

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

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?

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.



Statically Typed



Dynamically Typed

3. What is an Interpreted language?

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?

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?

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:

- A **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 keywords such as **global**.

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

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 **parentheses** `('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:

```
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?

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-

- **None Type:**

`None` keyword represents the null values in Python. Boolean equality operation can be performed using these `NoneType` objects.

Class Name	Description
<code>NoneType</code>	Represents the NULL values in Python.

- **Numeric Types:**

There are three distinct numeric types - **integers**, **floating-point numbers**, and **complex numbers**. Additionally, **booleans** are a sub-type of integers.

Class Name	Description
<code>int</code>	Stores integer literals including hex, octal and binary numbers as integers
<code>float</code>	Stores literals containing decimal values and/or exponent signs as floating-point numbers
<code>complex</code>	Stores complex numbers in the form $(A + Bj)$ and has attributes: <code>real</code> and <code>imag</code>
<code>bool</code>	Stores boolean value (True or False).

*Note: The standard library also includes **fractions** to store rational numbers and **decimal** to store floating-point numbers with user-defined precision.*

- **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.

Class Name	Description
<code>list</code>	Mutable sequence used to store collection of items.
<code>tuple</code>	Immutable sequence used to store collection of items.
<code>range</code>	Represents an immutable sequence of numbers generated during execution.
<code>str</code>	Immutable sequence of Unicode code points to store textual data.

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`.

- **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*.

Class Name	Description
<code>dict</code>	Stores comma-separated list of key: value pairs

- **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.

Class Name	Description
<code>set</code>	Mutable unordered collection of distinct hashable objects.
<code>frozenset</code>	Immutable collection of distinct hashable objects.

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.

- **Modules:**

Module is an additional built-in type supported by the Python Interpreter. It supports 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.

- **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?

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.

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

9. What are modules and packages in Python?

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 hierachial 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?

- **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.
- **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.
- **Private** attributes 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. What is the use of self in Python?

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 keyword. But unlike in C++, `self` is not a keyword in Python.

12. What is `__init__`?

`__init__` is a constructor 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.

```
# 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?

Break	The break statement terminates the loop immediately and the control flows to the statement after the body of the loop.
Continue	The continue statement terminates the current iteration of the statement, skips the rest of the code in the current iteration and the control flows to the next iteration of the loop.
Pass	As explained above, the pass keyword in Python is generally used to fill up empty blocks and is similar to an empty statement represented by a semi-colon in languages such as Java, C++, Javascript, etc.

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

14. What are unit tests in Python?

- Unit test is a unit testing framework of Python.
- Unit testing means testing different components of software separately. 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 software. 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?

- 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?

- 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 stop.
- **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 tuples**.

```
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?

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

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

- 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.

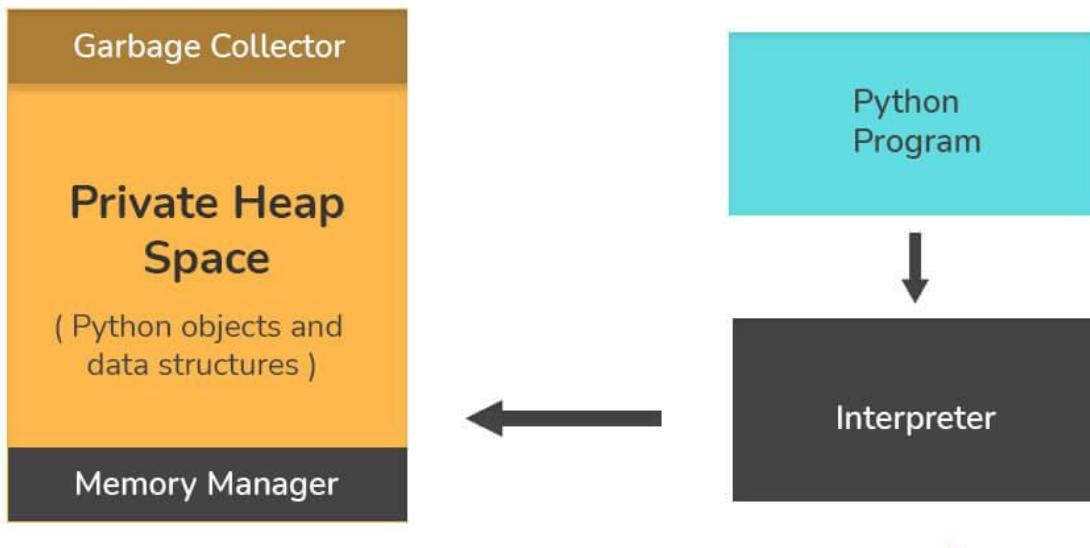
```
import array
a = array.array('i', [1, 2, 3])
for i in a:
    print(i, end=' ') #OUTPUT: 1 2 3
a = array.array('i', [1, 2, 'string']) #OUTPUT: TypeError: an integer is required (got type str)
a = [1, 2, 'string']
for i in a:
    print(i, end=' ') #OUTPUT: 1 2 string
```

Python Interview Questions for Experienced

19. How is memory managed in Python?

- 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.

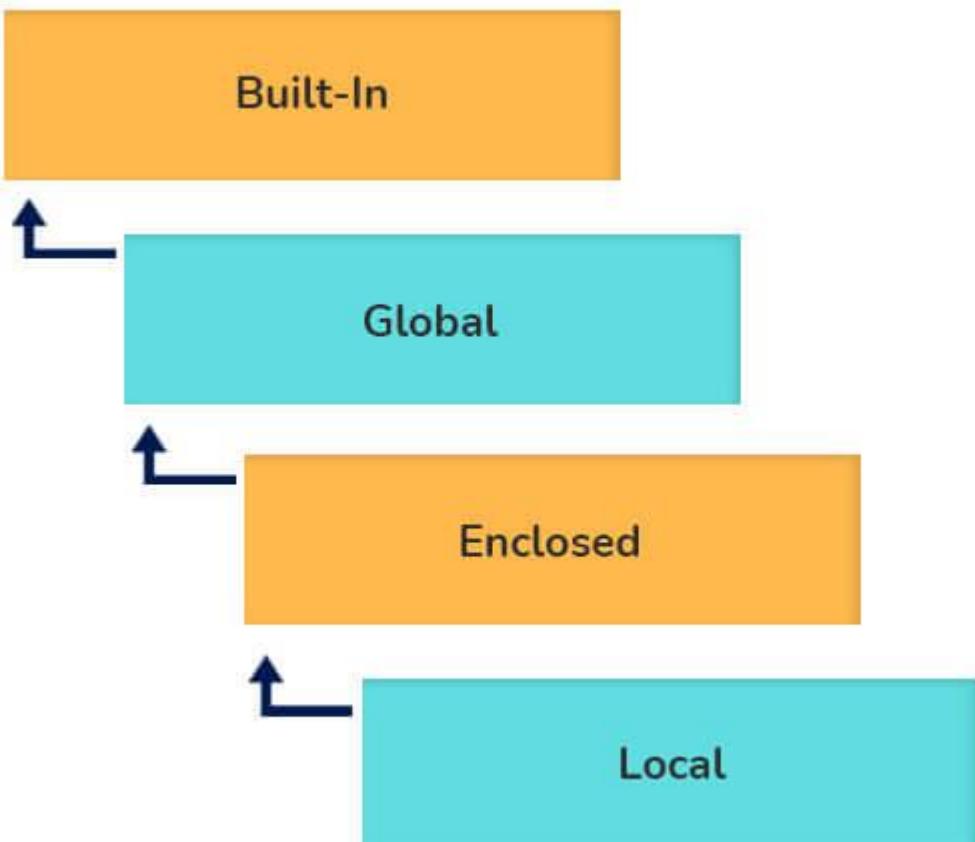


20. What are Python namespaces? Why are they used?

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.



21. What is Scope Resolution in Python?

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.

```
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 behavior can be overridden using the **global** keyword inside the function, as shown in the following example:

```
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
```

22. What are decorators in Python?

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:

```
# 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' function, plays a significant role here. It is implemented to enforce **encapsulation** and thus, keep itself hidden from the global scope.

```
# 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!'
```

23. What are Dict and List comprehensions?

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

```
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

```
my_list = [2, 3, 5, 7, 11]
squared_list = [x**2 for x in my_list 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 comprehension
# 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.

```
a = [1, 2, 3]
b = [7, 8, 9]
[(x + y) for (x,y) in zip(a,b)] # parallel iterators
# output => [8, 10, 12]
[(x,y) for x in a for y in b] # 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 elements.

```
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]
```

Note: List comprehensions have the same effect as the map method in other languages. They follow the mathematical set builder notation rather than map and filter functions in Python.

24. What is lambda in Python? Why is it used?

Lambda is an anonymous function in Python, that can accept any number of arguments, but can only have a single expression. It is generally used in situations requiring an anonymous function for a short time period. Lambda functions can be used in either of the two ways:

- Assigning lambda functions to a variable:

```
mul = lambda a, b : a * b
print(mul(2, 5)) # output => 10
```

- Wrapping lambda functions inside another function:

```
def myWrapper(n):
    return lambda a : a * n
mulFive = myWrapper(5)
print(mulFive(2)) # output => 10
```

25. How do you copy an object in Python?

In Python, the assignment statement (`=` operator) does not copy objects. Instead, it creates a binding between the existing object and the target variable name. To create copies of an object in Python, we need to use the `copy` module. Moreover, there are two ways of creating copies for the given object using the `copy` module -

Shallow Copy is a bit-wise copy of an object. The copied object created has an exact copy of the values in the original object. If either of the values is a reference to other objects, just the reference addresses for the same are copied. **Deep Copy** copies all values recursively from source to target object, i.e. it even duplicates the objects referenced by the source object.

```
from copy import copy, deepcopy
list_1 = [1, 2, [3, 5], 4]
## shallow copy
list_2 = copy(list_1)
list_2[3] = 7
list_2[2].append(6)
list_2 # output => [1, 2, [3, 5, 6], 7]
list_1 # output => [1, 2, [3, 5, 6], 4]
## deep copy
list_3 = deepcopy(list_1)
list_3[3] = 8
list_3[2].append(7)
list_3 # output => [1, 2, [3, 5, 6, 7], 8]
list_1 # output => [1, 2, [3, 5, 6], 4]
```

26. What is the difference between `xrange` and `range` in Python?

`xrange()` and `range()` are quite similar in terms of functionality. They both generate a sequence of integers, with the only difference that `range()` returns a **Python list**, whereas, `xrange()` returns an **xrange object**.

So how does that make a difference? It sure does, because unlike `range()`, `xrange()` doesn't generate a static list, it creates the value on the go. This technique is commonly used with an object-type **generator** and has been termed as "**yielding**".

Yielding is crucial in applications where memory is a constraint. Creating a static list as in `range()` can lead to a **Memory Error** in such conditions, while, `xrange()` can handle it optimally by using just enough memory for the generator (significantly less in comparison).

```
for i in xrange(10): # numbers from 0 to 9
```

```
print i      # output => 0 1 2 3 4 5 6 7 8 9
for i in xrange(1,10):  # numbers from 1 to 9
    print i      # output => 1 2 3 4 5 6 7 8 9
for i in xrange(1, 10, 2):  # skip by two for next
    print i      # output => 1 3 5 7 9
```

Note: `xrange` has been **deprecated** as of **Python 3.x**. Now `range` does exactly the same as what `xrange` used to do in **Python 2.x**, since it was way better to use `xrange()` than the original `range()` function in Python 2.x.

27. What is pickling and unpickling?

Python library offers a feature - **serialization** out of the box. Serializing an object refers to transforming it into a format that can be stored, so as to be able to deserialize it, later on, to obtain the original object. Here, the **pickle** module comes into play.

Pickling:

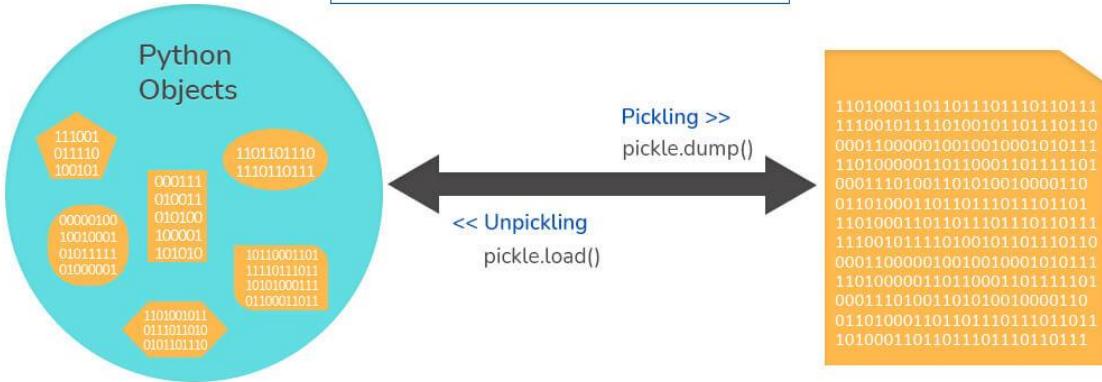
- Pickling is the name of the serialization process in Python. Any object in Python can be serialized into a byte stream and dumped as a file in the memory. The process of pickling is compact but pickle objects can be compressed further. Moreover, pickle keeps track of the objects it has serialized and the serialization is portable across versions.
- The function used for the above process is `pickle.dump()`.

Unpickling:

- Unpickling is the complete inverse of pickling. It deserializes the byte stream to recreate the objects stored in the file and loads the object to memory.
- The function used for the above process is `pickle.load()`.

Note: *Python has another, more primitive, serialization module called `marshall`, which exists primarily to support .pyc files in Python and differs significantly from the `pickle`.*

The Pickle Module



28. What are generators in Python?

Generators are functions that return an iterable collection of items, one at a time, in a set manner. Generators, in general, are used to create iterators with a different approach. They employ the use of `yield` keyword rather than `return` to return a **generator** object.

Let's try and build a generator for fibonacci numbers -

```
## generate fibonacci numbers upto n
def fib(n):
    p, q = 0, 1
    while(p < n):
        yield p
        p, q = q, p + q
x = fib(10) # create generator object

## iterating using __next__(), for Python2, use next()
x.__next__() # output => 0
x.__next__() # output => 1
x.__next__() # output => 1
x.__next__() # output => 2
x.__next__() # output => 3
x.__next__() # output => 5
x.__next__() # output => 8
x.__next__() # error

## iterating using loop
for i in fib(10):
    print(i) # output => 0 1 1 2 3 5 8
```

29. What is PYTHONPATH in Python?

PYTHONPATH is an environment variable which you can set to add additional directories where Python will look for modules and packages. This is especially useful in maintaining Python libraries that you do not wish to install in the global default location.

30. What is the use of help() and dir() functions?

help() function in Python is used to display the documentation of modules, classes, functions, keywords, etc. If no parameter is passed to the **help()** function, then an interactive **help utility** is launched on the console. **dir()** function tries to return a valid list of attributes and methods of the object it is called upon. It behaves differently with different objects, as it aims to produce the most relevant data, rather than the complete information.

- For Modules/Library objects, it returns a list of all attributes, contained in that module.
- For Class Objects, it returns a list of all valid attributes and base attributes.
- With no arguments passed, it returns a list of attributes in the current scope.

31. What is the difference between .py and .pyc files?

- .py files contain the source code of a program. Whereas, .pyc file contains the bytecode of your program. We get bytecode after compilation of .py file (source code). .pyc files are not created for all the files that you run. It is only created for the files that you import.
- Before executing a python program python interpreter checks for the compiled files. If the file is present, the virtual machine executes it. If not found, it checks for .py file. If found, compiles it to .pyc file and then python virtual machine executes it.
- Having .pyc file saves you the compilation time.

32. How Python is interpreted?

- Python as a language is not interpreted or compiled. Interpreted or compiled is the property of the implementation. Python is a bytecode(set of interpreter readable instructions) interpreted generally.
- Source code is a file with .py extension.
- Python compiles the source code to a set of instructions for a virtual machine. The Python interpreter is an implementation of that virtual machine. This intermediate format is called “bytecode”.

- .py source code is first compiled to give .pyc which is bytecode. This bytecode can be then interpreted by the official CPython or JIT(Just in Time compiler) compiled by PyPy.

33. How are arguments passed by value or by reference in python?

- **Pass by value:** Copy of the actual object is passed. Changing the value of the copy of the object will not change the value of the original object.
- **Pass by reference:** Reference to the actual object is passed. Changing the value of the new object will change the value of the original object.

In Python, arguments are passed by reference, i.e., reference to the actual object is passed.

```
def appendNumber(arr):
    arr.append(4)
arr = [1, 2, 3]
print(arr) #Output: => [1, 2, 3]
appendNumber(arr)
print(arr) #Output: => [1, 2, 3, 4]
```

34. What are iterators in Python?

- An iterator is an object.
- It remembers its state i.e., where it is during iteration (see code below to see how)
- `__iter__()` method initializes an iterator.
- It has a `__next__()` method which returns the next item in iteration and points to the next element. Upon reaching the end of iterable object `__next__()` must return `StopIteration` exception.
- It is also self-iterable.
- Iterators are objects with which we can iterate over iterable objects like lists, strings, etc.

```
class ArrayList:
    def __init__(self, number_list):
        self.numbers = number_list
    def __iter__(self):
        self.pos = 0
        return self
    def __next__(self):
        if(self.pos < len(self.numbers)):
            self.pos += 1
            return self.numbers[self.pos - 1]
```

```
else:  
    raise StopIteration  
array_obj = ArrayList([1, 2, 3])  
it = iter(array_obj)  
print(next(it)) #output: 2  
print(next(it)) #output: 3  
print(next(it))  
#Throws Exception  
#Traceback (most recent call last):  
#...  
#StopIteration
```

35. Explain how to delete a file in Python?

Use command **os.remove(file_name)**

```
import os  
os.remove("ChangedFile.csv")  
print("File Removed!")
```

36. Explain split() and join() functions in Python?

- You can use **split()** function to split a string based on a delimiter to a list of strings.
- You can use **join()** function to join a list of strings based on a delimiter to give a single string.

```
string = "This is a string."  
string_list = string.split(' ') #delimiter is ‘space’ character or ‘ ’  
print(string_list) #output: ['This', 'is', 'a', 'string.'][br/>print(' '.join(string_list)) #output: This is a string.
```

37. What does *args and **kwargs mean?

***args**

- *args is a special syntax used in the function definition to pass variable-length arguments.
- “*” means variable length and “args” is the name used by convention. You can use any other.

```
def multiply(a, b, *argv):  
    mul = a * b  
    for num in argv:
```

```
mul *= num
return mul
print(multiply(1, 2, 3, 4, 5)) #output: 120
```

**kwargs

- **kwargs is a special syntax used in the function definition to pass variable-length keyworded arguments.
- Here, also, “kwargs” is used just by convention. You can use any other name.
- Keyworded argument means a variable that has a name when passed to a function.
- It is actually a dictionary of the variable names and its value.

```
def tellArguments(**kwargs):
    for key, value in kwargs.items():
        print(key + ": " + value)
tellArguments(arg1 = "argument 1", arg2 = "argument 2", arg3 = "argument
3")
#output:
# arg1: argument 1
# arg2: argument 2
# arg3: argument 3
```

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

- Negative indexes are the indexes from the end of the list or tuple or string.
- **Arr[-1]** means the last element of array **Arr[]**

```
arr = [1, 2, 3, 4, 5, 6]
#get the last element
print(arr[-1]) #output 6
#get the second last element
print(arr[-2]) #output 5
```

Python OOPS Interview Questions

39. How do you create a class in Python?

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

```
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:

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

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

```
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:

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

    def introduce(self):
        print("Hello I am " + self.emp_name)
```

The self parameter in the init and introduce functions represent the reference to the current class instance which is used for accessing attributes and methods of that class. The self parameter has to be the first parameter of any method defined inside the class. The method of the class InterviewbitEmployee can be accessed as shown below:

```
emp_1.introduce()
```

The overall program would look like this:

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

    def introduce(self):
        print("Hello I am " + self.emp_name)

# create an object of InterviewbitEmployee class
emp_1 = InterviewbitEmployee("Mr Employee")
print(emp_1.emp_name) #print employee name
```

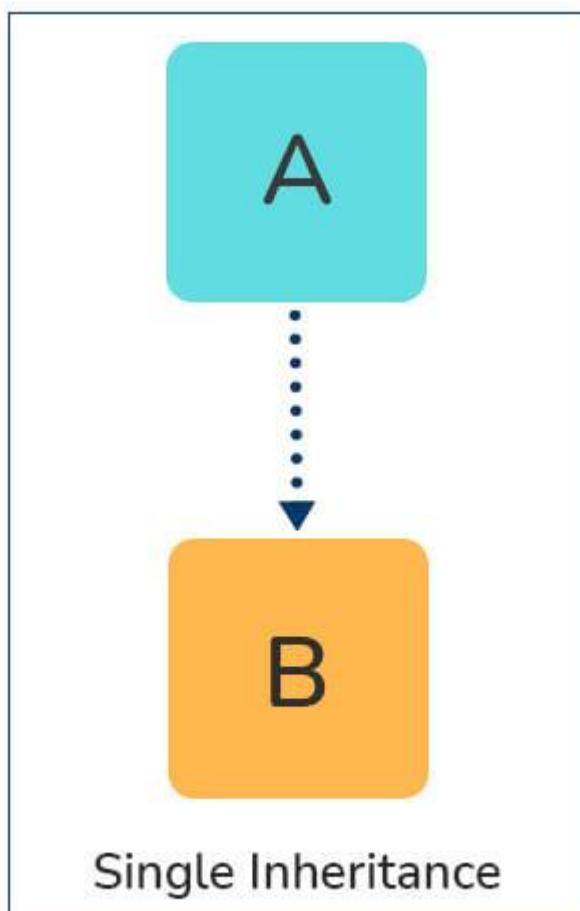
```
emp_1.introduce()      #introduce the employee
```

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

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.



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

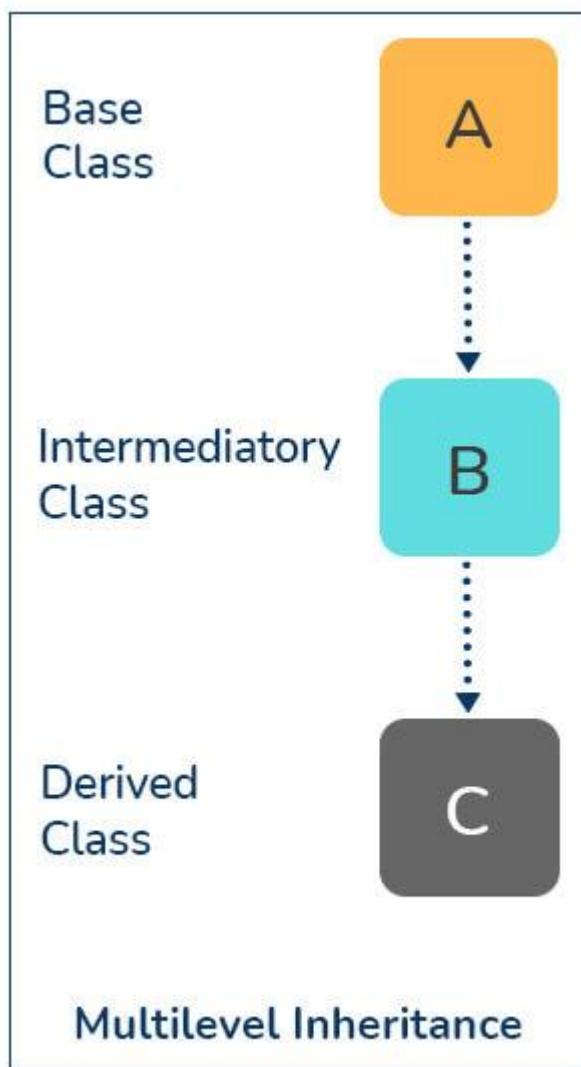
```

# 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()

```

- **Multi-level Inheritance:** The members of the parent class, A, are inherited by child class which is then inherited by another child class, B. The features of the base class and the derived class are further inherited into the new derived class, C. Here, A is the grandfather class of class C.



```

# Parent class
class A:
    def __init__(self, a_name):
        self.a_name = a_name

# Intermediate class
class B(A):
    def __init__(self, b_name, a_name):
        self.b_name = b_name
        # invoke constructor of class A
        A.__init__(self, a_name)

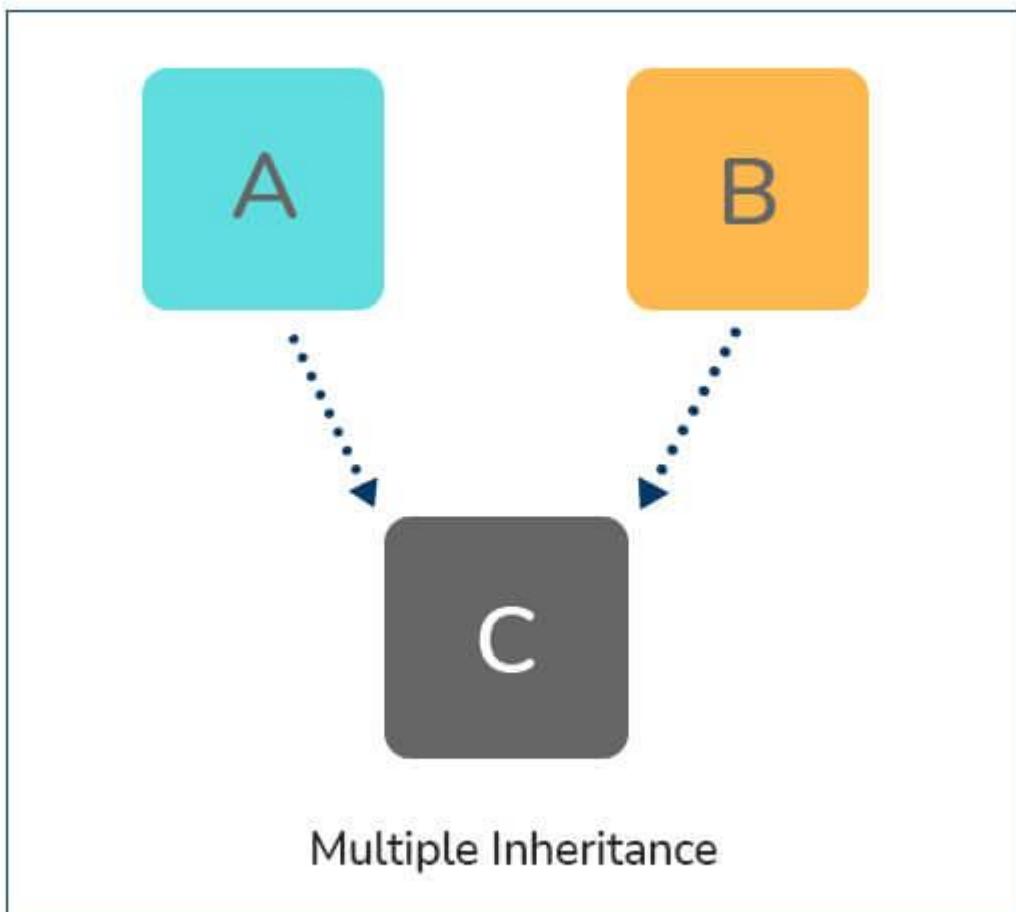
# Child class
class C(B):
    def __init__(self,c_name, b_name, a_name):
        self.c_name = c_name
        # invoke constructor of class B
        B.__init__(self, b_name, a_name)

    def display_names(self):
        print("A name : ", self.a_name)
        print("B name : ", self.b_name)
        print("C name : ", self.c_name)

# Driver code
obj1 = C('child', 'intermediate', 'parent')
print(obj1.a_name)
obj1.display_names()

```

- **Multiple Inheritance:** This is achieved when one child class derives members from more than one parent class. All features of parent classes are inherited in the child class.



```
# Parent class1
class Parent1:
    def parent1_func(self):
        print("Hi I am first Parent")

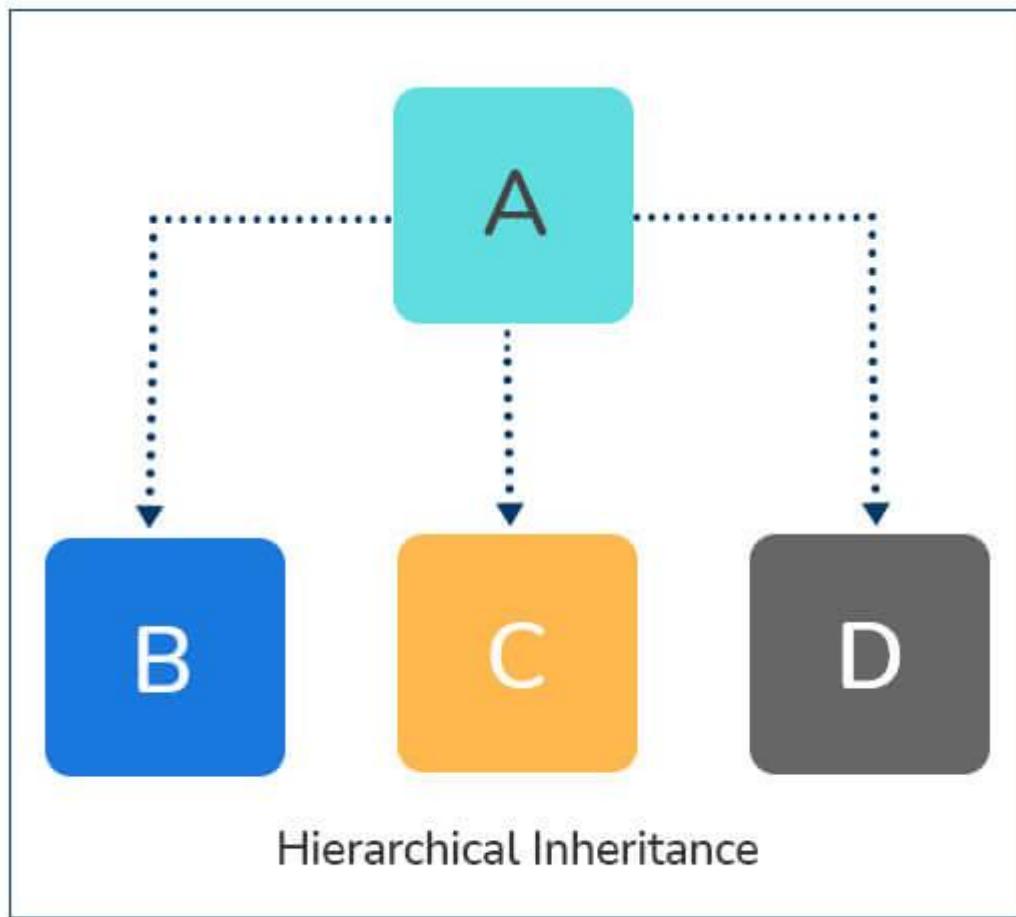
# Parent class2
class Parent2:
    def parent2_func(self):
        print("Hi I am second Parent")

# Child class
class Child(Parent1, Parent2):
    def child_func(self):
        self.parent1_func()
        self.parent2_func()

# Driver's code
obj1 = Child()
```

```
obj1.child_func()
```

- **Hierarchical Inheritance:** When a parent class is derived by more than one child class, it is called hierarchical inheritance.



```
# Base class
class A:
    def a_func(self):
        print("I am from the parent class.")

# 1st Derived class
class B(A):
    def b_func(self):
        print("I am from the first child.")

# 2nd Derived class
class C(A):
    def c_func(self):
```

```

print("I am from the second child.")

# Driver's code
obj1 = B()
obj2 = C()
obj1.a_func()
obj1.b_func() #child 1 method
obj2.a_func()
obj2.c_func() #child 2 method

```

41. How do you access parent members in the child class?

Following are the ways using which you can access parent class members within a child class:

- **By using Parent class name:** You can use the name of the parent class to access the attributes as shown in the example below:

```

class Parent(object):
    # Constructor
    def __init__(self, name):
        self.name = name

class Child(Parent):
    # Constructor
    def __init__(self, name, age):
        Parent.name = name
        self.age = age

    def display(self):
        print(Parent.name, self.age)

# Driver Code
obj = Child("Interviewbit", 6)
obj.display()

```

- **By using super():** The parent class members can be accessed in child class using the super keyword.

```

class Parent(object):
    # Constructor
    def __init__(self, name):
        self.name = name

```

```

class Child(Parent):
    # Constructor
    def __init__(self, name, age):
        """
        In Python 3.x, we can also use super().__init__(name)
        """

        super(Child, self).__init__(name)
        self.age = age

    def display(self):
        # Note that Parent.name cant be used
        # here since super() is used in the constructor
        print(self.name, self.age)

# Driver Code
obj = Child("Interviewbit", 6)
obj.display()

```

42. Are access specifiers used in python?

Python does not make use of access specifiers specifically like private, public, protected, etc. However, it does not derive this from any variables. It has the concept of imitating the behaviour of variables by making use of a single (protected) or double underscore (private) as prefixed to the variable names. By default, the variables without prefixed underscores are public.

Example:

```

# to demonstrate access specifiers
class InterviewbitEmployee:

    # protected members
    _emp_name = None
    _age = None

    # private members
    __branch = None

    # constructor
    def __init__(self, emp_name, age, branch):
        self._emp_name = emp_name
        self._age = age
        self.__branch = branch

```

```
#public member
def display():
    print(self._emp_name +" "+self._age+" "+self.__branch)
```

43. Is it possible to call parent class without its instance creation?

Yes, it is possible if the base class is instantiated by other child classes or if the base class is a static method.

44. How is an empty class created in python?

An empty class does not have any members defined in it. It is created by using the pass keyword (the pass command does nothing in python). We can create objects for this class outside the class.

For example-

```
class EmptyClassDemo:
    pass
obj=EmptyClassDemo()
obj.name="Interviewbit"
print("Name created= ",obj.name)
```

Output:

Name created = Interviewbit

45. Differentiate between new and override modifiers.

The new modifier is used to instruct the compiler to use the new implementation and not the base class function. The Override modifier is useful for overriding a base class function inside the child class.

46. Why is finalize used?

Finalize method is used for freeing up the unmanaged resources and clean up before the garbage collection method is invoked. This helps in performing memory management tasks.

47. What is init method in python?

The **init** method works similarly to the constructors in Java. The method is run as soon as an object is instantiated. It is useful for initializing any attributes or default behaviour of the object at the time of instantiation.

For example:

```

class InterviewbitEmployee:

    # init method / constructor
    def __init__(self, emp_name):
        self.emp_name = emp_name

    # introduce method
    def introduce(self):
        print('Hello, I am ', self.emp_name)

emp = InterviewbitEmployee('Mr Employee')  # __init__ method is called
here and initializes the object name with "Mr Employee"
emp.introduce()

```

48. How will you check if a class is a child of another class?

This is done by using a method called **issubclass()** provided by python. The method tells us if any class is a child of another class by returning true or false accordingly.

For example:

```

class Parent(object):
    pass

class Child(Parent):
    pass

# Driver Code
print(issubclass(Child, Parent))  #True
print(issubclass(Parent, Child))  #False

```

- We can check if an object is an instance of a class by making use of **isinstance()** method:

```

obj1 = Child()
obj2 = Parent()
print(isinstance(obj2, Child))  #False
print(isinstance(obj2, Parent))  #True

```

Python Pandas Interview Questions

49. What do you know about pandas?

- Pandas is an open-source, python-based library used in data manipulation applications requiring high performance. The name is derived from “Panel Data” having multidimensional data. This was developed in 2008 by Wes McKinney and was developed for data analysis.
- Pandas are useful in performing 5 major steps of data analysis - Load the data, clean/manipulate it, prepare it, model it, and analyze the data.

50. Define pandas dataframe.

A dataframe is a 2D mutable and tabular structure for representing data labelled with axes - rows and columns.

The syntax for creating dataframe:

```
import pandas as pd
dataframe = pd.DataFrame( data, index, columns, dtype)
```

where:

- data - Represents various forms like series, map, ndarray, lists, dict etc.
- index - Optional argument that represents an index to row labels.
- columns - Optional argument for column labels.
- Dtype - the data type of each column. Again optional.

51. How will you combine different pandas dataframes?

The dataframes can be combined using the below approaches:

- **append() method:** This is used to stack the dataframes horizontally.
Syntax:

```
df1.append(df2)
```

- **concat() method:** This is used to stack dataframes vertically. This is best used when the dataframes have the same columns and similar fields. Syntax:

```
pd.concat([df1, df2])
```

- **join() method:** This is used for extracting data from various dataframes having one or more common columns.

```
df1.join(df2)
```

52. Can you create a series from the dictionary object in pandas?

One dimensional array capable of storing different data types is called a series. We can create pandas series from a dictionary object as shown below:

```
import pandas as pd
dict_info = { 'key1' : 2.0, 'key2' : 3.1, 'key3' : 2.2}
series_obj = pd.Series(dict_info)
print (series_obj)
```

Output:

```
x    2.0
y    3.1
z    2.2
dtype: float64
```

If an index is not specified in the input method, then the keys of the dictionaries are sorted in ascending order for constructing the index. In case the index is passed, then values of the index label will be extracted from the dictionary.

53. How will you identify and deal with missing values in a dataframe?

We can identify if a dataframe has missing values by using the isnull() and isna() methods.

```
missing_data_count=df.isnull().sum()
```

We can handle missing values by either replacing the values in the column with 0 as follows:

```
df['column_name'].fillna(0)
```

Or by replacing it with the mean value of the column

```
df['column_name'] = df['column_name'].fillna((df['column_name'].mean()))
```

54. What do you understand by reindexing in pandas?

Reindexing is the process of conforming a dataframe to a new index with optional filling logic. If the values are missing in the previous index, then NaN/NA is placed in the location. A new object is returned unless a new index is produced that is equivalent to the current one. The copy value is set to False. This is also used for changing the index of rows and columns in the dataframe.

55. How to add new column to pandas dataframe?

A new column can be added to a pandas dataframe as follows:

```

import pandas as pd
data_info = {'first' : pd.Series([1, 2, 3], index=['a', 'b', 'c']),
             'second' : pd.Series([1, 2, 3, 4], index=['a', 'b', 'c', 'd'])}

df = pd.DataFrame(data_info)
#To add new column third
df['third']=pd.Series([10,20,30],index=['a','b','c'])
print (df)
#To add new column fourth
df['fourth']=df['first']+info['third']
print (df)

```

56. How will you delete indices, rows and columns from a dataframe?

To delete an Index:

- Execute `del df.index.name` for removing the index by name.
- Alternatively, the `df.index.name` can be assigned to None.
- For example, if you have the below dataframe:

Column 1	
Names	
John	1
Jack	2
Judy	3
Jim	4

- To drop the index name “Names”:

```

df.index.name = None
# Or run the below:
# del df.index.name
print(df)

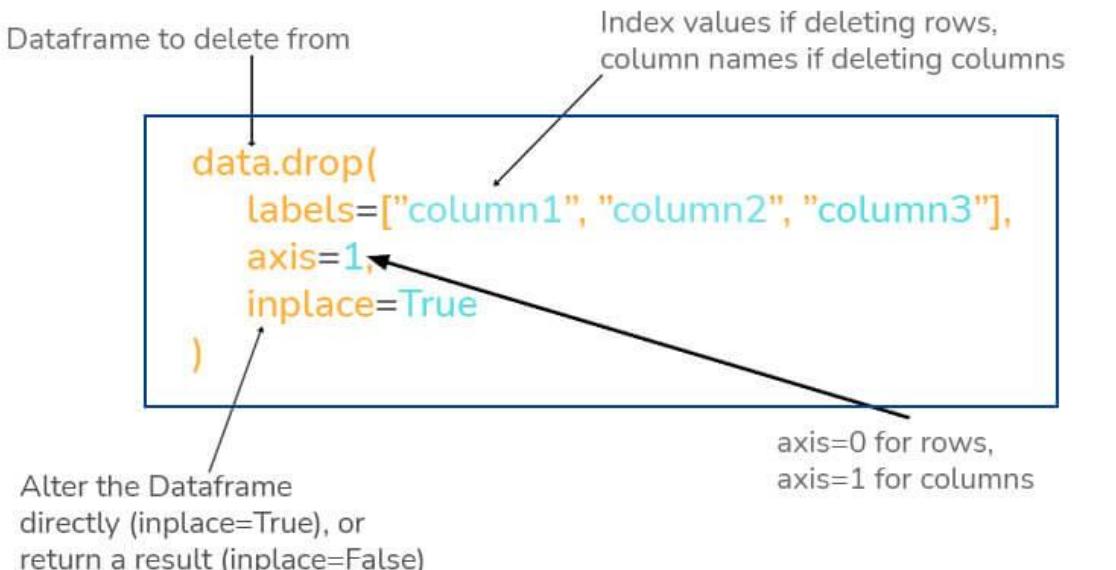
```

Column 1	
John	1
Jack	2
Judy	3
Jim	4

To delete row/column from dataframe:

- `drop()` method is used to delete row/column from dataframe.
- The axis argument is passed to the drop method where if the value is 0, it indicates to drop/delete a row and if 1 it has to drop the column.

- Additionally, we can try to delete the rows/columns in place by setting the value of inplace to True. This makes sure that the job is done without the need for reassignment.
- The duplicate values from the row/column can be deleted by using the `drop_duplicates()` method.



 InterviewBit

57. Can you get items of series A that are not available in another series B?

This can be achieved by using the `~` (not/negation symbol) and `isin()` method as shown below.

```
import pandas as pd
df1 = pd.Series([2, 4, 8, 10, 12])
df2 = pd.Series([8, 12, 10, 15, 16])
df1=df1[~df1.isin(df2)]
print(df1)
"""

```

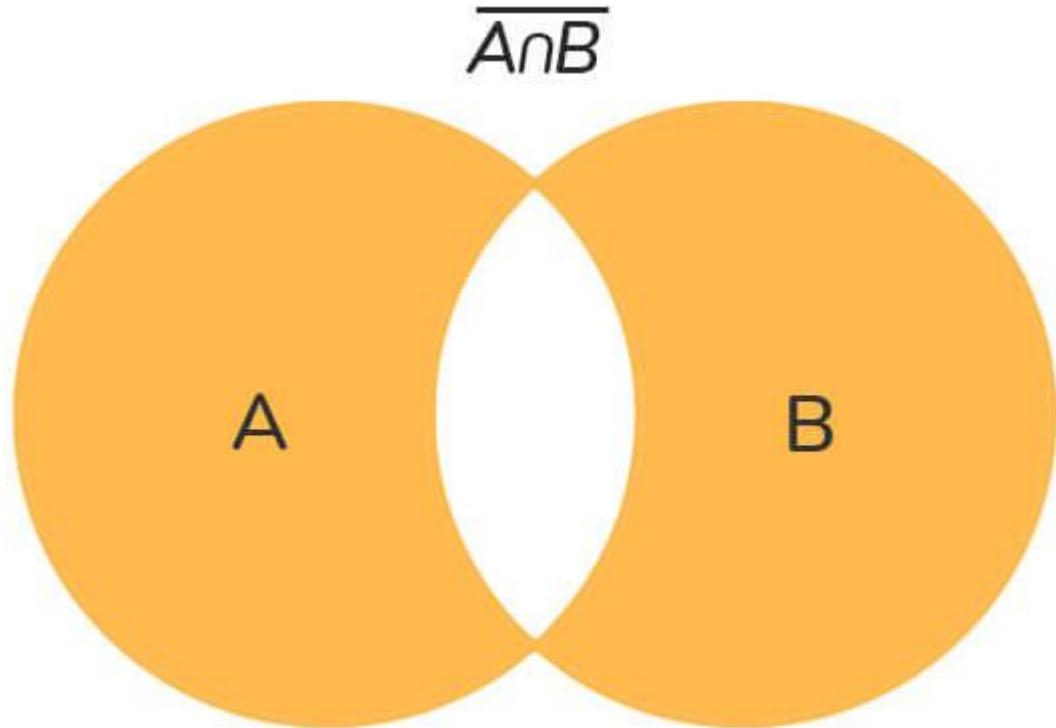
Output:

```
0  2
1  4
dtype: int64
"""


```

58. How will you get the items that are not common to both the given series A and B?

We can achieve this by first performing the union of both series, then taking the intersection of both series. Then we follow the approach of getting items of union that are not there in the list of the intersection.



The following code demonstrates this:

```
import pandas as pd
import numpy as np
df1 = pd.Series([2, 4, 5, 8, 10])
df2 = pd.Series([8, 10, 13, 15, 17])
p_union = pd.Series(np.union1d(df1, df2)) # union of series
p_intersect = pd.Series(np.intersect1d(df1, df2)) # intersection of series
unique_elements = p_union[~p_union.isin(p_intersect)]
print(unique_elements)
"""
```

Output:

```
0    2
1    4
2    5
5   13
```

```
6 15  
7 17  
dtype: int64  
"""
```

59. While importing data from different sources, can the pandas library recognize dates?

Yes, they can, but with some bit of help. We need to add the `parse_dates` argument while we are reading data from the sources. Consider an example where we read data from a CSV file, we may encounter different date-time formats that are not readable by the pandas library. In this case, pandas provide flexibility to build our custom date parser with the help of lambda functions as shown below:

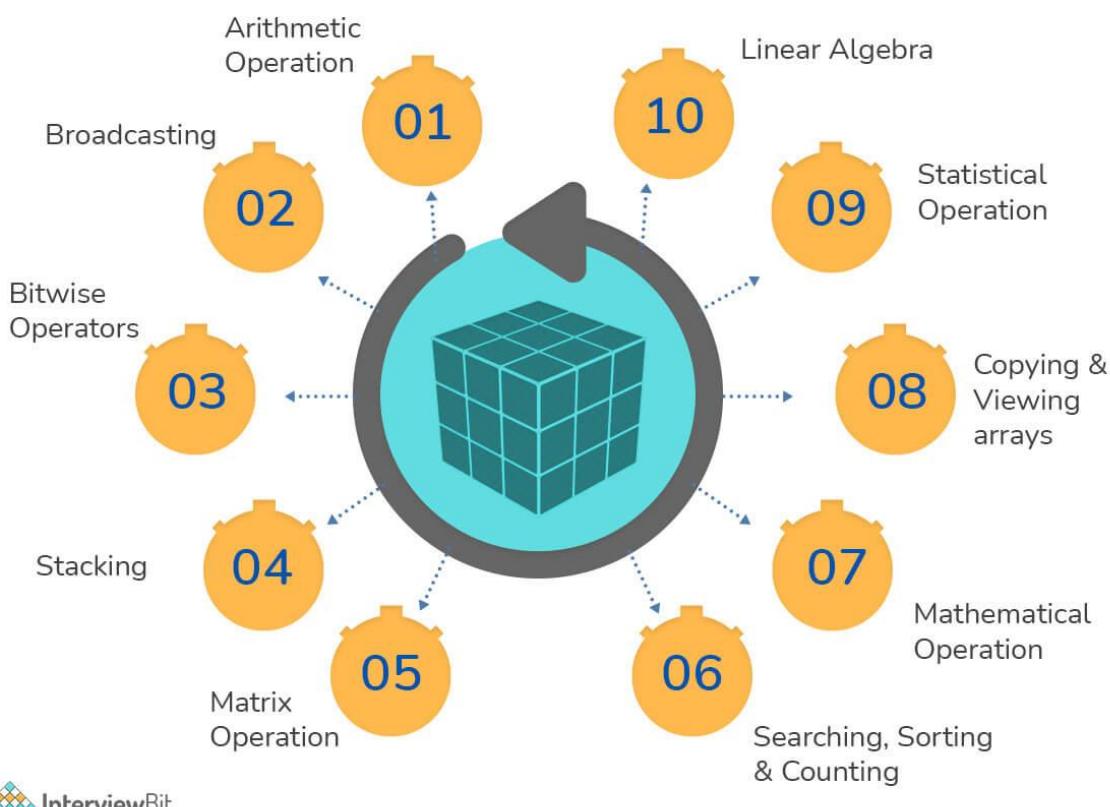
```
import pandas as pd  
from datetime import datetime  
dateparser = lambda date_val: datetime.strptime(date_val, '%Y-%m-%d  
%H:%M:%S')  
df = pd.read_csv("some_file.csv", parse_dates=['datetime_column'],  
date_parser=dateparser)
```

Numpy Interview Questions

60. What do you understand by NumPy?

NumPy is one of the most popular, easy-to-use, versatile, open-source, python-based, general-purpose package that is used for processing arrays. NumPy is short for NUMerical PYthon. This is very famous for its highly optimized tools that result in high performance and powerful N-Dimensional array processing feature that is designed explicitly to work on complex arrays. Due to its popularity and powerful performance and its flexibility to perform various operations like trigonometric operations, algebraic and statistical computations, it is most commonly used in performing scientific computations and various broadcasting functions. The following image shows the applications of NumPy:

Uses Of NumPy



 InterviewBit

61. How are NumPy arrays advantageous over python lists?

- The list data structure of python is very highly efficient and is capable of performing various functions. But, they have severe limitations when it comes to the computation of vectorized operations which deals with element-wise multiplication and addition. The python lists also require the information regarding the type of every element which results in overhead as type dispatching code gets executes every time any operation is performed on any element. This is where the NumPy arrays come into the picture as all the limitations of python lists are handled in NumPy arrays.
- Additionally, as the size of the NumPy arrays increases, NumPy becomes around 30x times faster than the Python List. This is because the Numpy arrays are densely packed in the memory due to their homogenous nature. This ensures the memory free up is also faster.

62. What are the steps to create 1D, 2D and 3D arrays?

- **1D array creation:**

```
import numpy as np
one_dimensional_list = [1,2,4]
one_dimensional_arr = np.array(one_dimensional_list)
print("1D array is : ",one_dimensional_arr)
```

- **2D array creation:**

```
import numpy as np
two_dimensional_list=[[1,2,3],[4,5,6]]
two_dimensional_arr = np.array(two_dimensional_list)
print("2D array is : ",two_dimensional_arr)
```

- **3D array creation:**

```
import numpy as np
three_dimensional_list=[[[1,2,3],[4,5,6],[7,8,9]]]
three_dimensional_arr = np.array(three_dimensional_list)
print("3D array is : ",three_dimensional_arr)
```

- **ND array creation:** This can be achieved by giving the ndmin attribute.
The below example demonstrates the creation of a 6D array:

```
import numpy as np
ndArray = np.array([1, 2, 3, 4], ndmin=6)
print(ndArray)
print('Dimensions of array:', ndArray.ndim)
```

63. You are given a numpy array and a new column as inputs. How will you delete the second column and replace the column with a new column value?

Example:

Given array:

```
[[35 53 63]
[72 12 22]
[43 84 56]]
```

New Column values:

```
[  
 20  
 30  
 40]
```

```
]
```

Solution:

```
import numpy as np
#inputs
inputArray = np.array([[35,53,63],[72,12,22],[43,84,56]])
new_col = np.array([[20,30,40]])
# delete 2nd column
arr = np.delete(inputArray , 1, axis = 1)
#insert new_col to array
arr = np.insert(arr , 1, new_col, axis = 1)
print (arr)
```

64. How will you efficiently load data from a text file?

We can use the method `numpy.loadtxt()` which can automatically read the file's header and footer lines and the comments if any.

This method is highly efficient and even if this method feels less efficient, then the data should be represented in a more efficient format such as CSV etc. Various alternatives can be considered depending on the version of NumPy used.

Following are the file formats that are supported:

- Text files: These files are generally very slow, huge but portable and are human-readable.
- Raw binary: This file does not have any metadata and is not portable. But they are fast.
- Pickle: These are borderline slow and portable but depends on the NumPy versions.
- HDF5: This is known as the High-Powered Kitchen Sink format which supports both PyTables and h5py format.
- .npy: This is NumPy's native binary data format which is extremely simple, efficient and portable.

65. How will you read CSV data into an array in NumPy?

This can be achieved by using the `genfromtxt()` method by setting the delimiter as a comma.

```
from numpy import genfromtxt
csv_data = genfromtxt('sample_file.csv', delimiter=',')
```

66. How will you sort the array based on the Nth column?

For example, consider an array arr.

```
arr = np.array([[8, 3, 2],  
               [3, 6, 5],  
               [6, 1, 4]])
```

Let us try to sort the rows by the 2nd column so that we get:

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

We can do this by using the sort() method in numpy as:

```
import numpy as np  
arr = np.array([[8, 3, 2],  
               [3, 6, 5],  
               [6, 1, 4]])  
#sort the array using np.sort  
arr = np.sort(arr.view('i8,i8,i8'),  
              order=['f1'],  
              axis=0).view(np.int)
```

We can also perform sorting and that too inplace sorting by doing:

```
arr.view('i8,i8,i8').sort(order=['f1'], axis=0)
```

67. How will you find the nearest value in a given numpy array?

We can use the argmin() method of numpy as shown below:

```
import numpy as np  
def find_nearest_value(arr, value):  
    arr = np.asarray(arr)  
    idx = (np.abs(arr - value)).argmin()  
    return arr[idx]  
#Driver code  
arr = np.array([ 0.21169,  0.61391, 0.6341, 0.0131, 0.16541,  0.5645,  0.5742])  
value = 0.52  
print(find_nearest_value(arr, value)) # Prints 0.5645
```

68. How will you reverse the numpy array using one line of code?

This can be done as shown in the following:

```
reversed_array = arr[::-1]
```

where **arr** = original given array, **reverse_array** is the resultant after reversing all elements in the input.

69. How will you find the shape of any given NumPy array?

We can use the **shape** attribute of the numpy array to find the shape. It returns the shape of the array in terms of row count and column count of the array.

```
import numpy as np
arr_two_dim = np.array([("x1", "x2", "x3", "x4"),
                       ("x5", "x6", "x7", "x8")])
arr_one_dim = np.array([3, 2, 4, 5, 6])
# find and print shape
print("2-D Array Shape: ", arr_two_dim.shape)
print("1-D Array Shape: ", arr_one_dim.shape)
"""
```

Output:

```
2-D Array Shape: (2, 4)
1-D Array Shape: (5,)
```

Python Libraries Interview Questions

70. Differentiate between a package and a module in python.

The module is a single python file. A module can import other modules (other python files) as objects. Whereas, a package is the folder/directory where different sub-packages and the modules reside.

A python module is created by saving a file with the extension of **.py**. This file will have classes and functions that are reusable in the code as well as across modules.

A python package is created by following the below steps:

- Create a directory and give a valid name that represents its operation.
- Place modules of one kind in this directory.
- Create **`__init__.py`** file in this directory. This lets python know the directory we created is a package. The contents of this package can be

imported across different modules in other packages to reuse the functionality.

71. What are some of the most commonly used built-in modules in Python?

Python modules are the files having python code which can be functions, variables or classes. These go by .py extension. The most commonly available built-in modules are:

- os
- math
- sys
- random
- re
- datetime
- JSON

72. What are lambda functions?

Lambda functions are generally inline, anonymous functions represented by a single expression. They are used for creating function objects during runtime. They can accept any number of parameters. They are usually used where functions are required only for a short period. They can be used as:

```
mul_func = lambda x,y : x*y  
print(mul_func(6, 4))  
# Output: 24
```

73. How can you generate random numbers?

Python provides a module called random using which we can generate random numbers.

- We have to import a random module and call the **random()** method as shown below:
 - The random() method generates float values lying between 0 and 1 randomly.

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

- To generate customised random numbers between specified ranges, we can use the **randrange()** method

Syntax: `randrange(beginning, end, step)`

For example:

```
import random  
print(random.randrange(5,100,2))
```

74. Can you easily check if all characters in the given string is alphanumeric?

This can be easily done by making use of the `isalnum()` method that returns true in case the string has only alphanumeric characters.

For Example -

```
"abdc1321".isalnum() #Output: True  
"xyz@123$".isalnum() #Output: False
```

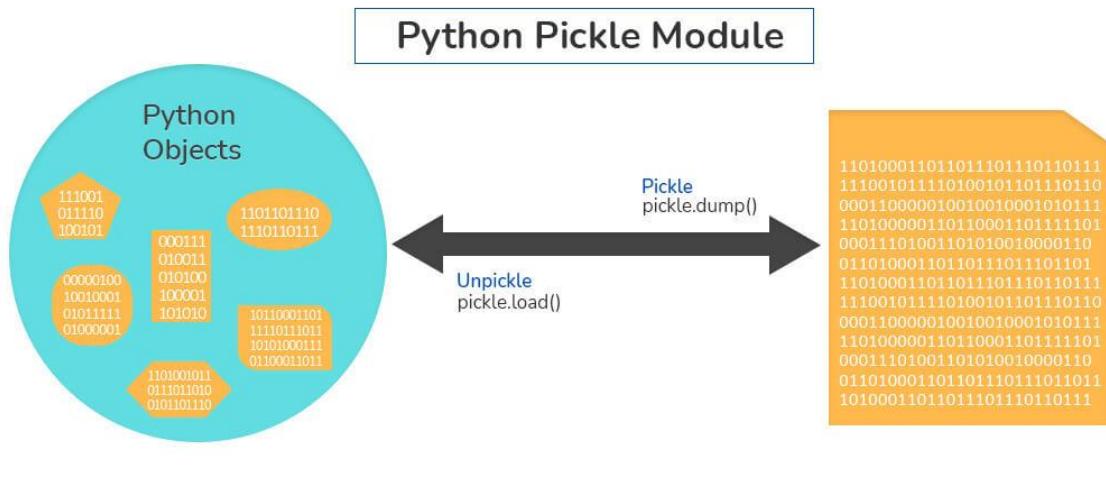
Another way is to use `match()` method from the `re` (regex) module as shown:

```
import re  
print(bool(re.match('[A-Za-z0-9]+$', 'abdc1321'))) # Output: True  
print(bool(re.match('[A-Za-z0-9]+$', 'xyz@123$'))) # Output: False
```

75. What are the differences between pickling and unpickling?

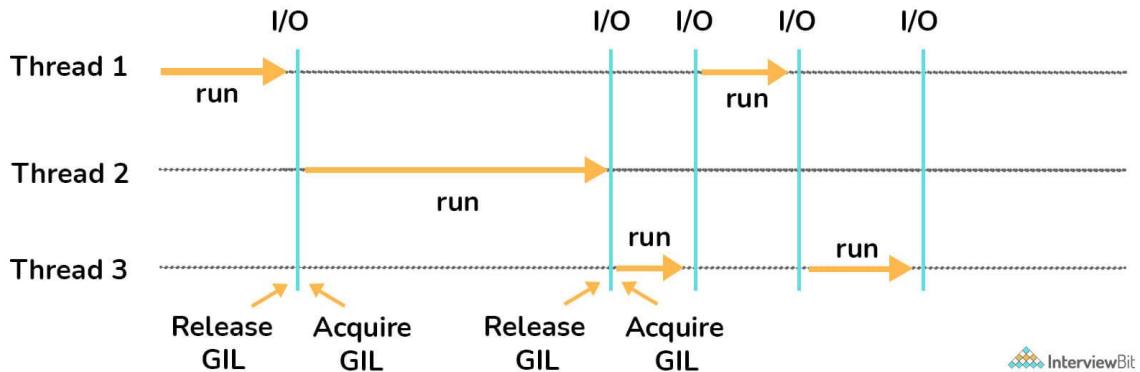
Pickling is the conversion of python objects to binary form. Whereas, unpickling is the conversion of binary form data to python objects. The pickled objects are used for storing in disks or external memory locations. Unpickled objects are used for getting the data back as python objects upon which processing can be done in python.

Python provides a `pickle` module for achieving this. Pickling uses the `pickle.dump()` method to dump python objects into disks. Unpickling uses the `pickle.load()` method to get back the data as python objects.



76. Define GIL.

GIL stands for Global Interpreter Lock. This is a mutex used for limiting access to python objects and aids in effective thread synchronization by avoiding deadlocks. GIL helps in achieving multitasking (and not parallel computing). The following diagram represents how GIL works.



Based on the above diagram, there are three threads. First Thread acquires the GIL first and starts the I/O execution. When the I/O operations are done, thread 1 releases the acquired GIL which is then taken up by the second thread. The process repeats and the GIL are used by different threads alternatively until the threads have completed their execution. The threads not having the GIL lock goes into the waiting state and resumes execution only when it acquires the lock.

77. Define PYTHONPATH.

It is an environment variable used for incorporating additional directories during the import of a module or a package. PYTHONPATH is used for checking if the imported packages or modules are available in the existing directories. Not just that, the interpreter uses this environment variable to identify which module needs to be loaded.

78. Define PIP.

PIP stands for Python Installer Package. As the name indicates, it is used for installing different python modules. It is a command-line tool providing a seamless interface for installing different python modules. It searches over the internet for the package and installs them into the working directory without the need for any interaction with the user. The syntax for this is:

```
pip install <package_name>
```

79. Are there any tools for identifying bugs and performing static analysis in python?

Yes, there are tools like PyChecker and Pylint which are used as static analysis and linting tools respectively. PyChecker helps find bugs in python source code files and raises alerts for code issues and their complexity. Pylint checks for the module's coding standards and supports different plugins to enable custom features to meet this requirement.

80. Differentiate between deep and shallow copies.

- Shallow copy does the task of creating new objects storing references of original elements. This does not undergo recursion to create copies of nested objects. It just copies the reference details of nested objects.
- Deep copy creates an independent and new copy of an object and even copies all the nested objects of the original element recursively.

81. What is main function in python? How do you invoke it?

In the world of programming languages, the main is considered as an entry point of execution for a program. But in python, it is known that the interpreter serially interprets the file line-by-line. This means that python does not provide `main()` function explicitly. But this doesn't mean that we cannot simulate the execution of main. This can be done by defining user-defined `main()` function and by using the `__name__` property of python file. This `__name__` variable is a special built-in variable that points to the name of the current module. This can be done as shown below:

```
def main():
    print("Hi Interviewbit!")
if __name__=="__main__":
    main()
```

Python Programming Examples

82. Write python function which takes a variable number of arguments.

A function that takes variable arguments is called a function prototype.

Syntax:

```
def function_name(*arg_list)
```

For example:

```
def func(*var):
    for i in var:
        print(i)
func(1)
func(20,1,6)
```

The * in the function argument represents variable arguments in the function.

83. WAP (Write a program) which takes a sequence of numbers and check if all numbers are unique.

You can do this by converting the list to set by using set() method and comparing the length of this set with the length of the original list. If found equal, return True.

```
def check_distinct(data_list):
    if len(data_list) == len(set(data_list)):
        return True
    else:
        return False;
print(check_distinct([1,6,5,8])) #Prints True
print(check_distinct([2,2,5,5,7,8])) #Prints False
```

84. Write a program for counting the number of every character of a given text file.

The idea is to use collections and pprint module as shown below:

```
import collections
```

```

import pprint
with open("sample_file.txt", 'r') as data:
    count_data = collections.Counter(data.read().upper())
    count_value = pprint.pformat(count_data)
print(count_value)

```

85. Write a program to check and return the pairs of a given array A whose sum value is equal to a target value N.

This can be done easily by using the phenomenon of hashing. We can use a hash map to check for the current value of the array, x. If the map has the value of (N-x), then there is our pair.

```

def print_pairs(arr, N):
    # hash set
    hash_set = set()

    for i in range(0, len(arr)):
        val = N - arr[i]
        if (val in hash_set):  # check if N-x is there in set, print the pair
            print("Pairs " + str(arr[i]) + ", " + str(val))
        hash_set.add(arr[i])

# driver code
arr = [1, 2, 40, 3, 9, 4]
N = 3
print_pairs(arr, N)

```

86. Write a Program to add two integers >0 without using the plus operator.

We can use bitwise operators to achieve this.

```

def add_nums(num1, num2):
    while num2 != 0:
        data = num1 & num2
        num1 = num1 ^ num2
        num2 = data << 1
    return num1
print(add_nums(2, 10))

```

87. Write a Program to solve the given equation assuming that a,b,c,m,n,o are constants:

$$\begin{aligned} ax + by &= c \\ mx + ny &= 0 \end{aligned}$$

By solving the equation, we get:

```
a, b, c, m, n, o = 5, 9, 4, 7, 9, 4
temp = a*n - b*m
if n != 0:
    x = (c*n - b*o) / temp
    y = (a*o - m*c) / temp
    print(str(x), str(y))
```

88. Write a Program to match a string that has the letter ‘a’ followed by 4 to 8 ‘b’s.

We can use the re module of python to perform regex pattern comparison here.

```
import re
def match_text(txt_data):
    pattern = 'ab{4,8}'
    if re.search(pattern, txt_data):  #search for pattern in txt_data
        return 'Match found'
    else:
        return('Match not found')
print(match_text("abc"))      #prints Match not found
print(match_text("aabbbbbc")) #prints Match found
```

89. Write a Program to convert date from yyyy-mm-dd format to dd-mm-yyyy format.

We can again use the re module to convert the date string as shown below:

```
import re
def transform_date_format(date):
    return re.sub(r'(\d{4})-(\d{1,2})-(\d{1,2})', '\2-\1', date)
date_input = "2021-08-01"
print(transform_date_format(date_input))
```

You can also use the datetime module as shown below:

```
from datetime import datetime
new_date = datetime.strptime("2021-08-01", "%Y-%m-%d").strftime("%d:%m:%Y")
print(new_data)
```

90. Write a Program to combine two different dictionaries. While combining, if you find the same keys, you can add the values of these same keys. Output the new dictionary

We can use the Counter method from the collections module

```
from collections import Counter
d1 = {'key1': 50, 'key2': 100, 'key3':200}
d2 = {'key1': 200, 'key2': 100, 'key4':300}
new_dict = Counter(d1) + Counter(d2)
print(new_dict)
```

91. How will you access the dataset of a publicly shared spreadsheet in CSV format stored in Google Drive?

We can use the StringIO module from the io module to read from the Google Drive link and then we can use the pandas library using the obtained data source.

```
from io import StringIO
import pandas
csv_link = "https://docs.google.com/spreadsheets/d/..."
data_source = StringIO(requests.get(csv_link).content))
dataframe = pd.read_csv(data_source)
print(dataframe.head())
```

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

LIST vs TUPLES	
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)

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. Learn more about Big Data and its applications from the [Data Engineering Training](#).

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. Python an interpreted language. Explain.

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.What are the benefits of using Python?

Ans: The benefits of using python are-

- 1.

- Easy to use**— Python is a high-level programming language that is easy to use, read, write and learn.
- Interpreted language**— Since python is interpreted language, it executes the code line by line and stops if an error occurs in any line.
- Dynamically typed**— the developer does not assign data types to variables at the time of coding. It automatically gets assigned during execution.
- Free and open-source**— Python is free to use and distribute. It is open source.
- Extensive support for libraries**— Python has vast libraries that contain almost any function needed. It also further provides the facility to import other packages using Python Package Manager(pip).
- Portable**— Python programs can run on any platform without requiring any change.
- The data structures used in python are user friendly.
- It provides more functionality with less coding.

Find out our Python Training in Top Cities/Countries

India	USA	Other Cities/Countries
Bangalore	New York	UK
Hyderabad	Chicago	London
Delhi	Atlanta	Canada
Chennai	Houston	Toronto
Mumbai	Los Angeles	Australia
Pune	Boston	UAE
Kolkata	Miami	Dubai
Ahmedabad	San Francisco	Philippines

Q7.What are Python namespaces?

Ans: A namespace in python refers to the name which is assigned to each object in python. The objects are variables and functions. As each object is created, its name along with space(the address of the outer function in which the object is), gets created. The namespaces are maintained in python like a dictionary where

the key is the namespace and value is the address of the object. There 4 types of namespace in python-

1. **Built-in namespace**— These namespaces contain all the built-in objects in python and are available whenever python is running.
2. **Global namespace**— These are namespaces for all the objects created at the level of the main program.
3. **Enclosing namespaces**— These namespaces are at the higher level or outer function.
4. **Local namespaces**— These namespaces are at the local or inner function.

Q8.What are decorators in Python?

Ans: Decorators are used to add some design patterns to a function without changing its structure. Decorators generally are defined before the function they are enhancing. To apply a decorator we first define the decorator function. Then we write the function it is applied to and simply add the decorator function above the function it has to be applied to. For this, we use the @ symbol before the decorator.

Q9.What are Dict and List comprehensions?

Ans: Dictionary and list comprehensions are just another concise way to define dictionaries and lists.

Example of list comprehension is-

```
1                               x=[i for i in range(5)]
```

The above code creates a list as below-

```
1                               4  
2                               [0,1,2,3,4]
```

Example of dictionary comprehension is-

```
1                               x=[i : i+2 for i in range(5)]
```

The above code creates a list as below-

```
1                               [0: 2, 1: 3, 2: 4, 3: 5, 4: 6]
```

Q10.What are the common built-in data types in Python?

Ans: The common built-in data types in python are-

Numbers– They include integers, floating-point numbers, and complex numbers. Eg. `1, 7.9, 3+4i`

List– An ordered sequence of items is called a list. The elements of a list may belong to different data types. Eg. `[5, 'market', 2.4]`

Tuple– It is also an ordered sequence of elements. Unlike lists , tuples are immutable, which means they can't be changed. Eg. `(3, 'tool', 1)`

String– A sequence of characters is called a string. They are declared within single or double-quotes. Eg. “`Sana`”, ‘`She is going to the market`’, etc.

Set– Sets are a collection of unique items that are not in order. Eg. `{7, 6, 8}`

Dictionary– A dictionary stores values in key and value pairs where each value can be accessed through its key. The order of items is not important.

Eg. `{1: 'apple', 2: 'mango'}`

Boolean– There are 2 boolean values- **True** and **False**.

Q11.What is the difference between .py and .pyc files?

Ans: The .py files are the python source code files. While the .pyc files contain the bytecode of the python files. .pyc files are created when the code is imported from some other source. The interpreter converts the source .py files to .pyc files which helps by saving time. You can get a better understanding with the [Data Engineering Course in Washington](#).

Q12.What is slicing in Python?

Ans: Slicing is used to access parts of sequences like lists, tuples, and strings. The syntax of slicing is-`[start:end:step]`. The step can be omitted as well. When we write `[start:end]` this returns all the elements of the sequence from the start (inclusive) till the end-1 element. If the start or end element is negative i, it means the ith element from the end. The step indicates the jump or how many elements have to be skipped. Eg. if there is a list- `[1, 2, 3, 4, 5, 6, 7, 8]`. Then `[-1:2:2]` will return elements starting from the last element till the third element by printing every second element.i.e. `[8, 6, 4]`.

Q13.What are Keywords in Python?

Ans: Keywords in python are reserved words that have special meaning. They are generally used to define type of variables. Keywords cannot be used for variable or function names. There are following 33 keywords in python-

- And
- Or
- Not
- If
- Elif
- Else
- For
- While
- Break
- As
- Def
- Lambda
- Pass
- Return
- True
- False
- Try
- With
- Assert
- Class
- Continue
- Del
- Except
- Finally
- From
- Global
- Import
- In
- Is
- None
- Nonlocal
- Raise
- Yield

Q14.What are Literals in Python and explain about different Literals

Ans: A literal in python source code represents a fixed value for primitive data types. There are 5 types of literals in python-

1. **String literals**— A string literal is created by assigning some text enclosed in single or double quotes to a variable. To create multiline literals, assign the multiline text enclosed in triple quotes. Eg.`name="Tanya"`

2. **A character literal**— It is created by assigning a single character enclosed in double quotes. Eg. `a='t'`
 3. **Numeric literals** include numeric values that can be either integer, floating point value, or a complex number. Eg. `a=50`
 4. **Boolean literals**— These can be 2 values- either True or False.
 5. **Literal Collections**— These are of 4 types-
 - a) List collections-Eg. `a=[1,2,3,'Amit']`
 - b) Tuple literals- Eg. `a=(5,6,7,8)`
 - c) Dictionary literals- Eg. `dict={1: 'apple', 2: 'mango, 3: 'banana'}`
 - d) Set literals- Eg. `{“Tanya”, “Rohit”, “Mohan”}`
6. Special literal- Python has 1 special literal None which is used to return a null variable.

Q15.How to combine dataframes in pandas?

Ans: The dataframes in python can be combined in the following ways-

1. Concatenating them by stacking the 2 dataframes vertically.
2. Concatenating them by stacking the 2 dataframes horizontally.
3. Combining them on a common column. This is referred to as joining.

The concat() function is used to concatenate two dataframes. Its syntax is- `pd.concat([dataframe1, dataframe2])`.

Dataframes are joined together on a common column called a key. When we combine all the rows in dataframe it is union and the join used is outer join. While, when we combine the common rows or intersection, the join used is the inner join. Its syntax is- `pd.concat([dataframe1, dataframe2], axis='axis', join='type_of_join')`

Q16.What are the new features added in Python 3.9.0.0 version?

Ans: The new features in Python 3.9.0.0 version are-

- New Dictionary functions Merge() and Update(|=)
- New String Methods to Remove Prefixes and Suffixes
- **Type Hinting Generics in Standard Collections**
- New Parser based on PEG rather than LL1
- New modules like zoneinfo and graphlib
- **Improved Modules like ast, asyncio, etc.**

- Optimizations such as optimized idiom for assignment, signal handling, optimized python built ins, etc.
- Deprecated functions and commands such as deprecated parser and symbol modules, deprecated functions, etc.
- Removal of erroneous methods, functions, etc.

Q17. How is memory managed in Python?

Ans: Memory is managed in Python in the following ways:

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.

Q18. What is namespace in Python?

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

Q19. 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.



Q20. 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

Q21.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:

```
1           a=2
2           def add():
3               b=3
4               c=a+b
5               print(c)
6           add()
```

Output: 5

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

Q22. Is python case sensitive?

Ans: Yes. Python is a case sensitive language.

Q23.What is type conversion in Python?

Ans: Type conversion refers to the conversion of one data type into 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,img) – This function converts real numbers to complex(real,img) number.

Q24. 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%

Q25. 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.

Q26. 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:

```

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

```

Output:

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

Q27. 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:

```

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

```

Output: Hi, Welcome to Edureka

Q28.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.

```
1                     class Employee:  
2                         def __init__(self, name, age,salary):  
3                             self.name = name  
4                             self.age = age  
5                             self.salary = 20000  
6                         E1 = Employee("XYZ", 23, 20000)  
7                         # E1 is the instance of class Employee.  
8                         #__init__ allocates memory for E1.  
9                         print(E1.name)  
10                        print(E1.age)  
11                        print(E1.salary)
```

Output:

XYZ

23

20000

Q29.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:

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

Output: 11

Q30. 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.

Q31. How does break, continue and pass work?

Break	Allows loop termination when some condition is met and the control goes to the next statement.
Continue	Allows skipping some part of a loop when some specific condition is met and control is transferred to the beginning of the loop
Pass	Used when you need some block of code syntactically, but you want no execution. This is basically a null operation. Nothing happens when the code is executed.

Q32. What does [::-1] do?

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

For example:

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

Q33. How can you randomize the items of a list in place in Python?

Ans: Consider the example shown below:

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

The output of the following code is as below.

[‘Flying’, ‘Keep’, ‘Blue’, ‘High’, ‘The’, ‘Flag’]

Q34. What are python iterators?

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

Q35. 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:

```
1          import random  
2          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].It 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 independent multiple random number generators.

Q36. 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 xrange 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.

Data Science Training

Data Science and Machine Learning Internship Program

Reviews

5(13)

Python Certification Training Course

Reviews

5(40833)

Data Science with Python Certification Course

Reviews

5(111964)

Python Machine Learning Certification Training

Reviews

5(13308)

Data Analytics with R Programming Certification Training

Reviews

5(25957)

Data Science with R Programming Certification Training Course

Reviews

5(39636)

SAS Training and Certification

Reviews

5(5295)

Statistics Essentials for Analytics

Reviews

5(6366)

Analytics for Retail Banks

Reviews

5(1563)

Next

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.

Q37. 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:

```
1 <span data-mce-type="bookmark" style="display: inline-block; width: 0px; overflow:  
2 <pre><span>#Comments in Python s  
3 print("Comments in Python start w
```

Output: Comments in Python start with a #

Q38. 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.

Q39. What are the generators in python?

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

Q40. 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.

Q41. How will you convert a string to all lowercase?

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

Example:

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

Output: abcd

Q42. 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 use 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.

Q43.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:

```
1                                     """  
2                                     Using docstring as a comment.  
3                                     This code divides 2 numbers  
4                                     """  
5                                     x=8  
6                                     y=4  
7                                     z=x/y  
8                                     print(z)
```

Output: 2.0

Q44. 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

Q45. 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.

Q46. 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.

Q47. 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.

```
1           dict={'Country':'India','Capital':'Delhi','PM':'Modi'}  
1           print dict[Country]
```

Output:India

```
1 print dict[Capital]
```

Output:Delhi

```
1 print dict[PM]
```

Output:Modi

Q48. 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]

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.

Q49. 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.

Q50. What does len() do?

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

Example:

```
1 stg='ABCD'
```

```
2 len(stg)
```

Output:4

Q51. 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.

Q52. 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 used 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.

Q53. What are Python packages?

Ans: Python packages are namespaces containing multiple modules.

Q54. 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:

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

Q55. 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

Q56. 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.

Q57. 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:

```

1          a=arr.array('d',[1.1 ,2.1 ,3.1] )
2          a.append(3.4)
3          print(a)
4          a.extend([4.5,6.3,6.8])
5          print(a)
6          a.insert(2,3.8)
7          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])
```

Q58. 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:

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

Output:

4.6

3.1

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

Q59. 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 a procedural as well as structural language.

Check out these [AI and ML courses](#) by E & ICT Academy NIT Warangal to learn Python usage in AI ML and build a successful career.

Q60. 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.

Q61. 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.

Q62. What is the process of compilation and linking in python?

Ans: The compiling and linking allow 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’.

Q63. 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.

Q64. What is split used for?

The split() method is used to separate a given [String in Python](#).

Example:

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

Output: ['edureka', 'python']

Q65. How to import modules in python?

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

Example:

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

Next, in this Python Interview Questions blog, let's have a look at Object Oriented Concepts in Python.

Here is the list of Top 10 Trending Technologies in 2023 that will be in demand!

These Python Interview Questions and Answers will help you prepare for Python job interviews. Start your preparation by going through the most frequently asked questions on Python.

OOPS Python Interview Questions

Q66. 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 is 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.

Q67. How are classes created in Python?

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

Example:

```
1                     class Employee:  
2                         def __init__(self, name):  
3                             self.name = name  
4                         E1=Employee("abc")  
5                         print(E1.name)
```

Output: abc

Q68. 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:

```
1                     # m.py
```

```
2                     class MyClass:  
3                         def f(self):  
4                             print "f()"
```

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

```
1                     import m  
2                     def monkey_f(self):  
3                         print "monkey_f()"  
4  
5                     m.MyClass.f = monkey_f  
6  
7                     obj = m.MyClass()  
8                     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*.

Q69. 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.

Q70. 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.

Q71. Define encapsulation in Python?

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

Q72. 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.

Q73. 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.

Q74. 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-

```
1         class a:  
2             pass  
3             obj=a()  
4             obj.name="xyz"  
5             print("Name = ",obj.name)
```

Output:

Name = xyz

Q75. 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.

Next, let us have a look at some Basic Python Programs in these Python Interview Questions.

Basic Python Programs – Python Interview Questions

Q76. Write a program in Python to execute the Bubble sort algorithm.

```
1         def bs(a):
```

```
2 # a = name of list
3 b=len(a)-1nbsp;
4 # minus 1 because we always compare 2 adjacent values
5 for x in range(b):
6     for y in range(b-x):
7         a[y]=a[y+1]
8
9 a=[32,5,3,6,7,54,87]
10 bs(a)
```

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

Q77. Write a program in Python to produce Star triangle.

```
1 def pyfunc(r):
2     for x in range(r):
3         print(' '*(r-x-1)+'*'*(2*x+1))
4
pyfunc(9)
```

Output:

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

Q78. Write a program to produce Fibonacci series in Python.

```
1 # Enter number of terms needednbsp;#0,1,1,2,3,5....
2 a=int(input("Enter the terms"))
```

```
3             f=0;#first element of series
4             s=1#second element of series
5             if a==0:
6                 print("The requested series is",f)
7             else:
8                 print(f,s,end=" ")
9                 for x in range(2,a):
10                    print(next,end=" ")
11                    f=s
12                    s=next
```

Output: Enter the terms 5 0 1 1 2 3

Q79. Write a program in Python to check if a number is prime.

```
1             a=int(input("enter number"))
2             if a==1:
3                 for x in range(2,a):
4                     if(a%x)==0:
5                         print("not prime")
6                         break
7                     else:
8                         print("Prime")
9                     else:
10                        print("not prime")
```

Output:

enter number 3

Prime

Q80. Write a program in Python to check if a sequence is a Palindrome.

```
1             a=input("enter sequence")
2             b=a[::-1]
3             if a==b:
4                 print("palindrome")
5             else:
6                 print("Not a Palindrome")
```

Output:

enter sequence 323 palindrome

Q81. 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.

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

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

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

Q82. Write a sorting algorithm for a numerical dataset in Python.

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

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

Q83. Looking at the below code, write down the final values of A0, A1, ...An.

```

1          A0 = dict(zip(['a','b','c','d','e'),(1,2,3,4,5)))
2          A1 = range(10)A2 = sorted([i for i in A1 if i in A0])
3          A3 = sorted([A0[s] for s in A0])
4          A4 = [i for i in A1 if i in A3]
5          A5 = {i:i*i for i in A1}
6          A6 = [[i,i*i] for i in A1]
7          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]]

```

Next, in this Python Interview Questions let's have a look at some Python Libraries

Python Libraries – Python Interview Questions

Q84. 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 their 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 them. The user can modify the session if only it has the secret key Flask.secret_key.

Q85. 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.

Q86. 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.

Q87. Discuss Django architecture.

Ans: Django MVT Pattern:

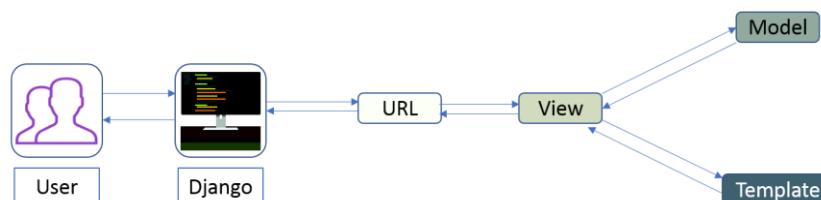


Figure: Python Interview Questions – Django Architecture

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.

Q88. 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.backends.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:

```
1             DATABASES = {  
2                 'default': {  
3                     'ENGINE': 'django.db.backends.sqlite3',  
4                     'NAME': os.path.join(BASE_DIR, 'db.sqlite3'),  
5                 }  
6             }
```

Q89. Give an example how you can write a VIEW in Django?

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

```
1             from django.http import HttpResponseRedirect  
2             import datetime  
3  
4             def Current_datetime(request):
```

```

5           now = datetime.datetime.now()
6           html = "It is now %s/body/html % now
7           return HttpResponseRedirect(html)

```

Returns the current date and time, as an HTML document

Q90. 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.

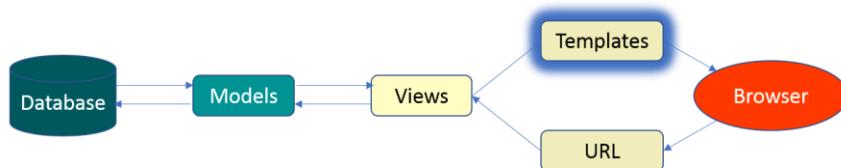


Figure: Python Interview Questions – Django Template

Q91. 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.

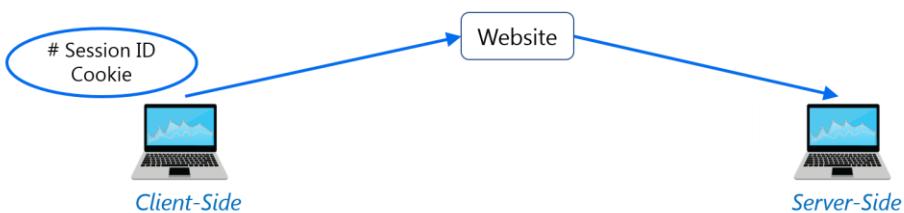


Figure: Python Interview Questions – Django Framework

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

Q92. 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.

Next in this Python Interview Question blog, let's have a look at questions related to Web Scraping

Web Scraping – Python Interview Questions

Q93. 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

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

Q94. 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>

Q95. 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:

```
1 from bs4 import BeautifulSoup  
2  
3 import requests  
4  
5 import sys
```

```

6         url = '<a href="http://www.imdb.com/chart/top">http://www.imdb.com/chart/top'
7             response = requests.get(url)
8             soup = BeautifulSoup(response.text)
9             tr = soup.findChildren("tr")
10            tr = iter(tr)
11            next(tr)
12
13            for movie in tr:
14                title = movie.find('td', {'class': 'titleColumn'}).find('a').content
15                year = movie.find('td', {'class': 'titleColumn'}).find('span', {'class': 'secondaryInfo'})
16                rating = movie.find('td', {'class': 'ratingColumn imdbRating'}).find('span')
17                row = title + ' - ' + year + ' ' + ' ' + rating
18
19            print(row)

```

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

Next in this Python Interview Questions blog, let's have a look at questions related to Data Analysis in Python.

Data Analysis – Python Interview Questions

Q96. 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.

Q97. 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](#).

Q98. 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:

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

Output

[4 3 1]

Q99. How do you calculate percentiles with Python/ NumPy?

Ans: We can calculate percentiles with the following code

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

Output:3

Q100. What is the difference between NumPy and SciPy?

Ans:

NumPy	SciPy
It refers to Numerical python.	It refers to Scientific python.
It has fewer new scientific computing features.	Most new scientific computing features belong in SciPy.
It contains less linear algebra functions.	It has more fully-featured versions of the linear algebra modules, as well as many other numerical algorithms.

NumPy has a faster processing speed.

SciPy on the other hand has slower computational speed.

Q101. 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.

Next in this Python Interview Questions blog, let's have a look at some MCQs

Multiple Choice Questions (MCQ) – Python Interview Questions

Q102. 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.

Q103. 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, $5//2 = 2.5$

Q104. 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.

Q105. 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.

Q106. 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.

Q107. What is the output of the following?

```
1                                         try:  
2                                         if '1' != 1:  
3                                         raise "someError"  
4                                         else:
```

```

5           print("someError has not occurred")
6           except "someError":
7           print ("someError has occurred")

```

- | | | | |
|----|-----------|---------|-----------|
| a) | someError | has | occurred |
| b) | someError | has | occurred |
| 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.

Q108. 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.

Q109. 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.

Q110. What is the output of the following?

- | | |
|---|----------|
| 1 | f = None |
| 2 | |

```

3         for i in range (5):
4             with open("data.txt", "w") as f:
5                 if (i > 2):
6                     break
7
8         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.

Q111. 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.

1. What is Python?

- Python is a **high-level, interpreted**, interactive, and **object-oriented** scripting language. It uses English keywords frequently. Whereas, other languages use punctuation, Python has fewer syntactic constructions.
- Python is designed to be highly **readable** and **compatible** with different platforms such as Mac, Windows, Linux, Raspberry Pi, etc.

Python is one of the most demanding skills right now in the market. Enroll in our [Python Certification Course](#) and become a Python Expert.

2. Python is an interpreted language. Explain.

An interpreted language is any programming language that executes its statements line by line. Programs written in Python run directly from the source code, with no intermediary compilation step.

3. What is the difference between lists and tuples?

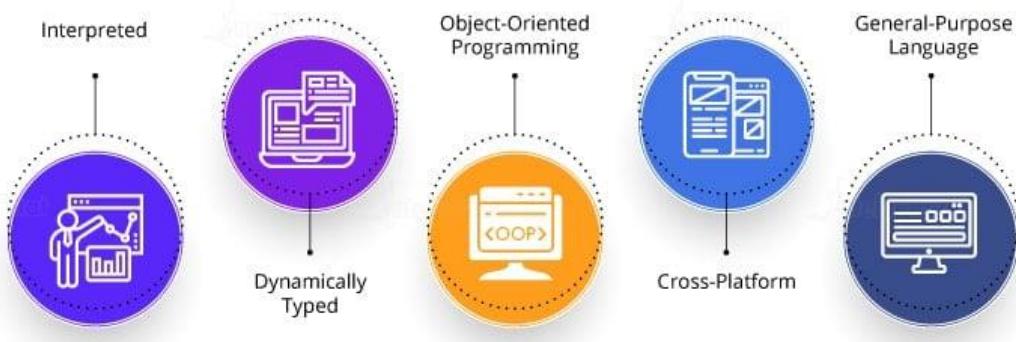
Lists	Tuples
Lists are mutable, i.e., they can be edited	Tuples are immutable (they are lists that cannot be edited)
Lists are usually slower than tuples	Tuples are faster than lists
Lists consume a lot of memory	Tuples consume less memory when compared to lists
Lists are less reliable in terms of errors as unexpected changes are more likely to occur	Tuples are more reliable as it is hard for any unexpected change to occur
Lists consist of many built-in functions.	Tuples do not consist of any built-in functions.
Syntax:	Syntax:
list_1 = [10, 'Intellipaat', 20]	tup_1 = (10, 'Intellipaat', 20)

4. What is pep 8?

PEP in Python stands for Python Enhancement Proposal. It is a set of rules that specify how to write and design Python code for maximum readability.

5. What are the Key features of Python?

The key features of Python are as follows:



- Python is an interpreted language, so it doesn't need to be compiled before execution, unlike [languages such as C](#).
- Python is dynamically typed, so there is no need to declare a variable with the data type. Python Interpreter will identify the data type on the basis of the value of the variable.

For example, in Python, the following code line will run without any error:

```
a = 100
a = "Intellipaat"
```

- Python follows an **object-oriented programming** paradigm with the exception of having access specifiers. Other than access specifiers (public and private keywords), Python has classes, inheritance, and all other usual OOPs concepts.
- Python is a **cross-platform language**, i.e., a Python program written on a Windows system will also run on a Linux system with little or no modifications at all.
- Python is literally a **general-purpose language**, i.e., Python finds its way in various domains such as web application development, automation, Data Science, Machine Learning, and more.

6. How is Memory managed in Python?

- Memory in Python is managed by Python private heap space. All Python objects and data structures are located in a private heap. This

private heap is taken care of by Python Interpreter itself, and a programmer doesn't have access to this private heap.

- Python memory manager takes care of the allocation of Python private heap space.
- Memory for Python private heap space is made available by Python's in-built garbage collector, which recycles and frees up all the unused memory.

7. What is PYTHONPATH?

PYTHONPATH has a role similar to PATH. This variable tells Python Interpreter where to locate the module files imported into a program. It should include the Python source library directory and the directories containing Python source code. PYTHONPATH is sometimes preset by Python Installer.

8. What are Python Modules?

Files containing Python codes are referred to as [**Python Modules**](#). This code can either be classes, functions, or variables and saves the programmer time by providing the predefined functionalities when needed. It is a file with ".py" extension containing an executable code.

Commonly used built modules are listed below:

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

9. What are python namespaces?

A Python namespace 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 its respective ‘object as value’.

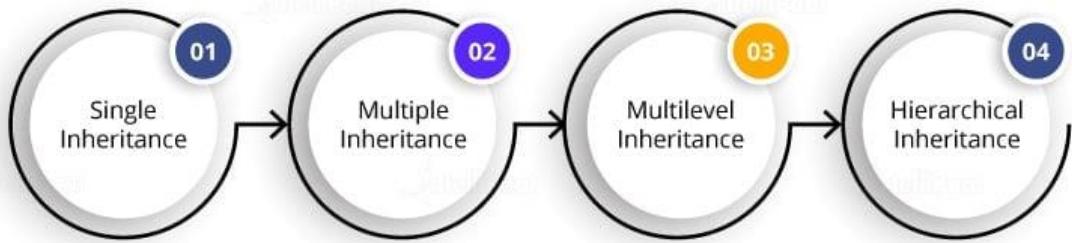
Let’s explore some examples of namespaces:

- **Local Namespace** consists of local names inside a function. It is temporarily created for a function call and gets cleared once the function returns.
- **Global Namespace** consists of names from various imported modules/packages that are being used in the ongoing project. It is created once the package is imported into the script and survives till the execution of the script.
- **Built-in Namespace** consists of built-in functions of core Python and dedicated built-in names for various types of exceptions.

Want to become a master in Python programming? Check out this [Python Training for Data Science](#) and excel in your Python career!

10. Explain Inheritance in Python with an example?

As Python follows an **object-oriented** programming paradigm, classes in Python have the ability to inherit the properties of another class. This process is known as inheritance. Inheritance provides the **code reusability feature**. The class that is being inherited is called a **superclass** or the parent class, and the class that inherits the superclass is called a **derived** or child class. The following types of inheritance are supported in Python:



- **Single inheritance:** When a class inherits only one superclass
- **Multiple inheritance:** When a class inherits multiple superclasses
- **Multilevel inheritance:** When a class inherits a superclass, and then another class inherits this derived class forming a ‘parent, child, and grandchild’ class structure
- **Hierarchical inheritance:** When one superclass is inherited by multiple derived classes

11. What is scope resolution?

A scope is a block of code where an object in Python remains relevant. Each and every object of python functions within its respective scope. As Namespaces uniquely identify all the objects inside a program but these namespaces also have a scope defined for them where you could use their objects without any prefix. It defines the accessibility and the lifetime of a variable.

Let's have a look on scope created as the time of code execution:

- A local scope refers to the local objects included in the current function.
- A global scope refers to the objects that are available throughout execution of the code.
- A module-level scope refers to the global objects that are associated with the current module in the program.
- An outermost scope refers to all the available built-in names callable in the program.

12. What is a dictionary in Python?

Python dictionary is one of the supported [data types in Python](#). It is an unordered collection of elements. The elements in dictionaries are stored as key-value pairs. Dictionaries are indexed by keys.

For example, below we have a dictionary named ‘dict’. It contains two keys, Country and Capital, along with their corresponding values, India and New Delhi.

Syntax:

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

Output: Country: India, Capital: New Delhi

13. What are functions in Python?

A function is a block of code which is executed only when a call is made to the function. **def** keyword is used to define a particular function as shown below:

```
def function():
print("Hi, Welcome to Intellipaat")
function(); # call to the function
```

Output:

Hi, Welcome to Intellipaat

14. What is `__init__` in Python?

Equivalent to constructors in OOP terminology, `__init__` is a reserved method in [Python classes](#). The `__init__` method is called automatically whenever a new object is initiated. This method allocates memory to the new object as soon as it is created. This method can also be used to initialize variables.

Syntax

```
(for defining the __init__ method):
```

```
class Human:
```

```
# init method or constructor
```

```
def __init__(self, age):
    self.age = age
# Sample Method
def say(self):
    print('Hello, my age is', self.age)
h= Human(22)
h.say()
```

Output:

Hello, my age is 22

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

Python supports the below-mentioned built-in data types:

Immutable data types:

- Number
- String
- Tuple

Mutable data types:

- List
- Dictionary
- set

16. What are local variables and global variables in Python?

Local variable: Any variable declared inside a function is known as Local variable and it's accessibility remains inside that function only.

Global Variable: Any variable declared outside the function is known as Global variable and it can be easily accessible by any function present throughout the program.

