Top 100 Python Interview Questions You Must Prepare In 2019

Last updated on Aug 14,2019 *379.7K Views*

Aayushi Johari
A technophile who likes writing about different technologies and spreading knowledge.

Python Certification is the most sought-after skill in programming domain. In this Python Interview Questions blog, I will introduce you to the most frequently asked questions in Python interviews. Our Python Interview Questions is the one-stop resource from where you can boost your interview preparation. We have 100+ questions on **Python Programming** basics which will help you with different expertise levels to reap the maximum benefit from our blog.

Edureka 2019 Tech Career Guide is out! Hottest job roles, precise learning paths, industry outlook & more in the guide. ***Download*** now.
Let us start by taking a look at some of the most frequently asked Python interview questions,

Q1. What is the difference between list and tuples in Python?
Q2. What are the key features of Python?
Q3. What type of language is python?
Q4. How is Python an interpreted language?
Q5. What is pep 8?
Q6. How is memory managed in Python?
Q7. What is name space in Python?
Q8. What is PYTHON PATH?
Q9. What are python modules?
Q10. What are local variables and global variables in Python?

We have compiled a list of top Python interview questions which are classified into 7 sections, namely:

- Basic Interview Questions
- OOPS Interview Questions
- Basic Python Programs
- Python Libraries Interview Questions
- Web Scraping Interview Questions
- Data Analysis Interview Questions
- Multiple Choice Questions (MCQ)

Before moving ahead, you may go through the recording of Python Interview Questions where our instructor has shared his experience and expertise that will help you to crack any Python Interview:

Python Interview Questions And Answers | Python Training | Edureka

If you have other doubts regarding Python, feel free to post them in our **QnA Forum**. Our expert team will get back to you at the earliest.

Basic Python Interview Questions

Q1. What is the difference between list and tuples in Python?

LIST	TUPLES
Lists are mutable i.e they can be edited.	Tuples are immutable (tuples are lists which can't be edited).
Lists are slower than tuples.	Tuples are faster than list.
Syntax: list_1 = [10, 'Chelsea', 20]	Syntax: tup_1 = (10, 'Chelsea' , 20)

LIST vs TUPLES

Q2. What are the key features of Python?

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

Q3. What type of language is python? Programming or scripting?

Ans: Python is capable of scripting, but in general sense, it is considered as a general-purpose programming language. To know more about Scripting, you can refer to the Python Scripting Tutorial.

Q4.How is Python an interpreted language?

Ans: An interpreted language is any programming language which is not in machine level code before runtime. Therefore, Python is an interpreted language.

Q5.What is pep 8?

Ans: PEP stands for **Python Enhancement Proposal.** It is a set of rules that specify how to format Python code for maximum readability.

Q6. How is memory managed in Python?

Ans:

1. Memory management in python is managed by ***Python private heap space***. All Python objects and data structures are located in a private heap. The programmer does not have access to this private heap. The python interpreter takes care of this instead.
2. The allocation of heap space for Python objects is done by Python's memory manager. The core API gives access to some tools for the programmer to code.
3. Python also has an inbuilt garbage collector, which recycles all the unused memory and so that it can be made available to the heap space.

Q7. What is namespace in Python?

Ans: A namespace is a naming system used to make sure that names are unique to avoid naming conflicts.

Q8. What is PYTHONPATH?

Ans: It is an environment variable which is used when a module is imported. Whenever a module is imported, PYTHONPATH is also looked up to check for the presence of the imported modules in various directories. The interpreter uses it to determine which module to load.

Q9. What are python modules? Name some commonly used built-in modules in Python?

Ans: Python modules are files containing Python code. This code can either be functions classes or variables. A Python module is a .py file containing executable code.

Some of the commonly used built-in modules are:

- os
- sys
- math
- random
- data time
- JSON

Q10.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 present in the local space and not in the global space.

Example:

```
a=2
def add():
b=3
c=a+b
print(c)
add()
```

Output: 5

When you try to access the local variable outside the function add(), it will throw an error.

Q11. Is python case sensitive?

Ans: Yes. Python is a case sensitive language.

Q12.What is type conversion in Python?

Ans: Type conversion refers to the conversion of one data type iinto another.

int() – converts any data type into integer type

float() – converts any data type into float type

ord() – converts characters into integer

hex() – converts integers to hexadecimal

oct() – converts integer to octal

tuple() – This function is used to convert to a tuple.

set() – This function returns the type after converting to set.

list() – This function is used to convert any data type to a list type.

dict() – This function is used to convert a tuple of order (key,value) into a dictionary.

str() – Used to convert integer into a string.

complex(real,imag) – This functionconverts real numbers to complex(real,imag) number.

Q13. How to install Python on Windows and set path variable?

Ans: To install Python on Windows, follow the below steps:

- Install python from this link: https://www.python.org/downloads/
- After this, install it on your PC. Look for the location where PYTHON has been installed on your PC using the following command on your command prompt: cmd python.
- Then go to advanced system settings and add a new variable and name it as PYTHON_NAME and paste the copied path.
- Look for the path variable, select its value and select 'edit'.
- Add a semicolon towards the end of the value if it's not present and then type %PYTHON_HOME%

Q14. Is indentation required in python?

Ans: Indentation is necessary for Python. It specifies a block of code. All code within loops, classes, functions, etc is specified within an indented block. It is usually done using four space characters. If your code is not indented necessarily, it will not execute accurately and will throw errors as well.

Q15. What is the difference between Python Arrays and lists?

Ans: Arrays and lists, in Python, have the same way of storing data. But, arrays can hold only a single data type elements whereas lists can hold any data type elements.

Example:

```
import array as arr
My_Array=arr.array('i',[1,2,3,4])
My_list=[1,'abc',1.20]
print(My_Array)
print(My_list)
```

Output:

array('i', [1, 2, 3, 4]) [1, 'abc', 1.2]

Q16. What are functions in Python?

Ans: A function is a block of code which is executed only when it is called. To define a Python function, the **def** keyword is used.

Example:

```
def Newfunc():
print("Hi, Welcome to Edureka")
Newfunc(); #calling the function
```

Output: Hi, Welcome to Edureka

Q17.What is __init__?

Ans: __init__ is a method or constructor in Python. This method is automatically called to allocate memory when a new object/ instance of a class is created. All classes have the __init__ method.

Here is an example of how to use it.

```
class Employee:
def __init__(self, name, age,salary):
self.name = name
self.age = age
self.salary = 20000
E1 = Employee("XYZ", 23, 20000)
# E1 is the instance of class Employee.
#__init__ allocates memory for E1.
print(E1.name)
print(E1.age)
print(E1.salary)
```

Output:

XYZ

23

20000

Q18.What is a lambda function?

Ans: An anonymous function is known as a lambda function. This function can have any number of parameters but, can have just one statement.

Example:

```
a = lambda x,y : x+y
print(a(5, 6))
```

Output: 11

Q19. What is self in Python?

Ans: Self is an instance or an object of a class. In Python, this is explicitly included as the first parameter. However, this is not the case in Java where it's optional. It helps to differentiate between the methods and attributes of a class with local variables.

The self variable in the init method refers to the newly created object while in other methods, it refers to the object whose method was called.

Q20. How does break, continue and pass work?

Break	Allows loop termination when some condition is met and the control is transferred to the next statement.
Continue	Allows skipping some part of a loop when some specific condition is met and the control is transferred to the beginning of the loop
Pass	Used when you need some block of code syntactically, but you want to skip its execution. This is basically a null operation. Nothing happens when this is executed.

Q21. What does [::-1} do?

