PYTHON INTERVIEW QUESTIONS

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.

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

```
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
NoneType	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
int	Stores integer literals including hex, octal and binary numbers as integers
float	Stores literals containing decimal values and/or exponent signs as floating-point numbers
complex	Stores complex numbers in the form (A + Bj) and has attributes: real and imag
bool	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
list	Mutable sequence used to store collection of items.
tuple	Immutable sequence used to store collection of items.
range	Represents an immutable sequence of numbers generated during execution.
str	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
dict	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
set	Mutable unordered collection of distinct hashable objects.
frozenset	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 hierarchial structuring of the module namespace using **dot notation**. As, **modules** help avoid clashes between global variable names, in a similar manner, **packages** help avoid clashes between module names. Creating a package is easy since it makes use of the system's inherent file structure. So just stuff the modules into a folder and there you have it, the folder name as the package name. Importing a module or its contents from this package requires the package name as prefix to the module name joined by a dot.

Note: You can technically import the package as well, but alas, it doesn't import the modules within the package to the local namespace, thus, it is practically useless.

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

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

```
# 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 sop.
- **step** is the number of steps to jump.
- Default value for **start** is 0, **stop** is number of items, **step** is 1.
- Slicing can be done on **strings, arrays, lists**, and **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**

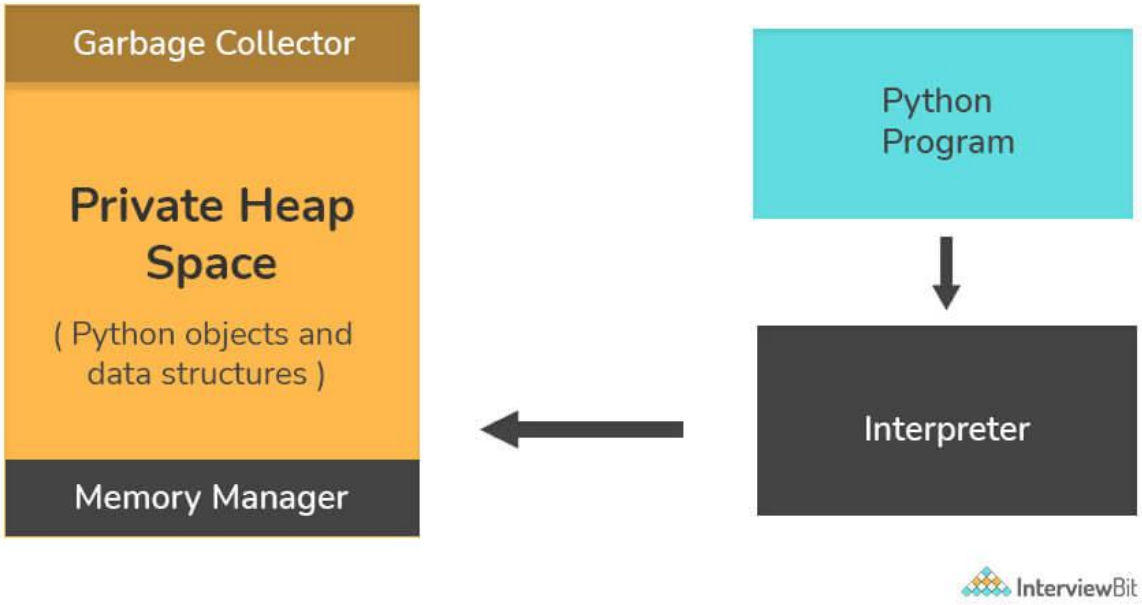
18. What is the difference between Python Arrays and lists?

- 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
```

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.



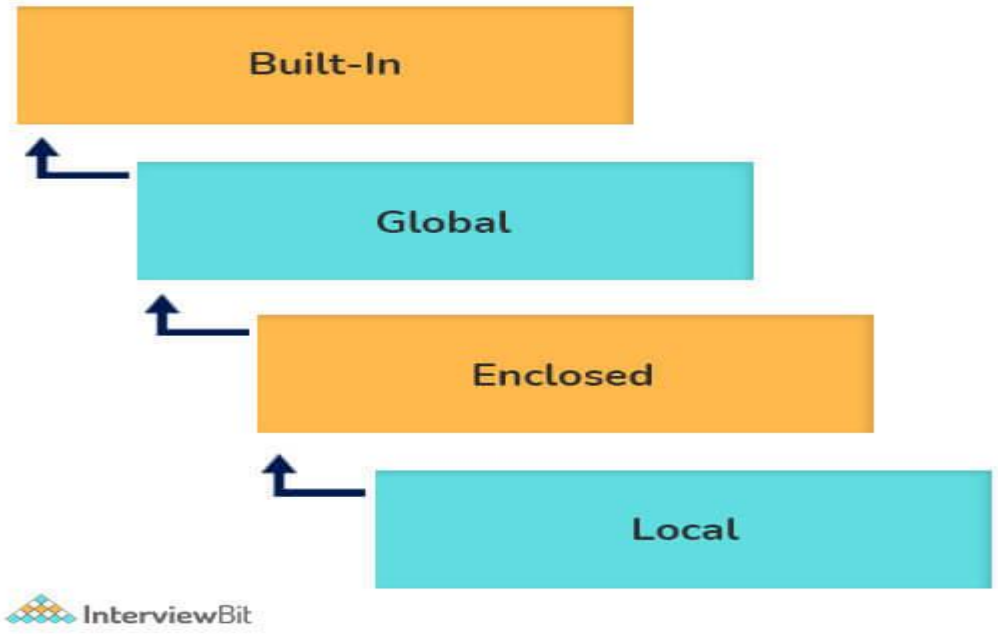
20. What are Python namespaces? Why are they used?

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 o 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.

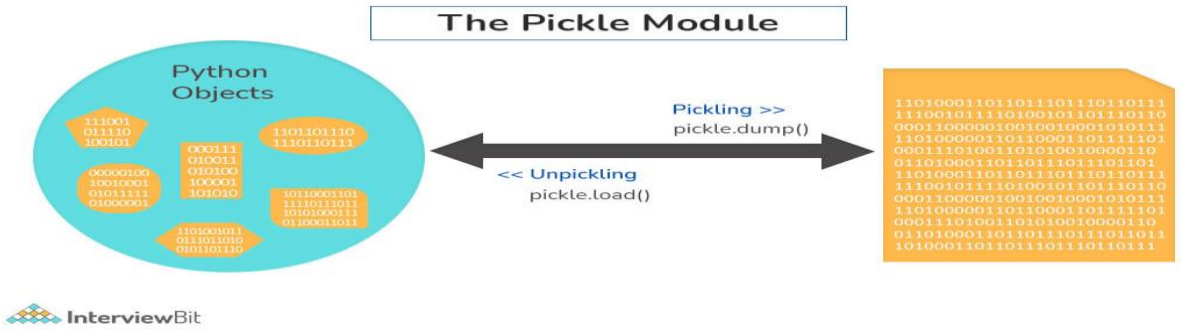
Pickling:

- 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**.



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.']
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
```

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.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:

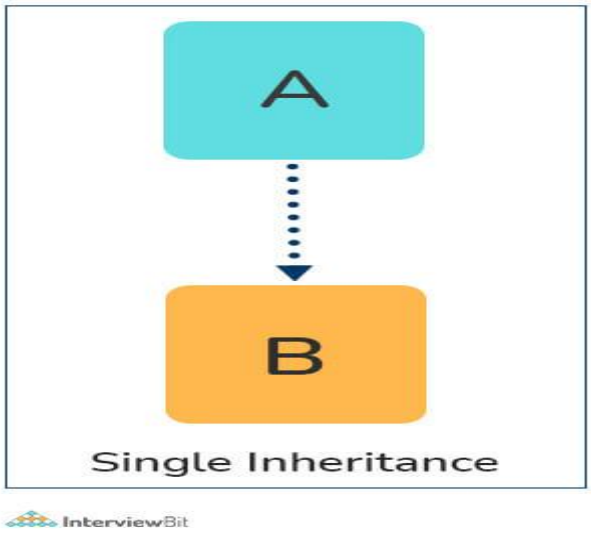
```
class InterviewbitEmployee:
  def __init__(self, emp_name):
     self.emp_name = emp_name

  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:

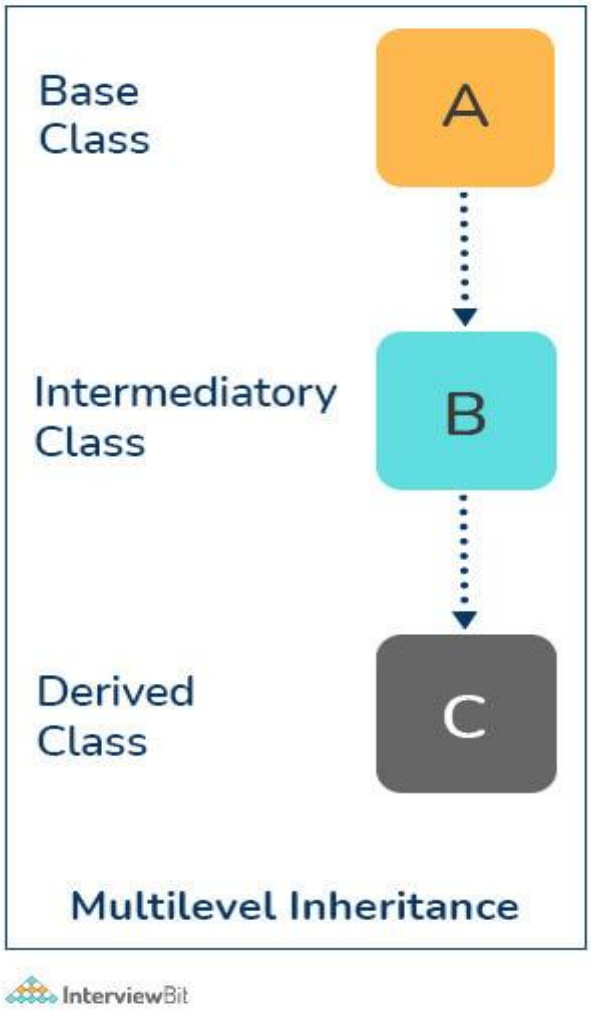
- **Single Inheritance**: Child class derives members of one parent class.




```
# 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()
```

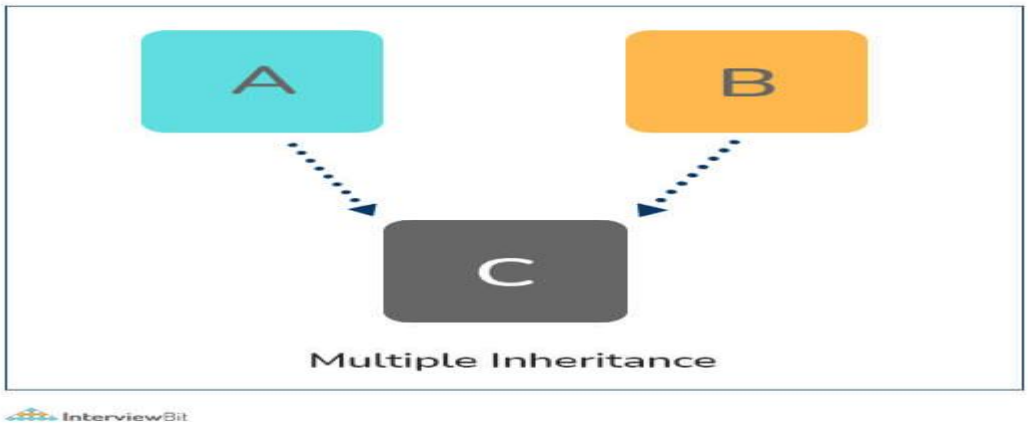
- **Multi-level Inheritance:** The members of the parent class, A, are inherited by child class which is then inherited by another child class, B. The features of the base class and the derived class are further inherited into the new derived class, C. Here, A is the grandfather class of class C.




```
# 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()
```

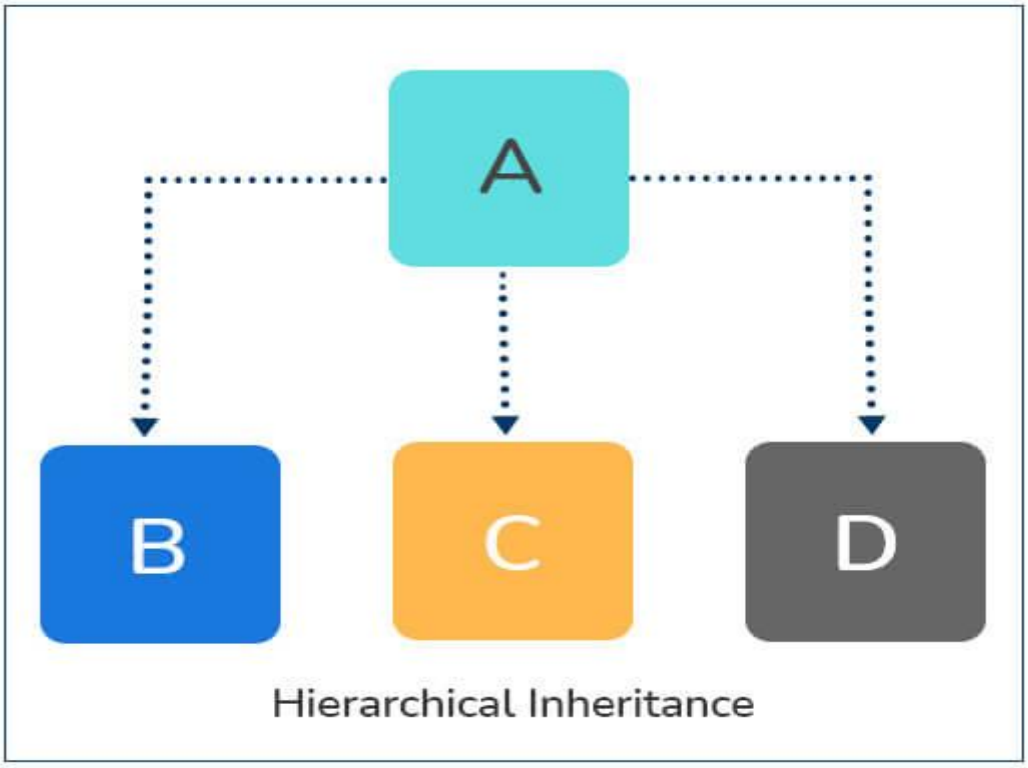
- **Multiple Inheritance:** This is achieved when one child class derives members from more than one parent class. All features of parent classes are inherited in the child class.




```
# 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.



InterviewBit

```
# 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
```

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 combines 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
```

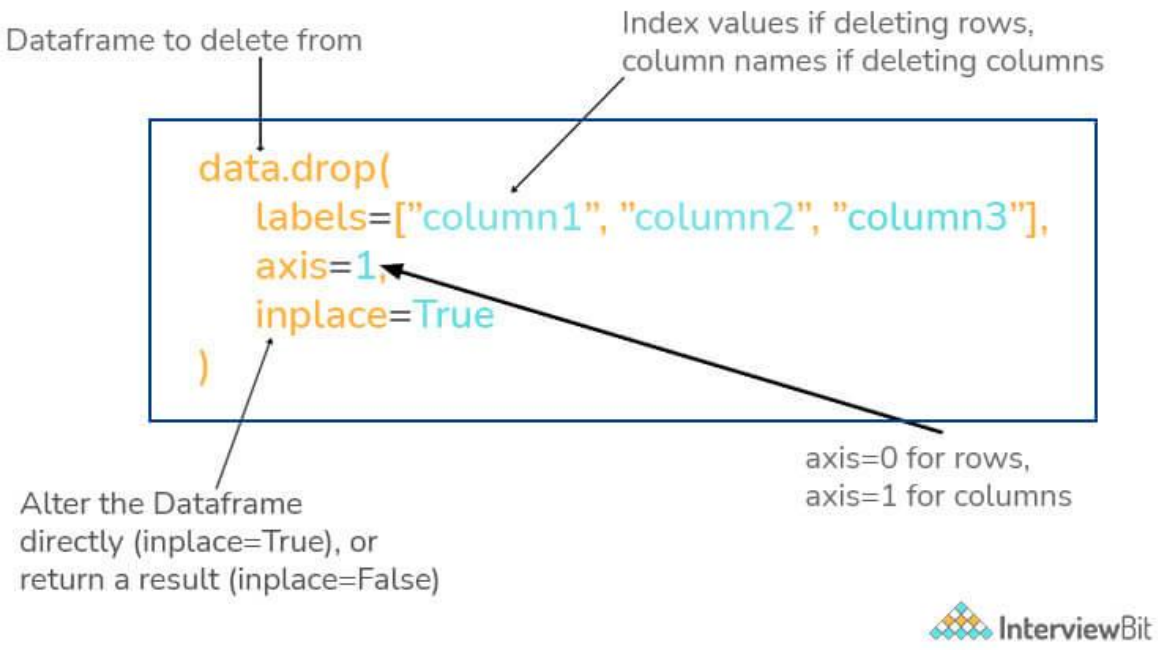
- To drop the index name "Names":

```
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.



