Python Interview Questions

Lavatech Technology

2022

Contents

1 Python Interview Questions for Freshers

1. **What is Python? What are the benefits of using Python?**

Answer:

Python is a high-level, interpreted, general-purpose programming language. Being a general-purpose language, it can be used to build almost any type of application with the right tools/libraries. Additionally, python supports objects, modules, threads, exception-handling, and automatic memory management which help in modelling real-world problems and building applications to solve these problems.
Benefits of using Python:

- Python is a general-purpose programming language that has a simple, easy-to-learn syntax that emphasizes readability and therefore reduces the cost of program maintenance. Moreover, the language is capable of scripting, is completely open-source, and supports third-party packages encouraging modularity and code reuse.
- Its high-level data structures, combined with dynamic typing and dynamic binding, attract a huge community of developers for Rapid Application Development and deployment.

2. **What is a dynamically typed language?**

Answer:

Before we understand a dynamically typed language, we should learn about what typing is. Typing refers to type-checking in programming languages. In a strongly-typed language, such as Python, "1" + 2 will result in a type error since these languages don't allow for "type-coercion" (implicit conversion of data types). On the other hand, a weakly-typed language, such as Javascript, will simply output "12" as result.
Type-checking can be done at two stages -

- Static - Data Types are checked before execution.
- Dynamic - Data Types are checked during execution.

Python is an interpreted language, executes each statement line by line and thus type-checking is done on the fly, during execution. Hence, Python is a Dynamically Typed Language.

3. **What is an Interpreted language?**

Answer:

An Interpreted language executes its statements line by line.Languages such as Python,Javascript,R,PHP,and Ruby are prime examples of Interpreted languages.Programs written in an interpreted language runs directly from the source code, with no intermediary compilation step.

4. **What is PEP 8 and why is it important?**

Answer:

PEP stands for **Python Enhancement Proposal.** A PEP is an official design document providing information to the Python community, or describing a new feature for Python or its processes. **PEP 8**is especially important since it documents the style guidelines for Python Code. Apparently contributing to the Python open-source community requires you to follow these style guidelines sincerely and strictly.

5. **What is Scope in Python?**

Answer:

Every object in Python functions within a scope. A scope is a block of code where an object in Python remains relevant.Namespaces uniquely identify all the objects inside a program. However, these namespaces also have a scope defined for them where you could use their objects without any prefix. A few examples of scope created during code execution in Python are as follows:

- **local scope** refers to the local objects available in the current function.
- **A global scope** refers to the objects available throughout the code execution since their inception.
- **A module-level** scope refers to the global objects of the current module accessible in the program.
- An **outermost scope** refers to all the built-in names callable in the program. The objects in this scope are searched last to find the name referenced.

Note: Local scope objects can be synced with global scope objects using keyword such as global.

6. **What are lists and tuples? What is the key difference between the two?**

Answer:

Lists and **Tuples** are **both sequence data types** that can store a collection of objects in Python. The objects stored in both sequences can have **different data types.** Lists are represented with square **brackets** ['sara', 6, 0.19] , while tuples are represented with **parantheses** ('ansh', 5, 0.97) .But what is the real difference between the two? The key difference between the two is that while **lists are mutable, tuples** on the other hand are **immutable** objects.This means that lists can be modified, appended or sliced on the go but tuples remain constant and cannot be modified in any manner. You can run the following example on Python IDLE to confirm the difference:

Code:

```
my_tuple = ('sara',6,5,0.97)
my_list = ['sara',6,5,0.97]
print(my_tuple[0]) # output => 'sara'
print(my_list[0]) # output => 'sara'
my_tuple[0] = 'ansh' # modifying tuple => throws an error
my list[0] = 'ansh' # modifying list => list modified
print(my_tuple[0]) # output => 'sara'
print(my_list[0]) # output => 'ansh'
```

7. **What are the common built-in data types in Python?**

Answer:

There are several built-in data types in Python. Although, Python doesn't require data types to be defined explicitly during variable declarations type errors are likely to occur if the knowledge of data types and their compatibility with each other are neglected. Python provides type() and isinstance() functions to check the type of these variables. These data types can be grouped into the following categories

(a) **None Type:**
None keyword represents the null values in Python. Boolean equality operation can be performed using these NoneType objects.