Ans: [::-1] is used to reverse the order of an array or a sequence.
For example:

```
import array as arr
My_Array=arr.array('i',[1,2,3,4,5])
My_Array[::-1]
```

Output: array('i', [5, 4, 3, 2, 1])

[::-1] reprints a reversed copy of ordered data structures such as an array or a list. the original array or list remains unchanged.

Q22. How can you randomize the items of a list in place in Python?

Ans: Consider the example shown below:

```
from random import shuffle
x = ['Keep', 'The', 'Blue', 'Flag', 'Flying', 'High']
shuffle(x)
print(x)
```

The output of the following code is as below.

```
['Flying', 'Keep', 'Blue', 'High', 'The', 'Flag']
```

Q23. What are python iterators?

Ans: Iterators are objects which can be traversed though or iterated upon.

Q24. How can you generate random numbers in Python?

Ans: Random module is the standard module that is used to generate a random number. The method is defined as:

```
import random
random.random
```

The statement random.random() method return the floating point number that is in the range of [0, 1). The function generates random float numbers. The methods that are used with the random class are the bound methods of the hidden instances. The instances of the Random can be done to show the multi-threading programs that creates a different instance of individual threads. The other random generators that are used in this are:

1. randrange(a, b): it chooses an integer and define the range in-between [a, b). It returns the elements by selecting it randomly from the range that is specified. It doesn't build a range object.
2. uniform(a, b): it chooses a floating point number that is defined in the range of [a,b).lyt returns the floating point number
3. normalvariate(mean, sdev): it is used for the normal distribution where the mu is a mean and the sdev is a sigma that is used for standard deviation.
4. The Random class that is used and instantiated creates an independent multiple random number generators.

Q25. What is the difference between range & xrange?

Ans: 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.

Q26. How do you write comments in python?

Ans: Comments in Python start with a # character. However, alternatively at times, commenting is done using docstrings(strings enclosed within triple quotes).

Example:

```
#Comments in Python start like this
print("Comments in Python start with a #")
```

Output: Comments in Python start with a #

Q27. What is pickling and unpickling?

Ans: 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.

Q28. What are the generators in python?

Ans: Functions that return an iterable set of items are called generators.

Q29. How will you capitalize the first letter of string?

Ans: In Python, the capitalize() method capitalizes the first letter of a string. If the string already consists of a capital letter at the beginning, then, it returns the original string.

Q30. How will you convert a string to all lowercase?

Ans: To convert a string to lowercase, lower() function can be used.

Example:

```
stg='ABCD'
print(stg.lower())
```

Output: abcd

Q31. How to comment multiple lines in python?

Ans: Multi-line comments appear in more than one line. All the lines to be commented are to be prefixed by a #. You can also a very good **shortcut method to comment multiple lines**. All you need to do is hold the ctrl key and **left click** in every place wherever you want to include a # character and type a # just once. This will comment all the lines where you introduced your cursor.

Q32.What are docstrings in Python?

Ans: Docstrings are not actually comments, but, they are ***documentation strings***. These docstrings are within triple quotes. They are not assigned to any variable and therefore, at times, serve the purpose of comments as well.

Example:

```
"""
Using docstring as a comment.
This code divides 2 numbers
"""
x=8
y=4
z=x/y
print(z)
```

Output: 2.0

Q33. What is the purpose of is, not and in operators?

Ans: Operators are special functions. They take one or more values and produce a corresponding result.

is: returns true when 2 operands are true (Example: “a” is ‘a’)

not: returns the inverse of the boolean value

in: checks if some element is present in some sequence

Q34. What is the usage of help() and dir() function in Python?

Ans: Help() and dir() both functions are accessible from the Python interpreter and used for viewing a consolidated dump of built-in functions.

1. Help() function: The help() function is used to display the documentation string and also facilitates you to see the help related to modules, keywords, attributes, etc.
2. Dir() function: The dir() function is used to display the defined symbols.

Q35. Whenever Python exits, why isn’t all the memory de-allocated?

Ans:

1. 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.
2. It is impossible to de-allocate those portions of memory that are reserved by the C library.
3. On exit, because of having its own efficient clean up mechanism, Python would try to de-allocate/destroy every other object.

Q36. What is a dictionary in Python?

Ans: 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]
```

```
India
```

```
print dict[Capital]
```

```
Delhi
```

```
print dict[PM]
```

```
Modi
```

Q37. How can the ternary operators be used in python?

Ans: The Ternary operator is the operator that is used to show the conditional statements. This consists of the true or false values with a statement that has to be evaluated for it.

Syntax:

The Ternary operator will be given as:
[on_true] if [expression] else [on_false]x, y = 25, 50big = x if x < y else y

Example:

The expression gets evaluated like if x<y else y, in this case if x<y is true then the value is returned as big=x and if it is incorrect then big=y will be sent as a result.

Q38. What does this mean: *args, **kwargs? And why would we use it?

Ans: We 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 don't know how many keyword arguments will be passed to a function, or it can be used to pass the values of a dictionary as keyword arguments. The identifiers args and kwargs are a convention, you could also use *bob and **billy but that would not be wise.

Q39. What does len() do?

Ans: It is used to determine the length of a string, a list, an array, etc.

Example:

```
stg='ABCD'
len(stg)
```

Q40. Explain split(), sub(), subn() methods of "re" module in Python.

Ans: 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.

Q41. What are negative indexes and why are they used?

Ans: 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.

Q42. What are Python packages?

Ans: Python packages are namespaces containing multiple modules.

Q43.How can files be deleted in Python?

Ans: To delete a file in Python, you need to import the OS Module. After that, you need to use the os.remove() function.

Example:

```
import os
os.remove("xyz.txt")
```

Q44. What are the built-in types of python?

Ans: Built-in types in Python are as follows –

- Integers
- Floating-point
- Complex numbers
- Strings
- Boolean
- Built-in functions

Q45. What advantages do NumPy arrays offer over (nested) Python lists?

Ans:

1. Python's lists are efficient general-purpose containers. They support (fairly) efficient insertion, deletion, appending, and concatenation, and Python's list comprehensions make them easy to construct and manipulate.

2. They have certain limitations: they don't support "vectorized" operations like elementwise addition and multiplication, and the fact that they can contain objects of differing types mean that Python must store type information for every element, and must execute type dispatching code when operating on each element.
3. NumPy is not just more efficient; it is also more convenient. You get a lot of vector and matrix operations for free, which sometimes allow one to avoid unnecessary work. And they are also efficiently implemented.
4. NumPy array is faster and You get a lot built in with NumPy, FFTs, convolutions, fast searching, basic statistics, linear algebra, histograms, etc.

Q46. How to add values to a python array?

Ans: Elements can be added to an array using the **append()**, **extend()** and the **insert (i,x)** functions.

Example:

```
a=arr.array('d', [1.1 , 2.1 ,3.1] )
a.append(3.4)
print(a)
a.extend([4.5,6.3,6.8])
print(a)
a.insert(2,3.8)
print(a)
```

Output:

array('d', [1.1, 2.1, 3.1, 3.4])

array('d', [1.1, 2.1, 3.1, 3.4, 4.5, 6.3, 6.8])

array('d', [1.1, 2.1, 3.8, 3.1, 3.4, 4.5, 6.3, 6.8])

Q47. How to remove values to a python array?

Ans: Array elements can be removed using **pop()** or **remove()** method. The difference between these two functions is that the former returns the deleted value whereas the latter does not.

Example:

```
a=arr.array('d', [1.1, 2.2, 3.8, 3.1, 3.7, 1.2, 4.6])
print(a.pop())
print(a.pop(3))
a.remove(1.1)
print(a)
```

Output:

4.6

3.1

array('d', [2.2, 3.8, 3.7, 1.2])

Q48. Does Python have OOps concepts?

Ans: Python is an object-oriented programming language. This means that any program can be solved in python by creating an object model. However, Python can be treated as procedural as well as structural language.

Q49. What is the difference between deep and shallow copy?

Ans: *Shallow copy* is used when a new instance type gets created and it keeps the values that are copied in the new instance. Shallow copy is used to copy the reference pointers just like it copies the values. These references point to the original objects and the changes made in any member of the class will also affect the original copy of it. Shallow copy allows faster execution of the program and it depends on the size of the data that is used.

Deep copy is used to store the values that are already copied. Deep copy doesn't copy the reference pointers to the objects. It makes the reference to an object and the new object that is pointed by some other object gets stored. The changes made in the original copy won't affect any other copy that uses the object. Deep copy makes execution of the program slower due to making certain copies for each object that is been called.

Q50. How is Multithreading achieved in Python?

Ans:

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

Q51. What is the process of compilation and linking in python?

Ans: The compiling and linking allows the new extensions to be compiled properly without any error and the linking can be done only when it passes the compiled procedure. If the dynamic loading is used then it depends on the style that is being provided with the system. The python interpreter can be used to provide the dynamic loading of the configuration setup files and will rebuild the interpreter.

The steps that are required in this as:

1. Create a file with any name and in any language that is supported by the compiler of your system. For example file.c or file.cpp
2. Place this file in the Modules/ directory of the distribution which is getting used.
3. Add a line in the file Setup.local that is present in the Modules/ directory.
4. Run the file using spam file.o
5. After a successful run of this rebuild the interpreter by using the make command on the top-level directory.
6. If the file is changed then run rebuildMakefile by using the command as 'make Makefile'.

Q52. What are Python libraries? Name a few of them.

Python libraries are a collection of Python packages. Some of the majorly used python libraries are – Numpy, Pandas, Matplotlib, Scikit-learn and many more.

Q53. What is split used for?

The split() method is used to separate a given string in Python.

Example:

```
a="edureka python"
print(a.split())
```

Output: ['edureka', 'python']

Q54. How to import modules in python?

Modules can be imported using the **import** keyword. You can import modules in three ways-

Example:

```
import array           #importing using the original module name
import array as arr    # importing using an alias name
from array import *    #imports everything present in the array module
```

OOPS Interview Questions

Q55. Explain Inheritance in Python with an example.

Ans: 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:

1. Single Inheritance – where a derived class acquires the members of a single super class.
2. Multi-level inheritance – a derived class d1 in inherited from base class base1, and d2 are inherited from base2.
3. Hierarchical inheritance – from one base class you can inherit any number of child classes
4. Multiple inheritance – a derived class is inherited from more than one base class.

Q56. How are classes created in Python?

Ans: Class in Python is created using the **class** keyword.

Example:

```
class Employee:
def __init__(self, name):
self.name = name
E1=Employee("abc")
print(E1.name)
```

Output: abc

Q57. What is monkey patching in Python?

Ans: In Python, the term monkey patch only refers to dynamic modifications of a class or module at run-time.

Consider the below example:

```
# m.py
class MyClass:
def f(self):
print "f()"
```

We can then run the monkey-patch testing like this:

```
import m
def monkey_f(self):
print "monkey_f()"

m.MyClass.f = monkey_f
obj = m.MyClass()
obj.f()
```

The output will be as below:

```
monkey_f()
```

As we can see, we did make some changes in the behavior of *f()* in *MyClass* using the function we defined, *monkey_f()*, outside of the module *m*.

Q58. Does python support multiple inheritance?

Ans: Multiple inheritance means that a class can be derived from more than one parent classes. Python does support multiple inheritance, unlike Java.

Q59. What is Polymorphism in Python?

Ans: Polymorphism means the ability to take multiple forms. So, for instance, if the parent class has a method named ABC then the child class also can have a method with the same name ABC having its own parameters and variables. Python allows polymorphism.

Q60. Define encapsulation in Python?

Ans: Encapsulation means binding the code and the data together. A Python class in an example of encapsulation.

Q61. How do you do data abstraction in Python?

Ans: Data Abstraction is providing only the required details and hiding the implementation from the world. It can be achieved in Python by using interfaces and abstract classes.

Q62.Does python make use of access specifiers?

Ans: Python does not deprive access to an instance variable or function. Python lays down the concept of prefixing the name of the variable, function or method with a single or double underscore to imitate the behavior of protected and private access specifiers.

Q63. How to create an empty class in Python?

Ans: An empty class is a class that does not have any code defined within its block. It can be created using the *pass* keyword. However, you can create objects of this class outside the class itself. IN PYTHON THE PASS command does nothing when its executed. it's a null statement.

For example-

```
class a:
  &amp;nbsp; pass
obj=a()
obj.name="xyz"
print("Name = ",obj.name)
```

Output:

```
Name =  xyz
```

Q64. What does an object() do?

Ans: It returns a featureless object that is a base for all classes. Also, it does not take any parameters.

Basic Python Programs

Q65. Write a program in Python to execute the Bubble sort algorithm.

```
def bs(a):&amp;nbsp; &amp;nbsp; &amp;nbsp; &amp;nbsp; &amp;nbsp; &amp;nbsp; &amp;nbsp;# a = name of list
  &amp;nbsp; b=len(a)-1&amp;nbsp; &amp;nbsp; &amp;nbsp; &amp;nbsp; &amp;nbsp;# minus 1 because we always compare 2 adjacent values
  &amp;nbsp; &amp;nbsp; &amp;nbsp; &amp;nbsp; &amp;nbsp; &amp;nbsp; &amp;nbsp; &amp;nbsp; &amp;nbsp; &amp;nbsp; &amp;nbsp; &amp;nbsp; &amp;nbsp; &amp;nbsp;
  &amp;nbsp; for x in range(b):
  &amp;nbsp; &amp;nbsp; &amp;nbsp; for y in range(b-x):
  &amp;nbsp; &amp;nbsp; &amp;nbsp; &amp;nbsp; &amp;nbsp; if a[y]>a[y+1]:
  &amp;nbsp; &amp;nbsp; &amp;nbsp; &amp;nbsp; &amp;nbsp; &amp;nbsp; &amp;nbsp; a[y],a[y+1]=a[y+1],a[y]
  &amp;nbsp; return a
a=[32,5,3,6,7,54,87]
bs(a)
```

Output: [3, 5, 6, 7, 32, 54, 87]

Q66. Write a program in Python to produce Star triangle.

```
def pyfunc(r):
    for x in range(r):
        print(' '*(r-x-1)+'*'*(2*x+1))
pyfunc(9)
```

Output:

```
        *
       ***
      *****
     *******
    *********
   ***********
  *************
 ***************
*****************
```

Q67. Write a program to produce Fibonacci series in Python.