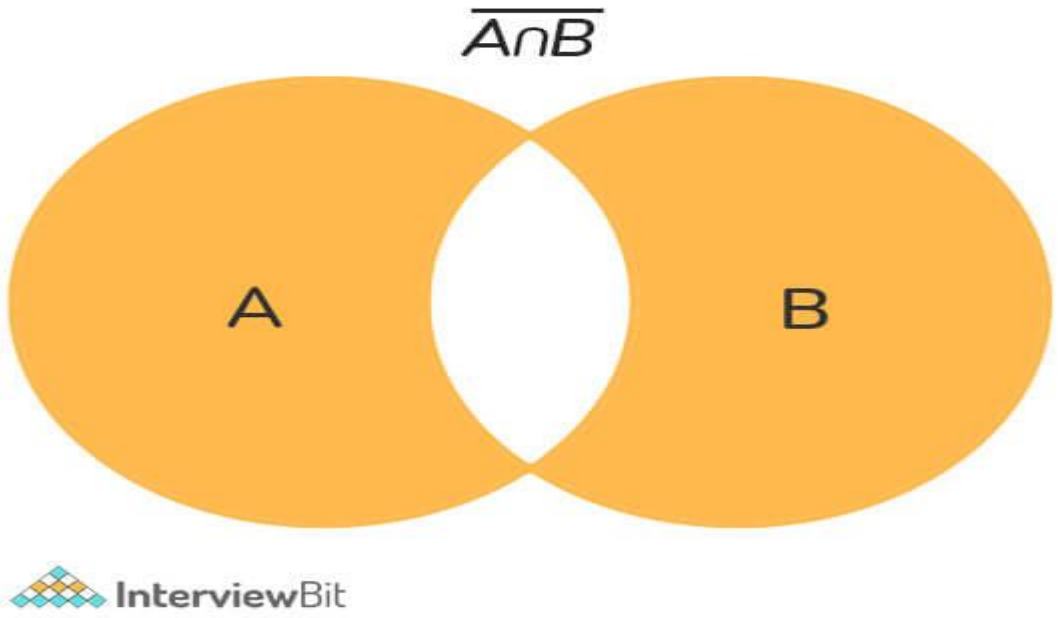
57. Can you get items of series A that are not available in another series B?

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
"""
```

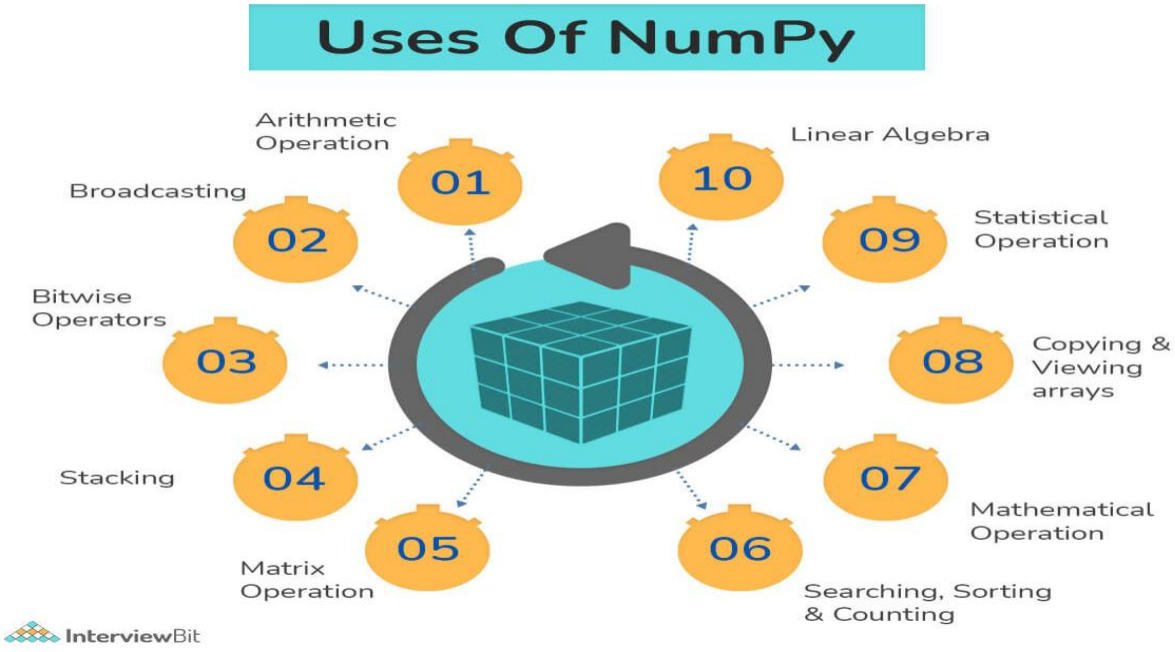
59. While importing data from different sources, can the pandas library recognize dates?

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)
```

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:



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,)
"""
```

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
```

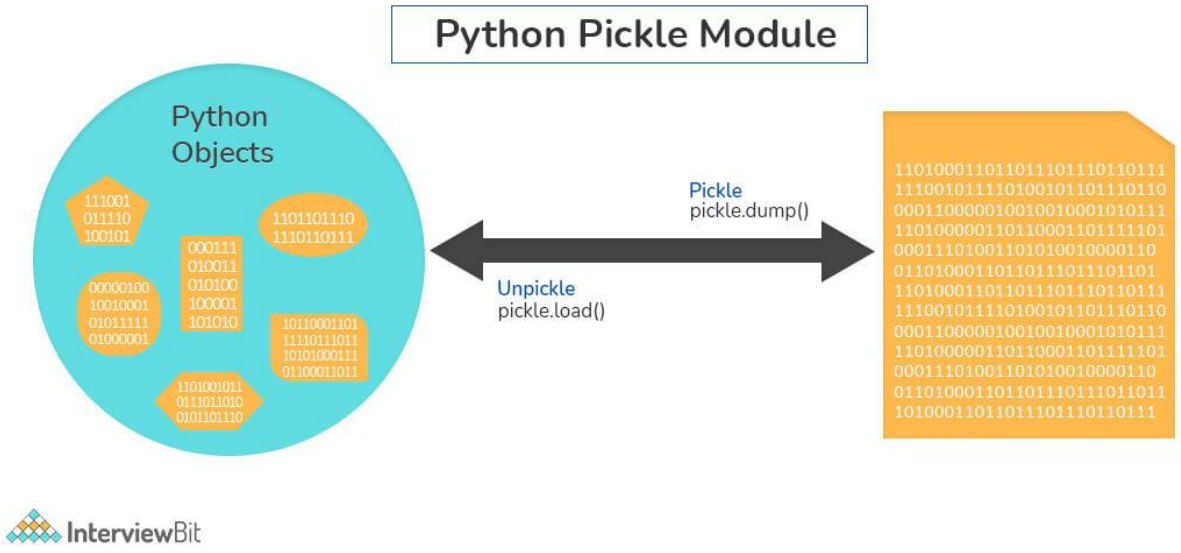
Another way is to use match() method from the re (regex) module as shown:

```
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?

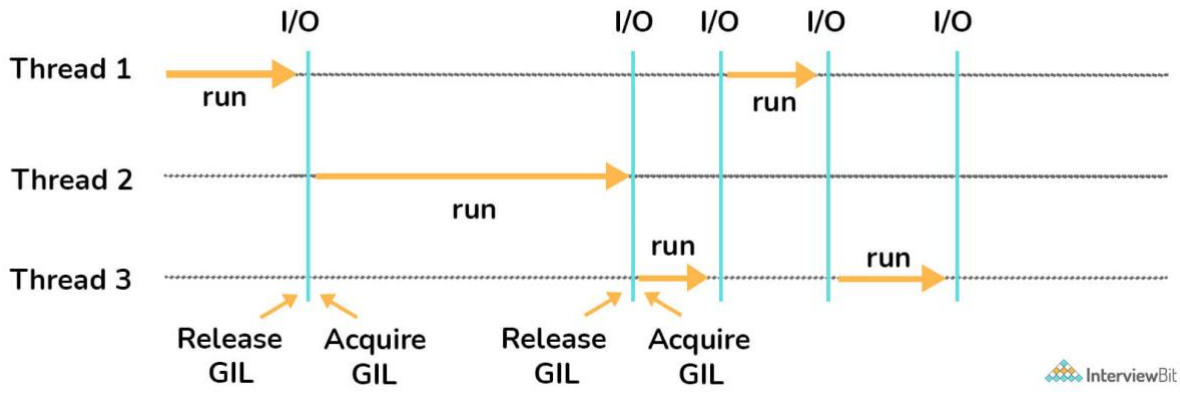
Pickling is the conversion of python objects to binary form. Whereas, unpickling is the conversion of binary form data to python objects. The pickled objects are used for storing in disks or external memory locations. Unpickled objects are used for getting the data back as python objects upon which processing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

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:

```
ax + by = c
mx + ny = o
```

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})', '\\3-\\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.StringIO(requests.get(csv_link).content))
dataframe = pd.read_csv(data_source)
print(dataframe.head())
```

2) Why Python?

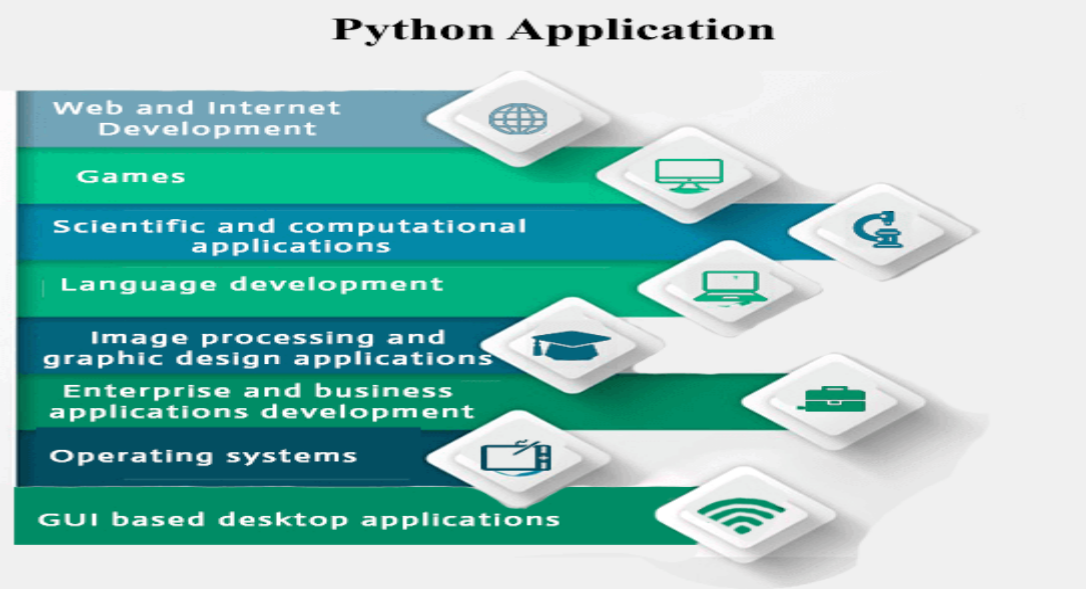
- Python is an interpreted, object-oriented, high-level programming language with dynamic semantics.
- Python is compatible with different platforms like **Windows, Mac, Linux, Raspberry Pi,** etc.
- Python has a simple syntax as compared to other languages.
- Python allows a developer to write programs 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 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

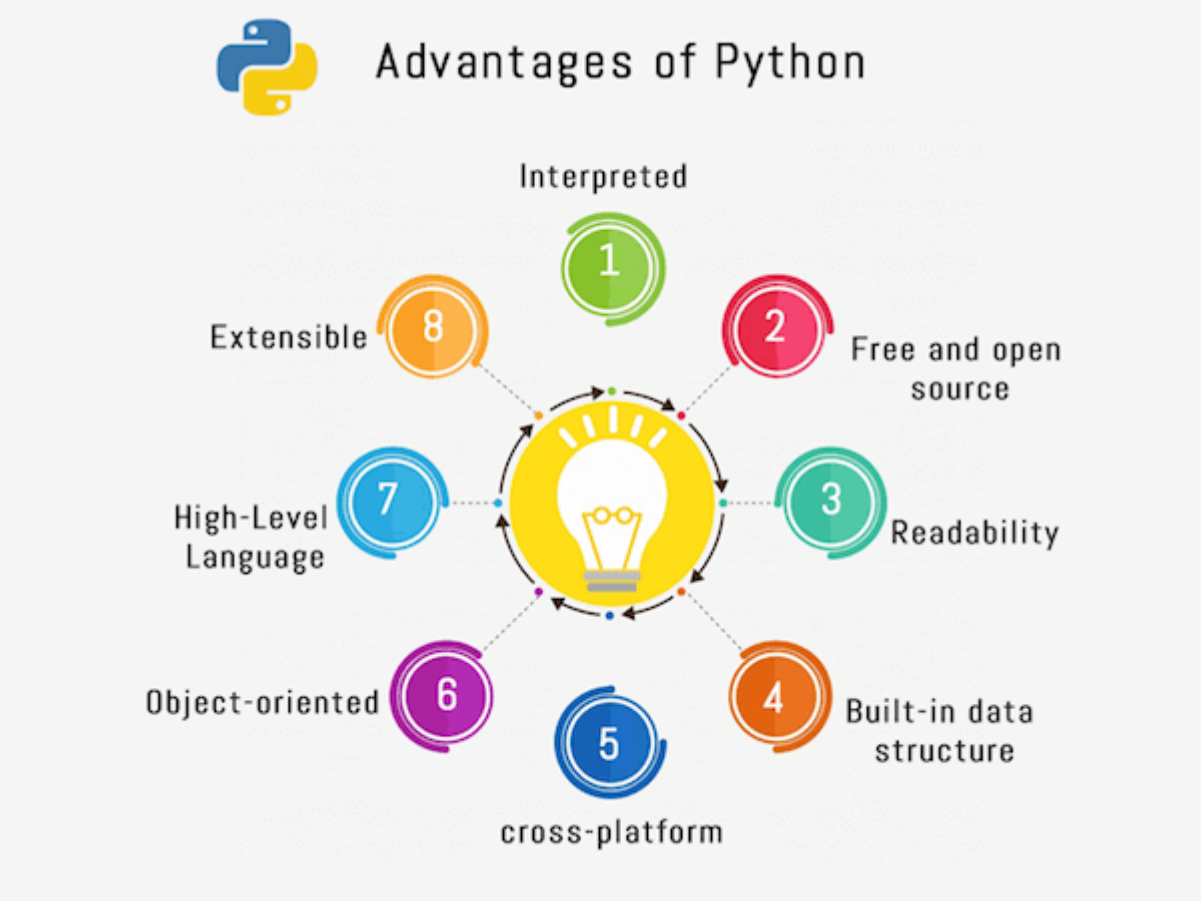
Object-oriented: Python allows to implement the Object-Oriented concepts to build application solution.

- 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.



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:

```
# in single quotes
single = 'JavaTpoint'
# in double quotes
double = "JavaTpoint"
# multi-line String
multi = """Java
            T
                point"""

print(single)
print(double)
print(multi)
```

Output:

```
JavaTpoint
JavaTpoint
Java
        T
            point
```

Numeric Literals

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

Example:

```
# Integer literal
```

```
a = 10
#Float Literal
b = 12.3
#Complex Literal
x = 3.14j
print(a)
print(b)
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:

```
p = (1 == True)
q = (1 == False)
r = True + 3
s = False + 7

print("p is", p)
print("q is", q)
print("r:", r)
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

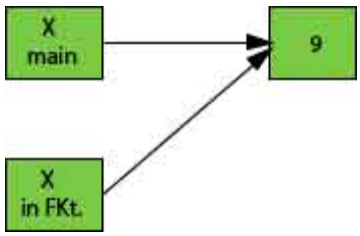
It returns an iterator from two or more iterators.