(b) **Numeric Types:**
There are three distinct numeric types - **integers, floating-point numbers,** and **complex numbers.** Additionally, **booleans**are a sub-type of integers. Note: The standard library also includes fractions to store rational numbers and decimal to store floating-point numbers with user-defined precision.

(c) **Sequence Types:**
According to Python Docs, there are three basic Sequence Types - **lists, tuples,**and **range** objects. Sequence types have the in and not in operators defined for their traversing their elements. These operators share the same priority as the comparison operations. Note: The standard library also includes additional types for processing: 1. Binary data such as bytearray bytes memoryview , and 2. Text strings such as str .

(d) **Mapping Types:**
A mapping object can map hashable values to random objects in Python. Mappings objects are mutable and there is currently only one standard mapping type, the **dictionary.**

(e) **Set Types:**
Currently, Python has two built-in set types - **set and frozenset.set** type is mutable and supports methods like add() and remove() . **frozenset** type is immutable and can't be modified after creation.
Note:set is mutable and thus cannot be used as key for a dictionary. On the other hand, frozenset is immutable and thus, hashable, and can be used as a dictionary key or as an element of another set.

(f) **Modules:**
Module is an additional built-in type supported by the Python Interpreter. Itnsupports one special operation, i.e., attribute access: mymod.myobj , where mymod is a module and myobj references a name defined in m's symbol table.The module's symbol table resides in a very special attribute of the module __dict__, but direct assignment to this module is neither possible nor recommended.

(g) **Callable Types:**
Callable types are the types to which function call can be applied. They can be **user-defined functions, instance methods, generator functions,** and some other **built-in functions, methods** and **classes.** Refer to the documentation at docs.python.org for a detailed view of the **callable types.**

8. **What is pass in Python?**

Answer:

The pass keyword represents a null operation in Python. It is generally used for the purpose of filling up empty blocks of code which may execute during runtime but has yet to be written. Without the **pass** statement in the following code, we may run into some errors during code execution.

Code:

```
def myEmptyFunc():
        # do nothing
        pass
myEmptyFunc() # nothing happens
## Without the pass keyword
# File"<stdin>",line 3
# Indentation Error:expected an indented block
```

9. **What are modules and packages in Python?**

Answer:

Python packages and Python modules are two mechanisms that allow for **modular programming**in Python. Modularizing has several advantages-

- **Simplicity:** :Working on a single module helps you focus on a relatively small portion of the problem at hand. This makes development easier and less error-prone.
- **Maintainability**: Modules are designed to enforce logical boundaries between different problem domains. If they are written in a manner that reduces interdependency, it is less likely that modifications in a module might impact other parts of the program.
- **Reusability**:Functions defined in a module can be easily reused by other parts of the application.
- **Scoping**:Modules typically define a separate namespace, which helps avoid confusion between identifiers from other parts of the program.

Modules in general, are simply Python files with a .py extension and can have a set of functions, classes, or variables defined and implemented. They can be imported and initialized once using the import statement. If partial functionality is needed, import the requisite classes or functions using from foo import bar .
Packages allow for hierarchial structuring of the module namespace using **dot notation.** As, **modules** help avoid clashes between global variable names, in a similar manner, **packages** help avoid clashes between module names. Creating a package is easy since it makes use of the system's inherent file structure. So just stuff the modules into a folder and there you have it, the folder name as the package name. Importing a module or its contents from this package requires the package name as prefix to the module name joined by a dot.
Note: You can technically import the package as well, but alas, it doesn't import the modules within the package to the local namespace, thus, it is practically useless.

10. **What are global, protected and private attributes in Python?**

Answer:

(a) **Global** variables are public variables that are defined in the global scope. To use the variable in the global scope inside a function, we use the global keyword.