```
g=4          #global variable
def func_multiply():
l=5      #local variable
m=g*l
return m
func_multiply()
```

Output: 20

If you try to access the local variable outside the multiply function then you will end up with getting an error.

17. What is type conversion in Python?

Python provides you with a much-needed functionality of converting one form of data type into the needed one and this is known as type conversion.

Type Conversion is classified into types:

1. Implicit Type Conversion: In this form of **type conversion python** interpreter helps in automatically converting the data type into another data type without any User involvement.

2. Explicit Type Conversion: In this form of Type conversion the data type is changed into a required type by the user.

Various Functions of explicit conversion are shown below:

int() – function converts any data type into integer.

float() – function converts any data type into float.

ord() – function returns an integer representing the Unicode character

hex() – function converts integers to hexadecimal strings.

oct() – function converts integer to octal strings.

tuple() – function convert to a tuple.

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

`list()` – function converts any data type to a list type.

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

`str()` – function used to convert integer into a string.

`complex(real,imag)` – function used to convert real numbers to complex(real,imag) numbers.

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

For installing Python on Windows, follow the steps shown below:

- Click on this link for installing the python:

[Download Python](#)

- After that, install it on your PC by looking for the location where PYTHON has been installed on your PC by executing the following command on command prompt;

cmd python.

- Visit advanced system settings and after that add a new variable and name it as PYTHON_NAME and paste the path that has been copied.
- Search for the path variable -> select its value and then select ‘edit’.
- Add a semicolon at the end of the value if it’s not present and then type %PYTHON_HOME%

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

List	Array
Consists of elements belonging to different data types	Consists of only those elements having the same data type

No need to import a module for list declaration	Need to explicitly import a module for array declaration
Can be nested to have different type of elements	Must have all nested elements of the same size
Recommended to use for shorter sequence of data items	Recommended to use for longer sequence of data items
More flexible to allow easy modification (addition or deletion) of data	Less flexible since addition or deletion has to be done element-wise
Consumes large memory for the addition of elements	Comparatively more compact in memory size while inserting elements
Can be printed entirely without using looping	A loop has to be defined to print or access the components
Syntax: list = [1,"Hello",'a','e']]	Syntax: import array array_demo = array.array('i', [1, 2, 3]) (array as integer type)

20. Is python case sensitive?

Yes, Python is a case sensitive language. This means that Function and function both are different in pythons like SQL and Pascal.

21. What does [::-1] do?

[::-1] ,this is an example of slice notation and helps to reverse the sequence with the help of indexing.

[Start,stop,step count]

Let's understand with an example of an array:

```
import array as arr
Array_d=arr.array('i',[1,2,3,4,5])
Array_d[::-1]      #reverse the array or sequence
```

Output: 5,4,3,2,1

22. What are Python packages?

A Python package refers to the collection of different sub-packages and modules based on the similarities of the function.

23. What are decorators in Python?

In Python, decorators are necessary functions that help add functionality to an existing function without changing the structure of the function at all. These are represented by `@decorator_name` in Python and are called in a bottom-up format.

Let's have a look how it works:

```
def decorator_lowercase(function): # defining python decorator
def wrapper():
func = function()
input_lowercase = func.lower()
return input_lowercase
return wrapper
@decorator_lowercase ##calling decoractor
def intro():           #Normal function
return 'Hello,I AM SAM'
hello()
```

Output: 'hello,i am sam'

24. Is indentation required in Python?

Indentation in Python is compulsory and is part of its syntax.

All programming languages have some way of defining the scope and extent of the block of codes. In Python, it is indentation. Indentation provides better readability to the code, which is probably why Python has made it compulsory.

.

25. How does break, continue, and pass work?

These statements help to change the phase of execution from the normal flow that is why they are termed loop control statements.

Python break: This statement helps terminate the loop or the statement and pass the control to the next statement.

Python continue: This statement helps force the execution of the next iteration when a specific condition meets, instead of terminating it.

Python pass: This statement helps write the code syntactically and wants to skip the execution. It is also considered a null operation as nothing happens when you execute the pass statement.

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

This can be easily achieved by using the **Shuffle()** function from the **random** library as shown below:

```
from random import shuffle  
List = ['He', 'Loves', 'To', 'Code', 'In', 'Python']  
shuffle(List)  
print(List)
```

Output: ['Loves', 'He', 'To', 'In', 'Python', 'Code']

27. How to comment with multiple lines in Python?

To add a multiple lines comment in python, all the line should be prefixed by #.

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

Generally, Python is an all purpose Programming Language ,in addition to that Python is also Capable to perform scripting.

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

To access an element from ordered sequences, we simply use the index of the element, which is the position number of that particular element. The index usually starts from 0, i.e., the first element has index 0, the second has 1, and so on.

Index from Rear : -6 -5 -4 -3 -2 -1



Index from Front : 0 1 2 3 4 5



When we use the index to access elements from the end of a list, it's called reverse indexing. In reverse indexing, the indexing of elements starts from the last element with the index number ‘-1’. The second last element has index ‘-2’, and so on. These indexes used in reverse indexing are called negative indexes.

Python Intermediate Interview Questions

30. Explain split(), sub(), subn() methods of “re” module in Python?

These methods belong to the [Python RegEx or ‘re’ module](#) and are used to modify strings.

- **split():** This method is used to split a given string into a list.
- **sub():** This method is used to find a substring where a regex pattern matches, and then it replaces the matched substring with a different string.
- **subn():** This method is similar to the sub() method, but it returns the new string, along with the number of replacements.

Learn more about Python from this [Python Training in New York](#) to get ahead in your career!

31. What do you mean by Python literals?

Literals refer to the data which will be provided to a given in a variable or constant.

Literals supported by python are listed below:

String Literals

These literals are formed by enclosing text in the single or double quotes.

For Example:

“Intellipaat”

‘45879’

Numeric Literals

Python numeric literals support three types of literals

Integer:I=10

Float: i=5.2

Complex:1.73j

Boolean Literals

Boolean literals help to denote boolean values. It contains either True or False.

x=True

32. What is a map function in Python?

The map() function in Python has two parameters, function and iterable. The map() function takes a function as an argument and then applies that function to all the elements of an iterable, passed to it as another argument. It returns an object list of results.

For example:

```
def calculateSq(n):  
    return n*n  
numbers = (2, 3, 4, 5)  
result = map( calculateSq, numbers)  
print(result)
```

Interested in learning Python? Check out this [Python Training in Sydney](#)!

33. What are the generators in python?

Generator refers to the function that returns an iterable set of items.

34. What are python iterators?

These are the certain objects that are easily traversed and iterated when needed.

35. Do we need to declare variables with data types in Python?

No. Python is a dynamically typed language, I.E., Python Interpreter automatically identifies the data type of a variable based on the type of value assigned to the variable.

36. What are Dict and List comprehensions?

[Python comprehensions](#) are like decorators, that help to build altered and filtered lists, dictionaries, or sets from a given list, dictionary, or set. Comprehension saves a lot of time and code that might be considerably more complex and time-consuming.

Comprehensions are beneficial in the following scenarios:

- Performing mathematical operations on the entire list
- Performing conditional filtering operations on the entire list
- Combining multiple lists into one
- Flattening a multi-dimensional list

For example:

```
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}
```

37. How do you write comments in python?

[Python Comments](#) are the statement used by the programmer to increase the readability of the code. With the help of #, you can define the single comment and the other way to do commenting is to use the docstrings(strings enclosed within triple quotes).

For example:

```
#Comments in Python
print("Comments in Python ")
```

Master Python by taking up this online [Python Course in Toronto!](#)!

38. Is multiple inheritance supported in Python?

Yes, unlike Java, Python provides users with a wide range of support in terms of inheritance and its usage. Multiple inheritance refers to a scenario where a class is instantiated from more than one individual parent class. This provides a lot of functionality and advantages to users.

39. What is the difference between range & xrange?

Functions in Python, range() and xrange() are used to iterate in a for loop for a fixed number of times. Functionality-wise, both these functions are the same. The difference comes when talking about Python version support for these functions and their return values.

range() Method	xrange() Method
In Python 3, xrange() is not supported; instead, the range() function is used to iterate in for loops	The xrange() function is used in Python 2 to iterate in for loops
It returns a list	It returns a generator object as it doesn't really generate a static list at the run time
It takes more memory as it keeps the entire list of iterating numbers in memory	It takes less memory as it keeps only one number at a time in memory

40. What is pickling and unpickling?

The Pickle module accepts the Python object and converts it into a string representation and stores it into a file by using the dump function. This process is called pickling. On the other hand, the process of retrieving the original Python objects from the string representation is called unpickling.

Want to know about the real-world uses of Python? Read our detailed blog on [Python Applications](#) now.

41. What do you understand by Tkinter?

Tkinter is a built-in Python module that is used to create GUI applications. It is Python's standard toolkit for GUI development. Tkinter comes with Python, so there is no separate installation needed. You can start using it by importing it in your script.

Why Python?



42. Is Python fully object oriented?

Python does follow an object-oriented programming paradigm and has all the basic OOPs concepts such as inheritance, polymorphism, and more, with the exception of access specifiers. Python doesn't support strong encapsulation (adding a private keyword before data members). Although, it has a convention that can be used for data hiding, i.e., prefixing a data member with two underscores.

43. Differentiate between NumPy and SciPy?

NumPy	SciPy
NumPy stands for Numerical Python	SciPy stands for Scientific Python
It is used for efficient and general numeric computations on numerical data saved in arrays. E.g., sorting, indexing, reshaping, and more	This module is a collection of tools in Python used to perform operations such as integration, differentiation, and more
There are some linear algebraic functions available in this module, but they are not full-fledged	Full-fledged algebraic functions are available in SciPy for algebraic computations

44. Explain all file processing modes supported in Python?

Python has various file processing modes.

For opening files, there are three modes:

- read-only mode (r)
- write-only mode (w)
- read-write mode (rw)

For opening a text file using the above modes, we will have to append ‘t’ with them as follows:

- read-only mode (rt)
- write-only mode (wt)
- read-write mode (rwt)

Similarly, a binary file can be opened by appending ‘b’ with them as follows:

- read-only mode (rb)
- write-only mode (wb)
- read-write mode (rwb)

To append the content in the files, we can use the append mode (a):

- For text files, the mode would be ‘at’
- For binary files, it would be ‘ab’

45. What do file-related modules in Python do? Can you name some file-related modules in Python?

Python comes with some file-related modules that have functions to manipulate text files and binary files in a file system. These modules can be used to create text or binary files, update their content, copy, delete, and more.

Some file-related modules are os, os.path, and shutil.os. The os.path module has functions to access the file system, while the shutil.os module can be used to copy or delete files.

46. Explain the use of the 'with' statement and its syntax?

In Python, using the ‘with’ statement, we can open a file and close it as soon as the block of code, where ‘with’ is used, exits. In this way, we can opt for not using the close() method.

```
with open("filename", "mode") as file_var:
```

47. Write a code to display the contents of a file in reverse?

To display the contents of a file in reverse, the following code can be used:

```
for line in reversed(list(open(filename.txt))):  
    print(line.rstrip())
```

48. Which of the following is an invalid statement?

1. xyz = 1,000,000
2. x y z = 1000 2000 3000
3. x,y,z = 1000, 2000, 3000
4. x_y_z = 1,000,000

Ans. 2 statement is invalid.

49. Write a command to open the file c:\hello.txt for writing?

Command:

```
f= open("hello.txt", "wt")
```

50. What does len() do?

len() is an inbuilt function used to calculate the length of sequences like list, [python string](#), and array.

```
my_list=[1,2,3,4,5]  
len(my_list)
```

51. What does *args and **kwargs mean?

- `.*args`: It is used to pass multiple arguments in a function.
- `**kwargs`: It is used to pass multiple keyworded arguments in a function in python.

Want to know about the real-world uses of Python? Read our detailed blog on [Python Project ideas](#) now.

52. How will you remove duplicate elements from a list?

To remove duplicate elements from the list we use the `set()` function.

Consider the below example:

```
demo_list=[5,4,4,6,8,12,12,1,5]
unique_list = list(set(demo_list))
output:[1,5,6,8,12]
```

53. How can files be deleted in Python?

You need to import the OS Module and use `os.remove()` function for deleting a file in python.

consider the code below:

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

54. How will you read a random line in a file?

We can read a random line in a file using the random module.

For example:

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

55. Write a Python program to count the total number of lines in a text file?

```
def file_count(fname):
    with open(fname) as f:
        for i, l in enumerate(f):
            pass
    return i+1
print("Total number of lines in the text file: ", file_count("file.txt"))
```

56. What would be the output if I run the following code block?

```
list1 = [2, 33, 222, 14, 25]
print(list1[-2])
```

1. **14**
2. **33**
3. **25**
4. **Error**

Ans. **output:**14

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

Operators are referred to as special functions that take one or more values(operands) and produce a corresponding result.

- **is:** returns the true value when both the operands are true (Example: “x” is ‘x’)
- **not:** returns the inverse of the boolean value based upon the operands (example:”1” returns “0” and vice-versa.
- **In:** helps to check if the element is present in a given Sequence or not.

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

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

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

The ternary operator is the operator that is used to show [conditional statements in Python](#). This consists of the boolean true or false values with a statement that has to be checked.

Syntax:

```
[on_true] if [expression] else [on_false]
```

Explanation:

The above expression is evaluated like if $x < y$ else y , in this case, if $x < y$ is true then the value is returned as $count=x$ and if it is incorrect then $count=y$ will be stored to result.

60. How to add values to a python array?

In python, adding elements in an array can be easily done with the help of `extend()`,`append()` and `insert()` functions.

Consider the following example:

```
x=arr.array('d', [11.1 , 2.1 ,3.1] )
x.append(10.1)
print(x) #[11.1,2.1,3.1,10.1]
x.extend([8.3,1.3,5.3])
print(x) #[11.1,2.1,3.1,10.1,8.3,1.3,5.3]
x.insert(2,6.2)
```

```
print(x)    # [11.1,2.1,6.2,3.1,10.1,8.3,1.3,5.3]
```

61. How to remove values to a python array?

Elements can be removed from the python array using `pop()` or `remove()` methods.

`pop()`: This function will return the removed element .

`remove()`:It will not return the removed element.

Consider the below example :

```
x=arr.array('d', [8.1, 2.4, 6.8, 1.1, 7.7, 1.2, 3.6])
print(x.pop())
print(x.pop(3))
x.remove(8.1)
print(x)
```

Output:

```
3.6
1.1 # element popped at 3 rd index
array('d', [ 2.4, 6.8, 7.7, 1.2])
```

62. Write a code to sort a numerical list in Python?

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

```
list = ["2", "5", "7", "8", "1"]
list = [int(i) for i in list]
list.sort()
print (list)
```

63. Can you write an efficient code to count the number of capital letters in a file?

The normal solution for this problem statement would be as follows:

```
with open(SOME_LARGE_FILE) as countletter:
```

```
    count = 0
    text = countletter.read()
    for character in text:
        if character.isupper():
            count += 1
```

To make this code more efficient, the whole code block can be converted into a one-liner code using the feature called generator expression. With this, the equivalent code line of the above code block would be as follows:

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

64. How will you reverse a list in Python?

The function `list.reverse()` reverses the objects of a list.

65. How will you remove the last object from a list in Python?

```
list.pop(obj=list[-1]):
```

Here, `-1` represents the last element of the list. Hence, the `pop()` function removes the last object (`obj`) from the list.

Get certified in Python from the top [Python Course in Singapore](#) now!

66. How can you generate random numbers in Python?

This achieved with importing the `random` module, it is the module that is used to generate random numbers.

Syntax:

```
import random
random.random # returns the floating point random number between the range
of [0,1].
```

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

`lower()` function is used to convert a string to lowercase.

For Example:

```
demo_string='ROSES'  
print(demo_string.lower())
```

Learn the complete Python Training in Hyderabad in 24 hours!

68. Why would you use NumPy arrays instead of lists in Python?

NumPy arrays provide users with three main advantages as shown below:

- NumPy arrays consume a lot less memory, thereby making the code more efficient.
- NumPy arrays execute faster and do not add heavy processing to the runtime.
- NumPy has a highly readable syntax, making it easy and convenient for programmers.

69. What is Polymorphism in Python?

Polymorphism is the ability of the code to take multiple forms. Let's say, if the parent class has a method named XYZ then the child class can also have a method with the same name XYZ having its own variables and parameters.

70. Define encapsulation in Python?

encapsulation in Python refers to the process of wrapping up the variables and different functions into a single entity or capsule. Python class is the best example of encapsulation in python.

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

Nested Lists:

- Python lists are efficient general-purpose containers that support efficient operations like insertion, appending, deletion and concatenation.
- The limitations of lists are that they don't support "vectorized" operations like element wise 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.

Numpy:

- NumPy is more efficient and more convenient as you get a lot of vector and matrix operations for free, which helps to avoid unnecessary work and complexity of the code. Numpy is also efficiently implemented when compared to nested
- NumPy array is faster and contains a lot of built-in functions which will help in FFTs, convolutions, fast searching, linear algebra, basic statistics, histograms, etc.

Advanced Python Interview Questions for Experienced Professionals

72. What is the lambda function in Python?

A lambda function is an anonymous function (a function that does not have a name) in Python. To define anonymous functions, we use the 'lambda' keyword instead of the 'def' keyword, hence the name 'lambda function'. Lambda functions can have any number of arguments but only one statement.

For example:

```
l = lambda x,y : x*y
print(l(5, 6))
```

Output:30

73. What is self in Python?

Self is an object or an instance of a class. This is explicitly included as the first parameter in Python. On the other hand, in Java it is optional. It helps 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.

Syntax:

Class A:

```
def func(self):  
    print("Hi")
```

74. What is the difference between append() and extend() methods?

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

- append(element): Adds the given element at the end of the list that called this append() method
- extend(another-list): Adds the elements of another list at the end of the list that called this extend() method

For in-depth knowledge, check out our [Python Tutorial](#) and boost your Python skills!

75. How does Python Flask handle database requests?

Flask supports a database-powered application (RDBS). Such a system requires creating a schema, which needs piping the schema.sql file into the sqlite3 command. Python developers need to install the sqlite3 command to create or initiate the database in Flask.

Flask allows to request for a database in three ways:

- `before_request()`: They are called before a request and pass no arguments.
- `after_request()`: They are called after a request and pass the response that will be sent to the client.
- `teardown_request()`: They are called in a situation when an exception is raised and responses are not guaranteed. They are called after the response has been constructed. They are not allowed to modify the request, and their values are ignored.

76. What is docstring in Python?

Python lets users include a description (or quick notes) for their methods using documentation strings or docstrings. Docstrings are different from regular comments in Python as, rather than being completely ignored by the Python Interpreter like in the case of comments, these are defined within triple quotes.

Syntax:

```
"""
Using docstring as a comment.
This code add two numbers
"""

x=7
y=9
z=x+y
print(z)
```

77. How is Multithreading achieved in Python?

Python has a multi-threading package ,but commonly not considered as good practice to use it as it will result in increased code execution time.

- Python has a constructor called the Global Interpreter Lock (GIL). The GIL ensures that only one of your ‘threads’ can execute at one

time. The process makes sure that a thread acquires the GIL, does a little work, then passes the GIL onto the next thread.

- This happens at a very Quick instance of time and that's why to the human eye it seems like your threads are executing parallelly, but in reality they are executing one by one by just taking turns using the same CPU core.

78. What is slicing in Python?

Slicing is a process used to select a range of elements from sequence data type like list, string and tuple. Slicing is beneficial and easy to extract out the elements. It requires a : (colon) which separates the start index and end index of the field. All the data sequence types List or tuple allows us to use slicing to get the needed elements. Although we can get elements by specifying an index, we get only a single element whereas using slicing we can get a group or appropriate range of needed elements.

Syntax:

```
List_name[start:stop]
```

79. What is functional programming? Does Python follow a functional programming style? If yes, list a few methods to implement functionally oriented programming in Python.

Functional programming is a coding style where the main source of logic in a program comes from functions.

Incorporating functional programming in our codes means writing pure functions.

Pure functions are functions that cause little or no changes outside the scope of the function. These changes are referred to as side effects. To reduce side effects, pure functions are used, which makes the code easy-to-follow, test, or debug.

Python does follow a functional programming style. Following are some examples of functional programming in Python.

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

```
list(filter(lambda x:x>6,range(9))) [7, 8]
```

`map()`: Map applies a function to every element in an iterable.

```
list(map(lambda x:x**2,range(5))) [0, 1, 4, 9, 16, 25]
```

`reduce()`: Reduce repeatedly reduces a sequence pair-wise until it reaches a single value.

```
from functools import reduce >>> reduce(lambda x,y:x-y,[1,2,3,4,5]) -13
```

80. Which one of the following is not the correct syntax for creating a set in Python?

1. `set([[1,2],[3,4],[4,5]])`
2. `set([1,2,2,3,4,5])`
3. `{1,2,3,4}`
4. `set((1,2,3,4))`

Ans.

```
set([[1,2],[3,4],[4,5]])
```

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

81. What is monkey patching in Python?

Monkey patching is the term used to denote the modifications that are done to a class or a module during the runtime. This can only be done as Python supports changes in the behavior of the program while being executed.

The following is an example, denoting monkey patching in Python:

```
# monkeyy.py
class X:
    def func(self):
        print "func() is being called"
```

The above module (`monkeyy`) is used to change the behavior of a function at the runtime as shown below:

```
import monkeyy
def monkey_f(self):
    print "monkey_f() is being called"
    # replacing address of "func" with "monkey_f"
monkeyy.X.func = monkey_f
obj = monk.X()
# calling function "func" whose address got replaced
# with function "monkey_f()"
obj.func()
```

82. What is the difference between / and // operator in Python?

- / : is a division operator and returns the Quotient value.

10/3

3.33

- // : is known as floor division operator and used to return only the value of quotient before decimal

10//3

3

83. What is pandas?

Pandas is an open source python library which supports data structures for data based operations associated with data analyzing and data Manipulation . Pandas with its rich sets of features fits in every role of data operation,whether it be related to implementing different algorithms or for solving complex business problems. Pandas helps to deal with a number of files in performing certain operations on the data stored by files.

84. What are dataframes?

A dataframe refers to a two dimensional mutable data structure or data aligned in the tabular form with labeled axes(rows and column).

Syntax:

```
pandas.DataFrame( data, index, columns, dtype)
```

- **data**:It refers to various forms like ndarray, series, map, lists, dict, constants and can take other DataFrame as Input.
- **index**:This argument is optional as the index for row labels will be automatically taken care of by pandas library.
- **columns**:This argument is optional as the index for column labels will be automatically taken care of by pandas library.
- **Dtype**: refers to the data type of each column.

85. How to combine dataframes in pandas?

The different dataframes can be easily combined with the help of functions listed below:

- **Append()**:

This function is used for horizontal stacking of dataframes.

```
data_frame1.append(data_frame2)
```

- **concat()**: This function is used for vertical stacking and best suites when the dataframes to be combined possess the same column and similar fields.

```
pd.concat([data_frame1, data_frame2])
```

- **join()**: This function is used to extract data from different dataframes which have one or more columns common.

```
data_frame1.join(data_frame2)
```

86. How do you identify missing values and deal with missing values in Dataframe?

Identification:

isnull() and isna() functions are used to identify the missing values in your data loaded into dataframe.

```
missing_count=data_frame1.isnull().sum()
```

Handling missing Values:

There are two ways of handling the missing values :

Replace the missing values with 0

```
df['col_name'].fillna(0)
```

Replace the missing values with the mean value of that column

```
df['col_name'] = df['col_name'].fillna((df['col_name'].mean()))
```

87. What is regression?

Regression is termed as supervised machine learning algorithm technique which is used to find the correlation between variables and help to predict the dependent variable(y) based upon the independent variable (x). It is mainly used for prediction, time series modeling, forecasting, and determining the causal-effect relationship between variables.

[Scikit library](#) is used in python to implement the regression and all machine learning algorithms.

There are two different types of regression algorithms in machine learning :

Linear Regression: Used when the variables are continuous and numeric in nature.

Logistic Regression: Used when the variables are continuous and categorical in nature.

88. What is classification?

Classification refers to a predictive modeling process where a class label is predicted for a given example of input data. It helps categorize the provided input into a label that other observations with similar features have. For example, it can be used for classifying a mail whether it is spam or not, or for checking whether users will churn or not based on their behavior.

These are some of the classification algorithms used in Machine Learning:

- **Decision tree**
- Random forest classifier
- Support vector machine

89. How do you split the data in train and test dataset in python?

This can be achieved by using the scikit machine learning library and importing train_test_split function in python as shown below:

```
Import sklearn.model_selection.train_test_split
```

```
# test size =30% and train= 70 %
```

```
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.33,  
random_state=0).
```

90. What is SVM?

Support vector machine (SVM) is a supervised machine learning model that considers the classification algorithms for two-group classification problems. Support vector machine is a representation of the training data as points in space are separated into categories with the help of a clear gap that should be as wide as possible.

91. Write a code to get the indices of N maximum values from a NumPy array?

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

```
import numpy as np

ar = np.array([1, 3, 2, 4, 5, 6])

print(ar.argsort()[-3:][::-1])
```

92. What is the easiest way to calculate percentiles when using Python?

The easiest and the most efficient way you can calculate percentiles in Python is to make use of NumPy arrays and its functions.

Consider the following example:

```
import numpy as np
a = np.array([1,2,3,4,5,6,7])
p = np.percentile(a, 50) #Returns the 50th percentile which is also the median
print(p)
```

93. Write a Python program to check whether a given string is a palindrome or not, without using an iterative method?

A palindrome is a word, phrase, or sequence that reads the same backward as forward, e.g., madam, nurses run, etc.

Consider the below code:

```
def fun(string):
    s1 = string
    s = string[::-1]
    if(s1 == s):
        return true
    else:
```

```
return false  
print(fun("madam"))
```

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

```
def sum(num):  
if len(num) == 1:  
    return num[0]          #with only one element in the list, the sum result will be  
    equal to the element.  
else:  
    return num[0] + sum(num[1:])  
print(sum([2, 4, 5, 6, 7]))
```

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

```
def bubbleSort(x):  
  
n = len(x)  
  
# Traverse through all array elements  
  
for i in range(n-1):  
  
    for j in range(0, n-i-1):  
  
        if x[j] > x[j+1] :  
  
            x[j], x[j+1] = x[j+1], x[j]  
  
# Driver code to test above  
  
arr = [25, 34, 47, 21, 22, 11, 37]  
  
bubbleSort(arr)
```

```
print ("Sorted array is:")  
  
for i in range(len(arr)):  
  
    print ("%d" %arr[i]),
```

Output:

11,21,22,25,34,37,47

96. Write a program in Python to produce Star triangle?

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

Output:

```
*
```

```
***
```

```
*****
```

```
*****
```

```
*****
```

```
*****
```

```
*****
```

```
*****
```

```
*****
```

```
*****
```

97. Write a program to produce Fibonacci series in Python?

Fibonacci series refers to the series where the element is the sum of two elements prior to it.

```
n = int(input("number of terms? "))  
n1, n2 = 0, 1  
count = 0
```

```
if n <= 0:  
    print("Please enter a positive integer")  
elif n == 1:  
    print("Fibonacci sequence upto",nterms,:")  
    print(n1)  
else:  
    print("Fibonacci sequence:")  
    while count < n:  
        print(n1)  
        nth = n1 + n2  
        n1 = n2  
        n2 = nth  
        count += 1
```

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

```
num = 13  
  
if num > 1:  
  
    for i in range(2, int(num/2)+1):  
  
        if (num % i) == 0:  
  
            print(num, "is not a prime number")  
  
            break  
  
    else:  
  
        print(num, "is a prime number")  
  
    else:  
  
        print(num, "is not a prime number")
```

Output:

13 is a prime number

99. Write a sorting algorithm for a numerical dataset in Python?

code to sort a list in Python:

```
my_list = ["8", "4", "3", "6", "2"]  
  
my_list = [int(i) for i in list]  
  
my_list.sort()  
  
print (my_list)
```

Output:

2,3,4,6,8

100. Write a Program to print ASCII Value of a character in python?

```
x= 'a'  
  
# print the ASCII value of assigned character stored in x  
  
print(" ASCII value of " + x + " is", ord(x))
```

Output: 65

1) What is Python?

What is PYTHON ?



Python was created by **Guido van Rossum**, and released in **1991**.

It is a general-purpose computer programming language. It is a high-level, object-oriented language which can run equally on different platforms such as Windows, Linux, UNIX, and Macintosh. Its high-level built-in data structures, combined with dynamic typing and dynamic binding. It is widely used in data science, machine learning and artificial intelligence domain.

It is easy to learn and require less code to develop the applications.

It is widely used for:

- Web development (server-side).
- Software development.
- Mathematics.
- System scripting.

2) Why Python?

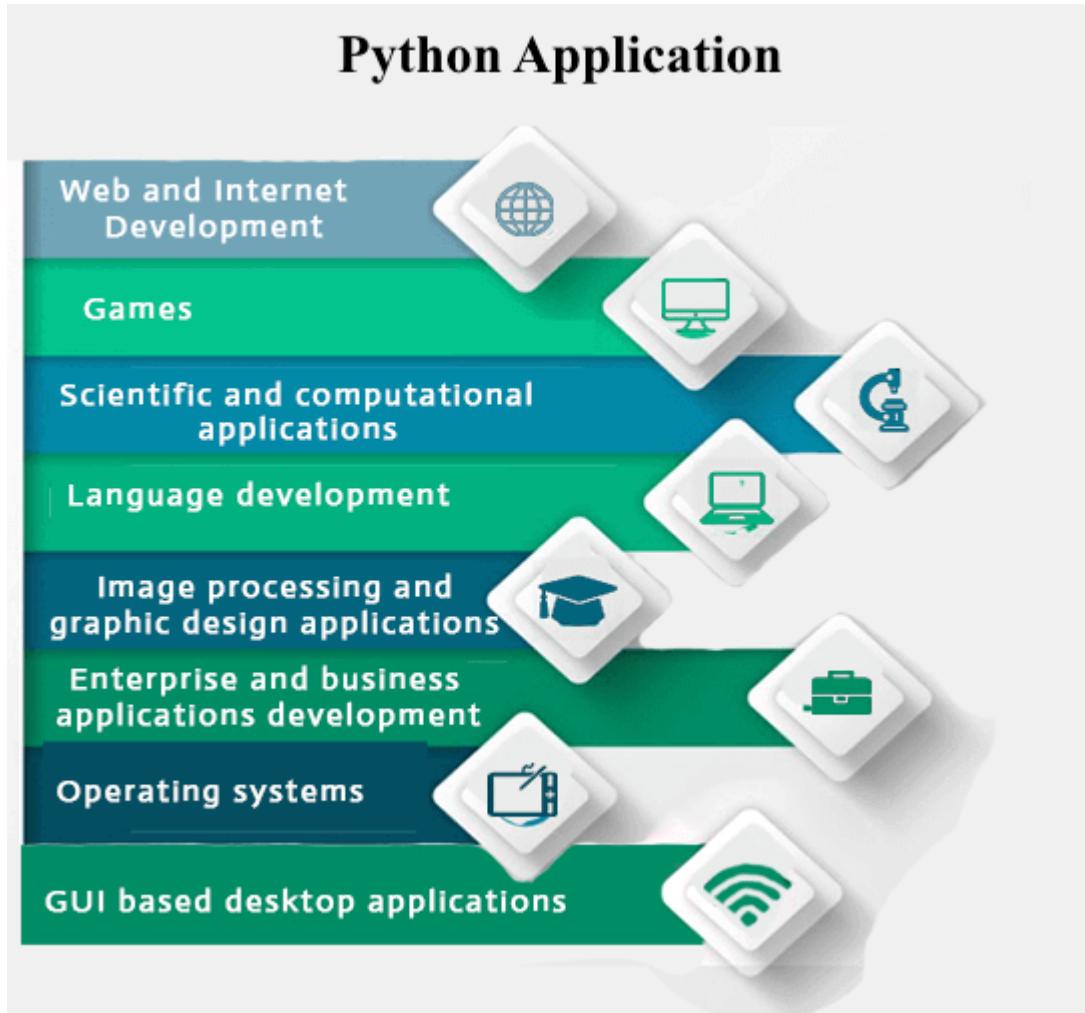
- Python is an interpreted, object-oriented, high-level programming language with dynamic semantics.
 - Python is compatible with different platforms like **Windows, Mac, Linux, Raspberry Pi**, etc.
 - Python has a simple syntax as compared to other languages.
 - Python allows a developer to write programs with fewer lines than some other programming languages.
 - Python runs on an interpreter system, means that the code can be executed as soon as it is written. It helps to provide a prototype very quickly.
 - Python can be described as a procedural way, an object-orientated way or a functional way.
 - The Python interpreter and the extensive standard library are available in source or binary form without charge for all major platforms, and can be freely distributed.
-

3) What are the applications of Python?

Python is used in various software domains some application areas are given below.

- Web and Internet Development
- Games
- Scientific and computational applications
- Language development
- Image processing and graphic design applications
- Enterprise and business applications development
- Operating systems
- GUI based desktop applications

Python Application



Python provides various web frameworks to develop web applications. The popular python web frameworks are **Django**, **Pyramid**, **Flask**.

Python's standard library supports for E-mail processing, FTP, IMAP, and other Internet protocols.

Python's **SciPy** and **NumPy** helps in scientific and computational application development.

Python's **Tkinter** library supports to create a desktop based GUI applications.

4) What are the advantages of Python?

Advantages of Python are:

- Python is **Interpreted** language

Interpreted: Python is an interpreted language. It does not require prior compilation of code and executes instructions directly.

- It is Free and open source

Free and open source: It is an open-source project which is publicly available to reuse. It can be downloaded free of cost.

- It is **Extensible**

Extensible: It is very flexible and extensible with any module.

- Object-oriented

Object-oriented: Python allows to implement the Object-Oriented concepts to build application solution.

- It has Built-in data structure

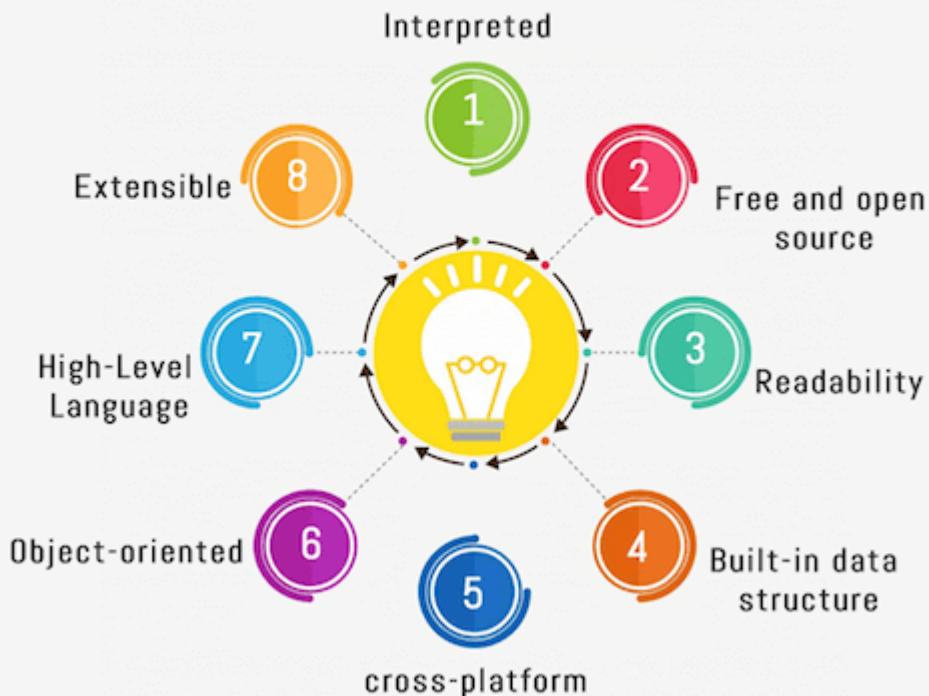
Built-in data structure: Tuple, List, and Dictionary are useful integrated data structures provided by the language.

- Readability
- High-Level Language
- Cross-platform

Portable: Python programs can run on cross platforms without affecting its performance.



Advantages of Python



5) What is PEP 8?

PEP 8 stands for **Python Enhancement Proposal**, it can be defined as a document that helps us to provide the guidelines on how to write the Python code. It is basically a set of rules that specify how to format Python code for maximum readability. It was written by Guido van Rossum, Barry Warsaw and Nick Coghlan in 2001.

6) What do you mean by Python literals?

Literals can be defined as a data which is given in a variable or constant. Python supports the following literals:

String Literals

String literals are formed by enclosing text in the single or double quotes. For example, string literals are string values.

Example:

1. # in single quotes
2. single = 'JavaTpoint'
3. # in double quotes
4. double = "JavaTpoint"
5. # multi-line String
6. multi = """Java
7. T
8. point""
- 9.
10. **print**(single)
11. **print**(double)
12. **print**(multi)

Output:

```
JavaTpoint
JavaTpoint
Java
    T
        point
```

Numeric Literals

Python supports three types of numeric literals integer, float and complex.

Example:

1. # Integer literal
2. a = 10
3. #Float Literal
4. b = 12.3
5. #Complex Literal
6. x = 3.14j
7. **print**(a)
8. **print**(b)
9. **print**(x)

Output:

```
10  
12.3  
3.14j
```

Boolean Literals

Boolean literals are used to denote Boolean values. It contains either True or False.

Example:

1. p = (1 == True)
2. q = (1 == False)
3. r = True + 3
4. s = False + 7
- 5.
6. **print("p is", p)**
7. **print("q is", q)**
8. **print("r:", r)**
9. **print("s:", s)**

Output:

```
p is True  
q is False  
r: 4  
s: 7
```

Special literals

Python contains one special literal, that is, '**None**'. This special literal is used for defining a null variable. If 'None' is compared with anything else other than a 'None', it will return false.

Example:

1. word = None
2. **print(word)**

Output:

```
None
```

7) Explain Python Functions?

A function is a section of the program or a block of code that is written once and can be executed whenever required in the program. A function is a block of self-contained statements which has a valid name, parameters list, and body. Functions make programming more functional and modular to perform modular tasks. Python provides several built-in functions to complete tasks and also allows a user to create new functions as well.

There are three types of functions:

- **Built-In Functions:** copy(), len(), count() are the some built-in functions.
- **User-defined Functions:** Functions which are defined by a user known as user-defined functions.
- **Anonymous functions:** These functions are also known as lambda functions because they are not declared with the standard def keyword.

Example: A general syntax of user defined function is given below.

1. **def** function_name(parameters list):
 2. **#--- statements---**
 3. **return** a_value
-

8) What is zip() function in Python?

Python **zip()** function returns a zip object, which maps a similar index of multiple containers. It takes an iterable, convert into iterator and aggregates the elements based on iterables passed. It returns an iterator of tuples.

Signature

1. **zip(iterator1, iterator2, iterator3 ...)**

Parameters

iterator1, iterator2, iterator3: These are iterator objects that are joined together.

Return

It returns an iterator from two or more iterators.

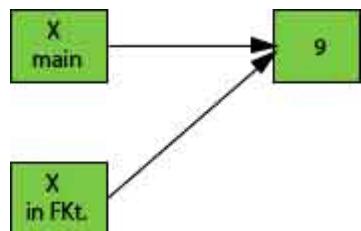
Note: If the given lists are of different lengths, zip stops generating tuples when the first list ends. It means two lists are having 3, and 5 lengths will create a 3-tuple.

9) What is Python's parameter passing mechanism?

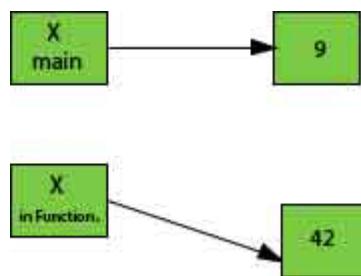
There are two parameters passing mechanism in Python:

- o Pass by references
- o Pass by value

By default, all the parameters (arguments) are passed "by reference" to the functions. Thus, if you change the value of the parameter within a function, the change is reflected in the calling function as well. It indicates the original variable. For example, if a variable is declared as `a = 10`, and passed to a function where its value is modified to `a = 20`. Both the variables denote to the same value.



The pass by value is that whenever we pass the arguments to the function only values pass to the function, no reference passes to the function. It makes it immutable that means not changeable. Both variables hold the different values, and original value persists even after modifying in the function.



Python has a default argument concept which helps to call a method using an arbitrary number of arguments.

10) How to overload constructors or methods in Python?

Python's constructor: `_init__()` is the first method of a class. Whenever we try to instantiate an object `_init__()` is automatically invoked by python to initialize members of an object. We can't overload constructors or methods in Python. It shows an error if we try to overload.

Example:

```
1. class student:  
2.     def __init__(self, name):  
3.         self.name = name  
4.     def __init__(self, name, email):  
5.         self.name = name  
6.         self.email = email  
7.  
8. # This line will generate an error  
9. st = student("rahul")  
10.  
11.# This line will call the second constructor  
12.st = student("rahul", "rahul@gmail.com")  
13.print("Name: ", st.name)  
14.print("Email id: ", st.email)
```

Output:

```
Name: rahul  
Email id: rahul@gmail.com
```

11) What is the difference between remove() function and del statement?

The user can use the `remove()` function to delete a specific object in the list.

Example:

```
1. list_1 = [ 3, 5, 7, 3, 9, 3 ]  
2. print(list_1)  
3. list_1.remove(3)  
4. print("After removal: ", list_1)
```

Output:

```
[3, 5, 7, 3, 9, 3]
```

```
After removal: [5, 7, 3, 9, 3]
```

If you want to delete an object at a specific location (index) in the list, you can either use **del** or **pop**.

Example:

1. `list_1 = [3, 5, 7, 3, 9, 3]`
2. `print(list_1)`
3. `del list_1[2]`
4. `print("After deleting: ", list_1)`

Output:

```
[3, 5, 7, 3, 9, 3]
```

```
After deleting: [3, 5, 3, 9, 3]
```

Note: You don't need to import any extra module to use these functions for removing an element from the list.

We cannot use these methods with a tuple because the tuple is different from the list.

12) What is swapcase() function in the Python?

It is a string's function which converts all uppercase characters into lowercase and vice versa. It is used to alter the existing case of the string. This method creates a copy of the string which contains all the characters in the swap case. If the string is in lowercase, it generates a small case string and vice versa. It automatically ignores all the non-alphabetic characters. See an example below.

Example:

1. `string = "IT IS IN LOWERCASE."`
2. `print(string.swapcase())`
- 3.
4. `string = "it is in uppercase."`
5. `print(string.swapcase())`

Output:

it is in lowercase.

IT IS IN UPPERCASE.

13) How to remove whitespaces from a string in Python?

To remove the whitespaces and trailing spaces from the string, Python provides `strip([str])` built-in function. This function returns a copy of the string after removing whitespaces if present. Otherwise returns original string.

Example:

1. `string = " javatpoint "`
2. `string2 = " javatpoint "`
3. `string3 = " javatpoint"`
4. `print(string)`
5. `print(string2)`
6. `print(string3)`
7. `print("After stripping all have placed in a sequence:")`
8. `print(string.strip())`
9. `print(string2.strip())`
10. `print(string3.strip())`

Output:

```
javatpoint
javatpoint
javatpoint
```

After stripping all have placed in a sequence:

```
Javatpoint
javatpoint
javatpoint
```

14) How to remove leading whitespaces from a string in the Python?

To remove leading characters from a string, we can use `lstrip()` function. It is Python string function which takes an optional char type parameter. If a parameter is provided, it removes the character. Otherwise, it removes all the leading spaces from the string.

Example:

1. string = " javatpoint "
2. string2 = " javatpoint "
3. **print**(string)
4. **print**(string2)
5. **print**("After stripping all leading whitespaces:")
6. **print**(string.lstrip())
7. **print**(string2.lstrip())

Output:

```
javatpoint
javatpoint
After stripping all leading whitespaces:
javatpoint
javatpoint
```

		j	a	v	a	t	p	o	i	n	t	
--	--	---	---	---	---	---	---	---	---	---	---	--

After stripping, all the whitespaces are removed, and now the string looks like the below:

j	a	v	a	t	p	o	i	n	t	
---	---	---	---	---	---	---	---	---	---	--

15) Why do we use join() function in Python?

The join() is defined as a string method which returns a string value. It is concatenated with the elements of an iterable. It provides a flexible way to concatenate the strings. See an example below.

Example:

1. str = "Rohan"
2. str2 = "ab"
3. **# Calling function**
4. str2 = str.join(str2)
5. **# Displaying result**
6. **print**(str2)

Output:

aRohanb

16) Give an example of shuffle() method?

This method shuffles the given string or an array. It randomizes the items in the array. This method is present in the random module. So, we need to import it and then we can call the function. It shuffles elements each time when the function calls and produces different output.

Example:

1. `# import the random module`
2. `import random`
3. `# declare a list`
4. `sample_list1 = ['Z', 'Y', 'X', 'W', 'V', 'U']`
5. `print("Original LIST1: ")`
6. `print(sample_list1)`
7. `# first shuffle`
8. `random.shuffle(sample_list1)`
9. `print("\nAfter the first shuffle of LIST1: ")`
10. `print(sample_list1)`
11. `# second shuffle`
12. `random.shuffle(sample_list1)`
13. `print("\nAfter the second shuffle of LIST1: ")`
14. `print(sample_list1)`

Output:

Original LIST1:
['Z', 'Y', 'X', 'W', 'V', 'U']

After the first shuffle of LIST1:
['V', 'U', 'W', 'X', 'Y', 'Z']

After the second shuffle of LIST1:
['Z', 'Y', 'X', 'U', 'V', 'W']

17) What is the use of break statement?

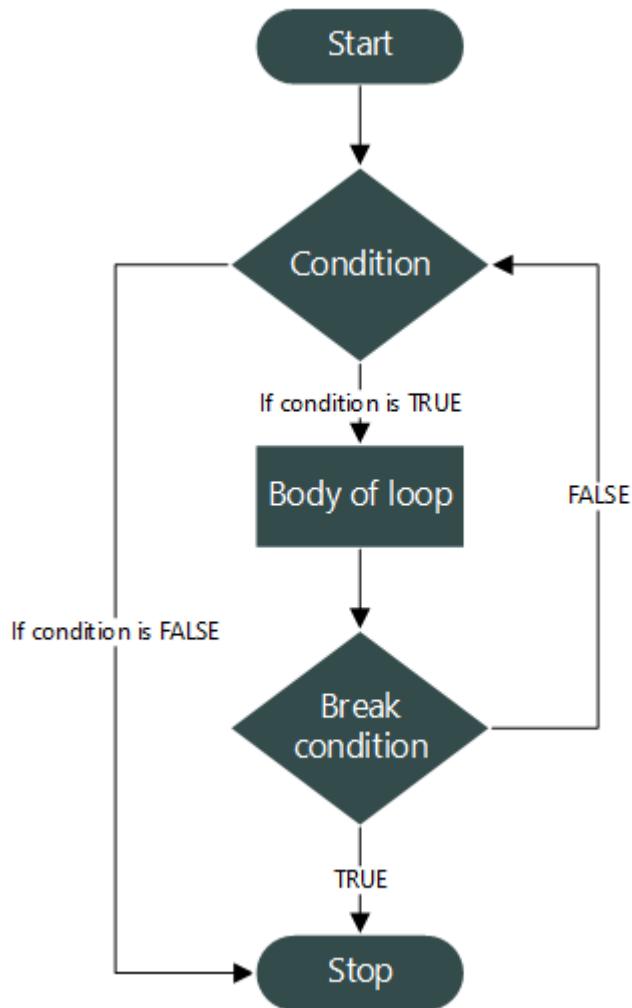
The break statement is used to terminate the execution of the current loop. Break always breaks the current execution and transfer control to outside the current block. If the block is in a loop, it exits from the loop, and if the break is in a nested loop, it exits from the innermost loop.

Example:

```
1. list_1 = ['X', 'Y', 'Z']
2. list_2 = [11, 22, 33]
3. for i in list_1:
4.     for j in list_2:
5.         print(i, j)
6.         if i == 'Y' and j == 33:
7.             print('BREAK')
8.             break
9.     else:
10.    continue
11. break
```

Output:

```
2
X 11
X 22
X 33
Y 11
Y 22
Y 33
BREAK
```



Python Break statement flowchart.

18) What is tuple in Python?

A tuple is a built-in data collection type. It allows us to store values in a sequence. It is immutable, so no change is reflected in the original data. It uses () brackets rather than [] square brackets to create a tuple. We cannot remove any element but can find in the tuple. We can use indexing to get elements. It also allows traversing elements in reverse order by using negative indexing. Tuple supports various methods like max(), sum(), sorted(), Len() etc.

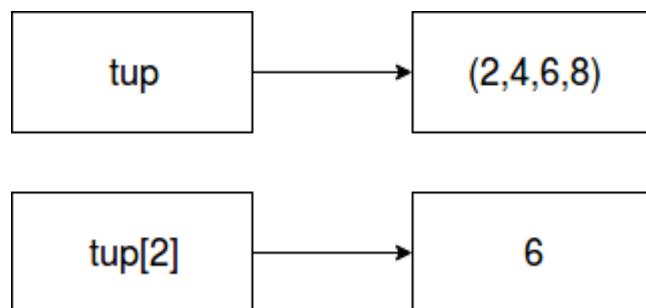
To create a tuple, we can declare it as below.

Example:

1. **# Declaring tuple**
2. **tup = (2,4,6,8)**
3. **# Displaying value**
4. **print(tup)**
- 5.
6. **# Displaying Single value**
7. **print(tup[2])**

Output:

```
(2, 4, 6, 8)  
6
```



It is immutable. So updating tuple will lead to an error.

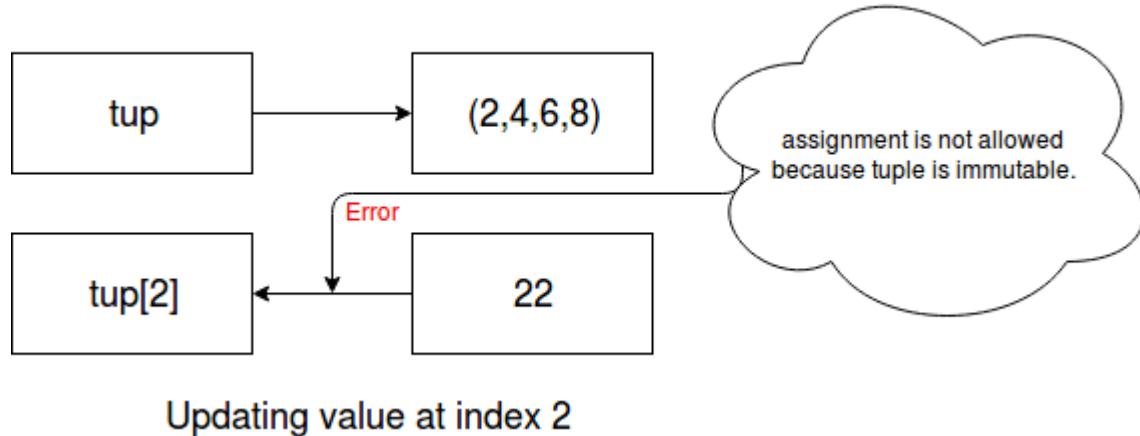
Example:

1. **# Declaring tuple**
2. **tup = (2,4,6,8)**
3. **# Displaying value**
4. **print(tup)**
- 5.
6. **# Displaying Single value**
7. **print(tup[2])**
- 8.
9. **# Updating by assigning new value**
10. **tup[2]=22**
11. **# Displaying Single value**
12. **print(tup[2])**

Output:

```
tup[2]=22
```

```
TypeError: 'tuple' object does not support item assignment  
(2, 4, 6, 8)
```



19) Which are the file related libraries/modules in Python?

The Python provides libraries/modules that enable you to manipulate text files and binary files on the file system. It helps to create files, update their contents, copy, and delete files. The libraries are os, os.path, and shutil.

Here, os and os.path - modules include a function for accessing the filesystem while shutil - module enables you to copy and delete the files.

20) What are the different file processing modes supported by Python?

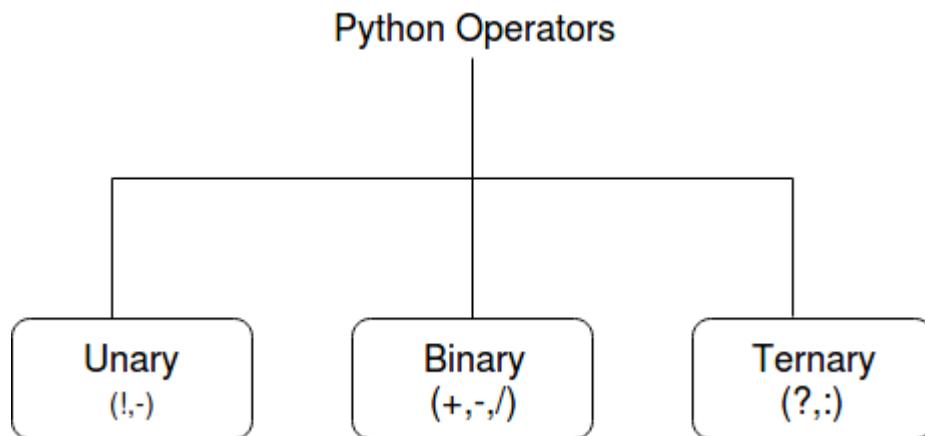
Python provides **four** modes to open files. The read-only (r), write-only (w), read-write (rw) and append mode (a). 'r' is used to open a file in read-only mode, 'w' is used to open a file in write-only mode, 'rw' is used to open in reading and write mode, 'a' is used to open a file in append mode. If the mode is not specified, by default file opens in read-only mode.

- Read-only mode (r): Open a file for reading. It is the default mode.
- Write-only mode (w): Open a file for writing. If the file contains data, data would be lost. Other a new file is created.
- Read-Write mode (rw): Open a file for reading, write mode. It means updating mode.

- o Append mode (a): Open for writing, append to the end of the file, if the file exists.
-

21) What is an operator in Python?

An operator is a particular symbol which is used on some values and produces an output as a result. An operator works on operands. Operands are numeric literals or variables which hold some values. Operators can be unary, binary or ternary. An operator which requires a single operand known as a **unary operator**, which require two operands known as a **binary operator** and which require three operands is called **ternary operator**.



Example:

1. # Unary Operator
2. A = 12
3. B = -(A)
4. **print** (B)
5. # Binary Operator
6. A = 12
7. B = 13
8. **print** (A + B)
9. **print** (B * A)
10. #Ternary Operator
11. A = 12
12. B = 13
13. min = A **if** A < B **else** B
- 14.

15. `print(min)`

Output:

```
# Unary Operator  
-12  
# Binary Operator  
25  
156  
# Ternary Operator  
12
```

22) What are the different types of operators in Python?

Python uses a rich set of operators to perform a variety of operations. Some individual operators like membership and identity operators are not so familiar but allow to perform operations.

- Arithmetic Operators
- Relational Operators
- Assignment Operators
- Logical Operators
- Membership Operators
- Identity Operators
- Bitwise Operators

python Operators

Arithmetic Operator

Relational Operator

Assignment Operator

Logical Operator

Bitwise Operator

Membership Operator

Identity Operator

Arithmetic operators perform basic arithmetic operations. For example "+" is used to add and "?" is used for subtraction.

Example:

1. # Adding two values
2. `print(12+23)`
3. # Subtracting two values
4. `print(12-23)`
5. # Multiplying two values
6. `print(12*23)`
7. # Dividing two values
8. `print(12/23)`

Output:

```
35  
-11  
276  
0.5217391304347826
```

Relational Operators are used to comparing the values. These operators test the conditions and then returns a boolean value either True or False.

```
# Examples of Relational Operators
```

Example:

1. a, b = 10, 12
2. `print(a==b) # False`
3. `print(a<b) # True`
4. `print(a<=b) # True`
5. `print(a!=b) # True`

Output:

```
False  
True  
True  
True
```

Assignment operators are used to assigning values to the variables. See the examples below.

Example:

1. `# Examples of Assignment operators`
2. `a=12`
3. `print(a) # 12`
4. `a += 2`
5. `print(a) # 14`
6. `a -= 2`
7. `print(a) # 12`
8. `a *=2`
9. `print(a) # 24`
10. `a **=2`

```
11.print(a) # 576
```

Output:

```
12  
14  
12  
24  
576
```

Logical operators are used to performing logical operations like And, Or, and Not. See the example below.

Example:

1. `# Logical operator examples`
2. `a = True`
3. `b = False`
4. `print(a and b) # False`
5. `print(a or b) # True`
6. `print(not b) # True`

Output:

```
False  
True  
True
```

Membership operators are used to checking whether an element is a member of the sequence (list, dictionary, tuples) or not. Python uses two membership operators in and not in operators to check element presence. See an example.

Example:

1. `# Membership operators examples`
2. `list = [2,4,6,7,3,4]`
3. `print(5 in list) # False`
4. `cities = ("india","delhi")`
5. `print("tokyo" not in cities) #True`

Output:

False

True

Identity Operators (is and is not) both are used to check two values or variable which are located on the same part of the memory. Two variables that are equal does not imply that they are identical. See the following examples.

Example:

1. `# Identity operator example`
2. `a = 10`
3. `b = 12`
4. `print(a is b) # False`
5. `print(a is not b) # True`

Output:

False

True

Bitwise Operators are used to performing operations over the bits. The binary operators (&, |, OR) work on bits. See the example below.

Example:

1. `# Identity operator example`
2. `a = 10`
3. `b = 12`
4. `print(a & b) # 8`
5. `print(a | b) # 14`
6. `print(a ^ b) # 6`
7. `print(~a) # -11`

Output:

8

14

6

-11

23) How to create a Unicode string in Python?

In Python 3, the old Unicode type has replaced by "str" type, and the string is treated as Unicode by default. We can make a string in Unicode by using `str.encode("utf-8")` function.

Example:

1. `unicode_1 = ("\\u0123", "\\u2665", "\\U0001f638", "\\u265E", "\\u265F", "\\u2168")`
2. `print(unicode_1)`

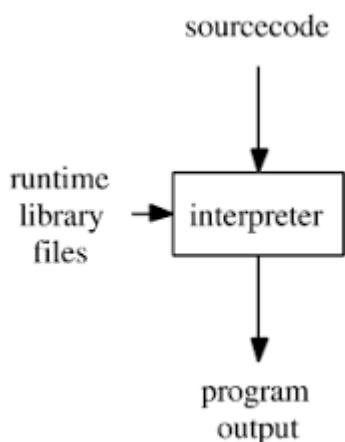
Output:

```
unicode_1: ('g', '♥', '😺', '🐴', ' Milf', 'IX')
```

24) Is Python interpreted language?

Python is an interpreted language. The Python language program runs directly from the source code. It converts the source code into an intermediate language code, which is again translated into machine language that has to be executed.

Unlike Java or C, Python does not require compilation before execution.



25) How is memory managed in Python?

Memory is managed in Python in the following ways:

- 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.
 - 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.
 - 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.
-

26) What is the Python decorator?

Decorators are very powerful and a useful tool in Python that allows the programmers to add functionality to an existing code. This is also called metaprogramming because a part of the program tries to modify another part of the program at compile time. It allows the user to wrap another function to extend the behaviour of the wrapped function, without permanently modifying it.

Example:

```
1. def function_is_called():
2.     def function_is_returned():
3.         print("JavaTpoint")
4.         return function_is_returned
5.     new_1 = function_is_called()
6.     # Outputs "JavaTpoint"
7.     new_1()
```

Output:

JavaTpoint

Functions vs. Decorators

A function is a block of code that performs a specific task whereas a decorator is a function that modifies other functions.

27) What are the rules for a local and global variable in Python?

Global Variables:

- Variables declared outside a function or in global space are called global variables.
- If a variable is ever assigned a new value inside the function, the variable is implicitly local, and we need to declare it as 'global' explicitly. To make a variable globally, we need to declare it by using global keyword.
- Global variables are accessible anywhere in the program, and any function can access and modify its value.

Example:

1. A = "JavaTpoint"
2. **def** my_function():
3. **print**(A)
4. my_function()

Output:

```
JavaTpoint
```

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

Example:

1. **def** my_function2():
2. K = "JavaTpoint Local"
3. **print**(K)
4. my_function2()

Output:

28) What is the namespace in Python?

The namespace is a fundamental idea to structure and organize the code that is more useful in large projects. However, it could be a bit difficult concept to grasp if you're new to programming. Hence, we tried to make namespaces just a little easier to understand.

A namespace is defined as a simple system to control the names in a program. It ensures that names are unique and won't lead to any conflict.

Also, Python implements namespaces in the form of dictionaries and maintains name-to-object mapping where names act as keys and the objects as values.

29) What are iterators in Python?

In Python, iterators are used to iterate a group of elements, containers like a list. Iterators are the collection of items, and it can be a list, tuple, or a dictionary. Python iterator implements `__itr__` and `next()` method to iterate the stored elements. In Python, we generally use loops to iterate over the collections (list, tuple).

In simple words: Iterators are objects which can be traversed though or iterated upon.

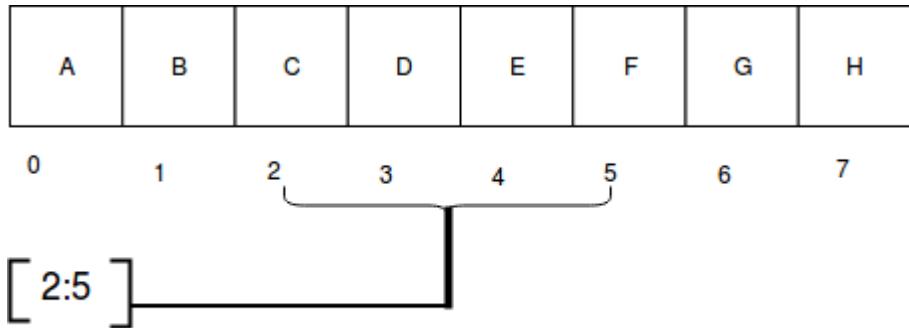
30) What is a generator in Python?

In Python, the generator is a way that specifies how to implement iterators. It is a normal function except that it yields expression in the function. It does not implements `__itr__` and `next()` method and reduce other overheads as well.

If a function contains at least a yield statement, it becomes a generator. The `yield` keyword pauses the current execution by saving its states and then resume from the same when required.

31) What is slicing in Python?

Slicing is a mechanism used to select a range of items from sequence type like list, tuple, and string. It is beneficial and easy to get elements from a range by using slice way. It requires a : (colon) which separates the start and end index of the field. All the data collection types List or tuple allows us to use slicing to fetch elements. Although we can get elements by specifying an index, we get only single element whereas using slicing we can get a group of elements.



Example:

1. Q = "JavaTpoint, Python Interview Questions!"
2. **print**(Q[2:25])

Output:

32) What is a dictionary in Python?

The Python dictionary is a built-in data type. It defines a one-to-one relationship between keys and values. Dictionaries contain a pair of keys and their corresponding values. It stores elements in key and value pairs. The keys are unique whereas values can be duplicate. The key accesses the dictionary elements.

Keys index dictionaries.

Example:

The following example contains some keys Country Hero & Cartoon. Their corresponding values are India, Modi, and Rahul respectively.

1. dict = {'Country': 'India', 'Hero': 'Modi', 'Cartoon': 'Rahul'}
2. **print** ("Country: ", dict['Country'])

3. `print ("Hero: ", dict['Hero'])`
4. `print ("Cartoon: ", dict['Cartoon'])`

Output:

```
Country: India
Hero: Modi
Cartoon: Rahul
```

33) What is Pass in Python?

Pass specifies a Python statement without operations. It is a placeholder in a compound statement. If we want to create an empty class or functions, the pass keyword helps to pass the control without error.

Example:

1. `class Student:`
 2. `pass # Passing class`
 3. `class Student:`
 4. `def info():`
 5. `pass # Passing function`
-

34) Explain docstring in Python?

The Python docstring is a string literal that occurs as the first statement in a module, function, class, or method definition. It provides a convenient way to associate the documentation.

String literals occurring immediately after a simple assignment at the top are called "attribute docstrings".

String literals occurring immediately after another docstring are called "additional docstrings".

Python uses triple quotes to create docstrings even though the string fits on one line.

Docstring phrase ends with a period(.) and can be multiple lines. It may consist of spaces and other special chars.

Example:

1. `# One-line docstrings`
 2. `def hello():`
 3. `"""A function to greet."""`
 4. `return "hello"`
-

35) What is a negative index in Python and why are they used?

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 used 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.

36) What is pickling and unpickling in Python?

The Python pickle is defined as a module which accepts any Python object and converts it into a string representation. It dumps the Python object into a file using the `dump` function; this process is called **Pickling**.

The process of retrieving the original Python objects from the stored string representation is called as **Unpickling**.

37) Which programming language is a good choice between Java and Python?

Java and Python both are object-oriented programming languages. Let's compare both on some criteria given below:

Criteria	Java	Python
----------	------	--------

Ease of use	Good	Very Good
Coding Speed	Average	Excellent
Data types	Static type	Dynamic type
Data Science and Machine learning application	Average	Very Good

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

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

Help() function: The help() function is used to display the documentation string and also facilitates us to see the help related to modules, keywords, and attributes.

Dir() function: The dir() function is used to display the defined symbols.

39) What are the differences between Python 2.x and Python 3.x?

Python 2.x is an older version of Python. Python 3.x is newer and latest version. Python 2.x is legacy now. Python 3.x is the present and future of this language.

The most visible difference between Python2 and Python3 is in print statement (function). In Python 2, it looks like print "Hello", and in Python 3, it is print ("Hello").

String in Python2 is ASCII implicitly, and in Python3 it is Unicode.

The xrange() method has removed from Python 3 version. A new keyword as is introduced in Error handling.

40) How Python does Compile-time and Run-time code checking?

In Python, some amount of coding is done at compile time, but most of the checking such as type, name, etc. are postponed until code execution. Consequently, if the Python code references a user-defined function that does not

exist, the code will compile successfully. The Python code will fail only with an exception when the code execution path does not exist.

41) What is the shortest method to open a text file and display its content?

The shortest way to open a text file is by using "with" command in the following manner:

Example:

1. `with open("FILE NAME", "r") as fp:`
2. `fileData = fp.read()`
3. `# To print the contents of the file`
4. `print(fileData)`

Output:

"The data of the file will be printed."

42) What is the usage of enumerate () function in Python?

The enumerate() function is used to iterate through the sequence and retrieve the index position and its corresponding value at the same time.

Example:

1. `list_1 = ["A","B","C"]`
2. `s_1 = "Javatpoint"`
3. `# creating enumerate objects`
4. `object_1 = enumerate(list_1)`
5. `object_2 = enumerate(s_1)`
- 6.
7. `print ("Return type:",type(object_1))`
8. `print (list(enumerate(list_1)))`
9. `print (list(enumerate(s_1)))`

Output:

Return type:

```
[(0, 'A'), (1, 'B'), (2, 'C')]  
[(0, 'J'), (1, 'a'), (2, 'v'), (3, 'a'), (4, 't'), (5, 'p'), (6, 'o'), (7, 'i'), (8, 'n'), (9, 't')]
```

43) Give the output of this example: A[3] if A=[1,4,6,7,9,66,4,94].

Since indexing starts from zero, an element present at 3rd index is 7. So, the output is 7.

44) What is type conversion in Python?

Type conversion refers to the conversion of one data type into 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,img) - This function converts real numbers to complex(real,img) number.

45) How to send an email in Python Language?

To send an email, Python provides smtplib and email modules. Import these modules into the created mail script and send mail by authenticating a user.

It has a method SMTP(smtp-server, port). It requires two parameters to establish SMTP connection.

A simple example to send an email is given below.

Example:

```
1. import smtplib  
2. # Calling SMTP  
3. s = smtplib.SMTP('smtp.gmail.com', 587)  
4. # TLS for network security  
5. s.starttls()  
6. # User email Authentication  
7. s.login("sender@email_id", "sender_email_id_password")  
8. # Message to be sent  
9. message = "Message_sender_need_to_send"  
10.# Sending the mail  
11.s.sendmail("sender@email_id ", "receiver@email_id", message)
```

46) What is the difference between Python Arrays and lists?

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:

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

Output:

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

47) What is lambda function in Python?

The anonymous function in python is a function that is defined without a name. The normal functions are defined using a keyword "def", whereas, the anonymous functions are defined using the lambda function. **The anonymous functions are also called as lambda functions.**

48) Why do lambda forms in Python not have the statements?

Lambda forms in Python does not have the statement because it is used to make the new function object and return them in runtime.

49) What are functions in Python?

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:

1. `def New_func():`
2. `print ("Hi, Welcome to JavaTpoint")`
3. `New_func() #calling the function`

Output:

```
Hi, Welcome to JavaTpoint
```

50) What is `__init__`?

The `__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.

Example:

1. `class Employee_1:`
2. `def __init__(self, name, age,salary):`
3. `self.name = name`
4. `self.age = age`
5. `self.salary = 20000`

6. E_1 = Employee_1("pqr", 20, 25000)
7. # E1 is the instance of class Employee.
8. #__init__ allocates memory for E1.
9. **print**(E_1.name)
10. **print**(E_1.age)
11. **print**(E_1.salary)

Output:

```
pqr  
20  
25000
```

51) What is self in Python?

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.

52) How can you generate random numbers in Python?

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

1. **import** random
2. 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:

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.

uniform(a, b): it chooses a floating point number that is defined in the range of [a,b). It returns the floating point number

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.

The Random class that is used and instantiated creates independent multiple random number generators.

53) What is PYTHONPATH?

PYTHONPATH 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.

54) What are python modules? Name some commonly used built-in modules in Python?

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
-

55) What is the difference between range & xrange?

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 xrange 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.

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

- 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.
- 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.
- NumPy is not just more efficient; it is also more convenient. We get a lot of vector and matrix operations for free, which sometimes allow one to avoid unnecessary work. And they are also efficiently implemented.
- NumPy array is faster and we get a lot built in with NumPy, FFTs, convolutions, fast searching, basic statistics, linear algebra, histograms, etc.

57) Mention what the Django templates consist of.

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.

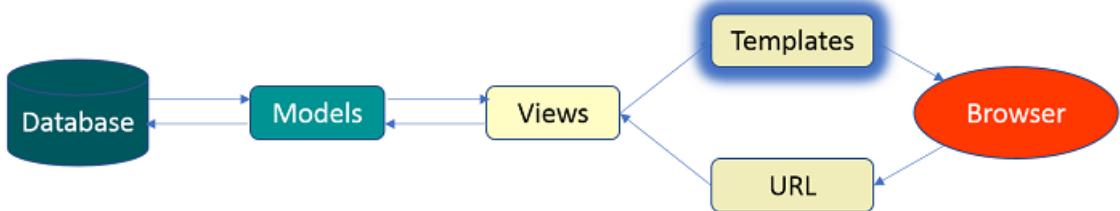


Figure: Python Interview Questions – Django Template

58) Explain the use of session in Django framework?

Django provides a session that lets the user 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 good from a security perspective.

Interview based Multiple Choice Questions on Python

- 1) Which of the following statements is/are TRUE in respect of the Python programming language?

Statement 1: Python is an interpreted, high-level, general-purpose programming language.

Statement 2: Python provides high-level data structures along with dynamic binding and typing for Rapid Application Development and deployment.

Statement 3: Python is a Statically typed Programming language.

Options:

- a. Only Statement 1
- b. Statement 1 and 2
- c. Statement 1 and 3
- d. All Statements are Correct

Show Answer Workspace

2) What is the full form of PEP?

Options:

- a. Python Enhancement Proposal
- b. Python Enchantment Proposal
- c. Programming Enhancement Proposition
- d. Python Enrichment Program

Show Answer Workspace

3) Which of the following statements is/are NOT correct regarding Memory Management in Python?

Statement 1: Python Memory Manager handles the management regarding memory in Python

Statement 2: Python uses CMS (Concurrent Mark Sweep) approach as its Garbage Collection technique.

Statement 3: Python offers a core garbage collection to recycle the vacant memory for the private heap space.

Options:

- a. Only Statement 3
- b. Statement 1 and 3

- c. Statement 2 and 3
- d. Only Statement 2

Show Answer Workspace

4) Which of the following statements is/are NOT correct in respect to Python namespaces?

Statement 1: Python implements the namespace in the form of Array.

Statement 2: Python namespaces are classified into three types - local, global and built-in.

Statement 3: A Python namespace ensures that the names of the objects in a program are unique and can be utilized, deprived of any inconsistency.

Options:

- a. Only Statement 1
- b. Only Statement 3
- c. Statement 1 and 2
- d. Statement 1 and 3

Show Answer Workspace

5) Which of the following is invalid in terms of Variable Names?

Options:

- a. `_mystr = "Hello World!"`
- b. `__mystr = "Hello World!"`
- c. `__mystr__ = "Hello World!"`
- d. None of the mentioned

Show Answer Workspace

6) In respect to the scope in Python, which of the following statements is/are TRUE?

Statement 1: A variable created within a function belongs to the local scope and can be used outside that function.

Statement 2: A local scope is referred to the object present through the execution of code since its inception.

Statement 3: A local scope is referred to the local object present in the current function.

Options:

- a. Only Statement 2
- b. Statement 1 and 3
- c. Only Statement 3
- d. All Statements are True

Show Answer Workspace

7) What will the following snippet of code print?

Code:

```
1. # assigning a variable
2. myint = 10
3.
4. # defining a function
5. def myfunction():
6.     # reassigning a variable
7.     myint = 20
8.
9. # calling the function
10.myfunction()
11.
12.# printing the value of the variable
13.print(myint)
```

Options:

- a. 10
- b. 20
- c. 0
- d. Traceback Error

Show Answer Workspace

8) What will the following snippet of code yield?

Code:

```
1. # assigning a variable
2. myint = 21
3.
4. # using if False statement
5. if False:
6.     # reassigning a variable
7.     myint = 34
8.
9. # defining a function
10.def myfunction():
11.    if True:
12.        # reassigning a variable
13.        myint = 65
14.
15.# calling the function
16.myfunction()
17.
18.# printing the value of the variable
19.print(myint)
```

Options:

- a. 65
- b. Traceback Error
- c. 21
- d. 34

Show Answer Workspace

9) Among the following statements based on the difference between lists and tuples, which one statement is TRUE?

Statement 1: List is a sequence data structure, whereas Tuple is not.

Statement 2: Lists are immutable; however, Tuples are mutable.

Statement 3: Tuple is a sequence data structure, whereas List is not.

Statement 4: Tuples are immutable; however, Lists are mutable.

Options:

- a. Statement 1
- b. Statement 4
- c. Statement 2
- d. Statement 3

Show Answer Workspace

10) What will be the output of the following snippet of code?

Code:

1. # defining a list
2. `my_list = [7, 9, 8, 2, 5, 0, 1, 3, 6]`
- 3.
4. # using the pop() function
5. `my_list.pop(2)`
- 6.
7. # printing the final list
8. `print(my_list)`

Options:

- 1. `[7, 9, 2, 5, 0, 1, 3, 6]`
- 2. `[7, 9, 8, 2, 5, 0, 3, 6]`
- 3. `[7, 8, 2, 5, 0, 1, 3, 6]`
- 4. `[7, 9, 8, 2, 5, 0, 1, 6]`

1. What is Python?

Python is a high-level and object-oriented programming language with unified semantics designed primarily for developing apps and the web. It is the core language in the field of Rapid Application Development (RAD) as it offers options such as dynamic binding and dynamic typing.

2. What are the benefits of Python?

The benefits of Python are as follows:

- **Speed and Productivity:** Utilizing the productivity and speed of Python will enhance the process control capabilities and possesses strong integration.
- **Extensive Support for Libraries:** Python provides a large standard library that includes areas such as operating system interfaces, web service tools, internet protocols, and string protocols. Most of the programming tasks are already been scripted in the standard library which reduces effort and time.
- **User-friendly Data Structures:** Python has an in-built dictionary of data structures that are used to build fast user-friendly data structures.
- **Existence of Third-Party Modules:** The presence of third-party modules in the Python Package Index (PyPI) will make Python capable to interact with other platforms and languages.
- **Easy Learning:** Python provides excellent readability and simple syntaxes to make it easy for beginners to learn.

3. What are the key features of Python?

The following are the significant features of Python, and they are:

- **Interpreted Language:** Python is an interpreted language that is used to execute the code line by line at a time. This makes debugging easy.
- **Highly Portable:** Python can run on different platforms such as Unix, Macintosh, Linux, Windows, and so on. So, we can say that it is a highly portable language.
- **Extensible:** It ensures that the Python code can be compiled on various other languages such as C, C++, and so on.
- **GUI programming Support:** It implies that Python provides support to develop graphical user interfaces

4. What type of language is Python? Programming or Scripting?

Python is suitable for scripting, but in general, it is considered a general-purpose programming language.

Top 10 Programming Languages that you need to watch out to boost your career in 2022

5. What are the applications of Python?

The applications of Python are as follows:

- GUI-based desktop applications
- Image processing applications
- Business and Enterprise applications
- Prototyping
- Web and web framework applications

6. What is the difference between a list and a tuple in Python?

The difference between a tuple and list is as follows:

List	Tuple
The list is mutable (can be changed)	A tuple is immutable (remains constant)
These lists performance is slower	Tuple performance is faster when compared to lists
Syntax: list_1 = [20, 'Mindmajix', 30]	Syntax: tup_1 = (20, 'Mindmajix', 30)

7. What are the global and local variables in Python?

Global Variables in Python: The variables that are declared outside the function are called global variables. These variables can be accessed or invoked by any function in the program.

Example:

```
def v():
    print g
g = "welcome to mindmajix"
v()
```

Output:

```
Welcome to mindmajix
```

Local Variables in Python: The variables that are declared inside a function are called local variables. These types of variables can be accessed only inside the function.

8. Define PYTHON PATH?

PYTHONPATH is an environmental variable that is used when we import a module. Suppose at any time we import a module, PYTHONPATH is used to check the presence of the modules that are imported in different directories. Loading of the module will be determined by interpreters.

9. What are the two major loop statements?

for and while

10. What do you understand by the term PEP 8?

PEP 8 is the Python latest coding convention and it is abbreviated as Python Enhancement Proposal. It is all about how to format your Python code for maximum readability.

11. How memory management is done in Python?

- In Python memory management is done using private heap space. The private heap is the storage area for all the data structures and objects. The interpreter has access to the private heap and the programmer cannot access this private heap.
- The storage allocation for the data structures and objects in Python is done by the memory manager. The access for some tools is provided by core API for programmers to code.
- The built-in garbage collector in Python is used to recycle all the unused memory so that it can be available for heap storage area.

12. Java vs Python

The major difference between Java and Python are as follows:

Function	Java	Python
----------	------	--------

Coding

In Java, we need to write long code to print something.

In Python coding is smaller when compared.

Syntax

In Java we need to put a semicolon at the end of the statement and also code must be placed in curly braces.

Whereas, in Python indentation is mandatory as it improves readability of the code.

Dynamic

In Java, we need to declare the type for each variable

In this case, codes are dynamically typed and also known as duck typing.

Easy to use

Java is not easy to use because of its larger coding

In Python, it is very easy to code and perform very well.

Databases

Java Database Connectivity (JDBC) is more popular and used most commonly.

In Python database access layers are weaker when compared to Java.

13. Define modules in Python?

The module is defined as a file that includes a set of various functions and Python statements that we want to add to our application.

Example of creating a module:

In order to create a module first, we need to save the code that we want in a file with .py extension.

Save the module with module.py

```
def wishes(name):  
    Print("Hi, " + name)
```

14. What are the built-in types available in Python?

The built-in types in Python are as follows:

- Integer

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

15. What are Python Decorators?

Decorator is the most useful tool in Python as it allows programmers to alter the changes in the behavior of class or function.

An example for Python Decorator is:

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

16. How do we find bugs and statistical problems in Python?

We can detect bugs in python source code using a static analysis tool named PyChecker. Moreover, there is another tool called PyLint that checks whether the Python modules meet their coding standards or not.

17. What is the difference between .py and .pyc files?

.py files are Python source files. .pyc files are the compiled bytecode files that are generated by the Python compiler

18. How do you invoke the Python interpreter for interactive use?

By using python or pythonx. y we can invoke a Python interpreter. where x.y is the version of the Python interpreter.

19. Define String in Python?

String in Python is formed using a sequence of characters. Value once assigned to a string cannot be modified because they are immutable objects. String literals in Python can be declared using double quotes or single quotes.

Example:

```
print("Hi")
print('Hi')
```

20. What do you understand by the term namespace in Python?

A namespace in Python can be defined as a system that is designed to provide a unique name for every object in python. Types of namespaces that are present in Python are:

- Local namespace
- Global namespace
- Built-in namespace

Scope of an object in Python:

Scope refers to the availability and accessibility of an object in the coding region.

21. How do you create a Python function?

Functions are defined using the def statement.

An example might be def foo(bar):

22. Define iterators in Python?

In Python, an iterator can be defined as an object that can be iterated or traversed upon. In another way, it is mainly used to iterate a group of containers, elements, the same as a list.

23. How does a function return values?

Functions return values using the return statement.

24. Define slicing in Python?

Slicing is a procedure used to select a particular range of items from sequence types such as Strings, lists, and so on.

25. How can Python be an interpreted language?

As in Python the code which we write is not machine-level code before runtime so, this is the reason why Python is called an interpreted language.

26. What happens when a function doesn't have a return statement? Is this valid?

Yes, this is valid. The function will then return a None object. The end of a function is defined by the block of code that is executed (i.e., the indenting) not by any explicit keyword.

27. Define package in Python?

In Python packages are defined as the collection of different modules.

28. How can we make a Python script executable on Unix?

In order to make a Python script executable on Unix, we need to perform two things. They are:

Script file mode must be executable and

The first line should always begin with #.

29. Which command is used to delete files in Python?

OS.unlink(filename) or OS.remove(filename) are the commands used to delete files in Python Programming.

Example:

```
import OS  
OS.remove("abc.txt")
```

30. Define pickling and unpickling in Python?

Pickling in Python: The process in which the pickle module accepts various Python objects and converts them into a string representation and dumps the file accordingly using the dump function is called pickling.

Unpickling in Python: The process of retrieving actual Python objects from the stored string representation is called unpickling.

31. Explain the difference between local and global namespaces?

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

32. What is a boolean in Python?

Boolean is one of the built-in data types in Python, it mainly contains two values, which are true and false.

Python `bool()` is the method used to convert a value to a boolean value.

Syntax for `bool()` method: `bool([a])`

33. What are Python String formats and Python String replacements?

Python String Format: The `String format()` method in Python is mainly used to format the given string into an accurate output or result.

Syntax for String `format()` method:

```
template.format(p0, p1, ..., k0=v0, k1=v1, ...)
```

Python String Replace: This method is mainly used to return a copy of the string in which all the occurrence of the substring is replaced by another substring.

Syntax for String `replace()` method:

```
str.replace(old, new [, count])
```

34. Name some of the built-in modules in Python?

The built-in modules in Python are:

- `sys` module
- `OS` module
- `random` module
- `collection` module
- `JSON`
- `Math` module

35. What are the functions in Python?

In Python, functions are defined as a block of code that is executable only when it is called. The `def` keyword is used to define a function in Python.

Example:

```
def Func():  
    print("Hello, Welcome toMindmajix")  
Func(); #calling the function
```

Output: Hello, Welcome to Mindmajix

36. What are Dict and List comprehensions in Python?

These are mostly used as syntax constructions to ease the creation of lists and dictionaries based on existing iterable.

37. Define the term lambda?

Lambda is the small anonymous function in Python that is often used as an inline function.

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

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

39. Define self in Python?

In Python self is defined as an object or an instance of a class. This self is explicitly considered as the first parameter in Python. Moreover, we can also access all the methods and attributes of the classes in Python programming using self keyword.

In the case of the init method, self refers to the newer creation of the object. Whereas in the case of other methods self refers to the object whose method was called.

40. What is __init__?

The __init__ is a special type of method in Python that is called automatically when the memory is allocated for a new object. The main role of __init__ is to initialize the values of instance members for objects.

Example:

```
class Student:  
    def __init__(self, name, age, marks):  
        self.name = name  
        self.age = age  
        self.marks = 950  
    S1 = Student("ABC", 22, 950)  
    # S1 is the instance of class Student.
```

```
# __init__ allocates memory for S1.  
print(S1.name)  
print(S1.age)  
print(S1.marks)
```

Output:

```
ABC  
22  
950
```

41. Define generators in Python?

The way of implementing an effective representation of iterators is known as generators. It is only the normal function that yields expression in the function.

42. Define docstring in Python?

The docstring in Python is also called a documentation string, it provides a way to document the Python classes, functions, and modules.

43. How do we convert the string to lowercase?

the lower() function is used to convert string to lowercase.

Example:

```
str = 'XYZ'  
print(str.lower())
```

Output:

```
xyz
```

44. How to remove values from a Python array?

Ans: The elements can be removed from a Python array using the remove() or pop() function. The difference between pop() and remove() will be explained in the below example.

Example:

```
x = arr.array('d', [ 1.0, 2.2, 3.4, 4.8, 5.2, 6.6, 7.3])
print(x.pop())
print(x.pop(3))
x.remove(1.0)
print(a)
```

Output:

```
7.3
4.8
array('d', [2.2, 3.4, 5.2, 6.6])
```

45. What is Try Block?

A block that is preceded by the try keyword is known as a try block

Syntax:

```
try{
    //statements that may cause an exception
}
```

46. Why do we use the split method in Python?

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

Example:

```
x = "Mindmajix Online Training"
print(a.split())
```

Output:

```
['Mindmajix', 'Online', 'Training']
```

47. How can we access a module written in Python from C?

We can access the module written in Python from C by using the following method.

```
Module == PyImport_ImportModule("<modulename>");
```

48. How do you copy an object in Python?

To copy objects in Python we can use methods called `copy.copy()` or `copy.deepcopy()`.

49. How do we reverse a list in Python?

By using the `list.reverse()`: we can reverse the objects of the list in Python.

Python Programming Interview Questions:

Following are the Python Programming Interview Questions with Answers

50. How can we debug a Python program?

By using the following command we can debug the Python program

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

51. Write a program to count the number of capital letters in a file?

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

52. Write a program to display the Fibonacci sequence in Python?

```
# Displaying Fibonacci sequence  
n = 10  
# first two terms  
n0 = 0  
n1 = 1  
#Count  
x = 0  
# check if the number of terms is valid
```

```
if n <= 0:  
    print("Enter positive integer")  
elif n == 1:  
    print("Numbers in Fibonacci sequence upto",n,:")  
    print(n0)  
else:  
    print("Numbers in Fibonacci sequence upto",n,:")  
    while x < n:  
        print(n0,end=',')  
        nth = n0 + n1  
        n0 = n1  
        n1 = nth  
        x += 1
```

Output:

0, 1, 1, 2, 3, 5, 8, 13, 21, 34,

53. Write a program in Python to produce a Star triangle?

The code to produce a star triangle is as follows:

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

Output:

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

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

54. Write a program to check whether the given number is prime or not?

The code to check prime number is as follows:

```
# program to check the number is prime or not  
n1 = 409  
# num1 = int(input("Enter any one number: "))  
# prime number is greater than 1  
if n1 > 1:  
    # check the following factors  
    for x in range(2,num1):  
        if (n1 % x) == 0:  
            print(n1,"is not a prime number")  
            print(x,"times",n1//x,"is",num)  
            break  
        else:  
            print(n1,"is a prime number")  
    # if input number is smaller than  
    # or equal to the value 1, then it is not prime number  
else:  
    print(n1,"is not a prime number")
```

Output:

```
409 is a prime number
```

55. Write Python code to check the given sequence is a palindrome or not?

```
# Python code to check a given sequence  
# is palindrome or not  
my_string1 = 'MOM'  
My_string1 = my_string1.casefold()  
# reverse the given string
```

```
rev_string1 = reversed(my_string1)
# check whether the string is equal to the reverse of it or not
if list(my_string1) == list(rev_string1):
    print("It is a palindrome")
else:
    print("It is not a palindrome")
```

Output:

it is a palindrome

56. Write Python code to sort a numerical dataset?

The code to sort a numerical dataset is as follows:

```
list = [ "13", "16", "1", "5" , "8"]
list = [int(x) for x in the list]
list.sort()
print(list)
```

Output:

1, 5, 8, 13, 16

57. What is the output of the following code?

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

The output of the following code is

[['a', 'b'], ['c', 'd']]

Python Developer Interview Questions

Following are Python Developer Interview Questions with Answers

58. What is the procedure to install Python on Windows and set path variables?

We need to implement the following steps to install Python on Windows, and they are:

- First, you need to install Python from <https://www.python.org/downloads/>
- After installing Python on your PC, find the place where it is located in your PC using the cmd python command.
- Then visit advanced system settings on your PC and add a new variable. Name the new variable as PYTHON_NAME then copy the path and paste it.
- Search for the path variable and select one of the values for it and click on ‘edit’.
- Finally, we need to add a semicolon at the end of the value, and if the semicolon is not present then type %PYTHON_NAME%.

59. Differentiate between SciPy and NumPy?

The difference between SciPy and NumPy is as follows:

NumPy

Numerical Python is called NumPy

It is used for performing general and efficient computations on numerical data which is saved in arrays. For example, indexing, reshaping, sorting, and so on

There are some of the linear algebraic functions present in this module but they are not fully fledged.

SciPy

Scientific Python is called SciPy

This is an entire collection of tools Python mainly used to perform operations like differentiation, integration and more.

For performing algebraic computations module contains some of the fully-fledged operations

60. How do Python arrays and lists differ from each other?

The difference between Python array and Python list is as follows:

Arrays

The array is defined as a linear structure that is used to store only homogeneous data.

Since array stores only a similar type of data so it occupies less amount of memory when compared to the list.

The length of the array is fixed at the time of designing and no more elements can be added in the middle.

Lists

The list is used to store arbitrary and heterogeneous data

List stores different types of data so it requires huge amount of memory

The length of the list is not fixed, and adding items in the middle is possible in lists.

61. Can we make multi-line comments in Python?

In python, there is no specific syntax to display multi-line comments like in other languages. In order to display multi-line comments in Python, programmers use triple-quoted (docstrings) strings. If the docstring is not used as the first statement in the present method, it will not be considered by the Python parser.

62. What is the difference between range and xrange?

Both methods are mainly used in Python to iterate the for loop for a fixed number of times. They differ only when we talk about Python versions.

The difference between range and xrange is as follows:

Range() method

The xrange() method is not supported in Python3 so that the range() method is used for iteration in for loops.

Xrange() method

The xrange() method is used only in Python version 2 for the iteration in loops.

The list is returned by this range() method

It only returns the generator object because it doesn't produce a static list during run time.

It occupies a huge amount of memory as it stores the complete list of iterating numbers in memory.

It occupies less memory because it only stores one number at a time in memory.

Following are the Python Advanced Interview Questions with Answers

63. What is Django?

Django is an advanced python web framework that supports agile growth and clean pragmatic design, built through experienced developers, this cares much about the trouble of web development, so you can concentrate on writing your app without wanting to reinvent the wheel.

64. List the features of Django?

- Excellent documentation
- Python web framework
- SEO optimized
- High scalability
- Versatile in nature
- Offers high security
- Thoroughly tested
- Provides rapid Development

65. Which level framework does Django belong to?

Django is a high-level Python web framework that was developed for realistic design, clean, rapid development.

66. What are the advantages of Django?

- One of the important advantages of Django is it is a framework of python language which is very simple to learn
- Django is a multifaceted framework
- When it comes to security Django is the best framework
- Scalability is added advantage of Django

67. Why should we use the Django framework?

The main goal to designing Django is to make it simple to its users, to do this Django uses:

- The principles concerning rapid development, which implies developers can complete more than one iteration at a time without beginning the full schedule from scratch;
- DRY philosophy — Do not Replicate Yourself — that means developers can reuse surviving code and also focus on the individual one.

68. List the common security issues that can be avoided by using Django?

A few common security issues that can be avoided by using Django are:

- Clickjacking
- Cross-site scripting and
- SQL injection

69. List a few of the famous companies that are using Django?

Few well-known companies that are using the Django framework are

- Instagram
- Spotify
- Mozilla
- Dropbox
- NASA

70. What can we do with the Django framework?

Here is an exciting reality: Django has first created to power a web application as a newspaper publisher, the Lawrence Journal-World. You all can demand it to be very good about handling projects by volumes from the text files, media, including extremely high traffic, or else something that operates as a web publication.

71. List steps for setting up static files in Django?

Ans: There are only three main steps for setting up Django static files

- Firstly set STATIC_ROOT in settings.py
- Run manage.py collect static
- Setting up a static file entry pythonAnywhere tab

72. Is Django stable?

Yes, Django is used by many famous companies because it is quite stable.

73. Differentiate Django reusability code with other frameworks?

Django web framework is operated and also maintained by an autonomous and non-profit organization designated as Django Software Foundation (DSF). The initial foundation goal is to promote, support, and advance this Django Web framework.

74. How can we handle URLs in Django?

```
from django.contrib import admin

from django.urls import path

urlpatterns = [
    path('appmajix/', appmajix.site.urls),
]
```

75. List the mandatory files of the Django project?

- manage.py
- settings.py
- __init__.py
- urls.py
- wsgi.py

76. Explain the Django session?

A session comprises a mechanism to store information on a specific server-side at the interaction by the web application. By default, session reserves in the database and allows file-based and cache-based sessions.

77. Why do we use a cookie in Django?

A cookie is a piece of information that is stored in a client's browser for a specific time. When the specific time is completed cookie gets automatically removed from the client browser.

78. Mentions the methods used for setting and getting cookie values?

The two methods to set and get cookie values are

- Set_cookie this method is used to set the values of the cookie
- Get_cookie this method is used to get the values of the cookie

79. What is the use of Django-admin.py?

Django-admin.py is a command-line argument that is utilized for administrative tasks

80. What is the use of manage.py?

It is an automatically built file inside each Django project. It is a flat wrapper encompassing the Django-admin.py. It possesses the following usage:

- It establishes your project's package on sys.path.
- It fixes the DJANGO_SETTING_MODULE environment variable to point to your project's setting.py file.

81. Why is Django loosely packed?

Django has described as a loosely coupled framework because of the MTV architecture it's based upon. Django's architecture means a variant of MVC architecture and also MTV is helpful because this completely separates the server code of the client's machine.

82. List the ways we add view functions to urls.py?

- Adding a function view
- Adding a class-based view

83. Explain how can we build or set up the database in Django?

we can make use of the edit mysite/setting.py command, which is a simple Python module that consists of levels for presenting or displaying Django settings.

By default Django uses SQLite; this also makes it easy for Django users in case of any other type of installations. For example, if your database choice is different then you need to follow certain keys in the DATABASE like default items to match database connection settings.

- Engines: By these engines you change the database by using commands such as ‘django.db.backends.postgresql_psycopg2’, ‘django.db.backends.sqlite3’, ‘django.db.backends.oracle’, ‘django.db.backends.mysql’, and so on.
- Name: This represents the name of your own database. If you are familiar with using SQLite as your database, in such cases database is available as a file on your particular system. Moreover, the name should be as a fully absolute or exact path along with the file name of the particular file.
- Suppose if we are not using SQLite as your database then additional settings such as password, user, the host must be added.

Django mainly uses SQLite as its default database to store entire information in a single file of the filesystem. If you want to use different database servers rather than SQLite, then make use of database administration tools to create a new database for the Django project. Another way is by using your own database in that place, and the remaining is to explain Django about how to use it. This is the place in which Python’s project settings.py file comes into the picture.

We need to add the below code to the setting.py file:

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

84. List out the inheritance styles in Django?

There are three possible inheritance styles in Django, and they are:

- Proxy models: This style is mainly used for those who want to modify the Python level behavior of the model, without modifying the model’s fields.

- Abstract Base Classes: This inheritance style is used only when we want to make parent class hold the data which they don't want to repeat again in the child class.
- Multi-table Inheritance: This inheritance style is used only when we want to subclass an existing model and there must a database table designed for each model on its own.

85. How to save an image locally using Python in which we already know the URL address?

The following code is used to save the image locally from the URL address which we know.

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

86. How can we access sessions in flask?

A session will basically allow us to remember information from one request to another. In a flask, a signed cookie is used to make the user look at the session contents and modify them. Moreover, the user can change the session only when the secret key named Flask.secret_key is present.

87. Is flask an MVC model? If yes, justify your answer by showing an example of your application with the help of the MVC pattern?

Basically, a flask is a minimalistic framework that behaves the same as the MVC framework. So MVC will be perfectly suitable for the flask and we will consider the MVC pattern in the below example.