```
# Enter number of terms needed&amp;nbsp; &amp;nbsp; &amp;nbsp; &amp;nbsp; &amp;nbsp; &amp;nbsp; &amp;nbsp; &amp;nbsp; &amp;nbsp;
&amp;nbsp;#0,1,1,2,3,5....
a=int(input("Enter the terms"))
f=0&amp;nbsp; &amp;nbsp; &amp;nbsp; &amp;nbsp; &amp;nbsp; &amp;nbsp; &amp;nbsp; &amp;nbsp; &amp;nbsp; &amp;nbsp; &amp;nbsp; &amp;nbsp; &amp;nbsp; &amp;nbsp; &amp;nbsp;
&amp;nbsp; &amp;nbsp; &amp;nbsp; &amp;nbsp; &amp;nbsp; &amp;nbsp;#first element of series
s=1&amp;nbsp; &amp;nbsp; &amp;nbsp; &amp;nbsp; &amp;nbsp; &amp;nbsp; &amp;nbsp; &amp;nbsp; &amp;nbsp; &amp;nbsp; &amp;nbsp; &amp;nbsp; &amp;nbsp; &amp;nbsp; &amp;nbsp;
&amp;nbsp; &amp;nbsp; &amp;nbsp; &amp;nbsp; &amp;nbsp; &amp;nbsp;#second element of series
if a<=0:
  &amp;nbsp; print("The requested series is
",f)
else:
  &amp;nbsp; print(f,s,end=" ")
  &amp;nbsp; for x in range(2,a):
  &amp;nbsp; &amp;nbsp; &amp;nbsp; next=f+s&amp;nbsp; &amp;nbsp; &amp;nbsp; &amp;nbsp; &amp;nbsp; &amp;nbsp; &amp;nbsp; &amp;nbsp; &amp;nbsp; &amp;nbsp; &amp;nbsp;
&amp;nbsp; &amp;nbsp;
  &amp;nbsp; &amp;nbsp; &amp;nbsp; print(next,end=" ")
  &amp;nbsp; &amp;nbsp; &amp;nbsp; f=s
  &amp;nbsp; &amp;nbsp; &amp;nbsp; s=next</pre>
```

Output: Enter the terms 5 0 1 1 2 3

Q68. Write a program in Python to check if a number is prime.

```
a=int(input("enter number"))&amp;nbsp; &amp;nbsp; &amp;nbsp;
if a>1:
  &amp;nbsp; for x in range(2,a):
  &amp;nbsp; &amp;nbsp; &amp;nbsp; if(a%x)==0:
  &amp;nbsp; &amp;nbsp; &amp;nbsp; &amp;nbsp; &amp;nbsp; print("not prime")
  &amp;nbsp; &amp;nbsp; &amp;nbsp; &amp;nbsp; &amp;nbsp; break
  &amp;nbsp; else:
  &amp;nbsp; &amp;nbsp; &amp;nbsp; print("Prime")
else:
  &amp;nbsp; print("not prime")
```

Output:

enter number 3

Prime

Q69. Write a program in Python to check if a sequence is a Palindrome.

```
a=input("enter sequence")
b=a[::-1]
if a==b:
  &amp;nbsp; print("palindrome")
else:
  &amp;nbsp; print("Not a Palindrome")
```

Output:

enter sequence 323 palindrome

Q70. 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.

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

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

We will now try to transform this into a single line.

```
count sum(1 for line in fh for character in line if character.isupper())
```

Q71. Write a sorting algorithm for a numerical dataset in Python.

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

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

Q72. Looking at the below code, write down the final values of A0, A1, ...An.

```
A0 = dict(zip(('a','b','c','d','e'),(1,2,3,4,5)))
A1 = range(10)A2 = sorted([i for i in A1 if i in A0])
A3 = sorted([A0[s] for s in A0])
A4 = [i for i in A1 if i in A3]
A5 = {i:i*i for i in A1}
A6 = [[i,i*i] for i in A1]
print(A0,A1,A2,A3,A4,A5,A6)
```

Ans: The following will be the final outputs of A0, A1, ... A6

```
A0 = {'a': 1, 'c': 3, 'b': 2, 'e': 5, 'd': 4} # the order may vary
A1 = range(0, 10)
A2 = []
A3 = [1, 2, 3, 4, 5]
A4 = [1, 2, 3, 4, 5]
A5 = {0: 0, 1: 1, 2: 4, 3: 9, 4: 16, 5: 25, 6: 36, 7: 49, 8: 64, 9: 81}
A6 = [[0, 0], [1, 1], [2, 4], [3, 9], [4, 16], [5, 25], [6, 36], [7, 49], [8, 64], [9, 81]]
```

Python Libraries Interview Questions

Q73. Explain what Flask is and its benefits?

Ans: Flask is a web microframework for Python based on "Werkzeug, Jinja2 and good intentions" BSD license. Werkzeug and Jinja2 are two of its dependencies. This means it will have little to no dependencies on external libraries. It makes the framework light while there is a little dependency to update and fewer security bugs.

A session basically allows you to remember information from one request to another. In a flask, a session 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.

Q74. Is Django better than Flask?

Ans: Django and Flask map the URL's or addresses typed in the web browsers to functions in Python.

Flask is much simpler compared to Django but, Flask does not do a lot for you meaning you will need to specify the details, whereas Django does a lot for you wherein you would not need to do much work. Django consists of prewritten code, which the user will need to analyze whereas Flask gives the users to create their own code, therefore, making it simpler to understand the code. Technically both are equally good and both contain their own pros and cons.

Q75. Mention the differences between Django, Pyramid and Flask.

Ans:

- 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 is built 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.
- Django can also be used for larger applications just like Pyramid. It includes an ORM.

Q76. Discuss Django architecture.

Ans: Django MVT Pattern:

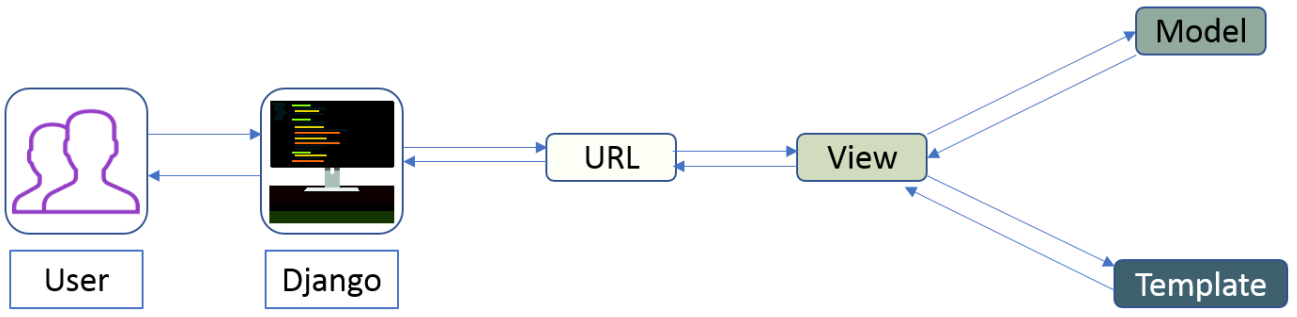

Figure: *Python Interview Questions – Django Architecture*

The developer provides the Model, the view and the template then just maps it to a URL and Django does the magic to serve it to the user.

Q77. Explain how you can set up the Database in Django.

Ans: You can use the command edit mysite/setting.py, it is a normal python module with module level representing Django settings.

Django uses SQLite by default; it is easy for Django users as such it won't require any other type of installation. In the case your database choice is different that you have to the following keys in the DATABASE 'default' item to match your database connection settings.

- **Engines**: you can change the database by using 'django.db.backends.sqlite3' , 'django.db.backeneds.mysql', 'django.db.backends.postgresql_psycopg2', 'django.db.backends.oracle' and so on
- **Name**: The name of your database. In the case if you are using SQLite as your database, in that case, database will be a file on your computer, Name should be a full absolute path, including the file name of that file.
- If you are not choosing SQLite as your database then settings like Password, Host, User, etc. must be added.

Django uses SQLite as a default database, it stores data as a single file in the filesystem. If you do have a database server—PostgreSQL, MySQL, Oracle, MSSQL—and want to use it rather than SQLite, then use your database's administration tools to create a new database for your Django project. Either way, with your (empty) database in place, all that remains is to tell Django how to use it. This is where your project's settings.py file comes in.

We will add the following lines of code to the *setting.py* file:

```
DATABASES = {
     'default': {
          'ENGINE' : 'django.db.backends.sqlite3',
          'NAME' : os.path.join(BASE_DIR, 'db.sqlite3'),
     }
}
```

Q78. Give an example how you can write a VIEW in Django?

Ans: This is how we can use write a view in Django:

```
from django.http import HttpResponse
import datetime

def Current_datetime(request):
     now = datetime.datetime.now()
     html = "<html><body>It is now %s</body></html> % now
     return HttpResponse(html)
```

Returns the current date and time, as an HTML document

Q79. Mention what the Django templates consist of.

Ans: The template is a simple text file. It can create any text-based format like XML, CSV, HTML, etc. A template contains variables that get replaced with values when the template is evaluated and tags (% tag %) that control the logic of the template.

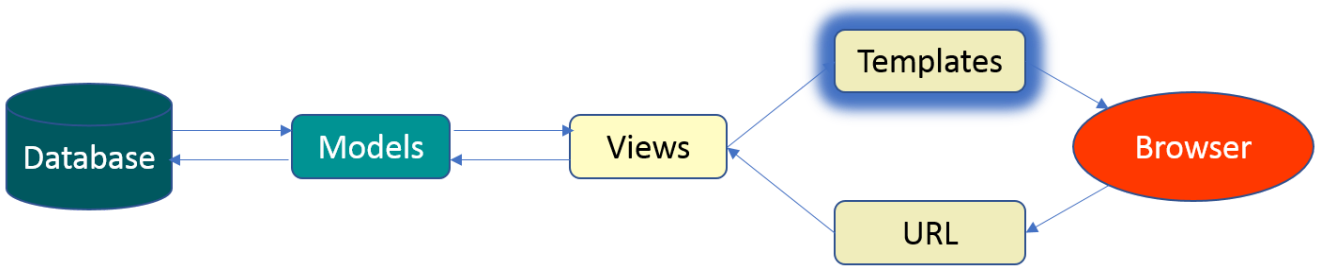

Figure: *Python Interview Questions – Django Template*

Q80. Explain the use of session in Django framework?

Ans: Django provides a session that lets you store and retrieve data on a per-site-visitor basis. Django abstracts the process of sending and receiving cookies, by placing a session ID cookie on the client side, and storing all the related data on the server side.

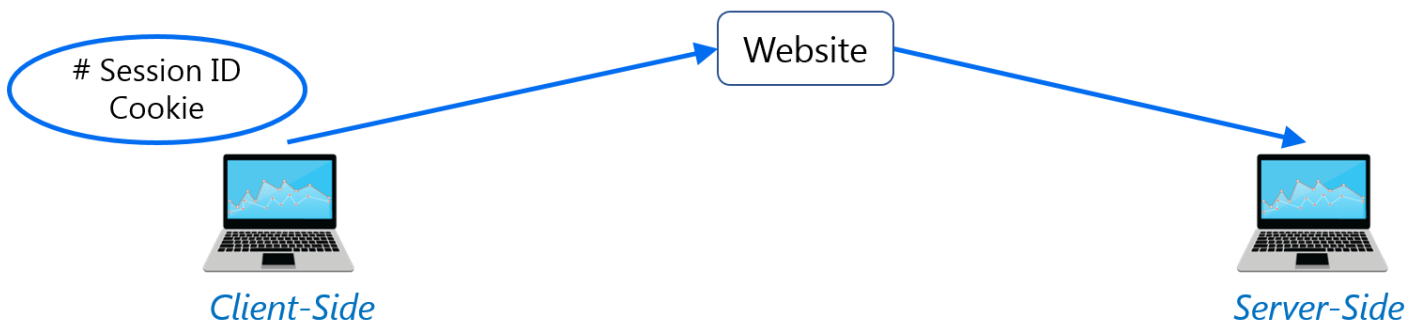

Figure: *Python Interview Questions – Django Framework*

So the data itself is not stored client side. This is nice from a security perspective.

Q81. List out the inheritance styles in Django.

Ans: In Django, there are three possible inheritance styles:

1. Abstract Base Classes: This style is used when you only want parent's class to hold information that you don't want to type out for each child model.
2. Multi-table Inheritance: This style is used If you are sub-classing an existing model and need each model to have its own database table.
3. Proxy models: You can use this model, If you only want to modify the Python level behavior of the model, without changing the model's fields.

Web Scraping – Python Interview Questions

Q82. How To Save An Image Locally Using Python Whose URL Address I Already Know?

Ans: We will use the following code to save an image locally from an URL address

```
import urllib.request
urllib.request.urlretrieve("URL", "local-filename.jpg")
```

Q83. How can you Get the Google cache age of any URL or web page?

Ans: Use the following URL format:

`http://webcache.googleusercontent.com/search?q=cache:URLGOESHERE`

Be sure to replace "URLGOESHERE" with the proper web address of the page or site whose cache you want to retrieve and see the time for. For example, to check the Google Webcache age of edureka.co you'd use the following URL:

`http://webcache.googleusercontent.com/search?q=cache:edureka.co`

Q84. You are required to scrap data from IMDb top 250 movies page. It should only have fields movie name, year, and rating.

Ans: We will use the following lines of code:

```
from bs4 import BeautifulSoup

import requests
import sys

url = 'http://www.imdb.com/chart/top'
response = requests.get(url)
soup = BeautifulSoup(response.text)
tr = soup.findChildren("tr")
tr = iter(tr)
next(tr)

for movie in tr:
title = movie.find('td', {'class': 'titleColumn'} ).find('a').contents[0]
year = movie.find('td', {'class': 'titleColumn'} ).find('span', {'class': 'secondaryInfo'}).contents[0]
rating = movie.find('td', {'class': 'ratingColumn imdbRating'} ).find('strong').contents[0]
row = title + ' - ' + year + ' ' + ' ' + rating

print(row)
```

The above code will help scrap data from IMDb's top 250 list

Data Analysis – Python Interview Questions

Q85. What is map function in Python?

Ans: *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.

Q86. Is python numpy better than lists?

Ans: We use python numpy array instead of a list because of the below three reasons:

1. Less Memory
2. Fast
3. Convenient

For more information on these parameters, you can refer to this section – Numpy Vs List.

Q87. How to get indices of N maximum values in a NumPy array?

Ans: 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])
```

Output

```
[ 4 3 1 ]
```

Q88. How do you calculate percentiles with Python/ NumPy?

Ans: We can calculate percentiles with the following code

```
import numpy as np
a = np.array([1,2,3,4,5])
p = np.percentile(a, 50) #Returns 50th percentile, e.g. median
print(p)
```

Output

```
3
```

Q89. What is the difference between NumPy and SciPy?

Ans:

1. In an ideal world, NumPy would contain nothing but the array data type and the most basic operations: indexing, sorting, reshaping, basic elementwise functions, et cetera.

2. 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.
3. 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.
4. 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.

Q90. How do you make 3D plots/visualizations using NumPy/SciPy?

Ans: Like 2D plotting, 3D graphics is beyond the scope of NumPy and SciPy, but just as in the 2D case, packages exist that integrate with NumPy. Matplotlib provides basic 3D plotting in the mplot3d subpackage, whereas Mayavi provides a wide range of high-quality 3D visualization features, utilizing the powerful VTK engine.

Multiple Choice Questions (MCQ)

Q91. Which of the following statements create a dictionary? (Multiple Correct Answers Possible)

a) d = {}
b) d = {"john":40, "peter":45}
c) d = {40:"john", 45:"peter"}
d) d = (40:"john", 45:"50")

Answer: b, c & d.

Dictionaries are created by specifying keys and values.

