comprehensions provide an easier way to create the corresponding object using the existing iterable. As per official Python documents, the list comprehensions are usually faster than the standard loops. But it's something that may change between releases.

The <List/Dict> Comprehensions Examples.

```
#Simple Iteration
item = []
for n in range(10):
    item.append(n*2)
print(item)
```

#List Comprehension

```
item = [n*2 for n in range(10)]
print(item)
```

Both the above examples would yield the same output.

```
Python 2.7.10 (default, Jul 14 2015, 19:46:27)
[GCC 4.8.2] on linux

[0, 2, 4, 6, 8, 10, 12, 14, 16, 18]
```

```
#Dict Comprehension
item = {n: n*2 for n in range(10)}
print(item)
```

```
Python 2.7.10 (default, Jul 14 2015, 19:46:27)
[GCC 4.8.2] on linux

{0: 0, 1: 2, 2: 4, 3: 6, 4: 8, 5: 10, 6: 12, 7: 14, 8: 16, 9: 18}
```

Q-6. What are the methods you know to copy an object in Python?

Ans. Commonly, we use `copy.copy()` or `copy.deepcopy()` to perform copy operations on objects. Not all objects support these methods but most do.

But some objects are easier to copy. Like the dictionary objects provide a <copy()> method.

E.g. Simple way to copy an object in Python

```
item = {n: n*2 for n in range(10)}
newdict = item.copy()
print(newdict)
```

Q-7. Can you write code to determine the name of an object in Python?

Ans. No objects in Python have any associated names. So there is no way of getting the one for an object. The assignment is only the means of binding a name to the value. The name then can only refer to accessing the value. The most we can do is to find the reference name of the object.

Python code to check the name of the object.

```
class Test:
    def __init__(self, name):
        self.cards = []
        self.name = name

    def __str__(self):
        return '{} holds ...'.format(self.name)

obj1 = Test('obj1')
print(obj1)

obj2 = Test('obj2')
print(obj2)
```

Q-8. Can you write code to check whether the given object belongs to a class or its subclass?

Ans. Python has a built-in method to list the instances of an object that may consist of many classes. It returns in the form of a table containing tuples instead of individual classes. Its syntax is as follows.

```
<isinstance(obj, (class1, class2, ...))>
```

The above method checks the presence of an object in one of the classes. The built-in types can also have many formats of the same function like <isinstance(obj, str)> or <isinstance(obj, (int, long, float, complex))>.

Also, it's not recommended to use the built-in classes. Create a user-defined class instead.

We can take the following example to determine the object of a particular class.

Python code to check whether the object belongs to a class.

```
def lookUp(obj):
    if isinstance(obj, Mailbox):
        print("Look for a mailbox")
    elif isinstance(obj, Document):
        print("Look for a document")
    else:
        print("Unidentified object")
```

Q-9. What is the result of the following Python program?

Ans. The example code is as follows.

```
def multiplexers ():

    return [lambda n: index * n for index in range (4)]

print([m (2) for m in multiplexers ()])
```

```
Python 2.7.10 (default, Jul 14 2015, 19:46:27)
[GCC 4.8.2] on linux

[6, 6, 6, 6]
```

The output of the above code is <[6, 6, 6, 6]>. It's because of the late binding as the value of the variable <index> gets looked up after a call to any of the multiplexer functions.

Q-10. What is the result of the below lines of code?

Here is the example code.

```
def fast (items= []):
    items.append (1)
    return items

print(fast ())
print(fast ())
```

Ans. The above code will give the following result.

```
Python 2.7.10 (default, Jul 14 2015, 19:46:27)
[GCC 4.8.2] on linux

[1]
[1, 1]
```

The function <fast> evaluates its arguments only once after the function gets defined. However, since <items> is a list, it'll get modified by appending a <1> to it.

Q-11. What is the result of the below Python code?

```
keyword = 'aeioubcdfg'
print(keyword [:3] + keyword [3:])
```

Ans. The above code will produce the following result.

```
<'aeioubcdfg'>
```

In Python, while performing string slicing, whenever the indices of both slices collide, an <+> operator gets applied to concatenate them.

Q-12. How would you produce a list with unique elements from a list with duplicate elements?

Ans. Iterating the list is not a desirable solution. The right answer should look like this.

```
duplicates = ['a','b','c','d','d','d','e','a','b','f','g','g','h']
uniqueItems = list(set(duplicates))
print(sorted(uniqueItems))
```

```
Python 2.7.10 (default, Jul 14 2015, 19:46:27)
[GCC 4.8.2] on linux

['a', 'b', 'c', 'd', 'e', 'f', 'g', 'h']
```

Q-13. Can you iterate over a list of words and use a dictionary to keep track of the frequency(count) of each word? Consider the below example.

```
{'Number':Frequency, '2':2, '3':2}
```

Ans. Please find the below code.

```
def dic(words):
  wordList = {}
  for index in words:
    try:
      wordList[index] += 1
    except KeyError:
      wordList[index] = 1
  return wordList

wordList='1,3,2,4,5,3,2,1,4,3,2'.split(',')
print(wordList)
print(dic(wordList))
```

Copy

```
Python 2.7.10 (default, Jul 14 2015, 19:46:27)
[GCC 4.8.2] on linux

['1', '3', '2', '4', '5', '3', '2', '1', '4', '3', '2']
{'1': 2, '3': 3, '2': 3, '5': 1, '4': 2}
```

Q-14. What is the result of the following Python code?

Copy

```
class Test(object):
    def __init__(self):
        self.x = 1

t = Test()
print(t.x)
print(t.x)
print(t.x)
print(t.x)
```

Ans. All print statements will display <1>. It's because the value of an object's attribute(x) is never changing.

Copy

```
Python 2.7.10 (default, Jul 14 2015, 19:46:27)
[GCC 4.8.2] on linux

1
1
1
1
```

Also, <x> becomes a part of the public members of class Test.

Hence, it can be accessed directly.

Q-15. Can you describe what's wrong with the below code?

```
testProc([1, 2, 3]) # Explicitly passing in a list
testProc()  # Using a default empty list

def testProc(n = []):
    # Do something with n

print(n)
```

Ans. The above code would throw a <NameError>.

The variable n is local to the function <testProc> and can't be accessed outside. So, printing it won't be possible. We have compiled 50 more Python programming interview questions to help you secure your dream job. Please do check out them.

Before Wrap up

We are dedicated to providing fresh and valuable content to our readers, ensuring their satisfaction with the latest compilation of Python interview questions and answers.

If you have any suggestions for new topics, kindly inform us, and we will gladly incorporate them into our plans. Your feedback regarding the post and its content is highly appreciated.

Lastly, we need your support to continue. If you like our tutorials, share this post on social media like Facebook/Twitter.

Keep Learning.
TechBeamers