```
from flask import Flask
In this code your,
app = Flask(__name__)
@app.route("/")
Def hey():
return "Welcome to Appmajix"
app.run(debug = True)
```

The following code can be fragmented into

Configuration part will be,

In this code your,

```
app = Flask(__name__)
```

View part will be,

```
@app.route("/")
Def hey():
    return "Welcome to Appmajix"
```

While your main part will be,

```
app.run(debug = True)
```

88. What are the database connections in Python Flask, explain?

Database-powered applications are supported by the flask. The relational database systems need to create a schema that requires piping the schema.sql file into an SQLite3 command. So, in this case, you need to install the SQLite3 command on your system to initiate and create the database in the flask.

We can request a database using flask in three ways, and they are:

- **before_request()**: Using this we can request database before only without passing arguments.
- **after_request()**: This method is called after requesting the database and also send the response to the client.
- **teardown_request()**: This method is called in the cases where the responses are not guaranteed and the exception is raised. They have no access to modify the request.

89. Explain the procedure to minimize or lower the outages of the Memcached server in your Python development?

The following are the steps used to minimize the outages of the Memcached server in your Python development, and they are.

- When a single instance fails, this will impact on larger load of the database server. The client makes the request when the data is reloaded. In order to avoid this, the code that you have written must be used to lower cache stampedes then it will be used to leave a minimal impact.
- The other way is to bring out the instance of the Memcached on a new machine by using the IP address of the lost machine.

- Another important option is to lower the server outages is code. This code provides you with the liberty to modify the Memcached server list with minimal work
- another way is by setting a timeout value that will be one of the options for memcac
- Class Student:
- def __init__(self, name):
- self.name = name
- S1=Student("XYZ")
print(S1.name)
- Ask clients to implement the Memcached server outage. When the performance of the server goes down, the client keeps on sending a request until the timeout limit is reached.

90. What is the Dogpile effect?

This is defined as an occurrence of an event when the cache expires and also when the websites are hit with more requests by the client at a time. This dogpile effect can be averted by the use of a semaphore lock. If in the particular system the value expires then, first of all, the particular process receives the lock and begins generating new value.

91. What are the OOP's concepts available in Python?

Ans: Python is also an object-oriented programming language like other programming languages. It also contains different OOP's concepts, and they are

- Object
- Class
- Method
- Encapsulation
- Abstraction
- Inheritance
- Polymorphism

92. Define object in Python?

An object in Python is defined as an instance that has both state and behavior. Everything in Python is made of objects.

93. What is a class in Python?

Class is defined as a logical entity that is a huge collection of objects and it also contains both methods and attributes.

94. How to create a class in Python?

In Python programming, the class is created using a class keyword. The syntax for creating a class is as follows:

```
class ClassName:  
#code (statement-suite)
```

Example of creating a class in Python.

Output:

```
XYZ
```

95. What is the syntax for creating an instance of a class in Python?

Ans: The syntax for creating an instance of a class is as follows:

```
<object-name> = <class-name>(<arguments>)
```

96. Define what is “Method” in Python programming?

The Method is defined as the function associated with a particular object. The method which we define should not be unique as a class instance. Any type of object can have methods.

97. Does multiple inheritances are supported in Python?

Multiple inheritances are supported in python. It is a process that provides flexibility to inherit multiple base classes in a child class.

An example of multiple inheritances in Python is as follows:

```
class Calculus:  
def Sum(self,a,b):  
return a+b;
```

```
class Calculus1:  
    def Mul(self,a,b):  
        return a*b;  
  
class Derived(Calculus,Calculus1):  
    def Div(self,a,b):  
        return a/b;  
  
d = Derived()  
print(d.Sum(10,30))  
print(d.Mul(10,30))  
print(d.Div(10,30))
```

Output:

```
40  
300  
0.3333
```

98. What is data abstraction in Python?

In simple words, abstraction can be defined as hiding unnecessary data and showing or executing necessary data. In technical terms, abstraction can be defined as hiding internal processes and showing only the functionality. In Python abstraction can be achieved using encapsulation.

99. Define encapsulation in Python?

Encapsulation is one of the most important aspects of object-oriented programming. The binding or wrapping of code and data together into a single cell is called encapsulation. Encapsulation in Python is mainly used to restrict access to methods and variables.

100. What is polymorphism in Python?

By using polymorphism in Python we will understand how to perform a single task in different ways. For example, designing a shape is the task and various possible ways in shapes are a triangle, rectangle, circle, and so on.

101. Does Python make use of access specifiers?

Python does not make use of access specifiers and also it does not provide a way to access an instance variable. Python introduced a concept of prefixing the name of the method, function, or variable by using a double or single underscore to act like the behavior of private and protected access specifiers.

102. How can we create an empty class in Python?

Empty class in Python is defined as a class that does not contain any code defined within the block. It can be created using pass keywords and object to this class can be created outside the class itself.

Example:

```
class x:  
    pass  
obj=x()  
obj.id="123"  
print("Id = ",obj.id)
```

Output:

```
123
```

103. Define Constructor in Python?

Constructor is a special type of method with a block of code to initialize the state of instance members of the class. A constructor is called only when the instance of the object is created. It is also used to verify that they are sufficient resources for objects to perform a specific task.

There are two types of constructors in Python, and they are:

- Parameterized constructor
- Non-parameterized constructor

104. How can we create a constructor in Python programming?

The `_init_` method in Python stimulates the constructor of the class. Creating a constructor in Python can be explained clearly in the below example.

```
class Student:  
    def __init__(self,name,id):
```

```
self.id = id;  
self.name = name;  
def display (self):  
    print("ID: %d nName: %s"%(self.id,self.name))  
stu1 =Student("nirvi",105)  
stu2 = Student("tanvi",106)  
#accessing display() method to print employee 1 information  
stu1.display();  
#accessing display() method to print employee 2 information  
stu2.display();
```

Output:

```
ID: 1  
Name: nirvi  
ID: 106  
Name: Tanvi
```

105. Define Inheritance in Python?

When an object of child class has the ability to acquire the properties of a parent class then it is called inheritance. It is mainly used to acquire runtime polymorphism and also it provides code reusability.

1. In-place swapping of two numbers

```
p, q = 20, 40  
print(p, q)  
p, q = q, p  
print(p, q)
```

Output:

```
20 40  
40 20
```

2. Reversing a String in Python

```
x = "MindmajixTraining"  
print( "Reverse is" , x[:: -1]
```

Output:

Reverse is gniniarTxijamdnM

3. Create a single string from all the elements in the list

```
x = ["Mindmajix", "Online", "Training"]  
print(" ".join(x))
```

Output:

Mindmajix Online Training

4. Use of Enums in Python

```
class Training:  
    Mindmajix, Online, Mindmajix = range(3)  
    print(MyName.Mindmajix)  
    print(MyName.Online)  
    print(MyName.Mindmajix)
```

Output:

2
1
2

5. Return multiple values from functions

```
def a():  
    return 4, 5, 6, 7  
p, q, r, s = a()  
print(p, q, r, s)
```

Output:

4, 5, 6, 7

6. Print String N times

```
n = 3  
x = "Mindmajix";  
print(a * n)
```

Output:

```
MindmajixMindmajixMindmajix
```

7. Check the memory usage of an object

```
import sys  
x = 10  
print(sys.getsizeof(x))
```

Output:

```
28
```

8. Find the most frequent value in a list

```
test = [1, 2, 3, 9, 2, 7, 3, 5, 9, 9, 9]  
print(max(set(test), key = test.count))
```

Output:

```
9
```

9. Checking whether the two words are anagrams or not

```
from collections import Counter  
  
def is_anagram(str1, str2):  
    return Counter(str1) == Counter(str2)  
  
print(is_anagram('majix', 'magic'))  
print(is_anagram('majix', 'xijam'))
```

Output:

```
False
```

```
True
```

10. Print the file path of imported modules

```
import os;  
import socket;  
print(os)  
print(socket)
```

Output:

```
<module 'os' from '/usr/lib/python3.5/os.py'>  
<module 'socket' from '/usr/lib/python3.5/socket.py'>
```

1. Who is the person who created Python?



Dennis Ritchie



James Gosling



Guido Van Rossum



Graham Bell

2. What are some of the Python Software Foundation's (PSF) well-known sponsors?



Google, Bloomberg, Meta



AWS, NVIDIA



Microsoft, Salesforce, CapitalOne, Corning



All the above

3. After finding inspiration in the language, Guido Van Rossum began creating the Python programming language.

- C
- C++
- Java
- ABC

4. Python is a programming language at level .

- Low level
- Medium Level
- High Level
- None

5. The version of Python that is most frequently used is:

- 1.5
- 2.0
- 2.7
-

3.1

1. What is the Difference Between a Shallow Copy and Deep Copy?

Deepcopy creates a different object and populates it with the child objects of the original object. Therefore, changes in the original object are not reflected in the copy.

`copy.deepcopy()` creates a Deep Copy.

Shallow copy creates a different object and populates it with the references of the child objects within the original object. Therefore, changes in the original object are reflected in the copy.

`copy.copy` creates a Shallow Copy.

2. How Is Multithreading Achieved in Python?

Multithreading usually implies that multiple **threads** are executed concurrently. The Python Global Interpreter Lock doesn't allow more than one thread to hold the Python interpreter at that particular point of time. So multithreading in python is achieved through context switching. It is quite different from multiprocessing which actually opens up multiple processes across multiple threads.

3. Discuss Django Architecture.

Here you can also find a [comprehensive guide on Python Django Tutorial](#) that is very easy to understand.

Django is a web service used to build your web pages. Its architecture is as shown:

- Template: the front end of the web page
- Model: the back end where the data is stored
- View: It interacts with the model and template and maps it to the URL
- Django: serves the page to the user

4. What Advantage Does the Numpy Array Have over a Nested List?

Numpy is written in C so that all its complexities are backed into a simple to use a module. Lists, on the other hand, are dynamically typed. Therefore, Python must check the data type of each element every time it uses it. This makes Numpy arrays much faster than lists.

Numpy has a lot of additional functionality that list doesn't offer; for instance, a lot of things can be automated in Numpy.

5. What are Pickling and Unpickling?

Pickling	Unpickling
<ul style="list-style-type: none">• Converting a Python object hierarchy to a byte stream is called pickling• Pickling is also referred to as serialization	<ul style="list-style-type: none">• Converting a byte stream to a Python object hierarchy is called unpickling• Unpickling is also referred to as deserialization

If you just created a neural network model, you can save that model to your hard drive, pickle it, and then unpickle to bring it back into another software program or to use it at a later time.

The following are some of the most frequently asked Python interview questions

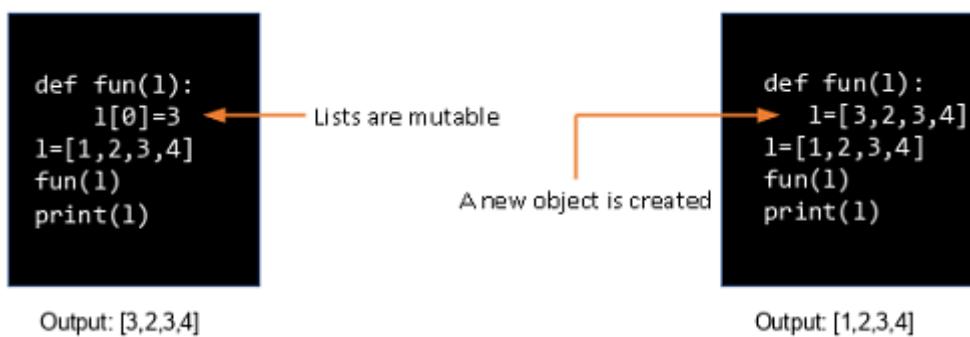
6. How is Memory managed in Python?

Python has a private heap space that stores all the **objects**. The Python memory manager regulates various aspects of this heap, such as sharing, caching, segmentation, and allocation. The user has no control over the heap; only the Python interpreter has access.

7. Are Arguments in Python Passed by Value or by Reference?

Arguments are passed in python by a reference. This means that any changes made within a function are reflected in the original object.

Consider two sets of code shown below:



In the first example, we only assigned a value to one element of ‘l’, so the output is `[3, 2, 3, 4]`.

In the second example, we have created a whole new object for ‘l’. But, the values `[3, 2, 3, 4]` doesn’t show up in the output as it is outside the definition of the function.

8. How Would You Generate Random Numbers in Python?

To generate random numbers in Python, you must first import the `random` module.

The `random()` function generates a random float value between 0 & 1.

```
> random.random()
```

The `randrange()` function generates a random number within a given range.

Syntax: `randrange(beginning, end, step)`

Example - > `random.randrange(1,10,2)`

9. What Does the // Operator Do?

In Python, the / operator performs division and returns the quotient in the float.

For example: 5 / 2 returns 2.5

The // operator, on the other hand, returns the quotient in integer.

For example: 5 // 2 returns 2

10. What Does the ‘is’ Operator Do?

The ‘is’ **operator** compares the id of the two objects.

list1=[1,2,3]

list2=[1,2,3]

list3=list1

list1 == list2 True

list1 is list2 False

list1 is list3 True

11. What Is the Purpose of the Pass Statement?

The pass statement is used when there's a syntactic but not an operational requirement. For example - The program below prints a string ignoring the spaces.

```
var="Si mplilea rn"
```

```
for i in var:
```

```
    if i==" ":
```

```
        pass
```

```
    else:
```

```
print(i,end="")
```

Here, the pass statement refers to ‘no action required.’

12. How Will You Check If All the Characters in a String Are Alphanumeric?

Python has an inbuilt method `isalnum()` which returns true if all characters in the string are alphanumeric.

Example -

```
>> "abcd123".isalnum()
```

Output: True

```
>>"abcd@123#".isalnum()
```

Output: False

Another way is to use regex as shown.

```
>>import re
```

```
>>bool(re.match('[A-Za-z0-9]+$', 'abcd123'))
```

Output: True

```
>> bool(re.match('[A-Za-z0-9]+$', 'abcd@123'))
```

Output: False

13. How Will You Merge Elements in a Sequence?

There are three types of sequences in Python:

- Lists
- Tuples
- Strings

Example of Lists -

```
>>l1=[1,2,3]
```

```
>>l2=[4,5,6]
```

```
>>l1+l2
```

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

Example of Tuples -

```
>>t1=(1,2,3)
```

```
>>t2=(4,5,6)
```

```
>>t1+t2
```

Output: (1,2,3,4,5,6)

Example of String -

```
>>s1="Simpli"
```

```
>>s2="learn"
```

```
>>s1+s2
```

Output: 'Simplilearn'

14. How Would You Remove All Leading Whitespace in a String?

Python provides the inbuilt function `lstrip()` to remove all leading spaces from a string.

```
>>"    Python".lstrip
```

Output: Python

15. How Would You Replace All Occurrences of a Substring with a New String?

The `replace()` function can be used with strings for replacing a substring with a given string. Syntax:

```
str.replace(old, new, count)
```

`replace()` returns a new string without modifying the original string.

Example -

```
>>"Hey John. How are you, John?".replace("john","John",1)
```

Output: "Hey John. How are you, John?"

16. What Is the Difference Between Del and Remove() on Lists?

del	remove()
<ul style="list-style-type: none">• <code>del</code> removes all elements of a list within a given range• Syntax: <code>del list[start:end]</code>	<ul style="list-style-type: none">• <code>remove()</code> removes the first occurrence of a particular character• Syntax: <code>list.remove(element)</code>

Here is an example to understand the two statements -

```
>>lis=['a', 'b', 'c', 'd']
```

```
>>del lis[1:3]
```

```
>>lis
```

Output: ["a", "d"]

```
>>lis=['a', 'b', 'b', 'd']
```

```
>>lis.remove('b')
```

```
>>lis
```

Output: ['a', 'b', 'd']

Note that in the range 1:3, the elements are counted up to 2 and not 3.



17. How Do You Display the Contents of a Text File in Reverse Order?

You can display the contents of a text file in reverse order using the following steps:

- Open the file using the `open()` function
- Store the contents of the file into a list
- Reverse the contents of the list
- Run a `for loop` to iterate through the list

18. Differentiate Between `append()` and `extend()`.

append()	extend()
<ul style="list-style-type: none">• <code>append()</code> adds an element to the end of the list• Example - <pre>>>lst=[1,2,3]</pre>	<ul style="list-style-type: none">• <code>extend()</code> adds elements from an iterable to the end of the list• Example -

```
>>lst.append(4)
```

```
>>lst
```

```
Output:[1,2,3,4]
```

```
>>lst=[1,2,3]
```

```
>>lst.extend([4,5,6])
```

```
>>lst
```

```
Output:[1,2,3,4,5,6]
```

19. What Is the Output of the below Code? Justify Your Answer.

```
>>def addToList(val, list=[]):
```

```
>> list.append(val)
```

```
>> return list
```

```
>>list1 = addToList(1)
```

```
>>list2 = addToList(123,[])
```

```
>>list3 = addToList('a')
```

```
>>print ("list1 = %s" % list1)
```

```
>>print ("list2 = %s" % list2)
```

```
>>print ("list3 = %s" % list3)
```

Output:

```
list1 = [1,'a']
```

```
list2 = [123]
```

```
list3 = [1,'a']
```

Note that list1 and list3 are equal. When we passed the information to the addToList, we did it without a second value. If we don't have an empty list as the second value, it will start off with an empty list, which we then append. For list2, we appended the value to an empty list, so its value becomes [123].

For list3, we're adding 'a' to the list. Because we didn't designate the list, it is a shared value. It means the list doesn't reset and we get its value as [1, 'a'].

Remember that a default list is created only once during the function and not during its call number.

20. What Is the Difference Between a List and a Tuple?

Lists are mutable while **tuples** are immutable.

Example:

List

```
>>lst = [1,2,3]
```

```
>>lst[2] = 4
```

```
>>lst
```

Output:[1,2,4]

Tuple

```
>>tpl = (1,2,3)
```

```
>>tpl[2] = 4
```

```
>>tpl
```

Output:TypeError: 'tuple'

the object does not support item

assignment

There is an error because you can't change the tuple 1 2 3 into 1 2 4. You have to completely reassign tuple to a new value.

21. What Is Docstring in Python?

This is one of the most frequently asked Python interview questions

Docstrings are used in providing documentation to various Python modules, classes, functions, and methods.

Example -

```
def add(a,b):  
    """This function adds two numbers.  
    sum=a+b  
    return sum  
  
sum=add(10,20)  
  
print("Accessing doctstring method 1:",add.__doc__)  
  
print("Accessing doctstring method 2:",end="")  
  
help(add)
```

Output -

Accessing docstring method 1: This function adds two numbers.

Accessing docstring method 2: Help on function add-in module __main__:

add(a, b)

This function adds two numbers.

22. How Do You Use Print() Without the Newline?

The solution to this depends on the Python version you are using.

Python v2

```
>>print("Hi. "),  
>>print("How are you?")
```

Output: Hi. How are you?

Python v3

```
>>print("Hi",end=" ")  
>>print("How are you?")
```

Output: Hi. How are you?

23. How Do You Use the Split() Function in Python?

The split() function splits a string into a number of strings based on a specific delimiter.

Syntax -

```
string.split(delimiter, max)
```

Where:

the delimiter is the character based on which the **string** is split. By default it is space.

max is the maximum number of splits

Example -

```
>>var="Red,Blue,Green,Orange"
```

```
>>lst=var.split(“,”,2)
```

```
>>print(lst)
```

Output:

```
[‘Red’,’Blue’,’Green, Orange’]
```

Here, we have a variable var whose values are to be split with commas. Note that ‘2’ indicates that only the first two values will be split.

24. Is Python Object-oriented or Functional Programming?

Python is considered a multi-paradigm language.

Python follows the object-oriented paradigm

- Python allows the creation of objects and their manipulation through specific methods
- It supports most of the features of OOPS such as inheritance and polymorphism

Python follows the functional programming paradigm

- Functions may be used as the first-class object
- Python supports Lambda functions which are characteristic of the functional paradigm

25. Write a Function Prototype That Takes a Variable Number of Arguments.

The function prototype is as follows:

```
def function_name(*list)
```

```
>>def fun(*var):
```

```
>> for i in var:
```

```
print(i)
```

```
>>fun(1)
```

```
>>fun(1,25,6)
```

In the above code, * indicates that there are multiple arguments of a variable.

26. What Are *args and *kwargs?

*args

- It is used in a function prototype to accept a varying number of arguments.
- It's an iterable object.
- Usage - def fun(*args)

*kwargs

- It is used in a function prototype to accept the varying number of keyworded arguments.
- It's an iterable object
- Usage - def fun(**kwargs):

```
fun(colour="red".units=2)
```

27. “in Python, Functions Are First-class Objects.” What Do You Infer from This?

It means that a function can be treated just like an object. You can assign them to variables, or pass them as arguments to other functions. You can even return them from other functions.

28. What Is the Output Of: Print(__name__)? Justify Your Answer.

`__name__` is a special variable that holds the name of the current module. Program execution starts from main or code with 0 indentations. Thus,

`__name__` has a value `__main__` in the above case. If the file is imported from another module, `__name__` holds the name of this module.

29. What Is a Numpy Array?

A numpy array is a grid of values, all of the same type, and is indexed by a tuple of non-negative integers. The number of dimensions determines the rank of the array. The shape of an array is a tuple of integers giving the size of the array along each dimension.

30. What Is the Difference Between Matrices and Arrays?

Matrices	Arrays
<ul style="list-style-type: none">• A matrix comes from linear algebra and is a two-dimensional representation of data• It comes with a powerful set of mathematical operations that allow you to manipulate the data in interesting ways	<ul style="list-style-type: none">• An array is a sequence of objects of similar data type• An array within another array forms a matrix

Python Interview Questions For Experienced

Next, let's learn about some advanced Python concepts in this Python Interview Questions tutorial.

31. How Do You Get Indices of N Maximum Values in a Numpy Array?

```
>>import numpy as np
```

```
>>arr=np.array([1, 3, 2, 4, 5])
```

```
>>print(arr.argsort( ) [ -N: ][: : -1])
```

32. How Would You Obtain the Res_set from the Train_set and the Test_set from Below?

```
>>train_set=np.array([1, 2, 3])
```

```
>>test_set=np.array([[0, 1, 2], [1, 2, 3]])
```

Res_set □ [[1, 2, 3], [0, 1, 2], [1, 2, 3]]

Choose the correct option:

1. res_set = train_set.append(test_set)
2. res_set = np.concatenate([train_set, test_set]))
3. resulting_set = np.vstack([train_set, test_set])
4. None of these

Here, options a and b would both do horizontal stacking, but we want vertical stacking. So, option c is the right statement.

```
resulting_set = np.vstack([train_set, test_set])
```

33. How Would You Import a Decision Tree Classifier in Sklearn? Choose the Correct Option.

1. from sklearn.decision_tree import DecisionTreeClassifier
2. from sklearn.ensemble import DecisionTreeClassifier
3. from sklearn.tree import DecisionTreeClassifier
4. None of these

Answer - 3. from sklearn.tree import DecisionTreeClassifier

34. You Have Uploaded the Dataset in Csv Format on Google Spreadsheet and Shared It Publicly. How Can You Access This in Python?

We can use the following code:

```
>>link = https://docs.google.com/spreadsheets/d/...
>>source = StringIO.StringIO(requests.get(link).content)
>>data = pd.read_csv(source)
```

35. What Is the Difference Between the Two Data Series given Below?

df['Name'] and df.loc[:, 'Name'], where:

```
df = pd.DataFrame(['aa', 'bb', 'xx', 'uu'], [21, 16, 50, 33], columns = ['Name', 'Age'])
```

Choose the correct option:

1. 1 is the view of original dataframe and 2 is a copy of original dataframe
2. 2 is the view of original dataframe and 1 is a copy of original dataframe
3. Both are copies of original dataframe
4. Both are views of original dataframe

Answer - 3. Both are copies of the original dataframe.

36. You Get the Error “temp.Csv” While Trying to Read a File Using Pandas. Which of the Following Could Correct It?

Error:

```
Traceback (most recent call last): File "<input>", line 1, in <module>
UnicodeEncodeError:
```

'ascii' codec can't encode character.

Choose the correct option:

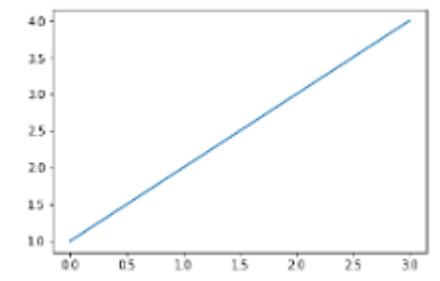
1. pd.read_csv("temp.csv", compression='gzip')

2. pd.read_csv("temp.csv", dialect='str')
3. pd.read_csv("temp.csv", encoding='utf-8')
4. None of these

The error relates to the difference between utf-8 coding and a Unicode.

So option 3. pd.read_csv("temp.csv", encoding='utf-8') can correct it.

37. How Do You Set a Line Width in the Plot given Below?



```
>>import matplotlib.pyplot as plt
```

```
>>plt.plot([1,2,3,4])
```

```
>>plt.show()
```

Choose the correct option:

1. In line two, write plt.plot([1,2,3,4], width=3)
2. In line two, write plt.plot([1,2,3,4], line_width=3)
3. In line two, write plt.plot([1,2,3,4], lw=3)
4. None of these

Answer - 3. In line two, write plt.plot([1,2,3,4], lw=3)

38. How Would You Reset the Index of a Dataframe to a given List? Choose the Correct Option.

1. df.reset_index(new_index,)

2. df.reindex(new_index,)
3. df.reindex_like(new_index,)
4. None of these

Answer - 3. df.reindex_like(new_index,)

39. How Can You Copy Objects in Python?

The function used to copy objects in Python are:

copy.copy for shallow copy and

copy.deepcopy() for deep copy

40. What Is the Difference Between range() and xrange() Functions in Python?

range()	xrange()
<ul style="list-style-type: none">• range returns a Python list object	<ul style="list-style-type: none">• xrange returns an xrange object

41. How Can You Check Whether a Pandas Dataframe Is Empty or Not?

The attribute df.empty is used to check whether a pandas data frame is empty or not.

```
>>import pandas as pd
```

```
>>df=pd.DataFrame({A:[]})
```

```
>>df.empty
```

Output: True

42. Write a Code to Sort an Array in Numpy by the (N-1)Th Column.

This can be achieved by using argsort() function. Let us take an array X; the code to sort the (n-1)th column will be x[x [: n-2].argsort()]

The code is as shown below:

```
>>import numpy as np  
  
>>X=np.array([[1,2,3],[0,5,2],[2,3,4]])  
  
>>X[X[:,1].argsort()]  
  
Output:array([[1,2,3],[0,5,2],[2,3,4]])
```

43. How Do You Create a Series from a List, Numpy Array, and Dictionary?

The code is as shown:

```
>> #Input  
  
>>import numpy as np  
  
>>import pandas as pd  
  
>>mylist = list('abcdefghijklmnopqrstuvwxyz')  
  
>>myarr = np.arange(26)  
  
>>mydict = dict(zip(mylist, myarr))  
  
>> #Solution  
  
>>ser1 = pd.Series(mylist)  
  
>>ser2 = pd.Series(myarr)  
  
>>ser3 = pd.Series(mydict)
```

```
>>print(ser3.head())
```

44. How Do You Get the Items Not Common to Both Series a and Series B?

```
>> #Input
```

```
>>import pandas as pd
```

```
>>ser1 = pd.Series([1, 2, 3, 4, 5])
```

```
>>ser2 = pd.Series([4, 5, 6, 7, 8])
```

```
>> #Solution
```

```
>>ser_u = pd.Series(np.union1d(ser1, ser2)) # union
```

```
>>ser_i = pd.Series(np.intersect1d(ser1, ser2)) # intersect
```

```
>>ser_u[~ser_u.isin(ser_i)]
```

45. How Do You Keep Only the Top Two Most Frequent Values as It Is and Replace Everything Else as ‘other’ in a Series?

```
>> #Input
```

```
>>import pandas as pd
```

```
>>np.random.RandomState(100)
```

```
>>ser = pd.Series(np.random.randint(1, 5, [12]))
```

```
>> #Solution
```

```
>>print("Top 2 Freq:", ser.value_counts())
```

```
>>ser[~ser.isin(ser.value_counts().index[:2])] = 'Other'
```

```
>>ser
```

46. How Do You Find the Positions of Numbers That Are Multiples of Three from a Series?

```
>> #Input  
  
>>import pandas as pd  
  
>>ser = pd.Series(np.random.randint(1, 10, 7))  
  
>>ser  
  
>> #Solution  
  
>>print(ser)  
  
>>np.argwhere(ser % 3==0)
```

47. How Do You Compute the Euclidean Distance Between Two Series?

The code is as shown:

```
>> #Input  
  
>>p = pd.Series([1, 2, 3, 4, 5, 6, 7, 8, 9, 10])  
  
>>q = pd.Series([10, 9, 8, 7, 6, 5, 4, 3, 2, 1])  
  
>> #Solution  
  
>>sum((p - q)**2)**.5  
  
>> #Solution using func  
  
>>np.linalg.norm(p-q)
```

You can see that the Euclidean distance can be calculated using two ways.

48. How Do You Reverse the Rows of a Data Frame?

```
>> #Input  
  
>>df = pd.DataFrame(np.arange(25).reshape(5, -1))  
  
>> #Solution  
  
>>df.iloc[:-1, :]
```

49. If You Split Your Data into Train/Test Splits, Is It Possible to over Fit Your Model?

Yes. One common beginner mistake is re-tuning a model or training new models with different parameters after seeing its performance on the test set.

50. Which Python Library Is Built on Top of Matplotlib and Pandas to Ease Data Plotting?

Seaborn is a [Python library](#) built on top of matplotlib and pandas to ease data plotting. It is a data visualization library in Python that provides a high-level interface for drawing statistical informative graphs.

Did you know the answers to these Python interview questions? If not, here is what you can do.

51. What are the important features of Python?

- Python is a scripting language. Python, unlike other programming languages like C and its derivatives, does not require compilation prior to execution.
- Python is dynamically typed, which means you don't have to specify the kinds of variables when declaring them or anything.
- Python is well suited to object-oriented programming since it supports class definition, composition, and inheritance.

52. What type of language is Python?

Although Python can be used to write scripts, it is primarily used as a general-purpose programming language.

53. Explain how Python is an interpreted language.

Any programming language that is not in machine-level code before runtime is called an interpreted language. Python is thus an interpreted language.

54. What is PEP 8?

PEP denotes Python Enhancement Proposal. It's a collection of guidelines for formatting Python code for maximum readability.

55. Explain Python namespace.

In Python, a namespace refers to the name that is assigned to each object.

56. What are decorators in Python?

Decorators are used for changing the appearance of a function without changing its structure. Decorators are typically defined prior to the function they are enhancing.

57. How to use decorators in Python?

Decorators are typically defined prior to the function they are enhancing. To use a decorator, we must first specify its function. Then we write the function to which it is applied, simply placing the decorator function above the function to which it must be applied.

58. Differentiate between .pyc and .py.

The .py files are the source code files for Python. The bytecode of the python files are stored in .pyc files, which are created when code is imported from another source. The interpreter saves time by converting the source .py files to .pyc files.

59. What is slicing in Python?

Slicing is a technique for gaining access to specific bits of sequences such as strings, tuples, and lists.

60. How to use the slicing operator in Python?

Slicing is a technique for gaining access to specific bits of sequences such as lists, tuples, and strings. The slicing syntax is [start:end:step]. This step can also be skipped. [start:end] returns all sequence items from the start (inclusive) to the end-1 element. It means the ith element from the end of the start or end element is negative i. The step represents the jump or the number of components that must be skipped.

61. What are keywords in python?

In Python, keywords are reserved words with a specific meaning. They are commonly used to specify the type of variables. Variable and function names cannot contain keywords. Following are the 33 keywords of Python:

- Yield
- For
- Else
- Elif
- If
- Not
- Or
- And
- Raise
- Nonlocal
- None
- Is
- In
- Import
- Global
- From

- Finally
- Except
- Del
- Continue
- Class
- Assert
- With
- Try
- False
- True
- Return
- Pass
- Lambda
- Def
- As
- Break
- While

62. How to combine dataframes in Pandas?

This is one of the most commonly asked Python interview questions

The following are the ways through which the data frames in Pandas can be combined:

- Concatenating them by vertically stacking the two dataframes.
- Concatenating them by horizontally stacking the two dataframes.
- Putting them together in a single column.

63. What are the key features of the Python 3.9.0.0 version?

- Zoneinfo and graphlib are two new modules.
- Improved modules such as asyncio and ast.
- Optimizations include improved idiom for assignment, signal handling, and Python built-ins.
- Removal of erroneous methods and functions.
- Instead of LL1, a new parser is based on PEG.
- Remove Prefixes and Suffixes with New String Methods.
- Generics with type hinting in standard collections.

64. In Python, how is memory managed?

- Python's private heap space is in charge of memory management. A private heap holds all Python objects and data structures. This secret heap is not accessible to the programmer. Instead, the Python interpreter takes care of it.
- Python also includes a built-in garbage collector, which recycles all unused memory and makes it available to the heap space.
- Python's memory management is in charge of allocating heap space for Python objects. The core API allows programmers access to some programming tools.

65. Explain PYTHONPATH.

It's an environment variable that is used when you import a module. When a module is imported, PYTHONPATH is checked to see if the imported modules are present in various folders. It is used by the interpreter to determine which module to load.

66. Explain global variables and local variables in Python.

Local Variables:

A local variable is any variable declared within a function. This variable exists only in local space, not in global space.

Global Variables:

Global variables are variables declared outside of a function or in a global space. Any function in the program can access these variables.

67. Is Python case sensitive?

Yes, Python is case sensitive.

68. How to install Python on Windows and set path variables?

- Download Python from <https://www.python.org/downloads/>
- Install it on your computer. Using your command prompt, look for the location where PYTHON is installed on your computer by typing cmd python.
- Then, in advanced system settings, create a new variable called PYTHON_NAME and paste the copied path into it.
- Search the path variable, choose its value and select ‘edit’.
- If the value doesn't have a semicolon at the end, add one, and then type %PYTHON HOME%.

69. Is it necessary to indent in Python?

Indentation is required in Python. It designates a coding block. An indented block contains all of the code for loops, classes, functions, and so on. Typically, four space characters are used. Your code will not execute correctly if it is not indented, and it will also generate errors.

70. On Unix, how do you make a Python script executable?

Script file should start with #!/usr/bin/env python.

71. What is the use of self in Python?

Self is used to represent the class instance. In Python, you can access the class's attributes and methods with this keyword. It connects the attributes to the

arguments. Self appears in a variety of contexts and is frequently mistaken for a term. Self is not a keyword in Python, unlike in C++.

72. What are the literals in Python?

For primitive data types, a literal in Python source code indicates a fixed value.

73. What are the types of literals in Python?

For primitive data types, a literal in Python source code indicates a fixed value. Following are the 5 types of literal in Python:

- String Literal: A string literal is formed by assigning some text to a variable that is contained in single or double-quotes. Assign the multiline text encased in triple quotes to produce multiline literals.
- Numeric Literal: They may contain numeric values that are floating-point values, integers, or complex numbers.
- Character Literal: It is made by putting a single character in double-quotes.
- Boolean Literal: True or False
- Literal Collections: There are four types of literals such as list collections, tuple literals, set literals, dictionary literals, and set literals.

74. What are Python modules? Name a few Python built-in modules that are often used.

Python modules are files that contain Python code. Functions, classes, or variables can be used in this code. A Python module is a .py file that contains code that may be executed. The following are the commonly used built-in modules:

- JSON
- data time
- random
- math

- sys
- OS

75. What is `_init_`?

`_init_` is a constructor or method in Python. This method is used to allocate memory when a new object is created.

76. What is the Lambda function?

A lambda function is a type of anonymous function. This function can take as many parameters as you want, but just one statement.

77. Why Lambda is used in Python?

Lambda is typically utilized in instances where an anonymous function is required for a short period of time. Lambda functions can be applied in two different ways:

- Assigning Lambda functions to a variable
- Wrapping Lambda function into another function

78. How does continue, break, and pass work?

Continue

When a specified condition is met, the control is moved to the beginning of the loop, allowing some parts of the loop to be transferred.

Break	When a condition is met, the loop is terminated and control is passed to the next statement.
Pass	When you need a piece of code syntactically but don't want to execute it, use this. This is a null operation.

79. What are Python iterators?

Iterators are things that can be iterated through or traversed.

80. Differentiate between range and xrange.

In terms of functionality, xrange and range are essentially the same. They both provide you the option of generating a list of integers to use whatever you want. The sole difference between range and xrange is that range produces a Python list object whereas xrange returns an xrange object. This is especially true if you are working with a machine that requires a lot of memory, such as a phone because range will utilize as much memory as it can to generate your array of numbers, which can cause a memory error and crash your program. It is a beast with a memory problem.

81. What are unpickling and pickling?

The Pickle module takes any Python object and converts it to a string representation, which it then dumps into a file using the dump method. This is known as pickling. Unpickling is the process of recovering original Python objects from a stored text representation.

82. What are generators in Python?

Functions which return an iterable set of items are known as generators.

83. How do you copy an object in Python?

The assignment statement (= operator) in Python does not copy objects. Instead, it establishes a connection between the existing object and the name of the target variable. The copy module is used to make copies of an object in Python. Furthermore, the copy module provides two options for producing copies of a given object –

Deep Copy: Deep Copy recursively replicates all values from source to destination object, including the objects referenced by the source object.

```
from copy import copy, deepcopy
```

```
list_1 = [1, 2, [3, 5], 4]
```

```
## shallow copy
```

```
list_2 = copy(list_1)
```

```
list_2[3] = 7
```

```
list_2[2].append(6)
```

```
list_2 # output => [1, 2, [3, 5, 6], 7]
```

```
list_1 # output => [1, 2, [3, 5, 6], 4]
```

```
## deep copy
```

```
list_3 = deepcopy(list_1)
```

```
list_3[3] = 8
```

```
list_3[2].append(7)
```

```
list_3 # output => [1, 2, [3, 5, 6, 7], 8]
```

```
list_1 # output => [1, 2, [3, 5, 6], 4]
```

Shallow Copy: A bit-wise copy of an object is called a shallow copy. The values in the copied object are identical to those in the original object. If one of the values is a reference to another object, only its reference addresses are copied.

84. In Python, are arguments provided by value or reference?

Pass by value: The actual item's copy is passed. Changing the value of the object's copy has no effect on the original object's value.

Pass by reference: The actual object is passed as a reference. The value of the old object will change if the value of the new object is changed.

Arguments are passed by reference in Python.

```
def appendNumber(arr):
```

```
    arr.append(4)
```

```
arr = [1, 2, 3]
```

```
print(arr) #Output: => [1, 2, 3]
```

```
appendNumber(arr)
```

```
print(arr) #Output: => [1, 2, 3, 4]
```

85. How to delete a file in Python?

Use command `os.remove(file_name)` to delete a file in Python.

86. Explain join() and split() functions in Python.

The `join()` function can be used to combine a list of strings based on a delimiter into a single string.

The `split()` function can be used to split a string into a list of strings based on a delimiter.

```
string = "This is a string."  
  
string_list = string.split(' ') #delimiter is 'space' character or ''  
  
print(string_list) #output: ['This', 'is', 'a', 'string.'][br/>  
print(' '.join(string_list)) #output: This is a string.
```

87. Explain `**kwargs` and `*args`.

`*args`

- The function definition uses the `*args` syntax to pass variable-length parameters.
- `"*"` denotes variable length, while "args" is the standard name. Any other will suffice.

`**kwargs`

- `**kwargs` is a special syntax for passing variable-length keyworded arguments to functions.
- When a variable is passed to a function, it is called a keyworded argument.
- "Kwargs" is also used by convention here. You are free to use any other name.

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

- The indexes from the end of the list, tuple, or string are called negative indexes.
- `Arr[-1]` denotes the array's last element. `Arr[]`

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

The capitalize() function in Python capitalizes a string's initial letter. It returns the original text if the string already contains a capital letter at the beginning.

90. What method will you use to convert a string to all lowercase?

The lower() function can be used to convert a string to lowercase.

91. In Python, how do you remark numerous lines?

Comments that involve multiple lines are known as multi-line comments. A # must prefix all lines that will be commented. You can also use a convenient shortcut to remark several lines. All you have to do is hold down the ctrl key and left-click anywhere you want a # character to appear, then input a # once. This will add a comment to every line where you put your cursor.

92. What are docstrings?

Docstrings are documentation strings. Within triple quotations are these docstrings. They are not allocated to any variable and, as a result, they can also be used as comments.

93. What is the purpose of ‘not’, ‘is’, and ‘in’ operators?

Special functions are known as operators. They take one or more input values and output a result.

not- returns the boolean value's inverse

is- returns true when both operands are true

in- determines whether a certain element is present in a series

94. What are the functions help() and dir() used for in Python?

Both help() and dir() are available from the Python interpreter and are used to provide a condensed list of built-in functions.

dir() function: The defined symbols are displayed using the dir() function.

`help()` function: The `help()` function displays the documentation string and also allows you to access help for modules, keywords, attributes, and other items.

95. Why isn't all the memory de-allocated when Python exits?

- When Python quits, some Python modules, especially those with circular references to other objects or objects referenced from global namespaces, are not necessarily freed or deallocated.
- Python would try to de-allocate/destroy all other objects on exit because it has its own efficient cleanup mechanism.
- It is difficult to de-allocate memory that has been reserved by the C library.

96. What is a dictionary in Python?

Dictionary is one of Python's built-in datatypes. It establishes a one-to-one correspondence between keys and values. Dictionary keys and values are stored in pairs in dictionaries. Keys are used to index dictionaries.

97. In Python, how do you utilize ternary operators?

The Ternary operator is the operator for displaying conditional statements. This is made of true or false values and a statement that must be evaluated.

98. Explain the `split()`, `sub()`, and `subn()` methods of the Python "re" module.

Python's "re" module provides three ways for modifying strings. They are:

`split ()`: a regex pattern is used to "separate" a string into a list

`subn()`: It works similarly to `sub()`, returning the new string as well as the number of replacements.

`sub()`: identifies all substrings that match the regex pattern and replaces them with a new string

99. What are negative indexes and why do we utilize them?

Python sequences are indexed, and they include both positive and negative values. Positive numbers are indexed with '0' as the first index and '1' as the second index, and so on.

The index for a negative number begins with '-1,' which is the last index in the sequence, and ends with '-2,' which is the penultimate index, and the sequence continues like a positive number. The negative index is used to eliminate all new-line spaces from the string and allow it to accept the last character S[:-1]. The negative index can also be used to represent the correct order of the string.

100. Explain Python packages.

Packages in Python are namespaces that contain numerous modules.

101. What are built-in types of Python?

Given below are the built-in types of Python:

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

102. What are the benefits of NumPy arrays over (nested) Python lists?

- Lists in Python are useful general-purpose containers. They allow for (relatively) quick insertion, deletion, appending, and concatenation, and Python's list comprehensions make them simple to create and operate.
- They have some limitations: they don't enable "vectorized" operations like elementwise addition and multiplication, and because they can include objects of different types, Python must maintain type information for each element and execute type dispatching code while working on it.

- NumPy arrays are faster, and NumPy comes with a number of features, including histograms, algebra, linear, basic statistics, fast searching, convolutions, FFTs, and more.

103. What is the best way to add values to a Python array?

The append(), extend(), and insert (i,x) procedures can be used to add elements to an array.

104. What is the best way to remove values from a Python array?

The pop() and remove() methods can be used to remove elements from an array. The difference between these two functions is that one returns the removed value while the other does not.

105. Is there an object-oriented Programming (OOps) concept in Python?

Python is a computer language that focuses on objects. This indicates that by simply constructing an object model, every program can be solved in Python. Python, on the other hand, may be used as both a procedural and structured language.

106. Differentiate between deep and shallow copy.

When a new instance type is formed, a shallow copy is used to maintain the values that were copied in the previous instance. Shallow copy is used to copy reference pointers in the same way as values are copied. These references refer to the original objects, and any modifications made to any member of the class will have an impact on the original copy. Shallow copy enables faster program execution and is dependent on the size of the data being utilized.

Deep copy is a technique for storing previously copied values. The reference pointers to the objects are not copied during deep copy. It creates a reference to an object and stores the new object that is referenced to by another object. The changes made to the original copy will have no effect on any subsequent copies that utilize the item. Deep copy slows down program performance by creating many copies of each object that is called.

107. What are Python libraries?

A Python library is a group of Python packages. Numpy, Pandas, Matplotlib, Scikit-learn, and many other Python libraries are widely used.

108. Why split is used?

In Python, the split() function is used to split a string.

109. How multithreading is achieved in Python?

- Although Python includes a multi-threading module, it is usually not a good idea to utilize it if you want to multi-thread to speed up your code.
- As this happens so quickly, it may appear to the human eye that your threads are running in parallel, but they are actually sharing the same CPU core.
- The Global Interpreter Lock is a Python concept (GIL). Only one of your 'threads' can execute at a moment, thanks to the GIL. A thread obtains the GIL, performs some work, and then passes the GIL to the following thread.

110. How are classes created in Python?

The class keyword in Python is used to construct a class.

111. What is pandas dataframe?

A dataframe is a 2D changeable and tabular structure for representing data with rows and columns labelled.

112. Explain monkey patching in Python.

Monkey patches are solely used in Python to run-time dynamic updates to a class or module.

113. How Python module is imported?

The import keyword can be used to import modules.

114. What is inheritance in Python?

Inheritance allows one class to gain all of another class's members (for example, attributes and methods). Inheritance allows for code reuse, making it easier to develop and maintain applications.

115. What are the different types of inheritance in Python?

The following are the various types of inheritance in Python:

- Single inheritance: The members of a single super class are acquired by a derived class.
- Multiple inheritance: More than one base class is inherited by a derived class.
- Multi-level inheritance: D1 is a derived class inherited from base1 while D2 is inherited from base2.
- Hierarchical Inheritance: You can inherit any number of child classes from a single base class.

116. Is multiple inheritance possible in Python?

A class can be inherited from multiple parent classes, which is known as multiple inheritance. In contrast to Java, Python allows multiple inheritance.

117. Explain polymorphism in Python.

The ability to take various forms is known as polymorphism. For example, if the parent class has a method named ABC, the child class can likewise have a method named ABC with its own parameters and variables. Python makes polymorphism possible.

118. What is encapsulation in Python?

Encapsulation refers to the joining of code and data. Encapsulation is demonstrated through a Python class.

119. In Python, how do you abstract data?

Only the necessary details are provided, while the implementation is hidden from view. Interfaces and abstract classes can be used to do this in Python.

120. Is access specifiers used in Python?

Access to an instance variable or function is not limited in Python. To imitate the behavior of protected and private access specifiers, Python introduces the idea of prefixing the name of the variable, function, or method with a single or double underscore.

121. How to create an empty class in Python?

A class that has no code defined within its block is called an empty class. The pass keyword can be used to generate it. You can, however, create objects of this class outside of the class. When used in Python, the PASS command has no effect.

122. What does an object() do?

It produces a featureless object that serves as the foundation for all classes. It also does not accept any parameters.

123. Write a Python program to generate a Star triangle.

1

2

3

4

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

Output:

*

124. Write a program to produce the Fibonacci series in Python.

1	# Enter number of terms needednbsp;#0,1,1,2,3,5....
2	a=int(input("Enter the terms"))
3	f=0:#first element of series
4	s=1#second element of series
5	if a==0:
6	print("The requested series is",f)
7	
8	else:

```

9           print(f,s,end=" ")

10          for x in range(2,a):

11          print(next,end=" ")

12          f=s

13          s=next

```

Output: Enter the terms 5 0 1 1 2 3

125. Make a Python program that checks if a sequence is a Palindrome.

```
a=input("enter sequence")
```

```
b=a[::-1]
```

```
if a==b:
```

```
    print("palindrome")
```

```
else:
```

```
    print("Not a Palindrome")
```

Output: enter sequence 323 palindrome

126. Make a one-liner that counts how many capital letters are in a file. Even if the file is too large to fit in memory, your code should work.

```
1  
2  
3  
4  
5  
6
```

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

Let us transform this into a single line

```
1
```

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

127. Can you write a sorting algorithm with a numerical dataset?

```
1  
2
```

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

3

list.sort()

4

print (list)

128. Check code given below, list the final value of A0, A1 ...An.

1

A0 =
dict(zip(('a','b','c','d','e'),(1,2,3,4,5)))

2

A1 = range(10)A2 = sorted([i for i
in A1 if i in A0])

3

A3 = sorted([A0[s] for s in A0])

4

A4 = [i for i in A1 if i in A3]

5

A5 = {i:i*i for i in A1}

6

A6 = [[i,i*i] for i in A1]

7

print(A0,A1,A2,A3,A4,A5,A6)

Here's the answer:

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]]

129. What is Flask and explain its benefits.

Flask is a Python web microframework based on the BSD license. Two of its dependencies are Werkzeug and Jinja2. This means it will have few, if any, external library dependencies. It lightens the framework while reducing update dependencies and security vulnerabilities.

A session is just a way of remembering information from one request to the next. A session in a flask employs a signed cookie to allow the user to inspect and edit the contents of the session. If the user only has the secret key, he or she can change the session. Flask.secret key.

130. Is Django better as compared to Flask?

Django and Flask map URLs or addresses entered into web browsers into Python functions.

Flask is easier to use than Django, but it doesn't do much for you, so you will have to specify the specifics, whereas Django does a lot for you and you won't have to do anything. Django has prewritten code that the user must examine, whereas Flask allows users to write their own code, making it easier to grasp. Both are technically excellent and have their own set of advantages and disadvantages.

131. Differentiate between Pyramid, Django, and Flask.

- Pyramid is designed for larger apps. It gives developers flexibility and allows them to utilize the appropriate tools for their projects. The database, URL structure, templating style, and other options are all available to the developer. Pyramid can be easily customized.
- Flask is a "microframework" designed for small applications with straightforward needs. External libraries are required in a flask. The flask is now ready for use.
- Django, like Pyramid, may be used for larger applications. It has an ORM in it.

132. In NumPy, how will you read CSV data into an array?

This may be accomplished by utilizing the `genfromtxt()` method with a comma as the delimiter.

133. What is GIL?

The term **GIL** stands for Global Interpreter Lock. This is a mutex that helps thread synchronization by preventing deadlocks by limiting access to Python objects. GIL assists with multitasking (and not parallel computing).

134. What is PIP?

PIP denotes Python Installer Package. It is used to install various Python modules. It's a command-line utility that creates a unified interface for installing various Python modules. It searches the internet for the package and installs it into the working directory without requiring any user intervention.

135. What is the use of sessions in the Django framework?

Django has a session feature that allows you to store and retrieve data for each site visitor. Django isolates the process of sending and receiving cookies by keeping all necessary data on the server-side and inserting a session ID cookie on the client-side.

136. Write a program that checks if all of the numbers in a sequence are unique.

```
def check_distinct(data_list):
    if len(data_list) == len(set(data_list)):
        return True
    else:
        return False;
print(check_distinct([1,6,5,8])) #Prints True
```

```
print(check_distinct([2,2,5,5,7,8])) #Prints False
```

137. What is an operator in Python?

An operator is a symbol that is applied to a set of values to produce a result. An operator manipulates operands. Numeric literals or variables that hold values are known as operands. Unary, binary, and ternary operators are all possible. The unary operator, which requires only one operand, the binary operator, which requires two operands, and the ternary operator, which requires three operands.

138. What are the various types of operators in Python?

- Bitwise operators
- Identity operators
- Membership operators
- Logical operators
- Assignment operators
- Relational operators
- Arithmetic operators

139. How to write a Unicode string in Python?

The old Unicode type has been replaced with the "str" type in Python 3, and the string is now considered Unicode by default. Using the `art.title.encode("utf-8")` function, we can create a Unicode string.

140. Explain the differences between Python 2.x and Python 3.x?

Python 2.x is an older version of the Python programming language. Python 3.x is the most recent version. Python 2.x is no longer supported. Python 3.x is the language's present and future.

In Python2, a string is inherently ASCII, while in Python 3, it is Unicode.

141. How to send an email in Python language?

Python includes the `smtplib` and `email` libraries for sending emails. Import these modules into the newly generated mail script and send mail to users who have been authenticated.

142. Create a program to add two integers >0 without using the plus operator.

```
def add_nums(num1, num2):  
  
    while num2 != 0:  
  
        data = num1 & num2  
  
        num1 = num1 ^ num2  
  
        num2 = data << 1  
  
    return num1  
  
print(add_nums(2, 10))
```

143. Create a program to convert dates from yyyy-mm-dd to dd-mm-yyyy.

We can use this module to convert dates:

```
import re  
  
def transform_date_format(date):  
  
    return re.sub(r'(\d{4})-(\d{1,2})-(\d{1,2})', '\g{3}-\g{2}-\g{1}', date)  
  
date_input = "2021-08-01"  
  
print(transform_date_format(date_input))
```

The `datetime` module can also be used, as demonstrated below:

```
from datetime import datetime
```

```
new_date = datetime.strptime("2021-08-01", "%Y-%m-%d").strftime("%d:%m:%Y")  
  
print(new_data)
```

144. Create a program that combines two dictionaries. If you locate the same keys during combining, you can sum the values of these similar keys. Create a new dictionary.

```
from collections import Counter  
  
d1 = {'key1': 50, 'key2': 100, 'key3': 200}  
  
d2 = {'key1': 200, 'key2': 100, 'key4': 300}  
  
new_dict = Counter(d1) + Counter(d2)  
  
print(new_dict)
```

145. Is there an inherent do-while loop in Python?

No.

146. What kind of joins are offered by Pandas?

There are four joins in Pandas: left, inner, right, and outer.

147. How are dataframes in Pandas merged?

The type and fields of the dataframes being merged determine how they are merged. If the data has identical fields, it is combined along axis 0, otherwise, it is merged along axis 1.

148. What is the best way to get the first five entries of a data frame?

We may get the top five entries of a data frame using the head(5) method. df.head() returns the top 5 rows by default. df.head(n) will be used to fetch the top n rows.

149. How can you access the data frame's latest five entries?

We may get the top five entries of a dataframe using the tail(5) method. df.tail() returns the top 5 rows by default. df.tail(n) will be used to fetch the last n rows.

150. Explain classifier.

Any data point's class is predicted using a classifier. Classifiers are hypotheses that are used to assign labels to data items based on their classification.

1. What is Python?

Python was created and first released in 1991 by Guido van Rossum. It is a high-level, general-purpose programming language emphasizing code readability and providing easy-to-use syntax. Several developers and programmers prefer using Python for their programming needs due to its simplicity. After 30 years, Van Rossum stepped down as the leader of the community in 2018.

Python interpreters are available for many operating systems. CPython, the reference implementation of Python, is open-source software and has a community-based development model, as do nearly all of its variant implementations. The non-profit Python Software Foundation manages Python and CPython.

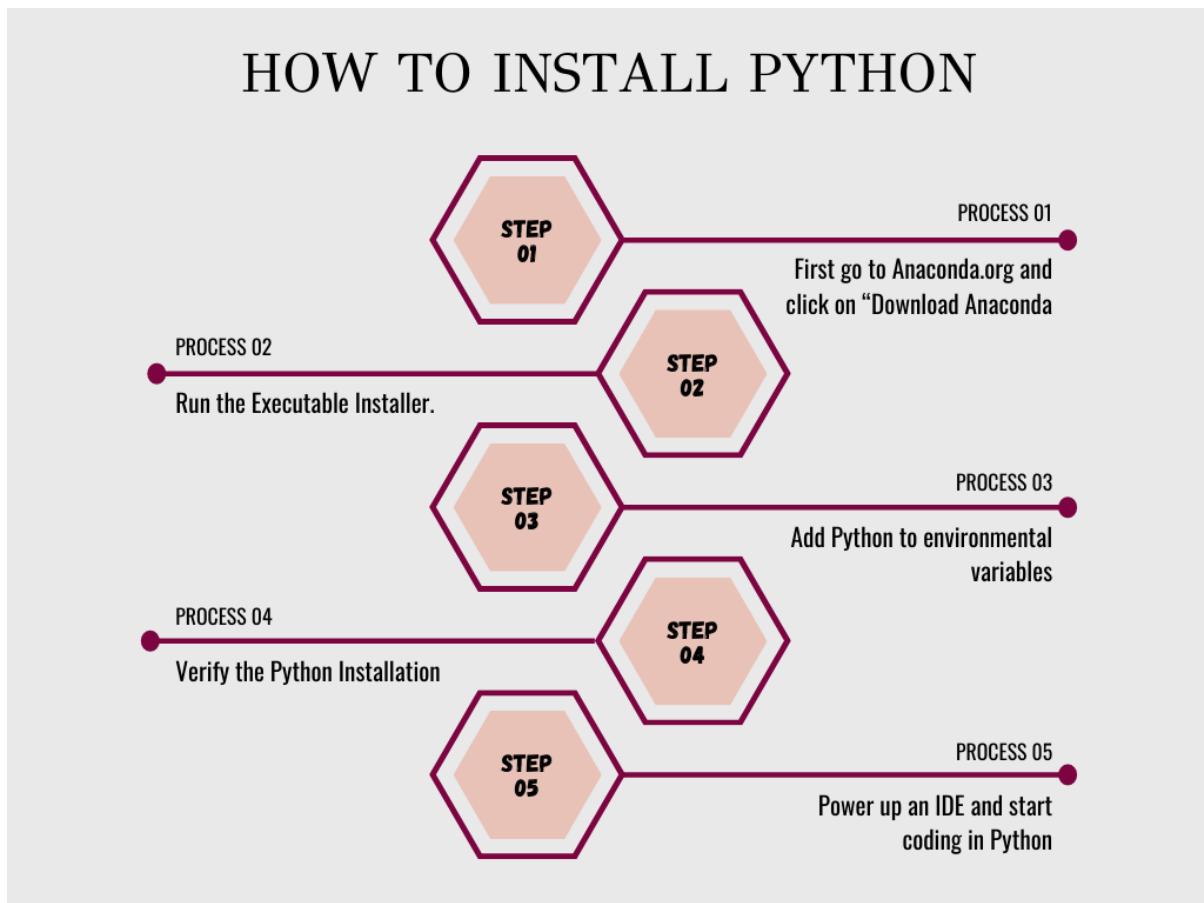
2. Why Python?

[Python](#) is a high-level, general-purpose programming language. Python is a programming language that may be used to create desktop GUI apps, websites, and online applications. As a high-level programming language, Python also allows you to concentrate on the application's essential functionality while handling routine programming duties. The basic grammar limitations of the programming language make it considerably easier to maintain the code base intelligible and the application manageable.

3. How to Install Python?

To Install Python, go to [Anaconda.org](#) and click on “Download Anaconda”. Here, you can download the latest version of Python. After Python is installed, it is a pretty straightforward process. The next step is to power up an IDE and start coding in Python. If you wish to learn more about the process, check out this [Python Tutorial](#). Check out [How to install python](#).

Check out this pictorial representation of python installation.



4. What are the applications of Python?

Python is notable for its general-purpose character, which allows it to be used in practically any software development sector. Python may be found in almost every new field. It is the most popular programming language and may be used to create any application.

– Web Applications

We can use Python to develop web applications. It contains HTML and XML libraries, JSON libraries, email processing libraries, request libraries, [beautiful soup](#) libraries, Feedparser libraries, and other internet protocols. Instagram uses Django, a Python web framework.

– Desktop GUI Applications

The Graphical User Interface (GUI) is a user interface that allows for easy interaction with any programme. Python contains the Tk GUI framework for creating user interfaces.

– Console-based Application

The command-line or shell is used to execute console-based programmes. These are computer programmes that are used to carry out orders. This type of programme was more common in the previous generation of computers. It is well-known for its REPL, or Read-Eval-Print Loop, which makes it ideal for command-line applications.

Python has a number of free libraries and modules that help in the creation of command-line applications. To read and write, the appropriate IO libraries are used. It has capabilities for processing parameters and generating console help text built-in. There are additional advanced libraries that may be used to create standalone console applications.

– Software Development

Python is useful for the software development process. It's a support language that may be used to establish control and management, testing, and other things.

- SCons are used to build control.
- Continuous compilation and testing are automated using Buildbot and Apache Gumps.

– Scientific and Numeric

This is the time of artificial intelligence, in which a machine can execute tasks as well as a person can. Python is an excellent programming language for artificial intelligence and machine learning applications. It has a number of scientific and mathematical libraries that make doing difficult computations simple.

Putting machine learning algorithms into practice requires a lot of arithmetic. Numpy, Pandas, Scipy, Scikit-learn, and other scientific and numerical [Python libraries](#) are available. If you know how to use Python, you'll be able to import libraries on top of the code. A few prominent machine library frameworks are listed below.

- SciPy
- [Scikit learn](#)
- NumPy
- Pandas
- Matplotlib

– Business Applications

Standard apps are not the same as business applications. This type of program necessitates a lot of scalability and readability, which Python gives.

Oddo is a Python-based all-in-one application that offers a wide range of business applications. The commercial application is built on the Tryton platform, which is provided by Python.

– Audio or Video-based Applications

Python is a versatile programming language that may be used to construct multimedia applications. TimPlayer, cplay, and other multimedia programmes written in Python are examples.

– 3D CAD Applications

Engineering-related architecture is designed using CAD (Computer-aided design). It's used to create a three-dimensional visualization of a system component. The following features in Python can be used to develop a 3D CAD application:

- Fandango (Popular)
- CAMVOX
- HeeksCNC
- AnyCAD
- RCAM

– Enterprise Applications

Python may be used to develop apps for usage within a business or organization. OpenERP, Tryton, Picalo all these real-time applications are examples.

– Image Processing Application

Python has a lot of libraries for working with pictures. The picture can be altered to our specifications. [OpenCV](#), Pillow, and SimpleITK are all image processing libraries present in python. In this topic, we've covered a wide range of applications in which Python plays a critical part in their development. We'll study more about Python principles in the upcoming tutorial.

5. What are the advantages of Python?

Python is a general-purpose dynamic programming language that is high-level and interpreted. Its architectural framework prioritizes code readability and utilizes indentation extensively.

- Third-party modules are present.
- Several support libraries are available (NumPy for numerical calculations, Pandas for data analytics, etc)
- Community development and open source
- Adaptable, simple to read, learn, and write
- Data structures that are pretty easy to work on
- High-level language
- The language that is dynamically typed (No need to mention data type based on the value assigned, it takes data type)
- Object-oriented programming language
- Interactive and transportable
- Ideal for prototypes since it allows you to add additional features with minimal code.
- Highly Effective
- Internet of Things (IoT) Possibilities
- Portable Interpreted Language across Operating Systems
- Since it is an interpreted language it executes any code line by line and throws an error if it finds something missing.
- Python is free to use and has a large open-source community.
- Python has a lot of support for libraries that provide numerous functions for doing any task at hand.
- One of the best features of Python is its portability: it can and does run on any platform without having to change the requirements.
- Provides a lot of functionality in lesser lines of code compared to other programming languages like Java, C++, etc.

6. What are the key features of Python?

Python is one of the most popular programming languages used by data scientists and AIML professionals. This popularity is due to the following key features of Python:

- Python is easy to learn due to its clear syntax and readability
- Python is easy to interpret, making debugging easy
- Python is free and Open-source
- It can be used across different languages
- It is an object-oriented language that supports concepts of classes

- It can be easily integrated with other languages like C++, Java, and more

7. What do you mean by Python literals?

A literal is a simple and direct form of expressing a value. Literals reflect the primitive type options available in that language. Integers, floating-point numbers, Booleans, and character strings are some of the most common forms of literal. Python supports the following literals:

Literals in Python relate to the data that is kept in a variable or constant. There are several types of literals present in Python

String Literals: It's a sequence of characters wrapped in a set of codes. Depending on the number of quotations used, there can be single, double, or triple strings. Single characters enclosed by single or double quotations are known as character literals.

Numeric Literals: These are unchangeable numbers that may be divided into three types: integer, float, and complex.

Boolean Literals: True or False, which signify ‘1’ and ‘0,’ respectively, can be assigned to them.

Special Literals: It's used to categorize fields that have not been generated. ‘None’ is the value that is used to represent it.

- String literals: “halo” , ‘12345’
- Int literals: 0,1,2,-1,-2
- Long literals: 89675L
- Float literals: 3.14
- Complex literals: 12j
- Boolean literals: True or False
- Special literals: None
- Unicode literals: u”hello”
- List literals: [], [5, 6, 7]
- Tuple literals: (), (9,), (8, 9, 0)
- Dict literals: {}, {'x':1}
- Set literals: {8, 9, 10}

8. What type of language is Python?

Python is an interpreted, interactive, object-oriented programming language. Classes, modules, exceptions, dynamic typing, and extremely high-level dynamic data types are all present.

Python is an interpreted language with dynamic typing. Because the code is not converted to a binary form, these languages are sometimes referred to as “scripting” languages. While I say dynamically typed, I’m referring to the fact that types don’t have to be stated when coding; the interpreter finds them out at runtime.

The readability of Python’s concise, easy-to-learn syntax is prioritized, lowering software maintenance costs. Python provides modules and packages, allowing for programme modularity and code reuse. The Python interpreter and its comprehensive standard library are free to download and distribute in source or binary form for all major platforms.

9. How is Python an interpreted language?

An interpreter takes your code and executes (does) the actions you provide, produces the variables you specify, and performs a lot of behind-the-scenes work to ensure it works smoothly or warns you about issues.

Python is not an interpreted or compiled language. The implementation’s attribute is whether it is interpreted or compiled. Python is a bytecode (a collection of interpreter-readable instructions) that may be interpreted in a variety of ways.

The source code is saved in a *.py file*.

Python generates a set of instructions for a virtual machine from the source code. This intermediate format is known as “bytecode,” and it is created by compiling .py source code into .pyc, which is bytecode. This bytecode can then be interpreted by the standard CPython interpreter or PyPy’s JIT (Just in Time compiler).

Python is known as an interpreted language because it uses an interpreter to convert the code you write into a language that your computer’s processor can understand. You will later download and utilise the Python interpreter to be able to create Python code and execute it on your own computer when working on a project.

10. What is pep 8?

PEP 8, often known as PEP8 or PEP-8, is a document that outlines best practices and recommendations for writing Python code. It was written in 2001 by Guido van Rossum, Barry Warsaw, and Nick Coghlan. The main goal of PEP 8 is to make Python code more readable and consistent.

Python Enhancement Proposal (PEP) is an acronym for Python Enhancement Proposal, and there are numerous of them. A Python Enhancement Proposal (PEP) is a document that explains new features suggested for Python and details elements of Python for the community, such as design and style.

11. What is namespace in Python?

In Python, a namespace is a system that assigns a unique name to each and every object. A variable or a method might be considered an object. Python has its own namespace, which is kept in the form of a Python dictionary. Let's look at a directory-file system structure in a computer as an example. It should go without saying that a file with the same name might be found in numerous folders. However, by supplying the absolute path of the file, one may be routed to it if desired.

A namespace is essentially a technique for ensuring that all of the names in a programme are distinct and may be used interchangeably. You may already be aware that everything in Python is an object, including strings, lists, functions, and so on. Another notable thing is that Python uses dictionaries to implement namespaces. A name-to-object mapping exists, with the names serving as keys and the objects serving as values. The same name can be used by many namespaces, each mapping it to a distinct object. Here are a few namespace examples:

Local Namespace: This namespace stores the local names of functions. This namespace is created when a function is invoked and only lives till the function returns.

Global Namespace: Names from various imported modules that you are utilizing in a project are stored in this namespace. It's formed when the module is added to the project and lasts till the script is completed.

Built-in Namespace: This namespace contains the names of built-in functions and exceptions.

12. What is PYTHON PATH?

PYTHONPATH is an environment variable that allows the user to add additional folders to the sys.path directory list for Python. In a nutshell, it is an environment variable that is set before the start of the Python interpreter.

13. What are Python modules?

A Python module is a collection of Python commands and definitions in a single file. In a module, you may specify functions, classes, and variables. A module can also include executable code. When code is organized into modules, it is easier to understand and use. It also logically organizes the code.

14. What are local variables and global variables in Python?

Local variables are declared inside a function and have a scope that is confined to that function alone, whereas global variables are defined outside of any function and have a global scope. To put it another way, local variables are only available within the function in which they were created, but global variables are accessible across the programme and throughout each function.

Local Variables

Local variables are variables that are created within a function and are exclusive to that function. Outside of the function, it can't be accessed.

Global Variables

Global variables are variables that are defined outside of any function and are available throughout the programme, that is, both inside and outside of each function.

15. Explain what Flask is and its benefits?

Flask is an open-source web framework. [Flask](#) is a set of tools, frameworks, and technologies for building online applications. A web page, a wiki, a huge web-based calendar software, or a commercial website is used to build this web app. Flask is a micro-framework, which means it doesn't rely on other libraries too much.

Benefits:

There are several compelling reasons to utilize Flask as a web application framework. Like-

- Unit testing support that is incorporated
- There's a built-in development server as well as a rapid debugger.
- Restful request dispatch with a Unicode basis
- The use of cookies is permitted.
- Templating WSGI 1.0 compatible jinja2
- Additionally, the flask gives you complete control over the progress of your project.
- HTTP request processing function
- Flask is a lightweight and versatile web framework that can be easily integrated with a few extensions.
- You may use your favorite device to connect. The main API for ORM Basic is well-designed and organized.
- Extremely adaptable
- In terms of manufacturing, the flask is easy to use.

16. Is Django better than Flask?

Django is more popular because it has plenty of functionality out of the box, making complicated applications easier to build. Django is best suited for larger projects with a lot of features. The features may be overkill for lesser applications.

If you're new to web programming, Flask is a fantastic place to start. Many websites are built with Flask and receive a lot of traffic, although not as much as Django-based websites. If you want precise control, you should use flask, whereas a Django developer relies on a large community to produce unique websites.

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

Flask is a “micro framework” designed for smaller applications with less requirements. Pyramid and Django are both geared at larger projects, but they approach extension and flexibility in different ways.

A pyramid is designed to be flexible, allowing the developer to use the best tools for their project. This means that the developer may choose the database, URL structure, templating style, and other options. Django aspires to include all of the batteries that a web application would require, so programmers simply need to open the box and start working, bringing in Django’s many components as they go.

Django includes an ORM by default, but Pyramid and Flask provide the developer control over how (and whether) their data is stored. SQLAlchemy is the most popular ORM for non-Django web apps, but there are lots of alternative options, ranging from DynamoDB and MongoDB to simple local persistence like LevelDB or regular SQLite. Pyramid is designed to work with any sort of persistence layer, even those that have yet to be conceived.

Django	Pyramid	Flask
It is a python framework.	It is the same as Django	It is a micro-framework.
It is used to build large applications.	It is the same as Django	It is used to create a small application.
It includes an ORM.	It provides flexibility and the right tools.	It does not require external libraries.

18. Discuss Django architecture

Django has an MVC (Model-View-Controller) architecture, which is divided into three parts:

1. Model

The Model, which is represented by a database, is the logical data structure that underpins the whole programme (generally relational databases such as MySql, Postgres).

2. View

The View is the user interface, or what you see when you visit a website in your browser. HTML/CSS/Javascript files are used to represent them.

3. Controller

The Controller is the link between the view and the model, and it is responsible for transferring data from the model to the view.

Your application will revolve around the model using MVC, either displaying or altering it.

19. Explain Scope in Python?

Think of scope as the father of a family; every object works within a scope. A formal definition would be this is a block of code under which no matter how many objects you declare they remain relevant. A few examples of the same are given below:

- **Local Scope:** When you create a variable inside a function that belongs to the local scope of that function itself and it will only be used inside that function.

Example:

```
def harshit_fun():

y = 100

print (y)
```

```
harshit_func()
```

```
100
```

- **Global Scope:** When a variable is created inside the main body of python code, it is called the global scope. The best part about global scope is they are accessible within any part of the python code from any scope be it global or local.

Example:

```
y = 100
```

```
def harshit_func():
```

```
print (y)

harshit_func()

print (y)
```

- **Nested Function:** This is also known as a function inside a function, as stated in the example above in local scope variable y is not available outside the function but within any function inside another function.

Example:

```
def first_func():

y = 100

def nested_func1():

print(y)

nested_func1()

first_func()
```

- **Module Level Scope:** This essentially refers to the global objects of the current module accessible within the program.
- **Outermost Scope:** This is a reference to all the built-in names that you can call in the program.

20. List the common built-in data types in Python?

Given below are the most commonly used built-in datatypes :

Numbers: Consists of integers, floating-point numbers, and complex numbers.

List: We have already seen a bit about lists, to put a formal definition a list is an ordered sequence of items that are mutable, also the elements inside lists can belong to different data types.

Example:

```
list = [100, "Great Learning", 30]
```

Tuples: This too is an ordered sequence of elements but unlike lists tuples are immutable meaning it cannot be changed once declared.

Example:

```
tup_2 = (100, "Great Learning", 20)
```

String: This is called the sequence of characters declared within single or double quotes.

Example:

```
"Hi, I work at great learning"
```

```
'Hi, I work at great learning'
```

Sets: Sets are basically collections of unique items where order is not uniform.

Example:

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

Dictionary: A dictionary always stores values in key and value pairs where each value can be accessed by its particular key.

Example:

```
[12] harshit = {1:'video_games', 2:'sports', 3:'content'}
```

Boolean: There are only two boolean values: **True** and **False**

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

The attributes of a class are also called variables. There are three access modifiers in Python for variables, namely

- a. public** – The variables declared as public are accessible everywhere, inside or outside the class.

b. private – The variables declared as private are accessible only within the current class.

c. protected – The variables declared as protected are accessible only within the current package.

Attributes are also classified as:

- **Local attributes** are defined within a code-block/method and can be accessed only within that code-block/method.
- **Global attributes** are defined outside the code-block/method and can be accessible everywhere.

```
class Mobile:  
  
    m1 = "Samsung Mobiles" //Global attributes  
  
    def price(self):  
  
        m2 = "Costly mobiles" //Local attributes  
  
        return m2  
  
    Sam_m = Mobile()  
  
print(Sam_m.m1)
```

22. What are Keywords in Python?

Keywords in Python are reserved words that are used as identifiers, function names, or variable names. They help define the structure and syntax of the language.

There are a total of 33 keywords in Python 3.7 which can change in the next version, i.e., Python 3.8. A list of all the keywords is provided below:

Keywords in Python:

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

23. What is the difference between lists and tuples in Python?

List and tuple are [data structures in Python](#) that may store one or more objects or values. Using square brackets, you may build a list to hold numerous objects in one variable. Tuples, like arrays, may hold numerous items in a single variable and are defined with parenthesis.

Lists	Tuples
Lists are mutable.	Tuples are immutable.
The impacts of iterations are Time Consuming.	Iterations have the effect of making things go faster.
The list is more convenient for actions like insertion and deletion.	The items may be accessed using the tuple data type.

Lists take up more memory.	When compared to a list, a tuple uses less memory.
There are numerous techniques built into lists.	There aren't many built-in methods in Tuple.
Changes and faults that are unexpected are more likely to occur.	It is difficult to take place in a tuple.
They consume a lot of memory given the nature of this data structure	They consume less memory
Syntax: list = [100, "Great Learning", 30]	Syntax: tup_2 = (100, "Great Learning", 20)

24. How can you concatenate two tuples?

Let's say we have two tuples like this ->

tup1 = (1,"a",True)

tup2 = (4,5,6)

Concatenation of tuples means that we are adding the elements of one tuple at the end of another tuple.

Now, let's go ahead and concatenate tuple2 with tuple1:

Code:

```
tup1=(1,"a",True)
```

```
tup2=(4,5,6)
```

```
tup1+tup2
```

All you have to do is, use the ‘+’ operator between the two tuples and you’ll get the concatenated result.

Similarly, let’s concatenate tuple1 with tuple2:

Code:

```
tup1=(1,"a",True)  
tup2=(4,5,6)  
tup2+tup1
```

25. What are functions in Python?

Ans: Functions in Python refer to blocks that have organized, and reusable codes to perform single, and related events. Functions are important to create better modularity for applications that reuse a high degree of coding. Python has a number of built-in functions like print(). However, it also allows you to create user-defined functions.

26. How can you initialize a 5*5 numpy array with only zeroes?

We will be using the `.zeros()` method.

```
import numpy as np  
  
n1=np.zeros((5,5))  
  
n1
```

Use `np.zeros()` and pass in the dimensions inside it. Since we want a 5*5 matrix, we will pass (5,5) inside the `.zeros()` method.

27. What are Pandas?

Pandas is an open-source python library that has a very rich set of data structures for data-based operations. Pandas with their cool features fit in every role of data operation, whether it be academics or solving complex business

problems. Pandas can deal with a large variety of files and are one of the most important tools to have a grip on.

28. What are data frames?

A pandas dataframe is a data structure in pandas that is mutable. Pandas have support for heterogeneous data which is arranged across two axes. (rows and columns).

Reading files into pandas:-

12	Import pandas as pddf=p.read_csv("mydata.csv")
----	--

Here, df is a pandas data frame. read_csv() is used to read a comma-delimited file as a dataframe in pandas.

29. What is a Pandas Series?

Series is a one-dimensional panda's data structure that can data of almost any type. It resembles an excel column. It supports multiple operations and is used for single-dimensional data operations.

Creating a series from data:

Code:

```
import pandas as pd  
  
data=["1",2,"three",4.0]  
  
series=pd.Series(data)  
  
print(series)  
  
print(type(series))
```

30. What do you understand about pandas groupby?

A pandas groupby is a feature supported by pandas that are used to split and group an object. Like the sql/mysql/oracle groupby it is used to group data by classes, and entities which can be further used for aggregation. A dataframe can be grouped by one or more columns.

Code:

```
df = pd.DataFrame({'Vehicle':['Etios','Lamborghini','Apache200','Pulsar200'],
'Type':["car","car","motorcycle","motorcycle"]})
```

```
df
```

To perform groupby type the following code:

```
df.groupby('Type').count()
```

31. How to create a dataframe from lists?

To create a dataframe from lists,

- 1) create an empty dataframe
- 2) add lists as individuals columns to the list

Code:

```
df=pd.DataFrame()  
  
bikes=["bajaj","tvs","herohonda","kawasaki","bmw"]  
  
cars=["lamborghini","masserati","ferrari","hyundai","ford"]  
  
df["cars"]=cars  
  
df["bikes"]=bikes  
  
df
```

32. How to create a data frame from a dictionary?

A dictionary can be directly passed as an argument to the DataFrame() function to create the data frame.

Code:

```
import pandas as pd

bikes=["bajaj","tvs","herohonda","kawasaki","bmw"]

cars=["lamborghini","masserati","ferrari","hyundai","ford"]

d={"cars":cars,"bikes":bikes}

df=pd.DataFrame(d)

df
```

33. How to combine dataframes in pandas?

Two different data frames can be stacked either horizontally or vertically by the concat(), append(), and join() functions in pandas.

Concat works best when the data frames have the same columns and can be used for concatenation of data having similar fields and is basically vertical stacking of dataframes into a single dataframe.

Append() is used for horizontal stacking of data frames. If two tables(dataframes) are to be merged together then this is the best concatenation function.

Join is used when we need to extract data from different dataframes which are having one or more common columns. The stacking is horizontal in this case.

Before going through the questions, here's a quick video to help you refresh your memory on Python.

34. What kind of joins does pandas offer?

Pandas have a left join, inner join, right join, and outer join.

35. How to merge dataframes in pandas?

Merging depends on the type and fields of different dataframes being merged. If data has similar fields data is merged along axis 0 else they are merged along axis 1.

36. Give the below dataframe drop all rows having Nan.

The dropna function can be used to do that.

```
df.dropna(inplace=True)
```

```
df
```

37. How to access the first five entries of a dataframe?

By using the head(5) function we can get the top five entries of a dataframe. By default df.head() returns the top 5 rows. To get the top n rows df.head(n) will be used.

38. How to access the last five entries of a dataframe?

By using the tail(5) function we can get the top five entries of a dataframe. By default df.tail() returns the top 5 rows. To get the last n rows df.tail(n) will be used.

39. How to fetch a data entry from a pandas dataframe using a given value in index?

To fetch a row from a dataframe given index x, we can use loc.

Df.loc[10] where 10 is the value of the index.

Code:

```
import pandas as pd  
  
bikes=["bajaj","tvs","herohonda","kawasaki","bmw"]
```

```
cars=["lamborghini","masserati","ferrari","hyundai","ford"]
```

```
d={"cars":cars,"bikes":bikes}
```

```
df=pd.DataFrame(d)
```

```
a=[10,20,30,40,50]
```

```
df.index=a
```

```
df.loc[10]
```

40. What are comments and how can you add comments in Python?

Comments in Python refer to a piece of text intended for information. It is especially relevant when more than one person works on a set of codes. It can be used to analyse code, leave feedback, and debug it. There are two types of comments which includes:

1. Single-line comment
2. Multiple-line comment

Codes needed for adding a comment

```
#Note –single line comment
```

```
"""Note
```

```
Note
```

```
Note""""—multiline comment
```

41. What is a dictionary in Python? Give an example.

A Python dictionary is a collection of items in no particular order. Python dictionaries are written in curly brackets with keys and values. Dictionaries are optimised to retrieve values for known keys.

Example

```
d={"a":1,"b":2}
```

42. What is the difference between a tuple and a dictionary?

One major difference between a tuple and a dictionary is that a dictionary is mutable while a tuple is not. Meaning the content of a dictionary can be changed without changing its identity, but in a tuple, that's not possible.

43. Find out the mean, median and standard deviation of this numpy array -> np.array([1,5,3,100,4,48])

```
import numpy as np

n1=np.array([10,20,30,40,50,60])

print(np.mean(n1))

print(np.median(n1))

print(np.std(n1))
```

44. What is a classifier?

A classifier is used to predict the class of any data point. Classifiers are special hypotheses that are used to assign class labels to any particular data point. A classifier often uses training data to understand the relation between input variables and the class. Classification is a method used in supervised learning in Machine Learning.

45. In Python how do you convert a string into lowercase?

All the upper cases in a string can be converted into lowercase by using the method: string.lower()

ex:

```
string = ‘GREATLEARNING’ print(string.lower())
```

o/p: greatlearning

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

One of the ways we can get a list of keys is by using: dict.keys()

This method returns all the available keys in the dictionary.

```
dict = {1:a, 2:b, 3:c} dict.keys()
```

```
o/p: [1, 2, 3]
```

47. How can you capitalize the first letter of a string?

We can use the **capitalize()** function to capitalize the first character of a string. If the first character is already in the capital then it returns the original string.

Syntax:

```
1 string_name.capitalize()
```

ex:

```
n = "greatlearning" print(n.capitalize())
```

```
o/p: Greatlearning
```

48. How can you insert an element at a given index in Python?

Python has an inbuilt function called the **insert()** function.

It can be used used to insert an element at a given index.

Syntax:

```
1 list_name.insert(index, element)
```

ex:

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

```
#insert 10 at 6th index
```

```
list.insert(6, 10)
```

```
o/p: [0,1,2,3,4,5,10,6,7]
```

49. How will you remove duplicate elements from a list?

There are various methods to remove duplicate elements from a list. But, the most common one is, converting the list into a set by using the set() function and using the list() function to convert it back to a list if required.

ex:

```
list0 = [2, 6, 4, 7, 4, 6, 7, 2]
```

```
list1 = list(set(list0)) print ("The list without duplicates : " + str(list1))
```

o/p: The list without duplicates : [2, 4, 6, 7]

50. What is recursion?

Recursion is a function calling itself one or more times in its body. One very important condition a recursive function should have to be used in a program is, it should terminate, else there would be a problem of an infinite loop.

51. Explain Python List Comprehension.

List comprehensions are used for transforming one list into another list. Elements can be conditionally included in the new list and each element can be transformed as needed. It consists of an expression leading to a for clause, enclosed in brackets.

For ex:

```
list = [i for i in range(1000)]
```

```
print list
```

52. What is the bytes() function?

The bytes() function returns a bytes object. It is used to convert objects into bytes objects or create empty bytes objects of the specified size.

53. What are the different types of operators in Python?

Python has the following basic operators:

Arithmetic (Addition(+), Subtraction(-), Multiplication(*), Division(/), Modulus(%)), **Relational** (<, >, <=, >=, ==, !=,), **Assignment** (=, +=, -=, /=, *=, %=), **Logical** (and, or not), Membership, Identity, and Bitwise Operators

54. What is the ‘with statement’?

The “with” statement in python is used in exception handling. A file can be opened and closed while executing a block of code, containing the “with” statement., without using the close() function. It essentially makes the code much easier to read.

55. What is a map() function in Python?

The map() function in Python is used for applying a function on all elements of a specified iterable. It consists of two parameters, function and iterable. The function is taken as an argument and then applied to all the elements of an iterable(passed as the second argument). An object list is returned as a result.

```
def add(n):  
  
    return n + n  
number= (15, 25, 35, 45)  
  
res= map(add, num)  
  
print(list(res))
```

o/p: 30,50,70,90

56. What is __init__ in Python?

`_init_` methodology is a reserved method in Python aka constructor in OOP. When an object is created from a class and `_init_` methodology is called to access the class attributes.

57. What are the tools present to perform static analysis?

The two static analysis tools used to find bugs in Python are Pychecker and Pylint. Pychecker detects bugs from the source code and warns about its style and complexity. While Pylint checks whether the module matches upto a coding standard.

58. What is pass in Python?

Pass is a statement that does nothing when executed. In other words, it is a Null statement. This statement is not ignored by the interpreter, but the statement results in no operation. It is used when you do not want any command to execute but a statement is required.

59. How can an object be copied in Python?

Not all objects can be copied in Python, but most can. We can use the “=” operator to copy an object to a variable.

ex:

```
var=copy.copy(obj)
```

60. How can a number be converted to a string?

The inbuilt function str() can be used to convert a number to a string.

61. What are modules and packages in Python?

Modules are the way to structure a program. Each Python program file is a module, importing other attributes and objects. The folder of a program is a package of modules. A package can have modules or subfolders.

62. What is the object() function in Python?

In Python, the object() function returns an empty object. New properties or methods cannot be added to this object.

63. What is the difference between NumPy and SciPy?

NumPy stands for Numerical Python while SciPy stands for Scientific Python. NumPy is the basic library for defining arrays and simple mathematical problems, while SciPy is used for more complex problems like numerical integration and optimization and machine learning and so on.

64. What does len() do?

len() is used to determine the length of a string, a list, an array, and so on.

ex:

```
str = "greatlearning"
```

```
print(len(str))
```

o/p: 13

65. Define encapsulation in Python?

Encapsulation means binding the code and the data together. A Python class for example.

66. What is the type () in Python?

`type()` is a built-in method that either returns the type of the object or returns a new type of object based on the arguments passed.

ex:

```
a = 100
```

```
type(a)
```

o/p: int

67. What is the split() function used for?

Split function is used to split a string into shorter strings using defined separators.

```
letters= ("A, B, C")
```

```
n = text.split(",")
```

```
print(n)
```

o/p: ['A', 'B', 'C']

68. What are the built-in types does python provide?

Python has following built-in data types:

Numbers: Python identifies three types of numbers:

1. Integer: All positive and negative numbers without a fractional part
2. Float: Any real number with floating-point representation
3. Complex numbers: A number with a real and imaginary component represented as $x+yi$. x and y are floats and j is $\sqrt{-1}$ (square root of -1 called an imaginary number)

Boolean: The Boolean data type is a data type that has one of two possible values i.e. True or False. Note that 'T' and 'F' are capital letters.

String: A string value is a collection of one or more characters put in single, double or triple quotes.

List: A list object is an ordered collection of one or more data items that can be of different types, put in square brackets. A list is mutable and thus can be modified, we can add, edit or delete individual elements in a list.

Set: An unordered collection of unique objects enclosed in curly brackets

Frozen set: They are like a set but immutable, which means we cannot modify their values once they are created.

Dictionary: A dictionary object is unordered in which there is a key associated with each value and we can access each value through its key. A collection of such pairs is enclosed in curly brackets. For example {‘First Name’: ‘Tom’, ‘last name’: ‘Hardy’} Note that Number values, strings, and tuples are immutable while List or Dictionary objects are mutable.

69. What is docstring in Python?

Python docstrings are the string literals enclosed in triple quotes that appear right after the definition of a function, method, class, or module. These are generally used to describe the functionality of a particular function, method, class, or module. We can access these docstrings using the `__doc__` attribute.

Here is an example:

```
def square(n):
```

```
'''Takes in a number n, returns the square of n'''\n\nreturn n**2\n\nprint(square.__doc__)
```

Ouput: Takes in a number n, returns the square of n.

70. How to Reverse a String in Python?

In Python, there are no in-built functions that help us reverse a string. We need to make use of an array slicing operation for the same.

1	str_reverse = string[::-1]
---	----------------------------

71. How to check the Python Version in CMD?

To check the Python Version in CMD, press CMD + Space. This opens Spotlight. Here, type “terminal” and press enter. To execute the command, type python –version or python -V and press enter. This will return the python version in the next line below the command.

72. Is Python case sensitive when dealing with identifiers?

Yes. Python is case-sensitive when dealing with identifiers. It is a case-sensitive language. Thus, variable and Variable would not be the same.

Python Interview Questions for Experienced

This section on Python Interview Questions for Experienced covers 20+ questions that are commonly asked during the interview process for landing a job as a Python experienced professional. These commonly asked questions can help you brush up your skills and know what to expect in your upcoming interviews.

73. How to create a new column in pandas by using values from other columns?

We can perform column based mathematical operations on a pandas dataframe. Pandas columns containing numeric values can be operated upon by operators.

Code:

```
import pandas as pd  
  
a=[1,2,3]  
  
b=[2,3,5]  
  
d={"col1":a,"col2":b}  
  
df=pd.DataFrame(d)  
  
df["Sum"]=df["col1"]+df["col2"]  
  
df["Difference"]=df["col1"]-df["col2"]  
  
df
```

Output:

	col1	col2	Sum	Difference
0	1	2	3	-1
1	2	3	5	-1
2	3	5	8	-2

74. What are the different functions that can be used by grouby in pandas ?

grouby() in pandas can be used with multiple aggregate functions. Some of which are sum(),mean(), count(),std().

Data is divided into groups based on categories and then the data in these individual groups can be aggregated by the aforementioned functions.

75. How to delete a column or group of columns in pandas? Given the below dataframe drop column “col1”.

drop() function can be used to delete the columns from a dataframe.

```
d={"col1":[1,2,3],"col2":["A","B","C"]}
```

```
df=pd.DataFrame(d)
```

```
df=df.drop(["col1"],axis=1)
```

```
df
```

76. Given the following data frame drop rows having column values as A.

Code:

```
d={"col1":[1,2,3],"col2":["A","B","C"]}
```

```
df=pd.DataFrame(d)
```

```
df.dropna(inplace=True)
```

```
df=df[df.col1!=1]
```

```
df
```

77. What is Reindexing in pandas?

Reindexing is the process of re-assigning the index of a pandas dataframe.

Code:

```
import pandas as pd
```

```
bikes=["bajaj","tvs","herohonda","kawasaki","bmw"]
```

```
cars=["lamborghini","masserati","ferrari","hyundai","ford"]
```

```
d={"cars":cars,"bikes":bikes}
```

```
df=pd.DataFrame(d)
```

```
a=[10,20,30,40,50]
```

```
df.index=a
```

```
df
```

78. What do you understand about the lambda function? Create a lambda function which will print the sum of all the elements in this list -> [5, 8, 10, 20, 50, 100]

Lambda functions are anonymous functions in Python. They are defined using the keyword lambda. Lambda functions can take any number of arguments, but they can only have one expression.

```
from functools import reduce
```

```
sequences = [5, 8, 10, 20, 50, 100]
```

```
sum = reduce (lambda x, y: x+y, sequences)
```

```
print(sum)
```

79. What is vstack() in numpy? Give an example.

vstack() is a function to align rows vertically. All rows must have the same number of elements.

Code:

```
import numpy as np
```

```
n1=np.array([10,20,30,40,50])
```

```
n2=np.array([50,60,70,80,90])
```

```
print(np.vstack((n1,n2)))
```

80. How to remove spaces from a string in Python?

Spaces can be removed from a string in python by using strip() or replace() functions. Strip() function is used to remove the leading and trailing white spaces while the replace() function is used to remove all the white spaces in the string:

```
string.replace(" ",""))
ex1: str1= "great learning"
```

```
print (str.strip())
```

o/p: great learning

```
ex2: str2="great learning"
```

```
print (str.replace(" ",""))
```

o/p: greatlearning

81. Explain the file processing modes that Python supports.

There are three file processing modes in Python: read-only(r), write-only(w), read-write(rw) and append (a). So, if you are opening a text file in say, read mode. The preceding modes become “rt” for read-only, “wt” for write and so on. Similarly, a binary file can be opened by specifying “b” along with the file accessing flags (“r”, “w”, “rw” and “a”) preceding it.

82. What is pickling and unpickling?

Pickling is the process of converting a Python object hierarchy into a byte stream for storing it into a database. It is also known as serialization. Unpickling is the reverse of pickling. The byte stream is converted back into an object hierarchy.

83. How is memory managed in Python?

This is one of the most commonly asked python interview questions

Memory management in python comprises a private heap containing all objects and data structure. The heap is managed by the interpreter and the programmer does not have access to it at all. The Python memory manager does all the memory allocation. Moreover, there is an inbuilt garbage collector that recycles and frees memory for the heap space.

84. What is unittest in Python?

Unittest is a unit testing framework in Python. It supports sharing of setup and shutdown code for tests, aggregation of tests into collections, test automation, and independence of the tests from the reporting framework.

85. How do you delete a file in Python?

Files can be deleted in Python by using the command `os.remove(filename)` or `os.unlink(filename)`

86. How do you create an empty class in Python?

To create an empty class we can use the `pass` command after the definition of the class object. A `pass` is a statement in Python that does nothing.

87. What are Python decorators?

Decorators are functions that take another function as an argument to modify its behavior without changing the function itself. These are useful when we want to dynamically increase the functionality of a function without changing it.

Here is an example:

```
def smart_divide(func):

    def inner(a, b):
        print("Dividing", a, "by", b)

        if b == 0:
            print("Make sure Denominator is not zero")

        return

    return func(a, b)

    return inner

@smart_divide
```

```
def divide(a, b):  
  
    print(a/b)  
  
divide(1,0)
```

Here smart_divide is a decorator function that is used to add functionality to simple divide function.

88. What is a dynamically typed language?

Type checking is an important part of any programming language which is about ensuring minimum type errors. The type defined for variables are checked either at compile-time or run-time. When the type-check is done at compile time then it is called static typed language and when the type check is done at run time, it's called dynamically typed language.

1. In dynamic typed language the objects are bound with type by assignments at run time.
2. Dynamically typed programming languages produce less optimized code comparatively
3. In dynamically typed languages, types for variables need not be defined before using them. Hence, it can be allocated dynamically.

89. What is slicing in Python?

Slicing in Python refers to accessing parts of a sequence. The sequence can be any mutable and iterable object. slice() is a function used in Python to divide the given sequence into required segments.

There are two variations of using the slice function. Syntax for slicing in python:

1. slice(start,stop)
2. slice(start, stop, step)

Ex:

```
Str1 = ("g", "r", "e", "a", "t", "l", "e", "a", "r", "n", "i", "n", "g")
```

```
substr1 = slice(3, 5)
```

```
print(Str1[substr1])
```

```
//same code can be written in the following way also
```

```
Str1 =("g", "r", "e", "a", "t", "l", "e", "a", "r", "n", "i", "n", "g")
```

```
print(Str1[3,5])
```

```
Str1 =("g", "r", "e", "a", "t", "l", "e", "a", "r", "n", "i", "n", "g")
```

```
substr1 = slice(0, 14, 2)
```

```
print(Str1[substr1])
```

```
//same code can be written in the following way also
```

```
Str1 =("g", "r", "e", "a", "t", "l", "e", "a", "r", "n", "i", "n", "g")
```

```
print(Str1[0,14, 2])
```

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

Python Arrays and List both are ordered collections of elements and are mutable, but the difference lies in working with them

Arrays store heterogeneous data when imported from the array module, but arrays can store homogeneous data imported from the numpy module. But lists can store heterogeneous data, and to use lists, it doesn't have to be imported from any module.