9) What is Python's parameter passing mechanism?

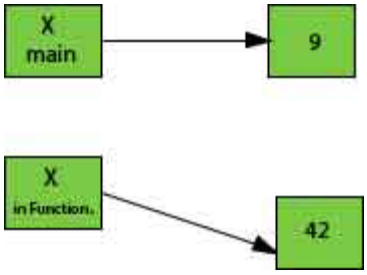
There are two parameters passing mechanism in Python:

- Pass by references
- 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 it's 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:

```
class student:
    def __init__(self, name):
        self.name = name
    def __init__(self, name, email):
        self.name = name
        self.email = email

# This line will generate an error
#st = student("rahul")

# This line will call the second constructor
st = student("rahul", "rahul@gmail.com")
print("Name: ", st.name)
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:

```
list_1 = [ 3, 5, 7, 3, 9, 3 ]
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 providies 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?

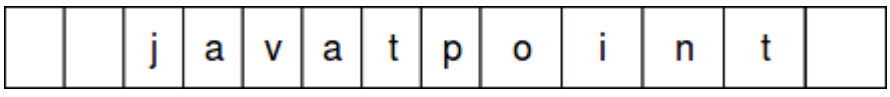
To remove leading characters from a string, we can use lstrip() function. It is Python string function which takes an optional char type parameter. If a parameter is provided, it removes the character. Otherwise, it removes all the leading spaces from the string.

Example:

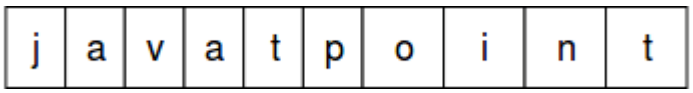
```
1. string = "  javatpoint "
2. string2 = "    javatpoint       "
3. print(string)
4. print(string2)
5. print("After stripping all leading whitespaces:")
6. print(string.lstrip())
7. print(string2.lstrip())
```

Output:

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



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



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:

```
str = "Rohan"
str2 = "ab"
# Calling function
str2 = str.join(str2)
# Displaying result
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:

```
# import the random module
import random
# declare a list
sample_list1 = ['Z', 'Y', 'X', 'W', 'V', 'U']
print("Original LIST1: ")
print(sample_list1)
# first shuffle
random.shuffle(sample_list1)
print("\nAfter the first shuffle of LIST1: ")
print(sample_list1)
```

```
# second shuffle
random.shuffle(sample_list1)
print("\nAfter the second shuffle of LIST1: ")
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.

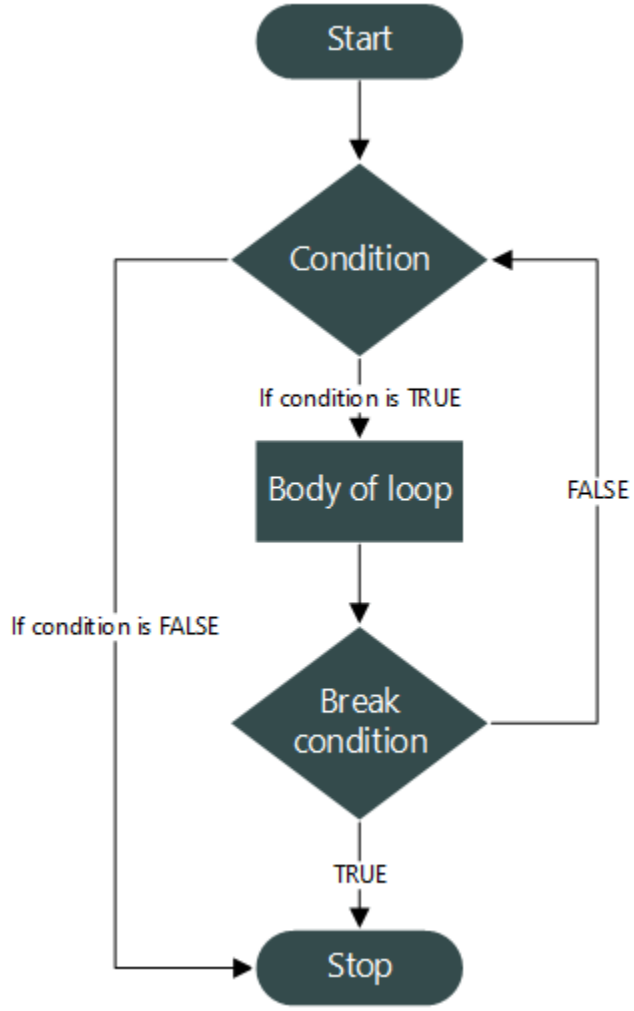
Example:

```
list_1 = ['X', 'Y', 'Z']
list_2 = [11, 22, 33]
for i in list_1:
    for j in list_2:
        print(i, j)
        if i == 'Y' and j == 33:
            print('BREAK')
            break
    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.

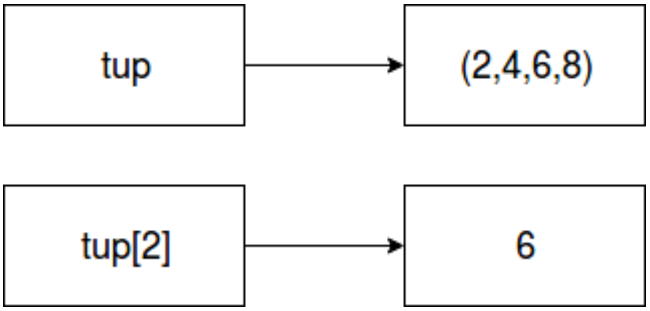
To create a tuple, we can declare it as below.

Example:

```
# Declaring tuple
tup = (2,4,6,8)
# Displaying value
print(tup)

# Displaying Single value
print(tup[2])
```

Output:

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



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

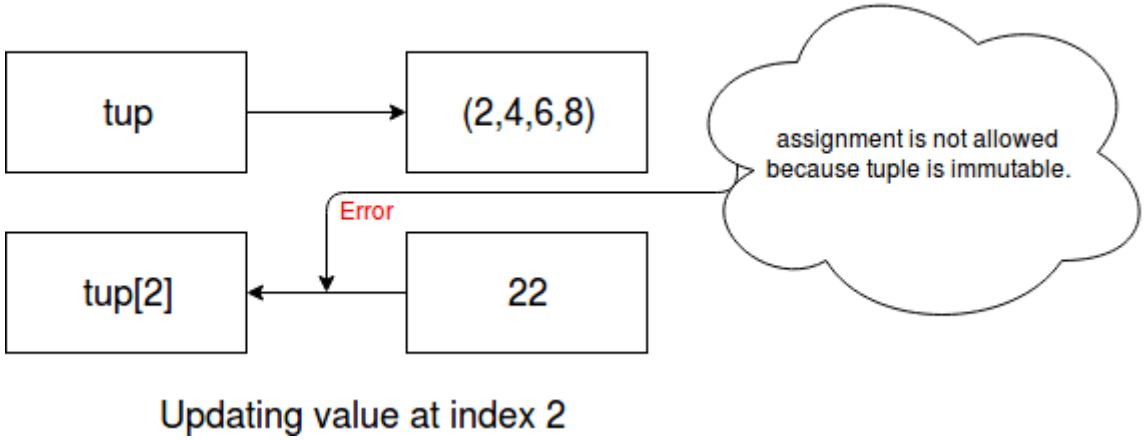
Example:

```
# Declaring tuple
tup = (2,4,6,8)
# Displaying value
print(tup)