(b) **Protected** attributes are attributes defined with an underscore prefixed to their identifier eg._sara. They can still be accessed and modified from outside the class they are defined in but a responsible developer should refrain from doing so.

(c) Privateattributes are attributes with double underscore prefixed to their identifier eg. __ansh. They cannot be accessed or modified from the outside directly and will result in an AttributeError if such an attempt is made.

11. **Whatis the use of self in Python?**

Answer:

Self is used to represent the instance of the class. With this keyword, you can access the attributes and methods of the class in python. It binds the attributes with the given arguments. self is used in different places and often thought to be a kewyord.But unlike in C++, self is not a keyword in Python.

12. **What is __init__?**

Answer:

__init__ is a contructor method in Python and is automatically called to allocate memory when a new object/instance is created. All classes have a __init__ method associated with them. It helps in distinguishing methods and attributes of a class from local variables.

Code:

```
# class definition
class Student:
    def __init__(self,fname,lname,age,section)
        self.firstname = fname
        self.lastname = lname
        self.age = age
        self.section = section
# creating a new object
stu1 = Student("Sara","Ansh",22,"A2")
```

13. **What is break, continue and pass in Python?**

Answer:

•

Code:

```
pat = [1,3,2,1,2,3,1,0,1,3]
for p in pat:
    pass
    if (p==0):
        current = p
        break
    elif (p
        continue
    print(p)          # output => 1 3 1 3 1
print(current)        # output =>0
```

14. **What are unit tests in Python?**

Answer:

- Unit test is a unit testing framework of Python.
- Unit testing means testing different components of software seperately.Can you think about why unit testing is important? Imagine a scenario, you are building software that uses three components namely A,B,and c.Now,suppose your software breaks at a point time. How will you find which component was responsible for breaking the software? Maybe it was component A that failed, which in turn failed component B, and this actually failed the softwares. There can be many such combinations.
- This is why it is necessary to test each and every component properly so that we know which component might be highly responsible for the failure of the software.

15. **What is docstring in Python?**

Answer:

- Documentation string or docstring is a multiline string used to document a specific code segment.
- The docstring should describe what the function or method does.

16. **What is slicing in Python?**

Answer:

- As the name suggests, 'slicing' is taking parts of.
- Syntax for slicing is **[start : stop : step]**
- **start** is the starting index from where to slice a list or tuple.
- **stop** is the ending index or where to sop.
- **step** is the number of steps to jump.
- Default value for **start**is 0, **stop** is number of items, **step** is 1.
- Slicing can be done on **strings, arrays, lists,** and textbftuples.

Code:

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

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

Answer:

Script file must begin with "**#!/usr/bin/env python**"

18. **What is the difference between Python Arrays and lists?**

Answer:

- Arrays in python can only contain elements of same data types i.e., data type of array should be homogeneous. It is a thin wrapper around C language arrays and consumes far less memory than lists.
- Lists in python can contain elements of different data types i.e., data type of lists can be heterogeneous. It has the disadvantage of consuming large memory.

Code:

```
import array
a = array.array('i',[1, 2, 3])
for i in a:
    print(i, end=' ') # OUTPUT: 1 2 3
a = array.array('i', [1, 2, 'string']) # OUTPUT: TypeError: an integer is required
a = [1, 2, 'string']
for i in a:
    print(i, end=' ') # OUTPUT: 1 2 string
```

2 Python Interview Questions for Experienced

1. **How is memory managed in Python?**

   Answer:

   - Memory management in Python is handled by the **Python Memory Manager.** The memory allocated by the manager is in form of a **private heap space** dedicated to Python. All Python objects are stored in this heap and being private, it is inaccessible to the programmer. Though, python does provide some core API functions to work upon the private heap space.
   - Additionally, Python has an in-built garbage collection to recycle the unused memory for the private heap space.