```
import array as a1
```

```
array1 = a1.array('i', [1 , 2 ,5] )
```

```
print (array1)
```

Or,

```
import numpy as a2  
  
array2 = a2.array([5, 6, 9, 2])  
  
print(array2)
```

1. Arrays have to be declared before using it but lists need not be declared.
2. Numerical operations are easier to do on arrays as compared to lists.

91. What is Scope Resolution in Python?

The variable's accessibility is defined in python according to the location of the variable declaration, called the scope of variables in python. Scope Resolution refers to the order in which these variables are looked for a name to variable matching. Following is the scope defined in python for variable declaration.

- a. Local scope – The variable declared inside a loop, the function body is accessible only within that function or loop.
- b. Global scope – The variable is declared outside any other code at the topmost level and is accessible everywhere.
- c. Enclosing scope – The variable is declared inside an enclosing function, accessible only within that enclosing function.
- d. Built-in Scope – The variable declared inside the inbuilt functions of various modules of python has the built-in scope and is accessible only within that particular module.

The scope resolution for any variable is made in java in a particular order, and that order is

Local Scope -> enclosing scope -> global scope -> built-in scope

92. What are Dict and List comprehensions?

List comprehensions provide a more compact and elegant way to create lists than for-loops, and also a new list can be created from existing lists.

The syntax used is as follows:

1 a for a in iterator

Or,

1 a for a in iterator if condition

Ex:

```
list1 = [a for a in range(5)]
```

```
print(list1)
```

```
list2 = [a for a in range(5) if a < 3]
```

```
print(list2)
```

Dictionary comprehensions provide a more compact and elegant way to create a dictionary, and also, a new dictionary can be created from existing dictionaries.

The syntax used is:

1 {key: expression for an item in iterator}

Ex:

```
dict([(i, i*2) for i in range(5)])
```

93. What is the difference between xrange and range in Python?

range() and xrange() are inbuilt functions in python used to generate integer numbers in the specified range. The difference between the two can be understood if python version 2.0 is used because the python version 3.0 xrange() function is re-implemented as the range() function itself.

With respect to python 2.0, the difference between range and xrange function is as follows:

1. range() takes more memory comparatively
2. xrange(), execution speed is faster comparatively

- range () returns a list of integers and xrange() returns a generator object.

Example:

```
for i in range(1,10,2):  
  
    print(i)
```

94. What is the difference between .py and .pyc files?

.py are the source code files in python that the python interpreter interprets.

.pyc are the compiled files that are bytecodes generated by the python compiler, but .pyc files are only created for inbuilt modules/files.

Python Programming Interview Questions

Apart from having theoretical knowledge, having practical experience and knowing programming interview questions is a crucial part of the interview process. It helps the recruiters understand your hands-on experience. These are 45+ of the most commonly asked Python programming interview questions.

Here is a pictorial representation of how to generate the python programming output.

WHAT IS PYTHON PROGRAMMING ?



95. You have this covid-19 dataset below:

This is one of the most commonly asked python interview questions

From this dataset, how will you make a bar-plot for the top 5 states having maximum confirmed cases as of 17=07-2020?

sol:

```
#keeping only required columns
```

```
df = df[['Date', 'State/UnionTerritory','Cured','Deaths','Confirmed']]
```

```
#renaming column names
```

```
df.columns = ['date', 'state', 'cured', 'deaths', 'confirmed']
```

```
#current date
```

```
today = df[df.date == '2020-07-17']
```

```
#Sorting data w.r.t number of confirmed cases
```

```
max_confirmed_cases=today.sort_values(by="confirmed",ascending=False)
```

```
max_confirmed_cases
```

```
#Getting states with maximum number of confirmed cases
```

```
top_states_confirmed=max_confirmed_cases[0:5]
```

```
#Making bar-plot for states with top confirmed cases
```

```
sns.set(rc={'figure.figsize':(15,10)})
```

```
sns.barplot(x="state",y="confirmed",data=top_states_confirmed,hue="state")  
  
plt.show()
```

Code explanation:

We start off by taking only the required columns with this command:

```
df = df[['Date', 'State/UnionTerritory','Cured','Deaths','Confirmed']]
```

Then, we go ahead and rename the columns:

```
df.columns = ['date', 'state','cured','deaths','confirmed']
```

After that, we extract only those records, where the date is equal to 17th July:

```
today = df[df.date == '2020-07-17']
```

Then, we go ahead and select the top 5 states with maximum no. of covid cases:

```
max_confirmed_cases=today.sort_values(by="confirmed",ascending=False)  
  
max_confirmed_cases  
  
top_states_confirmed=max_confirmed_cases[0:5]
```

Finally, we go ahead and make a bar-plot with this:

```
sns.set(rc={'figure.figsize':(15,10)})  
  
sns.barplot(x="state",y="confirmed",data=top_states_confirmed,hue="state")  
  
plt.show()
```

Here, we are using the seaborn library to make the bar plot. The “State” column is mapped onto the x-axis and the “confirmed” column is mapped onto the y-axis. The color of the bars is determined by the “state” column.

96. From this covid-19 dataset:

How can you make a bar plot for the top 5 states with the most amount of deaths?

```
max_death_cases=today.sort_values(by="deaths",ascending=False)
```

```
max_death_cases
```

```
sns.set(rc={'figure.figsize':(15,10)})
```

```
sns.barplot(x="state",y="deaths",data=top_states_death,hue="state")
```

```
plt.show()
```

Code Explanation:

We start off by sorting our dataframe in descending order w.r.t the “deaths” column:

```
max_death_cases=today.sort_values(by="deaths",ascending=False)
```

```
Max_death_cases
```

Then, we go ahead and make the bar-plot with the help of seaborn library:

```
sns.set(rc={'figure.figsize':(15,10)})
```

```
sns.barplot(x="state",y="deaths",data=top_states_death,hue="state")
```

```
plt.show()
```

Here, we are mapping the “state” column onto the x-axis and the “deaths” column onto the y-axis.

97. From this covid-19 dataset:

How can you make a line plot indicating the confirmed cases with respect to date?

Sol:

```
maha = df[df.state == 'Maharashtra']

sns.set(rc={'figure.figsize':(15,10)})

sns.lineplot(x="date",y="confirmed",data=maha,color="g")

plt.show()
```

Code Explanation:

We start off by extracting all the records where the state is equal to “Maharashtra”:

```
maha = df[df.state == 'Maharashtra']
```

Then, we go ahead and make a line-plot using seaborn library:

```
sns.set(rc={'figure.figsize':(15,10)})

sns.lineplot(x="date",y="confirmed",data=maha,color="g")

plt.show()
```

Here, we map the “date” column onto the x-axis and the “confirmed” column onto the y-axis.

98. On this “Maharashtra” dataset:

How will you implement a linear regression algorithm with “date” as the independent variable and “confirmed” as the dependent variable? That is you have to predict the number of confirmed cases w.r.t date.

```
from sklearn.model_selection import train_test_split
```

```
maha['date']=maha['date'].map(dt.datetime.toordinal)
```

```
maha.head()
```

```
x=maha['date']
```

```
y=maha['confirmed']
```

```
x_train,x_test,y_train,y_test=train_test_split(x,y,test_size=0.3)
```

```
from sklearn.linear_model import LinearRegression
```

```
lr = LinearRegression()
```

```
lr.fit(np.array(x_train).reshape(-1,1),np.array(y_train).reshape(-1,1))

lr.predict(np.array([[737630]]))
```

Code solution:

We will start off by converting the date to ordinal type:

```
from sklearn.model_selection import train_test_split

maha['date']=maha['date'].map(dt.datetime.toordinal)
```

This is done because we cannot build the linear regression algorithm on top of the date column.

Then, we go ahead and divide the dataset into train and test sets:

```
x_train,x_test,y_train,y_test=train_test_split(x,y,test_size=0.3)
```

Finally, we go ahead and build the model:

```
from sklearn.linear_model import LinearRegression

lr = LinearRegression()

lr.fit(np.array(x_train).reshape(-1,1),np.array(y_train).reshape(-1,1))

lr.predict(np.array([[737630]]))
```

99. On this customer_churn dataset:

This is one of the most commonly asked python interview questions

Build a Keras sequential model to find out how many customers will churn out on the basis of tenure of customer?

```
from keras.models import Sequential
```

```
from keras.layers import Dense  
  
model = Sequential()  
  
model.add(Dense(12, input_dim=1, activation='relu'))  
  
model.add(Dense(8, activation='relu'))  
  
model.add(Dense(1, activation='sigmoid'))  
  
model.compile(loss='binary_crossentropy', optimizer='adam',  
metrics=['accuracy'])  
  
model.fit(x_train, y_train, epochs=150, validation_data=(x_test,y_test))  
  
y_pred = model.predict_classes(x_test)  
  
from sklearn.metrics import confusion_matrix
```

```
confusion_matrix(y_test,y_pred)
```

Code explanation:

We will start off by importing the required libraries:

```
from Keras.models import Sequential
```

```
from Keras.layers import Dense
```

Then, we go ahead and build the structure of the sequential model:

```
model = Sequential()  
  
model.add(Dense(12, input_dim=1, activation='relu'))  
  
model.add(Dense(8, activation='relu'))  
  
model.add(Dense(1, activation='sigmoid'))
```

Finally, we will go ahead and predict the values:

```
model.compile(loss='binary_crossentropy', optimizer='adam',  
metrics=['accuracy'])  
  
model.fit(x_train, y_train, epochs=150, validation_data=(x_test,y_test))  
  
y_pred = model.predict_classes(x_test)  
  
from sklearn.metrics import confusion_matrix  
  
confusion_matrix(y_test,y_pred)
```

100. On this iris dataset:

Build a decision tree classification model, where the dependent variable is “Species” and the independent variable is “Sepal.Length”.

```
y = iris[['Species']]
```

```
x = iris[['Sepal.Length']]  
  
from sklearn.model_selection import train_test_split  
  
x_train,x_test,y_train,y_test=train_test_split(x,y,test_size=0.4)  
  
from sklearn.tree import DecisionTreeClassifier  
  
dtc = DecisionTreeClassifier()  
  
dtc.fit(x_train,y_train)  
  
y_pred=dtc.predict(x_test)  
  
from sklearn.metrics import confusion_matrix  
  
confusion_matrix(y_test,y_pred)  
(22+7+9)/(22+2+0+7+7+11+1+1+9)
```

Code explanation:

We start off by extracting the independent variable and dependent variable:

```
y = iris[['Species']]  
x = iris[['Sepal.Length']]
```

Then, we go ahead and divide the data into train and test set:

```
from sklearn.model_selection import train_test_split  
  
x_train,x_test,y_train,y_test=train_test_split(x,y,test_size=0.4)
```

After that, we go ahead and build the model:

```
from sklearn.tree import DecisionTreeClassifier  
  
dtc = DecisionTreeClassifier()  
  
dtc.fit(x_train,y_train)  
  
y_pred=dtc.predict(x_test)
```

Finally, we build the confusion matrix:

```
from sklearn.metrics import confusion_matrix  
  
confusion_matrix(y_test,y_pred)  
  
(22+7+9)/(22+2+0+7+7+11+1+1+9)
```

101. On this iris dataset:

Build a decision tree regression model where the independent variable is “petal length” and dependent variable is “Sepal length”.

```
x= iris[['Petal.Length']]
```

```
y = iris[['Sepal.Length']]  
  
x_train,x_test,y_train,y_test=train_test_split(x,y,test_size=0.25)  
  
from sklearn.tree import DecisionTreeRegressor  
  
dtr = DecisionTreeRegressor()  
  
dtr.fit(x_train,y_train)  
  
y_pred=dtr.predict(x_test)  
  
y_pred[0:5]  
  
from sklearn.metrics import mean_squared_error  
  
mean_squared_error(y_test,y_pred)
```

102. How will you scrape data from the website “cricbuzz”?

```
import sys
```

```
import time
```

```
from bs4 import BeautifulSoup
```

```
import requests
```

```
import pandas as pd
```

```
try:
```

```
    #use the browser to get the url. This is suspicious command that might  
    blow up.
```

```
    page=requests.get('cricbuzz.com')          # this might throw an  
    exception if something goes wrong.
```

```
except Exception as e:                      # this describes what to do if an  
    exception is thrown
```

```
    error_type, error_obj, error_info = sys.exc_info()    # get the exception  
    information
```

```
print ('ERROR FOR LINK:',url) #print the link that cause the problem
```

```
print (error_type, 'Line:', error_info.tb_lineno) #print error info and line that threw the exception
```

```
#ignore this page. Abandon this and go back.
```

```
time.sleep(2)
```

```
soup=BeautifulSoup(page.text,'html.parser')
```

```
links=soup.find_all('span',attrs={'class':'w_tle'})
```

```
links
```

```
for i in links:
```

```
    print(i.text)
```

```
    print("\n")
```

103. Write a user-defined function to implement the central-limit theorem. You have to implement the central limit theorem on this “insurance” dataset:

You also have to build two plots on “Sampling Distribution of BMI” and “Population distribution of BMI”.

```
df = pd.read_csv('insurance.csv')
```

```
series1 = df.charges
```

```
series1.dtype
```

```
def central_limit_theorem(data,n_samples = 1000, sample_size = 500,  
min_value = 0, max_value = 1338):
```

“”” Use this function to demonstrate Central Limit Theorem.

```
data = 1D array, or a pd.Series
```

```
n_samples = number of samples to be created
```

```
sample_size = size of the individual sample
```

```
min_value = minimum index of the data
```

```
max_value = maximum index value of the data “””
```

```
%matplotlib inline
```

```
import pandas as pd
```

```
import numpy as np
```

```
import matplotlib.pyplot as plt
```

```
import seaborn as sns
```

```
b = { }
```

```
for i in range(n_samples):
```

```
    x = np.unique(np.random.randint(min_value, max_value, size =  
sample_size)) # set of random numbers with a specific size
```

```
b[i] = data[x].mean() # Mean of each sample
```

```
c = pd.DataFrame()
```

```
c['sample'] = b.keys() # Sample number
```

```
c['Mean'] = b.values() # mean of that particular sample
```

```
plt.figure(figsize=(15,5))
```

```
plt.subplot(1,2,1)
```

```
sns.distplot(c.Mean)
```

```
plt.title(f"Sampling Distribution of bmi.\nbc = {round(c.Mean.mean(), 3)} & SE = {round(c.Mean.std(),3)}")
```

```
plt.xlabel('data')
```

```
plt.ylabel('freq')

plt.subplot(1,2,2)

sns.distplot(data)

plt.title(f'population Distribution of bmi. \n \u03bc = {round(data.mean(), 3)} & \u03c3 = {round(data.std(),3)}')

plt.xlabel('data')

plt.ylabel('freq')

plt.show()

central_limit_theorem(series1,n_samples = 5000, sample_size = 500)
```

Code Explanation:

We start off by importing the insurance.csv file with this command:

```
df = pd.read_csv('insurance.csv')
```

Then we go ahead and define the central limit theorem method:

```
def central_limit_theorem(data,n_samples = 1000, sample_size = 500,  
min_value = 0, max_value = 1338):
```

This method comprises of these parameters:

- Data
- N_samples
- Sample_size
- Min_value
- Max_value

Inside this method, we import all the required libraries:

```
import pandas as pd  
  
import numpy as np  
  
import matplotlib.pyplot as plt  
  
import seaborn as sns
```

Then, we go ahead and create the first sub-plot for “Sampling distribution of bmi”:

```
plt.subplot(1,2,1)  
  
sns.distplot(c.Mean)  
  
plt.title(f"Sampling Distribution of bmi. \n \u03bc = {round(c.Mean.mean(),  
3)} & SE = {round(c.Mean.std(),3)}")  
  
plt.xlabel('data')  
  
plt.ylabel('freq')
```

Finally, we create the sub-plot for “Population distribution of BMI”:

```
plt.subplot(1,2,2)  
  
sns.distplot(data)
```

```
plt.title(f'population Distribution of bmi. \n \u03bc = {round(data.mean(), 3)} & \u03c3 = {round(data.std(),3)}")\n\nplt.xlabel('data')\n\nplt.ylabel('freq')\n\nplt.show()
```

104. Write code to perform sentiment analysis on amazon reviews:

This is one of the most commonly asked python interview questions.

```
import pandas as pd\n\n\n\n\nimport numpy as np\n\n\n\n\nimport matplotlib.pyplot as plt\n\n\n\n\nfrom tensorflow.python.keras import models, layers, optimizers\n\n\n\n\nimport tensorflow\n\n\n\n\nfrom tensorflow.keras.preprocessing.text import Tokenizer,\n    text_to_word_sequence\n\n\n\n\nfrom tensorflow.keras.preprocessing.sequence import pad_sequences
```

```
import bz2

from sklearn.metrics import f1_score, roc_auc_score, accuracy_score

import re

%matplotlib inline

def get_labels_and_texts(file):

    labels = []

    texts = []

    for line in bz2.BZ2File(file):

        x = line.decode("utf-8")

        labels.append(int(x[9]) - 1)
```

```
texts.append(x[10:].strip())
```

```
return np.array(labels), texts
```

```
train_labels, train_texts = get_labels_and_texts('train.ft.txt.bz2')
```

```
test_labels, test_texts = get_labels_and_texts('test.ft.txt.bz2')
```

```
Train_labels[0]
```

```
Train_texts[0]
```

```
train_labels=train_labels[0:500]
```

```
train_texts=train_texts[0:500]
```

```
import re
```

```
NON_ALPHANUM = re.compile(r'[\W]')
```

```
NON_ASCII = re.compile(r'[^a-zA-Z\s]')

def normalize_texts(texts):

    normalized_texts = []

    for text in texts:

        lower = text.lower()

        no_punctuation = NON_ALPHANUM.sub(r' ', lower)

        no_non_ascii = NON_ASCII.sub(r'', no_punctuation)

        normalized_texts.append(no_non_ascii)

    return normalized_texts

train_texts = normalize_texts(train_texts)
```

```
test_texts = normalize_texts(test_texts)

from sklearn.feature_extraction.text import CountVectorizer

cv = CountVectorizer(binary=True)

cv.fit(train_texts)

X = cv.transform(train_texts)

X_test = cv.transform(test_texts)

from sklearn.linear_model import LogisticRegression

from sklearn.metrics import accuracy_score

from sklearn.model_selection import train_test_split

X_train, X_val, y_train, y_val = train_test_split(
```

```
X, train_labels, train_size = 0.75)

for c in [0.01, 0.05, 0.25, 0.5, 1]:

    lr = LogisticRegression(C=c)

    lr.fit(X_train, y_train)

    print ("Accuracy for C=%s: %s"

           % (c, accuracy_score(y_val, lr.predict(X_val)))))

lr.predict(X_test[29])
```

105. Implement a probability plot using numpy and matplotlib:

sol:

```
import numpy as np
```

```
import pylab
```

```
import scipy.stats as stats  
  
from matplotlib import pyplot as plt  
  
n1=np.random.normal(loc=0,scale=1,size=1000)  
  
np.percentile(n1,100)  
  
n1=np.random.normal(loc=20,scale=3,size=100)  
  
stats.probplot(n1,dist="norm",plot=pylab)  
  
plt.show()
```

106. Implement multiple linear regression on this iris dataset:

The independent variables should be “Sepal.Width”, “Petal.Length”, “Petal.Width”, while the dependent variable should be “Sepal.Length”.

Sol:

```
import pandas as pd  
  
iris = pd.read_csv("iris.csv")
```

```
iris.head()
```

```
x = iris[['Sepal.Width','Petal.Length','Petal.Width']]
```

```
y = iris[['Sepal.Length']]
```

```
from sklearn.model_selection import train_test_split
```

```
x_train, x_test, y_train, y_test = train_test_split(x, y, test_size = 0.35)
```

```
from sklearn.linear_model import LinearRegression
```

```
lr = LinearRegression()
```

```
lr.fit(x_train, y_train)
```

```
y_pred = lr.predict(x_test)
```

```
from sklearn.metrics import mean_squared_error
```

```
mean_squared_error(y_test, y_pred)
```

Code solution:

We start off by importing the required libraries:

```
import pandas as pd  
  
iris = pd.read_csv("iris.csv")  
  
iris.head()
```

Then, we will go ahead and extract the independent variables and dependent variable:

```
x = iris[['Sepal.Width', 'Petal.Length', 'Petal.Width']]  
  
y = iris[['Sepal.Length']]
```

Following which, we divide the data into train and test sets:

```
from sklearn.model_selection import train_test_split  
  
x_train, x_test, y_train, y_test = train_test_split(x, y, test_size = 0.35)
```

Then, we go ahead and build the model:

```
from sklearn.linear_model import LinearRegression  
  
lr = LinearRegression()  
  
lr.fit(x_train, y_train)  
  
y_pred = lr.predict(x_test)
```

Finally, we will find out the mean squared error:

```
from sklearn.metrics import mean_squared_error  
  
mean_squared_error(y_test, y_pred)
```

107. From this credit fraud dataset:

Find the percentage of transactions that are fraudulent and not fraudulent. Also build a logistic regression model, to find out if the transaction is fraudulent or not.

Sol:

nfcnt=0

```
notFraud=data_df['Class']
```

```
for i in range(len(notFraud)):
```

if notFraud[i]==0:

`nfcnt=nfcnt+1`

nfcount

```
per_nf=(nfcnt/lens(notFraud))*100
```

```
print('percentage of total not fraud transaction in the dataset: ',per_nf)
```

```
fcount=0
```

```
Fraud=data_df['Class']
```

```
for i in range(len(Fraud)):
```

```
    if Fraud[i]==1:
```

```
        fcount=fcount+1
```

```
fcount
```

```
per_f=(fcount/len(Fraud))*100
```

```
print('percentage of total fraud transaction in the dataset: ',per_f)
```

```
x=data_df.drop(['Class'], axis = 1)#drop the target variable
```

```
y=data_df['Class']

xtrain, xtest, ytrain, ytest = train_test_split(x, y, test_size = 0.2, random_state = 42)

logisticreg = LogisticRegression()

logisticreg.fit(xtrain, ytrain)

y_pred = logisticreg.predict(xtest)

accuracy= logisticreg.score(xtest,ytest)

cm = metrics.confusion_matrix(ytest, y_pred)

print(cm)
```

**108. Implement a simple CNN on the MNIST dataset using Keras.
Following this, also add in drop-out layers.**

Sol:

```
from __future__ import absolute_import, division, print_function
```

```
import numpy as np

# import keras

from tensorflow.keras.datasets import cifar10, mnist

from tensorflow.keras.models import Sequential

from tensorflow.keras.layers import Dense, Activation, Dropout, Flatten,
Reshape

from tensorflow.keras.layers import Convolution2D, MaxPooling2D

from tensorflow.keras import utils

import pickle

from matplotlib import pyplot as plt

import seaborn as sns
```

```
plt.rcParams[‘figure.figsize’] = (15, 8)

%matplotlib inline

# Load/Prep the Data

(x_train, y_train_num), (x_test, y_test_num) = mnist.load_data()

x_train = x_train.reshape(x_train.shape[0], 28, 28, 1).astype(‘float32’)

x_test = x_test.reshape(x_test.shape[0], 28, 28, 1).astype(‘float32’)

x_train /= 255

x_test /= 255

y_train = utils.to_categorical(y_train_num, 10)

y_test = utils.to_categorical(y_test_num, 10)
```

```
print('— THE DATA —')

print('x_train shape:', x_train.shape)

print(x_train.shape[0], 'train samples')

print(x_test.shape[0], 'test samples')

TRAIN = False

BATCH_SIZE = 32

EPOCHS = 1

# Define the Type of Model

model1 = tf.keras.Sequential()

# Flatten Images to Vector
```

```
model1.add(Reshape((784,), input_shape=(28, 28, 1)))  
  
# Layer 1  
  
model1.add(Dense(128, kernel_initializer='he_normal', use_bias=True))  
  
model1.add(Activation("relu"))  
  
# Layer 2  
  
model1.add(Dense(10, kernel_initializer='he_normal', use_bias=True))  
  
model1.add(Activation("softmax"))  
  
# Loss and Optimizer  
  
model1.compile(loss='categorical_crossentropy', optimizer='adam',  
metrics=['accuracy'])
```

```
# Store Training Results
```

```
early_stopping = keras.callbacks.EarlyStopping(monitor='val_acc',  
patience=10, verbose=1, mode='auto')
```

```
callback_list = [early_stopping]# [stats, early_stopping]
```

```
# Train the model
```

```
model1.fit(x_train, y_train, nb_epoch=EPOCHS, batch_size=BATCH_SIZE,  
validation_data=(x_test, y_test), callbacks=callback_list, verbose=True)
```

```
#drop-out layers:
```

```
# Define Model
```

```
model3 = tf.keras.Sequential()
```

```
# 1st Conv Layer
```

```
model3.add(Convolution2D(32, (3, 3), input_shape=(28, 28, 1)))
```

```
model3.add(Activation('relu'))
```

```
# 2nd Conv Layer
```

```
model3.add(Convolution2D(32, (3, 3)))
```

```
model3.add(Activation('relu'))
```

```
# Max Pooling
```

```
model3.add(MaxPooling2D(pool_size=(2,2)))
```

```
# Dropout
```

```
model3.add(Dropout(0.25))
```

```
# Fully Connected Layer
```

```
model3.add(Flatten())
```

```
model3.add(Dense(128))
```

```
model3.add(Activation('relu'))
```

```
# More Dropout
```

```
model3.add(Dropout(0.5))
```

```
# Prediction Layer
```

```
model3.add(Dense(10))
```

```
model3.add(Activation('softmax'))
```

```
# Loss and Optimizer
```

```
model3.compile(loss='categorical_crossentropy', optimizer='adam',  
metrics=['accuracy'])
```

```
# Store Training Results

early_stopping = tf.keras.callbacks.EarlyStopping(monitor='val_acc',
patience=7, verbose=1, mode='auto')

callback_list = [early_stopping]

# Train the model

model3.fit(x_train, y_train, batch_size=BATCH_SIZE, nb_epoch=EPOCHS,
            validation_data=(x_test, y_test), callbacks=callback_list)
```

109. Implement a popularity-based recommendation system on this movie lens dataset:

```
import os

import numpy as np

import pandas as pd

ratings_data = pd.read_csv("ratings.csv")
```

```
ratings_data.head()
```

```
movie_names = pd.read_csv("movies.csv")
```

```
movie_names.head()
```

```
movie_data = pd.merge(ratings_data, movie_names, on='movieId')
```

```
movie_data.groupby('title')['rating'].mean().head()
```

```
movie_data.groupby('title')['rating'].mean().sort_values(ascending=False).head()
```

```
movie_data.groupby('title')['rating'].count().sort_values(ascending=False).head()
```

```
ratings_mean_count =  
pd.DataFrame(movie_data.groupby('title')['rating'].mean())
```

```
ratings_mean_count.head()
```

```
ratings_mean_count['rating_counts'] =  
pd.DataFrame(movie_data.groupby('title')['rating'].count())
```

```
ratings_mean_count.head()
```

110. Implement the naive Bayes algorithm on top of the diabetes dataset:

```
import numpy as np # linear algebra
```

```
import pandas as pd # data processing, CSV file I/O (e.g. pd.read_csv)
```

```
import matplotlib.pyplot as plt      # matplotlib.pyplot plots data
```

```
%matplotlib inline
```

```
import seaborn as sns
```

```
pdata = pd.read_csv("pima-indians-diabetes.csv")
```

```
columns = list(pdata)[0:-1] # Excluding Outcome column which has only
```

```
pdata[columns].hist(stacked=False, bins=100, figsize=(12,30), layout=(14,2));
```

```
# Histogram of first 8 columns
```

However, we want to see a correlation in graphical representation so below is the function for that:

```
def plot_corr(df, size=11):
```

```
    corr = df.corr()
```

```
    fig, ax = plt.subplots(figsize=(size, size))
```

```
    ax.matshow(corr)
```

```
    plt.xticks(range(len(corr.columns)), corr.columns)
```

```
    plt.yticks(range(len(corr.columns)), corr.columns)
```

```
plot_corr(pdata)
```

```
from sklearn.model_selection import train_test_split
```

```
X = pdata.drop('class',axis=1) # Predictor feature columns (8 X m)
```

```
Y = pdata['class'] # Predicted class (1=True, 0=False) (1 X m)
```

```
x_train, x_test, y_train, y_test = train_test_split(X, Y, test_size=0.3,  
random_state=1)
```

```
# 1 is just any random seed number
```

```
x_train.head()
```

```
from sklearn.naive_bayes import GaussianNB # using Gaussian algorithm from  
Naive Bayes
```

```
# creatw the model
```

```
diab_model = GaussianNB()
```

```
diab_model.fit(x_train, y_train.ravel())
```

```
diab_train_predict = diab_model.predict(x_train)
```

```
from sklearn import metrics  
  
print("Model Accuracy: {:.4f}".format(metrics.accuracy_score(y_train,  
diab_train_predict)))
```

```
print()  
  
diab_test_predict = diab_model.predict(x_test)
```

```
from sklearn import metrics  
  
print("Model Accuracy: {:.4f}".format(metrics.accuracy_score(y_test,  
diab_test_predict)))  
  
print()
```

```
print("Confusion Matrix")  
  
cm=metrics.confusion_matrix(y_test, diab_test_predict, labels=[1, 0])
```

```
df_cm = pd.DataFrame(cm, index = [i for i in ["1","0"]],  
                      columns = [i for i in ["Predict 1","Predict 0"]])  
  
plt.figure(figsize = (7,5))  
  
sns.heatmap(df_cm, annot=True)
```

111. How can you find the minimum and maximum values present in a tuple?

Solution ->

We can use the min() function on top of the tuple to find out the minimum value present in the tuple:

```
tup1=(1,2,3,4,5)
```

```
min(tup1)
```

Output

1

We see that the minimum value present in the tuple is 1.

Analogous to the min() function is the max() function, which will help us to find out the maximum value present in the tuple:

```
tup1=(1,2,3,4,5)
```

```
max(tup1)
```

Output

We see that the maximum value present in the tuple is 5.

112. If you have a list like this -> [1,”a”,2,”b”,3,”c”]. How can you access the 2nd, 4th and 5th elements from this list?

Solution ->

We will start off by creating a tuple that will comprise the indices of elements that we want to access.

Then, we will use a for loop to go through the index values and print them out.

Below is the entire code for the process:

```
indices = (1,3,4)
```

```
for i in indices:
```

```
    print(a[i])
```

113. If you have a list like this -> [“sparta”,True,3+4j,False]. How would you reverse the elements of this list?

Solution ->

We can use the reverse() function on the list:

```
a.reverse()
```

```
a
```

**114. If you have dictionary like this – >
fruit={“Apple”:10,”Orange”:20,”Banana”:30,”Guava”:40}. How would you update the value of ‘Apple’ from 10 to 100?**

Solution ->

This is how you can do it:

```
fruit["Apple"]=100
```

```
fruit
```

Give in the name of the key inside the parenthesis and assign it a new value.

115. If you have two sets like this -> s1 = {1,2,3,4,5,6}, s2 = {5,6,7,8,9}. How would you find the common elements in these sets.

Solution ->

You can use the intersection() function to find the common elements between the two sets:

```
s1 = {1,2,3,4,5,6}
```

```
s2 = {5,6,7,8,9}
```

```
s1.intersection(s2)
```

We see that the common elements between the two sets are 5 & 6.

116. Write a program to print out the 2-table using while loop.

Solution ->

Below is the code to print out the 2-table:

Code

```
i=1
```

```
n=2
```

```
while i<=10:
```

```
    print(n,"*", i, "=", n*i)
```

```
    i=i+1
```

Output

We start off by initializing two variables ‘i’ and ‘n’. ‘i’ is initialized to 1 and ‘n’ is initialized to ‘2’.

Inside the while loop, since the ‘i’ value goes from 1 to 10, the loop iterates 10 times.

Initially $n \cdot i$ is equal to $2 \cdot 1$, and we print out the value.

Then, ‘i’ value is incremented and $n \cdot i$ becomes $2 \cdot 2$. We go ahead and print it out.

This process goes on until i value becomes 10.

117. Write a function, which will take in a value and print out if it is even or odd.

Solution ->

The below code will do the job:

```
def even_odd(x):
    if x%2==0:
        print(x," is even")
    else:
        print(x, " is odd")
```

Here, we start off by creating a method, with the name ‘even_odd()’. This function takes a single parameter and prints out if the number taken is even or odd.

Now, let’s invoke the function:

```
even_odd(5)
```

We see that, when 5 is passed as a parameter into the function, we get the output -> ‘5 is odd’.

118. Write a python program to print the factorial of a number.

This is one of the most commonly asked python interview questions

Solution ->

Below is the code to print the factorial of a number:

```
factorial = 1

#check if the number is negative, positive or zero

if num<0:

    print("Sorry, factorial does not exist for negative numbers")

elif num==0:

    print("The factorial of 0 is 1")

else

    for i in range(1,num+1):

        factorial = factorial*i

    print("The factorial of",num,"is",factorial)
```

We start off by taking an input which is stored in ‘num’. Then, we check if ‘num’ is less than zero and if it is actually less than 0, we print out ‘Sorry, factorial does not exist for negative numbers’.

After that, we check,if ‘num’ is equal to zero, and if that’s the case, we print out ‘The factorial of 0 is 1’.

On the other hand, if ‘num’ is greater than 1, we enter the for loop and calculate the factorial of the number.

119. Write a python program to check if the number given is a palindrome or not

Solution ->

Below is the code to Check whether the given number is palindrome or not:

```
n=int(input("Enter number:"))

temp=n

rev=0

while(n>0)

    dig=n%10

    rev=rev*10+dig

    n=n//10

if(temp==rev):

    print("The number is a palindrome!")

else:

    print("The number isn't a palindrome!")
```

We will start off by taking an input and store it in ‘n’ and make a duplicate of it in ‘temp’. We will also initialize another variable ‘rev’ to 0.

Then, we will enter a while loop which will go on until ‘n’ becomes 0.

Inside the loop, we will start off by dividing ‘n’ with 10 and then store the remainder in ‘dig’.

Then, we will multiply ‘rev’ with 10 and then add ‘dig’ to it. This result will be stored back in ‘rev’.

Going ahead, we will divide ‘n’ by 10 and store the result back in ‘n’

Once the for loop ends, we will compare the values of ‘rev’ and ‘temp’. If they are equal, we will print ‘The number is a palindrome’, else we will print ‘The number isn’t a palindrome’.

120. Write a python program to print the following pattern ->

This is one of the most commonly asked python interview questions:

1

2 2

3 3 3

4 4 4 4

5 5 5 5 5

Solution ->

Below is the code to print this pattern:

```
#10 is the total number to print  
  
for num in range(6):  
  
    for i in range(num):  
  
        print(num,end=" ")#print number  
  
    #new line after each row to display pattern correctly  
  
    print("\n")
```

We are solving the problem with the help of nested for loop. We will have an outer for loop, which goes from 1 to 5. Then, we have an inner for loop, which would print the respective numbers.

121. Pattern questions. Print the following pattern

```
#
```

```
# #
```

```
# # #
```

```
# # # #
```

```
# # # # #
```

Solution ->

```
def pattern_1(num):  
  
    # outer loop handles the number of rows  
  
    # inner loop handles the number of columns  
  
    # n is the number of rows.  
  
    for i in range(0, n):  
  
        # value of j depends on i  
  
        for j in range(0, i+1):  
  
            # printing hashes  
  
            print("#",end="")  
  
    # ending line after each row
```

```
print("\r")

num = int(input("Enter the number of rows in pattern: "))

pattern_1(num)
```

122. Print the following pattern.

```
#  
  
# #  
  
# # #  
  
# # # #  
  
# # # # #
```

Solution ->

Code:

```
def pattern_2(num):  
  
    # define the number of spaces  
    k = 2*num - 2  
  
    # outer loop always handles the number of rows  
    # let us use the inner loop to control the number of spaces  
    # we need the number of spaces as maximum initially and then decrement it  
    # after every iteration
```

```
for i in range(0, num):

    for j in range(0, k):

        print(end=" ")

    # decrementing k after each loop

    k = k - 2

    # reinitializing the inner loop to keep a track of the number of columns

    # similar to pattern_1 function

    for j in range(0, i+1):

        print("# ", end="")

    # ending line after each row

    print("\r")

num = int(input("Enter the number of rows in pattern: "))

pattern_2(num)
```

123. Print the following pattern:

0

0 1

0 1 2

0 1 2 3

0 1 2 3 4

Solution ->

Code:

```
def pattern_3(num):

    # initialising starting number
    number = 1

    # outer loop always handles the number of rows
    # let us use the inner loop to control the number

    for i in range(0, num):

        # re assigning number after every iteration
        # ensure the column starts from 0
        number = 0

        # inner loop to handle number of columns
```

```
for j in range(0, i+1):

    # printing number

    print(number, end=" ")

    # increment number column wise

    number = number + 1

    # ending line after each row

    print("\r")

num = int(input("Enter the number of rows in pattern: "))

pattern_3(num)
```

124. Print the following pattern:

1

2 3

4 5 6

7 8 9 10

11 12 13 14 15

Solution ->

Code:

```
def pattern_4(num):

    # initialising starting number
    number = 1

    # outer loop always handles the number of rows
    # let us use the inner loop to control the number

    for i in range(0, num):

        # commenting the reinitialization part ensure that numbers are printed
        # continuously

        # ensure the column starts from 0
        number = 0

        # inner loop to handle number of columns
        for j in range(0, i+1):

            # printing number
            print(number, end=" ")


```

```
# increment number column wise  
  
number = number + 1  
  
# ending line after each row  
  
print("\r")  
  
num = int(input("Enter the number of rows in pattern: "))  
  
pattern_4(num)
```

125. Print the following pattern:

A

B B

C C C

D D D D

Solution ->

```
def pattern_5(num):  
  
    # initializing value of A as 65  
  
    # ASCII value equivalent  
  
    number = 65  
  
    # outer loop always handles the number of rows
```

```
for i in range(0, num):  
  
    # inner loop handles the number of columns  
  
    for j in range(0, i+1):  
  
        # finding the ascii equivalent of the number  
  
        char = chr(number)  
  
        # printing char value  
  
        print(char, end=" ")  
  
    # incrementing number  
  
    number = number + 1  
  
    # ending line after each row  
  
    print("\r")  
  
num = int(input("Enter the number of rows in pattern: "))  
  
pattern_5(num)
```

126. Print the following pattern:

A

B C

D E F

G H I J

K L M N O

P Q R S T U

Solution ->

```
def pattern_6(num):

    # initializing value equivalent to 'A' in ASCII

    # ASCII value

    number = 65

    # outer loop always handles the number of rows

    for i in range(0, num):

        # inner loop to handle number of columns

        # values changing acc. to outer loop

        for j in range(0, i+1):

            # explicit conversion of int to char

            # returns character equivalent to ASCII.

            char = chr(number)
```

```
# printing char value  
  
print(char, end=" ")  
  
# printing the next character by incrementing  
number = number +1  
  
# ending line after each row  
  
print("\r")  
  
num = int(input("enter the number of rows in the pattern: "))  
  
pattern_6(num)
```

127. Print the following pattern

```
#  
  
# #  
  
# # #  
  
# # # #  
  
# # # # #
```

Solution ->

Code:

```
def pattern_7(num):  
  
    # number of spaces is a function of the input num
```

```
k = 2*num - 2

# outer loop always handle the number of rows

for i in range(0, num):

    # inner loop used to handle the number of spaces

    for j in range(0, k):

        print(end=" ")

    # the variable holding information about number of spaces

    # is decremented after every iteration

    k = k - 1

    # inner loop reinitialized to handle the number of columns

    for j in range(0, i+1):

        # printing hash

        print("# ", end="")

    # ending line after each row
```

```
print("\r")  
  
num = int(input("Enter the number of rows: "))  
  
pattern_7(n)
```

128. If you have a dictionary like this -> d1={"k1":10,"k2":20,"k3":30}. How would you increment values of all the keys ?

```
d1={"k1":10,"k2":20,"k3":30}
```

```
for i in d1.keys():
```

```
    d1[i]=d1[i]+1
```

129. How can you get a random number in python?

Ans. To generate a random, we use a random module of python. Here are some examples To generate a floating-point number from 0-1

```
import random  
  
n = random.random()  
  
print(n)
```

To generate a integer between a certain range (say from a to b):

```
import random  
  
n = random.randint(a,b)  
  
print(n)
```

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

All of the project's settings, as well as database connection information, are contained in the `settings.py` file. Django works with the SQLite database by default, but it may be configured to operate with other databases as well.

Database connectivity necessitates full connection information, including the database name, user credentials, hostname, and drive name, among other things.

To connect to MySQL and establish a connection between the application and the database, use the `django.db.backends.mysql` driver.

All connection information must be included in the `settings` file. Our project's `settings.py` file has the following code for the database.

```
DATABASES = {  
    'default': {  
        'ENGINE': 'django.db.backends.mysql',  
        'NAME': 'djangoApp',  
        'USER': 'root',  
        'PASSWORD': 'mysql',  
        'HOST': 'localhost',  
        'PORT': '3306'  
    }  
}
```

This command will build tables for `admin`, `auth`, `contenttypes`, and `sessions`. You may now connect to the MySQL database by selecting it from the database drop-down menu.

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

The Django MVT Structure is incomplete without Django Views. A view function is a Python function that receives a Web request and delivers a Web response, according to the Django manual. This response might be a web page's HTML content, a redirect, a 404 error, an XML document, an image, or anything else that a web browser can display.

The HTML/CSS/JavaScript in your Template files is converted into what you see in your browser when you show a web page using Django views, which are part of the user interface. (Do not combine Django views with MVC views if you've used other MVC (Model-View-Controller) frameworks.) In Django, the views are similar.

```
# import Http Response from django
from django.http import HttpResponse
# get datetime
import datetime
# create a function
def geeks_view(request):
    # fetch date and time
    now = datetime.datetime.now()
    # convert to string
    html = "Time is {}".format(now)
    # return response
    return HttpResponse(html)
```

132. Explain the use of sessions in the Django framework?

Django (and much of the Internet) uses sessions to track the “status” of a particular site and browser. Sessions allow you to save any amount of data per

browser and make it available on the site each time the browser connects. The data elements of the session are then indicated by a “key”, which can be used to save and recover the data.

Django uses a cookie with a single character ID to identify any browser and its website associated with the website. Session data is stored in the site’s database by default (this is safer than storing the data in a cookie, where it is more vulnerable to attackers).

Django allows you to store session data in a variety of locations (cache, files, “safe” cookies), but the default location is a solid and secure choice.

Enabling sessions

When we built the skeleton website, sessions were enabled by default.

The config is set up in the project file (`locallibrary/locallibrary/settings.py`) under the `INSTALLED_APPS` and `MIDDLEWARE` sections, as shown below:

```
INSTALLED_APPS = [  
    ...  
    'django.contrib.sessions',  
    ....  
  
MIDDLEWARE = [  
    ...  
    'django.contrib.sessions.middleware.SessionMiddleware',  
    ...
```

Using sessions

The `request` parameter gives you access to the view’s session property (an `HttpRequest` passed in as the first argument to the view). The session id in the

browser's cookie for this site identifies the particular connection to the current user (or, to be more accurate, the connection to the current browser).

The session assets is a dictionary-like item that you can examine and write to as frequently as you need on your view, updating it as you go. You may do all of the standard dictionary actions, such as clearing all data, testing for the presence of a key, looping over data, and so on. Most of the time, though, you'll merely obtain and set values using the usual "dictionary" API.

The code segments below demonstrate how to obtain, change, and remove data linked with the current session using the key "my bike" (browser).

Note: One of the best things about Django is that you don't have to worry about the mechanisms that you think are connecting the session to the current request. If we were to use the fragments below in our view, we'd know that the information about my_bike is associated only with the browser that sent the current request.

```
# Get a session value via its key (for example 'my_bike'), raising a KeyError if
# the key is not present

my_bike= request.session['my_bike']

# Get a session value, setting a default value if it is not present ( 'mini')

my_bike= request.session.get('my_bike', 'mini')

# Set a session value

request.session['my_bike'] = 'mini'

# Delete a session value

del request.session['my_bike']
```

A variety of different methods are available in the API, most of which are used to control the linked session cookie. There are ways to verify whether the client browser supports cookies, to set and check cookie expiration dates, and to delete expired sessions from the data store, for example. How to utilise sessions has further information on the whole API (Django docs).

133. List out the inheritance styles in Django.

Abstract base classes: This inheritance pattern is used by developers when they want the parent class to keep data that they don't want to type out for each child model.

```
models.py
```

```
from django.db import models
```

```
# Create your models here.
```

```
class ContactInfo(models.Model):
```

```
    name=models.CharField(max_length=20)
```

```
    email=models.EmailField(max_length=20)
```

```
    address=models.TextField(max_length=20)
```

```
class Meta:
```

```
    abstract=True
```

```
class Customer(ContactInfo):
```

```
    phone=models.IntegerField(max_length=15)
```

```
class Staff(ContactInfo):
```

```
position=models.CharField(max_length=10)
```

```
admin.py
```

```
admin.site.register(Customer)
```

```
admin.site.register(Staff)
```

Two tables are formed in the database when we transfer these modifications. We have fields for name, email, address, and phone in the Customer Table. We have fields for name, email, address, and position in Staff Table. Table is not a base class that is built in This inheritance.

Multi-table inheritance: It is utilised when you wish to subclass an existing model and have each of the subclasses have its own database table.

```
model.py
```

```
from django.db import models
```

```
# Create your models here.
```

```
class Place(models.Model):
```

```
    name=models.CharField(max_length=20)
```

```
    address=models.TextField(max_length=20)
```

```
    def __str__(self):
```

```
        return self.name
```

```
class Restaurants(Place):
    serves_pizza=models.BooleanField(default=False)
    serves_pasta=models.BooleanField(default=False)

    def __str__(self):
        return self.serves_pasta
```

admin.py

```
from django.contrib import admin
from .models import Place,Restaurants
# Register your models here.
```

```
admin.site.register(Place)
admin.site.register(Restaurants)
```

Proxy models: This inheritance approach allows the user to change the behaviour at the basic level without changing the model's field.

This technique is used if you just want to change the model's Python level behaviour and not the model's fields. With the exception of fields, you inherit from the base class and can add your own properties.

- Abstract classes should not be used as base classes.
- Multiple inheritance is not possible in proxy models.

The main purpose of this is to replace the previous model's key functions. It always uses overridden methods to query the original model.

134. How can you get the Google cache age of any URL or web page?

Use the URL

<https://webcache.googleusercontent.com/search?q=cache:<your url without "http://>>

Example:

It contains a header like this:

This is Google's cache of <https://stackoverflow.com/>. It's a screenshot of the page as it looked at 11:33:38 GMT on August 21, 2012. In the meanwhile, the current page may have changed.

Tip: Use the find bar and press Ctrl+F or ⌘+F (Mac) to quickly find your search word on this page.

You'll have to scrape the resultant page, however the most current cache page may be found at this URL:

<http://webcache.googleusercontent.com/search?q=cache:www.something.com/path>

The first div in the body tag contains Google information.

you can Use CachedPages website

Large enterprises with sophisticated web servers typically preserve and keep cached pages. Because such servers are often quite fast, a cached page can frequently be retrieved faster than the live website:

- A current copy of the page is generally kept by Google (1 to 15 days old).
- Coral also retains a current copy, although it isn't as up to date as Google's.
- You may access several versions of a web page preserved over time using Archive.org.

So, the next time you can't access a website but still want to look at it, Google's cache version could be a good option. First, determine whether or not age is important.

135. Briefly explain about Python namespaces?

A namespace in python talks about the name that is assigned to each object in Python. Namespaces are preserved in python like a dictionary where the key of the dictionary is the namespace and value is the address of that object.

Different types are as follows:

- Built-in-namespace – Namespaces containing all the built-in objects in python.
- Global namespace – Namespaces consisting of all the objects created when you call your main program.
- Enclosing namespace – Namespaces at the higher lever.
- Local namespace – Namespaces within local functions.

136. Briefly explain about Break, Pass and Continue statements in Python ?

Break: When we use a break statement in a python code/program it immediately breaks/terminates the loop and the control flow is given back to the statement after the body of the loop.

Continue: When we use a continue statement in a python code/program it immediately breaks/terminates the current iteration of the statement and also skips the rest of the program in the current iteration and controls flows to the next iteration of the loop.

Pass: When we use a pass statement in a python code/program it fills up the empty spots in the program.

Example:

```
GL = [10, 30, 20, 100, 212, 33, 13, 50, 60, 70]
```

```
for g in GL:  
    pass  
  
    if (g == 0):  
  
        current = g  
  
        break  
  
    elif(g%2==0):  
  
        continue  
  
    print(g) # output => 1 3 1 3 1  
  
    print(current)
```

137. Give me an example on how you can convert a list to a string?

Below given example will show how to convert a list to a string. When we convert a list to a string we can make use of the “join” function to do the same.

```
fruits = [ ‘apple’, ‘orange’, ‘mango’, ‘papaya’, ‘guava’]  
  
listAsString = ‘ ‘.join(fruits)  
  
print(listAsString)
```

apple orange mango papaya guava

138. Give me an example where you can convert a list to a tuple?

The below given example will show how to convert a list to a tuple. When we convert a list to a tuple we can make use of the `<tuple()>` function but do remember since tuples are immutable we cannot convert it back to a list.

```
fruits = [‘apple’, ‘orange’, ‘mango’, ‘papaya’, ‘guava’]  
  
listAsTuple = tuple(fruits)
```

```
print(listAsTuple)  
  
('apple', 'orange', 'mango', 'papaya', 'guava')
```

139. How do you count the occurrences of a particular element in the list ?

In the list data structure of python we count the number of occurrences of an element by using count() function.

```
fruits = ['apple', 'orange', 'mango', 'papaya', 'guava']  
  
print(fruits.count('apple'))
```

Output: 1

140. How do you debug a python program?

There are several ways to debug a Python program:

- Using the print statement to print out variables and intermediate results to the console
- Using a debugger like pdb or ipdb
- Adding assert statements to the code to check for certain conditions

141. What is the difference between a list and a tuple in Python?

A list is a mutable data type, meaning it can be modified after it is created. A tuple is immutable, meaning it cannot be modified after it is created. This makes tuples faster and safer than lists, as they cannot be modified by other parts of the code accidentally.

142. How do you handle exceptions in Python?

Exceptions in Python can be handled using a try–except block. For example:

```
try:  
  
    # code that may raise an exception  
  
except SomeExceptionType:  
  
    # code to handle the exception
```

143. How do you reverse a string in Python?

There are several ways to reverse a string in Python:

- Using a slice with a step of -1:

```
string = "abcdefg"
```

```
reversed_string = string[::-1]
```

- Using the reversed function:

```
string = "abcdefg"
```

```
reversed_string = "".join(reversed(string))
```

- Using a for loop:

```
string = "abcdefg"
```

```
reversed_string = ""
```

```
for char in string:
```

```
    reversed_string = char + reversed_string
```

144. How do you sort a list in Python?

There are several ways to sort a list in Python:

- Using the sort method:

```
my_list = [3, 4, 1, 2]
```

```
my_list.sort()
```

- Using the sorted function:

```
my_list = [3, 4, 1, 2]
```

```
sorted_list = sorted(my_list)
```

- Using the sort function from the operator module:

```
from operator import itemgetter
```

```
my_list = [{"a": 3}, {"a": 1}, {"a": 2}]  
sorted_list = sorted(my_list, key=itemgetter("a"))
```

145. How do you create a dictionary in Python?

There are several ways to create a dictionary in Python:

- Using curly braces and colons to separate keys and values
- my_dict = {"key1": "value1", "key2": "value2"}
- Using the dict function:

```
my_dict = dict(key1="value1", key2="value2")
```

- Using the dict constructor:

```
my_dict = dict({"key1": "value1", "key2": "value2"})
```

1. Briefly explain some characteristics of Python.

Python is a general-purpose, high-level, interpreted language. It was specifically developed with the purpose of making the content readable. Python has often been compared to the English language, and it also has fewer syntactic constructions compared to other languages.

2. What are some distinct features of Python?

Some distinct features of Python are:

1. Structured and functional programming is supported.
2. It can be compiled to byte code to create larger applications.
3. Supports high-level dynamic data types.
4. Supports checking of dynamic data types.
5. Applies automated garbage collection.
6. It could be used effectively along with Java, COBRA, C, C++, ActiveX, and COM.

3. What is Pythonpath?

A Pythonpath tells the Python interpreter to locate the module files that can be imported into the program. It includes the Python source library directory and source code directory. You can preset Pythonpath as a Python installer.

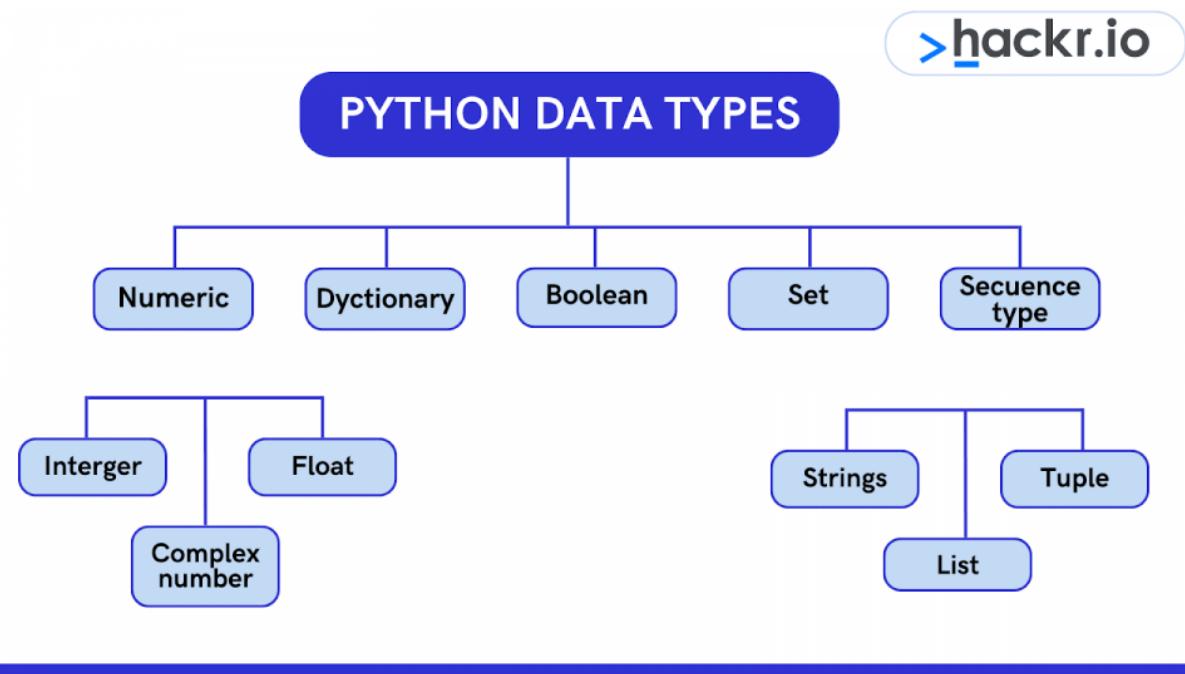
4. Why do we use the Pythonstartup environment variable?

The variable consists of the path in which the initialization file carrying the Python source code can be executed. This is needed to start the interpreter.

5. What is the Pythoncaseok environment variable?

The Pythoncaseok environment variable is applied in Windows with the purpose of directing Python to find the first case insensitive match in an import statement.

6. What are the supported standard data types in Python?



The supported standard data types in Python include:

1. Lists
2. Numbers
3. Strings
4. Dictionaries
5. Tuples

7. What are tuples?

Tuples are a sequence data type with immutable values in Python. The number of values in tuples is separated by commas.

8. What is the major difference between tuples and lists in Python?

There are several major differences between tuples and lists in Python, which include the following:

Tuples	List
Tuples are similar to a list, but they are enclosed within parentheses, unlike the list	The list is used to create a sequence
The element and size can be changed	The element and size cannot be changed
They cannot be updated	They can be updated
They act as read-only lists	They act as a changeable list
Tuples use parentheses	Lists use square brackets
Example: <code>tup = (1, "a", "string", 1+2)</code>	Example: <code>L = [1, "a" , "string" , 1+2]</code>

9. What are positive and negative indices?

Positive indices are applied when the search begins from left to right. In negative indices, the search begins from right to left. For example, in the array list of size n the positive index, the first index is 0, then comes 1, and until the last index is n-1. However, in the negative index, the first index is -n, then -(n-1) until the last index -1.

10. What is the permitted length of the identifier?

The length of the identifier in Python can be of any length. The longest identifier will be from PEP – 8 and PEP – 20.

11. What does the method object() do?

The method returns a featureless object that is base for all classes. This method does not take any parameters.

12. What is pep 8?

Python Enhancement Proposal or pep 8 is a set of rules that specify how to format Python code for maximum readability.

13. What is namespace in Python?

A namespace is a naming system used to make sure names are unique to avoid naming conflicts

14. Is indentation necessary in Python?

Indentation is required in Python if not done properly the code is not executed properly and might throw errors. Indentation is usually done using four space characters.

15. Define a function in Python.

A block of code that is executed when it is called is defined as a function. The keyword def is used to define a Python function.

16. Define self in Python.

Self is an instance of a class or an object in Python. It is included as the first parameter. It helps differentiate between the methods and attributes of a class with local variables.

17. What is the Pass statement?

A Pass statement in Python is used when we cannot decide what to do in our code, but we must type something to make it syntactically correct.

18. What are the limitations of Python?

There are limitations to Python, which include the following:

1. It has design restrictions.
2. It is slower when compared with C and C++ or Java.
3. It is inefficient for mobile computing.
4. It consists of an underdeveloped database access layer.

19. Do runtime errors exist in Python? Give an example.

Yes, runtime errors exist in Python. For example, if you are duck typing and things look like a duck, then it is considered a duck even if that is just a flag or

stamp. The code, in this case, would be a run-time error. For example, Print “Hackr io” would result in the runtime error of the missing parenthesis that is required by print().

20. Can we reverse a list in Python?

Yes, we can reserve a list in Python using the reverse() method. The code is as follows:

```
def reverse(s):  
    str = ""  
  
    for i in s:  
        str = i + str  
  
    return str
```

21. Why do we need a break?

Break helps control the Python loop by breaking the current loop from execution and transferring the control to the next block.

22. Why do we need a continue?

A continue helps control the Python loop by making jumps to the next iteration of the loop without exhausting it.

23. Can we use a break and continue together in Python? How?

Break and continue can be used together in Python. The break will stop the current loop from execution, while the jump will take it to another loop.

24. Does Python support an intrinsic do-while loop?

No, Python does not support an intrinsic do-while loop.

25. In how many ways can we apply reverse strings?

There are five ways in which the reverse string can be applied:

1. Loops
2. Recursions
3. Stacks
4. Extended slice syntax
5. Reversed function

26. Define slicing in Python.

Slicing refers to the mechanism to select the range of items from sequence types like lists, tuples, strings.

27. What is docstring?

Docstring is a Python documentation string, it is a way of documenting Python functions, classes, and modules.

28. How is a file deleted in Python?

The file can be deleted by either of these commands:

```
os.remove(filename)
```

```
os.unlink(filename)
```

29. What are the different stages of the life cycle of a thread?

The different stages of the life cycle of a thread are:

- **Stage 1:** Creating a class where we can override the run method of the Thread class.
- **Stage 2:** We make a call to start() on the new thread. The thread is taken forward for scheduling purposes.
- **Stage 3:** Execution takes place wherein the thread starts execution, and it reaches the running state.
- **Stage 4:** Thread waits until the calls to methods including join() and sleep() take place.
- **Stage 5:** After the waiting or execution of the thread, the waiting thread is sent for scheduling.
- **Stage 6:** Running thread is done by executing the terminates and reaches the dead state.

30. What are relational operators, assignment operators, and membership operators?

- The purpose of relational operators is to compare values.
- The assignment operators in Python can help in combining all the arithmetic operators with the assignment symbol.
- Membership operators in Python with the purpose to validate the membership of a value in a sequence.

31. How are identity operators different from membership operators?

Unlike membership operators, the identity operators compare the values to find out if they have the same value or not.

32. What are Python decorators?

A specific change made in Python syntax to alter the functions easily are termed as Python decorators.

33. Differentiate between list and tuple.

Tuple is not mutable it can be hashed eg. key for dictionaries. On the other hand, lists are mutable.

34. Describe multithreading in Python.

Using Multithreading to speed up the code is not the go-to option, even though Python comes with a multi-threading package.

The package has the GIL or Global Interpreter Lock, which is a construct. It ensures that only one of the threads executes at any given time. A thread acquires the GIL and then performs work before passing it to the next thread.

This happens so fast that to a user it seems that threads are executing in parallel. Obviously, this is not the case as they are just taking turns while using the same CPU core. GIL passing adds to the overall overhead to the execution.

As such, if you intend to use the threading package for speeding up the execution, using the package is not recommended.

35. Draw a comparison between the range and xrange in Python.

In terms of functionality, both range and xrange are identical. Both allow for generating a list of integers. The main difference between the two is that while range returns a Python list object, xrange returns an xrange object.

Xrange is not able to generate a static list at runtime the way range does. On the contrary, it creates values along with the requirements via a special technique called yielding. It is used with a type of object known as generators.

If you have an enormous range for which you need to generate a list, then xrange is the function to opt for. This is especially relevant for scenarios dealing with a memory-sensitive system, such as a smartphone.

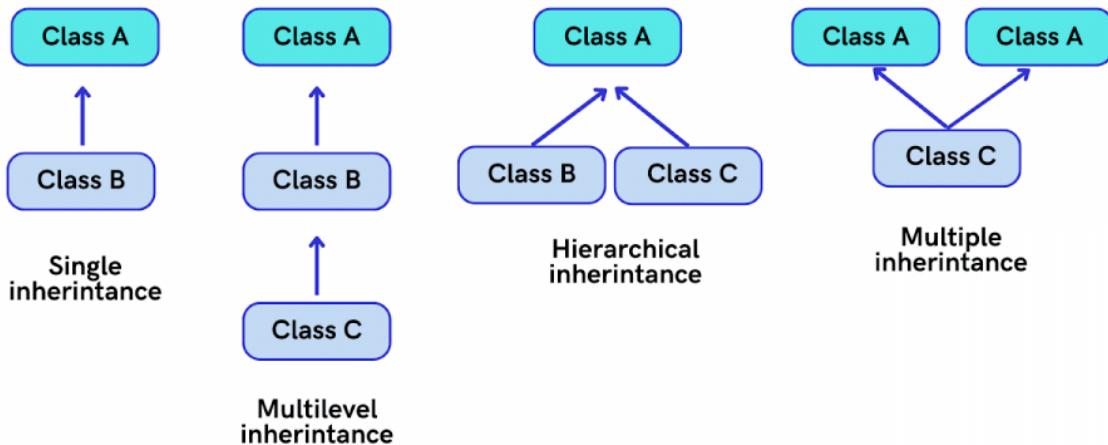
The range is a memory hog. Using it requires much more memory, especially if the requirement is gigantic. Hence, in creating an array of integers to suit the needs, it can result in a memory error and ultimately lead to a crash.

36. Explain Inheritance.

Inheritance enables a class to acquire all members of another class. These members can be attributes, methods, or both. By providing reusability, inheritance makes it easier to create as well as maintain an application.

TYPES OF INHERITANCE

>hackr.io



The class which acquires is known as the child class or the derived class. The one that it acquires from is known as the superclass, base class, or the parent class. There are 4 forms of inheritance supported by Python:

- **Single inheritance:** A single derived class acquires members from one superclass.
- **Multi-Level inheritance:** At least 2 different derived classes acquire members from two distinct base classes.
- **Hierarchical inheritance:** A number of child classes acquire members from one superclass
- **Multiple inheritance:** A derived class acquires members from several superclasses.

37. Explain how to acquire the Google cache age of any URL or webpage using Python.

In order to get the Google cache age of any URL or webpage using Python, the following URL format is used:

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

Simply replace URLGOESHERE with the web address of the website or webpage whose cache you need to retrieve and see in Python.

Python Advanced Interview Questions

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

We use a shallow copy when a new instance type gets created. It keeps the values that are copied in the new instance. Just like it copies the values, the shallow copy also copies the reference pointers.

Reference points copied in the shallow copy reference to the original objects. Any changes made in any member of the class affect the original copy of the same. Shallow copy enables faster execution of the program.

Deep copy is used for storing values that are already copied. Unlike shallow copy, it doesn't copy the reference pointers to the objects. Deep copy makes the reference to an object in addition to storing the new object that is pointed by some other object.

Changes made to the original copy will not affect any other copy that makes use of the referenced or stored object. Contrary to the shallow copy, deep copy makes the execution of a program slower. This is due to the fact that it makes some copies for each object that is called.

39. How do you distinguish between NumPy and SciPy?

Typically, NumPy contains nothing but the array data type and the most basic operations, such as basic element-wise functions, indexing, reshaping, and sorting. All the numerical code resides in SciPy.

As one of NumPy's most important goals is compatibility, the library tries to retain all features supported by either of its predecessors. Hence, NumPy contains a few linear algebra functions despite the fact that these more appropriately belong to the SciPy library.

SciPy contains fully-featured versions of the linear algebra modules available to NumPy in addition to several other numerical algorithms.

40. What is the output of the following code?

```
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 =
```

```
A6 = [[i,i*i] for i in A1]
```

```
print(A0,A1,A2,A3,A4,A5,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 =
```

```
A6 = [[0, 0], [1, 1], [2, 4], [3, 9], [4, 16], [5, 25], [6, 36], [7, 49], [8, 64], [9, 81]]
```

41. Explain dictionaries with an example.

A dictionary in the [Python programming](#) language is an unordered collection of data values such as a map. Dictionary holds the key:value pair. This helps define a one-to-one relationship between keys and values. Indexed by keys, a typical dictionary contains a pair of keys and corresponding values.

Let us take an example with three keys, namely website, language, and offering. Their corresponding values are hackr.io, Python, and Tutorials. The code for would be:

```
dict={'Website':'hackr.io','Language':'Python','Offering':'Tutorials'}
```

```
print dict[Website] #Prints hackr.io
```

```
print dict[Language] #Prints Python
```

```
print dict[Offering] #Prints Tutorials
```

42. Python supports negative indexes. What are they and why are they used?

The sequences in Python are indexed. It consists of positive and negative numbers. Positive numbers use 0 as the first index, 1 as the second index, and so on. Hence, any index for a positive number n is n-1.

Unlike positive numbers, index numbering for the negative numbers starts from -1 and it represents the last index in the sequence. Likewise, -2 represents the penultimate index. These are known as negative indexes. Negative indexes are used for:

- Removing any new-line spaces from the string, thus allowing the string to except the last character, represented as S[:-1]
- Showing the index to representing the string in the correct order

43. Suppose you need to collect and print data from IMDb top 250 Movies page. Write a program in Python for doing so. (NOTE: You can limit the displayed information for 3 fields; namely movie name, release year, and rating.)

```
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)
```

44. What is the output of the following code?

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

The output of the program will be “invalid code.” This is because a new exception class must inherit from a BaseException.

45. What is monkey patching in Python?

The dynamic modifications made to a class or module at runtime are termed as monkey patching in Python. Consider the following code snippet:

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

We can monkey-patch the program something like this:

```
import m  
  
def monkey_f(self):  
  
print "monkey_f()  
  
m.MyClass.f = monkey_f  
  
obj = m.MyClass()
```

```
obj.f()
```

The output for the program will be monkey_f().

The examples demonstrate changes made in the behavior of f() in MyClass using the function we defined i.e. monkey_f() outside of the module m.

46. Explain the process of compilation and linking.

In order to compile new extensions without any error, compiling and linking is used in Python. Linking initiates only and only when the compilation is complete.

In the case of dynamic loading, the process of compilation and linking depends on the style that is provided with the concerned system. In order to provide dynamic loading of the configuration setup files and rebuilding the interpreter, the Python interpreter is used.

47. What is Flask and what are the benefits of using it?

Flask is a web [microframework](#) for Python with Jinja2 and Werkzeug as its dependencies. As such, it has some notable advantages:

- Flask has little to no dependencies on external libraries.
- Because there is a little external dependency to update and fewer security bugs, the web microframework is lightweight.
- It has an inbuilt development server and a fast debugger.

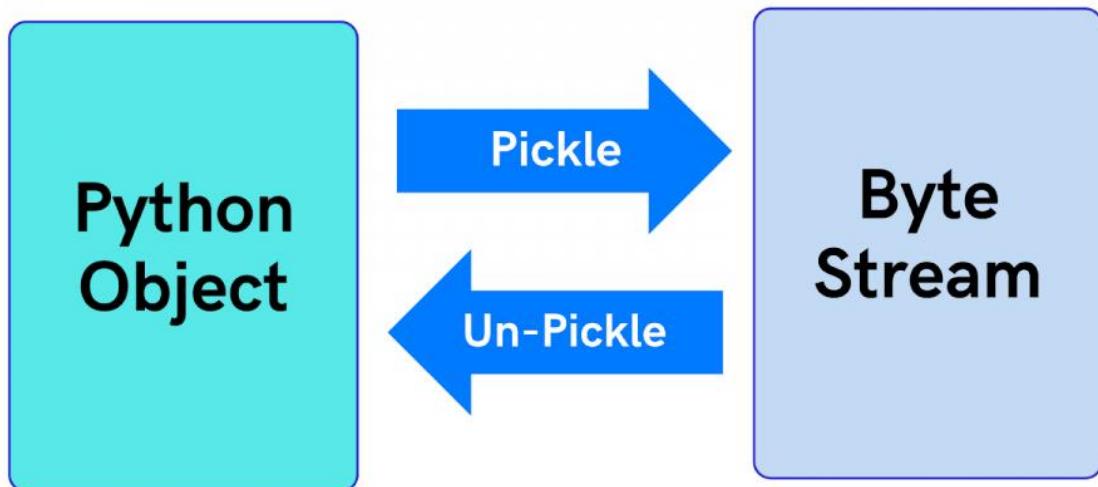
48. What is the map() function used for?

The map() function applies a given function to each item of an iterable. It then returns a list of the results. The value returned from the map() function can then be passed on to functions to the likes of the list() and set().

Typically, the given function is the first argument and the iterable is available as the second argument to a map() function. Several tables are given if the function takes in more than one argument.

49. What is Pickling and Unpickling?

>hackr.io



The Pickle module in Python allows accepting any object and then converting it into a string representation. It then dumps the same into a file by means of the dump function. This process is known as pickling. Unpickling retrieves the original Python objects from a stored string representation.

50. Whenever Python exits, not all the memory is deallocated. Why is it so?

Upon exiting, Python's built-in effective cleanup mechanism comes into play and tries to deallocate or destroy every other object. However, Python modules that have circular references to other objects, or the objects that are referenced from the global namespaces, aren't always deallocated or destroyed.

This is because it is not possible to deallocate those portions of the memory that are reserved by the C library.

51. Write a program in Python for getting indices of N maximum values in a NumPy array.

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

Output:

[4 3 1]

52. Write the code and output to show randomized items of a list.

Answer:

```
from random import shuffle  
  
x = ['hackr.io', 'Is', 'The', 'Best', 'For', 'Learning', 'Python']  
  
shuffle(x)  
  
print(x)
```

Output:

```
['For', 'Python', 'Learning', 'Is', 'Best', 'The', 'hackr.io']
```

53. How is memory managed in Python?

Python private heap space takes place of memory management in Python. It contains all Python objects and data structures. The interpreter is responsible to take care of this private heap and the programmer does not have access to it. The Python memory manager is responsible for the allocation of Python heap space for Python objects. The programmer may access some tools for the code with the help of the core API. Python also provides an inbuilt garbage collector, which recycles all the unused memory and frees the memory and makes it available to heap space.

54. What is the lambda function?

A lambda function is an anonymous function. This function can have only one statement but can have any number of parameters.

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

55. How are arguments passed in Python? By value or by reference?

All of the Python is an object and all variables hold references to the object. The reference values are according to the functions; as a result, the value of the reference cannot be changed.

56. What are the built-in types provided by Python?

Mutable built-in types:

- Lists
- Sets
- Dictionaries

Immutable built-in types:

- Strings
- Tuples
- Numbers

57. What are Python modules?

A file containing Python code like functions and variables is a Python module.
A Python module is an executable file with a .py extension. Some built-in modules are:

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

58. What is the // operator? What is its use?

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

59. What is the split function used for?

The split function breaks the string into shorter strings using the defined separator. It returns the list of all the words present in the string.

60. Explain the Dogpile effect.

The dogpile effect is when the cache expires and websites are hit by multiple requests made by the client at the same time. Using a semaphore lock prevents the Dogpile effect. In this system when a value expires, the first process acquires the lock and starts generating a new value.

61. What is a pass in Python?

The no-operation Python statement refers to a pass. It is a placeholder in the compound statement, where there should have a blank left or nothing written there.

62. What is [::-1] used for?

[::-1] reverses the order of an array or a sequence. However, the original array or the list remains unchanged.

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

63. How do you capitalize the first letter of string?

The capitalize() method capitalizes the first letter of the string, and if the letter is already capital it returns the original string

64. What are the is, not and in operators?

Operators are functions that take two or more values and return the corresponding result.

- **is:** returns true when two operands are true
- **not:** returns inverse of a boolean value
- **in:** checks if some element is present in some sequence.

65. How are modules imported in Python?

Modules are imported using the import keyword in either of the following three ways:

```
import array  
  
import array as arr  
  
from array import *
```

Q1. What is Python?

Ans. Python is a highly readable object-oriented programming language with automatic memory management. It is the language that can be written with English keywords, while other languages use punctuations.

Also called a scripting language, Python is easy to learn, portable, and open-source. It is designed to solve simple as well as complicated operations. It also supports multiple languages such as C, C++, and Java.

Example: To add two numbers

N1 = 1.5

N2 = 6.3

sum = N1 + N2

```
print('The sum of {0} and {1} is {2}'.format(N1, N2, sum))
```

Output:

The sum of 1.5 and 6.3 is 7.8

Q2. Explain the key features of Python.

Ans. Following are multiple features that make python different from other languages:

- **Easy to learn:** Python is a programmer-friendly language that it is easy to learn in just a few days. It uses fewer keywords as compared to other object-oriented languages and anyone can learn it
- **Object-oriented:** It is both procedure-oriented and object-oriented programming language. In procedure-oriented languages, procedures can be used again. It allows developers to use an object-oriented approach to develop applications.
- **Open-source and free:** It is an Open-source language that means anyone can easily access the programming and development. There is an open forum online where the number of coders contributes to improving this language.
- **High-level language:** It is defined as a high-level language because of this feature. You do not need to keep an eye on the programming structure, memory management, and architecture of the code.

- **Extendable & Scalable:** It is extendable as it can be extended by using different modules to its interpreters. It helps developers to modify the program. It also provides scalability to the large codes by providing support and good structure to it.

Q3. What is PEP 8?

Ans. PEP 8 (Python Enhancement Proposal) is a Python guide used for the formatting of code. It helps to increase the readability and provide the functionality to the source code.

Q4. Why is Python different from other scripting languages?

Ans. Python is different from other programming languages in many ways:

- Python is easy to read and write
- It is free. Users can easily share, copy, and edit it.
- It can be used with multiple platforms, so the programmers do not have the platform issue.
- It is an object-oriented programming language and a piece of programming code can be reusable.

Q5. Explain memory management.

Ans. In Python, memory management has different components that are allocated to do various tasks like sharing, caching, and segmentation. Python memory is managed by a private heap space. It also has a garbage collector that removes unwanted memory and free the heap space.

Following are two parts of memory:

- stack memory
- heap memory

Q6. What are pickling and unpickling in Python? Explain with example.

Ans. Pickling: It is the process in which objects are serialized and unserialized before it can be saved to the disk. It converts python objects into byte codes.

Unpickling: Unpickling is the opposite of pickling as it converts bytecode into python objects.

Example: Pickling

```
import pickle

def pickle_data():

    data = {

        'Emp name': 'John',
        'profession': 'Python Developer',
        'country': 'America'

    }

    filename = 'PersonalInfo'

    outfile = open(filename, 'wb')

    pickle.dump(data,outfile)

    outfile.close()

pickle_data()
```

Example: Unpickling

```
import pickle

def unpickling_data():

    file = open(filename,'xy')

    new_data = pickle.load(file)

    file.close()

    return new_data

print(unpickling_data())
```

Q7. What are the modules in Python? Name some built-in modules.

Ans. Python modules are the files that contain Python statement and definitions such as –

E.g., demo.py

Here, the module name is “demo.”

Python modules are used to break the large program into small segments that make the code manageable and organized.

Q8. What are the basic data types in Python?

Ans. There are five basic data types:

- Numbers
- String
- List
- Tuple
- Dictionary

Q9. What is the use of tuple?

Ans. Tuple is a data type similar to a list. It is used in programs to group the related data equivalent to the directory.

Example:

```
list_val = [5, 4, 3, 2]
```

```
tup_val = (5, 4, 3, 2)
```

```
print(list_val)
```

```
print(tup_val)
```

Output:

```
[5, 4, 3, 2]
```

```
(5, 4, 3, 2)
```

Q10. Explain local variables and global variables.

Ans. Global variables are declared outside the function and it can be used both inside and outside the function. It has a global scope.

Local variables are those which are declared inside the function or in the local scope which cannot be used outside the function.

Example:

A combination of global and local variable declaration

```
def foo(x, y):
```

```
    global a
```

```
    a = 62
```

```
    x,y = y,x
```

```
    b = 22
```

```
    b = 15
```

```
    c = 100
```

```
    print(a,b,x,y)
```

```
a, b, x, y = 1, 15, 2,3
```

```
foo(17, 3)
```

```
print(a, b, x, y)
```

Output:

```
62 15 3 15
```

```
22 15 2 3
```

Q11. What is the difference between Python Arrays and Lists?

Ans. Lists and arrays both are used to store data, but the difference is that the List can hold any data type while arrays can hold a single data type.

Example:

```
import array as arr  
  
My_Array=arr.array('i',[4,3,2,1])  
  
My_list=[5,'abc',1.20]  
  
print(My_Array)  
  
print(My_list)
```

Output:

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

Q12. What is lambda in Python?

Ans. Lambda is a keyword used to declare any function and that function is called lambda function. It is a simple function listed as an argument.

Example:

```
# Python code to illustrate square of a number  
  
# showing difference between def() and lambda().  
  
def square(a):  
  
return a*a;  
  
g = lambda b: b*b  
  
print(g(3))  
  
print(square(5))
```

Output:

9

25

Q13. Briefly explain Functions in Python.

Ans. Functions are the small segment of code, which are reusable codes used to perform actions when it is called. “def” keyword is used to define any function in Python.

Example:

```
def printstar( q ):
```

“This prints a passed string into this function”

```
print q
```

Return

Q14. Is Python an Interpreted or a Compiled language?

Ans. Python is an interpreted language. It converts the source code that is written by the programmer into an intermediate language, which is again translated into machine language before being executed.

Q15. Write a program to print odd numbers in a List?

Ans. We are using for loop to print odd numbers in a list

```
list1 = [10, 21, 4, 45, 66, 93] //list of numbers
```

```
for num in list1:
```

```
    if num % 2 != 0:
```

```
        print(num, end = " ")
```

Output:

21 45 93

Q16. What is a Dictionary in Python?

Ans. Dictionary in python is an associative array (also known as hashes) and is like a built-in mapping.

Q17. Are arguments passed by value or reference in Python?

Ans. In Python, arguments are always passed by value.

Q18. What is Web Scraping? How do you achieve it?

Ans. Web Scraping is a way of extracting the large amounts of information available on the websites and saving it onto the local machine or the database tables. Python has a few modules for scraping the web like urllib2, scrapy, pyquery, BeautifulSoup, etc.

Q19. How do you copy an object in Python?

Ans. We can copy an object by using the functions `copy.copy()` and `copy.deepcopy()`

Q20. Name a few libraries in Python used for data analysis and scientific computations.

Ans. Some of the libraries used for data analysis and scientific computations are NumPy, SciPy, Pandas, SciKit, Matplotlib, Seaborn.

Q21. Why should one use NumPy arrays instead of nested Python lists?

Ans. NumPy's arrays are more compact than Python lists — a list of lists as you describe, in Python, would take at least 20 MB or so, while a NumPy 3D array with single-precision floats in the cells would fit in 4 MB.

Q22. What is a Python decorator?

Ans. A Python decorator is a specific change to the Python syntax that allows us to alter functions and methods more conveniently.

Q23. What is Monkey Patching? How can you do it in Python?

Ans. Monkey Patching is the process of making changes to a module or class while the program is running. A Monkey Patch is a piece of code that extends or modifies other code at runtime (typically at startup).

Q24. What is a unittest?

Ans. The unit testing framework of Python is known as unittest. It has similar features with unit testing frameworks in other languages.

Unittest supports some important concepts of object-oriented Programming:

- Test fixture
- Test case
- Test suite
- Test runner

Example:

```
import unittest

class ABC(unittest.TestCase):

    def xyz():

        ...

    if __name__ == "__main__":
        unittest.main()
```

Q25. What is a negative index?

Ans. Python sequences can be indexed as positive and negative numbers. A negative index accesses elements from the end of the list counting backward.

Q26. What is the difference between Xrange() and range()?

Ans. Range() returns a list and Xrange() returns an Xrange object, which is kind of like an iterator and generates the numbers on demand.

Example:

```
a = range(1,10000)

# initializing a with xrange()
x = xrange(1,10000) testing the type of a
i ("The return type of range() is : ")
print (type(a))

# testing the type of x
print ("The return type of xrange() is : ")
print (type(x))
```

Output:

The return type of range() is :

<type 'list'>

The return type of xrange() is :

<type 'xrange'>

Q27. Define module and package.

Ans. A module is a Python object with arbitrarily named attributes that you can bind and reference.

A Python package is simply a directory of Python module(s).

Q28. Why don't lambda forms have statements?

Ans. It is because a lambda form is used to make a new function object and then return at runtime. Also, the syntactic framework of Python is unable to handle statements nested inside expressions.

Q29. What is Flask?

Ans. Flask (source code) is a Python micro web framework and it does not require particular tools or libraries. It is used for deploying python code into web apps.

Q30. How will you perform static analysis in a Python application or find bugs?

Ans. PyChecker can be helpful as a static analyzer to identify the bugs in the Python project. This also helps to find out the complexity-related bugs. Pylint is another tool that helps check if the Python module is at par with the coding standards.

Q31. When will you use the Python Decorator?

Ans. Python Decorator is used to adjusting the functions in Python syntax quickly.

Q32. Are Python strings immutable or mutable?

Ans. This is among the very commonly asked Python interview questions.

Python strings are immutable. Ironically, it is not a string, but a variable with a string value.

Q33. What is a pass in Python?

Ans. Pass stands for no-operation Python statement. It means that pass is a null operation; nothing happens when it is executed.

You might come across some confusing Python interview questions, MCQ is one of them, but do not get stumped. You should be thorough with your study and be well prepared for the Python interview questions.

Q34. Now, choose the right answer –When “else” in try-except-else is executed?

- In case of any exception
- When no exception is there
- When an exception occurs in the except block
- Always

Ans. c) when no exception occurs

Q35. What is slicing?

Ans. Slicing is a computationally fast way to methodically access a range of items from sequence types like list, tuple, strings, etc.

Q36. What is the output of the following code?

```
list1 = [5,4,3,2,1]
```

```
list2 = list1
```

```
list2[0] = 0;
```

```
print "list1= : ", list1
```

Ans.

Output:

```
list1= : [5,4,3,2,1]
```

Q37. How is the last object from a list removed?

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

Q38. What is docstring?

Ans. Python documentation strings (docstrings) provide an easy way to document Python functions, modules, and classes.

Q39. How to delete a file?

Ans. We can delete a file in Python by using a command `os.remove(filename)` or `os.unlink(filename)`.

Q40. What is the output of the following code?

`x = True`

`y = False`

`z = False`

`if x or y and z:`

`print "HELLOWORLD"`

`else:`

`print "helloworld"`

Ans. Output:

`HELLOWORLD`

Reason: Here, in Python AND operator has a higher preference than OR operator. So, (y and z) are evaluated first.

Q41. How many kinds of sequences does Python support? Name them.

Ans. Python supports seven sequence types –

- Str
- List
- Tuple

- Unicode
- byte array
- xrange
- buffer

Q42. How will you reload a Python module?

Ans. `reload()` is used to reload a previously imported module.

Q43. What is a set?

Ans. A [Python set](#) is an unordered collection of iterable and mutable data, and it has no duplicate elements.

Must Read: [Sets in Python](#)

Q44. Name some standard Python errors.

Ans. Some standard errors are –

- `TypeError`
- `ValueError`
- `NameError`
- `IOError`
- `IndexError`
- `KeyError`

We can use `dir(__builtin__)` to list all the errors.

Q45. What is Tkinter?

Ans. Tkinter is the de-facto standard GUI (Graphical User Interface) package of Python.

Q46. What is multithreading?

Ans. Multithreading stands for running a number of programs simultaneously by invoking multiple threads.

Example:

```
from threading import *
print(current_thread().getName())
def mt():
    print("Child Thread")
child=Thread(target=mt)
child.start()
print("Executing thread name :",current_thread().getName())
```

Output:

MainThread

Child Thread

Executing thread name : MainThread

Q47. How is a list reversed?

Ans. To reverse lists, one can use list.reverse()

Q48. How to capitalize the first letter of string?

Ans. To capitalize the first letter of the string, capitalize() method is used. If the string is already capitalized then it will return the original value.

Q49. Which python library is used for Machine learning?

Ans. Scikit-learn python Library is used for Machine learning.

Q50. What is the role of len() in python?

Ans. len() is used to determine the length of an array, list and string in the program.

Example:

```
str1='1234'
```

```
len(str1)
```

Q51. What is the output of the following code?

```
class Demo:
```

```
    def __init__(self, id):
```

```
        self.id = id
```

```
    id = 777
```

```
    acc = Acc(222)
```

```
    print acc.id
```

Ans. Output: 222

Reason: “Demo” class automatically calls the method “initl” and passes the object and “222” is assigned to the object called id. So, the value “777” cannot be called and the output will be “222”.

Q52. How to delete a file in Python?

Ans. OS Module needs to be installed to delete any file. After installing the module, os.remove() function is used to delete a file.

Q53. Write a code to test whether the number is in the defined range or not?

Ans.

```
def test_range(n1):
    if n1 in range(0, 555):
        print("%s is in range"%str(n1))
    else:
        print("%s is not in range"%str(n1))
```

Output:

```
test_range(555)
```

555 is not in the range

Q54. Write a code to convert a string into lowercase?

Ans. lower() is used to convert the string into lower case

```
str='XYZ'
```

```
print(str.lower())
```

Output: xyz

Q55. What is the output of the following code?

```
nameList = ['Joe', 'Nick', 'Bob', 'Harry']
```

```
print nameList[1][-1]
```

Ans. Output:

k

Reason: [-1] shows the last element or character of the string. In the above code, [1] represents the second string and [-1] represents the last character of the second string, i.e., “k.”

Q56. Which databases are supported by Python?

Ans. MySQL (Structured) and MongoDB (Unstructured) are supported by Python. First, the modules should be imported to the library to interact with the database.

Q57. What is the output of the following code?

```
demoCodes = [1, 2, 3, 4]  
demoCodes.append([5,6,7,8])  
print len(demoCodes)
```

Ans. Output: 5

Reason: ‘append’ method is used in the code, which has to append the existing object into the list. But the append method does not merge the list, which is added as an element. So, the output will be ‘5’.

Q58. What is the use of the ‘#’ symbol in Python?

Ans. ‘#’ symbol is used to symbolize the comments

```
print ("I am a quick learner")  
#print ("I am a quick learner")
```

Q59. Suppose a list1 is [2, 44, 191, 86], what would be the output for list1[-1]?

Ans. Output: 86

List1[-1] shows the last integer of the list

Q60. What is the maximum length of an identifier?

Ans. The maximum possible length of an identifier in python is 79 characters.

Q61. Can you tell me the generator functions in Python?

Ans. Generator functions help to declare a function that behaves like an iterator in a fast, easy, and neat way.

Q62. Write a code to display the current time.

Ans. Here is the code to represent the current time:

```
import datetime  
  
now = datetime.datetime.now()  
  
print ("Current date and time : ")  
  
print (now.strftime("%Y-%m-%d %H:%M:%S"))
```

Q63. Is Python a case-sensitive programming language?

Ans. Yes, it is a case-sensitive language like other languages such as Java, C, and C++.

Q64. Write a code to sort a numerical list in Python.

Ans. To sort a numerical list, use the following code:

```
list = ["2", "7", "3", "5", "1"]  
  
list = [int(i) for i in list]  
  
list.sort()  
  
print (list)
```

Q64. Write a code to display the contents of a file in reverse.

Ans. To reverse the content, use the following code:

```
for line in reversed(list(open(filename.txt))):  
  
    print(line.rstrip())
```

Q65. How to add array elements in programming?

Ans. We can add elements to an array with the help of append(), insert (i,y) and extend() functions.

Example:

```
x=arr.array('d', [1.2 , 2.2 ,3.2] )
```

```
x.append(3.3)
```

```
print(x)
```

```
x.extend([4.5,6.2,6.3])
```

```
print(x)
```

```
x.insert(2,3.8)
```

```
print(x)
```

Output:

```
array('d', [1.2, 2.2, 3.2, 3.3])
```

```
array('d', [1.2, 2.2, 3.2, 3.3, 4.5, 6.2, 6.3])
```

```
array('d', [1.2, 2.2, 3.8, 3.2, 3.3, 4.5, 6.2, 6.3])
```

Q66. Why is the split used in Python?

Ans. The split() method is used to separate two strings.

Example:

```
x="Naukri learning"
```

```
print(x.split())
```

Output: ['Naukri', 'learning']

Q67. How to create classes in Python?

Ans. Classes are user-defined which is defined with a class keyword

Example:

Class Student:

```
def __init__(self, name):
```

```
    self.name = name
```

```
S1 = Student ("xyz")
```

```
print (S1.name)
```

Output: xyz

Q68. How do we create an empty class?

Ans. An empty class is a blank class that does not have any code defined within its block. We can create an empty class using the pass keyword.

Example:

Class x:

```
&nbsp; pass
```

```
obj=x()
```

```
obj.name="xyz"
```

```
print ("Name = ",obj.name)
```

Output:

Name = xyz

Q69. Explain the difference between a shallow copy and a deep copy.

Ans. Shallow copy is used at the time of new instance creation, and it stores the copied values whereas in deep copy, the copying process executes in looping,

and copy of an object is copied in other objects. A shallow copy has faster program execution than a deep copy.

Q70. Which statement can we use if the statement is required syntactically, but no action is needed for the program?

Ans. Pass statement is used if the statement is required syntactically, but no action is required for the program

Example:

```
If(x>20)  
    print("Naukri")  
else  
    pass
```

Q71. What are the tools required to unit test your code?

Ans. To test units or classes, we can use the “unittest” python standard library. It is the easiest way to test code, and the features required are similar to the other unit testing tools like TestNG, JUnit.

Q72. How to get indices of N maximum values in a NumPy array?

Ans. With the help of below code, we can get the N maximum values in a NumPy array :

```
Import numpy as nm  
arr=nm.array([1, 6, 2, 4, 7])  
print (arr.argsort() [-3:] [::-1])
```

Output:

```
[ 4 6 1 ]
```

Q73. How can you use ternary operators (Ternary) in Python?

Ans. Ternary operators are used to display conditional statements. This consists of the true or false values. Syntax :

The ternary operator is indicated as:

[on_true] if [expression] else [on_false] x, y = 25, 50
big = x if x < y else y

Example: The expression is evaluated as if x < and else and, 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 returned as a result.

Q74. What does this mean? * args, ** kwargs? Why would we use it?

Ans. * Args is used when you are not sure how many arguments to pass to a function, or if you want to pass a list or tuple of stored arguments to a function.

** kwargs is used when we don't know how many keyword arguments to pass to a function, or it is used to pass the values from a dictionary as the keyword argument.

The args and kwargs identifiers are a convention, you can also use * bob and ** billy but that would not be wise

Q75. Does Python have OOps concepts?

Ans. Python is an object-oriented programming language. This means that any program can be solved in Python, creating an object model. However, Python can also be treated as a procedural and structural language.

Q76. What are compilation and linking process in Python?

Ans. Compilation and binding in Python allow new extensions to compile without any errors and binding can only be done when the build procedure passes. If dynamic loading is used, then it depends on the style supplied with the system. The Python interpreter can be used to provide dynamic loading of configuration files and rebuild the interpreter.

For this, the steps required in the process are:

Create a file with a name and in any language, which is compatible with your system compiler, example, file.co file.cpp

Locate the file in the Modules / directory of the distribution you are using.

Add a line in the Setup.local file that is present in the Modules / directory.

Run the file using spam file.o

After successful execution of this rebuild, the interpreter uses the make command in the top-level directory.

If the file is changed, then run rebuildMakefile using command like ‘make Makefile’.

Q77. How do I save an image locally using Python whose URL I already know?

Ans. We will use the following code to store an image locally from a URL

```
import urllib.request
```

```
urllib.request.urlretrieve ("URL", "file-name.jpg")
```

Q78. How can you get the time (age) of the Google cache of any URL or web page?

Ans. Using the following URL format:

```
http://webcache.googleusercontent.com/search?q=cache:EL-URL-VA-HERE
```

You should make sure to replace “EL-URL-GO-HERE” with the correct web address of the page or site to get the age of the Google cache.

Example – To check the age of Unipython’s Google web cache, you would use the following URL:

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

Q79. What is the map function in Python?

Ans. The map function takes two arguments, one is iterable and another is a function and applies the function to each element of the iterable. If the given function accepts more than 1 argument then many iterables are given.

Q80. How are percentages calculated with Python / NumPy?

Ans. Percentages can be calculated using the following code:

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

Q81. What is the difference between NumPy and SciPy?

Ans. Both NumPy and SciPy are modules of Python, and they are used for various operations of the data. NumPy stands for Numerical Python while SciPy stands for Scientific Python. The main differences are –

NumPy	SciPy
Makes Python an alternative to MatLab, IDL, and Yorick	A collection of tools for Python, used for general numerical computing in Python
Used for efficient operation on homogeneous data that are stored in arrays	Support operations like integration, differentiation, gradient optimization, etc.
Multi-dimensional array of objects, used for basic operations such as sorting, indexing, and elementary functioning on the array data type	No related array or list concepts as it is more functional

Suitable for computation of data and statistics, and basic mathematical calculation

Suitable for complex computing of numerical data

Q82. How 3D graphics/visualizations are made using NumPy / SciPy?

Ans. Like 2D plotting, 3D graphics are beyond the scope of NumPy and SciPy, but there are packages that can be integrated with NumPy. However, Matplotlib supplies basic 3D plotting in the mplot3d sub-package, while Mayavi offers a host of high-quality 3D viewing features, using the powerful VTK engine.

Q83. What is PYTHONPATH?

Ans. It is an environment variable and is used when importing a module. In addition, PYTHONPATH is used to check the presence of imported modules in some directories. The interpreter uses it to determine which module to load.

Q84. How to install Python on Windows and set a path variable?

Ans. – Install Python from this link: <https://www.python.org/downloads/>
– After that, install it on your PC. Find the location where Python has been installed on your PC using the following command on the command line: cmd
python
– Go to advanced system settings and add a new variable and name it PYTHON_NAME, and paste the copied path
– Find the path variable, select its value and select ‘edit’
– Add a semicolon after the value if it is not present and then write%
PYTHON_HOME%

Q85. Is indentation required in Python?

Ans. Indentation is very important in Python. It specifies a block of code. All the code within classes, functions, loops, etc., is specified within an indented block. Generally, this is done using four space characters. If your code is not indented, it will not execute accurately and will throw errors.

Q86. What is the Self in Python?

Ans. The Self in Python is an instance or object of a class. It is explicitly included as the first parameter. This helps distinguish between methods and attributes of a class with local variables.

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

Q87. How does break, continue and pass work?

Ans. Break – It allows the termination of the loop when some condition is met and control is transferred to the next instruction.

Continue – This lets you skip some part of a loop when a specific condition is met and control is transferred to the beginning of the loop.

Pass – It is used when a block of code is needed syntactically, however, its execution needs to be skipped. This is a null operation. Nothing happens when this runs.

Q88. What are the iterators in Python?

Ans. Iterators in Python are objects used to iterate all the elements of a collection.

Q89. How can you generate random numbers in Python?

Ans. The random module is the standard module for generating a random number. The method is defined as:

```
import random
```

`random.random`

The `random.random()` declaration returns the floating-point number that is in the range of [0, 1]. The function generates random floating numbers. The methods used with the `random` class are the bound methods of the hidden instances.

Random instances can be made to display multithreading programs that create a distinct instance of individual threads. The other random generators used are:

`randrange(a, b)`: choose an integer and define the range between [a, b). Returns elements by randomly picking them from the specified range. It does not construct a range object.

`Uniform(a, b)`: select a floating-point number that is defined in the range of [a, b). It returns the floating-point number.

`normalvariate(mean, sdev)`: used for normal distribution where `mu` is mean and `sdev` is sigma used for standard deviation.

Q90. What is the Dogpile effect?

Ans. This is one of those difficult Python interview questions to memorize at first, so give it a few tries.

The Dogpile effect occurs when a website's cache has expired and is hit by numerous requests at the same time. This causes a variety of problems, from increased large lag to massive errors. A system called traffic light blocking is used to prevent the effect of Dogpiles.

Q91. Explain what is encapsulation?

Ans. Encapsulation is one of the characteristics of the Python language because it is an object-oriented programming language. Encapsulation is the process of grouping data sets in one and only place. Along with members, encapsulation also returns functions.

Q92. When does Abnormal Termination occur?

Ans. First of all, I should mention that abend or abnormal termination is bad. You don't want it to happen during your programming experience. However, it is practically unavoidable, in one way or another especially when you are a beginner.

Abend is an error in your program during its execution, while the main tasks continue to perform processes. This is caused by a code error or some software problem.

Q93. Does the Python language have a compiler?

Ans. This is one of the most difficult Python interview questions, especially since so many people don't pay attention to it. Python clearly has a compiler, but it is very easy to miss. This is because it works automatically, you won't even notice.

Q94. What is polymorphism in Python?

Ans. Polymorphism is the ability to take many forms, for example, if the parent class has a method called ABC, it means that the child class can have a method with the same name ABC. This contains its own parameters and variables. Polymorphism is allowed in Python.

Q95. How is data abstraction done in Python?

Ans. It can be achieved in Python using abstract classes and interfaces. Data abstraction only supplies the necessary details and hides the implementation.

Abstraction is selecting data from a larger pool to show only the relevant details to the object.

Q96. Does Python use access specifiers?

Ans. Python does not have access modifiers. Rather it establishes the concept of prefixing the variable, method, or function name with a single or double underscore to mimic the behavior of the private and protected access specifiers.

Q97. Explain the bytes() function in Python.

Ans. The bytes() function in Python returns a bytes object. It converts objects into bytes objects. It also creates empty bytes objects of the specified size.

Q98. Explain the ‘with statement’.

Ans. In Python, the ‘with statement’ is used for exception handling and resource management. It makes the code cleaner and readable as it allows a file to be opened and closed while executing a block of code containing the ‘with statement’.

Q99. What is Pandas?

Ans. Pandas is an open-source data analysis and manipulation tool built on top of the Python programming language. It is a Python library that provides fast and high-performance data structures and data analysis tools. It is widely used for data science and machine learning tasks.

Pandas is built on top of Numpy that offers flexibility in creating data structures for data science, enabling you to create multidimensional, tabular, heterogeneous, data structures. It also lets users perform data manipulation and time series. Python with Pandas is used in different domains like analytics, finance, economics, statistics, etc.

Q100. What is a Pandas Series?

Ans. Pandas Series is a one-dimensional array that can hold data of any type, like integer, string, float, python objects. A Pandas Series is like a column in an excel sheet. It represents a single column in memory, which can either belong to or be independent of a DataFrame.

We can create a Pandas Series using the below constructor –

pandas.Series(data, index, dtype, copy)

Example:

The below code will create a Series from ndarray. We will import a numpy module and use array() function.