# Displaying Single value
print(tup[2])

# 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)
```



Updating value at index 2

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?

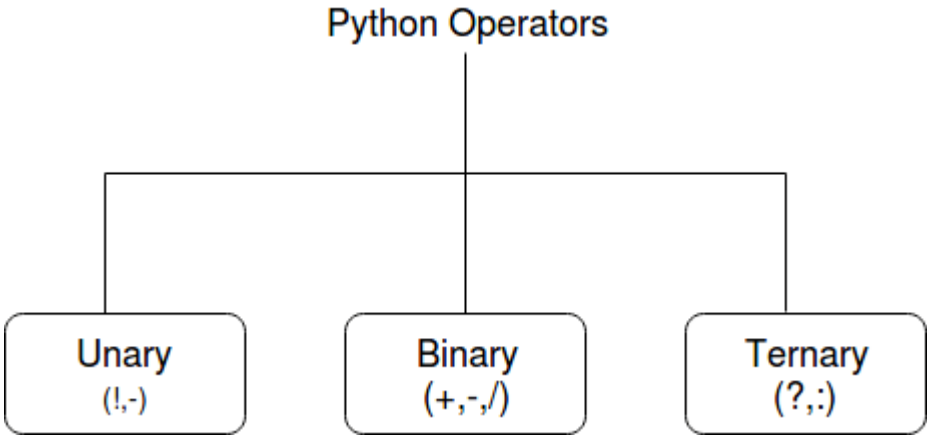
Python provides **four** modes to open files. The read-only (r), write-only (w), read-write (rw) and append mode (a). 'r' is used to open a file in read-only mode, 'w' is used to open a file in write-only mode, 'rw' is used to open in reading and write mode, 'a' is used to open a file in append mode. 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.
- 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:

```
# Unary Operator
A = 12
B = -(A)
print (B)
# Binary Operator
A = 12
B = 13
print (A + B)
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 OperatorsRelational Operators
- Assignment Operators
- Logical Operators
- Membership Operators
- Identity Operators
- Bitwise Operators

Arithmetic operators perform basic arithmetic operations. For example "+" is used to add and "?" is used for subtraction.

Example:

```
# Adding two values
print(12+23)
# Subtracting two values
print(12-23)
# Multiplying two values
print(12*23)
# Dividing two values
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:

```
a, b = 10, 12
print(a==b) # False
print(a<b) # True
print(a<=b) # True
print(a!=b) # True
```

Output:

```
False
True
True
True
```

Assignment operators are used to assigning values to the variables. See the examples below.

Example:

```
# Examples of Assignment operators
a=12
print(a) # 12
a += 2
print(a) # 14
a -= 2
print(a) # 12
a *=2
print(a) # 24
a **=2
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:

```
# Logical operator examples
a = True
b = False
print(a and b) # False
print(a or b) # True
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:

```
# Membership operators examples
list = [2,4,6,7,3,4]
print(5 in list) # False
cities = ("india","delhi")
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 art.title.encode("utf-8") function.

Example:

```
1. unicode_1 = ("\u0123", "\u2665", "\U0001f638", "\u265E", "\u265F", "\u2168"
   )
2. print (unicode_1)
```

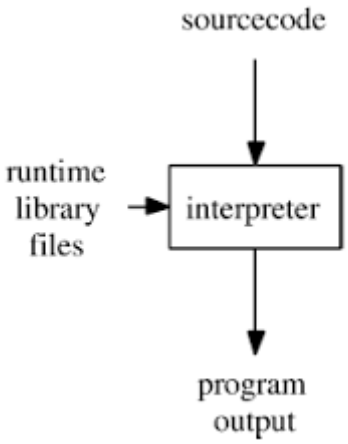
Output:

```
unicode_1: ('ģ', '♥', '😸', '♞', '♟', '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:

```
def function_is_called():
    def function_is_returned():
        print("JavaTpoint")
    return function_is_returned
new_1 = function_is_called()
# Outputs "JavaTpoint"
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:

```
JavaTpoint Local
```

28) What is the namespace in Python?

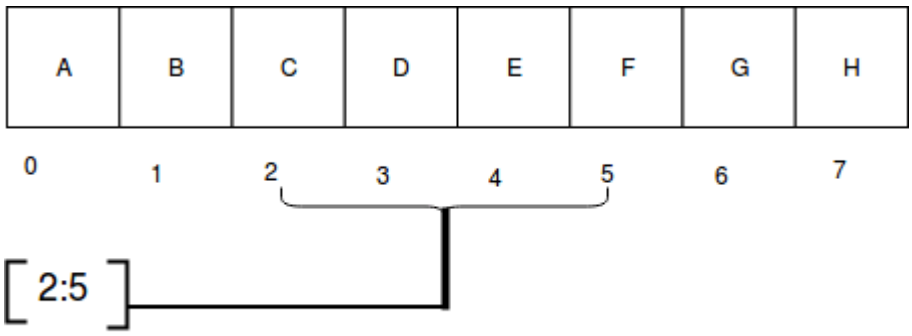
The namespace is a fundamental idea to structure and organize the code that is more useful in large projects. However, it could be a bit difficult concept to grasp if you're new to programming. Hence, we tried to make namespaces just a little easier to understand.

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.

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:

```
vaTpoint, Python Interv
```

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:

```
# One-line docstrings
def hello():
    """A function to greet."""
    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 uses as first index and '1' as the second index and the process go 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 iinto another.

int() - converts any data type into integer type

float() - converts any data type into float type

ord() - converts characters into integer

hex() - converts integers to hexadecimal

oct() - converts integer to octal

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

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

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

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

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

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

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:

```
import smtplib
# Calling SMTP
s = smtplib.SMTP('smtp.gmail.com', 587)
# TLS for network security
s.starttls()
# User email Authentication
s.login("sender@email_id", "sender_email_id_password")
# Message to be sent
message = "Message_sender_need_to_send"
```

```
# Sending the mail
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:

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

```
def New_func():
    print ("Hi, Welcome to JavaTpoint")
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).Iyt 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.

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 x range returns an xrange object.

This means that xrange doesn't actually generate a static list at run-time like range does. It creates the values as you need them with a special technique called yielding. This technique is used with a type of object known as generators. That means that if you have a really gigantic range you'd like to generate a list for, say one billion, xrange is the function to use.

This is especially true if you have a really memory sensitive system such as a cell phone that you are working with, as range will use as much memory as it can to create your array of integers, which can result in a Memory Error and crash your program. It's a memory hungry beast.

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.

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.

Q6.What are the benefits of using Python?

Ans: The benefits of using python are-

1.
    0. **Easy to use**– Python is a high-level programming language that is easy to use, read, write and learn.
    1. **Interpreted language**– Since python is interpreted language, it executes the code line by line and stops if an error occurs in any line.
    2. **Dynamically typed**– the developer does not assign data types to variables at the time of coding. It automatically gets assigned during execution.
    3. **Free and open source**– Python is free to use and distribute. It is open source.
    4. **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).
    5. **Portable**– Python programs can run on any platform without requiring any change.
    6. The data structures used in python are user friendly.
    7. It provides more functionality with less coding.

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.

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-

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

The above code creates a list as below-

```
4

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

Example of dictionary comprehension is-

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

The above code creates a list as below-

```
[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**.

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**– They 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.

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.

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:

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

Output: 5

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

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

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.

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:

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

Output: Hi, Welcome to Edureka

Q31. How does break, continue and pass work?

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

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

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

For example:

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

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

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:

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

Output: abcd

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:

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

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]x, y = 25, 50big = x if x < y else y

Example:

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

Q50. What does len() do?

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

Example:

```
1stg='ABCD'
2len(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.

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:

```
import os
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

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:

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

Output:

array(‘d’, [1.1, 2.1, 3.1, 3.4])

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

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

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 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 allows the new extensions to be compiled properly without any error and the linking can be done only when it passes the compiled procedure. If the dynamic loading is used then it depends on the style that is being

provided with the system. The python interpreter can be used to provide the dynamic loading of the configuration setup files and will rebuild the interpreter.

The steps that are required in this as:

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

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:

```
a="edureka python"
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:

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

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 in inherited from base class base1, and d2 are inherited from base2.
3. **Hierarchical inheritance** – from one base class you can inherit any number of child classes
4. **Multiple inheritance** – a derived class is inherited from more than one base class.

Q67. How are classes created in Python?

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

Example:

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

Output: abc

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.

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 in 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-

```
class a:
  pass
obj=a()
obj.name="xyz"
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.

e powerful VTK engine.

Question: What are the positive and negative indices?

Answer: In the positive indices are applied the search beings from left to the right. In the case of the 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 will be -1.

Question: What can be the length of the identifier in Python?

Answer: The length of the identifier in Python can be of any length. The longest identifier will violate from PEP – 8 and PEP – 20.

Question: Define Pass statement in Python?

Answer: A Pass statement in Python is used when we cannot decide what to do in our code, but we must type something for making syntactically correct.

Question: What are the limitations of Python?

Answer: There are certain limitations of 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 in mobile computing.
4. It consists of an underdeveloped database access layer.

Question: Do runtime errors exist in Python? Give an example?

Answer: Yes, runtime errors exist in Python. For example, if you are duck typing and things look like a duck, then it is considered as a duck even if that is just a flag or stamp or any other thing. The code, in this case, would be A Run-time error. For example, Print “Hackr io”, then the runtime error would be the missing parenthesis that is required by print ( ).

Question: Can we reverse a list in Python?

Answer: Yes, we can reserve a list in Python using the reverse() method. The code can be depicted as follows.