Q92. Which one of these is floor division?

a) /
b) //
c) %
d) None of the mentioned

Answer: b) //

When both of the operands are integer then python chops out the fraction part and gives you the round off value, to get the accurate answer use floor division. For ex, 5/2 = 2.5 but both of the operands are integer so answer of this expression in python is 2. To get the 2.5 as the answer, use floor division using //. So, 5//2 = 2.5

Q93. What is the maximum possible length of an identifier?

a) 31 characters
b) 63 characters
c) 79 characters
d) None of the above

Answer: d) None of the above

Identifiers can be of any length.

Q94. Why are local variable names beginning with an underscore discouraged?

a) they are used to indicate a private variables of a class
b) they confuse the interpreter
c) they are used to indicate global variables
d) they slow down execution

Answer: a) they are used to indicate a private variable of a class

As Python has no concept of private variables, leading underscores are used to indicate variables that must not be accessed from outside the class.

Q95. 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) a b c = 1000 2000 3000

Spaces are not allowed in variable names.

Q96. What is the output of the following?

```
try:
    if '1' != 1:
        raise "someError"
    else:
        print("someError has not occured")
except "someError":
    print ("someError has occured")
```

a) someError has occured
b) someError has not occured
c) invalid code
d) none of the above

Answer: c) invalid code

A new exception class must inherit from a BaseException. There is no such inheritance here.

Q97. Suppose list1 is [2, 33, 222, 14, 25], What is list1[-1] ?

a) Error
b) None
c) 25
d) 2

Answer: c) 25

The index -1 corresponds to the last index in the list.

Q98. To open a file c:scores.txt for writing, we use

a) outfile = open(“c:scores.txt”, “r”)
b) outfile = open(“c:scores.txt”, “w”)
c) outfile = open(file = “c:scores.txt”, “r”)
d) outfile = open(file = “c:scores.txt”, “o”)

Answer: b) The location contains double slashes ( ) and w is used to indicate that file is being written to.

Q99. What is the output of the following?

```
f = None

for i in range (5):
    with open("data.txt", "w") as f:
        if i > 2:
            break

print f.closed
```

a) True
b) False
c) None
d) Error

Answer: a) True

The WITH statement when used with open file guarantees that the file object is closed when the with block exits.

Q100. When will the else part of try-except-else be executed?

a) always
b) when an exception occurs
c) when no exception occurs
d) when an exception occurs into except block

Answer: c) when no exception occurs

The else part is executed when no exception occurs.

I hope this set of Python Interview Questions will help you in preparing for your interviews. All the best!

Got a question for us? Please mention it in the comments section and we will get back to you at the earliest.

If you wish to learn Python and gain expertise in quantitative analysis, data mining, and the presentation of data to see beyond the numbers by transforming your career into Data Scientist role, check out our interactive, live-online ***Python Certification Training****. You will use libraries like Pandas, Numpy, Matplotlib, Scipy, Scikit, Pyspark and master the concepts like Python machine learning, scripts, sequence, web scraping and big data analytics leveraging Apache Spark. The training comes with 24*7 support to guide you throughout your learning period.*

WebsiteSetup

Python
Cheat Sheet

Python 3 is a truly versatile programming language, loved both by web developers, data scientists and software engineers. And there are several good reasons for that!

- Python is open-source and has a great support community,
- Plus, extensive support libraries.
- Its data structures are user-friendly.

Once you get a hang of it, your development speed and productivity will soar!

Table of Contents

Python Basics: Getting Started

Most Windows and Mac computers come with Python pre-installed. You can check that via a Command Line search. The particular appeal of Python is that you can write a program in any text editor, save it in .py format and then run via a Command Line. But as you learn to write more complex code or venture into data science, you might want to switch to an IDE or IDLE.

What is IDLE (Integrated Development and Learning)

IDLE (Integrated Development and Learning Environment) comes with every Python installation. Its advantage over other text editors is that it highlights important keywords (e.g. string functions), making it easier for you to interpret code.

Shell is the default mode of operation for Python IDLE. In essence, it's a simple loop that performs that following four steps:

- Reads the Python statement
- Evaluates the results of it
- Prints the result on the screen
- And then loops back to read the next statement.

Python shell is a great place to test various small code snippets.

Main Python Data Types

Every value in Python is called an "**object**". And every object has a specific data type. The three most-used data types are as follows:

Integers (int) — an integer number to represent an object such as "number 3".

```
Integers                    -2, -1, 0, 1, 2, 3, 4, 5
```

Floating-point numbers (float) — use them to represent floating-point numbers.

```
Floating-point numbers      -1.25, -1.0, --0.5, 0.0, 0.5, 1.0, 1.25
```

Strings — codify a sequence of characters using a string. For example, the word "hello". In Python 3, strings are immutable. If you already defined one, you cannot change it later on.

While you can modify a string with commands such as **replace()** or **join()**, they will create a copy of a string and apply modification to it, rather than rewrite the original one.

```
Strings                     'yo', 'hey', 'Hello!', 'what's up!'
```

Plus, another three types worth mentioning are **list**s, **dict**ionaries, and **tuple**s. All of them are discussed in the next sections.

For now, let's focus on the **strings**.

How to Create a String in Python

You can create a string in three ways using **single**, **double** or **triple** quotes. Here's an example of every option:

Basic Python String

```
my_string = "Let's Learn Python!"
another_string = 'It may seem difficult first, but you
can do it!'
a_long_string = '''Yes, you can even master multi-line
strings
 that cover more than one line
 with some practice'''
```

IMP! Whichever option you choose, you should stick to it and use it consistently within your program.

As the next step, you can use the **print()** function to output your string in the console window. This lets you review your code and ensure that all functions well.

Here's a snippet for that:

```
print("Let's print out a string!")
```

String Concatenation

The next thing you can master is **concatenation** — a way to add two strings together using the "+" operator. Here's how it's done:

```
string_one = "I'm reading "
string_two = "a new great book!"
string_three = string_one + string_two
```

Note: You can't apply + operator to two different data types e.g. string + integer. If you try to do that, you'll get the following Python error:

```
TypeError: Can't convert 'int' object to str implicitly
```

String Replication

As the name implies, this command lets you repeat the same string several times. This is done using * operator. Mind that this operator acts as a replicator only with string data types. When applied to numbers, it acts as a multiplier.

String replication example:

```
'Alice' * 5 'AliceAliceAliceAliceAlice'
```

And with print ()

```
print("Alice" * 5)
```

And your output will be Alice written five times in a row.

Math Operators

For reference, here's a list of other math operations you can apply towards numbers:

Operators	Operation	Example
**	Exponent	2 ** 3 = 8
%	Modulus/Remainder	22 % 8 = 6
//	Integer division	22 // 8 = 2
/	Division	22 / 8 = 2.75
*	Multiplication	3 * 3 = 9
-	Subtraction	5 – 2 = 3
+	Addition	2 + 2 = 4

How to Store Strings in Variables

Variables in Python 3 are special symbols that assign a specific storage location to a value that's tied to it. In essence, variables are like special labels that you place on some value to know where it's stored.

Strings incorporate data. So you can "pack" them inside a variable. Doing so makes it easier to work with complex Python programs.

Here's how you can store a string inside a variable.

```
my_str = "Hello World"
```

Let's break it down a bit further:

- my_str is the variable name.
- = is the assignment operator.
- "Just a random string" is a value you tie to the variable name.

Now when you print this out, you receive the string output.

```
print(my_str)
```

```
= Hello World
```

See? By using variables, you save yourself heaps of effort as you don't need to retype the complete string every time you want to use it.

Built-in Functions in Python

You already know the most popular function in Python — **print()**. Now let's take a look at its equally popular cousins that are in-built in the platform.