2. **What are Python namespaces? Why are they used?**

   Answer:

   A namespace in Python ensures that object names in a program are unique and can be used without any conflict. Python implements these namespaces as dictionaries with 'name as key' mapped to a corresponding 'object as value'. This allows for multiple namespaces to use the same name and map it to a separate object. A few examples of namespaces are as follows:

   - **Local Namespace** includes local names inside a function. the namespace is temporarily created for a function call and gets cleared when the function returns.
   - **Global Namespace**includes names from various imported packages/ modules that are being used in the current project. This namespace is created when the package is imported in the script and lasts until the execution of the script.
   - **Built-in Namespace** includes built-in functions of core Python and built-in names for various types of exceptions.

   The lifecycle of a namespace depends upon the scope of objects they are mapped to. If the scope of an object ends, the lifecycle of that namespace comes to an end. Hence, it isn't possible to access inner namespace objects from an outer namespace.

3. **What is Scope Resolution in Python?**

Answer:

Sometimes objects within the same scope have the same name but function differently. In such cases, scope resolution comes into play in Python automatically. A few examples of such behavior are:

- Python modules namely 'math' and 'cmath' have a lot of functions that are common to both of them - log10(),acos(),exp() etc. To resolve this ambiguity, it is necessary to prefix them with their respective module, like math.exp() and cmath.exp().
- Consider the code below, an object temp has been initialized to 10 globally and then to 20 on function call. However, the function call didn't change the value of the temp globally. Here, we can observe that Python draws a clear line between global and local variables, treating their namespaces as separate identities.

Code:

```
temp = 10 # global-scope variable
def func():
        temp = 20 # local-scope variable
        print(temp)
print(temp) # output => 10
func() # output => 20
print(temp) # output => 10
```

This behaviour can be overridden using the global keyword inside the function, as shown in the following example:

Code:

```
temp = 10 # global-scope variable
def func():
        global temp
        temp = 20 # local-scope variable
        print(temp)
print(temp) # output => 10
func() # output => 20
print(temp) # output => 20
```

4. **What are decorators in Python?**

Answer:

Decorators in Python are essentially functions that add functionality to an existing function in Python without changing the structure of the function itself. They are represented the @decorator_name in Python and are called in a bottom-up fashion. For example:

Code:

```
# decorator function to convert to lowercase
def lowercase_decorator(function):
    def wrapper():
        func = function()
        string_lowercase = func.lower()
        return string_lowercase
    return wrapper
# decorator function to split words
def splitter_decorator(function):
    def wrapper():
        func = function()
        string_split = func.split()
        return string_split
return wrapper
@splitter_decorator # this is executed next
@lowercase_decorator # this is executed first
def hello():
    return 'Hello World'
hello () # output => [ 'hello' , 'world' ]
```

The beauty of the decorators lies in the fact that besides adding functionality to the output of the method, they can even **accept arguments** for functions and can further modify those arguments before passing it to the function itself. The **inner nested** function, i.e. 'wrapper' , plays a significant role here. It is implemented to enforce **encapsulation** and thus, keep itself hidden from the global scope.

Code:

```
# decorator function to capitalize names
def names_decorator(function):
    def wrapper(arg1, arg2):
        arg1 = arg1.capitalize()
        arg2 = arg2.capitalize()
        string_hello = function(arg1, arg2)
        return string_hello
    return wrapper
@names_decorator
def say_hello (name1,name2):
    return 'Hello ' + name1 + '! Hello ' + name2 + '!'
say_hello('sara','ansh') # output => 'Hello Sara! Hello Ansh!
```

5. **What are Dict and List comprehensions?**

Answer:

Python comprehensions, like decorators, are **syntactic sugar** constructs that help **build altered** and **filtered lists,** dictionaries, or sets from a given list, dictionary, or set. Using comprehensions saves a lot of time and code that might be considerably more verbose (containing more lines of code). Let's check out some examples, where comprehensions can be truly beneficial:

- **Performing mathematical operations on the entire list**

  Code:

```
my_list = [2, 3, 5, 7, 11]
squared_list = [x**2 for x in my_list] # list comprehension
# output => [4 , 9 , 25 , 49 , 121]
squared_dict = x:x**2 for x in my_list # dict comprehension
# output => {11: 121, 2: 4 , 3: 9 , 5: 25 , 7: 49 }
```

- **Performing conditional filtering operations on the entire list**

  Code:

```
my_list = [2, 3, 5, 7, 11]
squared_list = [x**2 for x in my_ist if x%2 != 0] # list comprehension
# output => [9 , 25 , 49 , 121]
squared_dict = {x:x**2 for x in my_list if x%2 != 0} # dict comprehen-
sion
# output => {11: 121, 3: 9 , 5: 25 , 7: 49}
```

- **Combining multiple lists into one**
  Comprehensions allow for multiple iterators and hence, can be used to combine multiple lists into one.

  Code:

```
a = [1, 2, 3]
b = [7, 8, 9]
```

$$(x+y)for(x,y)inzip(a,b)$$

```
# parallel iterators
# output => [8, 10, 12]
```

$$(x,y)forxinaforyinb$$

```
# nested iterators
# output => [(1, 7), (1, 8), (1, 9), (2, 7), (2, 8), (2, 9), (3, 7), (3, 8),
(3, 9)]
```

- **Flattening a multi-dimensional list**
  A similar approach of nested iterators (as above) can be applied to flatten a multi-dimensional list or work upon its inner element

  Code:

```
my_list = [[10,20,30],[40,50,60],[70,80,90]]
flattened = [x for temp in my_list for x in temp]
# output => [10, 20, 30, 40, 50, 60, 70, 80, 90]
```

3 Python OOPS Interview Questions

1. **How do you create a class in Python?**

Answer:

To create a class in python, we use the keyword “class” as shown in the example below:

Code:

```
class InterviewbitEmployee:
    def __init__(self, emp_name):
        self.emp_name = emp_name
```

To instantiate or create an object from the class created above, we do the following:

Code:

```
emp_1=InterviewbitEmployee("Mr. Employee")
```

To access the name attribute, we just call the attribute using the dot operator as shown below:

Code:

```
print(emp_1.emp_name)
# Prints Mr. Employee
```

To create methods inside the class, we include the methods under the scope of the class as shown below:

Code:

```
class InterviewbitEmployee:
    def __init__(self, emp_name):
        self.emp_name = emp_name
    def introduce(self):
        print("Hello I am " + self.emp_name)
```

2. **How does inheritance work in python? Explain it with an example.**

Answer:

Inheritance gives the power to a class to access all attributes and methods of another class. It aids in code reusability and helps the developer to maintain applications without redundant code. The class inheriting from another class is a child class or also called a derived class. The class from which a child class derives the members are called parent class or superclass.
Python supports different kinds of inheritance, they are:

- **Single Inheritance**: Child class derives members of one parent class.

  Code:

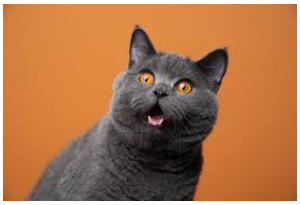
```
# Parent class
class ParentClass:
    def par_func(self):
        print("I am parent class function")

# Child class
class ChildClass(ParentClass):
    def child_func(self):
        print("I am child class function")

# Driver code
obj1 = ChildClass()
obj1.par_func()
obj1.child_func()
```

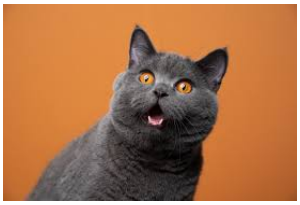
4 Python Pandas Interview Questions

1. Welcome



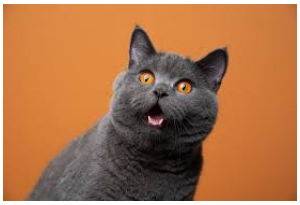
5 Numpy Interview Questions

1. Welcome



6 Python Libraries Interview Questions

1. Welcome



7 Python Programming Examples

1. Welcome