```
def reverse(s):
  str = ""
  for i in s:
    str = i + str
  return str
```

Question: Why do we need a break in Python?

Answer: Break helps in controlling the Python loop by breaking the current loop from execution and transfer the control to the next block.

Question: Why do we need a continue in Python?

Answer: A continue also helps in controlling the Python loop but by making jumps to the next iteration of the loop without exhausting it.

Question: Can we use a break and continue together in Python? How?

Answer: Break and continue can be used together in Python. The break will stop the current loop from execution, while jump will take to another loop.

Question: Does Python support an intrinsic do-while loop?

Answer: No Python does not support an intrinsic do-while loop.

Question: How many ways can be applied for applying reverse string?

Answer: There are five ways in which the reverse string can be applied which include the following.

1. Loop
2. Recursion
3. Stack
4. Extended Slice Syntax
5. Reversed

Question: What are the different stages of the Life Cycle of a Thread?

Answer: The different stages of the Life Cycle of a Thread can be stated as follows.

- **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 wait until the calls to methods including join() and sleep() takes 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.

Question: What is the purpose of relational operators in Python?

Answer: The purpose of relational operators in Python is to compare values.

Question: What are assignment operators in Python?

Answer: The assignment operators in Python can help in combining all the arithmetic operators with the assignment symbol.

Question: Why do we need membership operators in Python?

Answer: We need membership operators in Python with the purpose to confirm if the value is a member in another or not.

Question: How are identity operators different than the membership operators?

Answer: Unlike membership operators, the identity operators compare the values to find out if they have the same value or not.

Question: Describe how multithreading is achieved in Python.

Answer: Even though Python comes with a multi-threading package, if the motive behind multithreading is to speed the code then using the package is not the go-to option.

The package has something called the GIL or Global Interpreter Lock, which is a construct. It ensures that one and only one of the threads execute at any given time. A thread acquires the GIL and then do some 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. Moreover, GIL passing adds to the overall overhead to the execution.

Hence, if you intend to use the threading package for speeding up the execution, using the package is not recommended.

Question: Draw a comparison between the range and xrange in Python.

Answer: 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 a very 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 beast. 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 crashing the program.

Question: Explain Inheritance and its various types in Python?

Answer: Inheritance enables a class to acquire all the 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.

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 or base class or the parent class. There are 4 forms of inheritance supported by Python:

- Single Inheritance – A single derived class acquires from on single superclass.
- Multi-Level Inheritance – At least 2 different derived classes acquire from two distinct base classes.
- Hierarchical Inheritance – A number of child classes acquire from one superclass
- Multiple Inheritance – A derived class acquires from several superclasses.

Question: What is the // operator? What is its use?

Answer: 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.

Question: What is the split function used for?

Answer: 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.

Question: Explain the Dogpile effect.

Answer: The event 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 value expires, the first process acquires the lock and starts generating new value.

Question: What is a pass in Python?

Answer: No-operation Python statement refers to pass. It is a place holder in the compound statement, where there should have a blank left or nothing written there.

Question: Is Python a case sensitive language?

Answer: Yes Python is a case sensitive language.

Question: Define slicing in Python.

Answer: Slicing refers to the mechanism to select the range of items from sequence types like lists, tuples, strings.

Question: What are docstring?

Answer: Docstring is a Python documentation string, it is a way of documenting Python functions, classes and modules.

Question: What is [::-1} used for?

Answer: [::-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]
```

Question: Define Python Iterators.

Answer: Group of elements, containers or objects that can be traversed.

Question: How are comments written in Python?

Answer: Comments in Python start with a # character, they can also be written within docstring(String within triple quotes)

Question: How to capitalize the first letter of string?

Answer: Capitalize() method capitalizes the first letter of the string, and if the letter is already capital it returns the original string

Question: What is, not and in operators?

Answer: Operators are functions that take two or more values and returns 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.

Question: How are files deleted in Python?

Answer: To delete a file in Python:

1. Import OS module
2. Use os.remove() function

Question: How are modules imported in Python?

Answer: Modules are imported using the import keyword by either of the following three ways:

```
import array
import array as arr
from array import *
```

Question: What is monkey patching?

Answer: Dynamic modifications of a class or module at run-time refers to a monkey patch.

Question: Does Python supports multiple inheritances?

Answer: Yes, in Python a class can be derived from more than one parent class.

Question: What does the method object() do?

Answer: The method returns a featureless object that is base for all classes. This method does not take any parameters.

Question: What is pep 8?

Answer: Python Enhancement Proposal or pep 8 is a set of rules that specify how to format Python code for maximum readability.

Question: What is namespace in Python?

Answer: A naming system used to make sure that names are unique to avoid naming conflicts refers to as Namespace.

Question: Is indentation necessary in Python?

Answer: 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.

Question: Define a function in Python

Answer: A block of code that is executed when it is called is defined as a function. Keyword def is used to define a Python function.

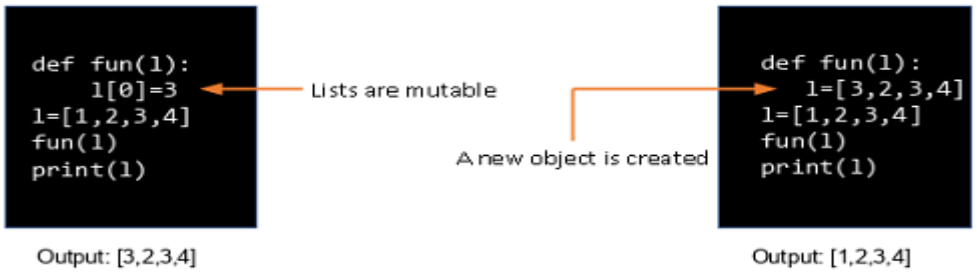
Question: Define self in Python

Answer: An instance of a class or an object is self in Python. It is included as the first parameter. It helps to differentiate between the methods and attributes of a class with local variables.

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.

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

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?

Learn data operations in Python, strings, conditional statements, error handling, and the commonly used Python web framework Django with the Python Training course.

16. What Is the Difference Between Del and Remove() on Lists?

del	remove()
• del removes all elements of a list within a given range • Syntax: del list[start:end]	• remove() removes the first occurrence of a particular character • Syntax: list.remove(element)

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()
• append() adds an element to the end of the list • Example - >>lst=[1,2,3] >>lst.append(4) >>lst Output:[1,2,3,4]	• extend() adds elements from an iterable to the end of the list • Example - >>lst=[1,2,3] >>lst.extend([4,5,6]) >>lst Output:[1,2,3,4,5,6]

21. What Is Docstring in Python?

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.

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.

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.

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
• 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	• An array is a sequence of objects of similar data type • An array within another array forms a matrix

40. What Is the Difference Between range() and xrange() Functions in Python?

range()	xrange()
• range returns a Python list object	• 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

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.

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

Python Tuples vs Lists – A Detailed Comparison

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
as	del	finally	in	or
assert	elif	for	is	pass
break	else	from	lambda	print
class	except	global	None	raise
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()
```

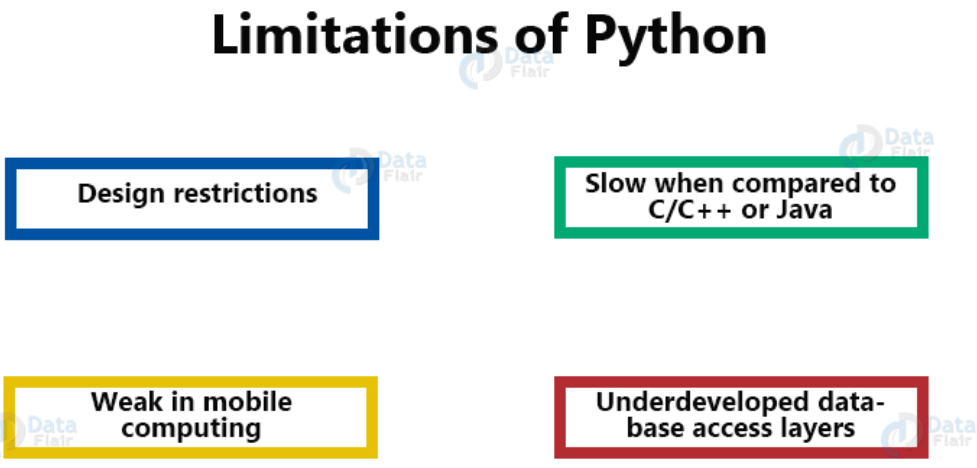
True

```
>>> '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:



- 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')
```

'}'

(Bonus: } – 125)

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=’abcdefghij’
>>> word[:3]+word[3:]
```

The output is ‘abcdefghij’. The first slice gives us ‘abc’, the next gives us ‘defghij’.