Input() Function

input() function is a simple way to prompt the user for some input (e.g. provide their name). All user input is stored as a string.

Here's a quick snippet to illustrate this:

```
name = input("Hi! What's your name? ")
print("Nice to meet you " + name + "!")

age = input("How old are you ")
print("So, you are already " + str(age) + " years old, "
+ name + "!")
```

When you run this short program, the results will look like this:

```
Hi! What's your name? "Jim"

Nice to meet you, Jim!

How old are you? 25

So, you are already 25 years old, Jim!
```

len() Function

len() function helps you find the length of any string, list, tuple, dictionary, or another data type. It's a handy command to determine excessive values and trim them to optimize the performance of your program.

Here's an input function example for a string:

```
# testing len()
str1 = "Hope you are enjoying our tutorial!"
print("The length of the string is :", len(str1))
```

Output:

```
The length of the string is: 35
```

filter()

Use the **Filter()** function to exclude items in an iterable object (lists, tuples, dictionaries, etc)

```
ages = [5, 12, 17, 18, 24, 32]

def myFunc(x):
  if x < 18:
    return False
  else:
    return True

adults = filter(myFunc, ages)

for x in adults:
  print(x)
```

(Optional: The PDF version of the checklist can also include a full table of all the in-built functions).

How to Define a Function

Apart from using in-built functions, Python 3 also allows you to define your own functions for your program.

To recap, a **function** is a block of coded instructions that perform a certain action. Once properly defined, a function can be reused throughout your program i.e. re-use the same code.

Here's a quick walkthrough explaining how to define a function in Python:

First, use **def** keyword followed by the function **name()**:. The parentheses can contain any parameters that your function should take (or stay empty).

```
def name():
```

Next, you'll need to add a second code line with a 4-space indent to specify what this function should do.

```
def name():
    print("What's your name?")
```

Now, you have to call this function to run the code.

```
name.py
def name():
    print("What's your name?")

hello()
```

Now, let's take a look at a defined function with a parameter — an entity, specifying an argument that a function can accept.

```
def add_numbers(x, y, z):
    a = x + y
    b = x + z
    c = y + z
    print(a, b, c)

add_numbers(1, 2, 3)
```

In this case, you pass the number 1 in for the x parameter, 2 in for the y parameter, and 3 in for the z parameter. The program will that do the simple math of adding up the numbers:

Output:

```
a = 1 + 2
b = 1 + 3
c = 2 + 3
```

How to Pass Keyword Arguments to a Function

A function can also accept keyword arguments. In this case, you can use parameters in random order as the Python interpreter will use the provided keywords to match the values to the parameters.

Here's a simple example of how you pass a keyword argument to a function.

```
# Define function with parameters
def product_info(product name, price):
    print("productname: " + product name)
    print("Price " + str(dollars))

# Call function with parameters assigned as above
product_info("White T-shirt", 15 dollars)

# Call function with keyword arguments
product_info(productname="jeans", price=45)
```

Output:

```
Productname: White T-shirt
Price: 15
Productname: Jeans
Price: 45
```

Lists

Lists are another cornerstone data type in Python used to specify an ordered sequence of elements. In short, they help you keep related data together and perform the same operations on several values at once. Unlike strings, lists are mutable (=changeable).

Each value inside a list is called an **item** and these are placed between square brackets.

Example lists

```
my_list = [1, 2, 3]
my_list2 = [“a”, “b”, “c”]
my_list3 = [“4”, d, “book”, 5]
```

Alternatively, you can use **list()** function to do the same:

```
alpha_list = list((“1”, “2”, “3”))
print(alpha_list)
```

How to Add Items to a List

You have two ways to add new items to existing lists.

The first one is using **append()** function:

```
beta_list = [“apple”, “banana”, “orange”]
beta_list.append(“grape”)
print(beta_list)
```

The second option is to **insert()** function to add an item at the specified index:

```
beta_list = [“apple”, “banana”, “orange”]
beta_list.insert(“2 grape”)
print(beta_list)
```

How to Remove an Item from a List

Again, you have several ways to do so. First, you can use **remove()** function:

```
beta_list = [“apple”, “banana”, “orange”]
beta_list.remove(“apple”)
print(beta_list)
```

Secondly, you can use the **pop() function**. If no index is specified, it will remove the last item.

```
beta_list = [“apple”, “banana”, “orange”]
beta_list.pop()
print(beta_list)
```

The last option is to use **del keyword** to remove a specific item:

```
beta_list = [“apple”, “banana”, “orange”]
del beta_list [1]
print(beta_list)
```

P.S. You can also apply del towards the entire list to scrap it.

Combine Two Lists

To mash up two lists use the + operator.

```
my_list = [1, 2, 3]
my_list2 = [“a”, “b”, “c”]
combo_list = my_list + my_list2
combo_list
[1, 2, 3, ‘a’, ‘b’, ‘c’]
```

Create a Nested List

You can also create a list of your lists when you have plenty of them :)

```
my_nested_list = [my_list, my_list2]
my_nested_list
[[1, 2, 3], [‘a’, ‘b’, ‘c’]]
```

Sort a List

Use the **sort()** function to organize all items in your list.

```
alpha_list = [34, 23, 67, 100, 88, 2]
alpha_list.sort()
alpha_list
[2, 23, 34, 67, 88, 100]
```

Slice a List

Now, if you want to call just a few elements from your list (e.g. the first 4 items), you need to specify a range of index numbers separated by a colon [x:y]. Here's an example:

```
alpha_list[0:4]
[2, 23, 34, 67]
```

Change Item Value on Your List

You can easily overwrite a value of one list items:

```
beta_list = [“apple”, “banana”, “orange”]
beta_list[1] = “pear”
print(beta_list)
```

Output:

```
[‘apple’, ‘pear’, ‘cherry’]
```

Loop Through the List

Using **for loop** you can multiply the usage of certain items, similarly to what * operator does. Here's an example:

```
for x in range(1,4):
    beta_list += [‘fruit’]
    print(beta_list)
```

Copy a List

Use the built-in **copy()** function to replicate your data:

```
beta_list = [“apple”, “banana”, “orange”]
beta_list = beta_list.copy()
print(beta_list)
```

Alternatively, you can copy a list with the **list()** method:

```
beta_list = [“apple”, “banana”, “orange”]
beta_list = list (beta_list)
print(beta_list)
```

List Comprehensions

List comprehensions are a handy option for creating lists based on existing lists. When using them you can build by using **strings** and **tuples** as well.

List comprehensions examples

```
list_variable = [x for x in iterable]
```

Here's a more complex example that features math operators, integers, and the **range()** function:

```
number_list = [x ** 2 for x in range(10) if x % 2 == 0]
print(number_list)
```

Tuples

Tuples are similar to lists — they allow you to display an ordered sequence of elements. However, they are immutable and you can't change the values stored in a tuple.

The advantage of using tuples over lists is that the former are slightly faster. So it's a nice way to optimize your code.

How to Create a Tuple

```
my_tuple = (1, 2, 3, 4, 5)
my_tuple[0:3]
(1, 2, 3)
```

Note: Once you create a tuple, you can't add new items to it or change it in any other way!

How to Slide a Tuple

The process is similar to slicing lists.

```
numbers = (0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12)
print(numbers[1:11:2])
```

Output:

```
(1, 3, 5, 7, 9)
```

Convert Tuple to a List

Since Tuples are immutable, you can't change them. What you can do though is convert a tuple into a list, make an edit and then convert it back to a tuple.

Here's how to accomplish this:

```
x = ("apple", "orange", "pear")
y = list(x)
y[1] = "grape"
x = tuple(y)
print(x)
```

Dictionaries

A dictionary holds indexes with keys that are mapped to certain values. These key-value pairs offer a great way of organizing and storing data in Python. They are mutable, meaning you can change the stored information.