```
# import pandas as pd  
import pandas as pd  
  
# import numpy as np  
import numpy as np  
  
data = np.array(['n', 'a', 'u', 'k', 'r', 'i'])  
  
s = pd.Series(data)  
  
print s
```

Output:

```
0    n  
1    a  
2    u  
3    k  
4    r  
5    i  
  
dtype: object
```

Q101. What are Pandas DataFrames?

Ans. Pandas DataFrame is a two-dimensional tabular data structure with labeled axes. The data is aligned in a tabular manner in rows and columns. DataFrames are widely used in data science, machine learning, scientific computing, etc.

Here are some features of Dataframes:

- 2-dimensional
- Labeled axes (rows and columns)

- Size-mutable
- Arithmetic operations can be performed on rows and columns

Example:

The below code will create a DataFrame using a single list or a list of lists.

```
# import pandas as pd  
  
import pandas as pd  
  
data = [1,2,3,4,5]  
  
df = pd.DataFrame(data)  
  
print df
```

Output:

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

Q102. How to combine DataFrames in Pandas?

Ans. We can combine DataFrames using the following functions:

- **concat()** function: It is used for vertical stacking.

```
pd.concat([data_frame1, data_frame2])
```

- **append()**: It is used for horizontal stacking of DataFrames.

```
data_frame1.append(data_frame2)
```

- **join()**: It is used to extract data from different DataFrames which have one or more columns common.

```
data_frame1.join(data_frame2)
```

Q103. How to access the top n rows of a dataframe?

Ans. To access the top n rows of a dataframe, we will use df.head(n).

Q104. How to access the last n rows of a dataframe?

Ans. We will use df.tail(n) to access the last n rows of a dataframe

Q105. What are Python namespaces?

Ans. A namespace is a mapping from names to objects. It is a system that ensures that all the object names in a program are unique and can be used without any conflict. Python maintains a namespace in the form of a Python dictionary. These namespaces are implemented as dictionaries with ‘name as key’ mapped to its respective ‘object as value’. Namespaces have different lifetimes as they are often created at different points in time.

Some of the namespaces in a Python program are:

- **Local Namespace** – it contains local names inside a function. The local namespace is created for a function call and lasts until the function returns.
- **Global Namespace** – It consists of the names from various imported modules that are being used in the ongoing project. This namespace is created when the package is imported into the script and lasts until the script ends.
- **Built-In Namespace** – This namespace contains built-in functions and built-in exception names.

Q106. What is Inheritance in Python?

Ans. Inheritance is the capability of classes in Python to inherit the properties or attributes of another class. A new class is defined with little or no modification to an existing class. The new class is called derived or child class and the class from which it inherits is called the base or parent class. Inheritance provides the code reusability feature. It is transitive, which means that if class B inherits from another class A, then all the subclasses of B would automatically inherit from class A.

Python supports the following types of inheritance:

- **Single Inheritance:** A class inherits only one superclass.
- **Multiple Inheritance:** A class inherits multiple superclasses.
- **Multilevel Inheritance:** Features of the base class and the derived class are further inherited into the new derived class. A class inherits a superclass, and then another class inherits the derived class forming a ‘parent, child, and grandchild’ class structure.
- **Hierarchical inheritance:** More than one derived class are created from a single base class.

Q107. What are Python literals?

Ans. Literals refer to the raw value or data given in a variable or constant. They represent a fixed value in the source code. In Python, there are various types of literals:

- String literals
- Numeric literals
- Boolean literals
- Literal Collections
- Special literals
- **String Literals:** These are created by enclosing text in single or double-quotes. There are two types of Strings Single-line and Multi-line String.

Example: “Literal” , ‘12345’

- **Numeric Literals:** They support three types of literals:

- Integer:I=20

- Float: i=3.6
- Complex:2+7j

- Boolean Literals:** There are two boolean literals – true and false.
- Literal Collections:** There are four different types of literal collections:

- List literals
- Tuple literals
- Set literals
- Dict literals

- Special Literals:** Python supports one special literal i.e., None. None specifies that field that is not created.

Q108. Write a Python program to produce half pyramid using *.

Ans. Below is the code to print half pyramid using *

```
n = int(input("Enter the number of rows"))
```

```
for i in range(0, n):
```

```
    for j in range(0, i + 1):
```

```
        print("* ", end="")
```

```
    print()
```

Output:

```
*
```

```
* *
```

```
* * *
```

```
* * * *
```

```
* * * * *
```

Q109. Write a python program to check if the number given is a palindrome or not

Ans. Below is the code to check if the given number is palindrome or not:

```
num=input("Enter a number:")  
if num==num[::-1]  
    print ("It is a Palindrome!")  
else:  
    print("It is not a Palindrome!")
```

Output:

Case 1:

Enter a number: 12321

It is a Palindrome!

Case 2:

Enter number: 5678

It is not a Palindrome!

Q110. Write a sorting algorithm for a numerical dataset in Python.

Ans. Below is the sorting algorithm for a numerical dataset in Python

```
list = ["9", "5", "2", "0", "8"]  
list = [int(i) for i in list]  
list.sort()  
print (list)
```

Q111. Explain how can you make a Python Script executable on Unix?

Ans: Python Script file must begin with `#!/usr/bin/env python`

Q112. What is Scope Resolution in Python?

Ans: In some cases, objects within the same scope have the same name. However, they work differently. In such cases, scope resolution help in Python automatically.

For Example:- Python modules namely ‘math’ and ‘cmath’ have a few function names in common. Such as –

```
log10()
```

```
, acos()
```

```
exp(),
```

etc. To resolve this ambiguity, it's essential to prefix each function with its respective module, such as:

```
math.exp()
```

[Copy code](#)

and

```
cmath.exp()
```

[Copy code](#)

Q.1. What are the key features of Python?

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

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

Q.2. Differentiate between lists and tuples.

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

Examples:

```
>>> mylist=[1,3,3]
>>> mylist[1]=2
>>> mytuple=(1,3,3)
>>> mytuple[1]=2
```

Traceback (most recent call last):

File “<pyshell#97>”, line 1, in <module>

mytuple[1]=2

TypeError: ‘tuple’ object does not support item assignment

Q.3. Explain the ternary operator in Python.

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

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

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

Below is how you would use it:

```
>>> a,b=2,3
>>> min=a if a<b else b
>>> min
2
>>> print("Hi") if a<b else print("Bye")
```

Hi

Q.4. What are negative indices?

Let’s take a list for this.

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

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

```
>>> mylist[-3]
```

6

This also helps with slicing from the back:

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

[3, 4, 5, 6, 7]

Q.5. Is Python case-sensitive?

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

```
>>> myname='Ayushi'
```

```
>>> Myname
```

Traceback (most recent call last):

File “<pyshell#3>”, line 1, in <module>

Myname

NameError: name ‘Myname’ is not defined

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

Q.6. How long can an identifier be in Python?

According to the official Python documentation, an identifier can be of any length. However, PEP 8 suggests that you should limit all lines to a maximum of 79 characters. Also, PEP 20 says ‘readability counts’. So, a very long identifier will violate PEP-8 and PEP-20.

Apart from that, there are certain rules we must follow to name one:

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

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

Q.7. How would you convert a string into lowercase?

We use the lower() method for this.

```
>>> 'AyuShi'.lower()
```

'ayushi'

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

```
>>> 'AyuShi'.upper()
```

'AYUSHI'

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

```
>>> 'AyuShi'.isupper()
```

False

```
>>> 'AYUSHI'.isupper()
```

True

```
>>> 'ayushi'.islower()
```

True

```
>>> '@yu$hi'.islower()
```

True

```
>>> '@YU$HI'.isupper()
```

True

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

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

```
>>> 'The Corpse Bride'.istitle()
```

True

Q.8. What is the pass statement in Python?

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

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

```
    pass
```

```
>>>
```

Similarly, the break statement breaks out of a loop.

```
>>> for i in range(7):
```

```
    if i==3: break
```

```
    print(i)
```

1

2

Finally, the continue statement skips to the next iteration.

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

1

2

4

5

6

Hope you have read all the basic Python Interview Questions and Answers. Now, let's move towards the second part of the blog – Most asked Python Interview Questions and Answers for freshers

Frequently Asked Python Interview Questions and Answers for Freshers

While solving or answering these questions, if you feel any difficulty, comment us. DataFlair is always ready to help you.

Q.9. Explain help() and dir() functions in Python.

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

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

Help on function copy in module copy:

copy(x)

Shallow copy operation on arbitrary Python objects.

See the module's __doc__ string for more info.

The dir() function displays all the members of an object(any kind).

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

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

Q.10. How do you get a list of all the keys in a dictionary?

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

For this, we use the function `keys()`.

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

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

Q.11. What is slicing?

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

```
>>> (1,2,3,4,5)[2:4]
```

`(3, 4)`

```
>>> [7,6,8,5,9][2:]
```

`[8, 5, 9]`

```
>>> 'Hello'[:-1]
```

`'Hell'`

Q.12. How would you declare a comment in Python?

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

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

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

Q.13. How will you check if all characters in a string are alphanumeric?

For this, we use the method `isalnum()`.

Q.14. How will you capitalize the first letter of a string?

Simply using the method `capitalize()`.

```
>>> 'ayushi'.capitalize()
```

`'Ayushi'`

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

`<class 'method_descriptor'>`

However, it will let other characters be.

```
>>> '@yushi'.capitalize()
```

```
'@yushi'  
>>> 'Ayushi123'.isalnum()
```

True
>>> 'Ayushi123!'.isalnum()

False
Other methods that we have include:

```
>>> '123.3'.isdigit()  
False  
>>> '123'.isnumeric()
```

True
>>> 'ayushi'.islower()

True
>>> 'Ayushi'.isupper()

False
>>> 'Ayushi'.istitle()

True
>>> ' '.isspace()

True
>>> '123F'.isdecimal()

False
Q.15. We know Python is all the rage these days. But to be truly accepting of a great technology, you must know its pitfalls as well. Would you like to talk about this?

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

Limitations of Python

Design restrictions

Slow when compared to C/C++ or Java

Weak in mobile computing

Underdeveloped database access layers

- Python's interpreted nature imposes a speed penalty on it.
- While Python is great for a lot of things, it is weak in mobile computing, and in browsers.
- Being dynamically-typed, Python uses duck-typing (If it looks like a duck, it must be a duck). This can raise runtime errors.
- Python has underdeveloped database access layers. This renders it a less-than-perfect choice for huge database applications.
- And then, well, of course. Being easy makes it addictive. Once a Python-coder, always a Python coder.

Q.16. With Python, how do you find out which directory you are currently in?

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

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

```
'C:\\Users\\lifei\\AppData\\Local\\Programs\\Python\\Python36-32'  
>>> type(os.getcwd)
```

```
<class 'builtin_function_or_method'>
```

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

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

```
'C:\\Users\\lifei\\Desktop'
```

Q.17. How do you insert an object at a given index in Python?

Let's build a list first.

```
>>> a=[1,2,4]
```

Now, we use the method insert. The first argument is the index at which to insert, the second is the value to insert.

```
>>> a.insert(2,3)
>>> a
```

[1, 2, 3, 4]

Q.18. And how do you reverse a list?

Using the reverse() method.

```
>>> a.reverse()
>>> a
```

[4, 3, 2, 1]

You can also do it via slicing from right to left:

```
>>> a[::-1]
>>> a
```

[1, 2, 3, 4]

This gives us the original list because we already reversed it once. However, this does not modify the original list to reverse it.

Q.19. What is the Python interpreter prompt?

This is the following sign for [Python Interpreter](#):

```
>>>
```

If you have worked with the IDLE, you will see this prompt.

Q.20. How does a function return values?

A function uses the ‘return’ keyword to return a value. Take a look:

```
>>> def add(a,b):
    return a+b
```

Q.21. How would you define a block in Python?

For any kind of statements, we possibly need to define a block of code under them. However, Python does not support curly braces. This means we must end such statements with colons and then indent the blocks under those with the same amount.

```
>>> if 3>1:
    print("Hello")
    print("Goodbye")
```

Hello
Goodbye

Q.22. Why do we need break and continue in Python?

Both break and continue are statements that control flow in [Python loops](#). break stops the current loop from executing further and transfers the control to the next block. continue jumps to the next iteration of the loop without exhausting it.

Q.23. Will the do-while loop work if you don't end it with a semicolon?

Trick question! Python does not support an intrinsic do-while loop. Secondly, to terminate do-while loops is a necessity for languages like C++.

Q.24. In one line, show us how you'll get the max alphabetical character from a string.

For this, we'll simply use the max function.

```
>>> max('flyiNg')  
'y'
```

The following are the ASCII values for all the letters of this string-

f- 102

l- 108

y- 121

i- 105

N- 78

g- 103

By this logic, try to explain the following line of code-

```
>>> max('fly{ }iNg')  
'{'
```

Q.25. What is Python good for?

Python is a jack of many trades, check out Applications of Python to find out more.

Meanwhile, we'll say we can use it for:

- Web and Internet Development
- Desktop GUI
- Scientific and Numeric Applications
- Software Development Applications
- Applications in Education
- Applications in Business
- Database Access

- Network Programming
- Games, 3D Graphics
- Other Python Applications

Q.26. Can you name ten built-in functions in Python and explain each in brief?

Ten Built-in Functions, you say? Okay, here you go.

`complex()`- Creates a complex number.

```
>>> complex(3.5,4)
```

(3.5+4j)

`eval()`- Parses a string as an expression.

```
>>> eval('print(max(22,22.0)-min(2,3))')
```

20

`filter()`- Filters in items for which the condition is true.

```
>>> list(filter(lambda x:x%2==0,[1,2,0,False]))
```

[2, 0, False]

`format()`- Lets us format a string.

```
>>> print("a={0} but b={1}".format(a,b))
```

a=2 but b=3

`hash()`- Returns the hash value of an object.

```
>>> hash(3.7)
```

644245917

`hex()`- Converts an integer to a hexadecimal.

```
>>> hex(14)
```

'0xe'

`input()`- Reads and returns a line of string.

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

Enter a number7

'7'

`len()`- Returns the length of an object.

```
>>> len('Ayushi')
```

6

`locals()`- Returns a dictionary of the current local symbol table.

```
>>> locals()
```

```
{'__name__': '__main__', '__doc__': None, '__package__': None,
 '__loader__': <class '__frozen_importlib.BuiltinImporter'>, '__spec__':
None, '__annotations__': {}, '__builtins__': <module 'builtins' (built-in)>,
 'a': 2, 'b': 3}
```

`open()`- Opens a file.

```
>>> file=open('tabs.txt')
```

Q.27. What will the following code output?

```
>>> word='abcdefghijkl'
>>> word[:3]+word[3:]
```

The output is ‘abcdefghijkl’. The first slice gives us ‘abc’, the next gives us ‘defghijkl’.

Q.28. How will you convert a list into a string?

We will use the `join()` method for this.

```
>>> nums=['one','two','three','four','five','six','seven']
>>> s=' '.join(nums)
>>> s
```

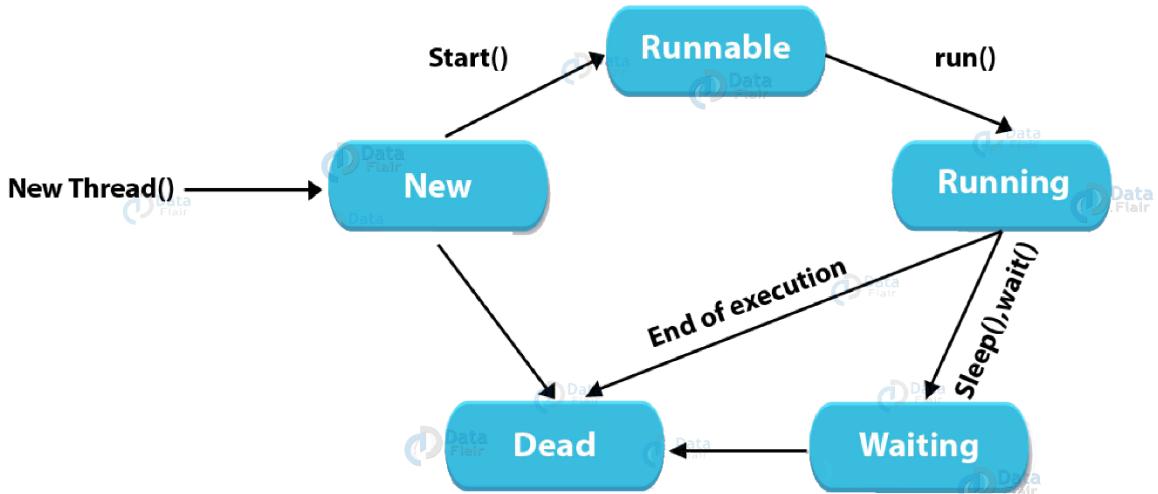
‘one two three four five six seven’

Q.29. How will you remove a duplicate element from a list?

We can turn it into a set to do that.

```
>>> list=[1,2,1,3,4,2]
>>> set(list)
{1, 2, 3, 4}
```

Q.30. Can you explain the life cycle of a thread?



- To create a thread, we create a class that we make override the run method of the thread class. Then, we instantiate it.
- A thread that we just created is in the new state. When we make a call to start() on it, it forwards the threads for scheduling. These are in the ready state.
- When execution begins, the thread is in the running state.
- Calls to methods like sleep() and join() make a thread wait. Such a thread is in the waiting/blocked state.
- When a thread is done waiting or executing, other waiting threads are sent for scheduling.
- A running thread that is done executing terminates and is in the dead state.

Basic Python Program Interview Questions and Answers

Q.31. What is a dictionary in Python?

A [python dictionary](#) is something I have never seen in other languages like C++ or Java programming. It holds key-value pairs.

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

`<class 'dict'>`

1. `>>> roots[9]`

`3`

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

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

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

Q.32. Explain the //, %, and ** operators in Python.

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

```
>>> 7//2
```

3

Normal division would return 3.5 here.

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

```
>>> 2**10
```

1024

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

```
>>> 13%7
```

6

```
>>> 3.5%1.5
```

0.5

Q.33. What do you know about relational operators in Python.

Python Relational Operators

Operator	Description
<	Less than
>	Greater than
<=	Less than or equal to
>=	Greater than or equal to
=	Equal to
!=	Not equal to

Relational operators compare values.

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

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

False

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

```
>>> 1.1+2.2>3.3
```

True

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

Less than or equal to (\leq) If the value on the left is lesser than or equal to, it returns True.

```
>>> 3.0<=3
```

True

Greater than or equal to (\geq) If the value on the left is greater than or equal to, it returns True.

```
>>> True>=False
```

True

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

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

True

Not equal to (\neq) If the two values are unequal, it returns True.

```
>>> True!=0.1
```

True

```
>>> False!=0.1
```

True

You will surely face a question from Python Operators. There are chances that question may be in an indirect way. Prepare yourself for it with the best guide – [Python Operators](#)

Q.34. What are assignment operators in Python?

Python Assignment Operators

Operator	Description
=	Assign
+=	Add and Assign
-=	Subtract and Assign
*=	Multiply and Assign
/=	Divide and Assign
%=	Modulus and Assign
**=	Exponent and Assign
//=	Floor-Divide and Assign

We can combine all arithmetic operators with the assignment symbol.

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

8

```
>>> a-=1  
>>> a
```

7

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

14

```
>>> a/=2  
>>> a
```

7.0

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

49.0

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

16.0

```
>>> a%=4  
>>> a
```

0.0

Q.35. Explain logical operators in Python.

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

>>> **False** and **True**

False

>>> $7 < 7$ or **True**

True

>>> not $2 == 2$

False

Q.36. What are membership operators?

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

>>> 'me' in 'disappointment'

True

>>> 'us' not in 'disappointment'

True

Q.37. Explain identity operators in Python.

The operators ‘is’ and ‘is not’ tell us if two values have the same identity.

>>> 10 is '10'

False

>>> **True** is not **False**

True

Q.38. Finally, tell us about bitwise operators in Python.

Python Bitwise Operators

Operator	Description
&	Binary AND
	Binary OR
^	Binary XOR
~	Binary One's Complement
<<	Binary Left-Shift
>>	Binary Right-Shift

These operate on values bit by bit.

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

```
>>> 0b110 & 0b010
```

2

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

```
>>> 3|2
```

3

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

```
>>> 3^2
```

1

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

```
>>> ~2
```

-3

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

```
>>> 1<<2
```

4

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

Binary Right-Shift (>>)

```
>>> 4>>2
```

1

Q.39. What data types does Python support?

Python provides us with five kinds of data types:

Numbers – Numbers use to hold numerical values.

```
>>> a=7.0
```

```
>>>
```

Strings – A string is a sequence of characters. We declare it using single or double quotes.

```
>>> title="Ayushi's Book"
```

Lists – A list is an ordered collection of values, and we declare it using square brackets.

```
>>> colors=['red','green','blue']
```

```
>>> type(colors)
```

```
<class 'list'>
```

Tuples – A tuple, like a list, is an ordered collection of values. The difference. However, is that a tuple is immutable. This means that we cannot change a value in it.

```
>>> name=('Ayushi','Sharma')
>>> name[0]='Avery'
```

Traceback (most recent call last):

File “<pyshell#129>”, line 1, in <module>

name[0]='Avery'

TypeError: ‘tuple’ object does not support item assignment

Dictionary – A dictionary is a data structure that holds key-value pairs. We declare it using curly braces.

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

<class ‘dict’>

```
>>> type({ })
```

<class ‘dict’>

We can also use a dictionary comprehension:

```
>>> squares={x:x**2 for x in range(1,6)}
>>> squares
```

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

Q.40. What is a docstring?

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

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

Hi

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

```
>>> sayhi.__doc__
'\n\tThis function prints Hi\n\t'
```

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

Q.41. How would you convert a string into an int in Python?

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

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

227

Let's check the types:

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

<class 'str'>

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

<class 'int'>

Q.42. How do you take input in Python?

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

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

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

Enter a number7

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

```
>>> type(a)
```

<class 'str'>

Multiplying this by 2 gives us this:

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

```
>>> a
```

'77'

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

We use the `int()` function for this.

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

Enter a number7

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

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

```
>>> a
```

Q.43. What is a function?

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

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

3.5

Q.44. What is recursion?

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

```
>>> def facto(n):  
    if n==1: return 1  
    return n*facto(n-1)  
>>> facto(4)
```

24

Q.45. What does the function zip() do?

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

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

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

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

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

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

Q.46. How do you calculate the length of a string?

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

```
>>> len('Ayushi Sharma')
```

13

Q.47. Explain Python List Comprehension.

The [list comprehension in python](#) is a way to declare a list in one line of code.

Let's take a look at one such example.

```
>>> [i for i in range(1,11,2)]
```

[1, 3, 5, 7, 9]

```
>>> [i*2 for i in range(1,11,2)]
```

[2, 6, 10, 14, 18]

Q.48. How do you get all values from a Python dictionary?

We saw previously, to get all keys from a dictionary, we make a call to the keys() method. Similarly, for values, we use the method values().

```
>>> 'd' in {'a':1,'b':2,'c':3,'d':4}.values()
```

False

```
>>> 4 in {'a':1,'b':2,'c':3,'d':4}.values()
```

True

Q.49. What if you want to toggle case for a Python string?

We have the swapcase() method from the str class to do just that.

```
>>> 'AyuShi'.swapcase()
```

'aYUsHI'

Let's apply some concepts now, shall we? Questions 50 through 52 assume the string 'I love Python'. You need to do the needful.

Q.50. Write code to print only upto the letter t.

```
>>> i=0  
>>> while s[i]!='t':  
print(s[i],end='')  
i+=1
```

I love Py

Q.51. Write code to print everything in the string except the spaces.

```
>>> for i in s:  
if i==' ': continue  
print(i,end="")
```

IlovePython

Q.52. Now, print this string five times in a row.

```
>>> for i in range(6):  
print(s)
```

I love Python
I love Python

Okay, moving on to more domains to conquer.

Q.53. What is the purpose of bytes() in Python?

`bytes()` is a built-in function in Python that returns an immutable bytes object. Let's take an example.

```
>>> bytes([2,4,8])  
b'\x02\x04\x08'  
>>> bytes(5)  
b'\x00\x00\x00\x00\x00'  
>>> bytes('world','utf-8')
```

`b'world'`

Q.54. What is a control flow statement?

A Python program usually starts to execute from the first line. From there, it moves through each statement just once and as soon as it's done with the last statement, it transactions the program. However, sometimes, we may want to take a more twisted path through the code. Control flow statements let us disturb the normal execution flow of a program and bend it to our will.

Q.55. Create a new list to convert the following list of number strings to a list of numbers.

`nums=['22','68','110','89','31','12']`

We will use the `int()` function with a list comprehension to convert these strings into integers and put them in a list.

```
>>> [int(i) for i in nums]
```

`[22, 68, 110, 89, 31, 12]`

Q.56. Given the first and last names of all employees in your firm, what data type will you use to store it?

I can use a dictionary to store that. It would be something like this-

```
{'first_name':'Ayushi','second_name':'Sharma'}
```

Q.57. How would you work with numbers other than those in the decimal number system?

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

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

```
>>> int(0b1010)
```

`10`

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

```
>>> bin(0xf)
```

‘0b1111’

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

```
>>> oct(8)
```

‘0o10’

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

```
>>> hex(16)
```

‘0x10’

```
>>> hex(15)
```

‘0xf’

DataFlair’s latest article on [Python Numbers with Examples](#)

Q.58. What does the following code output?

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

([10, ‘a’], [123], [10, ‘a’])

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

([10],[123],[‘a’])

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

Q.59. How many arguments can the range() function take?

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

a. One argument

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

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

[0, 1, 2, 3, 4]

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

```
[]  
>>> list(range(0))
```

```
[]  
b. Two arguments
```

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

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

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

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

```
[]  
>>> list(range(-3,4))
```

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

```
c. Three arguments
```

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

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

```
[2, 4, 6, 8]
```

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

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

```
Q.60. What is PEP 8?
```

PEP 8 is a coding convention that lets us write more readable code. In other words, it is a set of recommendations.

```
Q.61. How is Python different from Java?
```

Dimensions	Python	Java
Verbosity	Concise	Verbose
Performance	Interpreted, slower	Faster
Learning Curve	Easier than Java	Easy
Typing discipline	Dynamically-typed (duck-typing)	Statically-typed
Best for	Data Science, AI, Machine Learning	Embedded and cross-platform applications

Following is the comparison of Python vs Java –

- Java is faster than Python
- Python mandates indentation. Java needs braces.
- Python is dynamically-typed; Java is statically typed.
- Python is simple and concise; Java is verbose
- Python is interpreted
- Java is platform-independent
- Java has stronger database-access with JDBC

Q.62. What is the best code you can write to swap two numbers?

I can perform the swapping in one statement.

```
>>> a,b=b,a
```

Here's the entire code, though-

```
>>> a,b=2,3  
>>> a,b=b,a  
>>> a,b
```

(3, 2)

Q.63. How can you declare multiple assignments in one statement?

This is one of the most asked interview questions for Python freshers –

There are two ways to do this:

First –

```
>>> a,b,c=3,4,5 #This assigns 3, 4, and 5 to a, b, and c respectively
```

Second –

```
>>> a=b=c=3 #This assigns 3 to a, b, and c
```

Q.64. If you are ever stuck in an infinite loop, how will you break out of it?

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

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

```
7  
7  
7  
7  
7  
7  
7  
7
```

7
7
7
7
7
7
7
7
7
7
7
7
7
7

Traceback (most recent call last):

File "<pyshell#332>", line 1, in <module>
counterfunc(7)
File "<pyshell#331>", line 2, in counterfunc
while(n==7):print(n)
KeyboardInterrupt

Technical Python Interview Questions and Answers

Q.65. How do we execute Python?

Python files first compile to bytecode. Then, the host executes them.

Revise the concept of [Python Compiler](#)

Q.66. Explain Python's parameter-passing mechanism.

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

Q.67. What is the with statement in Python?

The with statement in Python ensures that cleanup code is executed when working with unmanaged resources by encapsulating common preparation and cleanup tasks. It may be used to open a file, do something, and then automatically close the file at the end. It may be used to open a database connection, do some processing, then automatically close the connection to ensure resources are closed and available for others. with will cleanup the resources even if an exception is thrown. This statement is like the using statement in C#.

Consider you put some code in a try block, then in the finally block, you close any resources used. The with statement is like syntactic sugar for that.

The syntax of this control-flow structure is:

with expression [as variable]:

....with-block

```
>>> with open('data.txt') as data:
```

```
#processing statements
```

Q.68. How is a .pyc file different from a .py file?

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

Python OOPS Interview Questions and Answers

Q.69. What makes Python object-oriented?

Again the frequently asked Python Interview Question

Python is object-oriented because it follows the Object-Oriented programming paradigm. This is a paradigm that revolves around classes and their instances (objects). With this kind of programming, we have the following features:

- Encapsulation
- Abstraction
- Inheritance
- Polymorphism
- Data hiding

Q.70. How many types of objects does Python support?

Objects in Python are mutable and immutable. Let's talk about these.

Immutable objects- Those which do not let us modify their contents. Examples of these will be tuples, booleans, strings, integers, floats, and complexes.

Iterations on such objects are faster.

```
>>> tuple=(1,2,4)
```

```
>>> tuple
```

(1, 2, 4)

```
>>> 2+4j
```

(2+4j)

Mutable objects – Those that let you modify their contents. Examples of these are lists, sets, and dicts. Iterations on such objects are slower.

```
>>> [2,4,9]
```

[2, 4, 9]

```
>>> dict1={1:1,2:2}
```

```
>>> dict1
```

{1: 1, 2: 2}

While two equal immutable objects' reference variables share the same address, it is possible to create two mutable objects with the same content.

1.

What will be the output of the code below? Explain your answer.

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

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

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

How would you modify the definition of `extendList` to produce the presumably desired behavior?

[Hide answer](#)

The output of the above code will be:

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

Many will mistakenly expect `list1` to be equal to `[10]` and `list3` to be equal to `['a']`, thinking that the `list` argument will be set to its default value of `[]` each time `extendList` is called.

However, what actually happens is that the new default list is created *only once* when the function is *defined*, and that same list is then used subsequently whenever `extendList` is invoked without a `list` argument being specified. This is because expressions in default arguments are calculated when the function is defined, not when it's called.

`list1` and `list3` are therefore operating on the *same* default list, whereas `list2` is operating on a separate list that it created (by passing its own empty list as the value for the `list` parameter).

The definition of the `extendList` function could be modified as follows, though, to *always* begin a new list when no `list` argument is specified, which is more likely to have been the desired behavior:

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

With this revised implementation, the output would be:

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

2.

What will be the output of the code below? Explain your answer.

```
def multipliers():  
    return [lambda x : i * x for i in range(4)]  
  
print [m(2) for m in multipliers()]
```

How would you modify the definition of `multipliers` to produce the presumably desired behavior?

Hide answer

The output of the above code will be `[6, 6, 6, 6]` (*not* `[0, 2, 4, 6]`).

The reason for this is that Python's closures are [late binding](#). This means that the values of variables used in closures are looked up at the time the inner function is called. So as a result, when *any* of the functions returned by `multipliers()` are called, the value of `i` is looked up in the surrounding scope *at that time*. By then, regardless of which of the returned functions is called, the `for` loop has completed and `i` is left with its final value of 3.

Therefore, every returned function multiplies the value it is passed by 3, so since a value of 2 is passed in the above code, they all return a value of 6 (i.e., 3×2).

(Incidentally, as pointed out in [The Hitchhiker's Guide to Python](#), there is a somewhat widespread misconception that this has something to do with lambdas, which is not the case. Functions created with a `lambda` expression are in no way special and the same behavior is exhibited by functions created using an ordinary `def`.)

Below are a few examples of ways to circumvent this issue.

One solution would be use a [Python generator](#) as follows:

```
def multipliers():
    for i in range(4): yield lambda x: i * x
```

Another solution is to create a closure that binds immediately to its arguments by using a default argument. For example:

```
def multipliers():
    return [lambda x, i=i : i * x for i in range(4)]
```

Or alternatively, you can use the `functools.partial` function:

```
from functools import partial
from operator import mul
```

```
def multipliers():
    return [partial(mul, i) for i in range(4)]
```

Finally, the easiest fix may be to simply replace the return value's `[]` with `()`:

```
def multipliers():
    return (lambda x : i * x for i in range(4))
```

3.

What will be the output of the code below? Explain your answer.

```
class Parent(object):
    x = 1
```

```
class Child1(Parent):
```

```
 pass
```

```
class Child2( Parent):  
 pass
```

```
print Parent.x, Child1.x, Child2.x
```

```
Child1.x = 2
```

```
print Parent.x, Child1.x, Child2.x
```

```
Parent.x = 3
```

```
print Parent.x, Child1.x, Child2.x
```

Hide answer

The output of the above code will be:

1	1	1
1	2	1
3	2	3

1. Can you differentiate between a List and a Tuple?

Lists and tuples are Python data structures. The list is dynamic and whereas the tuple has static characteristics. They both have various advantages and use cases.

List

The list is the mutable data type, consumes more memory, and it is better for element insertion and deletion. Furthermore, it has several building functions, and the implication of iterations is slower compared to Tuple.

Example:

```
a_list = ["Data", "Camp", "Tutorial"]
```

Tuple

The Tuple is an immutable data type, and it is generally used for accessing the elements. It is faster and consumes less memory, but it lacks built-in methods.

Example:

```
a_tuple = ("Data", "Camp", "Tutorial")
```

2. What is `__init__()` in Python?

It is known as a constructor in OOP terminology. It is used to initiate a state when you create a new object. For example, you can assign values to object properties or run the operations that are necessary when the object is created.

The `__init__()` method is reserved for Python classes, and it is called automatically when you create a new object.

Example:

We have created a `book_shop` class and added the constructor and `book()` function. The constructor will store the book title name and the `book()` function will print the book name.

To test our code we have initialized the `b` object with “Sandman” and executed the `book()` function.

```
class book_shop:  
  
    # constructor  
  
    def __init__(self, title):  
        self.title = title  
  
    # Sample method  
  
    def book(self):  
        print('The tile of the book is', self.title)
```

```
b = book_shop('Sandman')  
b.book()
```

3. What is the difference between a mutable data type and an immutable data type?

The mutable Python data types can be modified, and they can change at runtime, for example, a List, Dictionary, and Set.

The immutable Python data types can not be changed or modified, and they remain unchanged during runtime, for example, a Numeric, String, and Tuple.

4. Explain List, Dictionary, and Tuple comprehension with an example.

List

List comprehension offers one-liner syntax to create a new list based on the values of the existing list. You can use a `for loop` to replicate the same thing, but it will require you to write multiple lines, and sometimes it can get complex.

List comprehension eases the creation of the list based on existing iterable.

```
my_list = [i for i in range(1, 10)]  
  
my_list  
  
# [1, 2, 3, 4, 5, 6, 7, 8, 9]
```

Dictionary

Similar to a List comprehension, you can create a dictionary based on an existing table with a single line of code. You need to enclose the operation with curly brackets `{}`.

```
my_dict = {i for i in range(1, 10)}  
  
my_dict  
  
# {1, 2, 3, 4, 5, 6, 7, 8, 9}
```

Tuple

It is a bit different for Tuples. You can create Tuple comprehension using round brackets `()`, but it will return a generator object, not a tuple comprehension.

You can run the loop to extract the elements or convert them to a list.

```
my_tuple = (i for i in range(1, 10))

my_tuple

# <generator object <genexpr> at 0x7fb91b151430>
```

5. What is monkey patching in Python?

Monkey patching in Python is a dynamic technique that can change the behavior of the code at run-time. In short, you can modify a class or module at run-time.

Example:

Let's learn monkey patching with an example.

1. We have created a class `monkey` with a `patch()` function. We have also created a `monk_p` function outside the class.
2. We will now replace the `patch` with the `monk_p` function by assigning `monkey.patch` to `monk_p`.
3. In the end, we will test the modification by creating the object using the `monkey` class and running the `patch()` function.

Instead of displaying “patch() is being called”, it has displayed “monk_p() is being called”.

```
class monkey:

    def patch(self):

        print ("patch() is being called")



def monk_p(self):
```

```
print ("monk_p() is being called")  
  
# replacing address of "patch" with "monk_p"  
  
monkey.patch = monk_p  
  
  
  
obj = monkey()  
  
  
  
obj.patch()  
  
# monk_p() is being called
```

6. What is the Python “with” statement designed for?

The `with` statement is used for exception handling to make code cleaner and simpler. It is generally used for the management of common resources like creating, editing, and saving a file.

Example:

Instead of writing multiple lines of open, try, finally, and close, you can create and write a text file using the `with` statement. It is simple.

```
# using with statement  
  
with open('myfile.txt', 'w') as file:  
  
    file.write('DataCamp Black Friday Sale!!!')
```

7. Why use else in try/except construct in Python?

`try:` and `except:` are commonly known for exceptional handling in Python, so where does `else:` come in handy? `else:` will be triggered when no exception is raised.

Example:

Let's learn more about `else:` with a couple of examples.

1. On the first try, we entered 2 as the numerator and “d” as the denominator. Which is incorrect, and `except:` was triggered with “Invalid input!”.
2. On the second try, we entered 2 as the numerator and 1 as the denominator and got the result 2. No exception was raised, so it triggered the `else:` printing the message “Division is successful.”

```
try:
```

```
    num1 = int(input('Enter Numerator: '))

    num2 = int(input('Enter Denominator: '))

    division = num1/num2

    print(f'Result is: {division}'")
```

```
except:
```

```
    print('Invalid input!')
```

```
else:
```

```
    print('Division is successful.')
```

```
## Try 1 ##
```

```
# Enter Numerator: 2
```

```
# Enter Denominator: d
```

```
# Invalid input!
```

```
## Try 2 ##  
  
# Enter Numerator: 2  
  
# Enter Denominator: 1  
  
# Result is: 2.0  
  
# Division is successful.
```

8. What are the advantages of NumPy over regular Python lists?

Memory

Numpy arrays consume less memory.

For example, if you create a list and a Numpy array of a thousand elements. The list will consume 48K bytes, and the Numpy array will consume 8k bytes of memory.

Speed

Numpy arrays take less time to perform the operations on arrays than lists.

For example, if we are multiplying two lists and two Numpy arrays of 1 million elements together. It took 0.15 seconds for the list and 0.0059 seconds for the array to operate.

Vesititly

Numpy arrays are convenient to use as they offer simple array multiple, addition, and a lot more built-in functionality. Whereas Python lists are incapable of running basic operations.

9. What is the difference between merge, join and concatenate?

Merge

Merge two DataFrames named series objects using the unique column identifier.

It requires two DataFrame, a common column in both DataFrame, and “how” you want to join them together. You can left, right, outer, inner, and cross join two data DataFrames. By default, it is an inner join.

```
pd.merge(df1, df2, how='outer', on='Id')
```

Join

Join the DataFrames using the unique index. It requires an optional `on` argument that can be a column or multiple column names. By default, the join function performs a left join.

```
df1.join(df2)
```

Concatenate

Concatenate joins two or multiple DataFrames along a particular axis (rows or columns). It doesn't require an `on` argument.

```
pd.concat(df1, df2)
```

- **join()**: it combines two DataFrames by index.
- **merge()**: it combines two DataFrames by the column or columns you specify.
- **concat()**: it combines two or more DataFrames vertically or horizontally.

10. How do you identify and deal with missing values?

Identifying missing values

We can identify missing values in the DataFrame by using the `isnull()` function and then applying `sum()`. `Isnull()` will return boolean values, and the sum will give you the number of missing values in each column.

In the example, we have created a dictionary of lists and converted it into a pandas DataFrame. After that, we used isnull().sum() to get the number of missing values in each column.

```
import pandas as pd
```

```
import numpy as np
```

```
# dictionary of lists
```

```
dict = {'id':[1, 4, np.nan, 9],  
       'Age': [30, 45, np.nan, np.nan],  
       'Score':[np.nan, 140, 180, 198]}
```

```
# creating a DataFrame
```

```
df = pd.DataFrame(dict)
```

```
df.isnull().sum()
```

```
# id    1
```

```
# Age   2
```

```
# Score 1
```

Dealing with missing values

There are various ways of dealing with missing values.

1. Drop the entire row or the columns if it consists of missing values using `dropna()`. This method is not recommended, as you will lose important information.
2. Fill the missing values with the constant, average, backward fill, and forward fill using the `fillna()` function.
3. Replace missing values with a constant String, Integer, or Float using the `replace()` function.
4. Fill in the missing values using an interpolation method.

Note: make sure you are working with a larger dataset while using the `dropna()` function.

```
# drop missing values
```

```
df.dropna(axis = 0, how ='any')
```

```
#fillna  
df.fillna(method ='bfill')  
  
#replace null values with -999  
df.replace(to_replace = np.nan, value = -999)  
  
# Interpolate  
df.interpolate(method ='linear', limit_direction ='forward')
```

	<code>id</code> float64	<code>Age</code> float64	<code>Score</code> float64
0	1.0	30.0	nan
1	4.0	45.0	140.0
2	6.5	45.0	180.0
3	9.0	45.0	198.0

11. Which all Python libraries have you used for visualization?

Data visualization is the most important part of data analysis. You get to see your data in action, and it helps you find hidden patterns.

The most popular Python data visualization libraries are:

1. Matplotlib
2. Seaborn
3. Plotly

4. Bokeh

In Python, we generally use **Matplotlib** and **seaborn** to display all types of data visualization. With a few lines of code, you can use it to display scatter plot, line plot, box plot, bar chart, and many more.

For interactive and more complex applications, we use **Plotly**. You can use it to create colorful interactive graphs with a few lines of code. You can zoom, apply animation, and even add control functions. Plotly provides more than 40 unique types of charts, and we can even use them to create a web application or dashboard.

Bokeh is used for detailed graphics with a high level of interactivity across large datasets.

12. How can you replace string space with a given character in Python?

It is a simple string manipulation challenge. You have to replace the space with a specific character.

Example 1: a user has provided the string “I vey u” and the character “o”, and the output will be “loveyou”.

Example 2: a user has provided the string “D t C mpBl ckFrid yS le” and the character “a”, and the output will be “DataCampBlackFridaySale”.

In the `str_replace()` function, we will loop over each letter of the string and check if it is space or not. If it consists of space, we will replace it with the specific character provided by the user. Finally, we will be returning the modified string.

```
def str_replace(text,ch):  
  
    result = ""  
  
    for i in text:  
  
        if i == ' ':  
  
            i = ch  
  
        result += i  
  
    return result
```

```
text = "D t C mpBl ckFrid yS le"
```

```
ch = "a"
```

```
str_replace(text,ch)
```

```
# 'DataCampBlackFridaySale'
```

13. Given a positive integer num, write a function that returns True if num is a perfect square else False.

This has a relatively straightforward solution. You can check if the number has a perfect square root by:

1. Finding the square root of the number and converting it into an integer.
2. Applying the square to the square root number and checking if it's a perfect square root.
3. Returning the result as a boolean.

Test 1

We have provided number 10 to the `valid_square()` function.

1. By taking the square root of the number, we get 3.1622776601683795.
2. By converting it into an integer, we get 3.
3. Then, take the square of 3 and get 9.
4. 9 is not equal to the number, so the function will return False.

a

Test 2

We have provided number 36 to the `valid_square()` function.

1. By taking the square root of the number, we get 6.
2. By converting it into an integer, we get 6.

3. Then, take the square of 6 and get 36.
4. 36 is equal to the number, so the function will return True.

```
def valid_square(num):  
  
    square = int(num**0.5)  
  
    check = square**2==num  
  
    return check  
  
  
valid_square(10)  
  
# False  
  
valid_square(36)  
  
# True
```

POWERED BY DATACAMP WORKSPACE [COPY CODE](#)

14. Given an integer n, return the number of trailing zeroes in n factorial n!

To pass this challenge, you have to first calculate n factorial ($n!$) and then calculate the number of trailing zeros.

Finding factorial

In the first step, we will use a while loop to iterate over the n factorial and stop when the n is equal to 1.

Calculating trailing zeros

In the second step, we will calculate the trailing zero, not the total number of zeros. There is a huge difference.

$7! = 5040$

The seven factorials have a total of two zeros and only one trailing zero, so our solution should return 1.

1. Convert the factorial number to a string.
2. Read it back and apply for a loop.
3. If the number is 0, add +1 to the result, otherwise break the loop.
4. Returns the result.

The solution is elegant but requires attention to detail.

```
def factorial_trailing_zeros(n):
```

```
    fact = n
```

```
    while n > 1:
```

```
        fact *= n - 1
```

```
        n -= 1
```

```
    result = 0
```

```
    for i in str(fact)[::-1]:
```

```
        if i == "0":
```

```
            result += 1
```

```
        else:
```

```
            break
```

```
    return result
```

```
factorial_trailing_zeros(10)
```

```
# 2
```

```
factorial_trailing_zeros(18)
```

```
# 3
```

15. String segmentation

You are provided with a large string and a dictionary of the words. You have to find if the input string can be segmented into words using the dictionary or not.

Dictionary of words

data

cam

camp

lack

Input string "datacamp" can be segmented

data

camp

Input string "datafang" cannot be segmented

data

fang

The solution is reasonably straightforward. You have to segment a large string at each point and check if the string can be segmented to the words in the dictionary.

1. Run the loop using the length of the large string.
2. We will create two substrings.

3. The first substring will check each point in the large string from $s[0:i]$
4. If the first substring is not in the dictionary, it will return False.
5. If the first substring is in the dictionary, it will create the second substring using $s[i:0]$.
6. If the second substring is in the dictionary or the second substring is of zero length, then return True. Recursively call `can_segment_str()` with the second substring and return True if it can be segmented.

```
def can_segment_str(s, dictionary):  
  
    for i in range(1, len(s) + 1):  
  
        first_str = s[0:i]  
  
        if first_str in dictionary:  
  
            second_str = s[i:]  
  
            if (  
  
                not second_str  
  
                or second_str in dictionary  
  
                or can_segment_str(second_str, dictionary)  
  
            ):  
  
                return True  
  
    return False  
  
  
s = "datacamp"  
  
dictionary = ["data", "camp", "cam", "lack"]
```

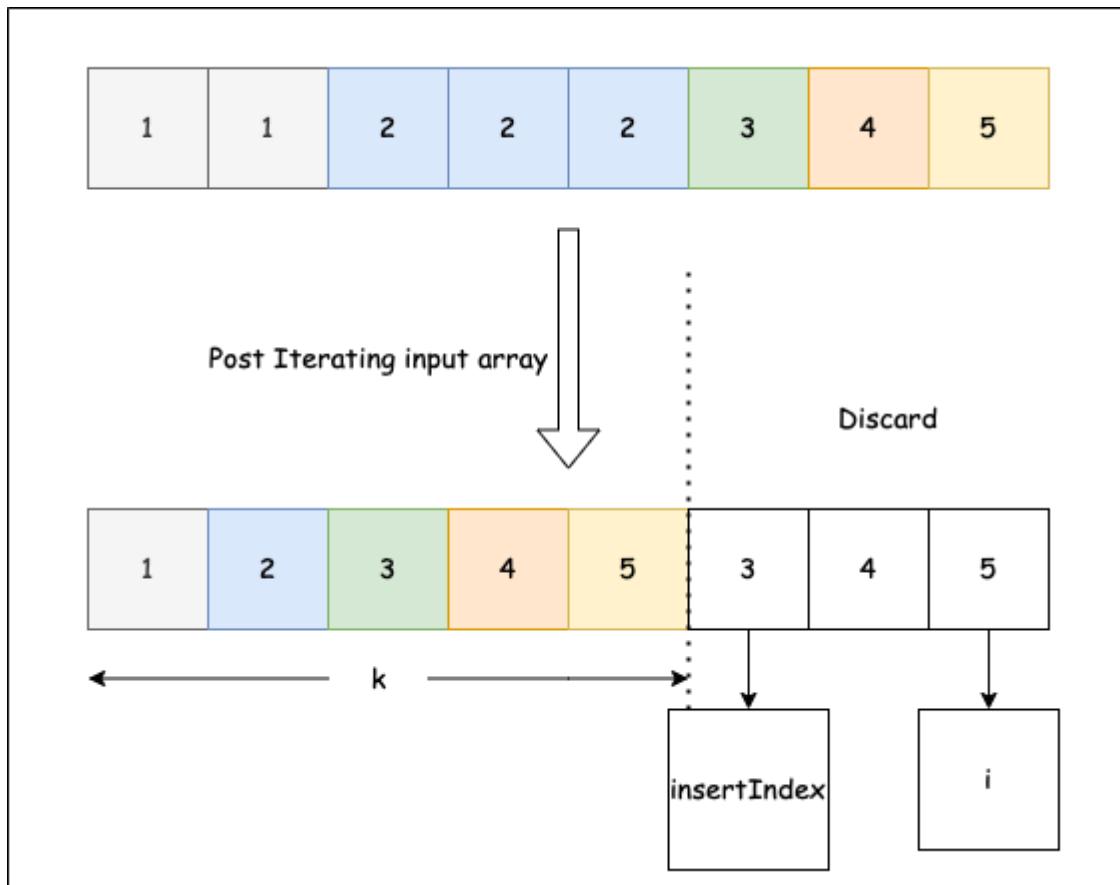
```
can_segment_string(s, dictionary)
```

```
# True
```

16. Remove duplicates from sorted array

Given an integer sorted array in increasing order, remove the duplicate numbers such that each unique element appears only once. Make sure you keep the final order of the array the same.

It is impossible to change the length of the array in Python, so we will place the result in the first part of the array. After removing duplicates, we will have k elements, and the first k elements in the array should hold the results.



Example 1: input array is [1,1,2,2], the function should return 2.

Example 2: input array is [1,1,2,3,3], the function should return 3.

Solution:

1. Run the loop for the range of 1 to the size of the array.

2. Check if the previous number is unique or not. We are comparing previous elements with the current one.
3. If it is unique, update the array using insertIndex, which is 1 at the start, and add +1 to the insertIndex.
4. Return insertIndex as it is the k.

This question is relatively straightforward once you know how. If you put more time into understanding the statement, you can easily come up with a solution.

```
def removeDuplicates(array):  
  
    size = len(array)  
  
    insertIndex = 1  
  
    for i in range(1, size):  
  
        if array[i - 1] != array[i]:  
  
            # Updating insertIndex in our main array  
  
            array[insertIndex] = array[i]  
  
            # Incrementing insertIndex count by 1  
  
            insertIndex = insertIndex + 1  
  
    return insertIndex  
  
  
  
array_1 = [1,2,2,3,3,4]  
  
removeDuplicates(array_1)  
  
# 4
```

```
array_2 = [1,1,3,4,5,6,6]
```

```
removeDuplicates(array_2)
```

```
# 5
```

17. Find the maximum single sell profit

You are provided with the list of stock prices, and you have to return the buy and sell price to make the highest profit.

Note: We have to make maximum profit from a single buy/sell, and if we can't make a profit, we have to reduce our losses.

Example 1: stock_price = [8, 4, 12, 9, 20, 1], buy = 4, and sell = 20.
Maximizing the profit.

Example 2: stock_price = [8, 6, 5, 4, 3, 2, 1], buy = 6, and sell = 5. Minimizing the loss.

Solution:

1. We will calculate the global profit by subtracting global sell (the first element in the list) from current buy (the second element in the list).
2. Run the loop for the range of 1 to the length of the list.
3. Within the loop, calculate the current profit using list elements and current buy value.
4. If the current profit is greater than the global profit, change the global profit with the current profit and global sell to the i element of the list.
5. If the current buy is greater than the current element of the list, change the current buy with the current element of the list.
6. In the end, we will return global buy and sell value. To get global buy value, we will subtract global sell from global profit.

The question is a bit tricky, and you can come up with your unique algorithm to solve the problems.

```
def buy_sell_stock_prices(stock_prices):
```

```
    current_buy = stock_prices[0]
```

```
global_sell = stock_prices[1]

global_profit = global_sell - current_buy

for i in range(1, len(stock_prices)):

    current_profit = stock_prices[i] - current_buy

    if current_profit > global_profit:

        global_profit = current_profit

        global_sell = stock_prices[i]

    if current_buy > stock_prices[i]:

        current_buy = stock_prices[i]

return global_sell - global_profit, global_sell
```

```
stock_prices_1 = [10,9,16,17,19,23]
```

```
buy_sell_stock_prices(stock_prices_1)
```

```
# (9, 23)
```

```
stock_prices_2 = [8, 6, 5, 4, 3, 2, 1]
```

```
buy_sell_stock_prices(stock_prices_2)
```

```
# (6, 5)
```

Amazon Python interview questions

Amazon Python interview questions can vary greatly but could include:

18. Find the missing number in the array

You have been provided with the list of positive integers from 1 to n. All the numbers from 1 to n are present except x, and you must find x.

Example:

4 5 3 2 8 1 6

- n = 8
- missing number = 7

This question is a simple math problem.

1. Find the sum of all elements in the list.
2. By using arithmetic series sum formula, we will find the expected sum of the first n numbers.
3. Return the difference between the expected sum and the sum of the elements.

```
def find_missing(input_list):
```

```
    sum_of_elements = sum(input_list)
```

```
    # There is exactly 1 number missing
```

```
    n = len(input_list) + 1
```

```

actual_sum = (n * ( n + 1 )) / 2

return int(actual_sum - sum_of_elements)

list_1 = [1,5,6,3,4]

find_missing(list_1)

# 2

```

19. Pythagorean Triplet in an array

Write a function that returns True if there is a Pythagorean triplet that satisfies $a^2 + b^2 = c^2$

Example:

Input	Output
[3, 1, 4, 6, 5]	True
[10, 4, 6, 12, 5]	False

Solution:

1. Square all the elements in the array.
2. Sort the array in increasing order.
3. Run two loops. The outer loop starts from the last index of the array to 1 and the inner loop starts from (outer_loop_index -1) to the start.
4. Create set() to store the elements between outer loop index and inner loop index.

5. Check if there is a number present in the set which is equal to ($\text{array}[\text{outerLoopIndex}] - \text{array}[\text{innerLoopIndex}]$). If yes, return True, else False.

```
def checkTriplet(array):
```

```
    n = len(array)
```

```
    for i in range(n):
```

```
        array[i] = array[i]**2
```

```
    array.sort()
```

```
    for i in range(n - 1, 1, -1):
```

```
        s = set()
```

```
        for j in range(i - 1, -1, -1):
```

```
            if (array[i] - array[j]) in s:
```

```
                return True
```

```
                s.add(array[j])
```

```
    return False
```

```
arr = [3, 2, 4, 6, 5]
```

```
checkTriplet(arr)
```

```
# True
```

20. How many ways can you make change with coins and a total amount?

We need to create a function that takes a list of coin denominations and total amounts and returns the number of ways we can make the change.

In the example, we have provided coin denominations [1, 2, 5] and the total amount of 5. In return, we got five ways we can make the change.

Denomination	1,2,5
Amount	5
No. of combination	
1, 1, 1, 1, 1	
1, 1, 1, 2	
1, 2, 2	
5	
Total No. of ways	5

Solution:

1. We will create the list of size amount + 1. Additional spaces are added to store the solution for a zero amount.
2. We will initiate a solution list with 1.
3. We will run two loops. The outer loop will return the number of denominations, and the inner loop will run from the range of the outer loop index to the amount +1.
4. The results of different denominations are stored in the array solution.
$$\text{solution}[i] = \text{solution}[i] + \text{solution}[i - \text{den}]$$
5. The process will be repeated for all the elements in the denomination list, and at the last element of the solution list, we will have our number.

```
def solve_coin_change(denominations, amount):
```

```
solution = [0] * (amount + 1)

solution[0] = 1

for den in denominations:

    for i in range(den, amount + 1):

        solution[i] += solution[i - den]

return solution[len(solution) - 1]
```

denominations = [1,2,5]

amount = 5

solve_coin_change(denominations,amount)

4

21. Define a lambda function, an iterator, and a generator in Python.

The Lambda function is also known as an anonymous function. You can add any number of parameters but with only one statement.

An iterator is an object that we can use to iterate over iterable objects like lists, dictionaries, tuples, and sets.

The generator is a function similar to a normal function, but it generates a value using the yield keyword instead of return. If the function body contains yield, it automatically becomes a generator.

22. Given an array arr[], find the maximum $j - i$ such that $arr[j] > arr[i]$

This question is quite straightforward but requires special attention to detail. We are provided with an array of positive integers. We have to find the maximum difference between $j - i$ where $array[j] > array[i]$.

Examples:

1. Input: [20, 70, 40, 50, 12, 38, 98], Output: 6 ($j = 6, i = 0$)
2. Input: [10, 3, 2, 4, 5, 6, 7, 8, 18, 0], Output: 8 ($j = 8, i = 0$)

Solution:

1. Calculate the length of the array and initiate max difference with -1.
2. Run two loops. The outer loop picks elements from the left, and the inner loop compares the picked elements with elements starting from the right side.
3. Stop the inner loop when the element is greater than the picked element and keep updating the maximum difference using $j - i$.

```
def max_index_diff(array):  
  
    n = len(array)  
  
    max_diff = -1  
  
    for i in range(0, n):  
  
        j = n - 1  
  
        while(j > i):  
  
            if array[j] > array[i] and max_diff < (j - i):  
  
                max_diff = j - i  
  
            j -= 1  
  
    return max_diff
```

```
array_1 = [20,70,40,50,12,38,98]
```

```
max_index_diff(array_1)
```

```
# 6
```

POWERED BY DATACAMP WORKSPACE [COPY CODE](#)

23. How would you use the ternary operators in Python?

Ternary operators are also known as conditional expressions. They are operators that evaluate expression based on conditions being True and False.

You can write conditional expressions in a single line instead of writing using multiple lines of if-else statements. It allows you to write clean and compact code.

For example, we can convert nested if-else statements into one line, as shown below.

If-else statement

```
score = 75
```

```
if score < 70:
```

```
    if score < 50:
```

```
        print('Fail')
```

```
    else:
```

```
        print('Merit')
```

```
else:
```

```
print('Distinction')  
# Distinction
```

POWERED BY DATACAMP WORKSPACE[COPY CODE](#)

Nested Ternary Operator

```
print('Fail' if score < 50 else 'Merit' if score < 70 else 'Distinction')  
# Distinction
```

What is Lambda in Python?

A lambda function is also known as an anonymous function, a Python function without a name. It can take any number of arguments but evaluates and returns only one expression.

Syntax:

```
lambda arguments : expression
```

Example:

```
lambda_add = lambda a : a + 10  
print(lambda_add (7))  
# 17
```

Is map faster than for loop?

“map” is faster than “for loop” if you are applying the function to every item of an iterable.

1— What is the difference between a list and a tuple? When should you use each?

A list is a mutable data structure while a tuple is an immutable one.

A mutable object in Python has the ability to change its values.

Lists are dynamic: you can add items to them or override and remove existing ones.

Tuples are fixed-size: they don't have an `append` or an `extend` method. You cannot remove items from them either.

Both tuples and lists support indexing and allow using the `in` operator to check for existing elements in them.

→ **There are some situations where I think tuples might be useful.**

- If you declare a collection of items that you know will never change or that you will loop over only without changing its values, use tuples.
- If you look for performance, tuples are faster than lists since they're read-only structures. If you don't need write operations, consider using tuples.
- Tuples can make your code safer if you want to prevent accidentally writing data that doesn't need to be changed.

Here's a code sample that shows how tuples differ from lists.

2 — What is the difference between multiprocessing and multithreading? When should you use each?

Multiprocessing and Multithreading are programming paradigms that aim to speed up your code.

When you use multiprocessing, you parallelize your computation over processes. Processes are independent and don't communicate with each other: they don't share the same memory area and have strict isolation between. In terms of applications, multiprocessing is suited for CPU-intensive workloads. It does,

however, have a large memory footprint that is proportional to the number of processes.

On the other hand, in multithreaded applications, threads live inside a single process. Consequently, they share the same memory area: they can modify the same variables and can interfere with one another. While processes are strictly executed in parallel, only one thread is executed at a given point in time in Python, and this is due to the Global Interpreter Lock ([GIL](#)). Multithreading is suited to IO-bound applications such as web scraping or fetching data from a database.

→ If you want to learn more about multithreading and multiprocessing, I recommend you go through this amazing blog [post](#) that draws a comprehensive picture of the two concepts.

3 — What is the difference between a module, a package, and a library?

A module is simply a Python file that's intended to be imported into scripts or other modules. It contains functions, classes, and global variables.

A package is a collection of modules that are grouped together inside a folder to provide consistent functionality. Packages can be imported just like modules. They usually have an `__init__.py` file in them that tells the Python interpreter to process them as such.

A library is a collection of packages.

4 — What is the problem with multi-threading in python?

The Global Interpreter Lock (or GIL) prevents the python interpreter from executing more than one thread at the same time. Put simply, the GIL forces that only one thread is executed at any point in time in Python.

This represents a big performance bottleneck in CPU-bound applications that rely on multithreaded code.

5 — What are decorators? Can you describe a situation in which decorators are worth using?

A decorator is a function that receives a function as input and returns a function as output. The goal of a decorator is to extend the behavior of the input function without changing its core mechanism.

Using a decorator also prevents you from repeating yourself. It forces you to write a generic code once and then tap it to every function that needs it.

A typical use-case where decorators shine is **logging**.

Imagine, for example, that you want to log to the terminal all the values of the parameters that are passed to every function that is called in your program. You can go through every function definition and write that down or you can just write one single decorator that does this logging task and apply it to all the functions that need it.

Applying a decorator to a function is only a matter of adding a single line above that function's definition.

```
# without decorator def my_awesome_function():
    # do awesome stuff # with a decorator @my_awesome_decorator
def my_awesome_function():
    # do even more awesome stuff
```

Here's a code sample that creates a decorator called `log` that logs the values of the parameters that are passed to a function.

Decorators can also be used for other purposes such as timing functions, validating input data, enforcing access control and authentication, caching, etc.

, caching, etc.

6 — How to properly write data to a file? What can go wrong otherwise?

Using a context manager is key.

When you use the `open` statement without a context manager and some exception occurs before you close the file (closing the file is something you must remember when opening a file this way) memory issues could happen and the file might be corrupted along the way.

When you use `with` to open a file and an exception occurs, Python guarantees that the file is closed.

7 — Are function arguments passed by reference or by value?

All function arguments are passed by reference in Python: this means that if you pass a parameter to a function, the function gets a reference to that same object.

If the object is mutable and the function changes it, the parameter will mutate in the outer scope of the function. Let's see an example:

8 — How to override the way objects are printed?

Use the `__str__` and the `__repr__` dunder methods.

Here's an example that demonstrates how an instance from the Person class can be nicely formatted when printed to the console.

9 — Write a function that computes the factorial of an integer n

Recursivity is key

10 — What is the difference between the `is` and `==` operators?

`==` is an operator that tests the equality while `is` is an operator that tests for identity.

Two objects can have equal values without necessarily being identical (i.e. having the same memory address).

Remember that `a is b` is syntactic sugar for `id(a) == id(b)`.

11 — When shouldn't you use the `assert` statement?

The `assert` statement is useful for internal testing and sanity checks.

However, it shouldn't be used to perform data validation or error handling because it's generally disabled in production code for performance reasons.

Imagine if you check for admin privileges using assert: this can introduce a big security leak in production.

Instead of using the assert statement, you can throw a custom error.

12 — What is a Python generator?

A Python generator is a function that produces a sequence of items.

Generators look like typical functions but their behavior is different. For starters, instead of using the return statement, they use the yield statement.

Then, calling the generator function doesn't run the function: it only creates a generator object. The code of the generator only executes when the next function is applied to the generator object or if the generator is iterated over (in this case, the next function is implicitly called)

The number of times the next function is called on the generator object is equal to the number of times the yield statement is invoked in the generator function.

You can define generators using for-loops or generator expressions.

13 — What is the difference between a class method and a static method? When should you use which?

A static method is a method that knows anything about the class or the instance that had called it. It's a method that logically belongs to the class but doesn't have implicit arguments.

A static method can be either called on the class or any of its instances.

A class method is a method that gets passed the class it was called on, much like `self` is passed to other instance methods in a class. **The mandatory argument of a class method isn't a class instance: it's actually the class itself.**

A typical use-case of class methods is providing an alternative way to construct instances: a class method that does this is known as a *factory of the class*.

Here's an Employee class that uses a class method that creates an instance in a slightly different way than the main constructor of the class.

14— Give an example of how you use zip and enumerate

The `zip` function takes multiple iterables as input and aggregates them in a tuple. For example, this can be useful if you want to loop over two lists at the same time.

The `enumerate` function allows to loop over an iterable and access both the running index and the item at the same time.

15 — How would you use *args and **kwargs in a given function?

*args and **kwargs make Python functions more flexible by accepting a variable number of arguments.

- *args pass a variable number of non-keyworded arguments in a list
- **kwargs pass a variable number of keyword arguments in a dictionary

Here's an example of a function that takes a variable number of keyworded arguments that are collected in a dictionary called `data` (note that it doesn't need to be named `kwargs`)

```
● ● ●

def show_user_info(**data):
    # data is a dict
    for key, value in data.items():
        print(f"{key}: {value}")

>>> show_user_info(first_name="John", last_name="Doe")
first_name: John
last_name: Doe

>>> show_user_info(first_name="John", last_name="Doe", job="Data scientist")
first_name: John
last_name: Doe
job: Data scientist
```

16 — Give an example of functional programming using map

17 — what is the difference between the continue and break statements

The `break` statement terminates the loop that contains it. The program immediately moves to the code section that is in the outer scope of the loop.

```
for item in sequence:  
    # code in the inner scope of the loop  
  
    if condition:  
        break  
  
    # code in the outer scope of the loop
```



```
while input_condition:  
    # code in the inner scope of the loop  
  
    if condition:  
        break  
  
    # code in the outer scope of the loop
```

On the other hand, the `continue` statement skips the rest of the code of the current iteration and move to the next iteration.

```
for item in sequence:  
    # code in the inner scope of the loop  
  
    if condition:  
        continue
```

```
    # code in the inner scope of the loop
```

```
while input_condition:  
    # code in the inner scope of the loop  
  
    if condition:  
        continue
```

```
    # code in the inner scope of the loop
```

18 — How to prevent a function from being called an unnecessary amount of time?

Use caching.

If the output that is associated with a given input doesn't change over a period of time, using caching would make sense for the function.

A typical scenario would be querying a web server: if you query a URL the first time and you know that response won't change, you can cache the result.

19 — Give some PEP8 guidelines

- Use 4 spaces per indentation level.
- Imports should be grouped in the following order:
 1. Standard library imports.
 2. Related third-party imports.
 3. Local application/library-specific imports.
- Function and variable names should be lowercase and separated by underscores
- Class names use the CapWords convention.

20 — How to read an 8GB file in Python with a computer that has 2GB of RAM?

This solution works for any large (and even larger) files.

When you open the file, all you need to do is use the file object as an iterator: while looping over this file object, you'll be fetching one line at a time and the previous lines will be cleared from memory (i.e. they are garbage collected).

This way, the file will never be entirely loaded in memory and your processing will be done on the go.

```
with open("./large_dataset.txt") as input_file:
```

```
    for line in input_file:
```

```
        process_line(line)
```

1) What is Python? What are the benefits of using Python?

Python is a programming language with objects, modules, threads, exceptions, and automatic memory management. The benefits of python are that it is simple and easy, portable, extensible, build-in data structure, and it is open-source.

2) What is PEP 8?

PEP 8 is a coding convention, a set of recommendation, about how to write your Python code more readable.

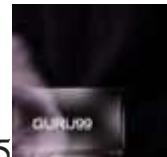


EXPLORE MORE

Learn Java

Programming with Beginners Tutorial08:32

Linux Tutorial for



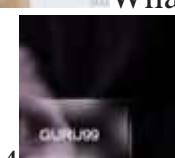
Beginners: Introduction to Linux Operating...01:35

What is



Integration Testing Software Testing Tutorial03:04

What is JVM



(Java Virtual Machine) with Architecture JAVA...02:24

How to write



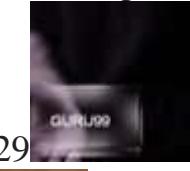
a TEST CASE Software Testing Tutorial01:08

Seven Testing

Principles Software Testing05:01

Linux File Permissions Commands

with Examples13:29



How to use Text tool in Photoshop CC

Tutorial08:32



What is NoSQL Database Tutorial02:00

Important Linux Commands for Beginners Linux Tutorial15:03

3) What is pickling and unpickling?

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.

4) How is Python interpreted?

Python language is an interpreted language. Python program runs directly from the source code. It converts the source code that is written by the programmer into an intermediate language, which is again translated into machine language that has to be executed.

5) How is memory managed in Python?

Python memory is managed by Python private heap space. All Python objects and data structures are located in a private heap. The programmer does not have an access to this private heap, and the interpreter takes care of this Python private heap.

The allocation of Python heap space for Python objects is done by the Python memory manager. The core API gives access to some tools for the programmer to code.

Python also has an inbuilt garbage collector, which recycles all the unused memory and frees the memory and makes it available to the heap space.

6) What are the tools that help to find bugs or perform the static analysis?

PyChecker is a static analysis tool that detects the bugs in Python source code and warns about the style and complexity of the bug. Pylint is another tool that verifies whether the module meets the coding standard.

7) What are Python decorators?

A Python decorator is a specific change that we make in Python syntax to alter functions easily.

8) What is the difference between list and tuple?

The difference between list and tuple is that list is mutable while tuple is not. Tuple can be hashed, for example., as a key for dictionaries.

9) How are arguments passed by value or by reference?

Everything in Python is an object, and all variables hold references to the objects. The reference values are according to the functions. Therefore, you cannot change the value of the references. However, you can change the objects if it is mutable.

10) What is Dict and List comprehensions are?

They are syntax constructions to ease the creation of a Dictionary or List based on existing iterable.

11) What are built-in type does python provides?

Python provides two built-in types: 1) Mutable and 2) Immutable.

Mutable built-in types are:

- List
- Sets
- Dictionaries
- Immutable built-in types
- Strings
- Tuples
- Numbers

Immutable built-in types are:

- Strings
- Tuples
- Numbers

12) Explain namespace in Python

In Python, every name introduced has a place where it lives and can be hooked for. This is known as a namespace. It is like a box where a variable name is mapped to the object placed. Whenever the variable is searched out, this box will be searched to get the corresponding object.

13) What is lambda in Python?

It is a single expression anonymous function often used as inline function.

14) Why lambda forms in python do not have statements?

A lambda form in python does not have statements as it is used to make new function object and then return them at runtime.

15) Explain pass in Python

Pass means no-operation Python statement, or in other words, it is a place holder in a compound statement, where there should be a blank left, and nothing has to be written there.

16) In Python what are iterators?

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

17) What is the unittest in Python?

A unit testing framework in Python is known as unittest. It supports sharing of setups, automation testing, shutdown code for tests, aggregation of tests into collections, etc.

18) Explain slicing in Python?

A mechanism to select a range of items from sequence types like list, tuple, strings etc., is known as slicing.

19) What are generators in Python?

The way of implementing iterators are known as generators. It is a normal function except that it yields expression in the function.

20) What is docstring in Python?

A Python documentation string is known as docstring, it is a way of documenting Python functions, modules, and classes.

21) How can you copy an object in Python?

To copy an object in Python, you can try a `copy.copy()` or `copy.deepcopy()` for the general case. You cannot copy all objects but most of them.

22) What is negative index in Python?

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

23) How can you convert a number to a string?

In order to convert a number into a string, use the inbuilt function `str()`. If you want a octal or hexadecimal representation, use the inbuilt function `oct()` or `hex()`.

24) What is the difference between xrange and range?

Xrange returns the xrange object while range returns the list and uses the same memory and no matter what the range size is.

25) What is module and package in Python?

In Python, module is the way to structure a program. Each Python program file is a module, which imports other modules like objects and attributes.

The folder of Python program is a package of modules. A package can have modules or subfolders.

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

Here are the rules for local and global variables in Python:

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

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

27) How can you share global variables across modules?

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

28) Explain how can you make a Python Script executable on Unix?

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

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

29) Explain how to delete a file in Python?

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

30) Explain how can you generate random numbers in Python?

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

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

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

31) How can you access a module written in Python from C?

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

```
Module = PyImport_ImportModule("<modulename>");
```

32) What is the use of // operator in Python?

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

33) Mention five benefits of using Python

Here are the five benefits of using Python:

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

34) Mention the use of the split function in Python

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

35) Explain Flask and its benefits

Flask is a web micro framework for Python based on “Werkzeug, Jinja 2 and good intentions” BSD licensed. Werkzeug and jingja are two of its dependencies.

Flask is part of the micro-framework. Which means it will have little to no dependencies on external libraries. It makes the framework light while there is a little dependency to update and less security bugs.

36) What is the difference between Django, Pyramid, and Flask?

Flask is a “microframework” primarily build for a small application with simpler requirements. In a flask, you don’t have to use external libraries. Flask is ready to use.

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

37) What is Flask-WTF and what are their features?

Flask-WTF offers simple integration with WTForms. Features include for Flask WTF are:

- Integration with WTForms
- Secure form with CSRF token
- Global CSRF protection
- Internationalization integration
- Recaptcha supporting
- File upload that works with Flask Uploads

38) Explain what is the common way for the Flask script to work?

The common way for the flask script to work is:

- Either it should be the import path for your application
- Or the path to a Python file

39) Explain how you can access sessions in Flask?

A session basically allows you to remember information from one request to another. In a flask, it uses a signed cookie so the user can look at the session contents and modify. The user can modify the session if only it has the secret key Flask.secret_key.

40) Is Flask an MVC model, and if yes give an example showing MVC pattern for your application?

Basically, Flask is a minimalistic framework that behaves same as MVC framework. So MVC is a perfect fit for Flask, and the pattern for MVC we will consider for the following example

```
from flask import Flask  
app = Flask(__name__)  
  
@app.route("/")
```

Def hello():

```
    return "Hello World"  
  
app.run(debug = True)
```

In this code your Configuration part will be from flask import Flask
`app = Flask(__name__)`
View part will be
`@app.route("/")`

Def hello():

```
    return "Hello World"  
While you model or main part will be  
app.run(debug = True)
```

41) Explain database connection in Python Flask?

Flask supports database-powered applications (RDBS). Such a system requires creating a schema, which requires piping the shema.sql file into a sqlite3 command. So you need to install sqlite3 command in order to create or initiate the database in Flask.

Flask allows to request database in three ways

- **before_request()**: It is called before a request and pass no arguments
- **after_request()**: It is called after a request and pass the response that will be sent to the client
- **teardown_request()**: It is called in a situation when exception is raised, and response is not guaranteed. They are called after the response has been constructed. They are not allowed to modify the request, and their values are ignored.

42) If you have multiple Memcache servers, and one of them fails that contain data, will it try to get them?

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

43) Explain how you can minimize the Memcached server outages in your Python Development?

- When one instance fails, several of them goes down, this will put a larger load on the database server when lost data is reloaded as the client make a

request. To avoid this, if your code has been written to minimize cache stampedes, then it will leave a minimal impact

- Another way is to bring up an instance of memcached on a new machine using the lost machine's IP address
- Code is another option to minimize server outages as it gives you the liberty to change the Memcached server list with minimal work
- Setting timeout value is another option that some Memcached clients implement for Memcached server outage. When your Memcached server goes down, the client will keep trying to send a request till the time-out limit is reached.

44) Explain what is Dogpile effect? How can you prevent this effect?

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

45) Explain how memcached should not be used in your Python project?

Here are the ways you should not use memcached in your Python project:

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

46) What is Python If Statement?

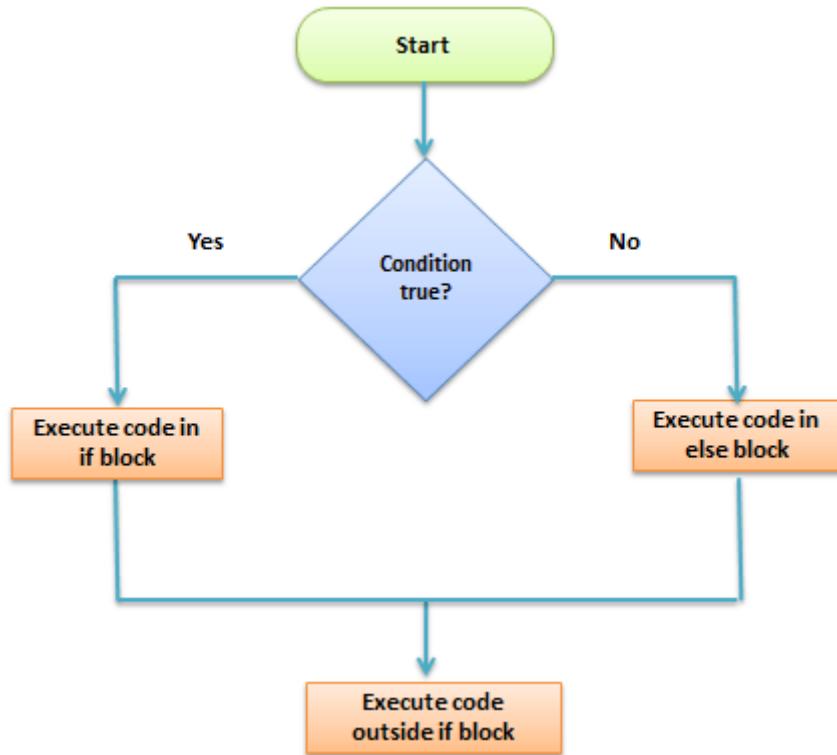
Python if Statement is used for decision-making operations. It contains a body of code that runs only when the condition given in the if statement is true. If the condition is false, then the optional else statement runs, which contains some code for the else condition.

When you want to justify one condition while the other condition is not true, then you use Python if-else statement.

Python if Statement Syntax:

```
if expression  
Statement  
else  
Statement
```

Python if...else Flowchart



Let's see an example of Python if else Statement:

```
1 #  
2 # Example file for working with conditional statement  
3 #  
4 def main():  
5     x, y = 2, 8  
6  
7     if (x < y):  
8         st = "x is less than y"  
9         print(st)  
10  
11 if __name__ == "__main__":  
12     main()  
13  
14
```

Run Python11.1

C:\Users\DK\Desktop>python c:\python\Python11.1.py
x is less than y

Let's see an example of Python if else Statement:

```
def main():  
    x,y=2,8  
  
    if(x < y):  
        st= "x is less than y"  
        print(st)  
  
if __name__ == "__main__":  
    main()
```

47) Explain While loop in Python with example

While loop does the exact same thing what “if statement” does, but instead of running the code block once, they jump back to the point where it began the code and repeat the whole process again.

The syntax of while loop is as follows:

while expression
Statement

The example of while loop is as follows:

```
x=0
#define a while loop
while(x <4):
    print(x)
    x = x+1
```

48) What is enumerate() in Python?

[Enumerate\(\) in Python](#) is a built-in function used for assigning an index to each item of the iterable object. It adds a loop on the iterable objects while keeping track of the current item and returns the object in an enumerable form. This object can be used in a for loop to convert it into a list by using list() method.

Example of enumerate() is as follows:

Suppose we want to do numbering for our month (Jan, Feb, Marc,June), so we declare the variable i that enumerate the numbers while m will print the number of month in list.

```
#use a for loop over a collection
Months = ["Jan","Feb","Mar","April","May","June"]
for i, m in enumerate (Months):
    print(i,m)
```

use the break and continue statements

```
#for x in range (10,20):
#    if (x == 15): break
#    if (x % 5 == 0) : continue
#    print x
```

49) How can you use for loop to repeat the same statement over and again?

You can use for loop for even repeating the same statement over and again. Here in the example, we have printed out the word “guru99” three times.

Example:

To repeat the same statement a number of times, we have declared the number in variable i (i in 123). So when you run the code as shown below, it prints the statement (guru99) that many times the number declared for our the variable in (i in 123).

```
for i in '123':  
    print ("guru99",i,)
```

50) What is Tuple Matching in Python?

Tuple Matching in Python is a method of grouping the tuples by matching the second element in the tuples. It is achieved by using a dictionary by checking the second element in each tuple in python programming. However, we can make new tuples by taking portions of existing tuples.

Syntax:

```
Tup = ('Jan','feb','march')
```

To write an empty tuple, you need to write as two parentheses containing nothing-

```
tup1 = ();
```

51) Explain Dictionary in Python with example

A Dictionary in Python is the unordered and changeable collection of data values that holds key-value pairs. Each key-value pair in the dictionary maps the key to its associated value making it more optimized. A Dictionary in python is declared by enclosing a comma-separated list of key-value pairs using curly braces({ }). Python Dictionary is classified into two elements: Keys and Values.

Syntax for Python Dictionary:

```
Dict = { 'Tim': 18, xyz,.. }
```

Example

```
Dict = { 'Tim': 18,'Charlie':12,'Tiffany':22,'Robert':25}  
print((Dict['Tiffany']))
```

52) How can you copy the entire dictionary to a new dictionary?

You can also copy the entire dictionary to a new dictionary. For example, here we have copied our original dictionary to the new dictionary name “Boys” and “Girls”.

Example

```
Dict = { 'Tim': 18,'Charlie':12,'Tiffany':22,'Robert':25}  
Boys = { 'Tim': 18,'Charlie':12,'Robert':25 }
```

```
Girls = {"Tiffany":22}
studentX=Boys.copy()
studentY=Girls.copy()
print(studentX)
print(studentY)
```

53) How can you Update Python Dictionary?

You can update a dictionary by adding a new entry or a key-value pair to an existing entry or by deleting an existing entry. Here in the example, we will add another name, “Sarah” to our existing dictionary.

Example

```
Dict = {"Tim": 18,'Charlie':12,'Tiffany':22,'Robert':25}
Dict.update({"Sarah":9})
print(Dict)
```

54) Give example of dictionary items() method

```
Dict = {"Tim": 18,'Charlie':12,'Tiffany':22,'Robert':25}
print("Students Name: % s" % list(Dict.items()))
```

55) How can you sort elements in Python dictionary?

In the dictionary, you can easily sort the elements. For example, if we want to print the name of the elements of our dictionary alphabetically, we have to use for loop. It will sort each element of the dictionary accordingly.

Example:

```
Dict = {"Tim": 18,'Charlie':12,'Tiffany':22,'Robert':25}
Boys = {"Tim": 18,'Charlie':12,'Robert':25}
Girls = {"Tiffany":22}
Students = list(Dict.keys())
Students.sort()
for S in Students:
    print(":.".join((S,str(Dict[S]))))
```

56) Give an example of Dictionary len() and Python List cmp() method

Dictionary len() Example:

```
Dict = {"Tim": 18,'Charlie':12,'Tiffany':22,'Robert':25}
print("Length : % d" % len (Dict))
```

cmp() Example:

```
Boys = { 'Tim': 18,'Charlie':12,'Robert':25}  
Girls = { "Tiffany":22}  
print cmp(Girls, Boys)
```

57) What are all dictionary methods:

Here is the list of dictionary methods:

- copy()
- update()
- items()
- sort()
- len()
- cmp()
- Str()

58) Explain Arithmetic operators with example

Arithmetic Operators perform various arithmetic calculations like addition, subtraction, multiplication, division, % modulus, exponent, etc. There are various methods for arithmetic calculation in Python, like you can use the eval function, declare variable & calculate, or call functions.

Example: For arithmetic operators, we will take a simple example of addition where we will add two-digit $4+5=9$

```
x= 4  
y= 5  
print(x + y)
```

59) Give example of logical operators

Example of logical operators:

```
a = True  
b = False  
print('a and b is',a and b))  
print('a or b is',a or b))  
print('not a is',not a))
```

60) Explain membership operators with example

These operators test for membership in a sequence such as lists, strings, or tuples. Two membership operators are used in Python. (in, not in). It gives the result based on the variable present in a specified sequence or string.

Example:

For example here, we check whether the value of x=4 and value of y=8 is available in list or not by using in and not in operators.

```
x = 4
y = 8
list = [1, 2, 3, 4, 5];
if ( x in list ):
    print("Line 1 - x is available in the given list")
else:
    print("Line 1 - x is not available in the given list")
if ( y not in list ):
    print("Line 2 - y is not available in the given list")
else:
    print("Line 2 - y is available in the given list")
```

61) Write code to demonstrate operator precedence in Python:

```
v = 4
w = 5
x = 8
y = 2
z = 0
z = (v+w) * x / y;
print("Value of (v+w) * x/ y is ", z)
```

62) Explain arrays in Pythons with example

A [Python Array](#) is a collection of a common type of data structures having elements with the same data type. It is used to store collections of data. In Python programming, arrays are handled by the “array” module. If you create arrays using the array module, elements of the array must be of the same numeric type.

Syntax to Create an Array in Python

You can declare an array in Python while initializing it using the following syntax.

```
arrayName = array.array(type code for data type, [array,items])
```

The following image explains the syntax.

Array Syntax

1. **Identifier:** specify a name like usually, you do for variables
2. **Module:** Python has a special module for creating array in Python, called “array” – you must import it before using it
3. **Method:** the array module has a method for initializing the array. It takes two arguments, type code, and elements.
4. **Type Code:** specify the data type using the type codes available (see list below)
5. **Elements:** specify the array elements within the square brackets, for example [130,450,103]

Example

```
import array as myarray  
abc = myarray.array('d', [2.5, 4.9, 6.7])
```

63) How can you access array elements?

You can access any array item by using its index.

The syntax is

arrayName[indexNum]

Example

```
import array  
balance = array.array('i', [300,200,100])  
print(balance[1])
```

64) How can you insert elements in array?

Python array insert operation enables you to insert one or more items into an array at the beginning, end, or any given index of the array. This method expects two arguments index and value.

The syntax is

```
arrayName.insert(index, value)
```

Example

Let us add a new value right after the second item of the array. Currently, our balance array has three items: 300, 200, and 100. Consider the second array item with a value of 200 and index 1.

In order to insert the new value right “after” index 1, you need to reference index 2 in your insert method, as shown in the below Python array example:

```
import array  
balance = array.array('i', [300,200,100])  
balance.insert(2, 150)  
print(balance)
```

65) How can you delete elements in array?

With this operation, you can delete one item from an array by value. This method accepts only one argument, value. After running this method, the array items are re-arranged, and indices are re-assigned.

The syntax is

```
arrayName.remove(value)
```

Example

Let’s remove the value of “3” from the array

```
import array as myarray
```

```
first = myarray.array('b', [2, 3, 4])
first.remove(3)
print(first)
```

66) How can you search and get the index of a value in an array?

With this operation, you can search for an item in an array based on its value. This method accepts only one argument, value. It is a non-destructive method, which means it does not affect the array values.

The syntax is

`arrayName.index(value)`

Example

Let's find the value of "3" in the array. This method returns the index of the searched value.

```
import array as myarray
number = myarray.array('b', [2, 3, 4, 5, 6])
print(number.index(3))
```

67) How can you reverse array in Python?

You can use `reverse()` to reverse array in Python.

Example:

```
import array as myarray
number = myarray.array('b', [1,2,3])
number.reverse()
print(number)
```

68) Give example to convert array to Unicode

The Example to convert array to Unicode is:

```
from array import array
p = array('u',[u'\u0050',u'\u0059',u'\u0054',u'\u0048',u"\u004F",u'\u004E'])
print(p)
q = p.tounicode()
print(q)
```

69) Give an example of a class in Python

Example of class in Python

```
# Example file for working with classes
class myClass():
    def method1(self):
        print("Guru99")

    def method2(self,someString):
        print("Software Testing:" + someString)

    def main():
        # exercise the class methods
        c = myClass ()
        c.method1()
        c.method2(" Testing is fun")

if __name__=="__main__":
    main()
```

70) Explain Inheritance with example

Inheritance is a feature used in [object-oriented programming](#); it refers to defining a new class with less or no modification to an existing class. The new class is called the derived class, and from one which it inherits is called the base. Python supports inheritance; it also supports multiple inheritances. A class can inherit attributes and behavior methods from another class called subclass or heir class.

Example of inheritance:

```
# Example file for working with classes
class myClass():
    def method1(self):
        print("Guru99")

class childClass(myClass):
    #def method1(self):
    #    #myClass.method1(self);
    #    #print ("childClass Method1")

    def method2(self):
        print("childClass method2")

def main():
```

```
# exercise the class methods
c2 = childClass()
c2.method1()
#c2.method2()
```

```
if __name__ == "__main__":
    main()
```

71) Give example of Python constructors

Example of Python Constructors

```
class User:
    name = ""

    def __init__(self, name):
        self.name = name

    def sayHello(self):
        print("Welcome to Guru99, " + self.name)

User1 = User("Alex")
User1.sayHello()
```

72) How can you access values in string?

Python does not support a character type, these are treated as strings of length one, also considered as a substring.

You can use square brackets for slicing along with the index or indices to obtain a substring.

```
var1 = "Guru99!"
var2 = "Software Testing"
print ("var1[0]:",var1[0])
print ("var2[1:5]:",var2[1:5])
```

73) Explain all string operators with example

String operators with example:

Operator	Description	Example
[]	Slice- it gives the letter from the given index	a[1] will give “u” from as such (0=G, 1=u, 2=t)

Operator	Description	Example
[:]	Range slice-it gives the characters from the given range	x [1:3] it will give “u
in	Membership-returns true if a letter exists in the given string	Guru. Remember it w
not in	Membership-returns true if a letter exists is not in the given string	which is G, it will co
r/R	Raw string suppresses the actual meaning of escape characters.	that is ur.
%r – Used for string format	%r – It insert the canonical string representation of the object (i.e., repr(o)) %s- It inserts the presentation string representation of the object (i.e., str(o)) %d- it will format a number for display	u is present in word G
% – Used for object (i.e., repr(o))	will give 1 (True)	l not present in word
*	It concatenates 2 strings	will give 1
*	Repeat	Print r'\n' prints \n and prints \n

74) Give example of sleep() function in Python

Example of sleep() function in Python

```
import time
print("Welcome to guru99 Python Tutorials")
time.sleep(5)
print("This message will be printed after a wait of 5 seconds")
```

75) What is timer method in Python?

Timer is a method available with Threading, and it helps to get the same functionality as Python time sleep.