A key value can be either a **string**, **Boolean**, or **integer**. Here's an example dictionary illustrating this:

```
Customer 1= {'username': 'john-sea', 'online': false,
'friends':100}
```

How to Create a Python Dictionary

Here's a quick example showcasing how to make an empty dictionary.

Option 1: **`new_dict = {}`**

Option 2: **`other_dict= dict()`**

And you can use the same two approaches to add values to your dictionary:

```
new_dict = {
  "brand": "Honda",
  "model": "Civic",
  "year": 1995
}
print(new_dict)
```

How to Access a Value in a Dictionary

You can access any of the values in your dictionary the following way:

```
x = new_dict[“brand”]
```

You can also use the following methods to accomplish the same.

- **dict.keys()** isolates keys
- **dict.values()** isolates values
- **dict.items()** returns items in a list format of (key, value) tuple pairs

Change Item Value

To change one of the items, you need to refer to it by its key name:

```
#Change the “year” to 2020:

new_dict= {
  “brand”: “Honda”,
  “model”: “Civic”,
  “year”: 1995
}
new_dict[“year”] = 2020
```

Loop Through the Dictionary

Again to implement looping, use for loop command.

Note: In this case, the return values are the keys of the dictionary. But, you can also return values using another method.

```
#print all key names in the dictionary

for x in new_dict:
  print(x)

#print all values in the dictionary

for x in new_dict:
  print(new_dict[x])

#loop through both keys and values

for x, y in my_dict.items():
  print(x, y)
```

If Statements (Conditional Statements) in Python

Just like other programming languages, Python supports the basic logical conditions from math:

- Equals: a == b
- Not Equals: a != b
- Less than: a < b
- Less than or equal to a <= b
- Greater than: a > b
- Greater than or equal to: a >= b

You can leverage these conditions in various ways. But most likely, you'll use them in "**if statements"** and **loops.**

If Statement Example

The goal of a conditional statement is to check if it's True or False.

```
if 5 > 1:
        print("That's True!")
```

Output:

```
That's True!
```

Nested If Statements

For more complex operations, you can create nested if statements. Here's how it looks:

```
x = 35

if x > 20:
  print("Above twenty,")
  if x > 30:
    print("and also above 30!")
```

Elif Statements

elif keyword prompts your program to try another condition if the previous one(s) was not true. Here's an example:

```
a = 45
b = 45
if b > a:
  print("b is greater than a")
elif a == b:
  print("a and b are equal")
```

If Else Statements

else keyword helps you add some additional filters to your condition clause. Here's how an if-elif-else combo looks:

```
if age < 4:
ticket_price = 0
elif age < 18:
ticket_price = 10
else: ticket_price = 15
```

If-Not-Statements

Not keyword let's you check for the opposite meaning to verify whether the value is NOT True:

```
new_list = [1, 2, 3, 4]
x = 10
if x not in new_list:
    print("'x' isn't on the list, so this is True!")
```

Pass Statements

If statements can't be empty. But if that's your case, add the **pass** statement to avoid having an error:

```
a = 33
b = 200

if b > a:
  pass
```

Python Loops

Python has two simple loop commands that are good to know:

- for loops
- while loops

Let's take a look at each of these.

For Loop

As already illustrated in the other sections of this Python checklist, **for loop** is a handy way for iterating over a sequence such as a list, tuple, dictionary, string, etc.

Here's an example showing how to loop through a string:

```
for x in "apple":
  print(x)
```

Plus, you've already seen other examples for lists and dictionaries.

While Loops

While loop enables you to execute a set of statements as long as the condition for them is true.

```
#print as long as x is less than 8

i = 1
while i< 8:
  print(x)
  i += 1
```

How to Break a Loop

You can also stop the loop from running even if the condition is met. For that, use the break statement both in while and for loops:

```
i = 1
while i < 8:
  print(i)
  if i == 4:
    break
  i += 1
```

Class

Since Python is an object-oriented programming language almost every element of it is an **object** — with its methods and properties.

Class acts as a blueprint for creating different objects. **Objects** are an instance of a class, where the class is manifested in some program.

How to Create a Class

Let's create a class named TestClass, with one property named z:

```
class TestClass:
  z = 5
```

How To Create an Object

As a next step, you can create an object using your class. Here's how it's done:

```
p1 = TestClass()
print(p1.x)
```

Further, you can assign different attributes and methods to your object. The example is below:

```
class car(object):
    """docstring"""

    def __init__(self, color, doors, tires):
        """Constructor"""
        self.color = color
        self.doors = doors
        self.tires = tires

    def brake(self):
        """
        Stop the car
        """
        return "Braking"

    def drive(self):
        """
        Drive the car
        """
        return "I'm driving!"
```

How to Create a Subclass

Every object can be further sub-classified. Here's an example

```
class Car(Vehicle):
    """
    The Car class
    """

    def brake(self):
        """
        Override brake method
        """
        return "The car class is breaking slowly!"

if __name__ == "__main__":
    car = Car("yellow", 2, 4, "car")
    car.brake()
    'The car class is breaking slowly!'
    car.drive()
    "I'm driving a yellow car!"
```

Dealing with Python Exceptions (Errors)

Python has a list of in-built exceptions (errors) that will pop up whenever you make a mistake in your code. As a newbie, it's good to know how to fix these.

The Most Common Python Exceptions

- `AttributeError` — pops up when an attribute reference or assignment fails.
- `IOError` — emerges when some I/O operation (e.g. an `open()` function) fails for an I/O-related reason, e.g., "file not found" or "disk full".
- `ImportError` — comes up when an import statement cannot locate the module definition. Also, when a from... import can't find a name that must be imported.
- `IndexError` — emerges when a sequence subscript is out of range.
- `KeyError` — raised when a dictionary key isn't found in the set of existing keys.
- `KeyboardInterrupt` — lights up when the user hits the interrupt key (such as Control-C or Delete).
- `NameError` — shows up when a local or global name can't be found.

- `OSError` — indicated a system-related error.
- `SyntaxError` — pops up when a parser encounters a syntax error.
- `TypeError` — comes up when an operation or function is applied to an object of inappropriate type.
- `ValueError` — raised when a built-in operation/function gets an argument that has the right type but not an appropriate value, and the situation is not described by a more precise exception such as IndexError.
- `ZeroDivisionError` — emerges when the second argument of a division or modulo operation is zero.

How to Troubleshoot the Errors

Python has a useful statement, design just for the purpose of handling exceptions – **try/except** statement. Here's a code snippet showing how you can catch KeyErrors in a dictionary using this statement:

```
my_dict = {“a”:1, “b”:2, “c”:3}
try:
        value = my_dict[“d”]
    except KeyError:
        print(“That key does not exist!”)
```

You can also detect several exceptions at once with a single statement. Here's an example for that:

```
my_dict = {“a”:1, “b”:2, “c”:3}
try:
    value = my_dict[“d”]
except IndexError:
    print(“This index does not exist!”)
except KeyError:
    print(“This key is not in the dictionary!”)
except:
    print(“Some other problem happened!”)
```

try/except with else clause

Adding an else clause will help you confirm that no errors were found:

```
my_dict = {“a”:1, “b”:2, “c”:3}

try:
    value = my_dict[“a”]
except KeyError:
    print(“A KeyError occurred!”)
else:
    print(“No error occurred!”)
```

Conclusions

Now you know the core Python concepts!

By no means is this Python checklist comprehensive. But it includes all the key data types, functions and commands you should learn as a beginner.

As always, we welcome your feedback in the comment section below!


WebsiteSetup