```
from threading import Timer

print('Code Execution Started')

def display():
    print('Welcome to Guru99 Tutorials')

t = Timer(5, display)
t.start()
```

76) Give example of calendar class

Example of calendar class

```
import calendar
# Create a plain text calendar
c = calendar.TextCalendar(calendar.THURSDAY)
str = c.formatmonth(2025, 1, 0, 0)
print(str)

# Create an HTML formatted calendar
hc = calendar.HTMLCalendar(calendar.THURSDAY)
str = hc.formatmonth(2025, 1)
print(str)
# loop over the days of a month
# zeroes indicate that the day of the week is in a next month or overlapping
month
for i in c.itermonthdays(2025, 4):
    print(i)

# The calendar can give info based on local such as names of days and months
# (full and abbreviated forms)
for name in calendar.month_name:
    print(name)
for day in calendar.day_name:
    print(day)
# calculate days based on a rule: For instance an audit day on the second
Monday of every month
# Figure out what days that would be for each month, we can use the script as
shown here
for month in range(1, 13):
    # It retrieves a list of weeks that represent the month
    mycal = calendar.monthcalendar(2025, month)
    # The first MONDAY has to be within the first two weeks
    week1 = mycal[0]
    week2 = mycal[1]
    if week1[calendar.MONDAY] != 0:
        auditday = week1[calendar.MONDAY]
    else:
        # if the first MONDAY isn't in the first week, it must be in the second
        week
        auditday = week2[calendar.MONDAY]
    print("%10s %2d" % (calendar.month_name[month], auditday))
```

77) Explain Python ZIP file with example

Python allows you to quickly create zip/tar archives.

Following command will zip the entire directory

```
shutil.make_archive(output_filename, 'zip', dir_name)
```

Following command gives you control on the files you want to archive

```
ZipFile.write(filename)
```

Example of Python ZIP file

```
import os
import shutil
from zipfile import ZipFile
from os import path
from shutil import make_archive

# Check if file exists
if path.exists("guru99.txt"):
    # get the path to the file in the current directory
    src = path.realpath("guru99.txt");
    # rename the original file
    os.rename("career.guru99.txt","guru99.txt")
    # now put things into a ZIP archive
    root_dir,tail = path.split(src)
    shutil.make_archive("guru99 archive","zip",root_dir)
# more fine-grained control over ZIP files
with ZipFile("testguru99.zip", "w") as newzip:
    newzip.write("guru99.txt")
    newzip.write("guru99.txt.bak")
```

78) What are the common examples of exceptions in Python?

The common examples of exceptions in Python are:

- Division by Zero
- Accessing a file that does not exist.
- Addition of two incompatible types
- Trying to access a nonexistent index of a sequence
- Removing the table from the disconnected database server.
- ATM withdrawal of more than the available amount

79) Explain important Python errors

The important Python errors are 1) ArithmeticError, 2) ImportError, and 3) IndexError.

- **ArithmeticError:** ArithmeticError act as a base class for all arithmetic exceptions. It is raised for errors in arithmetic operations.
- **ImportError:** ImportError is raised when you are trying to import a module which does not present. This kind of exception occurs if you have made a typing mistake in the module name or the module which is not present in the standard path.
- **IndexError:** An IndexError is raised when you try to refer a sequence which is out of range.

80) Explain JSON dumps() in Python with example

json.dumps() in Python is a method that converts dictionary objects of Python into JSON string data format. It is useful when the objects are required to be in string format for the operations like parsing, printing, etc.

Example:

```
import json

x = {
    "name": "Ken",
    "age": 45,
    "married": True,
    "children": ("Alice", "Bob"),
    "pets": ['Dog'],
    "cars": [
        {"model": "Audi A1", "mpg": 15.1},
        {"model": "Zeeb Compass", "mpg": 18.1}
    ]
}
# sorting result in ascending order by keys:
sorted_string = json.dumps(x, indent=4, sort_keys=True)
print(sorted_string)
```

81) Explain in detail JSON to Python (Decoding) with example

JSON string decoding is done with the help of inbuilt method **json.loads()** & **json.load()** of JSON library in Python.

Here translation table show example of JSON objects to Python objects which are helpful to perform decoding in Python of JSON string.

JSON	Python
Object	Dict
Array	List
String	Unicode
number – int	Number – int, long
number – real	Float
True	True
False	False
Null	None

The basic JSON to Python example of decoding with the help of `json.loads` function:

```
import json # json library imported
# json data string
person_data = '{ "person": { "name": "Kenn", "sex": "male", "age": 28}}'
# Decoding or converting JSON format in dictionary using loads()
dict_obj = json.loads(person_data)
print(dict_obj)
# check type of dict_obj
print("Type of dict_obj", type(dict_obj))
# get human object details
print("Person.....", dict_obj.get('person'))
```

82) Write code for encode() method

Code for encode() method:

```
# import JSONEncoder class from json
from json.encoder import JSONEncoder
colour_dict = { "colour": ["red", "yellow", "green"] }
# directly called encode method of JSON
JSONEncoder().encode(colour_dict)
```

83) Write a Python code for array in numpy to create Python Matrix

Code for array in numpy to create Python Matrix

```
import numpy as np
M1 = np.array([[5, -10, 15], [3, -6, 9], [-4, 8, 12]])
print(M1)
```

84) Write a Phyhon code for matrix subtraction

Phyhon code for matrix subtraction

```
import numpy as np  
M1 = np.array([[3, 6, 9], [5, -10, 15], [-7, 14, 21]])  
M2 = np.array([[9, -18, 27], [11, 22, 33], [13, -26, 39]])  
M3 = M1 - M2  
print(M3)
```

85) Write code for matrix multiplication

Code for matrix multiplication

```
import numpy as np
```

```
M1 = np.array([[3, 6], [5, -10]])  
M2 = np.array([[9, -18], [11, 22]])  
M3 = M1.dot(M2)  
print(M3)
```

86) Explain slicing of matrix with example

Slicing will return you the elements from the matrix based on the start /end index given.

The syntax for slicing is:

[start:end]

- If the start index is not given, it is considered as 0. For example [:5], it means as [0:5].
- If the end is not passed, it will take as the length of the array.
- If the start/end has negative values, it will the slicing will be done from the end of the array.

Before we work on slicing on a matrix, let us first understand how to apply slice on a simple array.

```
import numpy as np
```

```
arr = np.array([2,4,6,8,10,12,14,16])  
print(arr[3:6]) # will print the elements from 3 to 5  
print(arr[:5]) # will print the elements from 0 to 4
```

```
print(arr[2:]) # will print the elements from 2 to length of the array.  
print(arr[-5:-1]) # will print from the end i.e. -5 to -2  
print(arr[:-1]) # will print from end i.e. 0 to -2
```

87) Write Python code to find average via loop

Here is a code to find average via loop:

```
def cal_average(num):  
    sum_num = 0  
    for t in num:  
        sum_num = sum_num + t  
  
    avg = sum_num / len(num)  
    return avg  
  
print("The average is", cal_average([18,25,3,41,5]))
```

88) Write a code for list count

Here, is a Code for list count:

```
list1 = ['red', 'green', 'blue', 'orange', 'green', 'gray', 'green']  
color_count = list1.count('green')  
print('The count of color: green is ', color_count)
```

89) How can you count duplicate elements in a given list?

count duplicate elements in a given list

```
list1 = [2,3,4,3,10,3,5,6,3]  
elm_count = list1.count(3)  
print('The count of element: 3 is ', elm_count)
```

90) Write a code to get index of an element in a list using for loop

Code to get index of an element in a list using for loop:

```
my_list = ['Guru', 'Siya', 'Tiya', 'Guru', 'Daksh', 'Riya', 'Guru']  
all_indexes = []  
for i in range(0, len(my_list)) :  
    if my_list[i] == 'Guru' :  
        all_indexes.append(i)  
print("Originallist ", my_list)  
print("Indexes for element Guru : ", all_indexes)
```

91) Give an example of Python type()

```
str_list = "Welcome to Guru99"  
age = 50  
pi = 3.14  
c_num = 3j+10  
my_list = ["A", "B", "C", "D"]  
my_tuple = ("A", "B", "C", "D")  
my_dict = {"A": "a", "B": "b", "C": "c", "D": "d"}  
my_set = {'A', 'B', 'C', 'D'}  
  
print("The type is : ", type(str_list))  
print("The type is : ", type(age))  
print("The type is : ", type(pi))  
print("The type is : ", type(c_num))  
print("The type is : ", type(my_list))  
print("The type is : ", type(my_tuple))  
print("The type is : ", type(my_dict))  
print("The type is : ", type(my_set))
```

92) How can you print without a newline in Python?

From Python 3+, there is an additional parameter introduced for `print()` called `end=`. This parameter takes care of removing the newline that is added by default in `print()`.

In the Python 3 print without newline example below, we want the strings to print on the same line in Python. To get that working, just add `end=""` inside `print()` as shown in the example below:

```
print("Hello World ", end="")  
print("Welcome to Guru99 Tutorials")
```

93) How to print the star(*) pattern without newline and space?

Code to print the star(*) pattern without newline and space:

```
for i in range(0, 20):  
    print(*, end="")
```

1. What kind of language is Python?

Python is a general-purpose programming language created by Guido Van Rossum in 1991.

The official implementation of Python (CPython), is an **interpreted language** but other implementations allow you to compile your Python program as well.

Python is a **strongly-typed language** (since the interpreter keeps track of all the variables and their type) and a **dynamically-typed language** (since the type of the variables is checked at runtime by the Python interpreter).

It supports many programming paradigms and in particular procedural and object-oriented programming.

2. What does the acronym PEP stand for?

PEP stands for **Python Enhancement Proposal**. A PEP is basically a document that can be used to propose new features and enhancements to the Python maintainers or to describe a new feature or a process of Python to the community.

PEPs are an invaluable resource if you want to stay up to date with new implementations of Python or if you want to know the rationale behind a specific Python implementation.

3. What is PEP-8?

PEP-8 is probably the most well-known PEP of Python and it's a PEP that every good Python developer should know. PEP-8 describes the **style guidelines for Python code**. It includes recommendations about Python programming, suggested naming conventions, and how you should lay-out your code.

It goes without saying that your code could have custom-defined guidelines, but the ones suggested by the PEP-8 are a set of standard guidelines that are widely accepted by the community.

4. What is the PYTHONPATH environment variable?

The PYTHONPATH is an environment variable you can use if you need **to add other directories where the interpreter will look for modules**.

To set this variable, the format to be used is the same as the shell's PATH environment variable.

5. What's the difference between a module and a package in Python?

A module is just a single Python file you can import into other Python files to reuse code like functions, variables, objects, etc.

A package is a collection of modules stored in the same directory that allows you to better organize your code in a hierarchy of modules.

To create a package is really trivial. You just need to create a directory with a name of your choice, put inside this directory all the Python files you want to be considered as modules of this package, and then add a file named `__init__.py`.

6. What's the difference between a local and a global variable in Python?

The difference between a global and a local variable in Python is in their scope, which is the visibility they have from different parts of your program.

A local variable is a variable that **can be accessed only inside the function it is defined into**.

A global variable is declared outside a function and so **can be accessed globally**, either inside or outside the function it is defined into.

7. What are iterators in Python and how can you make an object iterable?

An iterator is **an object that allows you to traverse a container object**, iterating all the values it contains. If a container object can be iterated, it is said to be **iterable**.

In Python, there are lots of iterable objects, for example, **lists** and **strings**.

Iterating a list lets you iterate over every single value contained in a list while iterating a string lets you iterate over every char of the string, like in the following example:

```
input_string = "Hello Pythonista!"  
  
for char in input_string:  
    print(char)
```

However, you can easily build your custom iterable object by yourself just by implementing the **iterator protocol**. This protocol consists of two methods:

```
def __iter__(self):
```

```
def __next__(self):
```

In the `.__iter__()` method you have to return the object that can be iterated and in the `.__next__()` method you have to return each time a different value of your collection.

For a practical use case, look at this example that implements an iterable object that can return all the *Fibonacci sequence*. That is a sequence that starts with the 0 and 1 numbers and where each other number of the sequence is found by adding up the two numbers before it.

```
class fibonacci:
```

```
    def __init__(self, max=1000000):
```

```
        self.a, self.b = 0, 1
```

```
        self.max = max
```

```
    def __iter__(self):
```

```
        # Return the iterable object (self)
```

```
        return self
```

```
    def __next__(self):
```

```
        # When we need to stop the iteration we just need to raise
```

```
        # a StopIteration exception
```

```
        if self.a > self.max:
```

```
            raise StopIteration
```

```
        # save the value that has to be returned
```

```
        value_to_be_returned = self.a
```

```
        # calculate the next values of the sequence
```

```
        self.a, self.b = self.b, self.a + self.b
```

```
    return value_to_be_returned

my_fibonacci_iterable = fibonacci(1000)

for i in my_fibonacci_iterable:
    print(i)
```

As you can see, to create this iterable *Fibonacci* object in the `__iter__()` method, we have just returned the object itself, and in the `__next__()` method we have calculated the number to be returned at each iteration.

8. What is Python's string slicing feature?

String slicing is a feature of Python that **allows you to index and access portions of a string easily**. In Python, a string is an iterable object, which means you can index and access single characters by using the **slicing** feature as you would do for any other iterable object.

To slice a string, you just need to take the string and specify the start index and the end index of your slice in square brackets, separated by a colon.

For example, if you need to print the first three characters of a string, you can slice it as follows:

```
# Slice the input string from the beginning
input_string = "Hello Pythonista!"

first_three_characters = input_string[:3]
print(f"The first three characters are '{first_three_characters}'")
```

As you can see, if you omit the start index, the Python interpreter considers it like it was **zero**.

In the next example, you will slice the same string starting from the “P” character till the end of the string.

```
# Slice the input string from the seventh characters to the end
```

```
input_string = "Hello Pythonista!"  
  
substring_to_the_end = input_string[6:]  
print(f"A substring of the input string is '{substring_to_the_end}'")
```

Note that in this example, we have omitted the end index. If you omit the end index, it is automatically considered as the last index of your iterable object (in this case, the string).

Moreover, you can also use negative indexes if you want to indicate a position starting from the end of the string, like in this example:

```
# Slice the input string from the end  
input_string = "Hello Pythonista!"  
  
last_three_characters = input_string[-3:]  
print(f"The last three characters are '{last_three_characters}'")
```

And finally, you can obviously indicate both the start and the end index, for example, if you need to take a substring that is in the middle of your input string:

```
# Slice the input string in the middle  
input_string = "Hello Pythonista!"  
  
a_substring = input_string[6:12]  
print(f"A substring of the input string is '{a_substring}'")
```

Slicing is a really convenient way to manipulate your strings in Python and it works with every iterable object.

9. Tell me about Python's list comprehension feature and give an example of how to use it.

List comprehension is a Python feature that **allows you to create lists in a really concise way**.

Let's consider the following example:

```
squares = []
for i in range(1000):
    squares.append(i*i)
```

In this example, we are creating a list that contains the square of all the integers from 0 to 999. But there is a better and quicker way to achieve the same result by using list comprehension:

```
squares = [i*i for i in range(1000)]
```

That's it, just one line of code.

The basic syntax for a list comprehension is:

```
[expression for item in iterable]
```

If you need to specify a specific condition to be met to execute the **expression** you can use the **if** keyword and specify it at the end of the list comprehension like this:

```
[expression for item in iterable if condition]
```

For example, if you needed the first 1000 squares only for the odd numbers, you could write your list comprehension like this:

```
odd_squares = [i*i for i in range(1000) if i%2 != 0]
```

Interested in web development?

Learn all about it in our comprehensive (and free) ebook!

ENTER YOUR EMAIL ADDRESS

10. Is it possible to document your Python code using Python built-in features?

Yes! To document your **Python code** **you can simply use docstrings**. **Docstrings** are string literals that can be put inside your code just after the definition of a class, a method, or a function — or at the beginning of a

module. They can help you to create documentation, since they are accessible from the `__doc__` attribute of the objects and by using the standard `help()` function.

Docstrings can also be accessed by several Python tools like **Sphinx** and **Pdoc** to automatically generate the Python documentation of your code.

11. What are lambda functions?

Lambda functions are **short anonymous functions** that can have no more than one line of code.

To declare a lambda function, you just need to specify the keyword **lambda**, the input parameters you will get (if any), and then a colon to separate the expression that has to be evaluated and returned when the lambda is called.

Let's consider the following example:

```
>>> square = lambda x : x*x
```

In this example, we are defining an anonymous lambda function that takes a parameter (`x`) and returns to the caller its square ($x*x$). In this example, we have also assigned the function to a name (`square`) and so we can call this function like this:

```
>>> square(9)  
81
```

However, to assign a lambda to a name is often not necessary. For example, lambdas are often written **inline** and passed as a parameter to another function (like the `map()` or the `filter()` functions).

For example, let's say that we have a list of numbers and we want to create another list with their squares. In this case, a lambda passed to a `map()` function can solve our need with just one line of code:

```
>>> my_input_numbers = [1, 2, 3, 4, 5, 10, 100, 1000]  
>>> my_squares = map(lambda x : x*x, my_input_numbers)
```

The same result could be achieved also by using list comprehension like this:

```
>>> my_squares_list = [x*x for x in my_input_numbers]
```

So another question the interviewer could ask is: what's the difference between these two methods and what's the best one? The answer here is that while list comprehension creates a list, the map function simply returns a **map object** that is a Python **iterable** and that is **lazy**. This means that the evaluation of the lambda is done only when the object is actually iterated.

To get the idea, try to change the example above pretending that *my_input_numbers* was huge:

```
>>> my_input_numbers = range(10**100)
```

Now, execute both the map and the list comprehension example, and you will see that the map example keeps working and returns a map object that can be iterated, while the list comprehension one hangs consuming more and more memory.

12. How are Dictionaries implemented in Python?

Python dictionaries are implemented on top of another great technology — **hash tables**. A hash table is basically a collection of **key-value pairs** that are stored in a memory area that is divided into a specific number of buckets.

They are called **hash** tables because the key of each pair is hashed and this hash is used to calculate which bucket to assign to a single pair.

If two or more keys produce the same hash or if the algorithm used to assign a hash to a bucket put two different hash in a single bucket, you have a collision of hash or a collision of buckets. Collisions are often unavoidable, and for this reason, Python implements the **open addressing** collision resolution method.

Hash tables make Python dictionaries really reliable and fast because with hash tables the number of operations used to look up a single value in memory doesn't change, even if the table grows thousands of times.

If you want to learn more about Python hash tables you can refer to [this article](#).

13. Describe the difference between a shallow copy and a deep copy

First of all, let's say that this difference is only relevant when we're talking about compound objects or objects that contain other objects, like classes for example.

In the **shallow copy**, the process creates a new compound object and fills it up with **references** to the objects contained in the original object.

In the **deep copy**, the process creates a new compound object and fills it up with **real copies** of the objects contained in the original object.

This means that the contained objects are shared between the copies when you are dealing with shallow copies while they are not shared when you are dealing with deep copies.

In both cases, you can use the module *copy* that is included in the standard library and:

- to make a shallow copy use the method *.copy()* of the *copy* module
- to make a deep copy use the method *.deepcopy()* of the *copy* module.

Here's a practical example:

```
import copy

original_list = [1, [2, 3], 4, 5]
shallow_copied_list = copy.copy(original_list)
deep_copied_list = copy.deepcopy(original_list)

original_list[1][0] = 999

print(original_list)
print(shallow_copied_list)
print(deep_copied_list)
```

As you can see, in the shallow copied list the objects 2 and 3 are shared with the ones from the original list, so changing one of them in the original list has an effect also to the copied one. In the deep copied list instead, they are different objects and so any modification to the original list doesn't affect the deep copied one.

It goes without saying that a deep copy is slower to be created and has a bigger memory footprint than a shallow one.

14. Describe generators in Python and give a brief example of how to use them

A generator in Python **is an easy way of creating an iterable object** without having to implement the whole iteration protocol. It is just a simple function that contains a yield statement instead of the return one.

When the interpreter finds a yield statement, it returns a value like it would with a return one but instead of terminating the functions it puts it in pause, keeping the values and the state of all the variables and objects contained.

For example, let's say that we want to create a function to get the Fibonacci sequence like in the former example.

With a generator, we just need to create a function with a loop that calculates these numbers and after each number is found returns it by using the yield keyword:

```
def fibonacci():
    a, b = 0, 1
    while True:
        yield a
        a, b = b, a+b

if __name__ == '__main__':
    # Create a generator of fibonacci numbers smaller than 1 million
    fibonacci_generator = fibonacci()

    # print out all the sequence until CTRL-C is pressed
    try:
        for fibonacci_number in fibonacci_generator:
            print(fibonacci_number)
    except KeyboardInterrupt:
        print("terminated")
```

As you can see, creating the sequence with this method is really trivial since you just need to create a function that calculates all the sequence and returns the numbers to the caller with the yield keyword.

15. What is the main difference between a tuple and a list in Python?

Lists and tuples can appear very similar to beginners since both are compound objects that contain a sequence of elements and both can contain elements of different data types. However, **while lists are mutable objects, tuples are immutable ones** and this is the main difference between these two types.

Due to this fact, lists are usually slower and have a larger memory footprint but are far more flexible, while tuples are faster and lighter, being less flexible and immutable.

16. How can you serialize an object in Python?

Python has several built-in modules that can be used to serialize and deserialize objects, but the two most important ones are *pickle* and *json*.

To serialize in json format is the best choice if you need to have as a result a serialized object that is **human-readable** and **language-independent**. In other situations, you should probably choose the pickle format.

Both the *json* and the *pickle* module contain:

- A *.dump()* method to serialize objects writing an output file
- A *.dumps()* method to serialize objects in a simple string
- A *.load()* method to deserialize objects from an input file
- A *.loads()* method to deserialize object from an input string.

17. Are serialization and deserialization using the Python Pickle module safe operations?

No. The pickle module is a great way to serialize and deserialize Python's objects, but due to its powerful features, it is also really dangerous. In fact, when you serialize an object, you can use the *__setstate__()* method to insert additional initialization instructions that have to be executed upon the deserialization of the object itself.

That means that this code is executed when you **unpickle** the serialized object at runtime, so it's extremely important that you never unpickle objects that are coming from an untrusted source since you never know what it can contain.

18. Write a Python program to execute a binary search on a list

During the interview, you will probably be asked to write some code to check your skills. One of the most requested is to write some code to execute a binary search on a simple list.

A binary search is a search algorithm that allows you to find the position of an item inside a sorted list.

To give an example, here you find a possible solution to achieve this goal:

```
# remember that the input list HAS TO BE SORTED!
def bsearch(item, my_list):
    while True:
        # if the list is empty, return False
        if len(my_list) == 0:
            return False

        # look for the middle value
        mid = int(len(my_list)/2)
        element = my_list[mid]

        # if the middle value was the element we
        # were searching for, ok, we found it!
        if item == element:
            return True

        # if not, just remove the part of the list
        # that does not contain our element and retry
        if item > element:
            my_list = my_list[mid+1:]
        else:
            my_list = my_list[:mid]
```

This example could be rewritten also by using recursion, like in the example below:

```
# remember that the input list HAS TO BE SORTED!
def bsearch(item, my_list):

    # if the list is empty, return False
    if len(my_list) == 0:
        return False

    # look for the middle value
    mid = int(len(my_list)/2)
    element = my_list[mid]

    # if the middle value was the element we
    # were searching for, ok, we found it!
    if item == element:
        return True

    # if not, just remove the part of the list
    # that does not contain our element
    if item > element:
        return(bsearch(item, my_list[mid+1:]))
    else:
        return(bsearch(item, my_list[:mid]))
```

However, Python doesn't implement **tail call optimization** out of the box, and for this reason, in Python using recursion is often not a good idea.

19. How do you debug a Python program?

It's said that Python comes with **batteries included** because in its standard library you have hundreds of useful modules to do a wide range of different operations. One of the modules you could find is called *pdb* and is a powerful debugger.

Pdb supports a lot of powerful features like conditional breakpoints, evaluation of arbitrary Python code, inspection of stack frames, etc.

To use *pdb* starting from Python 3.7 you just need to use the built-in *breakpoint()* function. With this function, you break the execution of your code and bring up the debugger command line.

When the debugger command line shows up, you can use several commands like:

- *l* to list the program
- *s* to step in
- *n* to continue until the next line
- *c* to continue the execution
- *j* to jump to another part of the program
- *p* to evaluate expressions
- *q* to quit

There are several other commands you can use, but the ones above are probably the most used ones.

20. What are virtual environments, and how can you create a virtual environment by using the standard Python library?

Virtual environments are a great Python feature that allows you to create separate environments for running different Python code on the same machine.

In practice, a virtual environment is just a directory tree that contains a Python installation and some other additional packages.

Think of a typical case. You have program X that depends on a couple of third-party modules — module A in version 1.0 and module B in version 1.3.

Then, on the same machine, you also want to use the program Y that depends on module C in version 4.0 and module B in version 2.0.

Due to the fact that both the programs use module B but of a different major release, it's very probable that version 2.0 won't run properly for program X and version 1.3 won't run for program Y. So how can you deal with this?

The answer here is to use virtual environments. By using virtual environments, you could create two different environments, one for program X and one for program Y. Then, in each virtual environment, you will be able to install the version of the libraries you need.

The support for virtual environments is built-in with the Python standard library — into the *venv* module.

To create an environment, you just need to use:

```
python -m venv programx
```

Here, **programx** is the name of the virtual environment you will create. This command will create a **programx** directory in the current path with all you need. Now, to activate an environment under MS Windows you just need to use:

```
programx\Scripts\activate.bat
```

If you need to activate an environment under Linux or macOS you can use:

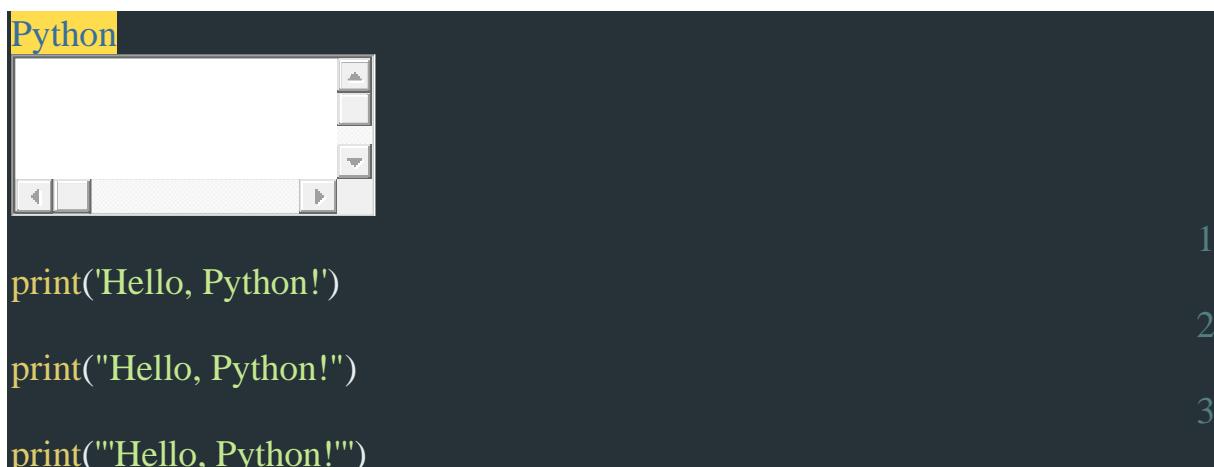
```
source programx/bin/activate
```

After the activation of an environment, you will be able to use *pip* to install all the packages you need, and they will be visible just inside this virtual environment.

However, module *venv* is not the only one you can use to create different environments. For example, another great tool is *conda* that is a part of the Anaconda Python distribution, and that allows you also to have different Python interpreters for each virtual environment you create.

Anaconda Python is also really popular among data scientists and if you need to do data analysis with Python.

1) Write the output for the below Hello, Python commands.



The screenshot shows a terminal window titled "Python". It contains the following code and its execution results:

```
print('Hello, Python!')
print("Hello, Python!")
print("Hello, Python!")
```

The output is displayed in three numbered columns:

- 1: Hello, Python!
- 2: Hello, Python!
- 3: Hello, Python!"

```
4  
print("Hello, Python")  
5  
print("Hello, Python")
```

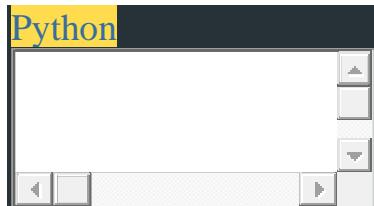
Output

```
Hello, Python!  
Hello, Python!  
Hello, Python!  
“Hello, Python!”  
‘Hello, Python!’
```

Explanation: In Python programming, a string can be enclosed inside single quotes, double quotes, or triple quotes.

2) What is the output of the Python add two numbers program?

Python Add Two Numbers Program



```
1  
#Python Add Two Numbers Program  
2  
number1 = 5  
3  
number2 = 4  
4  
# Add two numbers  
5  
sum = number1 + number2  
6  
# Display the sum of the two numbers  
7  
print('The sum of the two numbers is:' sum)
```

Output of the program:

```
File “”, line 7  
print('The sum of the two numbers is:' sum)
```

^

SyntaxError: invalid syntax

The above Python add two numbers program throws SyntaxError because, the comma(,) is missing in print statement. The below print statement will give you the sum of the two numbers.

```
Python
print('The sum of the two numbers is:', sum)
1
```

3) What is the output of the below sum of the two numbers Python program?

```
Python
num1 = input('Enter first number: ')
1
num2 = input('Enter second number: ')
2
sum = num1 + num2
3
print('The sum of the numbers is', sum)
4
```

The output of the sum of the two numbers program

Enter first number: 15

Enter second number: 10

The sum of the numbers is 1510

Python input() function always converts the user input into a string. Whether you enter an int or float value it will consider it as a string. We need to convert it into number using int() or float() functions. See the below example.

Read the input numbers from users

```
num1 = input('Enter the first number: ')
num2 = input('Enter the second number: ')
```

Converting and adding two numbers using int() & float() functions

```
sum = int(num1) + int(num2)
sum2 = float(num1) + float(num2)
```

Displaying the sum of two numbers

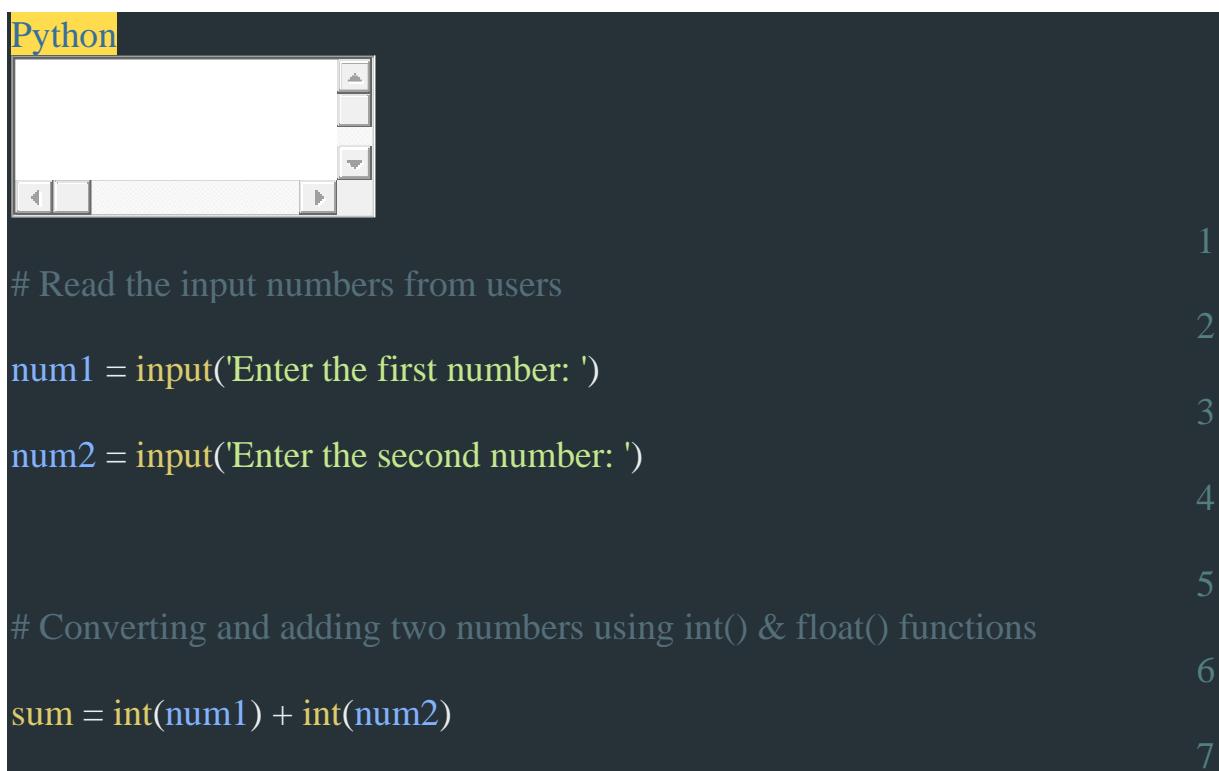
```
print('The sum of {0} and {1} is {2}'.format(num1, num2, sum))
print('The sum of {0} and {1} is {2}'.format(num1, num2, sum2))
```

The output of the sum of the two numbers program

```
Enter the first number: 15
Enter the second number: 10
The sum of 15 and 10 is 25
The sum of 15 and 10 is 25.0
```

4) Write a Python program to illustrate arithmetic operations (+,-,*,/)?

Here is the Python arithmetic operators program:



```
Python
# Read the input numbers from users
num1 = input('Enter the first number: ')
num2 = input('Enter the second number: ')
# Converting and adding two numbers using int() & float() functions
sum = int(num1) + int(num2)
```

```
8  
# Subtracting the two numbers  
9  
sub = int(num1) - int(num2)  
10  
11  
# Multiplying two numbers  
12  
mul = float(num1) * float(num2)  
13  
14  
#Dividing two numbers  
15  
div = float(num1) / float(num2)  
16  
17  
18  
# Displaying the results of arithmetic operations  
19  
print('The sum of {0} and {1} is {2}'.format(num1, num2, sum))  
20  
print('The subtraction of {0} and {1} is {2}'.format(num1, num2, sub))  
21  
print('The multiplication of {0} and {1} is {2}'.format(num1, num2, mul))  
22  
print('The division of {0} and {1} is {2}'.format(num1, num2, div))  
23
```

The output of the Python arithmetic operators program

Enter the first number: 25
Enter the second number: 10
The sum of 25 and 10 is 35
The subtraction of 25 and 10 is 15
The multiplication of 25 and 10 is 250.0
The division of 25 and 10 is 2.5

5) Python Program to Check if a Number is Odd or Even.

```
Python
1 #Python program to check if a number is odd or even
2
3 #To get the input from user
4 num1 = input("Enter a number: ")
5
6 #Checking whether the entered number is odd or even
7 if (int(num1) % 2) == 0:
8     print("{0} is Even number".format(num1))
9 else:
10    print("{0} is Odd number".format(num1))
11
```

The output of the Python odd or even program

Enter a number: 4

4 is Even number

Enter a number: 5

5 is Odd number

6) Write a Python program to find the maximum of two numbers?

Here is the Python program to find the maximum of two numbers:

```
Python
```

```
1 #To read the input value from the user
2 num1 = input('Enter the first number: ')
3
4 num2 = input('Enter the second number: ')
5
6 #Finding the maximum value using Python max() funtion
7 maximum = max(int(num1), int(num2))
8
9 #Displaying the maximum number
10 print("The maximum number is: ",maximum)
```

The output of the Python program to find the maximum number

Enter the first number: 23
Enter the second number: 45
The maximum number is: 45

7) Write a Python program to check prime numbers.

Here is the Python program to check whether a number is prime or not:

```
1 # Function to check prime number
2 def PrimeChecking(num):
```

```
3  
# Condition to check given number is more than 1  
4  
if num > 1:  
5  
    # For look to iterate over the given number  
6  
    for i in range(2, int(num/2) + 1):  
7  
        # Condition to check if the given number is divisible  
8  
        if (num % i) == 0:  
9  
            #If divisible by any number it's not a prime number  
10  
            print("The number ",num, "is not a prime number")  
11  
            break  
12  
        # Else print it as a prime number  
13  
    else:  
14  
        print("The number ",num, "is a prime number")  
15  
    # If the given number is 1  
16  
else:  
17  
    print("The number ",num, "is not a prime number")  
18  
# Input function to take the number from user  
19  
num = int(input("Enter a number to check prime or not: "))  
20  
# To print the result, whether a given number is prime or not  
21  
PrimeChecking(num)
```

#Output1 of the above Python program to check the prime number

Enter a number to check prime or not: 10

The number 10 is not a prime number

#Output2 of the above Python program to check the prime number

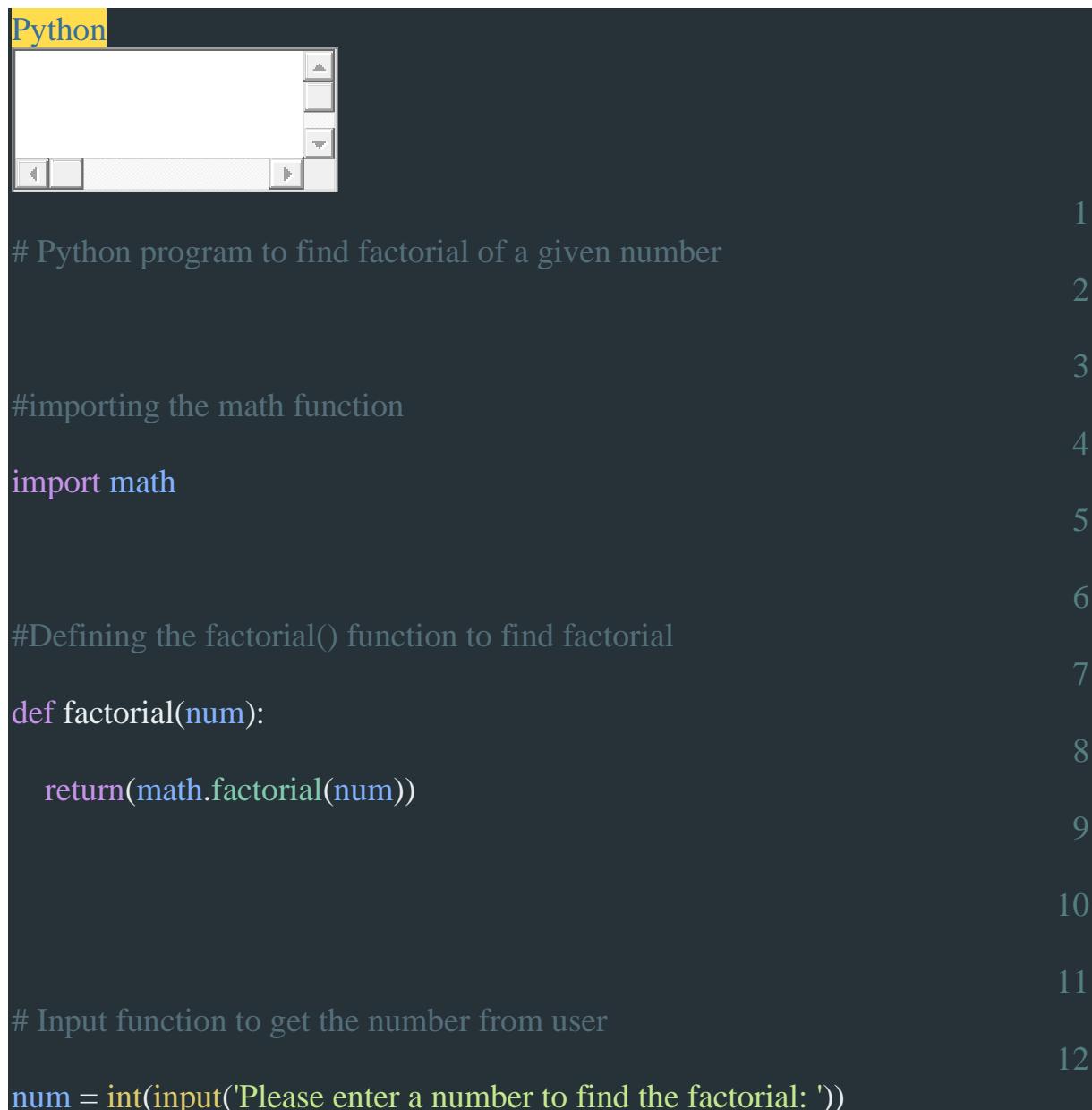
Enter a number to check prime or not: 37

The number 37 is a prime number

8) Write a Python factorial program without using if-else, for, and ternary operators.

Yes, we can write a Python program to find the factorial of a number using in-built Python functions.

math.factorial() function returns the factorial of a given number. Let's have a look at the Python factorial program using a math function:



A screenshot of a Python code editor window titled "Python". The code editor displays a numbered Python script. The numbers 1 through 12 are aligned to the right of the corresponding lines of code. The code itself is as follows:

```
1 # Python program to find factorial of a given number
2
3
4 #importing the math function
5 import math
6
7 #Defining the factorial() function to find factorial
8 def factorial(num):
9     return(math.factorial(num))
10
11
12 # Input function to get the number from user
13 num = int(input('Please enter a number to find the factorial: '))
```

```
13  
14  
#Printing the factorial of the given number  
15  
print("The factorial of the given number", num, "is",  
16  
    factorial(num))
```

Output1 of the Python factorial program

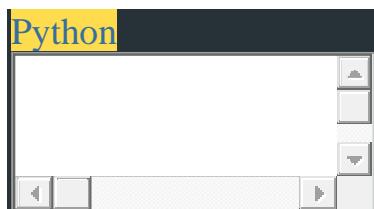
Please enter a number to find the factorial: 5
The factorial of the given number 5 is 120

The output2 of the Python factorial program

Pleas enter a number to find the factorial: 10
The factorial of the given number 10 is 3628800

9) Write a Python program to calculate the square root of a given number.

Here is the Python Program to calculate the square root:



```
1  
# Input function to get the input from the user  
2  
n = float(input('Enter a number: '))  
3  
4  
#Formula to calculate the square root of the number  
5  
n_sqrt = n ** 0.5  
6  
7  
#Printing the calculated square root of the given number  
8
```

```
print('The square root of {0} is {1}'.format(n ,n_sqrt))
```

Output1 of the Python program to find the square root

Enter a number: 2

The square root of 2.0 is 1.4142135623730951

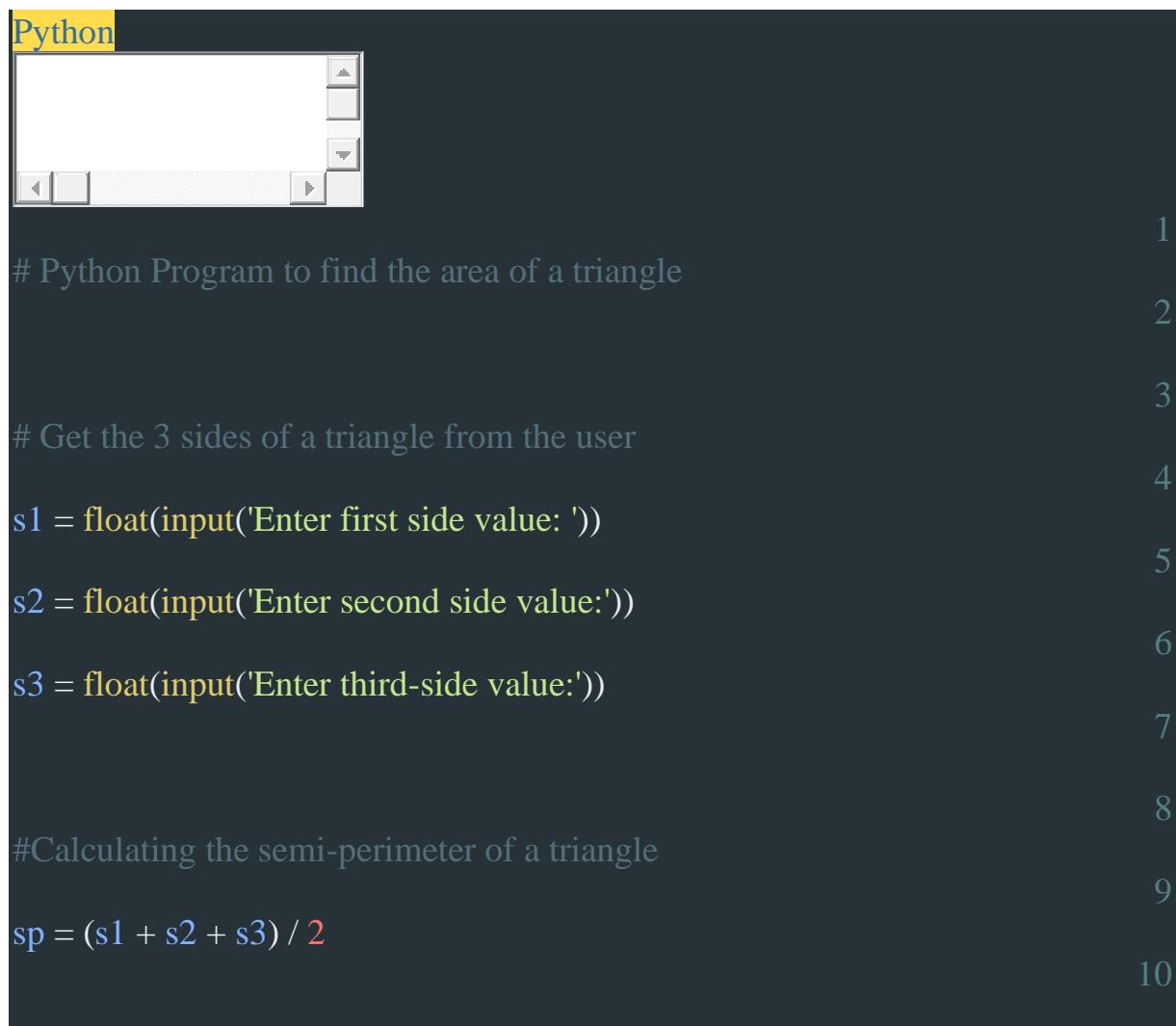
Output2 of the Python program to find the square root

Enter a number: 4

The square root of 4.0 is 2.0

10) Write a Python program to calculate the area of a triangle.

To calculate the area of a triangle we need the values of side one, side two, side three, and the semi-perimeter. Let's have a look at the Python area of the triangle program:



```
1 # Python Program to find the area of a triangle
2
3 # Get the 3 sides of a triangle from the user
4 s1 = float(input('Enter first side value: '))
5
6 s2 = float(input('Enter second side value: '))
7
8 s3 = float(input('Enter third-side value: '))
9
10 sp = (s1 + s2 + s3) / 2
```

```

#Calculating the area of a triangle
area = (sp*(sp-s1)*(sp-2)*(sp-s3)) ** 0.5
#print('The area of the triangle is: ', area)

```

The output of the Python area of the triangle program

Enter first side value: 3
 Enter the second side value:4
 Enter third-side value:5
 The area of the triangle is: 8.48528137423857

11) Write a Python program to check Armstrong's number.

The armstrong number can be defined as n-digit numbers equal to the sum of the nth powers of their digits are called armstrong numbers.

Armstrong Number Example:

153 is a armstrong number.

n=3 (numbr of digits)

nth powers of the digits – (111), (555), (333)

The number should equal the nth power of their digits, i.e

$$153 = (111) + (555) + (333).$$

Other Armstrong Numbers: 153, 370, 371, 407, 1634, 8208, 9474



```
1 # python program to check armstrong number
2
3
4 #Taking the input from user to check armstrong number
5 num=int(input("Enter the number to check armstrong number: "))
6
7 #Assigning the num value to arms
8 arms = num
9
10 #Finding the length of the number
11 length = len(str(num))
12 sum1 = 0
13
14 #Iterating the values to check armstrong number
15 while num != 0:
16     rem = num % 10
17     sum1 = sum1+(rem**length)
18     num = num//10
19
20 #Printing the result whether the given number is armstrong number or not
21 if arms == sum1:
22     print("The given number", arms, "is armstrong number")
```

```
else:
```

23

```
    print("The given number", arms, "is not an armstrong number")
```

Output1 of the Python armstrong number program

Enter the number to check armstrong number: 153

The given number 153 is armstrong number

Output2 of the Python armstrong number program

Enter the number to check armstrong number: 123

The given number 123 is not an armstrong number

12) Write a Python program to check leap year.

Do you know what is the leap year, it occurs once every 4 years. It contains additional days which makes a year 366 days.

Python

```
#Python program to check whether the given year is leap year or not
```

1

```
# Function implementation to check leap year
```

2

```
def LeapYear(Year):
```

3

```
    #Condition to check if the given year is leap year or not
```

4

```
    if((Year % 400 == 0) or
```

5

```
        (Year % 100 != 0) and
```

6

```
        (Year % 4 == 0)):
```

7

```
            print("The given Year is a leap year");
```

8

9

10

```
# Else it is not a leap year  
11  
else:  
12  
    print ("The given Year is not a leap year")  
13  
# Taking an input year from user  
14  
Year = int(input("Enter the year to check whether a leap year or not: "))  
15  
# Printing the leap year result  
16  
LeapYear(Year)  
17
```

Output1 of the Python leap year program

Enter the year to check whether a leap year or not: 2020
The given Year is a leap year

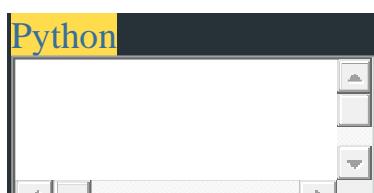
Output2 of the Python leap year program

Enter the year to check whether a leap year or not: 2021
The given year is not a leap year

13) Write a Python program to check prime numbers.

A prime number is a positive integer $p > 1$ that has no positive integer divisors other than 1 and p itself.

Example Prime Numbers: 2, 3, 5, 7, 11, 13, 17, 19, 23, 29, 31, 37,



```
1  
#Python program to check prime number  
2  
3
```

```
#Get the input to check whether a number is Prime or not  
4  
5  
prime_num = int(input("Enter a number to check whether it is prime or not: "))  
6  
if prime_num > 1:  
7  
    for i in range(2, int(prime_num/2)+1):  
8  
        if (prime_num % i) == 0:  
9  
            print(prime_num, "is not a prime number")  
10  
            break  
11  
    else:  
12  
        print(prime_num, "is a prime number")  
13  
else:  
14  
    print(prime_num, "is not a prime number")  
15
```

Output1 of Python program to check prime number

Enter a number to check whether it is prime or not: 7
7 is a prime number

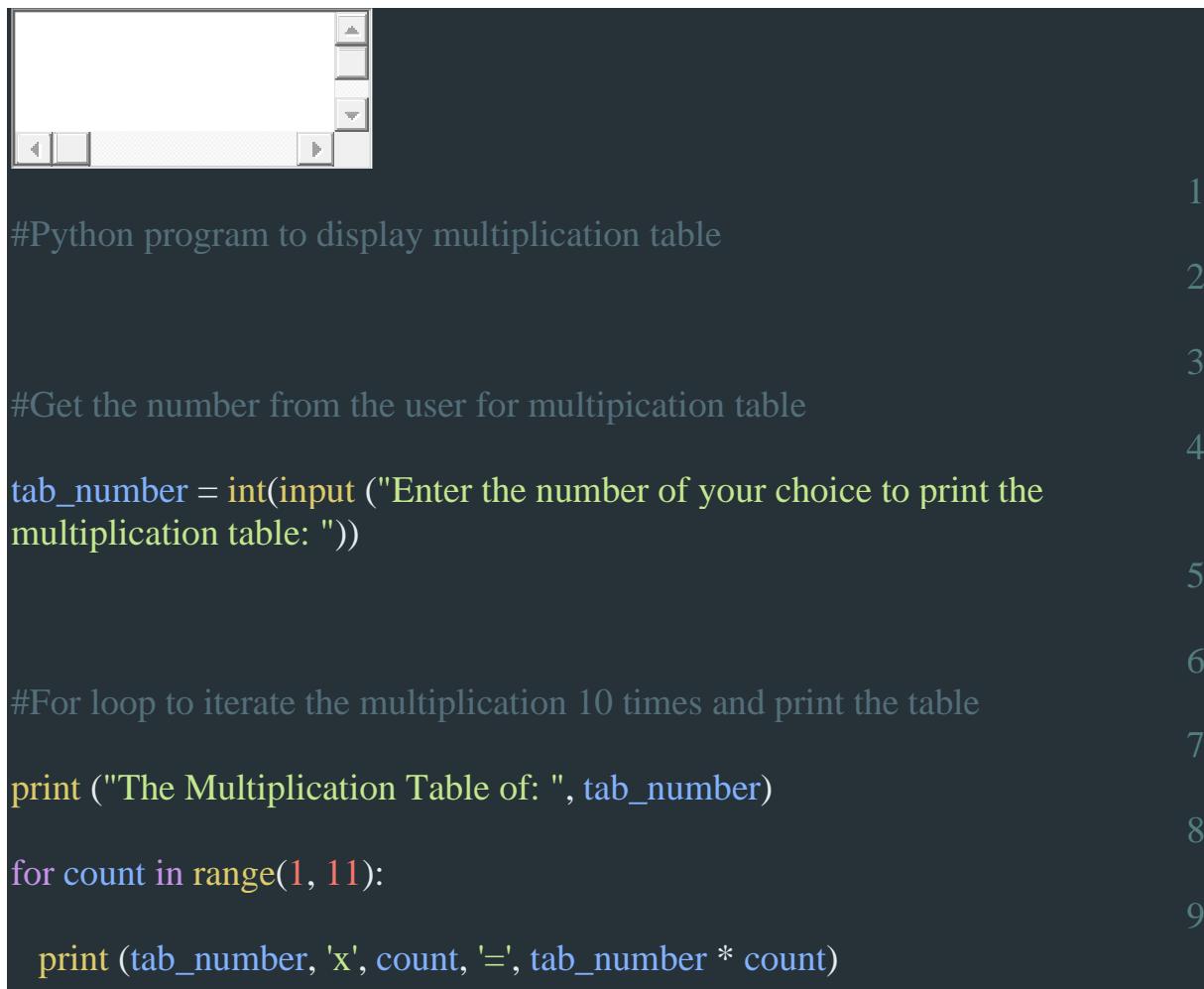
Output2 of Python program to check prime number

Enter a number to check whether it is prime or not: 10
10 is not a prime number

14) Write a Python program to display a multiplication table using for loop.

Here is the Python program to display the multiplication table of any number using for loop.

Python



A screenshot of a Python code editor window. The code is a script to print a multiplication table for a user-specified number. The code is numbered from 1 to 9 on the right side.

```
1 #Python program to display multiplication table
2
3 #Get the number from the user for multiplication table
4 tab_number = int(input ("Enter the number of your choice to print the
5 multiplication table: "))
6
7 #For loop to iterate the multiplication 10 times and print the table
8 print ("The Multiplication Table of: ", tab_number)
9 for count in range(1, 11):
10    print (tab_number, 'x', count, '=', tab_number * count)
```

The output of Python multiplication table

Enter the number of your choice to print the multiplication table: 10

The Multiplication Table of: 10

10 x 1 = 10

10 x 2 = 20

10 x 3 = 30

10 x 4 = 40

10 x 5 = 50

10 x 6 = 60

10 x 7 = 70

10 x 8 = 80

10 x 9 = 90

10 x 10 = 100

15) Write a Python program to swap two variables.

Python program to swap two variables.

Python

```
1 # Take inputs from the user to swap the two variables
2 num1 = input('Enter the first variable: ')
3 num2 = input('Enter the second variable: ')
4
5 #Printing the numbers before swap
6 print('The value of num1 before swapping: {}'.format(num1))
7 print('The value of num2 before swapping: {}'.format(num2))
8
9 #Use temporary variable and swap the values
10 temp = num1
11 num1 = num2
12 num2 = temp
13
14 #Printing the numbers after swap
15 print('The value of num1 after swapping: {}'.format(num1))
16 print('The value of num2 after swapping: {}'.format(num2))
```

The output of the Python program to swap two numbers:

Enter the first variable: 5
Enter the second variable: 10

The value of num1 before swapping: 5
The value of num2 before swapping: 10

The value of num1 after swapping: 10
The value of num2 after swapping: 5

16) Write a Python program to check if a given number is a Fibonacci number or not.

Here is the Python program to check the Fibonacci number:

```
Python
#Python progam to check fibonacci number
# First two fibonacci terms
fib_nums = [0, 1]
number = int(input('Enter the number you want to check for fibonacci number: '))
# Iterate fibonacci terms until the user number is reached
while fib_nums[-1] <= number:
    fib_nums.append(fib_nums[-1] + fib_nums[-2])
#printing the result whether the given number is fibonacci or not
```

```
if number in fib_nums:  
    print(f'Yes. {number} is a fibonacci number.')  
else:  
    print(f'No. {number} is NOT a fibonacci number.')  
13  
14  
15  
16
```

Output1 of Python program to check Fibonacci number

Enter the number you want to check for fibonacci number: 8
Yes. 8 is a fibonacci number.

Output2 of Python program to check Fibonacci number

Enter the number you want to check for fibonacci number: 10
No. 10 is NOT a fibonacci number.

17) Write a Python program to find the area of a circle.

The formula to find the area of the circle is: Area = $\pi * r^2$, where r is the radius of the circle.

Here is the Python program to find the area of the circle.

```
Python  
# Python program to find area of a circle  
def circ_Area(rad):  
    PI = 3.142  
    return PI * (rad*rad);  
1  
2  
3  
4  
5  
6
```

```
7  
rad = float(input('Enter the radius of the circle: '))  
8  
print("Area of the circle is %.6f" % circ_Area(rad));  
9
```

The output of the Python program is to find the area of a circle.

Enter the radius of the circle: 4
The area of the circle is 50.272000

18) Write a Python program to display the calendar.

Python program to display the calendar.

```
1  
Python  
2  
#Python program to display calendar  
3  
4  
import calendar  
5  
# Get the month and year from the users  
6  
year = int(input("Enter the year: "))  
7  
month = int(input("Enter the month: "))  
8  
# Displaying the calendar for the given year and month  
9  
print(calendar.month(year,month))  
10
```

The output of the Python program to display the calendar.

```
Enter the year: 2022
Enter the month: 10
October 2022
Mo Tu We Th Fr Sa Su
1 2
3 4 5 6 7 8 9
10 11 12 13 14 15 16
17 18 19 20 21 22 23
24 25 26 27 28 29 30
31
```

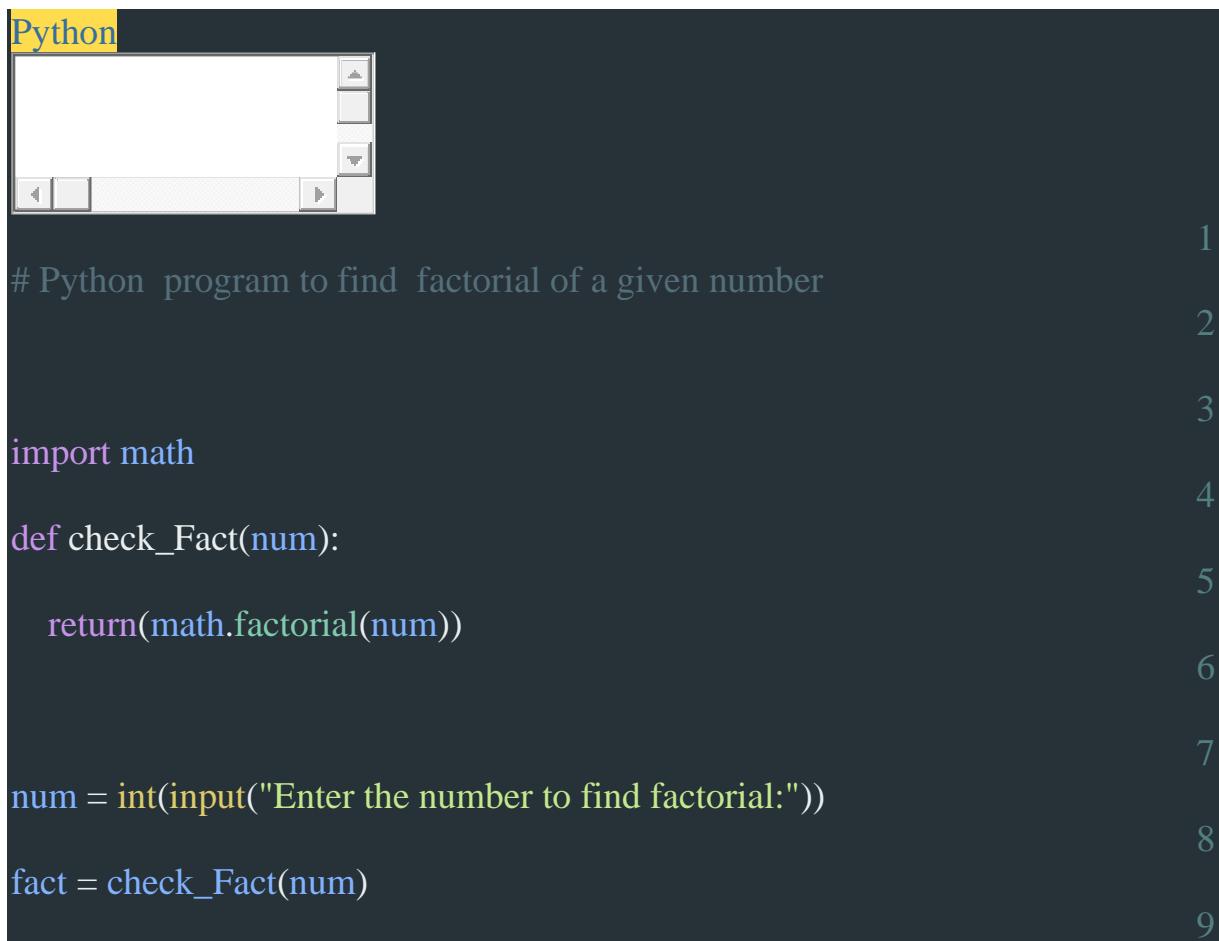
19) Write a Python program to find factorial of the given number.

The factorial formula:

$$n! = n * (n-1) * (n-2) * \dots * 1$$

$$4! = 4 \times 3 \times 2 \times 1 = 24$$

$$5! = 5 \times 4 \times 3 \times 2 \times 1 = 120$$



```
Python
# Python program to find factorial of a given number
import math
def check_Fact(num):
    return(math.factorial(num))
num = int(input("Enter the number to find factorial:"))
fact = check_Fact(num)
```

```
print("Factorial of the number", num, "is", fact)
```

10

The output of the Python factorial program

Enter the number to find factorial:5

Factorial of the number 5 is 120

20) Write a Python program to print all prime numbers in an interval.

Here is the Python program to display all the prime numbers within a range.

Python

```
# Python program to print all prime number in an interval
```

```
def disp_Prime(num1, num2):
```

```
    prime_List = []
```

```
    for i in range(num1, num2):
```

```
        if i == 0 or i == 1:
```

```
            continue
```

```
        else:
```

```
            for j in range(2, int(i/2)+1):
```

```
                if i % j == 0:
```

```
                    break
```

```
            else:
```

```

prime_List.append(i)                                14
return prime_List                                    15
                                                16
# Driver program                               17
starting_Num = int(input('Enter the starting range: '))
ending_Num = int(input('Enter the ending range: '))
lst = disp_Prime(starting_Num, ending_Num)          19
if len(lst) == 0:                                  20
    print("There are no prime numbers in this range") 21
else:
    print("The prime numbers in the given range are: ", lst) 23
                                                24
                                                25

```

The output of Python prime numbers display program

Enter the starting range: 5
 Enter the ending range: 25
 The prime numbers in this range are: [5, 7, 11, 13, 17, 19, 23]

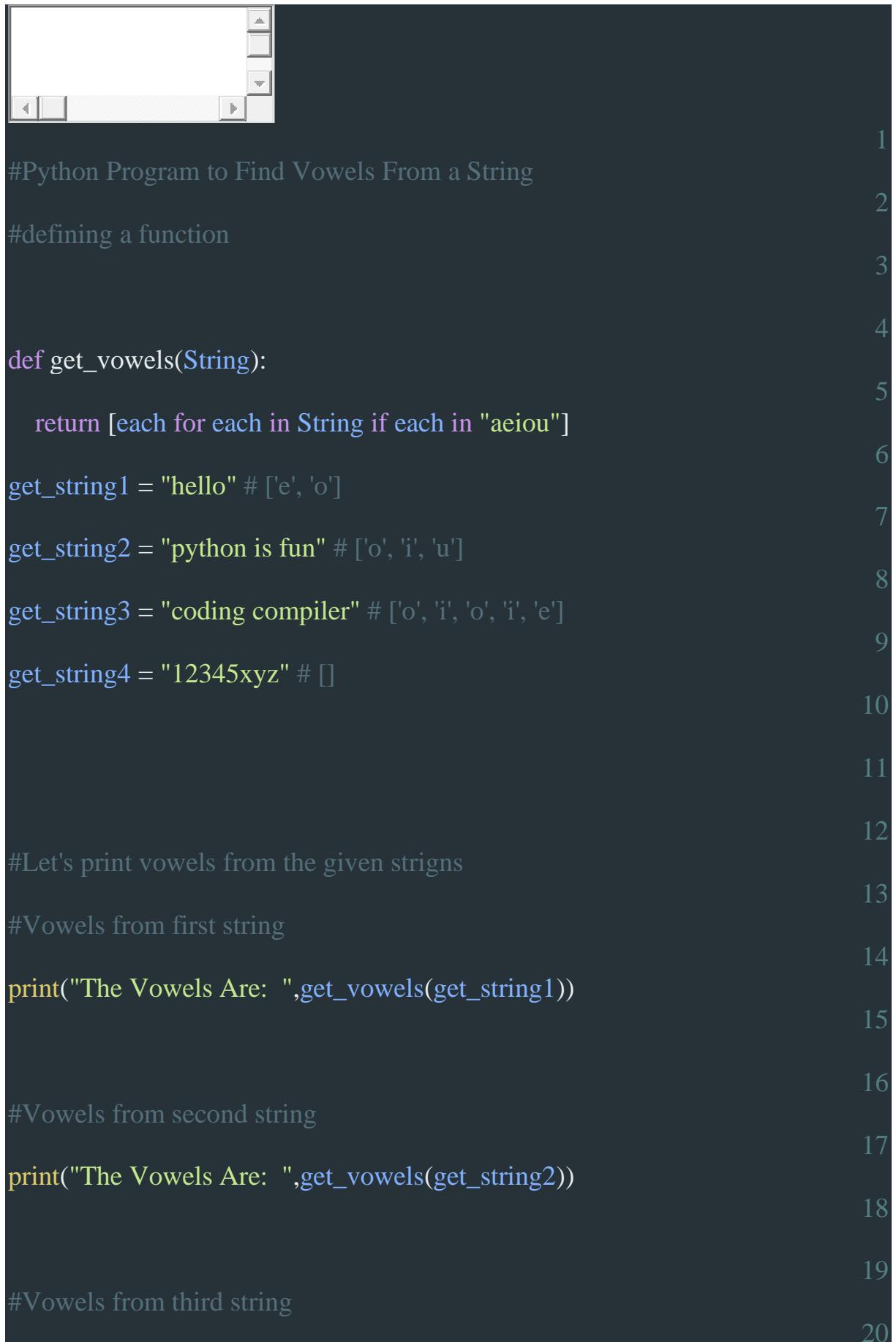
Python Strings Interview Questions & Answers

21) Write a Python Program to Find Vowels From a String.

Python Program to Find Vowels From a String

This Python coding example returns the vowels “a e i o u” present in a string. This Python program is useful when finding vowels, Let’s try this and see how it works.

Python



```
1 #Python Program to Find Vowels From a String
2
3 #defining a function
4
5 def get_vowels(String):
6     return [each for each in String if each in "aeiou"]
7
8 get_string1 = "hello" # ['e', 'o']
9
10 get_string2 = "python is fun" # ['o', 'i', 'u']
11
12 get_string3 = "coding compiler" # ['o', 'i', 'o', 'i', 'e']
13
14 get_string4 = "12345xyz" # []
15
16
17 #Let's print vowels from the given strigns
18
19 #Vowels from first string
20 print("The Vowels Are: ",get_vowels(get_string1))
21
22 #Vowels from second string
23 print("The Vowels Are: ",get_vowels(get_string2))
24
25 #Vowels from third string
26
```

```
print("The Vowels Are: ",get_vowels(get_string3))           21
                                                               22
#Vowels from fourth string                                23
                                                               24
print("The Vowels Are: ",get_vowels(get_string4))
```

The output of the above Python code:

```
The Vowels Are: ['e', 'o']
The Vowels Are: ['o', 'i', 'u']
The Vowels Are: ['o', 'i', 'o', 'i', 'e']
The Vowels Are: []
```

22) Write a Python Program to Convert Comma Separated List to a String.

Python Program to Convert Comma Separated List to a String

In this Python coding example we use the join() method to convert comma separated list to a string. Let's try this Python program and see how it works.

```
Python
# Python program to converting a comma separated list to a string
#comma seperated list
favorite_prog = ["Python", "SQL", "GO"]
#The join() method takes all items from the favorite_prog list and joins them
into one string.
```

```
print("What is your favorite programming language:", ", ".join(favorite_prog))
```

8

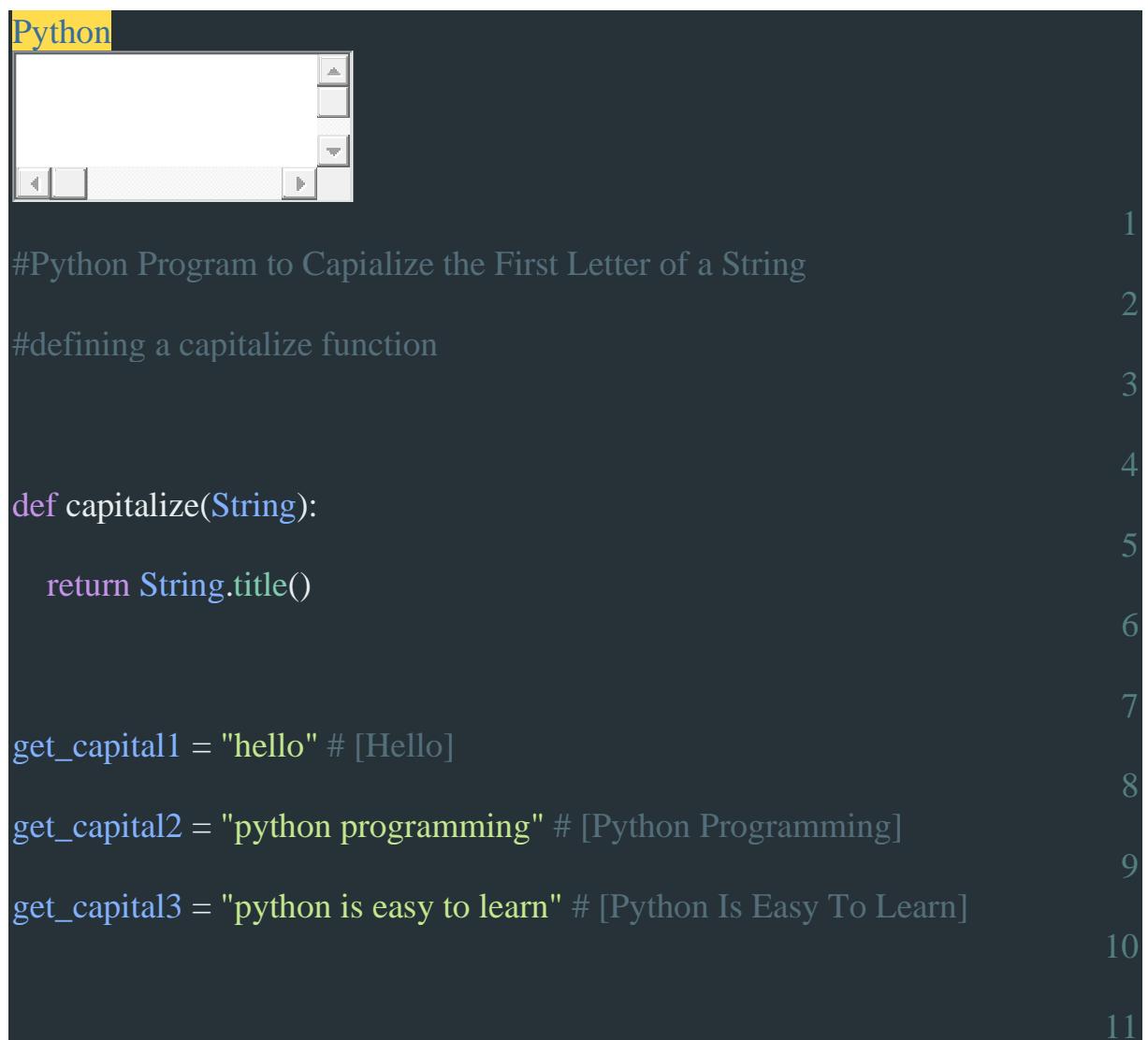
The output of the above Python code:

What is your favorite programming language: Python, SQL, GO

23) Write a Python Program to Capitalize the First Letter of a String.

Python program to convert every first letter of the characters in a string to an uppercase.

This Python coding example capitalizes every first letter of the characters in a string. This Python program is useful when converting the first letter of a string to a capital letter, Let's try this and see how it works.



```
Python
#Python Program to Capitalize the First Letter of a String
#defineing a capitalize function
def capitalize(String):
    return String.title()
get_capital1 = "hello" # [Hello]
get_capital2 = "python programming" # [Python Programming]
get_capital3 = "python is easy to learn" # [Python Is Easy To Learn]
```

```
12  
#Let's print capitalized string from the given strings  
13  
#Capitalized string from first string  
14  
print("The Capitalized String Is: ",capitalize(get_capital1))  
15  
16  
#Capitalized string from second string  
17  
print("The Capitalized String Is: ",capitalize(get_capital2))  
18  
19  
#Capitalized string from third string  
20  
print("The Capitalized String Is: ",capitalize(get_capital3))  
21
```

The output of the above Python code:

The Capitalized String Is: Hello
The Capitalized String Is: Python Programming
The Capitalized String Is: Python Is Easy To Learn

Python Implicit/Explicit Type Conversion Interview Questions

24) Write a Python Program to Implicit Type Conversion.

Python Program to Demonstrate Implicit Type Conversion.

This Python coding example explains implicit type conversion in Python. This Python program is useful when working with different data types and conversions. Let's try this and see how it works.



```
1 #Implicit type casting example
2
3 get_num1 = 199 #int
4
5 get_num2 = 1.25 #float
6
7 get_num3 = get_num1 + get_num2 #int+float=?
8
9
10 print("Datatype of get_num1:",type(get_num1))
11
12 print("Datatype of get_num2:",type(get_num2))
13
14
15 print("The value of get_num3 is:",get_num3)
16
17 print("Datatype of get_num3:",type(get_num3))
```

#Implicit conversion - Python always converts smaller data types to larger data types to avoid the loss of data.

The output of the above Python code:

Datatype of get_num1:

Datatype of get_num2:

The value of get_num3 is: 200.25

Datatype of get_num3: <class ‘float’>

25) What’s the Output of this Python program?

Let’s find out what’s the output of the below Python program.



```
1 #Let's find out what's the output of the below Python program.  
2  
3  
4 #Implicit type casting example  
5 num1 = 123  
6  
7 num2 = "456"  
8  
9 print("Data type of num_int:",type(num1))  
10 print("Data type of num_str:",type(num2))  
11 print("num1 data type is:",type(num1 + num2))
```

The output of the above Python code:

```
Data type of num_int:  
Data type of num_str:  
Traceback (most recent call last):  
File "", line 11, in  
TypeError: unsupported operand type(s) for +: 'int' and 'str'
```

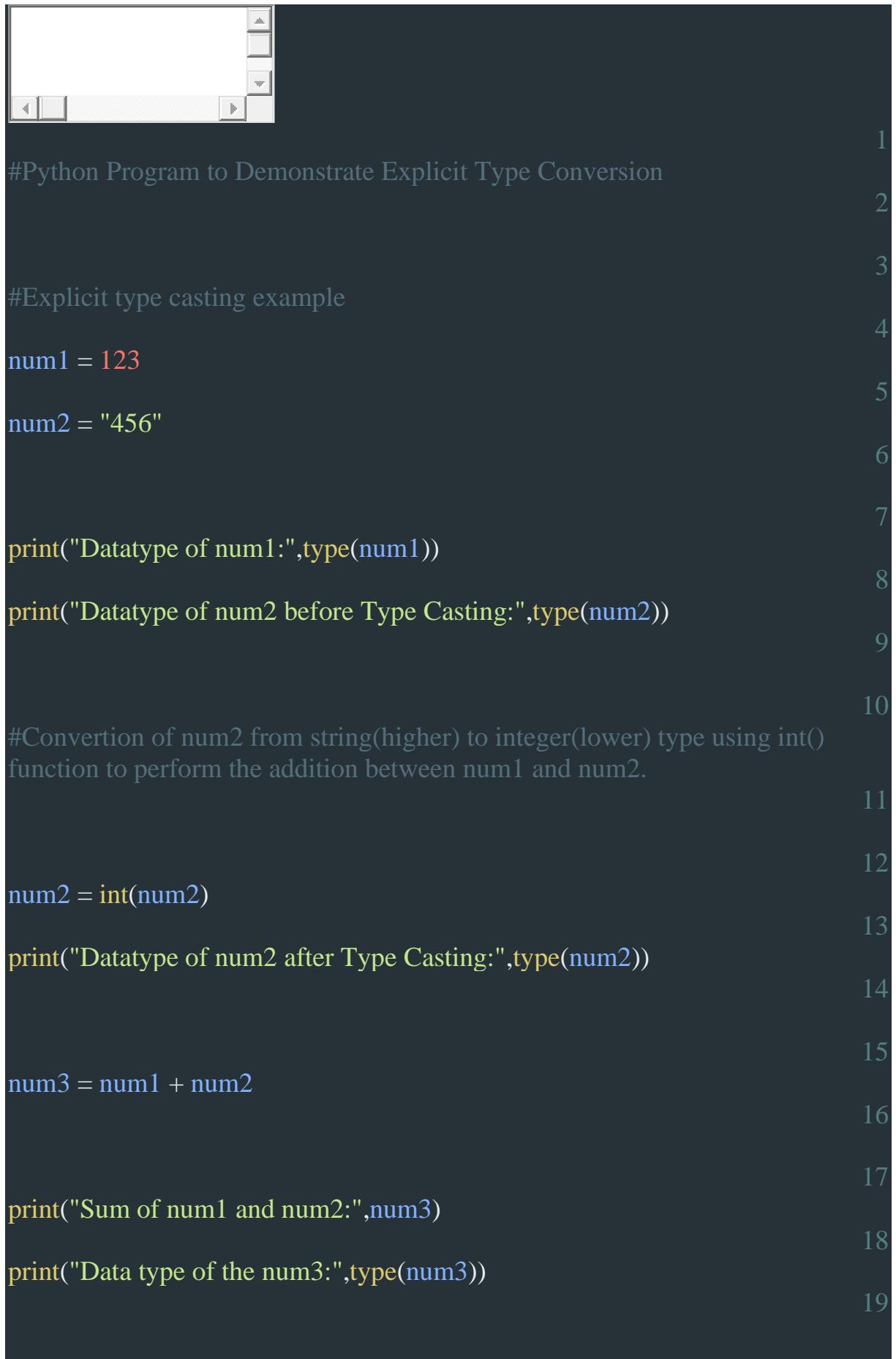
Python is not able to convert string type to integer type implicitly. So we can see TypeError in output.

26) Write a Python Program to Explicit Type Conversion.

Python Program to Demonstrate Explicit Type Conversion.

Python coding example explains explicit type conversion in Python. This Python program is useful when working with different data types and conversions. Let's try this Python code and see how it works.

Python



The image shows a screenshot of a Jupyter Notebook cell. The cell contains Python code demonstrating explicit type conversion. The code is numbered from 1 to 19 on the right side. The code itself is as follows:

```
1 #Python Program to Demonstrate Explicit Type Conversion
2
3 #Explicit type casting example
4 num1 = 123
5
6 num2 = "456"
7
8 print("Datatype of num1:",type(num1))
9
10 #Conversion of num2 from string(higher) to integer(lower) type using int()
11 # function to perform the addition between num1 and num2.
12
13 num2 = int(num2)
14
15 print("Datatype of num2 after Type Casting:",type(num2))
16
17 num3 = num1 + num2
18
19 print("Sum of num1 and num2:",num3)
20
21 print("Data type of the num3:",type(num3))
```

The output of the above Python code:

Datatype of num1: <class ‘int’>

Datatype of num2 before Type Casting: <class ‘str’>

Datatype of num2 after Type Casting: <class ‘int’>

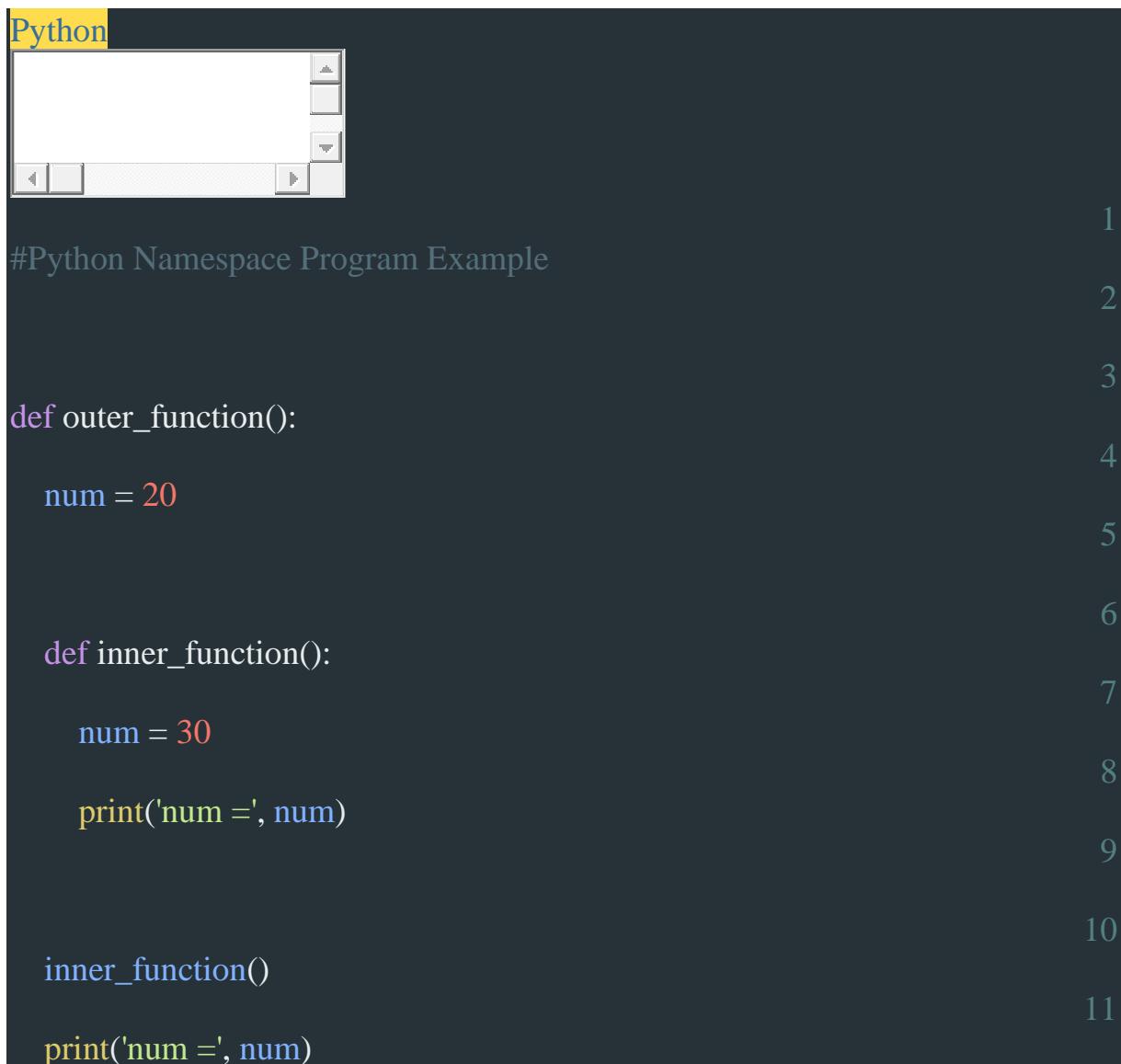
Sum of num1 and num2: 579

Data type of the num3: <class ‘int’>

Python Namespace Coding Interview Questions

27) What’s the Output of this Python Namespace program?

Python Namespace Program.



The screenshot shows a Python code editor window titled "Python". The code is as follows:

```
1 #Python Namespace Program Example
2
3
4 def outer_function():
5     num = 20
6
7     def inner_function():
8         num = 30
9         print('num =', num)
10
11     inner_function()
12     print('num =', num)
```

The code uses a standard Python syntax with nested functions. The variable 'num' is defined at two levels: once at the top level (outer function) and once inside the inner function. The inner function's definition overwrites the outer function's definition of 'num'. When the inner function is called, it prints its local value of 30. When the outer function is called, it prints its local value of 20.

```
12  
13  
14  
num = 10  
15  
outer_function()  
16  
print('num =', num)  
17
```

The output of the above Python code:

Three different variables “num” are defined in separate namespaces and accessed accordingly.

```
num = 30  
num = 20  
num = 10
```

28) What's the Output of this Python program?

Python Namespace Program.

```
Python  
1  
#Python Namespace Program Example  
2  
3  
def outer_function():  
4  
    global num  
5  
    num = 20  
6
```

```
7
def inner_function():
8
    global num
9
    num = 30
10
    print('num =', num)
11
12
inner_function()
13
print('num =', num)
14
15
16
num = 10
17
outer_function()
18
print('num =', num)
19
```

The output of the above Python code:

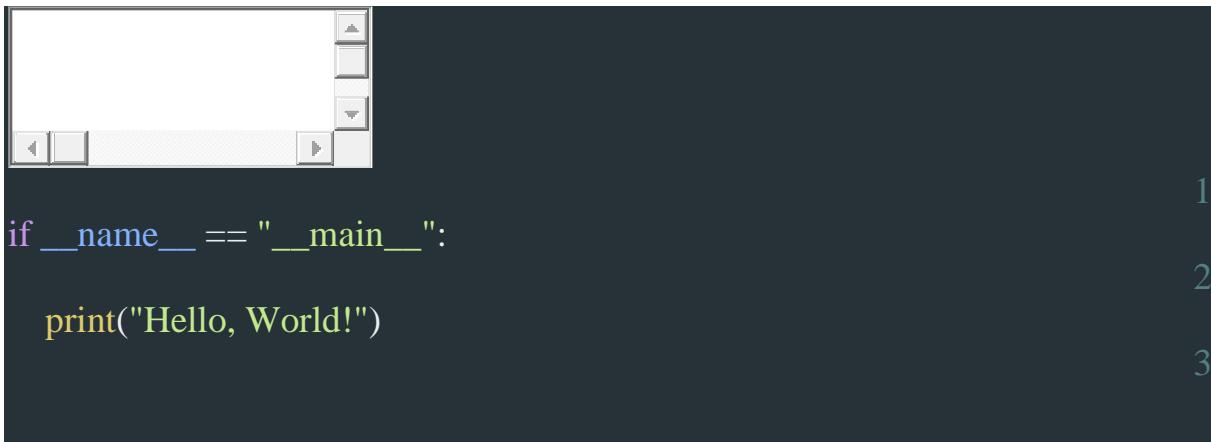
All references and assignments are to the global ‘num’ due to the use of the keyword ‘global’.

num = 30
num = 30
num = 30

29) What does this do, and why should one include the if statement?

Explain what the below Python program does.

Python



```
1 if __name__ == "__main__":
2     print("Hello, World!")
3
```

Answer#

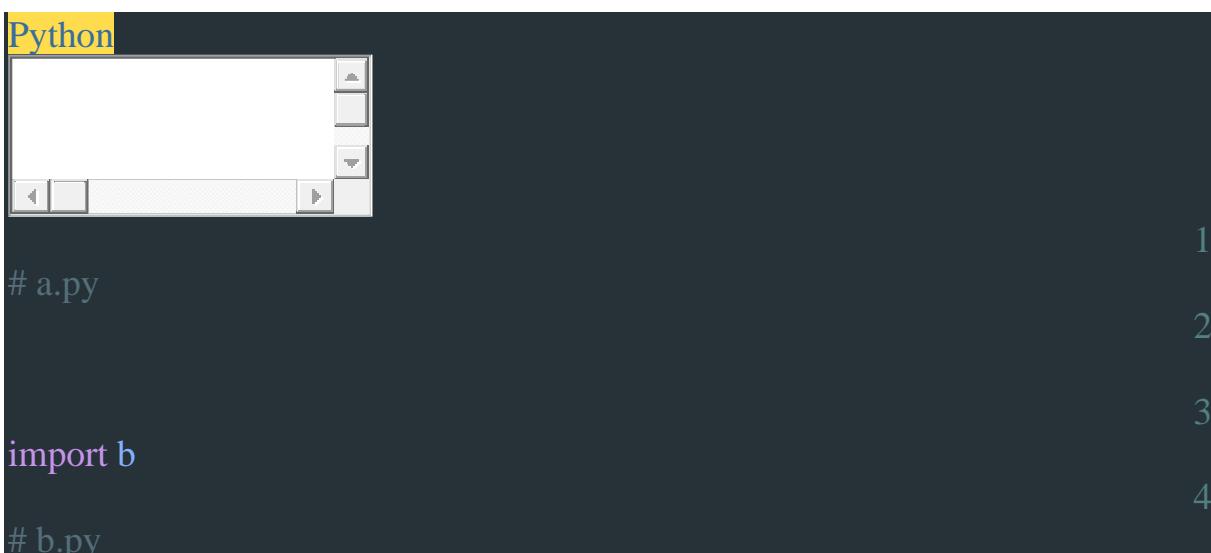
It's boilerplate code that protects users from accidentally invoking the script when they didn't intend to. Here are some common problems when the guard is omitted from a script:

If you import the guardless script in another script (e.g. `import my_script_without_a_name_eq_main_guard`), then the latter script will trigger the former to run at import time and use the second script's command line arguments. This is almost always a mistake.

If you have a custom class in the guardless script and save it to a pickle file, then unpickling it in another script will trigger an import of the guardless script, with the same problems outlined in the previous bullet.

Example Python Code:

Create the following two files:



```
Python
1 # a.py
2
3 import b
4 # b.py
```

```
5
6
7
print("__name__ equals " + __name__)
8
9
if __name__ == '__main__':
10
    print("if-statement was executed")
11
Now run each file individually.
12
13
14
Running python a.py:
15
16
$ python a.py
__name__ equals b
17
When a.py is executed, it imports the module b. This causes all the code inside
18
b to run. Python sets globals()['__name__'] in the b module to the module's
19
name, b.
#Running python b.py:
20
21
$ python b.py
22
__name__ equals __main__
23
```

if-statement was executed, when only the file b.py is executed, Python sets `globals()['name']` in this file to “**main**”. Therefore, the if statement evaluates to True this time.

30) What does `__all__` mean in Python?

Answer#

It's a list of public objects of that module, as interpreted by `import *`. It overrides the default of hiding everything that begins with an underscore.

Example Python Code:

Linked to, but not explicitly mentioned here, is exactly when `__all__` is used. It is a list of strings defining what symbols in a module will be exported when `from <module> import *` is used on the module.

For example, the following code in a `foo.py` explicitly exports the symbols `bar` and `baz`:

```
Python
1
2
3
4
5
6
7
8
9
__all__ = ['bar', 'baz']
waz = 5
bar = 10
def baz(): return 'baz'
# These symbols can then be imported like so:
```

```
10 from foo import *
11
12 print(bar)
13
14 print(baz)
15
16 # The following will trigger an exception, as "waz" is not exported by the
17 # module
18 print(waz)
```

If the `__all__` above is commented out, this code will then execute to completion, as the default behavior of `import *` is to import all symbols that do not begin with an underscore, from the given namespace.

Reference: <https://docs.python.org/tutorial/modules.html#importing-from-a-package>

NOTE: `__all__` affects the `from <module> import *` behavior only. Members that are not mentioned in `__all__` are still accessible from outside the module and can be imported with `from <module> import <member>`.

31) How to create a namespace package in Python?

In Python, a namespace package allows you to spread Python code among several projects. This is useful when you want to release related libraries as separate downloads. For example, with the directories `Package-1` and `Package-2` in `PYTHONPATH`,



Package-1/nspace/	<code>__init__.py</code>	2
Package-1/nspace/module1/	<code>__init__.py</code>	3
Package-2/nspace/	<code>__init__.py</code>	4
Package-2/nspace/module2/	<code>__init__.py</code>	5
the end-user can import <code>namespace.module1</code> and import <code>namespace.module2</code> .		

What's the best way to define a namespace package so more than one Python product can define modules in that namespace?

Answer#

On Python 3.3 you don't have to do anything, just don't put any `__init__.py` in your namespace package directories and it will just work. On pre-3.3, choose the `pkgutil.extend_path()` solution over the `pkg_resourcesdeclare_namespace()` one, because it's future-proof and already compatible with implicit namespace packages.

Python 3.3 introduces implicit namespace packages.

This means there are now three types of objects that can be created by an import `foo`:

A module represented by a `foo.py` file

A regular package, represented by a directory `foo` containing an `__init__.py` file

A namespace package, represented by one or more directories `foo` without any `__init__.py` files

32) How to create a Python namespace (`argparse.parse_args` value)?

To interactively test my python script, I would like to create a Namespace object, similar to what would be returned by `argparse.parse_args()`. The obvious way,



```
>>> import argparse  
2  
>>> parser = argparse.ArgumentParser()  
3  
>>> parser.parse_args()  
4  
Namespace()  
5  
>>> parser.parse_args("-a")  
6  
usage: [-h]  
7  
: error: unrecognized arguments: - a
```

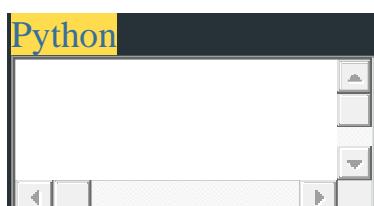
Process Python exited abnormally with code 2 may result in Python repl exiting (as above) on a silly error.

So, what is the easiest way to create a Python namespace with a given set of attributes?

E.g., I can create a dict on the fly (dict([("a",1),("b","c")])) but I cannot use it as a Namespace:

AttributeError: 'dict' object has no attribute 'a'

Answer#



```
Python  
1  
#You can create a simple class:  
2  
3  
class Namespace:  
4  
    def __init__(self, **kwargs):  
        self.__dict__.update(kwargs)  
5  
6
```

```
7  
#and it'll work the exact same way as the argparse Namespace class when it  
comes to attributes:
```

```
8  
9  
10  
>>> args = Namespace(a=1, b='c')
```

```
11  
>>> args.a
```

```
12  
>>> args.b
```

```
13  
'c'
```

```
14  
15  
#Alternatively, just import the class; it is available from the argparse module:
```

```
16  
17  
from argparse import Namespace
```

```
18  
19  
args = Namespace(a=1, b='c')
```

```
20  
As of Python 3.3, there is also types.SimpleNamespace, which essentially does  
the same thing:
```

```
21  
22  
>>> from types import SimpleNamespace
```

```
23  
>>> args = SimpleNamespace(a=1, b='c')
```

```
24  
>>> args.a
```

```
25  
1  
26  
>>> args.b
```

'c'

27

28

The two types are distinct; SimpleNamespace is primarily used for the sys.implementation attribute and the return value of time.get_clock_info().

Further comparisons:

- Both classes support equality testing; for two instances of the same class, instance_a == instance_b is true if they have the same attributes with the same values.
- Both classes have a helpful `__repr__` to show what attributes they have.
- Namespace() objects support containment testing; 'attrname' in instance is true if the namespace instance has an attribute named attrname. SimpleNamespace does not.
- Namespace() objects have an undocumented `._get_kwargs()` method that returns a sorted list of (name, value) attributes for that instance. You can get the same for either class using `sorted(vars(instance).items())`.
- While SimpleNamespace() is implemented in C and Namespace() is implemented in Python, attribute access is no faster because both use the same `__dict__` storage for the attributes. Equality testing and producing the representation are a little faster for SimpleNamespace() instances.

33) What does `__import__('pkg_resources').declare_namespace(__name__)` do?

In some `init.py` files of modules I saw such single line:

```
import('pkg_resources').declare_namespace(name)
```

What does it do and why people use it? Suppose it's related to dynamic importing and creating namespace at runtime.

Answer#

Here are two things:

1) `__import__` is a Python function that will import a package using a string as the name of the package. It returns a new object that represents the imported

package. So `foo = __import__('bar')` will import a package named bar and store a reference to its objects in a local object variable foo.

2 From setup utils `pkg_resources`' documentation, `declare_namespace()` “Declare[s] that the dotted package name is a “namespace package” whose contained packages and modules may be spread across multiple distributions.”

So `__import__('pkg_resources').declare_namespace(__name__)` will import the ‘`pkg_resources`’ package into a temporary and call the `declare_namespace` function stored in that temporary (the `__import__` function is likely used rather than the import statement so that there is no extra symbol left over named `pkg_resources`). If this code were in `my_namespace/__init__.py`, then `__name__` is `my_namespace` and this module will be included in the `my_namespace` namespace package.

34) What exactly does “`import *`” import?

In Python, what exactly does `import *` import? Does it import `__init__.py` found in the containing folder?

For example, is it necessary to declare from `project.model` import `__init__`, or is from `project.model` import `*` sufficient?

Answer#

The “advantage” of `from xyz import *` as opposed to other forms of import is that it imports everything (well, almost... [see (a) below] everything) from the designated module under the current module. This allows using the various objects (variables, classes, methods...) from the imported module without prefixing them with the module’s name.



The screenshot shows a terminal window with the title "Python". The window contains the following text:

```
#For example
>>> from math import *
```

The text is numbered 1, 2, and 3 on the right side of the terminal window.

```
4  
>>>pi  
5  
3.141592653589793  
6  
>>>sin(pi/2)  
7  
>>>1.0  
8
```

35) Why is “import *” bad in Python?

It is recommended to not to use import * in Python.

Can you please share the reason for that?

Answer#

Because it puts a lot of stuff into your namespace (might shadow some other object from the previous import and you won’t know about it).

Because you don’t know exactly what is imported and can’t easily find from which module a certain thing was imported (readability).

Because you can’t use cool tools like pyflakes to statically detect errors in your code.

36) What are Python namespaces and explain how to use namespaces & how this feature makes programming better?

Answer#

Namespace is a way to implement scope.

In Java (or C) the compiler determines where a variable is visible through static scope analysis.

In C, scope is either the body of a function or it’s global or it’s external. The compiler reasons this out for you and resolves each variable name based on scope rules. External names are resolved by the linker after all the modules are compiled.

In Java, scope is the body of a method function, or all the methods of a class. Some class names have a module-level scope, also. Again, the compiler figures this out at compile time and resolves each name based on the scope rules.

In Python, each package, module, class, function and method function owns a “namespace” in which variable names are resolved. Plus there’s a global namespace that’s used if the name isn’t in the local namespace.

Each variable name is checked in the local namespace (the body of the function, the module, etc.), and then checked in the global namespace.

Variables are generally created only in a local namespace. The global and nonlocal statements can create variables in other than the local namespace.

When a function, method function, module or package is evaluated (that is, starts execution) a namespace is created. Think of it as an “evaluation context”. When a function or method function, etc., finishes execution, the namespace is dropped. The variables are dropped. The objects may be dropped, also.

37) Is it a good idea to use the class as a namespace in Python?

Answer#

Yes, indeed. You can use Python classes strictly for namespacing as that is one of the special things they can do and do differently than modules. It’s a lot easier to define a class as a namespace inline in a file than to generate more files.

You should not do it without commenting on your code saying what it’s for. Python classes come in a lot of different forms and purposes and this makes it difficult to understand code you have not seen before.

A Python class used as a namespace is no less a Python class than one that meets the perception of what a class is in other languages. Python does not require a class to be instantiated to be useful. It does not require ivars and does not require methods. It is fairly flexible.

Classes can contain other classes too.

Lots of people have their ideas about what is or isn’t Pythonic. But if they were all worried about something like consistency, they’d push to have things like `len()` `dir()`, and `help()` be a method of objects rather than a global function.

38) Is it possible to add an object to the global namespace, for example, by using `globals()` or `dir()`?

```
Python
def insert_into_global_namespace(var_name, value):
    globals()[var_name] = value
insert_into_global_namespace('my_obj', 'an object')
print(f'my_obj = {my_obj}')
#But this only works in the current module.
```

Answer#

```
Python
#It is as simple as
globals()['var'] = "an object"
and/or
def insert_into_namespace(name, value, name_space=globals()):
    name_space[name] = value
```

```
8  
9  
insert_into_namespace("var", "an object")  
10  
  
11  
# Remark that globals is a built-in keyword, that is, 'globals' in  
__builtins__.__dict__ evaluates to True.
```

39) What's the python `__all__` module level variable for?

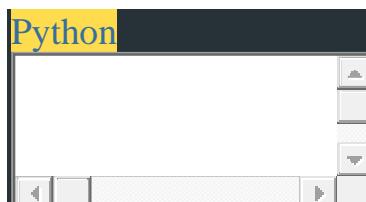
Answer#

It has two purposes:

Anybody who reads the source will know what the exposed public API is. It doesn't prevent them from poking around in private declarations but does provide a good warning not to.

When using `from mod import *`, only names listed in `__all__` will be imported. This is not as important, in my opinion, because importing everything is a really bad idea.

40) Is it possible to call static method from within class without qualifying the name in Python?



```
1  
2  
3  
4  
5  
6  
Python  
class MyClass:  
    @staticmethod  
    def foo():  
        print "hi"
```

```
@staticmethod
```

```
def bar():
```

```
    MyClass.foo()
```

7

8

Is there a way to make this work without naming MyClass in the call? i.e. so I can just say foo() on the last line?

Answer#

There is no way to use foo and get what you want. There is no implicit class scope, so foo is either a local or a global, neither of which you want.

You might find classmethods more useful:

Python

```
class MyClass:
```

1

```
    @classmethod
```

2

```
    def foo(cls):
```

3

```
        print "hi"
```

4

```
    @classmethod
```

5

```
    def bar(cls):
```

6

```
        cls.foo()
```

7

8

9

10

#This way, at least you don't have to repeat the name of the class.

41) What is the output of the below (`__class__.__name__`) Object Oriented Python code?

```
Python
1
2
3
4
5
6
class Programming:
    def name(self, name):
        return name
p = Programming()
print(p.__class__.__name__)

#Output
```

Programming

42) What's the output of the below (`type().__name__`) Python code?

```
Python
1
2
3
4
5
6
class Programming:
    def name(self, name):
        return name
p = Programming()
print(type(p).__name__)

#Output
```

#Output

Programming

43) What does super() do in Python? The difference between super().__init__() and explicit superclass __init__()

```
Python
1 #What's the difference between:
2
3
4 class Child(SomeBaseClass):
5     def __init__(self):
6         super(Child, self).__init__()
7
8 and:
9
10 class Child(SomeBaseClass):
11     def __init__(self):
12         SomeBaseClass.__init__(self)
```

I've seen super being used quite a lot in classes with only single inheritance. I can see why you'd use it in multiple inheritance but am unclear as to what the advantages are of using it in this kind of situation.

#Output

`super()` lets you avoid referring to the base class explicitly, which can be nice. But the main advantage comes with multiple inheritance, where all sorts of fun stuff can happen. See the standard docs on `super` if you haven't already.

Note that the syntax changed in Python 3.0: you can just say `super().__init__()` instead of `super(ChildB, self).__init__()` which IMO is quite a bit nicer.

What's the difference?

`SomeBaseClass.__init__(self)`
means to call `SomeBaseClass`'s `__init__`, while

`super().__init__()`
means to call a bound `__init__` from the parent class that follows
`SomeBaseClass`'s child class (the one that defines this method) in the instance's Method Resolution Order (MRO).

If the instance is a subclass of this child class, there may be a different parent that comes next in the MRO.

Explained simply

When you write a class, you want other classes to be able to use it. `super()` makes it easier for other classes to use the class you're writing.

Good architecture allows you to postpone decision-making as long as possible.

`super()` can enable that sort of architecture.

When another class subclasses the class you wrote, it could also be inherited from other classes. And those classes could have an `__init__` that comes after this `__init__` based on the ordering of the classes for method resolution.

Without `super` you would likely hard-code the parent of the class you're writing (like the example does). This would mean that you would not call the next `__init__` in the MRO, and you would thus not get to reuse the code in it.

If you're writing your own code for personal use, you may not care about this distinction. But if you want others to use your code, using `super` is one thing that allows greater flexibility for users of the code.

Python 2 versus 3

This works in Python 2 and 3:

```
super(Child, self).__init__()
```

This only works in Python 3:

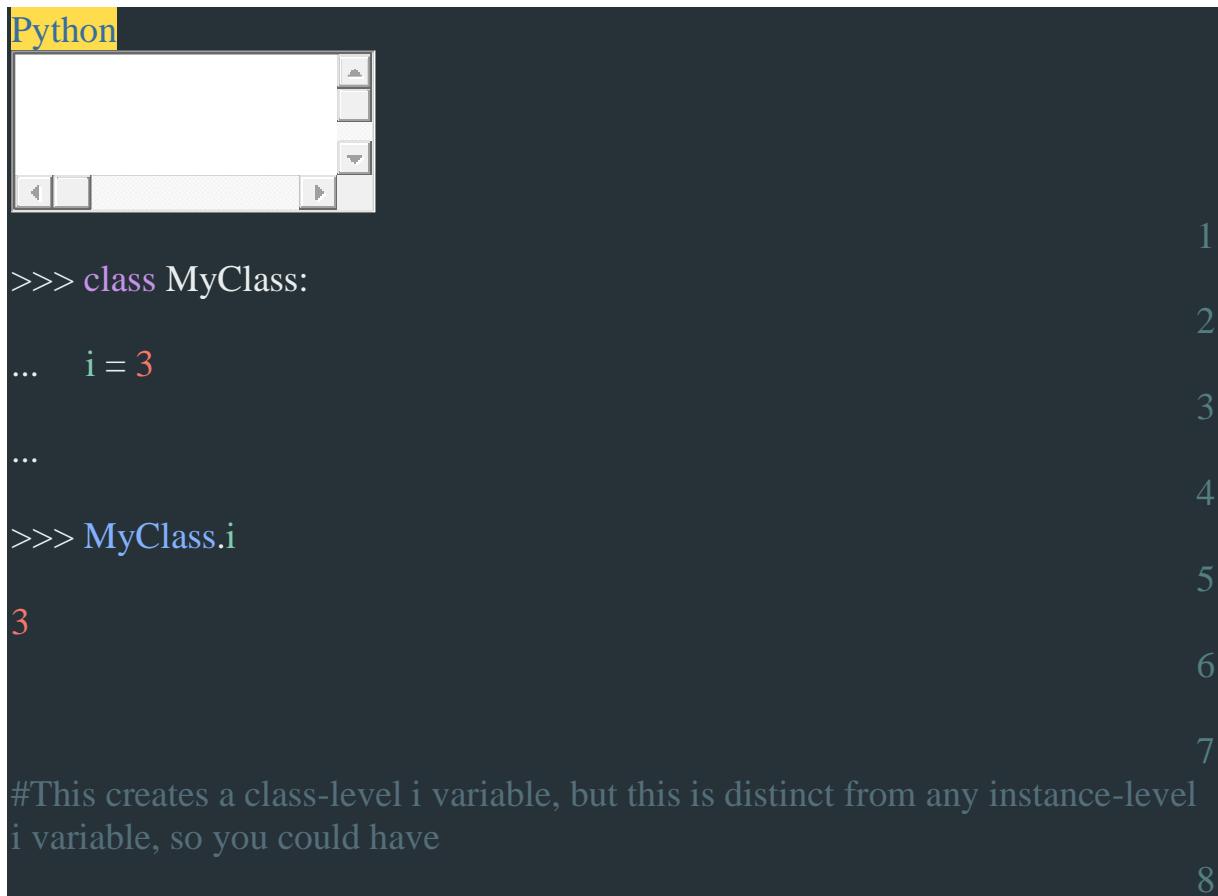
```
super().__init__()
```

It works with no arguments by moving up in the stack frame and getting the first argument to the method (usually `self` for an instance method or `cls` for a class method – but could be other names) and finding the class (e.g. `Child`) in the free variables (it is looked up with the name `class` as a free closure variable in the method).

44) How to create static class variables or methods in Python?

Answer#

Variables declared inside the class definition, but not inside a method are class or static variables:



The screenshot shows a terminal window with the title "Python". The code is as follows:

```
Python
>>> class MyClass:
...     i = 3
...
>>> MyClass.i
3
#This creates a class-level i variable, but this is distinct from any instance-level
i variable, so you could have
```

The code is numbered 1 through 8 on the right side.

```
>>> m = MyClass()                                     9
>>> m.i = 4                                         10
>>> MyClass.i, m.i                                11
>>> (3, 4)                                         12
>>> (3, 4)                                         13
```

This is different from C++ and Java, but not so different from C#, where a static member can't be accessed using a reference to an instance.

45) Why do Python classes inherit object?

Why does the following class declaration inherit from object?

```
class MyClass(object):
...  
...
```

Answer#

```
Python
#Python 3
class MyClass(object): = New-style class
class MyClass: = New-style class (implicitly inherits from object)
#Python 2
```

```
8  
class MyClass(object): = New-style class  
9  
class MyClass: = OLD-STYLE CLASS
```

Explanation:

When defining base classes in Python 3.x, you're allowed to drop the object from the definition. However, this can open the door for a seriously hard-to-track problem...

Python introduced new-style classes back in Python 2.2, and by now old-style classes are really quite old.

46) What is the purpose of the ‘self’ parameter in Python? Why is it needed?



#Consider this example:

```
1  
2  
3  
4  
5  
class MyClass:  
    def func(self, name):  
        self.name = name
```

Answer#

The reason you need to use self. is because Python does not use a special syntax to refer to instance attributes.

Python decided to do methods in a way that makes the instance to which the method belongs be passed automatically, but not received automatically: the first parameter of methods is the instance the method is called on.

That makes methods entirely the same as functions and leaves the actual name to use up to you (although the self is the convention, and people will generally

frown at you when you use something else.) `self` is not special to the code, it's just another object.

Python could have done something else to distinguish normal names from attributes — special syntax like Ruby has, or requiring declarations like C++ and Java do, or perhaps something yet more different — but it didn't.

Python's all for making things explicit, making it obvious what's what, and although it doesn't do it entirely everywhere, it does do it for instance attributes.

That's why assigning to an instance attribute needs to know what instance to assign to, and that's why it needs `self`.

```
Python
1
#Example: Let's say you have a class ClassA which contains a method methodA
2
3
4 def methodA(self, arg1, arg2):
5     # do something
6
7 #and ObjectA is an instance of this class.
8
9
10 #Now when ObjectA.methodA(arg1, arg2) is called, python internally converts
  it for you as:
11
12
13 ClassA.methodA(ObjectA, arg1, arg2)
14
15 #The self variable refers to the object itself.
```

47) What is the difference between old-style and new-style classes in Python?

Answer#

Up to Python 2.1, old-style classes were the only flavor available to the user.

The concept of the (old-style) class is unrelated to the concept of type: if `x` is an instance of an old-style class, then `x.class` designates the class of `x`, but `type(x)` is always `.`

This reflects the fact that all old-style instances, independently of their class, are implemented with a single built-in type, called an instance.

New-style classes were introduced in Python 2.2 to unify the concepts of class and type. A new-style class is simply a user-defined type, no more, no less.

If `x` is an instance of a new-style class, then `type(x)` is typically the same as `x.class` (although this is not guaranteed – a new-style class instance is permitted to override the value returned for `x.class`).

The major motivation for introducing new-style classes is to provide a unified object model with a full meta-model.

It also has a number of immediate benefits, like the ability to subclass most built-in types, or the introduction of “descriptors”, which enable computed properties.

For compatibility reasons, classes are still old-style by default.

New-style classes are created by specifying another new-style class (i.e. a type) as a parent class, or the “top-level type” object if no other parent is needed.

The behavior of new-style classes differs from that of old-style classes in a number of important details in addition to what `type` returns.

Some of these changes are fundamental to the new object model, like the way special methods are invoked. Others are “fixes” that could not be implemented before for compatibility concerns, like the method resolution order in case of multiple inheritances.

Python 3 only has new-style classes.

No matter if you subclass from the object or not, classes are new-style in Python 3.

Declaration-wise:

New-style classes inherit from an object, or from another new-style class.

```
class NewStyleClass(object):  
    pass
```

```
class AnotherNewStyleClass(NewStyleClass):  
    pass  
Old-style classes don't.
```

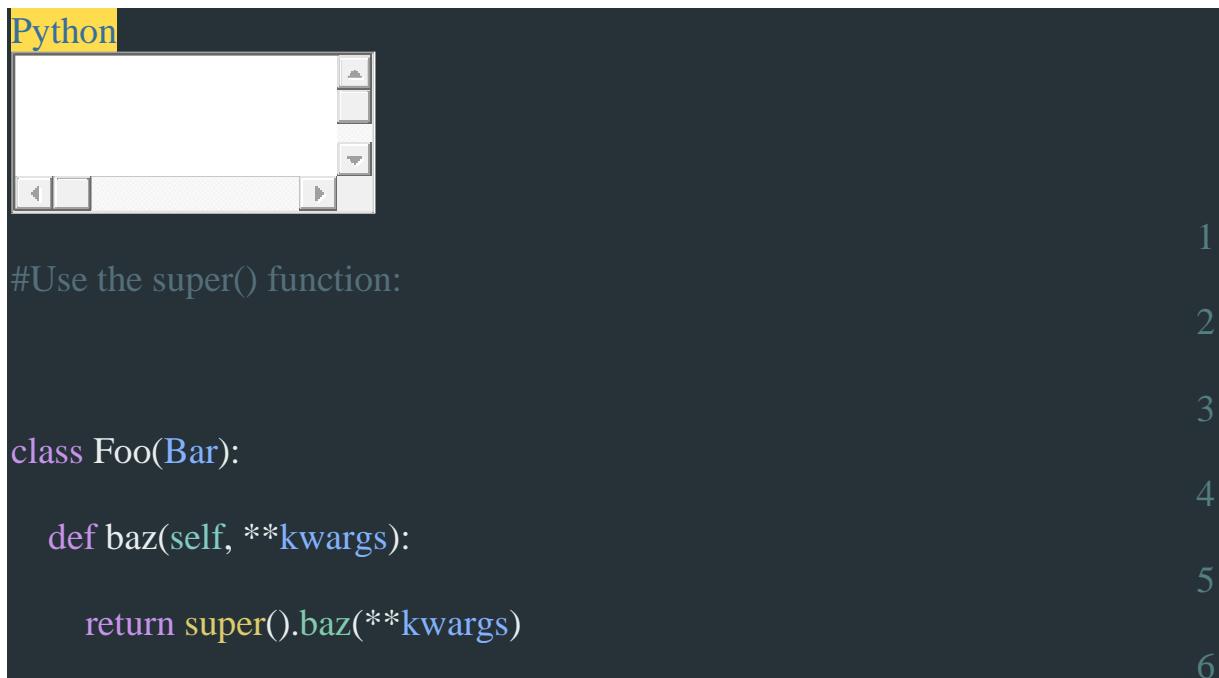
```
class OldStyleClass():  
    pass
```

Python 3 Note:

Python 3 doesn't support old-style classes, so either form noted above results in a new-style class.

48) How to call a parent class's method from a child class in Python?

Answer#



The screenshot shows a Python code editor window with a dark theme. The title bar says "Python". The code area contains the following:

```
1 #Use the super() function:  
2  
3  
4 class Foo(Bar):  
5     def baz(self, **kwargs):  
6         return super().baz(**kwargs)
```

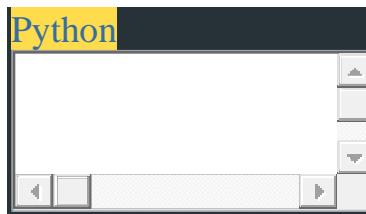
The code is numbered 1 through 6 on the right side. The first line starts with a hash symbol (#), indicating a comment. The fifth line contains the `super()` function call, which is highlighted in yellow.

```
7  
#For Python < 3, you must explicitly opt in to using new-style classes and use:  
8  
9  
class Foo(Bar):  
10  
    def baz(self, arg):  
11        return super(Foo, self).baz(arg)  
12
```

Example: super(type[, object-or-type])

Return a proxy object that delegates method calls to a parent or sibling class of type. This is useful for accessing inherited methods that have been overridden in a class.

The search order is the same as that used by getattr() except that the type itself is skipped.



```
1  
class A(object):    # deriving from 'object' declares A as a 'new-style-class'  
2  
    def foo(self):  
3        print "foo"  
4  
5  
class B(A):  
6  
    def foo(self):  
7        super(B, self).foo()  # calls 'A.foo()'  
8
```

```
myB = B()
```

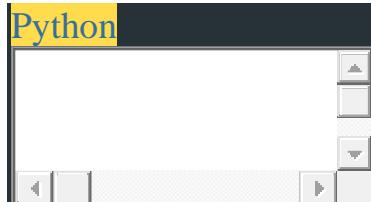
9

```
myB.foo()
```

10

49) How to print instances of a class using print() in Python?

When I try to print an instance of a class, I get an output like this:



```
>>> class Test():
```

1

```
...     def __init__(self):
```

2

```
...         self.a = 'foo'
```

3

```
...
```

4

```
>>> print(Test())
```

5

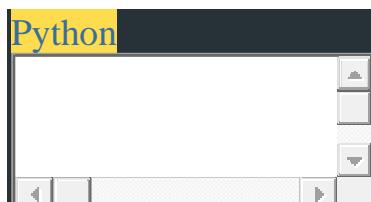
```
<__main__.Test object at 0x7fc9a9e36d60>
```

6

How can I define the printing behavior (or the string representation) of a class and its instances?

For example, referring to the above code, how can I modify the Test class so that printing an instance shows the value?

Answer#



```
>>> class Test:
```

1

2

```
...     def __repr__(self):          3
...         return "Test()"           4
...
...     def __str__(self):           5
...         return "member of Test"   6
...
...
>>> t = Test()                  7
>>> t                         8
Test()                          9
>>> print(t)                  10
                                11
member of Test
```

The `__str__` method is what gets called when you print it, and the `__repr__` method is what happens when you use the `repr()` function (or when you look at it with the interactive prompt).

If no `__str__` method is given, Python will print the result of `__repr__` instead. If you define `__str__` but not `__repr__`, Python will use what you see above as the `__repr__`, but still use `__str__` for printing.

50) What is the difference between `__init__` and `__call__`?

I want to know the difference between `__init__` and `__call__` methods.

```
Python
#For example:
class test:
```

```
4  
5  
def __init__(self):  
6    self.a = 10  
7  
8  
def __call__(self):  
9    b = 20  
10
```

Answer#

So, the `__init__` method is used when the class is called to initialize the instance, while the `__call__` method is called when the instance is called.

Example:

The `__init__` is used to initialize newly created object, and receives arguments used to do that:

```
Python  
1  
2  
3  
4  
5  
6  
7  
class Foo:  
    def __init__(self, a, b, c):  
        # ...  
x = Foo(1, 2, 3) # __init__
```

```

#The __call__ implements function call operator.          8
                                                 9
class Foo:                                         10
    def __call__(self, a, b, c):                     11
        # ...                                         12
                                                 13
x = Foo()                                         14
x(1, 2, 3) # __call__                         15

```

Python OOP Coding Interview Questions

51) What are meta classes in Python?

Answer#

A metaclass is the class of a class. A class defines how an instance of the class (i.e. an object) behaves while a metaclass defines how a class behaves. A class is an instance of a metaclass.

While in Python you can use arbitrary callables for metaclasses (like Jerub shows), the better approach is to make it an actual class itself. `type` is the usual metaclass in Python. `type` is itself a class, and it is its own type. You won't be able to recreate something like `type` purely in Python, but Python cheats a little. To create your own metaclass in Python you really just want to subclass `type`.

A metaclass is most commonly used as a class-factory. When you create an object by calling the class, Python creates a new class (when it executes the ‘class’ statement) by calling the metaclass.

Combined with the normal `__init__` and `__new__` methods, metaclasses, therefore, allow you to do ‘extra things’ when creating a class, like registering the new class with some registry or replacing the class with something else entirely.

When the class statement is executed, Python first executes the body of the class statement as a normal block of code. The resulting namespace (a dict) holds the attributes of the class-to-be.

The metaclass is determined by looking at the base classes of the class-to-be (metaclasses are inherited), at the **metaclass** attribute of the class-to-be (if any) or the **__metaclass__** global variable. The metaclass is then called with the name, bases and attributes of the class to instantiate it.

However, metaclasses actually define the type of a class, not just a factory for it, so you can do much more with them. You can, for instance, define normal methods on the metaclass.

These metaclass-methods are like class methods in that they can be called on the class without an instance, but they are also not like class methods in that they cannot be called on an instance of the class. `type.__subclasses__()` is an example of a method on the type metaclass.

You can also define the normal ‘magic’ methods, like **__add__**, **__iter__**, and **__getattr__**, to implement or change how the class behaves.

The **__metaclass__** attribute

In Python 2, you can add a **__metaclass__** attribute when you write a class (see next section for the Python 3 syntax):

```
class Foo(object):
    __metaclass__ = something...
    [...]
```

If you do so, Python will use the metaclass to create the class Foo.

Metaclasses in Python 3

The syntax to set the metaclass has been changed in Python 3:

```
class Foo(object, metaclass=something):
    ...
```

i.e. the **metaclass** attribute is no longer used, in favor of a keyword argument in the list of base classes.

The behavior of metaclasses however stays largely the same.

One thing added to metaclasses in Python 3 is that you can also pass attributes as keyword-arguments into a metaclass, like so:

```
class Foo(object, metaclass=something, kwarg1=value1, kwarg2=value2):
```

52) What is the difference between `@staticmethod` and `@classmethod` in Python?

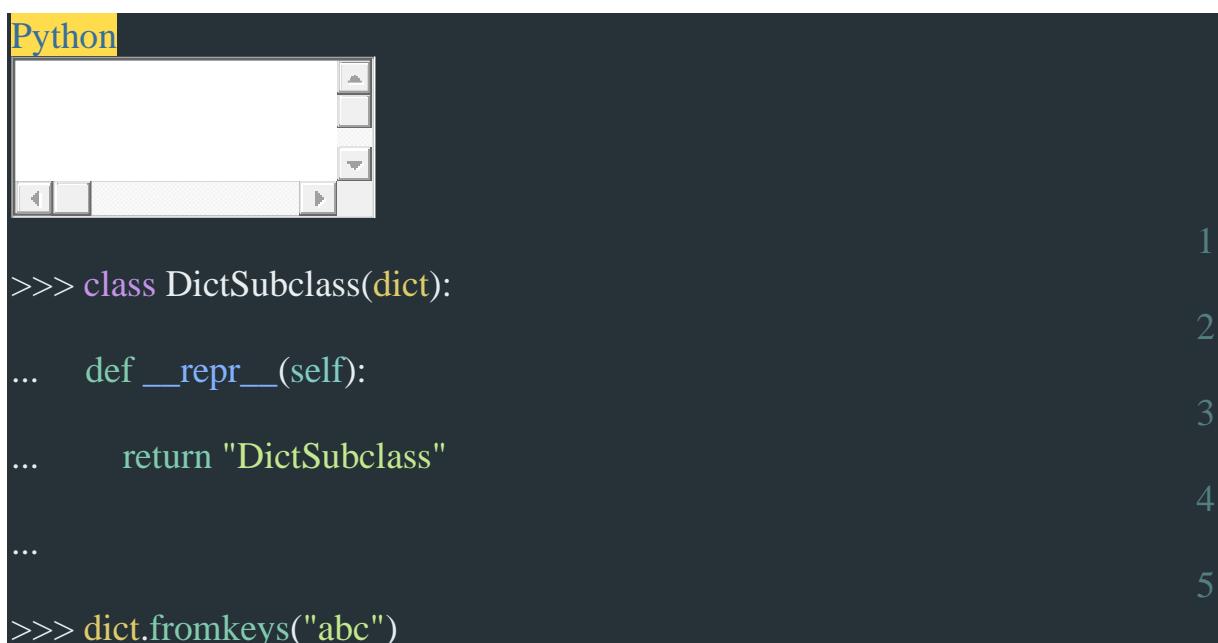
Answer#

A `staticmethod` is a method that knows nothing about the class or instance it was called on. It just gets the arguments that were passed, no implicit first argument. It is basically useless in Python — you can just use a module function instead of a `staticmethod`.

A `classmethod`, on the other hand, is a method that gets passed the class it was called on, or the class of the instance it was called on, as first argument.

This is useful when you want the method to be a factory for the class: since it gets the actual class it was called on as first argument, you can always instantiate the right class, even when subclasses are involved.

Observe for instance how `dict.fromkeys()`, a `classmethod`, returns an instance of the subclass when called on a subclass:

A screenshot of a Python terminal window. The title bar says "Python". The window contains a code editor with the following Python code:

```
>>> class DictSubclass(dict):
...     def __repr__(self):
...         return "DictSubclass"
...
>>> dict.fromkeys("abc")
```

Numbered vertical markers (1 through 5) are placed to the right of the code lines to indicate specific points of interest.

The output of the code is not visible in the screenshot.

```
6 {'a': None, 'c': None, 'b': None}
7
8 >>> DictSubclass.fromkeys("abc")
9
10 DictSubclass
```

53) What is the meaning of single and double underscore before an object name in Python?

Answer#

Single Underscore – In a class, names with a leading underscore indicate to other programmers that the attribute or method is intended to be used inside that class.

However, privacy is not enforced in any way. Using leading underscores for functions in a module indicates it should not be imported from somewhere else.

From the PEP-8 style guide:

`_single_leading_underscore`: weak “internal use” indicator. E.g. `from M import *` does not import objects whose name starts with an underscore.

Double Underscore (Name Mangling)

From the Python docs:

Any identifier of the form `__spam` (at least two leading underscores, at most one trailing underscore) is textually replaced with `_classname__spam`, where `classname` is the current class name with leading underscore(s) stripped.

This mangling is done without regard to the syntactic position of the identifier, so it can be used to define class-private instance and class variables, methods, variables stored in globals, and even variables stored in instances. private to this class on instances of other classes.

And a warning from the same page:

Name mangling is intended to give classes an easy way to define “private” instance variables and methods, without having to worry about instance variables defined by derived classes, or mucking with instance variables by code outside the class.

Note that the mangling rules are designed mostly to avoid accidents; it still is possible for a determined soul to access or modify a variable that is considered private.

Example:



```
Python
>>> class MyClass():
...     def __init__(self):
...         self.__superprivate = "Hello"
...         self._semiprivate = ", world!"
...
>>> mc = MyClass()
>>> print mc.__superprivate
Traceback (most recent call last):
File "<stdin>", line 1, in <module>
AttributeError: myClass instance has no attribute '__superprivate'
>>> print mc._semiprivate
, world!
>>> print mc.__dict__
1
2
3
4
5
6
7
8
9
10
11
12
13
14
```

```
{'_MyClass__superprivate': 'Hello', '_semiprivate': 'world!'}  
15
```

54) What are the differences between type() and isinstance() in Python?

Answer#

isinstance caters for inheritance (an instance of a derived class is an instance of a base class, too), while checking for equality of type does not (it demands identity of types and rejects instances of subtypes, AKA subclasses).

Normally, in Python, you want your code to support inheritance, of course (since inheritance is so handy, it would be bad to stop code using yours from using it!), so isinstance is less bad than checking identity of types because it seamlessly supports inheritance.

```
Python  
1  
#Using type:  
2  
3  
import types  
4  
5  
if type(a) is types.DictType:  
6  
    do_something()  
7  
if type(b) in types.StringTypes:  
8  
    do_something_else()  
9  
10  
11
```

```
#Using isinstance:  
12  
13  
if isinstance(a, dict):  
14  
    do_something()  
15  
if isinstance(b, str) or isinstance(b, unicode):  
16  
    do_something_else()  
17
```

55) What is a mixin and why is it useful in Python?

Answer#

A mixin is a special kind of multiple inheritance. There are two main situations where mixins are used:

- 1) You want to provide a lot of optional features for a class.
- 2) You want to use one particular feature in a lot of different classes.

For an example of number one, consider werkzeug's request and response system. I can make a plain old request object by saying:

```
Python  
1  
from werkzeug import BaseRequest  
2  
3  
class Request(BaseRequest):  
4  
    pass  
5
```

If I want to add accept header support, I would make that

Python

```
1 from werkzeug import BaseRequest, AcceptMixin  
2  
3 class Request(AcceptMixin, BaseRequest):  
4     pass
```

If I wanted to make a request object that supports accept headers, etags, authentication, and user agent support, I could do this:

Python

```
1 from werkzeug import BaseRequest, AcceptMixin, ETagRequestMixin,  
2 UserAgentMixin, AuthenticationMixin  
3  
4 class Request(AcceptMixin, ETagRequestMixin, UserAgentMixin,  
5 AuthenticationMixin, BaseRequest):  
6     pass
```

The difference is subtle, but in the above examples, the mixin classes weren't made to stand on their own. In more traditional multiple inheritance, the AuthenticationMixin (for example) would probably be something more like Authenticator. That is, the class would probably be designed to stand on its own.

56) What is the purpose of `__slots__` in Python?

Answer#

The special attribute `__slots__` allows you to explicitly state which instance attributes you expect your object instances to have, with the expected results:

- 1) faster attribute access.
- 2) space savings in memory.

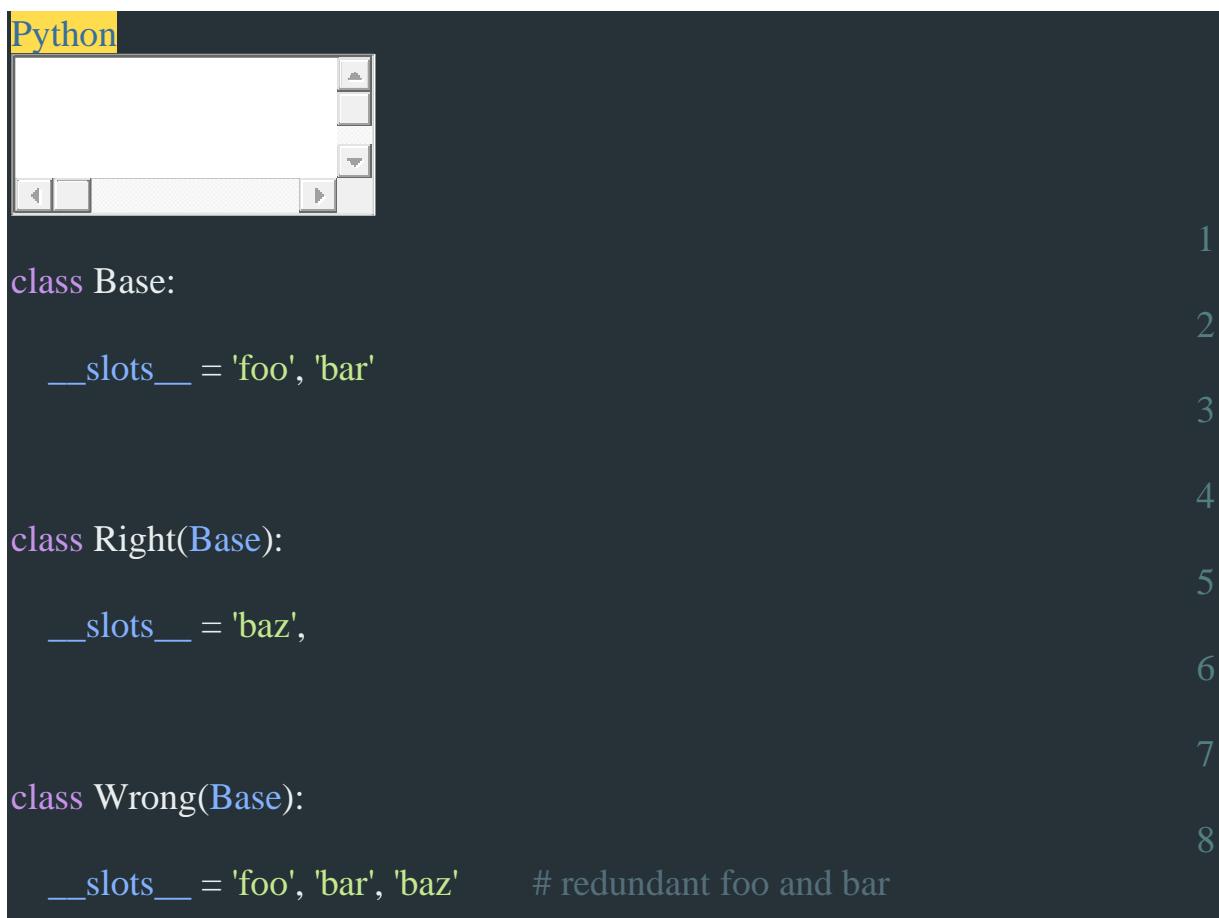
The space savings is from

- 1) Storing value references in slots instead of `__dict__`.
- 2) Denying `__dict__` and `__weakref__` creation if parent classes deny them and you declare `__slots__`.

Quick Caveats

A small caveat, you should only declare a particular slot one time in an inheritance tree.

For example:



The screenshot shows a Python code editor window with a yellow title bar labeled "Python". The code area contains the following Python code with line numbers 1 through 8 on the right:

```
1 class Base:  
2     __slots__ = 'foo', 'bar'  
3  
4 class Right(Base):  
5     __slots__ = 'baz',  
6  
7 class Wrong(Base):  
8     __slots__ = 'foo', 'bar', 'baz'      # redundant foo and bar
```

The code editor highlights line 8 with a red squiggle under the line, indicating a syntax error. The message "Redeclaration of __slots__" is displayed in the status bar at the bottom of the editor.

Python doesn't object when you get this wrong (it probably should), problems might not otherwise manifest, but your objects will take up more space than they otherwise should. Python 3.8:

```
Python
>>> from sys import getsizeof
>>> getsizeof(Right()), getsizeof(Wrong())
(56, 72)
#This is because the Base's slot descriptor has a slot separate from the Wrong's.
#This shouldn't usually come up, but it could:
>>> w = Wrong()
>>> w.foo = 'foo'
>>> Base.foo.__get__(w)
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
AttributeError: foo
>>> Wrong.foo.__get__(w)
'foo'
```

The biggest caveat is for multiple inheritance – multiple “parent classes with nonempty slots” cannot be combined.

To accommodate this restriction, follow best practices: Factor out all but one or all parents’ abstraction which their concrete class respectively and your new concrete class collectively will inherit from – giving the abstraction(s) empty slots (just like abstract base classes in the standard library).

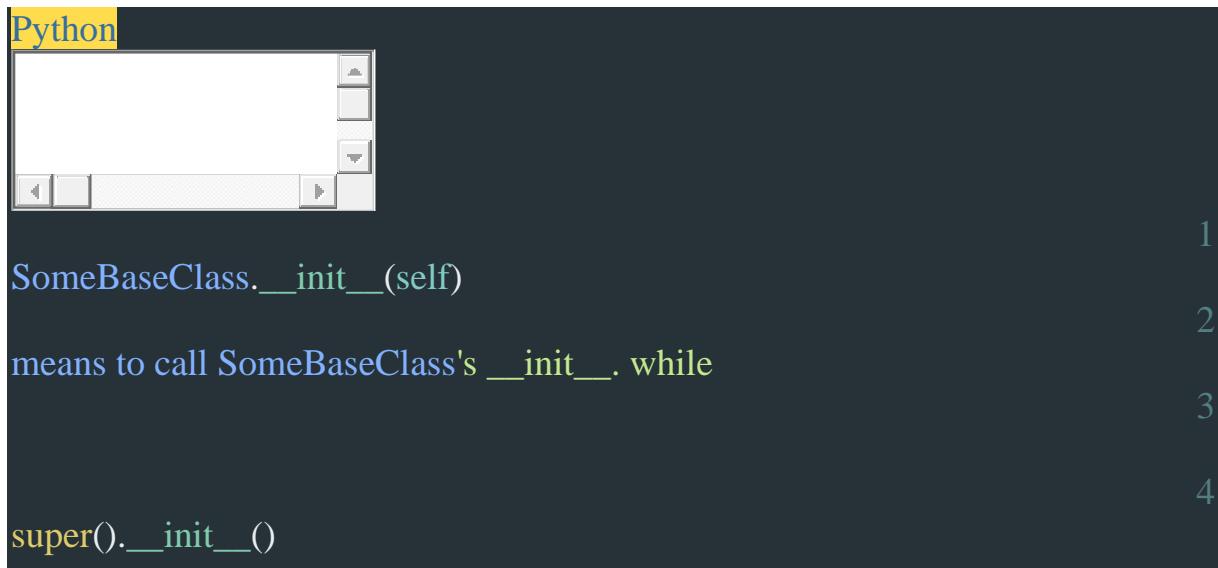
57) What does ‘super’ do in Python? What is the difference between `super().__init__()` and explicit superclass `__init__()`?

Answer#

The benefits of `super()` in single-inheritance are minimal — mostly, you don’t have to hard-code the name of the base class into every method that uses its parent methods.

However, it’s almost impossible to use multiple-inheritance without `super()`. This includes common idioms like mixins, interfaces, abstract classes, etc. This extends to code that later extends yours. If somebody later wanted to write a class that extended `Child` and a mixin, their code would not work properly.

What’s the difference?



SomeBaseClass.`__init__`(`self`)
means to call SomeBaseClass's `__init__`. while
`super().__init__()`

1
2
3
4

means to call a bound **init** from the parent class that follows `SomeBaseClass`'s child class (the one that defines this method) in the instance's Method Resolution Order (MRO).

If the instance is a subclass of this child class, there may be a different parent that comes next in the MRO.

Explained simply

When you write a class, you want other classes to be able to use it. `super()` makes it easier for other classes to use the class you're writing.

As Bob Martin says, the good architecture allows you to postpone decision-making as long as possible.

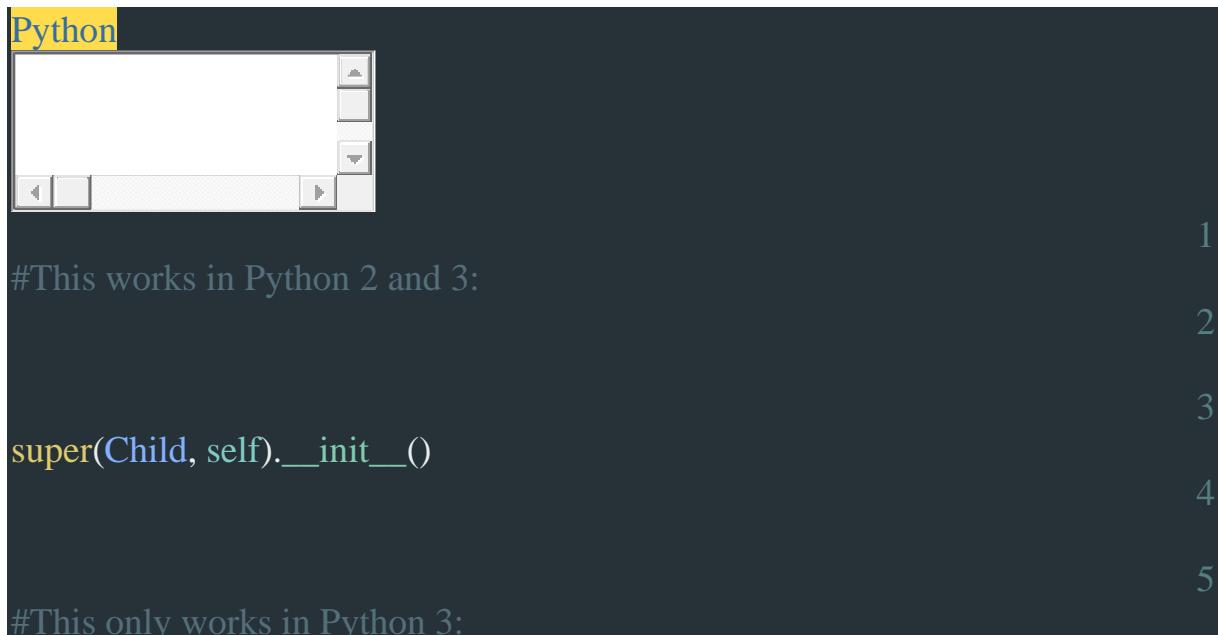
`super()` can enable that sort of architecture.

When another class subclasses the class you wrote, it could also be inheriting from other classes. And those classes could have an `__init__` that comes after this `__init__` based on the ordering of the classes for method resolution.

Without `super` you would likely hard-code the parent of the class you're writing (like the example does). This would mean that you would not call the next `__init__` in the MRO, and you would thus not get to reuse the code in it.

If you're writing your own code for personal use, you may not care about this distinction. But if you want others to use your code, using `super` is one thing that allows greater flexibility for users of the code.

Python 2 versus 3



```
Python
#This works in Python 2 and 3:
super(Child, self).__init__()
#This only works in Python 3:
```

The screenshot shows a terminal window titled "Python". The code in the terminal is as follows:

```
#This works in Python 2 and 3:
super(Child, self).__init__()
#This only works in Python 3:
```

The code consists of two parts. The first part, "#This works in Python 2 and 3:", contains the line `super(Child, self).__init__()`. The second part, "#This only works in Python 3:", is currently empty. The terminal window has a dark background with light-colored text. The title bar is blue with the word "Python". The bottom right corner of the terminal window has a small number "1" above "2", "3", "4", and "5".

6

7

super().__init__()

It works with no arguments by moving up in the stack frame and getting the first argument to the method (usually self for an instance method or cls for a class method – but could be other names) and finding the class (e.g. Child) in the free variables (it is looked up with the name __class__ as a free closure variable in the method).

58) Explain the ‘__enter__’ and ‘__exit__’ methods in Python?

Answer#

Using these magic methods (__enter__, __exit__) allows you to implement objects which can be used easily with the with statement.

The idea is that it makes it easy to build code which needs some ‘cleardown’ code executed (think of it as a try-finally block).

A useful example could be a database connection object (which then automatically closes the connection once the corresponding ‘with’-statement goes out of scope):



Python

```
1 class DatabaseConnection(object):
2
3     def __enter__(self):
4         # make a database connection and return it
5         ...
6         return self.dbconn
7
```

1
2
3
4
5
6
7

```
8 def __exit__(self, exc_type, exc_val, exc_tb):
9     # make sure the dbconnection gets closed
10    self.dbconn.close()
11    ...
12
```

As explained above, use this object with the with statement (you may need to do from **future** import `with_` statement at the top of the file if you're on Python 2.5).

with DatabaseConnection() as mydbconn:

do stuff

PEP343 — The ‘with’ statement’ has a nice writeup as well.

59) How do I implement interfaces in Python?

Answer#

Interfaces are not necessary in Python. This is because Python has proper multiple inheritance, and also ducktyping, which means that the places where you must have interfaces in Java, you don't have to have them in Python.

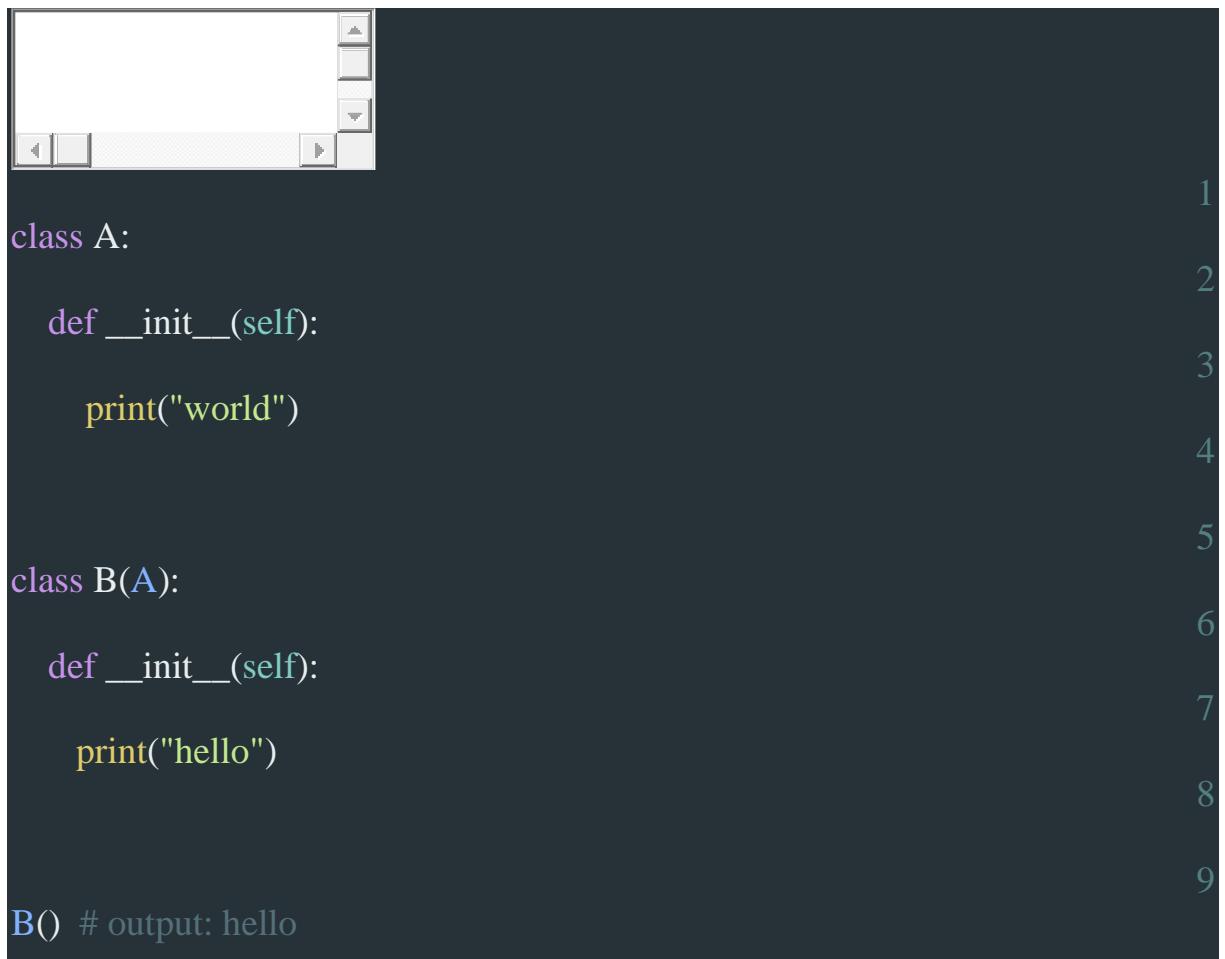
That said, there are still several uses for interfaces. Some of them are covered by Pythons Abstract Base Classes, introduced in Python 2.6. They are useful, if you want to make base classes that cannot be instantiated, but provide a specific interface or part of an implementation.

Another usage is if you somehow want to specify that an object implements a specific interface, and you can use ABC's for that too by subclassing from them. Another way is `zope.interface`, a module that is a part of the Zope Component Architecture, a really awesomely cool component framework.

Here you don't subclass from the interfaces, but instead mark classes (or even instances) as implementing an interface. This can also be used to look up components from a component registry. Supercool!

60) How to invoke the super constructor in Python?

Python



```
1 class A:
2     def __init__(self):
3         print("world")
4
5 class B(A):
6     def __init__(self):
7         print("hello")
8
9 B() # output: hello
```

In all other languages I've worked with the super constructor is invoked implicitly. How does one invoke it in Python? I would expect `super(self)` but this doesn't work.

Answer#

there are multiple ways to call super class methods (including the constructor), however in Python-3.x the process has been simplified:

Python-3.x



```
1 class A(object):
2     def __init__(self):
```

```
3
print("world")
4
5
class B(A):
6
def __init__(self):
7
    print("hello")
8
super().__init__()
```

Python-2.x

In python 2.x, you have to call the slightly more verbose version super(, self), which is equivalent to super() as per the docs.



```
1
class A(object):
2
def __init__(self):
3
    print "world"
4
5
class B(A):
6
def __init__(self):
7
    print "hello"
8
super(B, self).__init__()
```

Python Functions Coding Interview Questions

- 61) How to get a function name as a string in Python?

Python

```
1 def foo():
```

```
2     pass
```

```
4 >>> name_of(foo)
```

```
5 "foo"
```

Answer#

my_function.__name__

Using __name__ is the preferred method as it applies uniformly. Unlike func_name, it works on built-in functions as well:

Python

```
1 >>> import time
```

```
2 >>> time.time.func_name
```

```
3 Traceback (most recent call last):
```

```
4   File "<stdin>", line 1, in ?
```

```
5 AttributeError: 'builtin_function_or_method' object has no attribute 'func_name'
```

```
6 >>> time.time.__name__
```

```
7 'time'
```

Also, the double underscores indicate to the reader this is a special attribute. As a bonus, classes and modules have a `__name__` attribute too, so you only have to remember one special name.

62) What is the naming convention in Python for variables and functions?

Answer#

Function names should be lowercase, with words separated by underscores as necessary to improve readability.

Variable names follow the same convention as function names.

`mixedCase` is allowed only in contexts where that's already the prevailing style (e.g. `threading.py`), to retain backwards compatibility.

The Google Python Style Guide has the following convention:

`module_name`, `package_name`, `ClassName`, `method_name`, `ExceptionName`,
`function_name`, `GLOBAL_CONSTANT_NAME`, `global_var_name`,
`instance_var_name`, `function_parameter_name`, `local_var_name`.

A similar naming scheme should be applied to a `CLASS_CONSTANT_NAME`

63) How are Python lambdas useful?

Answer#

Lambda Expressions:

```
lambda x: x*2 + 2x - 5
```

Those things are actually quite useful. Python supports a style of programming called functional programming where you can pass functions to other functions to do stuff. Example:

```
mult3 = filter(lambda x: x % 3 == 0, [1, 2, 3, 4, 5, 6, 7, 8, 9])
```

sets `mult3` to `[3, 6, 9]`, those elements of the original list that are multiples of 3. This is shorter (and, one could argue, clearer) than

Python

```
def filterfunc(x):  
    return x % 3 == 0  
  
mult3 = filter(filterfunc, [1, 2, 3, 4, 5, 6, 7, 8, 9])
```

Of course, in this particular case, you could do the same thing as a list comprehension:

```
mult3 = [x for x in [1, 2, 3, 4, 5, 6, 7, 8, 9] if x % 3 == 0]
```

(or even as `range(3,10,3)`), but there are many other, more sophisticated use cases where you can't use a list comprehension and a lambda function may be the shortest way to write something out.

Returning a function from another function

Python

```
def transform(n):
```

```
...     return lambda x: x + n
```

```
...
```

```
f = transform(3)
```

```
f(4)
```

```
7
```

This is often used to create function wrappers, such as Python's decorators.

Combining elements of an iterable sequence with `reduce()`

```
reduce(lambda a, b: '{}, {}'.format(a, b), [1, 2, 3, 4, 5, 6, 7, 8, 9])  
'1, 2, 3, 4, 5, 6, 7, 8, 9'
```

Sorting by an alternate key

```
sorted([1, 2, 3, 4, 5, 6, 7, 8, 9], key=lambda x: abs(5-x))  
[5, 4, 6, 3, 7, 2, 8, 1, 9]
```

I use lambda functions on a regular basis. It took me a while to get used to them, but eventually, I came to understand that they're a very valuable part of the language.

lambda is just a fancy way of saying function. Other than its name, there is nothing obscure, intimidating, or cryptic about it. When you read the following line, replace lambda with function in your mind:

```
f = lambda x: x + 1  
f(3)  
4
```

It just defines a function of x. Some other languages, like R, say it explicitly:

```
f = function(x) { x + 1 }  
f(3)  
4
```

Do you see? It's one of the most natural things to do in programming.

64) How do I call a function from another .py file?

file.py contains a function named function. How do I import it?

```
from file.py import function(a,b)
```

The above gives an error:

```
ImportError: No module named 'file.py'; file is not a package
```

Answer#

First, import function from file.py:

from the file import function

Later, call the function using:

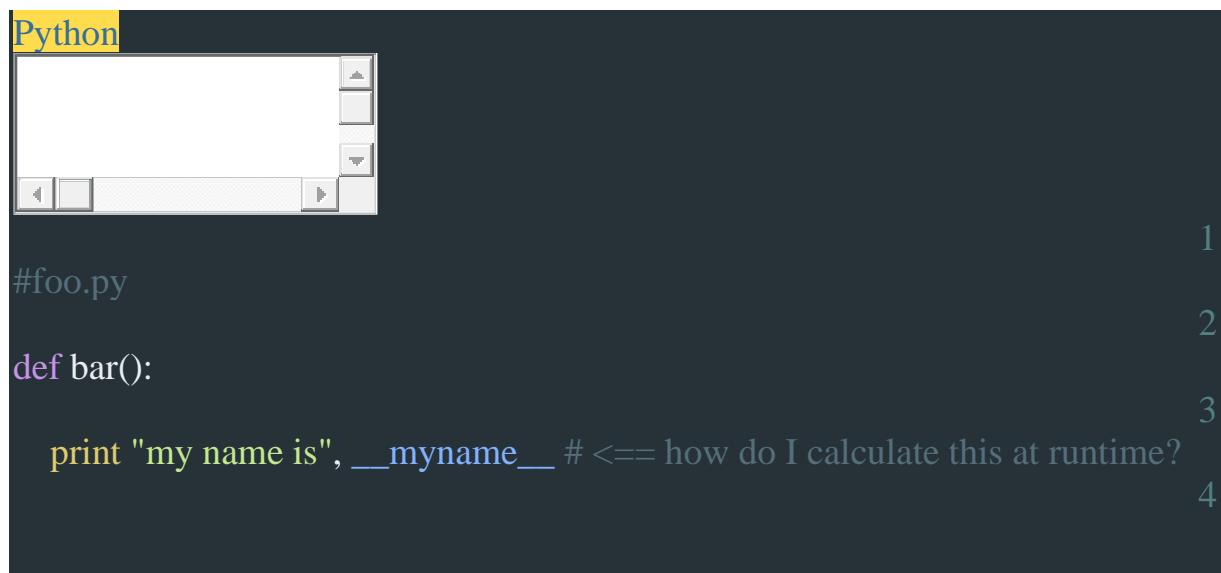
```
function(a, b)
```

Note that if you're trying to import functions from a.py to a file called b.py, you will need to make sure that a.py and b.py are in the same directory.

65) How to determine function name from within that function (without using traceback) in Python?

In Python, without using the traceback module, is there a way to determine a function's name from within that function?

Say I have a module foo with a function bar. When executing foo.bar(), is there a way for bar to know bar's name? Or better yet, foo.bar's name?



```
Python
#foo.py
def bar():
    print "my name is", __myname__ # <== how do I calculate this at runtime?
```

Answer#



```
Python
import inspect
```

```

def foo():
    print(inspect.stack()[0][3])
    print(inspect.stack()[1][3]) # will give the caller of foos name, if something
    called foo
foo()
#output:
foo
<module_caller_of_foo>

```

66) Why do some functions have underscores “ ” before and after the function name in Python?

Answer#

Descriptive: Naming Styles in Python

The following special forms using leading or trailing underscores are recognized (these can generally be combined with any case convention):

_single_leading_underscore: weak “internal use” indicator. E.g. from M import * does not import objects whose name starts with an underscore.

single_trailing_underscore_: used by convention to avoid conflicts with Python keyword, e.g.

Tkinter.Toplevel(master, class_=’ClassName’)

_double_leading_underscore: when naming a class attribute, invokes name mangling (inside class FooBar, __boo becomes _FooBar__boo; see below).

__double_leading_and_trailing_underscore__: “magic” objects or attributes that live in user-controlled namespaces. E.g. **init**, **import** or **file**. Never invent such names; only use them as documented.

Note that names with double leading and trailing underscores are essentially reserved for Python itself: “Never invent such names; only use them as documented”.

67) How to run a function from the command line in Python?

I have this code:

```
def hello():
    return 'Hi :)'
```

How would I run this directly from the command line?

Answer#

With the -c (command) argument (assuming your file is named foo.py):

```
$ python -c 'import foo; print foo.hello()'
```

Alternatively, if you don't care about namespace pollution:

```
$ python -c 'from foo import *; print hello()'
```

And the middle ground:

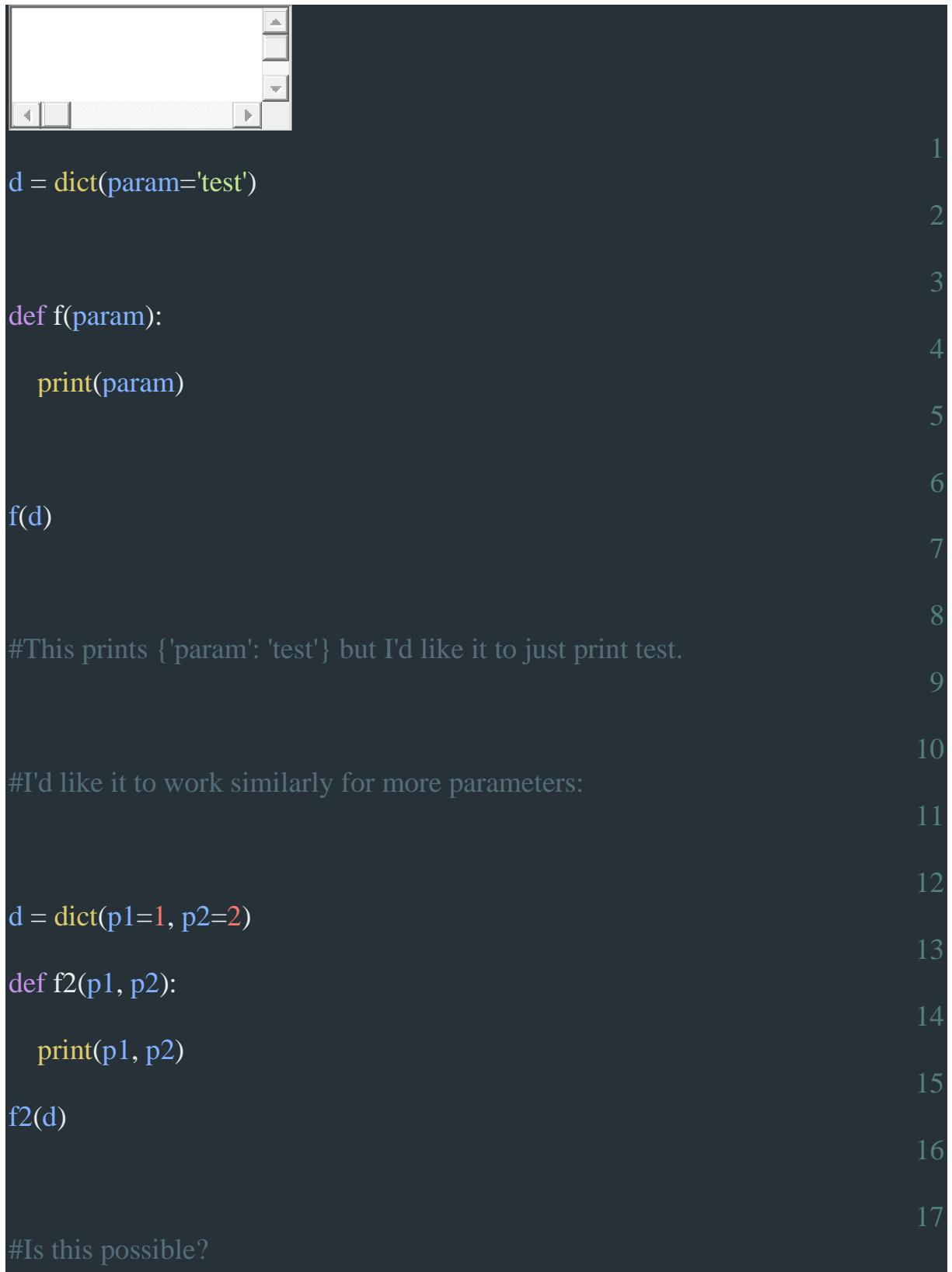
```
$ python -c 'from foo import hello; print hello()'
```

68) How to pass a dictionary to a function as keyword parameters in Python?

I'd like to call a function in python using a dictionary with matching key-value pairs for the parameters.

Here is some code:

Python



```
1
2
3
4
5
6
7
8
9
10
11
12
13
14
15
16
17
```

```
d = dict(param='test')
def f(param):
    print(param)
f(d)
#This prints {'param': 'test'} but I'd like it to just print test.

#I'd like it to work similarly for more parameters:
d = dict(p1=1, p2=2)
def f2(p1, p2):
    print(p1, p2)
f2(d)
#Is this possible?
```

Answer#

Just include the `**` operator to unpack the dictionary. This solves the issue of passing a dictionary to a function as a keyword parameter.

So here is an example:

```
Python
d = dict(p1=1, p2=2)
def f2(p1,p2):
    print p1, p2
f2(**d)
```

69) Is there a math nCr function in Python?

Answer# Python code for nCr (n Choose r) function:

The following program calculates nCr in an efficient manner (compared to calculating factorials etc.)

```
Python
import operator as op
from functools import reduce
def ncr(n, r):
    r = min(r, n-r)
    numer = reduce(op.mul, range(n, n-r, -1), 1)
    denom = reduce(op.mul, range(1, r+1), 1)
    return numer // denom # or / in Python 2
```

```
9  
10  
11  
#As of Python 3.8, binomial coefficients are available in the standard library as  
math.comb:  
12  
13  
14  
>>> from math import comb  
15  
>>> comb(10,3)  
120
```

70) How to send an email with Python? Python code for sending an email?

Answer#: **Python program for sending an email.**

We recommend using the standard packages `email` and `smtplib` together to send emails.

Please look at the following example (reproduced from the Python documentation).

Notice that if you follow this approach, the “simple” task is indeed simple, and the more complex tasks (like attaching binary objects or sending plain/HTML multipart messages) are accomplished very rapidly.

```
1  
2  
3  
4  
Python  
# Import smtplib for the actual sending function  
import smtplib  
# Import the email modules we'll need
```

```
5
from email.mime.text import MIMEText
6
7
# Open a plain text file for reading. For this example, assume that
8
# the text file contains only ASCII characters.
9
with open(textfile, 'rb') as fp:
10
    # Create a text/plain message
11
    msg = MIMEText(fp.read())
12
13
# me == the sender's email address
14
# you == the recipient's email address
15
msg['Subject'] = 'The contents of %s' % textfile
16
msg['From'] = me
17
msg['To'] = you
18
19
# Send the message via our own SMTP server, but don't include the
20
# envelope header.
21
s = smtplib.SMTP('localhost')
22
s.sendmail(me, [you], msg.as_string())
23
s.quit()
24
```

For sending emails to multiple destinations, you can follow the below Python example:

Python

```
1 # Import smtplib for the actual sending function
2 import smtplib
3
4 # Here are the email package modules we'll need
5 from email.mime.image import MIMEImage
6 from email.mime.multipart import MIMEMultipart
7
8 # Create the container (outer) email message.
9 msg = MIMEMultipart()
10 msg['Subject'] = 'Our family reunion'
11 # me == the sender's email address
12 # family = the list of all recipients' email addresses
13 msg['From'] = me
14 msg['To'] = ', '.join(family)
15 msg.preamble = 'Our family reunion'
16
17 # Assume we know that the image files are all in PNG format
18 for file in pngfiles:
19     # Open the files in binary mode. Let the MIMEImage class automatically
```

```

# guess the specific image type.          20
21
with open(file, 'rb') as fp:
22
    img = MIMEImage(fp.read())
23
msg.attach(img)
24
25
# Send the email via our own SMTP server.
26
s = smtplib.SMTP('localhost')
27
s.sendmail(me, family, msg.as_string())
28
s.quit()

```

As you can see, the header To in the MIMEText object must be a string consisting of email addresses separated by commas. On the other hand, the second argument to the sendmail function must be a list of strings (each string is an email address).

So, if you have three email addresses: person1@example.com, person2@example.com, and person3@example.com, you can do as follows (obvious sections omitted):

```

to = ["person1@example.com", "person2@example.com",
      "person3@example.com"]
msg['To'] = ",".join(to)
s.sendmail(me, to, msg.as_string())

```

the “,”.join(to) part makes a single string out of the list, separated by commas.

Python List/Loops Coding Interview Questions & Answers

71) What is the output of the below Python lists program?

Python

```
1
2
3
4
```

```
xs = [8, 23, 45]
for x in xs:
    print("item #{ } = { }".format(index, x))
```

Answer#:

NameError: name ‘index’ is not defined

72) How do I access the index while iterating over a sequence with a for loop in Python?

Answer#:

```
1
2
3
```

```
Python
xs = [8, 23, 45]
for idx, x in enumerate(xs):
    print(idx, x)
```

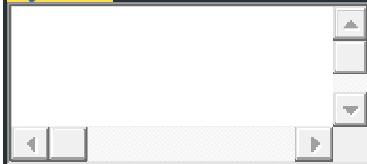
Output:

```
0 8
1 23
2 45
```

73) How to traverse a list in reverse order in Python?

Answer#:

Python



#Use the built-in reversed() function:

>>> a = ["foo", "bar", "baz"]

>>> for i in reversed(a):

... print(i)

...

baz

bar

foo

#To also access the original index, use enumerate() on your list before passing it to reversed():

>>> for i, e in reversed(list(enumerate(a))):

... print(i, e)

...

2 baz

1 bar

0 foo

Since enumerate() returns a generator and generators can't be reversed, you need to convert it to a list first.

74) What is the difference between range and xrange functions in Python?

Answer#:

In Python 2.x:

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

xrange is a sequence object that evaluates lazily.

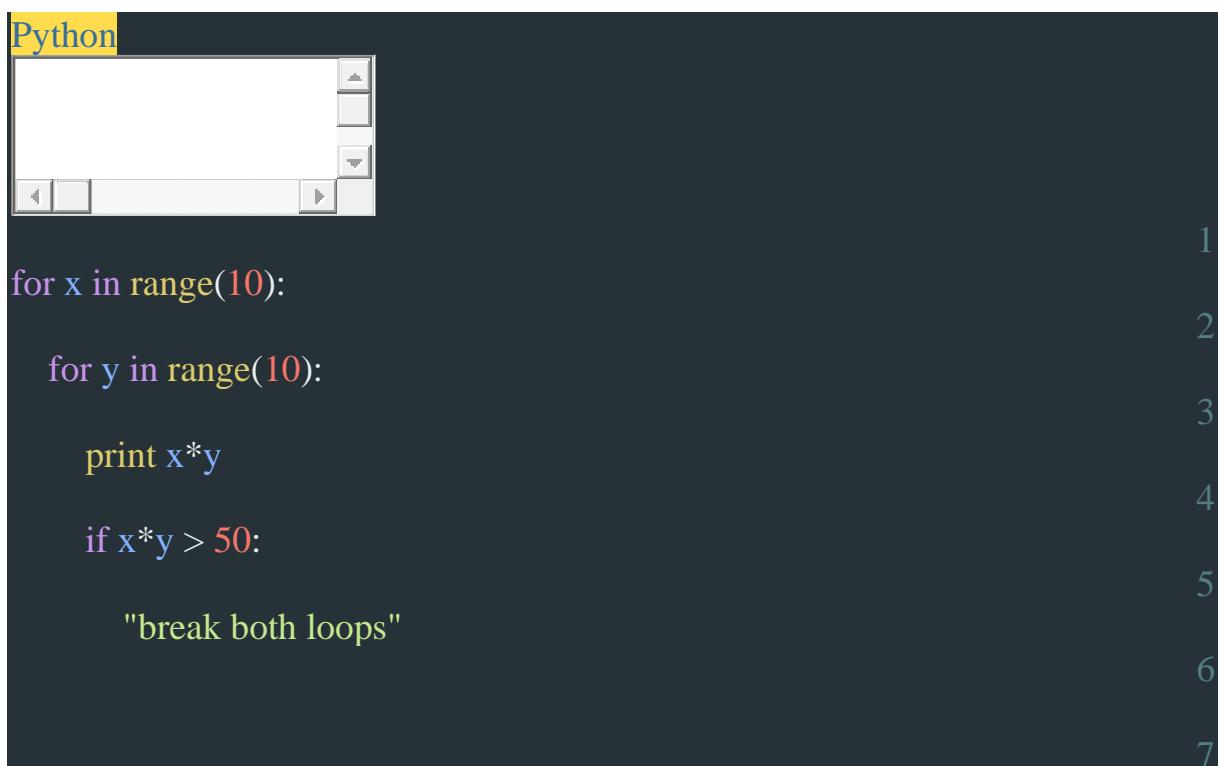
In Python 3:

range does the equivalent of Python 2's xrange. To get the list, you have to explicitly use list(range(...)).

xrange no longer exists.

75) How can I break out of multiple loops/nested-loops?

Is there an easier way to break out of nested loops than throwing an exception?



The screenshot shows a Python code editor window with the word "Python" in the tab bar. The code area contains the following Python script:

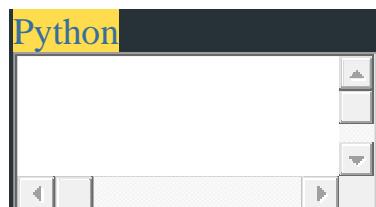
```
for x in range(10):
    for y in range(10):
        print x*y
        if x*y > 50:
            "break both loops"
```

The code is numbered from 1 to 7 on the right side. The interface includes a toolbar with icons for file operations, a status bar at the bottom, and scroll bars on the right and bottom.

```
8  
#I.e., is there a nicer way than:  
9  
10  
11 class BreakIt(Exception): pass  
12  
13 try:  
14     for x in range(10):  
15         for y in range(10):  
16             print x*y  
17             if x*y > 50:  
18                 raise BreakIt  
19 except BreakIt:  
20     pass
```

Answer# Here is the code for breaking the nested loops

Python



```
1  
for x in xrange(10):  
2     for y in xrange(10):  
3         print x*y  
4         if x*y > 50:
```

```
5
break
6
else:
7    continue # only executed if the inner loop did NOT break
8
break # only executed if the inner loop DID break
9
10
#The same works for deeper loops:
11
12
for x in xrange(10):
13
    for y in xrange(10):
14
        for z in xrange(10):
15
            print x,y,z
16
            if x*y*z == 30:
17
                break
18
            else:
19
                continue
20
                break
21
            else:
22
                continue
23
                break
```

76) How to loop backward using indices in Python?

I am trying to loop from 100 to 0. How do I do this in Python?

for i in range (100,0), It doesn't work. How to solve this?

Answer# 1

Try range(100,-1,-1), the 3rd argument being the increment to use.

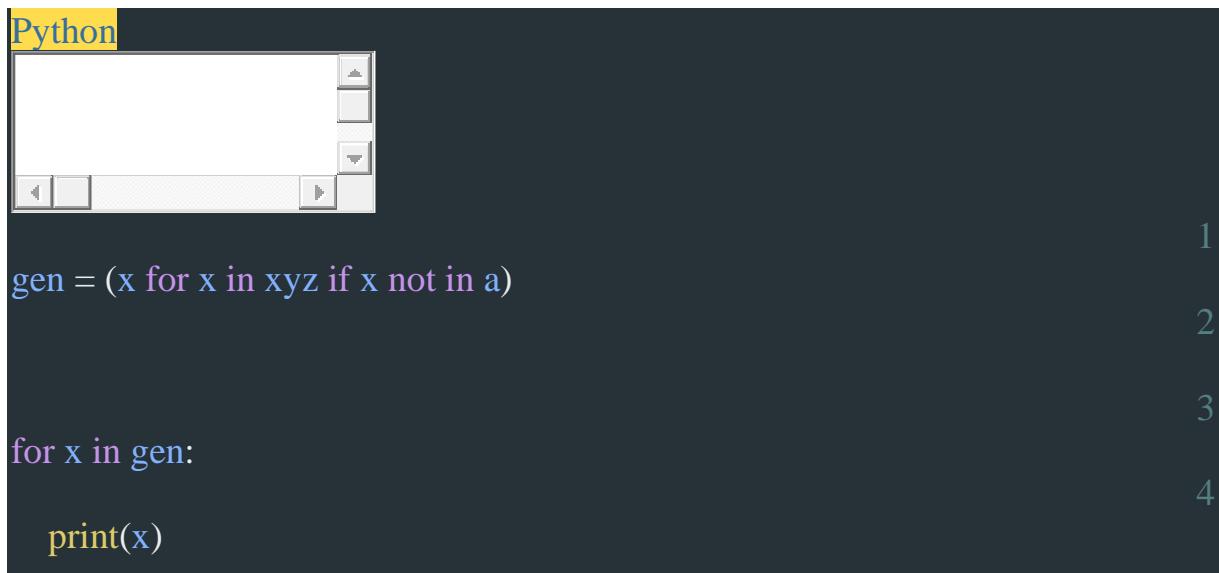
Answer# 2

```
for i in reversed(xrange(101)):  
    print i,
```

77) Is there any way in Python to combine for-loop and if-statement?

Answer#

Using generator expressions we can combine for-loop and if-statements in Python:



```
Python  
1  
gen = (x for x in xyz if x not in a)  
2  
3  
for x in gen:  
4  
    print(x)
```

78) How do I loop through a list by two items at a time in Python?

Answer#

You can use a range with a step size of 2:



```
Python  
1  
2  
3  
4
```

```

1 #Python 2
2
3
4 for i in xrange(0,10,2):
5     print(i)
6
7
8 #Python 3
9
10 for i in range(0,10,2):
    print(i)

```

Note: Use xrange in Python 2 instead of range because it is more efficient as it generates an iterable object, and not the whole list.

79) How do you create different variable names while in a loop in Python?

For example purposes...

```
for x in range(0,9):
    string'x' = "Hello"
```

So I end up with string1, string2, string3... all equaling "Hello"

Answer#

Sure you can; it's called a dictionary:



```
1 d = {}  
2  
3 for x in range(1, 10):  
4     d["string{0}".format(x)] = "Hello"  
5  
6  
7 >>> d["string5"]  
8  
9 'Hello'  
10  
11  
12 >>> d  
13  
14 {'string1': 'Hello',  
15  
16 'string2': 'Hello',  
17  
18 'string3': 'Hello',  
19  
20 'string4': 'Hello',  
21  
22 'string5': 'Hello',  
23  
24 'string6': 'Hello',  
25  
26 'string7': 'Hello',  
27  
28 'string8': 'Hello',  
29  
30 'string9': 'Hello'}
```

I said this somewhat tongue in check, but really the best way to associate one value with another value is a dictionary. That is what it was designed for!

80) Is there a “do … until” loop in Python?

Is there a

do until x:

…

in Python, or a nice way to implement such a looping construct?

Answer#

There is no do-while loop in Python.

This is a similar construct, taken from the link above.

```
Python
while True:
    do_something()
    if condition():
        break
1
2
3
4
```

81) How to reverse tuples in Python?

Answer#

There are two idiomatic ways to do this:

`reversed(x)` # returns an iterator
or

`x[::-1]` # returns a new tuple

Since tuples are immutable, there is no way to reverse a tuple in-place.

The iterator returned by `reversed` would be equivalent to

```
Python
def myreversed(seq):
1
2
```

```
for i in range(len(x) - 1, -1, -1):  
    yield seq[i]
```

3

i.e. it relies on the sequence having a known length to avoid having to actually reverse the tuple.

As to which is more efficient, I'd suspect it'd be the seq[::-1] if you are using all of it and the tuple is small and reversed when the tuple is large, but performance in python is often surprising so measure it!

82) How do I concatenate two lists in Python?

Answer#

Use the + operator to combine the lists:

```
Python  
listone = [1, 2, 3]  
listtwo = [4, 5, 6]  
joinedlist = listone + listtwo  
#Output:  
>>> joinedlist  
[1, 2, 3, 4, 5, 6]
```

1

2

3

4

5

6

7

8

9

83) How do I check if a list is empty?

For example, if passed the following:

```
a = []
```

How do I check to see if a is empty?

Answer#

```
if not a:  
    print("List is empty")
```

Using the implicit booleanness of the empty list is quite Pythonic.

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

Answer#

append appends a specified object at the end of the list:

The screenshot shows a terminal window titled "Python". It contains the following code examples:

```
Python  
>>> x = [1, 2, 3]  
>>> x.append([4, 5])  
>>> print(x)  
[1, 2, 3, [4, 5]]  
  
#extend extends the list by appending elements from the specified iterable:  
>>> x = [1, 2, 3]  
>>> x.extend([4, 5])
```

The code is numbered 1 through 10 on the right side of the terminal window.

```
>>> print(x)
```

11

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

85) How do I get the last element of a list in Python?

Answer#

some_list[-1] is the shortest and most Pythonic.

In fact, you can do much more with this syntax. The some_list[-n] syntax gets the nth-to-last element. So some_list[-1] gets the last element, some_list[-2] gets the second to last, etc, all the way down to some_list[-len(some_list)], which gives you the first element.

You can also set list elements in this way. For instance:

Python

```
>>> some_list = [1, 2, 3]
```

1

```
>>> some_list[-1] = 5 # Set the last element
```

2

```
>>> some_list[-2] = 3 # Set the second to last element
```

3

```
>>> some_list
```

4

```
[1, 3, 5]
```

5

Note that getting a list item by index will raise an IndexError if the expected item doesn't exist. This means that some_list[-1] will raise an exception if some_list is empty, because an empty list can't have a last element.

86) How do I sort a list of dictionaries by a value of the dictionary in Python?

How to sort a list of dictionaries by a specific key's value? Given:

```
[{'name': 'Homer', 'age': 39}, {'name': 'Bart', 'age': 10}]
```

When sorted by name, it should become:

```
[{'name': 'Bart', 'age': 10}, {'name': 'Homer', 'age': 39}]
```

Answer#

The sorted() function takes a key= parameter

```
newlist = sorted(list_to_be_sorted, key=lambda d: d['name'])
```

Alternatively, you can use operator.itemgetter instead of defining the function yourself

```
from operator import itemgetter  
newlist = sorted(list_to_be_sorted, key=itemgetter('name'))
```

For completeness, add reverse=True to sort in descending order

```
newlist = sorted(list_to_be_sorted, key=itemgetter('name'), reverse=True)
```

87) How can I randomly select an item from a list?

How do I retrieve an item at random from the following list?

```
foo = ['a', 'b', 'c', 'd', 'e']
```

Answer#

The screenshot shows a Python code editor window. At the top left, the word "Python" is highlighted in yellow. Below the title bar is a toolbar with several icons. The main area contains the following code and a question:

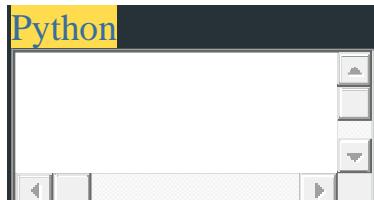
```
1  
2  
3  
4  
5  
Use random.choice():  
import random
```

The numbers 1 through 5 are aligned vertically to the right of the code lines. The question "Use random.choice():" is preceded by a question mark.

```
foo = ['a', 'b', 'c', 'd', 'e']
print(random.choice(foo))
```

6

For cryptographically secure random choices (e.g., for generating a passphrase from a wordlist), use secrets.choice():



```
import secrets
foo = ['battery', 'correct', 'horse', 'staple']
print(secrets.choice(foo))
```

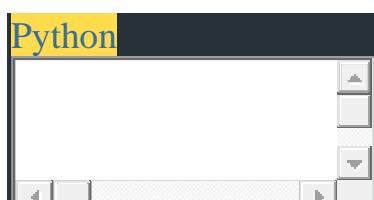
1

2

3

4

secrets is new in Python 3.6. On older versions of Python you can use the random.SystemRandom class:



```
import random
secure_random = random.SystemRandom()
print(secure_random.choice(foo))
```

1

2

3

4

5

88) How do I get the number of elements in a list in Python?

How do I get the number of elements in the list items?

```
items = ["apple", "orange", "banana"]
```

Answer#

The len() function can be used with several different types in Python – both built-in types and library types.

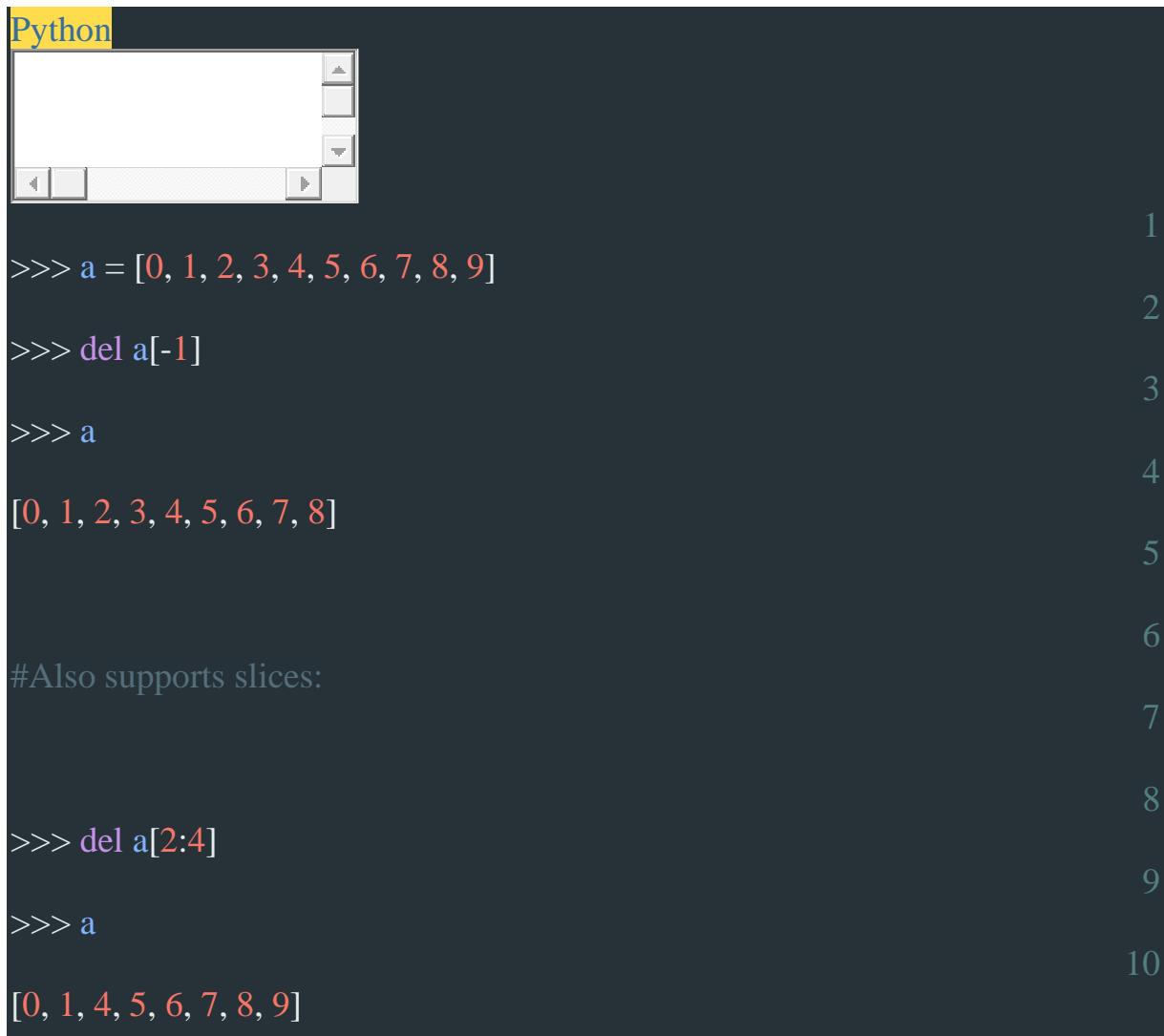
For example:

```
len([1, 2, 3])  
3
```

89) How to remove an element from a list by index in Python?

Answer#

Use del and specify the index of the element you want to delete:



A screenshot of a Python IDLE window. The title bar says "Python". The main area shows the following code execution:

```
>>> a = [0, 1, 2, 3, 4, 5, 6, 7, 8, 9]  
>>> del a[-1]  
>>> a  
[0, 1, 2, 3, 4, 5, 6, 7, 8]  
  
#Also supports slices:  
>>> del a[2:4]  
>>> a  
[0, 1, 4, 5, 6, 7, 8, 9]
```

The code is numbered 1 through 10 on the right side of the lines.

90) How do I count the occurrences of a list item in Python?

Answer#

If you only want a single item's count, use the count method:

```
[1, 2, 3, 4, 1, 4, 1].count(1)  
3
```

Important: this is very slow if you are counting multiple different items

Each count call goes over the entire list of n elements. Calling count in a loop n times means $n * n$ total checks, which can be catastrophic for performance.

If you want to count multiple items, use Counter, which only does n total checks.

```
Python  
>>> from collections import Counter  
>>> z = ['blue', 'red', 'blue', 'yellow', 'blue', 'red']  
>>> Counter(z)  
Counter({'blue': 3, 'red': 2, 'yellow': 1})
```

91) Why is it string.join(list) instead of list.join(string)?

Explain the below code:

Why not

```
["Hello", "world"].join("-")
```

Why this:

```
"-".join(["Hello", "world"])
```

Is there a specific reason it is like this?

Answer#

It's because any iterable can be joined (e.g, list, tuple, dict, set), but its contents and the “joiner” must be strings.

For example:

```
'join(['welcome', 'to', 'python', 'programming'])' "join('welcome', 'to',  
'python', 'programming'))'
```

```
'welcome_to_python_programming'
```

Using something other than strings will raise the following error:

TypeError: sequence item 0: expected str instance, int found

Python Arrays Coding Interview Questions

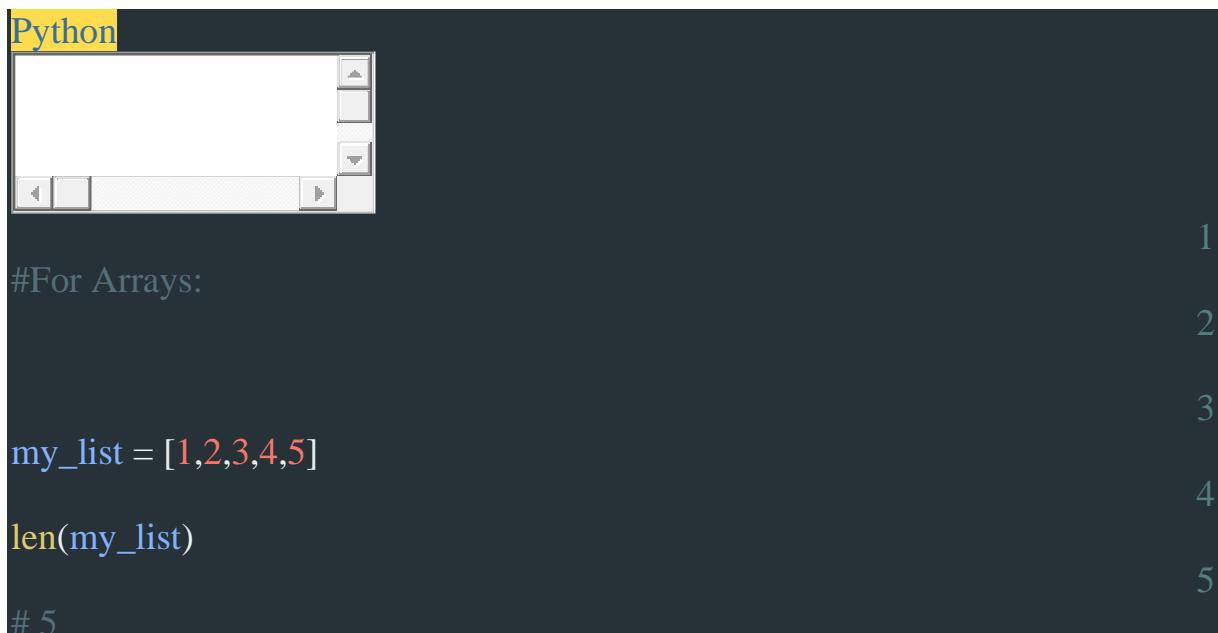
92) Is arr.__len__() the preferred way to get the length of an array in Python?

In Python, is the following the only way to get the number of elements?

arr.__len__()

If so, why the strange syntax?

Answer#



The screenshot shows a Python code editor window titled "Python". The code is as follows:

```
#For Arrays:  
my_list = [1,2,3,4,5]  
len(my_list)  
# 5
```

The code is numbered 1 through 5 on the right side. The output of the len() function call is shown as "# 5".

```
6  
7  
#The same works for tuples:  
8  
9  
my_tuple = (1,2,3,4,5)  
10  
len(my_tuple)  
11  
# 5  
12  
13  
#And strings, which are really just arrays of characters:  
14  
15  
my_string = 'hello world'  
16  
len(my_string)  
17  
# 11
```

It was intentionally done this way so that lists, tuples and other container types or iterables didn't all need to explicitly implement a public `.length()` method, instead you can just check the `len()` of anything that implements the 'magic' `__len__()` method.

93) How to check if multiple strings exist in another string in Python?

Answer#

You can use any:

```
Python  
1  
a_string = "A string is more than its parts!"
```

```
2 matches = ["more", "wholesome", "milk"]
3
4 if any(x in a_string for x in matches):
5
6 #Similarly to check if all the strings from the list are found, use all instead of
any.
```

94) Python list vs. array – when to use?

If you are creating a 1d array, you can implement it as a list, or else use the ‘array’ module in the standard library. I have always used lists for 1d arrays.

What is the reason or circumstance where I would want to use the array module instead?

Is it for performance and memory optimization, or am I missing something obvious?

Answer#

Basically, Python lists are very flexible and can hold completely heterogeneous, arbitrary data, and they can be appended to very efficiently, in amortized constant time. If you need to shrink and grow your list time-efficiently and without hassle, they are the way to go.

But they use a lot more space than C arrays, in part because each item in the list requires the construction of an individual Python object, even for data that could be represented with simple C types (e.g. float or uint64_t).

The `array.array` type, on the other hand, is just a thin wrapper on C arrays. It can hold only homogeneous data (that is to say, all of the same type) and so it uses only `sizeof(one object) * length` bytes of memory. Mostly, you should use it when you need to expose a C array to an extension or a system call (for example, `ioctl` or `fctnl`).

`array.array` is also a reasonable way to represent a mutable string in Python 2.x (`array('B', bytes)`). However, Python 2.6+ and 3.x offer a mutable byte string as `bytearray`.

However, if you want to do math on a homogeneous array of numeric data, then you're much better off using NumPy, which can automatically vectorize operations on complex multi-dimensional arrays.

To make a long story short: `array.array` is useful when you need a homogeneous C array of data for reasons other than doing math.

95) How do I print the full NumPy array, without truncation?

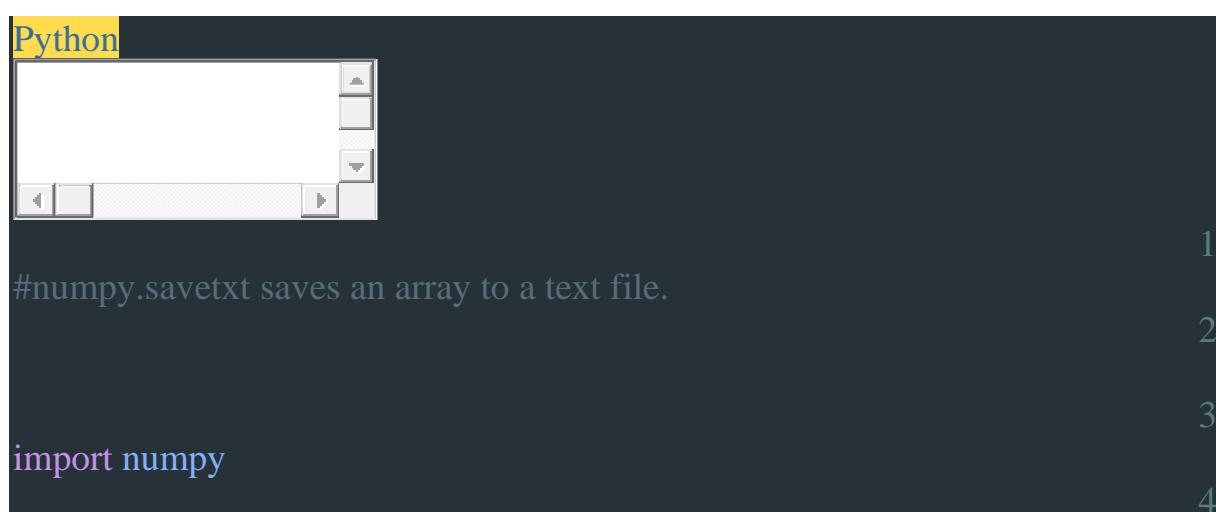
Answer#



```
Python
1 #Use numpy.set_printoptions:
2
3
4 import sys
5
6 import numpy
7
8 numpy.set_printoptions(threshold=sys.maxsize)
```

96) How do I dump a NumPy array into a csv file in a human-readable format?

Answer#



```
Python
1 #numpy.savetxt saves an array to a text file.
2
3
4 import numpy
```

```
a = numpy.asarray([ [1,2,3], [4,5,6], [7,8,9] ])
numpy.savetxt("foo.csv", a, delimiter=",")
```

5

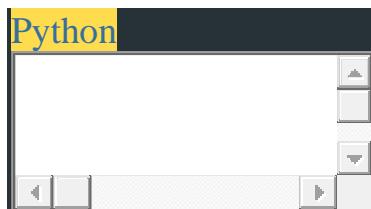
97) How do I access the ith column of a NumPy multidimensional array?

Given:

```
test = numpy.array([[1, 2], [3, 4], [5, 6]])
```

test[i] gives the ith row (e.g. [1, 2]). How do I access the ith column? (e.g. [1, 3, 5]). Also, would this be an expensive operation?

Answer#



#To access column 0:

```
>>> test[:, 0]
```

```
array([1, 3, 5])
```

To access row 0:

```
>>> test[0, :]
```

```
array([1, 2])
```

98) Is there a NumPy function to return the first index of something in an array?

Answer#

Yes, given an array, array, and a value, item to search for, you can use np.where as:

```
itemindex = numpy.where(array == item)
```

The result is a tuple with first all the row indices, then all the column indices.

For example, if an array is two dimensions and it contained your item at two locations then

```
Python
array[itemindex[0][0]][itemindex[1][0]]  
#would be equal to your item and so would be:  
array[itemindex[0][1]][itemindex[1][1]]
```

99) What are the advantages of NumPy over regular Python lists?

Answer#

NumPy's arrays are more compact than Python lists — a list of lists as you describe, in Python, would take at least 20 MB or so, while a NumPy 3D array with single-precision floats in the cells would fit in 4 MB. Access in reading and writing items is also faster with NumPy.

Maybe you don't care that much for just a million cells, but you definitely would for a billion cells — neither approach would fit in a 32-bit architecture, but with 64-bit builds NumPy would get away with 4 GB or so, Python alone would need at least about 12 GB (lots of pointers which double in size) — a much costlier piece of hardware!

The difference is mostly due to “indirectness” — a Python list is an array of pointers to Python objects, at least 4 bytes per pointer plus 16 bytes for even the

smallest Python object (4 for type pointer, 4 for reference count, 4 for value — and the memory allocators rounds up to 16).

A NumPy array is an array of uniform values — single-precision numbers takes 4 bytes each, double-precision ones, 8 bytes. Less flexible, but you pay substantially for the flexibility of standard Python lists!

100) Explain sorting arrays in NumPy by column?

```
Python
1
#For example, given:
2
3
4
a = array([[9, 2, 3],
5
[4, 5, 6],
6
[7, 0, 5]])
7
#I want to sort the rows of a by the second column to obtain:
8
9
array([[7, 0, 5],
10
[9, 2, 3],
11
[4, 5, 6]])
```

Answer#

To sort by the second column of a:

```
a[a[:, 1].argsort()]
```

101) How to get numpy array dimensions?

How do I get the dimensions of an array? For instance, this is 2×2 :

```
a = np.array([[1,2],[3,4]])
```

Answer#

Use `.shape` to obtain a tuple of array dimensions:

```
a.shape  
(2, 2)
```

102) What is the difference between `np.array()` and `np.asarray()`?

What is the difference between NumPy's `np.array` and `np.asarray`? When should I use one rather than the other? They seem to generate identical output.

Answer#

The definition of `asarray` is:

```
def asarray(a, dtype=None, order=None):  
    return array(a, dtype, copy=False, order=order)
```

So it is like `array`, except it has fewer options, and `copy=False`. `array` has `copy=True` by default.

The main difference is that `array` (by default) will make a copy of the object, while `asarray` will not unless necessary.

103) How do I concatenate two one-dimensional arrays in NumPy?



A screenshot of a Jupyter Notebook cell. The cell title is "Python". The code in the cell is:

```
import numpy as np  
a = np.array([1, 2, 3])
```

The cell has been run twice, indicated by the numbers 1 and 2 at the bottom right of the cell area.

3

```
b = np.array([4, 5])  
np.concatenate(a, b)
```

4

Answer#

Use:

```
np.concatenate([a, b])
```

The arrays you want to concatenate need to be passed in as a sequence, not as separate arguments.

From the NumPy documentation:

```
numpy.concatenate((a1, a2, ...), axis=0)
```

Join a sequence of arrays together.

It was trying to interpret your b as the axis parameter, which is why it complained it couldn't convert it into a scalar.

104) What is the difference between ndarray and array in NumPy?

Where is their implementation in the NumPy source code?

Answer#

numpy.array is just a convenience function to create an ndarray; it is not a class itself.

You can also create an array using numpy.ndarray, but it is not the recommended way. From the docstring of numpy.ndarray:

Arrays should be constructed using array, zeros or empty ... The parameters given here refer to a low-level method (ndarray(...)) for instantiating an array.

105) How does np.einsum work?

Given arrays A and B, their matrix multiplication followed by transpose is computed using (A @ B).T, or equivalently, using:

```
np.einsum("ij, jk -> ki", A, B)
```

Answer#

What does einsum do?

Imagine that we have two multi-dimensional arrays, A and B. Now let's suppose we want to...

multiply A with B in a particular way to create new array of products; and then maybe sum this new array along particular axes; and then maybe transpose the axes of the new array in a particular order.

There's a good chance that einsum will help us do this faster and more memory-efficiently than combinations of the NumPy functions like multiply, sum and transpose will allow.

106) How can I remove some specific elements from a numpy array? Say I have

```
import numpy as np
```

```
a = np.array([1,2,3,4,5,6,7,8,9])
```

I then want to remove 3,4,7 from a. All I know is the index of the values (index=[2,3,6]).

Answer#

Use numpy.delete() – returns a new array with sub-arrays along an axis deleted

```
numpy.delete(a, index)
```

For your specific question:



```
Python
import numpy as np
```

```
2  
3  
4  
5  
6  
7  
8  
a = np.array([1, 2, 3, 4, 5, 6, 7, 8, 9])  
index = [2, 3, 6]  
new_a = np.delete(a, index)  
print(new_a) #Prints `[1, 2, 5, 6, 8, 9]`
```

Note that numpy.delete() returns a new array since array scalars are immutable, similar to strings in Python, so each time a change is made to it, a new object is created.

Python Coding Interview Questions And Answers For Experienced

107) How do you debug a Python program?

Answer) By using this command we can debug a [python](#) program

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

108) What is <Yield> Keyword in Python?

A) The <yield> keyword in Python can turn any function into a generator. Yields work like a standard return keyword.

But it'll always return a generator object. Also, a function can have multiple calls to the <yield> keyword.

Example:

```
Python
```

```
def testgen(index):  
    weekdays = ['sun','mon','tue','wed','thu','fri','sat']  
    yield weekdays[index]  
    yield weekdays[index+1]  
day = testgen(0)  
print next(day), next(day)
```

Output: sun mon

109) How to convert a list into a string?

A) When we want to convert a list into a string, we can use the <".join()> method which joins all the elements into one and returns as a string.

Example:

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

110) How to convert a list into a tuple?

A) By using Python <tuple()> function we can convert a list into a tuple. But we can't change the list after turning it into tuple, because it becomes immutable.

Example:

Python

```
1 weekdays = ['sun','mon','tue','wed','thu','fri','sat']  
2  
listAsTuple = tuple(weekdays)  
3  
print(listAsTuple)
```

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

111) How to convert a list into a set?

A) User can convert list into set by using <set()> function.

Example:

Python

```
1 weekdays = ['sun','mon','tue','wed','thu','fri','sat','sun','tue']  
2  
listAsSet = set(weekdays)  
3  
print(listAsSet)
```

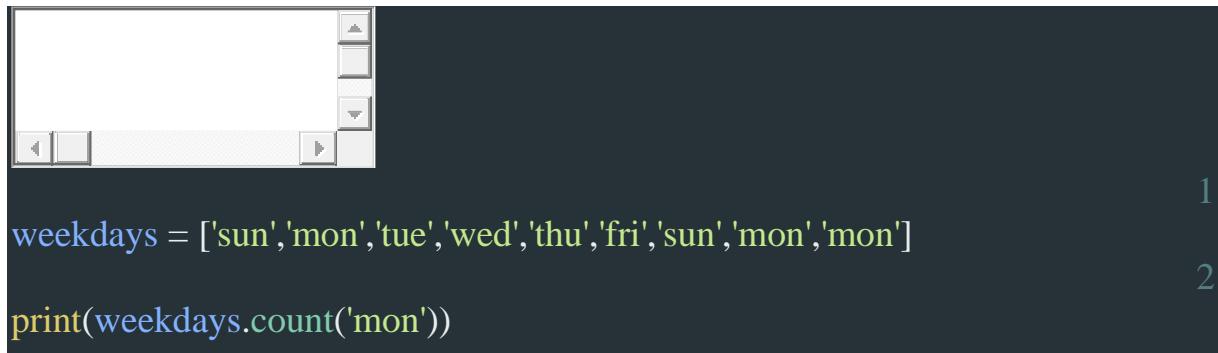
```
output: set(['wed', 'sun', 'thu', 'tue', 'mon', 'fri', 'sat'])
```

112) How to count the occurrences of a particular element in the list?

A) In Python list, we can count the occurrences of an individual element by using a <count()> function.

Example # 1:

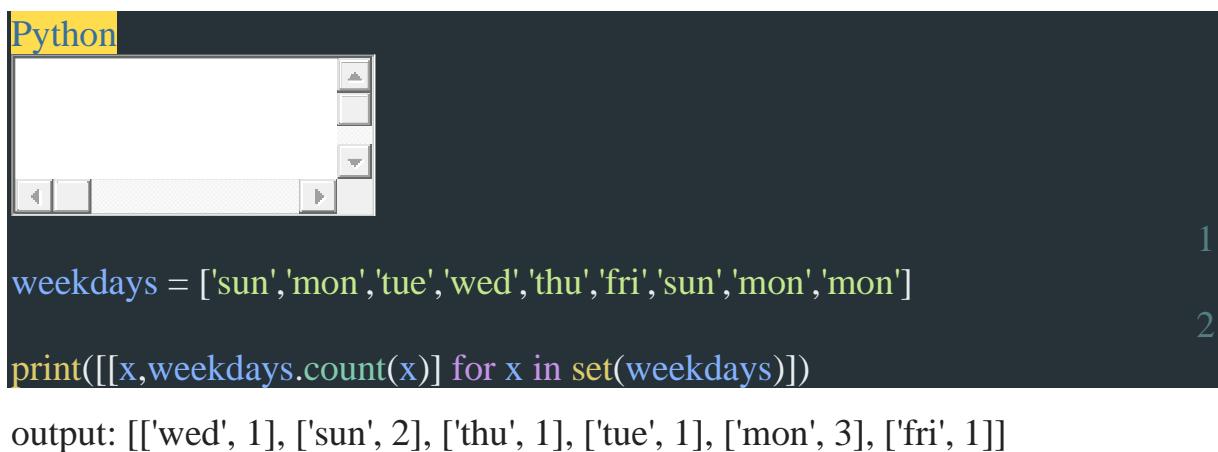
Python



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

Output: 3

Example # 2:



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

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

113) What is NumPy array?

A) NumPy arrays are more flexible than lists in Python. By using NumPy arrays reading and writing items is faster and more efficient.

114) How can you create Empty NumPy Array In Python?

A) We can create Empty NumPy Array in two ways in Python,

1) import numpy
numpy.array([])

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

115) What is a negative index in Python?

A) Python has a special feature like a negative index in Arrays and Lists. Positive index reads the elements from the starting of an array or list but in the negative index, Python reads elements from the end of an array or list.

116) What is the output of the below code?

```
Python
>> import array
>>> a = [1, 2, 3]
>>> print a[-3]
>>> print a[-2]
>>> print a[-1]
```

A) The output is: 3, 2, 1

Advanced Python Coding Interview Questions

117) What is the output of the below program?

```
Python
>>>names = ['Chris', 'Jack', 'John', 'Daman']
>>>print(names[-1][-1])
```

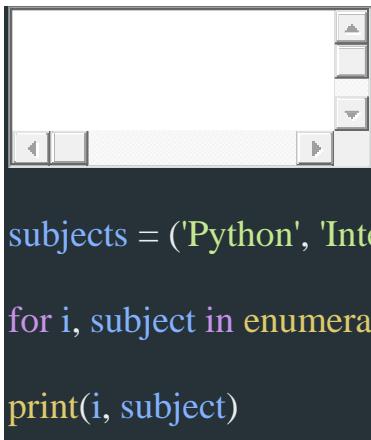
A) The output is: n

118) What is Enumerate() Function in Python?

A) The Python enumerate() function adds a counter to an iterable object. enumerate() function can accept sequential indexes starting from zero.

Python Enumerate Example:

```
Python
```



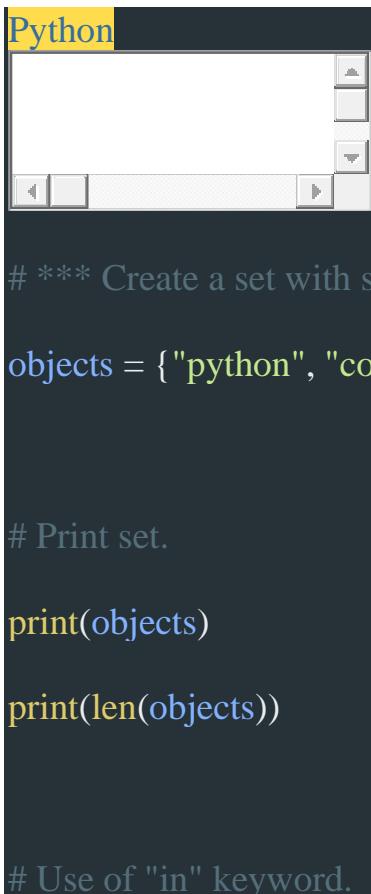
```
1 subjects = ('Python', 'Interview', 'Questions')
2
3 for i, subject in enumerate(subjects):
4     print(i, subject)
```

Output:

```
0 Python
1 Interview
2 Questions
```

119) What is data type SET in Python and how to work with it?

A) The Python data type “set” is a kind of collection. It has been part of Python since version 2.4. A set contains an unordered collection of unique and immutable objects.



```
1 # *** Create a set with strings and perform a search in set
2
3 objects = {"python", "coding", "tips", "for", "beginners"}
4
5
6 # Print set.
7
8 print(objects)
9
10 print(len(objects))
11
12
13 # Use of "in" keyword.
```

```
9
if "tips" in objects:
10
print("These are the best Python coding tips.")
11
12
# Use of "not in" keyword.
13
if "Java tips" not in objects:
14
print("These are the best Python coding tips not Java tips.")
15
16
# ** Output
17
{'python', 'coding', 'tips', 'for', 'beginners'}
18
5
19
These are the best Python coding tips.
20
These are the best Python coding tips not Java tips.
21
22
# *** Lets initialize an empty set
23
items = set()
24
25
# Add three strings.
26
items.add("Python")
27
items.add("coding")
28
items.add("tips")
29
30
```

```
print(items)
```

31

```
# ** Output
```

32

```
{'Python', 'coding', 'tips'}
```

33

120) How do you Concatenate Strings in Python?

A) We can use ‘+’ to concatenate strings.

Python Concatenating Example:



Python

```
# See how to use '+' to concatenate strings.
```

1

```
>>> print('Python' + ' Interview' + ' Questions')
```

2

```
# Output:
```

Python Interview Questions

121) How to generate random numbers in Python?

A) We can generate random numbers using different functions in Python. They are:

#1. random() – This command returns a floating point number, between 0 and 1.

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

#3. randint(X, Y) – This command returns a random integer between the values given as X and Y.

122) How to print the sum of the numbers starting from 1 to 100?

A) We can print sum of the numbers starting from 1 to 100 using this code:

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

In Python the range function does not include the end given. Here it will exclude 101.

Sum function print sum of the elements of range function, i.e 1 to 100.

123) How do you set a global variable inside a function?

A) Yes, we can use a global variable in other functions by declaring it as global in each function that assigns to it:

```
Python
1
globvar = 0
2
def set_globvar_to_one():
3
    global globvar # Needed to modify global copy of globvar
4
    globvar = 1
5
def print_globvar():
6
    print globvar # No need for global declaration to read value of globvar
7
    set_globvar_to_one()
8
    print_globvar() # Prints 1
```

124) What is the output of the program?

Python

```
1 names1 = ['Amir', 'Bear', 'Charlton', 'Daman']
2 names2 = names1
3 names3 = names1[:]
4
5 names2[0] = 'Alice'
6 names3[1] = 'Bob'
7
8 sum = 0
9
10 for ls in (names1, names2, names3):
11     if ls[0] == 'Alice':
12         sum += 1
13     if ls[1] == 'Bob':
14         sum += 10
15
16 print sum
```

A) 12

125) What is the output, Suppose list1 is [1, 3, 2], What is list1 * 2?

A) [1, 3, 2, 1, 3, 2]

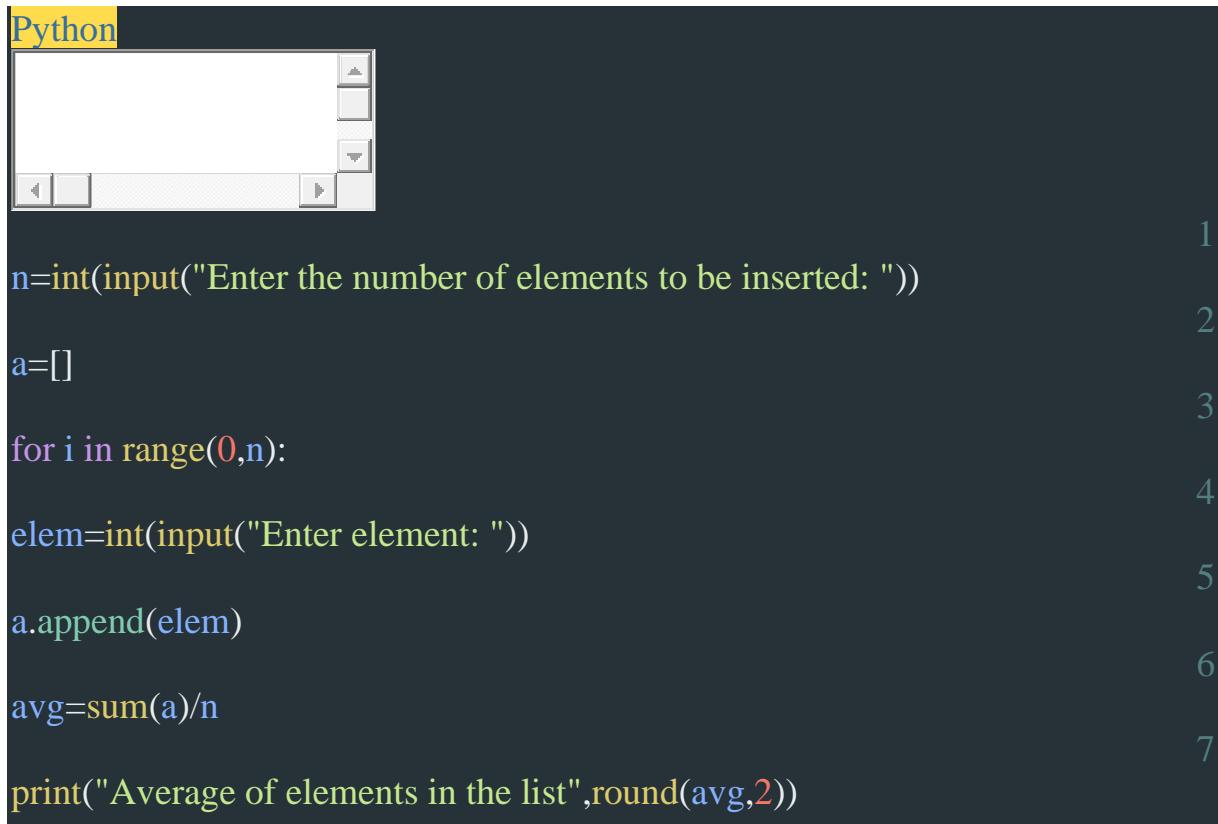
126) What is the output when we execute list("hello")?

A) ['h', 'e', 'l', 'l', 'o']

Python Coding Interview Questions And Answers For Experienced

127) Can you write a program to find the average of numbers in a list in Python?

A) Python Program to Calculate Average of Numbers:



```
1 n=int(input("Enter the number of elements to be inserted: "))
2 
3 a=[]
4 
5 for i in range(0,n):
6     elem=int(input("Enter element: "))
7     a.append(elem)
8 
9 print("Average of elements in the list",round(avg,2))
```

Output:

Enter the number of elements to be inserted: 3

Enter element: 23

Enter element: 45

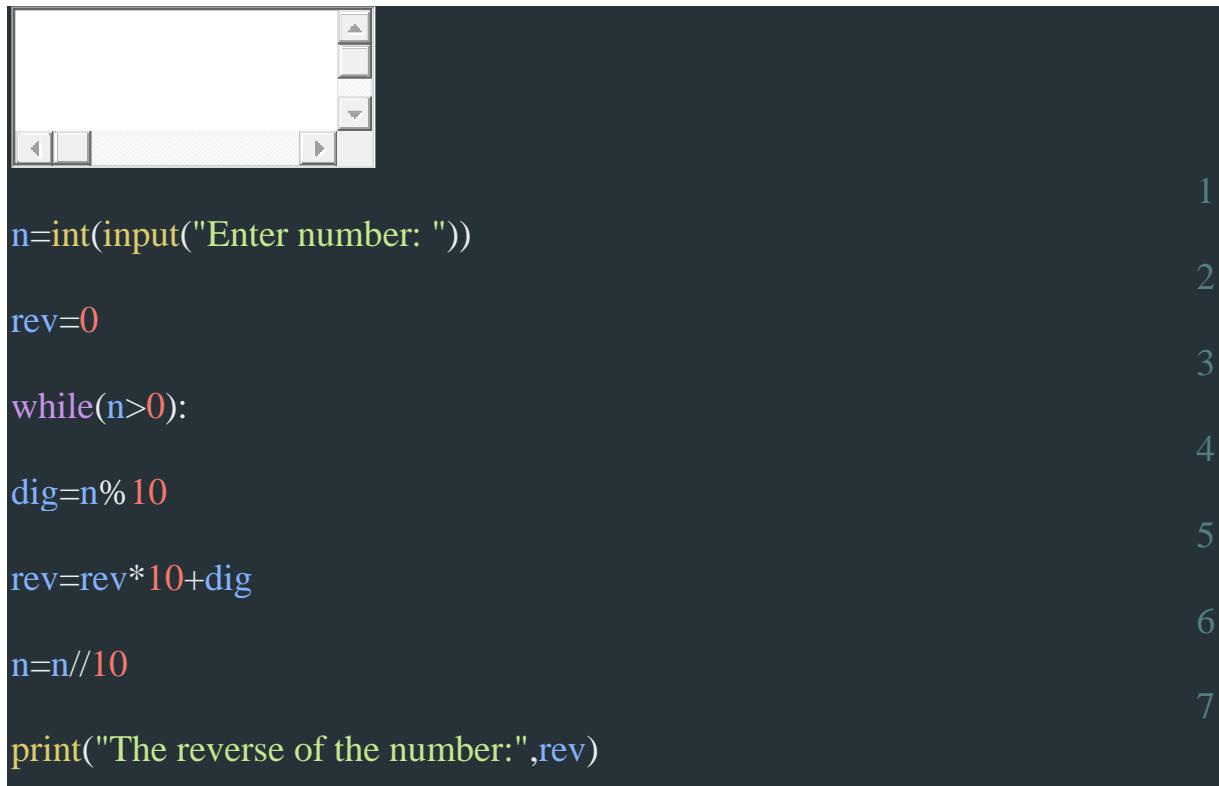
Enter element: 56

Average of elements in the list 41.33

128) Write a program to reverse a number in Python?

A) Python Program to Reverse a Number:



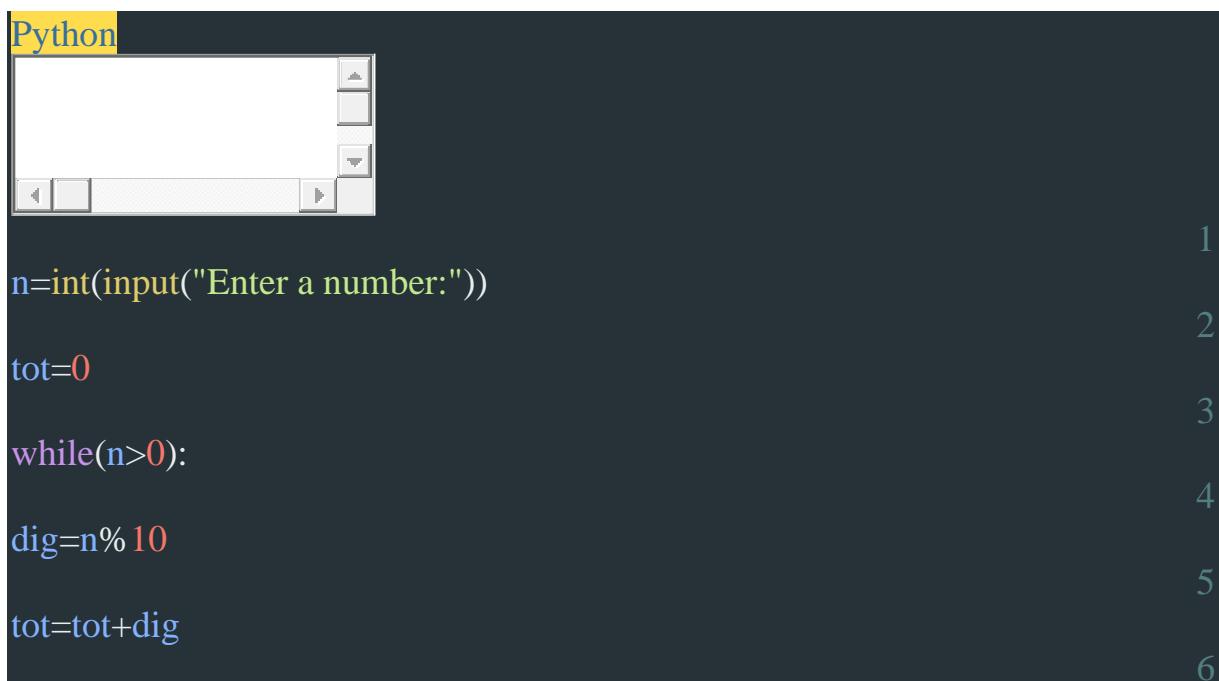
```
1 n=int(input("Enter number: "))
2 rev=0
3 while(n>0):
4     dig=n%10
5     rev=rev*10+dig
6     n=n//10
7 print("The reverse of the number:",rev)
```

Output:

```
Enter number: 143
The reverse of the number: 341
```

129) Write a program to find the sum of the digits of a number in Python?

A) Python Program to Find Sum of the Digits of a Number

```
1 n=int(input("Enter a number:"))
2 tot=0
3 while(n>0):
4     dig=n%10
5     tot=tot+dig
6
```

```
n=n//10  
print("The total sum of digits is:",tot)
```

7

Output:

Enter a number:1928
The total sum of digits is: 20

130) Write a Python Program to Check if a Number is a Palindrome or not?

A) Python Program to Check if a Number is a Palindrome or Not:

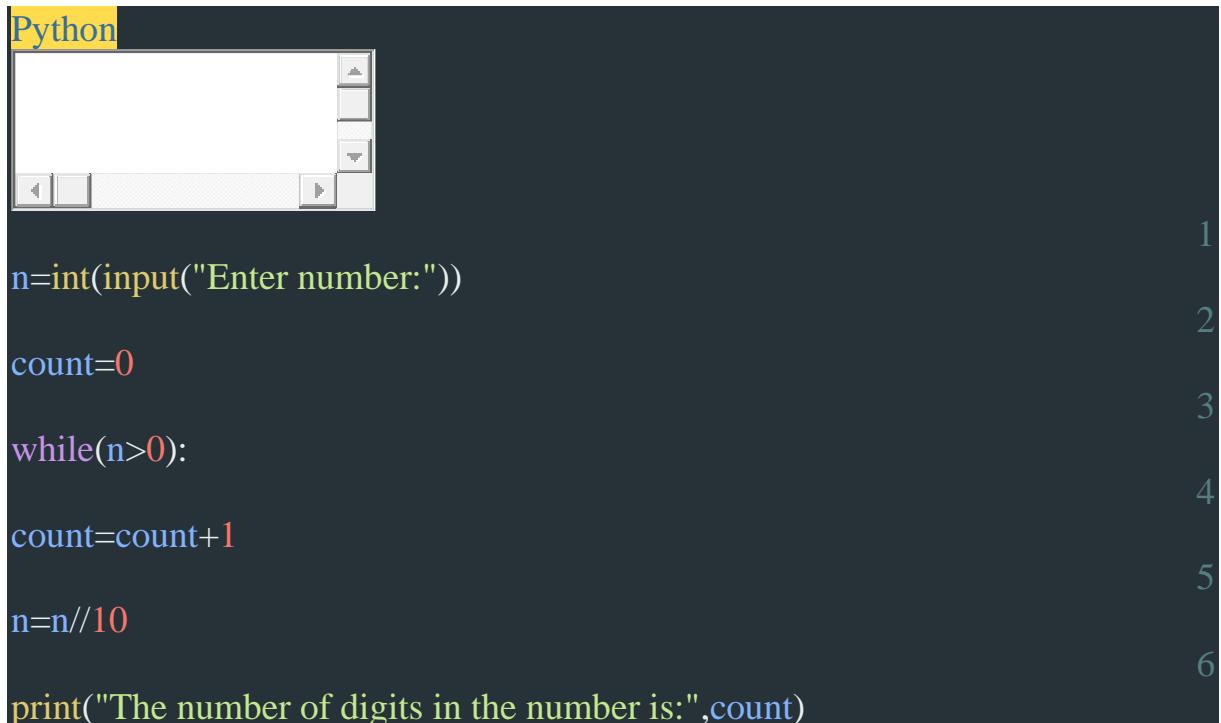
```
Python  
n=int(input("Enter number:"))  
temp=n  
rev=0  
while(n>0):  
    dig=n%10  
    rev=rev*10+dig  
    n=n//10  
if(temp==rev):  
    print("The number is a palindrome!")  
else:  
    print("The number isn't a palindrome!")
```

Output:

Enter number:151
The number is a palindrome!

131) Write a Python Program to Count the Number of Digits in a Number?

A) Python Program to Count the Number of Digits in a Number:



```
1
2
3
4
5
6
```

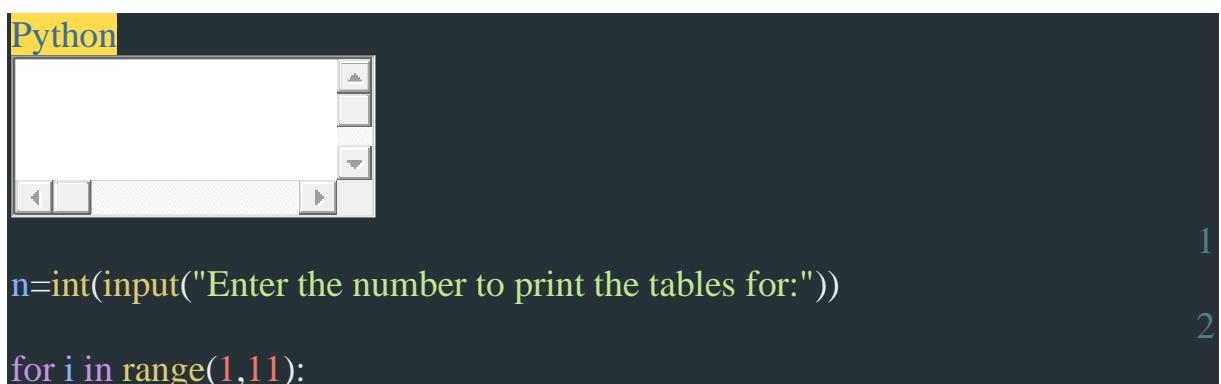
```
n=int(input("Enter number:"))
count=0
while(n>0):
    count=count+1
    n=n//10
print("The number of digits in the number is:",count)
```

Output:

Enter number:14325
The number of digits in the number is: 5

132) Write a Python Program to Print Table of a Given Number?

A) Python Program to Print Table of a Given Number:



```
1
2
3
4
5
6
```

```
n=int(input("Enter the number to print the tables for:"))
for i in range(1,11):
```

```
print(n,"x",i,"=",n*i)
```

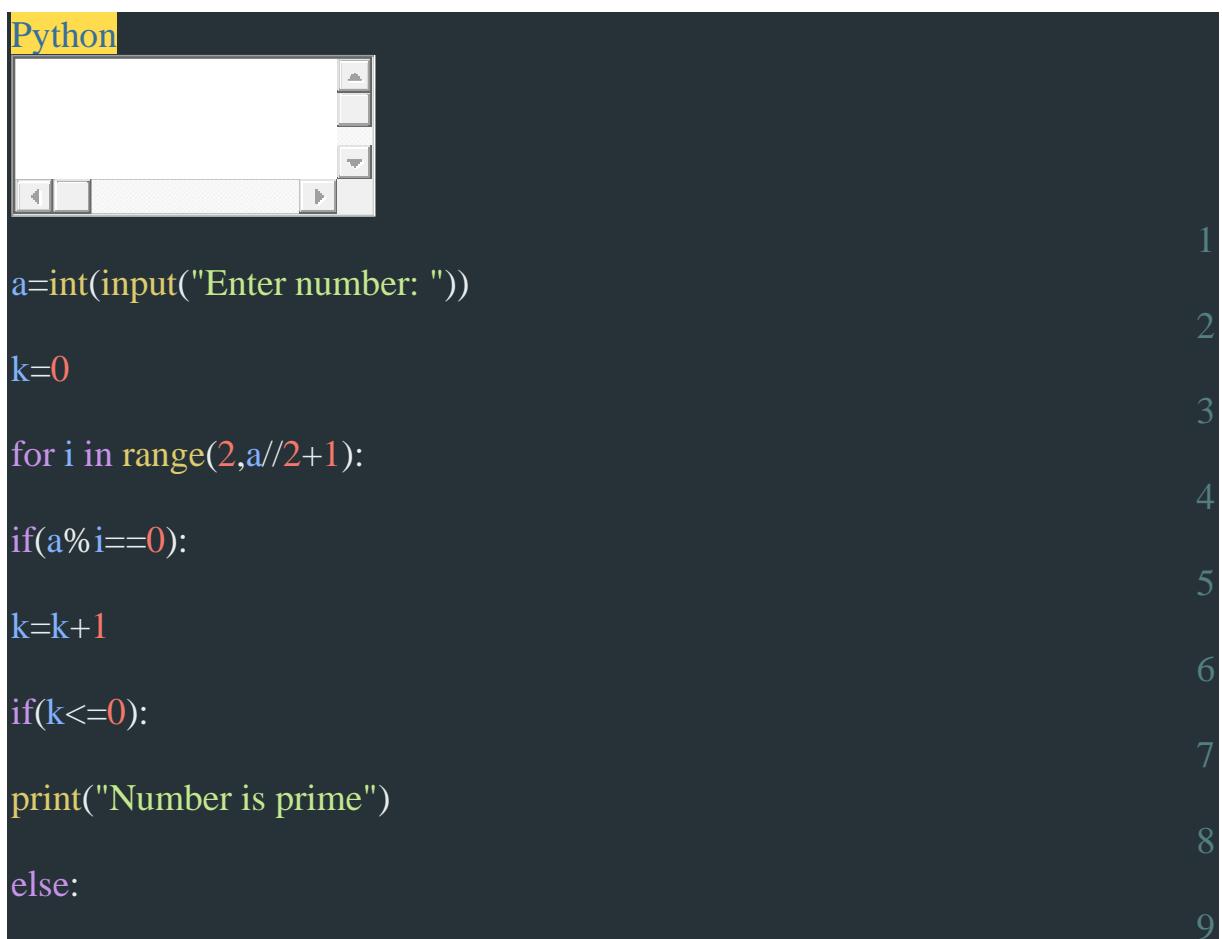
Output:

Enter the number to print the tables for:7

```
7 x 1 = 7
7 x 2 = 14
7 x 3 = 21
7 x 4 = 28
7 x 5 = 35
7 x 6 = 42
7 x 7 = 49
7 x 8 = 56
7 x 9 = 63
7 x 10 = 70
```

133) Write a Python Program to Check if a Number is a Prime Number?

A) Python Program to Check if a Number is a Prime Number:



The screenshot shows a Python code editor window with the word "Python" in a yellow bar at the top. The code itself is a prime number checker. It starts with a comment "1", followed by the input statement "a=int(input("Enter number: "))". A comment "2" follows. Then, the variable "k=0" is assigned, with a comment "3" next. A "for" loop begins with "for i in range(2,a//2+1)". A comment "4" is placed after the colon. Inside the loop, an "if" condition checks if "a%i==0", with a comment "5" after it. If true, "k=k+1" is executed, with a comment "6" after it. After the loop, an "if" condition checks if "k<=0", with a comment "7" after it. Finally, "print("Number is prime")" is called, with a comment "8" before it. An "else" block is present at the bottom, with a comment "9" after it.

```

1
a=int(input("Enter number: "))
2
k=0
3
for i in range(2,a//2+1):
4
    if(a%i==0):
5
        k=k+1
6
    if(k<=0):
7
        print("Number is prime")
8
else:
9

```

```
print("Number isn't prime")
```

Output:

Enter number: 7

Number is prime

134) Write a Python Program to Check if a Number is an Armstrong Number?

A) Python Program to Check if a Number is an Armstrong Number:

```
Python
1
n=int(input("Enter any number: "))
2
a=list(map(int,str(n)))
3
b=list(map(lambda x:x**3,a))
4
if(sum(b)==n):
5
    print("The number is an armstrong number. ")
6
else:
7
    print("The number isn't an arsmtrong number. ")
```

Output:

Enter any number: 371

The number is an armstrong number.

135) Write a Python Program to Check if a Number is a Perfect Number?

A) Python Program to Check if a Number is a Perfect Number:

```
Python
1
2
3
4
5
6
7
```

```
1
n = int(input("Enter any number: "))
2
sum1 = 0
3
for i in range(1, n):
4
    if(n % i == 0):
5
        sum1 = sum1 + i
6
    if (sum1 == n):
7
        print("The number is a Perfect number!")
8
else:
9
    print("The number is not a Perfect number!")
```

Output:

```
Enter any number: 6
The number is a Perfect number!
```

Python Developer Interview Questions And Answers

136) Write a Python Program to Check if a Number is a Strong Number?

A) Python Program to Check if a Number is a Strong Number:



```
1
sum1=0
2
num=int(input("Enter a number:"))
3
temp=num
4
while(num):
```

```

5
i=1
6
f=1
7
r=num%10
8
while(i<=r):
9
    f=f*i
10
    i=i+1
11
    sum1=sum1+f
12
num=num//10
13
if(sum1==temp):
14
    print("The number is a strong number")
15
else:
16
    print("The number is not a strong number")

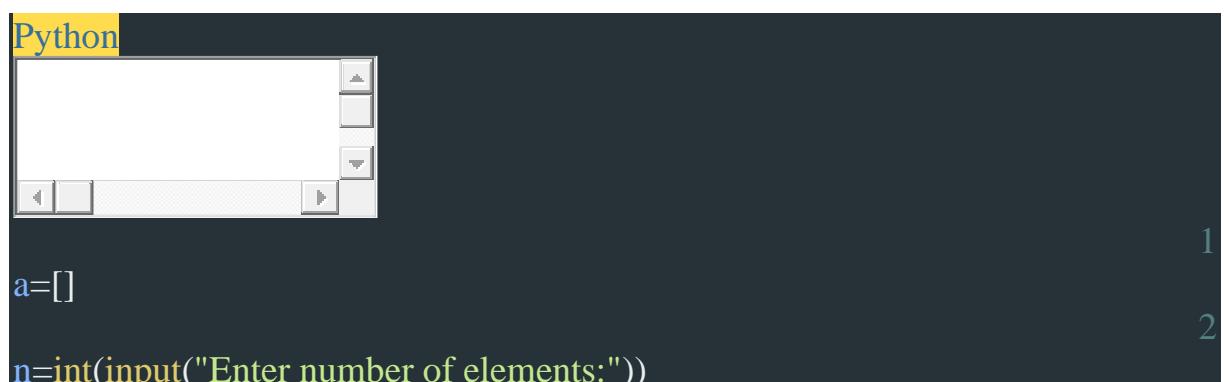
```

Output:

Enter a number:145
The number is a strong number.

137) Write a Python Program to Find the Second Largest Number in a List?

A) Python Program to Find the Second Largest Number in a List:



The screenshot shows a Python code editor window. The title bar says "Python". The code area contains the following lines:

```

1
a=[]
2
n=int(input("Enter number of elements:"))

```

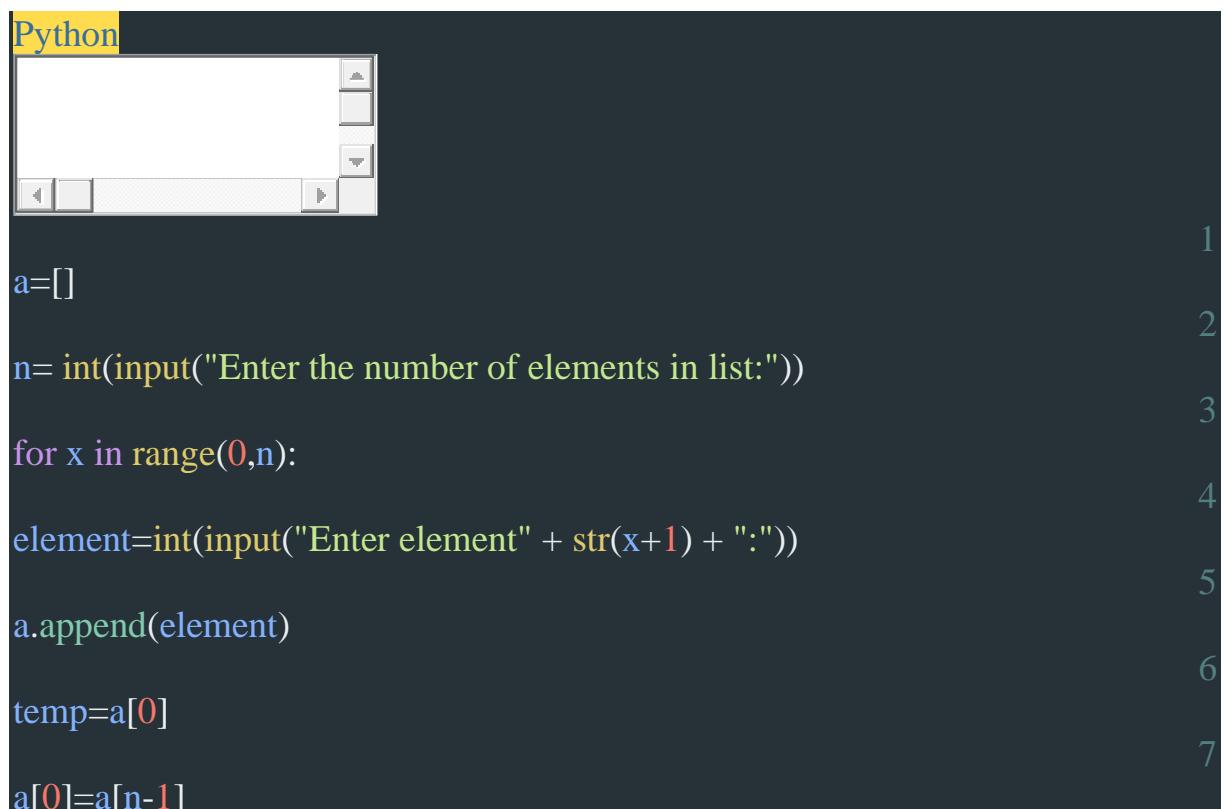
```
3  
for i in range(1,n+1):  
4  
b=int(input("Enter element:"))  
5  
a.append(b)  
6  
a.sort()  
7  
print("Second largest element is:",a[n-2])
```

Output:

```
Enter number of elements:4  
Enter element:23  
Enter element:56  
Enter element:39  
Enter element:11  
Second largest element is: 39
```

138) Write a Python Program to Swap the First and Last Value of a List?

A) Python Program to Swap the First and Last Value of a List:



The screenshot shows a Python code editor window with a dark theme. The title bar says "Python". The code area contains the following Python script:

```
1  
a=[]  
2  
n= int(input("Enter the number of elements in list:"))  
3  
for x in range(0,n):  
4  
element=int(input("Enter element" + str(x+1) + ":"))  
5  
a.append(element)  
6  
temp=a[0]  
7  
a[0]=a[n-1]
```

```
8  
a[n-1]=temp  
9  
print("New list is:")  
10  
print(a)
```

Output:

Enter the number of elements in list:4

Enter element1:23

Enter element2:45

Enter element3:67

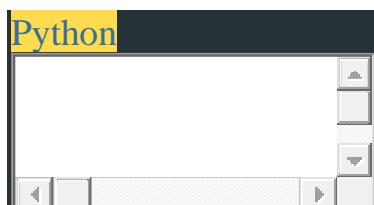
Enter element4:89

New list is:

[89, 45, 67, 23]

139) Write a Python Program to Check if a String is a Palindrome or Not?

A) Python Program to Check if a String is a Palindrome or Not:



```
1  
string=raw_input("Enter string:")  
2  
if(string==string[::-1]):  
3  
    print("The string is a palindrome")  
4  
else:  
5  
    print("The string isn't a palindrome")
```

Output:

Enter string: malayalam

The string is a palindrome

140) Write a Python Program to Count the Number of Vowels in a String?

A) Python Program to Count the Number of Vowels in a String:

```
Python
1
2
3
4
5
6
7
```

```
string=raw_input("Enter string:")
vowels=0
for i in string:
if(i=='a' or i=='e' or i=='i' or i=='o' or i=='u' or i=='A' or i=='E' or i=='I' or i=='O'
or i=='U'):
vowels=vowels+1
print("Number of vowels are:")
print(vowels)
```

Output:

```
Enter string: Hello world
Number of vowels are: 3
```

141) Write a Python Program to Check Common Letters in Two Input Strings?

A) Python Program to Check Common Letters in Two Input Strings:

```
Python
1
2
3
```

```
s1=raw_input("Enter first string:")
s2=raw_input("Enter second string:")
a=list(set(s1)&set(s2))
```

```
4  
print("The common letters are:")  
5  
for i in a:  
6  
    print(i)
```

Output:

```
Enter first string:Hello  
Enter second string:How are you  
The common letters are:  
H  
e  
o
```

1. What is Python? List some popular applications of Python in the world of technology?

Python is a widely-used general-purpose, high-level programming language. It was created by Guido van Rossum in 1991 and further developed by the Python Software Foundation. It was designed with an emphasis on code readability, and its syntax allows programmers to express their concepts in fewer lines of code.

It is used for:

- System Scripting
- Web Development
- Game Development
- Software Development
- Complex Mathematics

2. What are the benefits of using Python language as a tool in the present scenario?

Following are the benefits of using Python language:

- Object-Oriented Language
- High Level Language
- Dynamically Typed language
- Extensive support Libraries
- Presence of third-party modules
- Open source and community development
- Portable and Interactive
- Portable across Operating systems

3. Which sorting technique is used by sort() and sorted() functions of python?

Python uses [Tim Sort](#) algorithm for sorting. It's a stable sorting whose worst case is $O(N \log N)$. It's a hybrid sorting algorithm, derived from merge sort and insertion sort, designed to perform well on many kinds of real-world data.

4. Differentiate between List and Tuple?

Let's analyze the differences between List and Tuple:

List

- Lists are Mutable datatype.
- Lists consume more memory
- The list is better for performing operations, such as insertion and deletion.
- Implication of iterations is Time-consuming

Tuple

- Tuples are Immutable datatype.
- Tuple consume less memory as compared to the list
- Tuple data type is appropriate for accessing the elements
- Implication of iterations is comparatively Faster

5. How memory management is done in Python?

Python uses its private heap space to manage the memory. Basically, all the objects and data structures are stored in the private heap space. Even the programmer can not access this private space as the interpreter takes care of this space. Python also has an inbuilt garbage collector, which recycles all the unused memory and frees the memory and makes it available to the heap space.

6. What is PEP 8?

PEP 8 is a Python style guide. It is a document that provides the guidelines and best practices on how to write beautiful Python code. It promotes a very readable and eye-pleasing coding style.

7. Is Python a compiled language or an interpreted language?

Actually, Python is a partially compiled language and partially interpreted language. The compilation part is done first when we execute our code and this will generate byte code and internally this byte code gets converted by the python virtual machine(p.v.m) according to the underlying platform(machine+operating system).

8. How to delete a file using Python?

We can delete a file using Python by following approaches:

- `os.remove()`
- `os.unlink()`

9. What are Decorators?

Decorators are a very powerful and useful tool in Python as they are the specific change that we make in Python syntax to alter functions easily.

10. What is the difference between Mutable datatype and Immutable datatype?

Mutable data types can be edited i.e., they can change at runtime. Eg – List, Dictionary, etc.

Immutable data types can not be edited i.e., they can not change at runtime. Eg – String, Tuple, etc.

11. What is the difference between Set and Dictionary?

Set is an unordered collection of data type that is iterable, mutable, and has no duplicate elements.

Dictionary in Python is an unordered collection of data values, used to store data values like a map.

12. How do you debug a Python program?

By using this command we can debug a python program:

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

13. What is Pickling and Unpickling?

Pickle module accepts any Python object and converts it into a string representation and dumps it into a file by using the dump function, this process is called pickling. While the process of retrieving original Python objects from the stored string representation is called unpickling.

14. How are arguments passed by value or by reference in Python?

Everything in Python is an object and all variables hold references to the objects. The reference values are according to the functions; as a result, you cannot change the value of the references. However, you can change the objects if it is mutable.

15. What is List Comprehension? Give an Example.

List comprehension is a syntax construction to ease the creation of a list based on existing iterable.

For Example:

```
my_list = [i for i in range(1, 10)]
```

16. What is Dictionary Comprehension? Give an Example

Dictionary Comprehension is a syntax construction to ease the creation of a dictionary based on the existing iterable.

For Example: *my_dict = {i: i+7 for i in range(1, 10)}*

17. Is Tuple Comprehension? If yes, how and if not why?

(i for i in (1, 2, 3))

Tuple comprehension is not possible in Python because it will end up in a generator, not a tuple comprehension.

18. What is namespace in Python?

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

19. What is a lambda function?

A lambda function is an anonymous function. This function can have any number of parameters but, can have just one statement. For Example:

```
a = lambda x, y : x*y  
print(a(7, 19))
```

20. What is a pass in Python?

Pass means performing no operation or in other words, it is a place holder in the compound statement, where there should be a blank left and nothing has to be written there.

21. What is the difference between xrange and range function?

range() and xrange() are two functions that could be used to iterate a certain number of times in for loops in Python. In Python 3, there is no xrange, but the range function behaves like xrange in Python 2.

- *range()* – This returns a list of numbers created using range() function.
- *xrange()* – This function returns the generator object that can be used to display numbers only by looping. The only particular range is displayed on demand and hence called *lazy evaluation*.

22. What is difference between / and // in Python?

// represents floor division whereas / represents precise division. For Example:

```
5//2 = 2
```

```
5/2 = 2.5
```

23. What is zip function?

Python zip() function returns a zip object, which maps a similar index of multiple containers. It takes an iterable, converts into iterator and aggregates the elements based on iterables passed. It returns an iterator of tuples.

24. What is swapcase function in Python?

It is a string's function that converts all uppercase characters into lowercase and vice versa. It is used to alter the existing case of the string. This method creates a copy of the string which contains all the characters in the swap case. For Example:

```
string = "GeeksforGeeks"  
string.swapcase() ---> "gEEKSfORgEEKS"
```

25. What are Iterators in Python?

In Python, iterators are used to iterate a group of elements, containers like a list. Iterators are the collection of items, and it can be a list, tuple, or a dictionary. Python iterator implements `__itr__` and the `next()` method to iterate the stored elements. In Python, we generally use loops to iterate over the collections (list, tuple).

26. What are Generators in Python?

In Python, the generator is a way that specifies how to implement iterators. It is a normal function except that it yields expression in the function. It does not implements `__itr__` and `next()` method and reduces other overheads as well.

If a function contains at least a `yield` statement, it becomes a generator. The `yield` keyword pauses the current execution by saving its states and then resumes from the same when required.

27. What are the new features added in Python 3.8 version?

Following are the new features in Python 3.8 version:

- Positional Only parameter(/)
- Assignment Expression(:=)
- f-strings now support “=”
- `reversed()` works with a dictionary

28. What is monkey patching in Python?

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

```
# g.py
class GeeksClass:
    def function(self):
        print "function()

import m
def monkey_function(self):
    print "monkey_function()

m.GeeksClass.function = monkey_function
obj = m.GeeksClass()
obj.function()
```

29. Does Python supports multiple Inheritance?

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

30. What is Polymorphism in Python?

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.

31. Define encapsulation in Python?

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

32. How do you do data abstraction in Python?

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.

33. Which databases are supported by Python?

MySQL (Structured) and MongoDB (Unstructured) are the prominent databases that are supported natively in Python. Import the module and start using the functions to interact with the database.

34. How is Exceptional handling done in Python?

There are 3 main keywords i.e. try, except, and finally which are used to catch exceptions and handle the recovering mechanism accordingly. Try is the block of a code which is monitored for errors. Except block gets executed when an error occurs.

The beauty of the final block is to execute the code after trying for error. This block gets executed irrespective of whether an error occurred or not. Finally block is used to do the required cleanup activities of objects/variables.

35. What does '#' symbol do in Python?

'#' is used to comment out everything that comes after on the line.

36. Write a code to display the current time?

```
currenttime= time.localtime(time.time())
```

```
print ("Current time is", currenttime)
```

37. What is the difference between a shallow copy and deep copy?

Shallow copy is used when a new instance type gets created and it keeps values that are copied whereas deep copy stores values that are already copied.

A shallow copy has faster program execution whereas deep copy makes it slow.

38. What is PIP?

PIP is an acronym for Python Installer Package which provides a seamless interface to install various Python modules. It is a command-line tool that can search for packages over the internet and install them without any user interaction.

39. What is `__init__()` in Python?

Equivalent to constructors in OOP terminology, `__init__` is a reserved method in Python classes. The `__init__` method is called automatically whenever a new object is initiated. This method allocates memory to the new object as soon as it is created. This method can also be used to initialize variables.

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

Identifiers in Python can be of any length.

What is Python?

Python is one of the most popular programming languages and has many use cases. It's an interpreted object-oriented programming language and is considered a high-level language. Python has applications in many different fields, from [software engineering](#) to [data science](#). As a highly versatile language, it is a useful language to know in most positions related to software development, as it can complete tasks from application development to data analysis.

Python is a free and open-sourced language that has a lot of community support and many packages that are available within its large standard library. Over the years, a number of tools have been developed that allow higher-level, more complex problems to be solved than were possible in the past. This means that you have little need for external libraries, as much of Python's standard library provides the functionality that you need. As a highly extensible language, programs written in Python can be easily transferred to other languages.

Name Some Common Native Data Structures in Python. Of These, Which Are Mutable, and Which Are Immutable?

The four built-in data structures in Python are lists, tuples, sets, and dictionaries. Some of the key differences between these types have to do with their mutability. Lists and dictionaries are both mutable data structures, meaning they can be altered after they are created. Sets and tuples contain immutable objects, meaning they

cannot be changed after they are created. They contain iterable objects with no duplicate elements.

List

General purpose
Most widely used data structure
Grow and shrink size as needed
Sequence type
Sortable

Tuple

Immutable (can't add/change)
Useful for fixed data
Faster than Lists
Sequence type

Set

Store non-duplicate items
Very fast access vs Lists
Math Set ops (union, intersect)
Unordered

Dict

Key/Value pairs
Associative array, like Java HashMap
Unordered

Sets and tuples are more selective than lists. This makes them more efficient and allows the programs that use them to run faster than they would with lists.

Dictionaries use a key to reference their elements and are unordered with no duplicate elements. Their keys must be of an immutable type, as dictionaries themselves are mutable and key mutability could cause an issue with referencing elements within the dictionary.

The differences between different data structures depend on their use cases. Sets are used when you don't want to duplicate your items and need a simple, fast option. Lists are a one-size fits all option that might not be as fast as some but are able to grow and shrink as needed. Tuples are ideal for fixed data and are faster options for large sets that are contained in square brackets. Dictionaries are great for when you need your data easily referenced by unique keys.

Can You Explain What a List and Dict Comprehension Is?

Both a list comprehension and a dict comprehension are single-line syntactic constructs that allow for efficient use of code to accomplish complex tasks. This is a much quicker way than many conditional statements, like for and if loops. They are examples of some of the many special functions that Python offers that can make processes simpler. An example of a dict comprehension can be seen in this list of prices in a list being changed from US Dollars to British Pounds.

```
#item price in dollars
old_price = {'milk': 1.02, 'coffee': 2.5, 'bread': 2.5}

dollar_to_pound = 0.76
new_price = {item: value*dollar_to_pound for (item, value) in old_price.items()}
print(new_price)
```

Run Code »

The following is the output you would get:

```
{'milk': 0.7752, 'coffee': 1.9, 'bread': 1.9}
```

As you can see, the new dictionary has been updated to reflect the items for the last dictionary with a slight modification made. This method is fairly lightweight and doesn't require any complex programming to accomplish a simple task.

For an example of list comprehension, we can modify items in a list without the use of conditional statements.

```
fruits = ["apple", "banana", "cherry", "kiwi", "mango"]
newlist = []

for x in fruits:
    if "a" in x:
        newlist.append(x)

print(newlist)
```

Instead of a loop, we can carry the values over to a new list with the necessary modifications made.

```
fruits = ["apple", "banana", "cherry", "kiwi", "mango"]

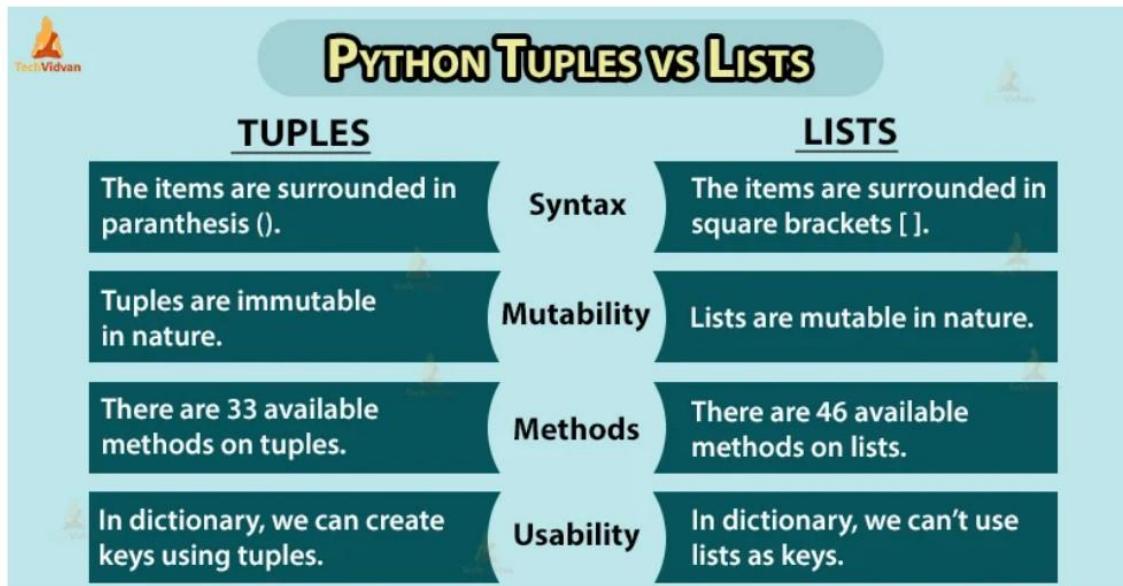
newlist = [x for x in fruits if "a" in x]

print(newlist)
```

Both of these options lower the amount of time per iteration and make more efficient use of memory. The for loop may take up more memory than the method that didn't utilize a loop. However, they are both proper syntax that will return the right result and are usually used for the same purpose, just in different contexts. Many developers prefer the use of either a list comprehension or a dict comprehension as it allows them to create more time-efficient, more easily executable code that uses

less memory. There may be some programs where a slightly slower program option is better as it will make more sense in the surrounding program's structure.

What Is the Difference Between a List and a Tuple?



Both a list and a tuple are a collection of items. The main difference between list objects and tuple objects is mutability. While you can make changes to a list, a tuple contains immutable objects that cannot be altered. This generally makes a tuple a faster option that doesn't use as much memory as a list. Another major difference is in how you define a list vs a tuple. When defining a tuple, you will use a pair of parentheses. When defining a list, you will use a pair of square brackets.

```
list_num = [1,2,3,4]
tup_num = (1,2,3,4)

print(list_num)
print(tup_num)
```

This numerical list of integers and the matching tuple would result in the following output:

```
[1,2,3,4]
(1,2,3,4)
```

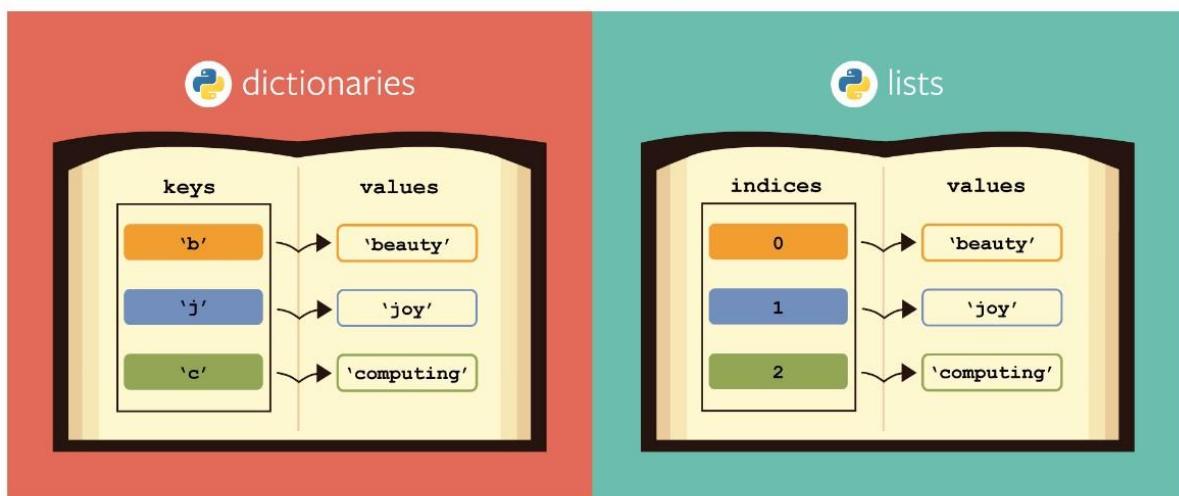
A similarity that these two collection data types have in common is that they are both a sequence data types that can store a wide range of objects.

When Would You Use a List vs. a Tuple vs. a Set in Python?

The differences between these data types come down to what they are being used for. A list is a widely used data structure that is good for many uses. They are ordered, changeable, and allow duplicate values. If you are working with a data set that shares many of these same characteristics, but the data needs to remain unchangeable, then you should use a tuple instead. Tuples are also ordered and allow duplicate members but remain unchanged.

For the purposes of memory usage, a tuple is a data type that would likely be used for memory-heavy tasks. As it is more specialized than a list, it takes up less memory and allows for faster iterations. While lists can be used for these kinds of programs, they would likely slow the program down and would not be ideal for use on large data sets that require a significant amount of memory usage.

What Are the Typical Characteristics of Elements in a List vs. in a Dictionary?



Elements in both Python lists and dictionaries are mutable and dynamic. They can both be nested and are both able to be indexed. However, it is important to note that the index for a Python dictionary is referenced with a letter. This index is actually called a "key." A list index is referenced with a numerical value, with the first element in the list having a value of 0 and the second element having a value of 1, and so on. As of Python version 3.7, a Python dictionary is no longer an unordered collection. In versions 3.6 and before, they were considered to be unordered collections as the items could not be referred to with an index.

What is PEP 8?

PEP 8 stands for "[Python Enhancement Proposal 8](#)." It is a document that was written by [Guido van Rossum](#) in 2001 that provides guidelines on how to write Python code. The key focus of PEP 8 is to enhance the readability of Python code in order to help facilitate consistency in how the code is structured. This allows for collaboration between developers. This standardization, along with the general simplicity behind the Python language, is one of the reasons Python is so widely used and popular with developers.

What Is Tkinter in Python Used For?

Tkinter stands for “TK interface” and is the most commonly used method to create Graphical User Interfaces (GUIs) in Python. As the only standard method that is built into Python’s library, it is the one most developers choose. By using Tkinter, developers are able to create desktop applications and other graphical interfaces.

What Is a Decorator?

In Python, a decorator function is a function that allows new functionality to be added to an existing object. This is done without any modifications being done to the structure of the object. Using a decorator function, a developer can use a function with some new features added to it while maintaining the integrity of that same function.

What's the Difference Between A for Loop and a While Loop?

For vs While Loop

Comparison Chart

For Loop	While Loop
The for loop is used for definite loops when the number of iterations is known.	The while loop is used when the number of iterations is not known.
For loops can have their counter variables declared in the declaration itself.	There is no built-in loop control variable with a while loop.
This is preferable when we know exactly how many times the loop will be repeated.	The while loop will continue to run infinite number of times until the condition is met.
The loop iterates infinite number of times if the condition is not specified.	If the condition is not specified, it shows a compilation error.

Loops are used to run a block of code over an iterable object. Both For and While Loops can accomplish this process with similar iterable objects. The main difference between the two is the number of iterations. It's not always known how many times you'll have to run a loop until the necessary result is reached. In these cases, a parameter will be set and a While Loop will run until the condition is met. Developers have to be careful to remember that, without a condition that stops the loop from running any further, a While Loop can become an infinite loop and will run until the entire program is terminated. This is not the case with a For Loop, which has a definite number of iterations that are easily established in the loop's declaration.

Tell Us the Difference Between a Shallow Copy and a Deep Copy

The difference between a Shallow and a Deep Copy is what information is shared and carried over to the second copy. With a Shallow Copy, only the reference address is copied. With a Deep Copy, both the original object and all repetitive copies are copied and stored.

What Makes Python an Interpreted Language?

Python is considered an interpreted language because it converts its programs into bytecode that is then executed by a Python virtual machine. This process requires the deployment of an interpreter that helps to make the code from Python understandable and usable by your computer. This interpretation process carries out implicit type conversion automatically. Through implicit type conversion, the other interpreted programming languages include PHP, Ruby, and JavaScript.

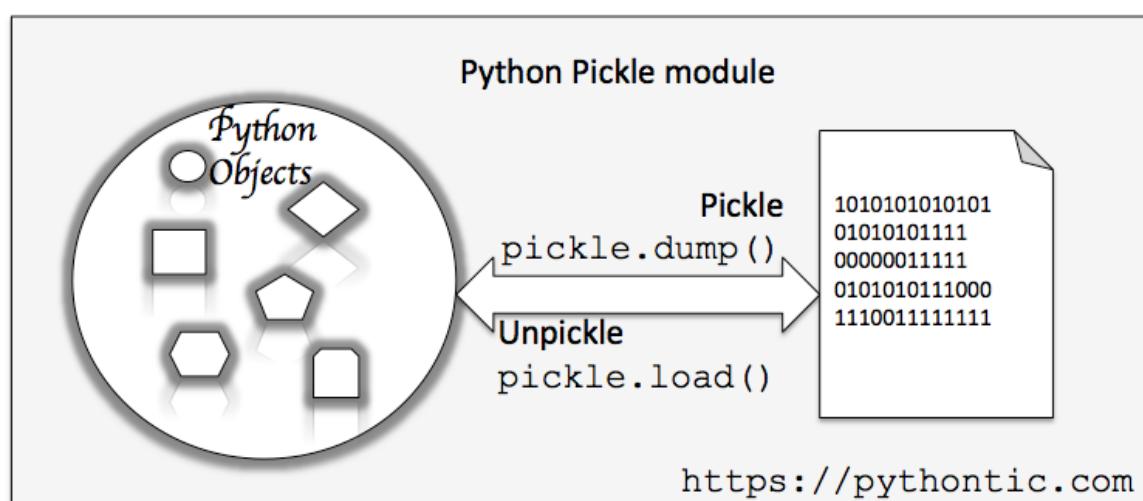
Is Python an Object-Oriented Programming Language?

Python is an [object-oriented programming language](#). While it has the ability to also allow for functional programming, it is primarily object-oriented.

How Is Memory Managed in Python?

Python's memory manager automatically allocates memory. Python contains a garbage collector that helps automate the reuse of all unused memory within a program. Objects are deleted from the heap memory as soon as they have no reference point. This allows memory to be freed up and allows for more efficient memory management.

Explain the Difference Between Pickling and Unpickling



Pickling is the process of converting the hierarchy of Python objects into a byte stream, and then into a Python file as a string representation. Unpickling is the opposite of this process. Python programs run from source code files and are directly interpreted by the byte code. Once the virtual machine begins to move the objects from a specific program to the computer, the pickle module carries out a process of serialization of those objects into an understandable language that the computer can store through a dump function.

This dump function's primary function is to convert the program to a Python file. Then, when a computer's Python source code files begin to convert back, the unpickling process starts. This reverse process is also accomplished through the pickle module when the executable code is converted back and the string representations are turned back into the original Python object.

Name Some Packages From the Python Standard Library That Are Useful for Data Science Projects

A package in Python is a collection of items that is able to be imported into a program in order to help facilitate complex processes. Many of the packages in the Python Standard Library are helpful for data science purposes. Most of these packages utilize Python to develop [machine learning algorithms](#), [data visualization](#), [exploratory data analysis](#), and [neural networks](#). While it's not necessary to know all of the packages in Python, a good knowledge of at least several options is highly recommended before you go into a Python interview. These are just a few of the many packages that are available.

Caffe

Caffe is a package that helps with image recognition processes. As a deep learning framework, it boasts a highly robust capacity that can process [60 million images a day](#).

PyTorch

PyTorch is a tool that helps in the building of dynamic neural networks. It has many features that make it a highly versatile option for those who are in need of a tool for [deep neural network](#) projects.

Theano

Theano is an open-source deep-learning library. While no more updates are being released on this package, similar frameworks exist that copy many of Theano's functionality.

TensorFlow

TensorFlow is a widely used machine learning package that is considered one of the most famous. It's a highly flexible option for data scientists who are specializing in numerical computation.

Keras

Keras is a popular tool that is built for experimentation. It is a deep-learning library that is popular due to its API, which is considered to be one of the more user-friendly options.

Pandas

The Pandas library is the go-to Python library for data analysis. It can be used for many different operations within the data science and data analytics specialty and is a popular option for large data sets. The Pandas library is one of the most popular options for data manipulation within Python. In order to import Pandas, the import statement “import pandas as pd” is used.

NumPy

NumPy is a package in Python that is used for scientific computing. As it was once part of SciPy, it shares many features. However, the two are optimized for different uses and were separated into two different packages in order to accommodate these differences. Through NumPy, you can get access to many scientific computing features that include linear algebra, basic statistics, and other mathematical functions. Universally named “np” when imported into a project, this tool is an easily accessible package. In order to import NumPy, use the import statement “import NumPy as np” at the beginning of a Python program.

Matplotlib

Matplotlib is a package that helps with [data visualization](#) and graph plotting.

SciPy

SciPy is a package that builds on NumPy. As a data science package, it also contains a number of sub-packages that contain many specialized scientific computing features. Like NumPy, SciPy is also used for basic statistics, linear algebra, and many other mathematical functions. It is widely considered a great option for those who need a one-stop shop for all of their scientific computing needs.

In Python, What Is a Lambda Function?

A [Python lambda function](#) is a function without any name or an anonymous function. A lambda function is an anonymous function due to it not having a function name until it's assigned one. It takes any number of arguments and then returns a single expression. They are typically used in order to take up less space for an argument. The syntax for a lambda function consists of an argument and an expression separated by a single ‘:’ in the middle. There can be any number of arguments, but only one expression. For a lambda function, a return statement is not necessary, as it is able to return an expression on its own.

Distinguish Between “Is” And “==”?

In Python, the operator “==” compares the value of two objects. The operator “is,” on the other hand, determines if two variables are pointing to the same object. A great example of this concept can be found in the following programs:

In this example, the operator “==” is checking to see if these two objects are equal in value.

```
a == None
```

In this example, the operator “==” is checking to see if these two objects are equal in value.

```
a is None
```

In this example, the operator will check if “a” refers to the None object. It will perform no further purposes than this one purpose.

What Are Some of the Disadvantages of Python?

One of the most common complaints that developers have about Python is poor memory efficiency. Its inefficient use of memory can heavily tax your RAM and make it difficult to run Python-based programs without using significant portions of memory. Python can also run very slowly and, as a result, can suffer from runtime errors. This is largely due to how Python is interpreted and is something that can bother developers who are used to much faster languages like C++ or Java.

Most of Python’s issues are related to its versatility. While it is not optimized for one particular task, it is still a very flexible language. It’s through this flexibility that many use cases for Python have developed over the years, making it virtually indispensable in a developer’s portfolio. The disadvantages of Python make it a less-than-ideal language for developing web applications and for game development due to its poor memory management and slow processing speeds.

How can you combine two DataFrames in Pandas?

There are a couple of ways that you can combine two Pandas DataFrames. The first way is to simply concatenate either the columns or rows of the Pandas DataFrames.

```
 import pandas as pd  
 df1 = pd.DataFrame({'id': ['A01', 'A02', 'A03', 'A04'],  
                    'Name': ['ABC', 'PQR', 'DEF', 'GHI']})  
  
df3 = pd.DataFrame({'City': ['MUMBAI', 'PUNE', 'MUMBAI', 'DELHI'],  
                    'Age': ['12', '13', '14', '12']})  
  
# the default behaviour is join='outer'  
# inner join  
  
result = pd.concat([df1, df3], axis=1, join='inner')  
display(result)
```

This method will give the following result:

	id	Name
0	A01	ABC
1	A02	PQR
2	A03	DEF
3	A04	GHI
0	B05	XYZ
1	B06	TUV
2	B07	MNO
3	B08	JKL

As you can see, this function takes the column from dataframe 2 and connects it to the end of the column from dataframe 1.

Another method involves using the append() function to concatenate. The code block below shows how this function can be used to join columns and rows together.




```

import pandas as pd
# First DataFrame
df1 = pd.DataFrame({'id': ['A01', 'A02', 'A03', 'A04'],
                     'Name': ['ABC', 'PQR', 'DEF', 'GHI']})

# Second DataFrame
df2 = pd.DataFrame({'id': ['B05', 'B06', 'B07', 'B08'],
                     'Name': ['XYZ', 'TUV', 'MNO', 'JKL']})

frames = [df1, df2]

result = pd.concat(frames)
display(result)

```

And the resulting table will be displayed:

	id	Name	City	Age
0	A01	ABC	NaN	NaN
1	A02	PQR	NaN	NaN
2	A03	DEF	NaN	NaN
3	A04	GHI	NaN	NaN
0	B05	XYZ	NaN	NaN
1	B06	TUV	NaN	NaN
2	B07	MNO	NaN	NaN
3	B08	JKL	NaN	NaN
0	NaN	NaN	MUMBAI	12
1	NaN	NaN	PUNE	13
2	NaN	NaN	MUMBAI	14
3	NaN	NaN	DELHI	12

A namespace is a system that has a unique name for each corresponding object. It contains each variable that you have defined within a certain section of Python code. The hierarchy of namespaces within Python consists of 4 major levels:

While these two methods both work well, many developers prefer the `concat` function over the `appfunction` due to it being a faster option. While both methods work to combine Dataframes in the Pandas library, the decision ultimately comes down to context and functionality.

What Is a Namespace in Python?

A namespace is a system that has a unique name for each corresponding object. It contains each variable that you have defined within a certain section of Python code. The hierarchy of namespaces within Python consists of 4 major levels:

1. Built-in
2. Global
3. Enclosing
4. Local

Built-in namespaces are the namespaces that are part of Python and are available the whole time that Python is running. A function within this namespace is known as a built-in function, and its objects are universal to Python code. Global namespaces are the namespaces that belong to the main program that runs when the interpreter begins to run. Enclosing namespaces and local namespaces are within individual programs and carry the values of variables and objects within the spaces where they are defined. Understanding the hierarchy of Python namespaces can help you understand the way that objects are named and the scope of variables in Python.

Describe Python's Parameter-Passing Mechanism

Parameters in Python are passed by reference. What this means is that if a parameter reference is changed, it reflects back to the calling function. As all Python data types are objects, a parameter passing is passed by assignment. Both the new variable and the original variables in Python will end up pointing to the same object.

What Benefits Do NumPy Arrays Have Compared to Python Lists?

NumPy is one of the major scientific computing packages in Python. Sets of data are organized by using NumPy arrays instead of lists. Depending on what you are using this array for, it can be a far better tool than a list. Arrays only allow for one type of data to be contained within itself. However, lists can [contain multiple data types](#) and, as a result, consume more memory when they are used.

NumPy is designed strictly for computational purposes that are related to numerical values. As it is more compact than Python lists and can be used more efficiently, it is the preferred method for numerical calculations.

Distinguish Between a Module and a Package

If you want to make a larger block of code with a specific function more usable, you can separate it into a Python module or a Python package. This process has a number of advantages, namely that it allows for simplicity and reusability of objects. In Python, multiple packages can be contained in one Python module. The key differences between the two are that packages can contain several modules. A Python module is a simple source code file that allows for easy implementation of certain processes. Packages are created in order to contain multiple Python modules that could be utilized when needed once a package is imported using an import statement.

Coding-Based Python Interview Questions

Here are some of the most commonly asked coding-based Python interview questions:

Reverse an Integer

Reversing an integer in Python is accomplished very simply by converting the number into a string, reversing it, and then displaying the reversed result. This method can be seen in the following example:

```
# the number to be reversed
num = 97402

#convert number to string
num_string = str(num)

# store the size of the number
size = len(num_string)

# use slicing to reverse
reversed_num = num_string[size::-1]

#print("Reversed Number is: " + reversed_num)
print("Reversed Number is: " + reversed_num)
```

As you can see, the integer goes under type conversion to a string and is stored in a new variable. The function reversed() reverses the string, which is then printed out. An integer is a built-in type of data within Python. This same process can work for another built-in data type like a floating-point number.

Check if a String Is a Palindrome

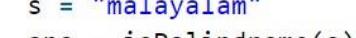
There are several ways that you can use Python to check if a string is a palindrome or not. You can design a program that checks if a statement is a palindrome and returns true or false values, depending if it is or not. The first method employs an if statement to check if or not the user's input can be reversed and retain its meaning. For this example, the word “malayalam” is input by a user.

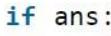
```
 # function which return reverse of a string
 def isPalindrome(s):
     return s == s[::-1]
 # Driver code
 s = "malayalam"
 ans = isPalindrome(s)

 if ans:
     print("Yes")
 else:
     print("No")
```

This program would return “Yes,” as the word “malayalam” can be reversed and still be read the same way. In this situation, the “Yes” is an indication of a true value.

The next method uses a conditional statement, which is, in this case, a For Loop to check if the word “malayalam” is a palindrome:

```
 # function which return reverse of a string
 def isPalindrome(s):
     return s == s[::-1]
 # Driver code
 s = "malayalam"
 ans = isPalindrome(s)

 if ans:
     print("Yes")
 else:
     print("No")
```

This is just another example of how to perform the same operation as the first method. For each of these programs, false values would be indicated by the program returning “No.” While there are many other options for checking if a string is a palindrome, these two examples show two very different methods that employ different tools to accomplish the same task.

Check if a String Only Contains Digits

You can do this with the `str.isdigit()` function. Using this function, a loop can be constructed that will return true or false values based on whether or not the string in question contains digits.

Convert Date From yyyy-mm-dd Format to dd-mm-yyyy Format.

```
from datetime import datetime

# current dateTime
now = datetime.now()

# convert to string
date_time_str = now.strftime("%Y-%m-%d %H:%M:%S")
print('DateTime String:', date_time_str)

# Output 2021-07-20 16:26:24
```

The process of converting the dates in Python from yyyy-mm-dd format to dd-mm-yyyy format involves taking the input for the date, then specifying the formatting of the date as being "%d/%m/%Y" within the formula .strftime(). The formatting is what matters in this program. Any date can be changed over to the dd-mm-yyyy format as long as it is specified within the formula.

Calculate Square of the Provided Number

There are many different ways that you can perform this calculation. This is a standard math equation that you can use in many different programs. For the first method, you can simply multiply the two numbers together:

```
# input a number
number = int (raw_input ("Enter an integer number: "))

# calculate square
square = int(math.pow (number, 2))

# print
print "Square of {0} is {1} ".format (number, square)
```

This method will return the square but is limited to the raw input from the user. While that's good for many use cases, it limits the numbers that can be multiplied by each other.

Another method is to use the inputted number and to use the two multiplication operators, and the number 2 to indicate that the number will be squared. This is a simple method that allows for the calculation of the square by using the variable "number" only once.

```
# Python program to calculate square of a number
# Method 2 (using number**2)

# input a number
number = int (raw_input ("Enter an integer number: "))

# calculate square
square = number**2

# print
print "Square of {0} is {1} ".format (number, square)
```

The last method is to use the math.pow() method, as seen in the example below:

```
# importing math library
import math

# input a number
number = int (raw_input ("Enter an integer number: "))

# calculate square
square = int(math.pow (number, 2))

# print
print "Square of {0} is {1} ".format (number, square)
```

Math.pow() will multiply the first variable in the argument by the power of whatever is put into the second part of the argument. You can use this method to calculate the cubed value, the 4th power, and so on.

Whichever method you will use depends on the context of that operation and what it is trying to accomplish. The best options are usually the options that use the least amount of memory and can be easily replicated or understood by your peers. This is why most companies attempt to standardize certain processes. Doing so makes projects much more communicable and allows for better collaborations between programmers.

Identify the Largest and Smallest Numbers in an Array

There are several methods that you can use to identify the minimum and maximum values in an array. This problem is especially valuable to be able to solve, as it shows up in many statistical analyses. Below are three options that you might use, depending on the situation.

The first method will use variables to store the maximum values and minimum values of the elements in the array.

Run

```
arr = [10, 89, 9, 56, 4, 80, 8]
mini = arr[0]
maxi = arr[0]

for i in range(len(arr)):
    if arr[i] < mini: mini = arr[i]

    if arr[i] > maxi: maxi = arr[i]

print (mini)
print (maxi)
```

This program will return the numbers 4 and 89. Through this method, we can see that a For Loop can help us find the maximum values and minimum values of an array's elements, and store them to variables that we can print, save for future use in the program, or both.

The second method utilizes the inbuilt `.sort()` function to find the value of the minimum and maximum values of its elements.

Run

```
arr = [10, 89, 9, 56, 4, 80, 8]
arr.sort()

print (arr[0])
print (arr[-1])
```

As you can see, using this method, we are able to shorten our program significantly and print the values directly. As this method does not utilize variables, we can print the minimum and maximum values but we cannot store them for later use. The print function would return the same values to us but we will not have those values stored.

The third method is one that utilizes the inbuilt functions `min(arr)` and `max(arr)`.

Run

```
arr = [10, 89, 9, 56, 4, 80, 8]

print (min(arr))
print (max(arr))
```

This method is the shortest so far and only requires that the program print the min and max values directly. There are many different variations of each method that could be run, but each of these is a concise and efficient method to find and print the minimum and maximum values of an array.

Check if the Given Array Is Monotonic

There are a couple of ways that you can check if an array is [monotonic](#). We'll explore two methods for this question that you may get asked in a Python interview. For the first method, we'll look at an example of a quick way to check if an array is monotonic:



```
def isMonotonic(A):
    return (all(A[i] <= A[i + 1] for i in range(len(A) - 1)) or
            all(A[i] >= A[i + 1] for i in range(len(A) - 1)))

# Driver program
A = [6, 5, 4]

# Print required result
print(isMonotonic(A))

# This code is written by
# Sanjit_Prasad
```

In this approach, an array is only monotonic if the monotone is decreasing or increasing. This is all done in just two lines in that initial code block. This approach accomplishes this check-in with very few lines of code, and with only one pass.

The next method that we'll use for this example is a little bit longer:



```
# Check if given array is Monotonic
def isMonotonic(A):
    x, y = [], []
    x.extend(A)
    y.extend(A)
    x.sort()
    y.sort(reverse=True)
    if(x == A or y == A):
        return True
    return False

# Driver program
A = [6, 5, 4]

# Print required result
print(isMonotonic(A))
```

This method utilizes a slightly longer approach where the array is separated into two new arrays that are then sorted into ascending and descending order. These arrays are then compared to the original array to see if they match. If either of them does, then it is determined that the original array was monotonic. This method employs a

much longer code block to accomplish what is essentially the same process as the first method

Q.1What are the advantages of Python over other languages?

Here are the advantages of Python over other languages.

- Presence of Third Party Modules
- Extensive Support Libraries
- Open Source and Community Development
- Learning Ease and Support Available
- User-friendly Data Structures
- Productivity and Speed

Q.2Why pass statement is used in Python?

It is used to Jump out of the loop.

Q.3How import modules are stored in Python?

Import modules are stored as Python code file.

Q.4Which character is used to indicate end of a block?

End of a block is represented by **colon (:)**

Q.5

What is the value in 'x' from the following expression

```
x = 17 / 2 * 3 + 2
```

The value of x is 27.5

Q.6Whati is Tuple in Python

A tuple is a sequence of immutable Python objects.

Q.7Where docstring is placed in a module?

Docstring is placed in the Last line.

Q.8What is Python and give some benefits of using Python?

Python is defined as a programming language with objects, modules, threads, exceptions and automatic memory management. It is very simple and easy, portable, extensible, has a build-in data structure and it is an open source.

Q.9Define PEP 8.

PEP 8 is defined as a coding convention, that gives a set of recommendation, about how to write your Python code more readable.

Q.10Define pickling and unpickling?

The process of picking involves pickle module accepts any Python object and converts it into a string representation and dumps it into a file by using dump function. Where on the other hand the process of retrieving original Python objects from the stored string representation is called unpickling.

Q.11 Given reason why Python is interpreted?

Python language is an interpreted language, such that the program runs directly from the source code. Python language converts the source code that is written by the programmer into an intermediate language, that is again translated into machine language which has to be executed.

Q.12 In Python, how is memory managed?

Steps in managing memory in Python are -

- The Python memory is primarily 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 and interpreter takes care of this Python private heap.
- The allocation of Python heap space for Python objects is done by Python memory manager.
- The core API gives access to some tools for the programmer to code.
- Python has an inbuilt garbage collector, that recycles all the unused memory and frees the memory and makes it available to the heap space.

Q.13 Specify the tools that helps in finding bugs or perform static analysis?

PyChecker is a static analysis tool that is used to detect the bugs in Python source code. PyChecker warns about the style and complexity of the bug. Also Pylint is another tool which verifies whether the module meets the coding standard.

Q.14 What do you understand by Python decorators?

Python decorator can be defined as a specific change that we make in Python syntax to alter functions easily.

Q.15 List the differences between list and tuple.

The point of difference between list and tuple is that list is mutable while tuple is not. Tuple can be hashed for instance as a key for dictionaries.

Q.16 How will you pass arguments by value or by reference?

Since everything in Python is an object and all variables hold references to the objects. Therefore the references values are according to the functions;

due to which you cannot change the value of the references. But we can change the objects if it is mutable.

Q.17 Define namespace in Python.

Since every name introduced has a place where it lives and can be hooked for, in Python. This is referred as namespace. Namespace is like a box where a variable name is mapped to the object placed. Therefore whenever the variable is searched out, this box will be searched, to get corresponding object.

Q.18 Give reason why lambda forms in python does not have statements?

Lambda form in python does not have statements because it is used to make new function object and then return them at runtime.

Q.19 What do you understand by unittest in Python?

unittest is a unit testing framework in Python. Unitest supports sharing of setups, automation testing, shutdown code for tests, aggregation of tests into collections and more.

Q.20 How do you define generators in Python?

Generators are the way of implementing iterators. It is just a normal function except that it yields expression in the function.

Q.21 How will you copy an object in Python?

In order to copy an object in Python, we should try `copy.copy()` or `copy.deepcopy()` for the general case. Since we cannot copy all objects but yes most of them.

Q.22 What do you understand negative index in Python

Python sequences are indexed in positive and negative numbers as well. Therefore for a positive index, 0 is the first index, 1 is the second index and so forth. On the other hand for negative index, (-1) is the last index and (-2) is the second last index and so forth.

Q.23 How will you convert a number to a string?

We must use the inbuilt function `str()` in order to convert a number into a string. In case you want a octal or hexadecimal representation, use the inbuilt function `oct()` or `hex()`.

Q.24 Differentiate between Xrange and range?

The primary difference is that `Xrange` returns the `xrange` object while `range` returns the list, and uses the same memory and no matter what the range size is.

Q.25 What do you understand by module and package in Python?

Module is the way to structure program such that each Python program file is a module, which imports other modules like objects and attributes. On

the other hand the folder of Python program is a package of modules. Such that a package can have modules or subfolders.

Q.26State the process to share global variables across modules?

In order to share global variables across modules within a single program, you must first create a special module. Then Import the config module in all modules of your application. Thereafter the module will be available as a global variable across modules.

Q.27What is Flask and list down its benefits?

Flask is defined as a web micro framework for Python that is based on "Werkzeug, Jinja 2 and good intentions" BSD licensed. Werkzeug and jingja are two of its dependencies. Since flask is part of the micro-framework. therefore it will have little to no dependencies on external libraries. Also it makes the framework light while there is little dependency to update and less security bugs.

Q.28Is python interpreted or compiled?

Python as a programming language has no saying about if it's a compiled or interpreted programming language, only the implementation of it. The terms interpreted or compiled is not a property of the language but a property of the implementation. Python program runs directly from the source code. so, Python will fall under byte code interpreted. The .py source code is first compiled to byte code as .pyc. This byte code can be interpreted (official CPython), or JIT compiled (PyPy). Python source code (.py) can be compiled to different byte code also like IronPython (.Net) or Jython (JVM).

Q.29What is PEP?

PEP stands for Python Enhancement Proposal. A PEP is a design document providing information to the Python community, or describing a new feature for Python or its processes or environment.

Q.30How is memory allocated for variables in python?

Due to Python's dynamic nature. The answer is one of:

- the byte size of the name of a variable as a string, plus two bytes/pointers
- one byte/pointer plus one byte/pointer for every unique attribute
- the memory of your entire application >more than the current memory of your application
- something else, depending on your definition of a variable

Q.31What is the difference between python arrays and lists.

Arrays can hold only a single data type element whereas lists can hold any data type elements.

Q.32What are negative indices used for?

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 the correct order.

Q.33Why do we use numpy mainly?

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

- Less Memory
- Fast
- Convenient

Q.34What is dictionary in python?

Dictionary in Python is an unordered collection of data values, used to store data values like a map, which unlike other Data Types that hold only single value as an element, Dictionary holds key:value pair.

Q.35What are *args and **kwargs?

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.

Q.36Does python support multi-threading? How do we achieve that?

Python does not support multithreading natively. However, there are libraries to add support of multithreading into python but it is generally not a good idea in Python.

Q.37What is monkey patching and mention its most common use?

Monkey patching is changing the behavior of a function or object after it has already been defined.

Most of the time it's a pretty terrible idea - it is usually best if things act in a well-defined way. One reason to monkey patch would be in testing. The mock package is very useful to this end.

Q.38What are the naming conventions of variables in python?

- Only use lower case while naming a Package in Python.
- Python loves the small case alphabets and hence suggests Module names should also follow the suite.
- Classes in Python subject to a different naming scheme called as CapWords convention.
- Global variables must all be in lowercase.
- Every instance method should have 'self' as its first argument.
- All class methods should have 'cls' as their first argument.
- All constants must be CAPITALISED completely.

Q.39How do we specify private and protected access specifiers?

We use double underscore as the beginning for a private variable name and single underscore for protected.

Q.40How will you manage different versions of your code?

Everyone should be knowing and using version management service like git or github as it will help a lot when working in large projects with big teams.

Q.41Explain python's garbage collection mechanism.

- Python maintains a count of the number of references to each object in memory. If a reference count goes to zero then the associated object is no longer live and the memory allocated to that object can be freed up for something else.
- Occasionally things called "reference cycles" happen. The garbage collector periodically looks for these and cleans them up. An example would be if you have two objects o1 and o2 such that o1.x == o2 and o2.x == o1. If o1 and o2 are not referenced by anything else then they shouldn't be live. But each of them has a reference count of 1.
- Certain heuristics are used to speed up garbage collection. For example, recently created objects are more likely to be dead. As objects are created, the garbage collector assigns them to generations. Each object gets one generation, and younger generations are dealt with first.

Q.42Write a program to swap two numbers?

a, b = b, a

Q.43What is slicing in python?

It is a single expression anonymous function often used as an inline function.

Q.44How do you use slicing operation to reverse a string?

original_string[::-1]

Q.45How do append() and extend() differ in a list?

Append is used to add elements. Extend is use to concatenate lists.

Q.46How do we write a multiline string?

By using the escape character backslash.

Eg. String="Hello I am a string\\

And I continue that string."

Q.47Find the number with maximum frequency in a list using single line of code.

max(lst,key=lst.count)

Q.48How do you give arguments to a program via a terminal?

By importing system module and then using argv variable.

Q.49What is the use of lambda function?

It is a single expression anonymous function often used as inline function.

Q.50Mention a few main differences in python 2.0 and 3.0?

- Division: 2.x returns integer, 3.x always gives float
- Print: 3.x always requires parenthesis
- Unicode: In Python 2, implicit str type is ASCII. But in Python 3.x implicit str type is Unicode

Q.51Given two sequences, how do you make a dictionary?

dict (zip(t1,t2))

here t1 and t2 are the two sequences. Note: both sequences should be of same length.

Q.52What are accessors, mutators, and @property?

What we call getters and setters in languages like Java, we term accessors and mutators in Python. In Java, if we have a user-defined class with a property 'x', we have methods like getX() and setX(). In Python, we have

`@property`, which is syntactic sugar for `property()`. This lets us get and set variables without compromising on the conventions.

Q.53How can you imitate switch-case behavior in python?

We use dictionary to approach to the solution of this problem. For ex.:

```
def switch_demo(argument):
```

```
    switcher = {
```

```
        1: "January",
```

```
        2: "February",
```

```
        3: "March",
```

```
        4: "April",
```

```
        5: "May",
```

```
        6: "June",
```

```
        7: "July",
```

```
        8: "August",
```

```
        9: "September",
```

```
        10: "October",
```

```
        11: "November",
```

```
        12: "December"
```

```
}
```

```
print switcher.get(argument, "Invalid month")
```

Q.54Write a program in Python to produce Star triangle.

```
def pyfunc(r):
```

```
    for x in range(r):
```

```
        print("*"*(r-x-1)+'*'*(2*x+1))
```

Q.55Write an efficient one-liner that will count the number of capital letters in a file.

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

Q.56Compare Django and flask.

- Flask provides simplicity, flexibility and fine-grained control. It is unopinionated (it lets you decide how you want to implement things).
- Django provides an all-inclusive experience: you get an admin panel, database interfaces, an ORM, and directory structure for your apps and projects out of the box.

Q.57What is map function in Python?

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

Q.58

What is the difference between NumPy and SciPy?

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

Q.59 Write a program for binary search.

```
def BinarySearch(lys, val):  
  
    first = 0  
  
    last = len(lys)-1  
  
    index = -1  
  
    while (first <= last) and (index == -1):  
  
        mid = (first+last)//2  
  
        if lys[mid] == val:  
  
            index = mid  
  
        else:  
  
            if val<lys[mid]:  
  
                last = mid -1
```

```
    else:  
  
        first = mid +1  
  
    return index
```

Q.60Write a program for interpolation search.

```
def InterpolationSearch(lys, val):  
  
    low = 0  
  
    high = (len(lys) - 1)  
  
    while low <= high and val >= lys[low] and val <= lys[high]:  
  
        index = low + int(((float(high - low) / ( lys[high] - lys[low])) * ( val -  
lys[low])))  
  
        if lys[index] == val:  
  
            return index  
  
        if lys[index] < val:  
  
            low = index + 1;  
  
        else:  
  
            high = index - 1;  
  
    return -1
```

Q.61What is JSON? Describe in brief how you'd convert JSON data into Python data?

JSON stands for JavaScript Object Notation.

JSON is generally built on the following two structures:

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

We use load() function from JSON module in python.

Q.62 Is python good for image processing?

It is highly dependant on what we mean by image processing here but a good starting point will be PIL library. It can be used to easily import images, do basic operations like rotation and resizing and saving images.

Q.63 Why is python so famous for ML?

This is because of a number of reason, the main points being:

- It has most of the libraries used for ML
- All the heavy backend is done in c or fortran hence speed issue does not come up
- It also has gained a historical consistency in this sector
- Python codes have great readability and helps in concentrating more in logic than syntax
- Python codes are in general much shorter which means lesser development time

Q.64 Compare Juia, R and Python.

Julia is the fastest language, so much so that a code which takes 5 seconds in Julia takes more than half an hour in python or R.

R is mainly for data scientist and not much anticipated for general purpose programming, hence for programmers, learning R is a bit of limiting.

Python has a balance between usability and speed. Its versatility makes it the first choice of programmers.

Q.65 What do you think about the future of python programming language?

According to all the current researches, the future (at least the near future) is going to be Artificial Intelligence and Machine Learning. Python is a programming language getting popular day by day due to its reduced complexity and the freedom it provides to work in various sectors of programming.

And because of high usage of python in the above field, it is considered one of the best languages to design a project related to AI and Machine Learning (although there are other things too used for that). So, Python serves to be the best ingredient of the dessert.