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
```

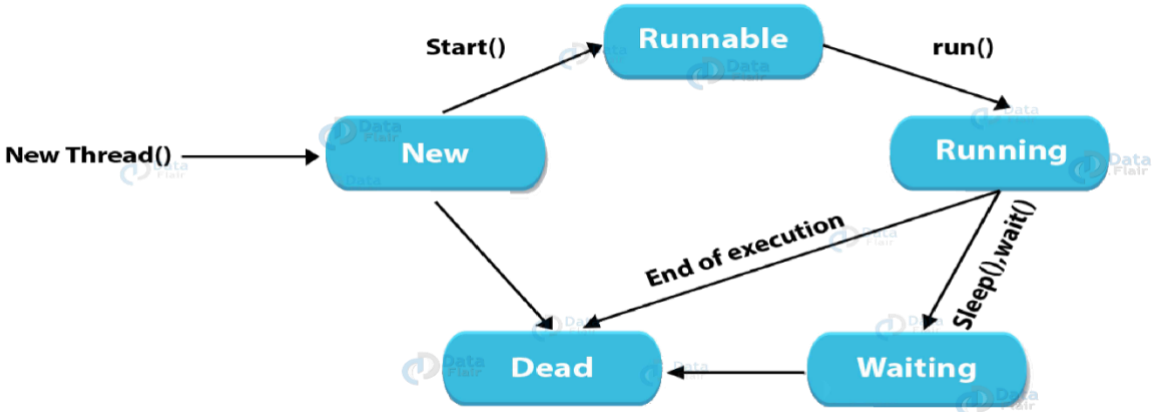
‘one two three four five six seven’

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.

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 (<=) If the value on the left is lesser than or equal to, it returns True.

>>> 3.0<=3

True

Greater than or equal to (>=) 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 (!=) 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
```

14

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 is 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

I love Python

I love Python

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'

Top Python Interview Questions and Answers

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'

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))
```

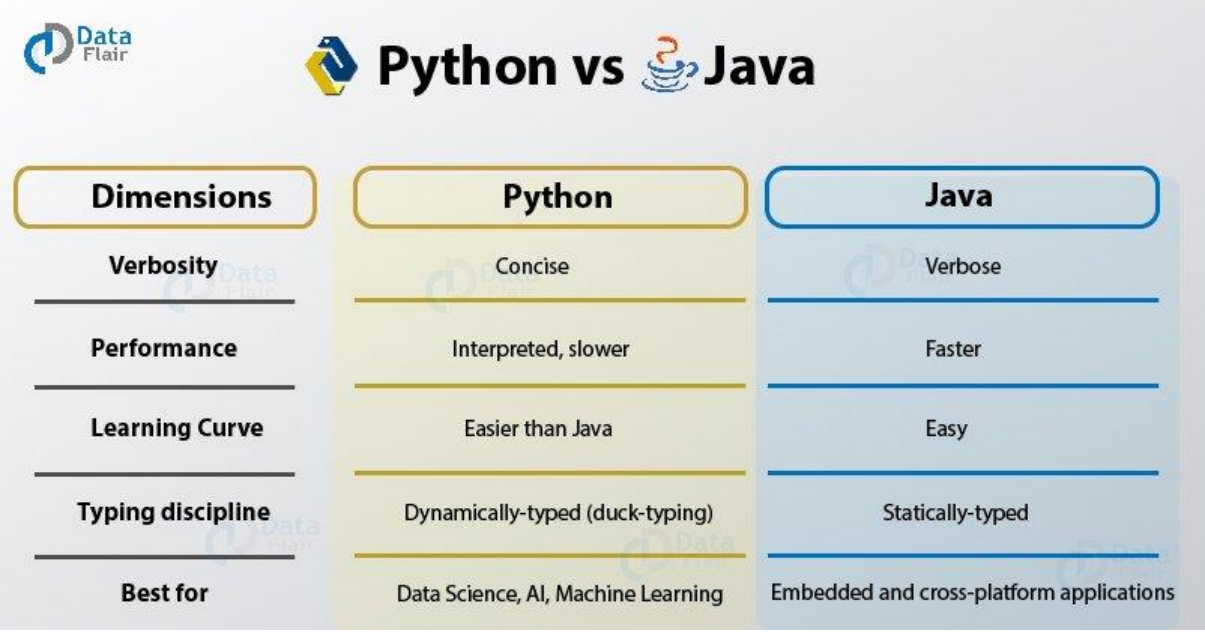
[2, 4, 6, 8]

```
>>> list(range(9,2,-1))
```

[9, 8, 7, 6, 5, 4, 3]

.

Q.61. How is Python different from Java?



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
```

What is Python?

Python is a high-level, interpreted, interactive and object-oriented scripting language. Python is designed to be highly readable. It uses English keywords frequently where as other languages use punctuation, and it has fewer syntactical constructions than other languages.

Name some of the features of Python.

Following are some of the salient features of python −

It supports functional and structured programming methods as well as OOP.

It can be used as a scripting language or can be compiled to byte-code for building large applications.

It provides very high-level dynamic data types and supports dynamic type checking.

It supports automatic garbage collection.

It can be easily integrated with C, C++, COM, ActiveX, CORBA, and Java.

What is the purpose of PYTHONSTARTUP environment variable?

PYTHONSTARTUP - It contains the path of an initialization file containing Python source code. It is executed every time you start the interpreter. It is named as .pythonrc.py in Unix and it contains commands that load utilities or modify PYTHONPATH.

What is the purpose of PYTHONCASEOK environment variable?

PYTHONCASEOK − It is used in Windows to instruct Python to find the first case-insensitive match in an import statement. Set this variable to any value to activate it.

What is the purpose of PYTHONHOME environment variable?

PYTHONHOME − It is an alternative module search path. It is usually embedded in the PYTHONSTARTUP or PYTHONPATH directories to make switching module libraries easy.

Is python a case sensitive language?

Yes! Python is a case sensitive programming language.

What are the supported data types in Python?

Python has five standard data types −

Numbers

String

List

Tuple

Dictionary

What is the output of print str if str = 'Hello World!'?

It will print complete string. Output would be Hello World!.

What is the output of print str[0] if str = 'Hello World!'?

It will print first character of the string. Output would be H.

What is the output of print str[2:5] if str = 'Hello World!'?

It will print characters starting from 3rd to 5th. Output would be llo.

What is the output of print str[2:] if str = 'Hello World!'?

It will print characters starting from 3rd character. Output would be llo World!.

What is the output of print str * 2 if str = 'Hello World!'?

It will print string two times. Output would be Hello World!Hello World!.

What is the output of print str + "TEST" if str = 'Hello World!'?

It will print concatenated string. Output would be Hello World!TEST.

What is the output of print list if list = ['abcd', 786 , 2.23, 'john', 70.2]?

It will print complete list. Output would be ['abcd', 786, 2.23, 'john', 70.200000000000003].

What is the output of print list[0] if list = ['abcd', 786 , 2.23, 'john', 70.2]?

It will print first element of the list. Output would be abcd.

What is the output of print list[1:3] if list = ['abcd', 786 , 2.23, 'john', 70.2]?

It will print elements starting from 2nd till 3rd. Output would be [786, 2.23].

What is the output of print list[2:] if list = ['abcd', 786 , 2.23, 'john', 70.2]?

It will print elements starting from 3rd element. Output would be [2.23, 'john', 70.200000000000003].

What is the output of print tinylist * 2 if tinylist = [123, 'john']?

It will print list two times. Output would be [123, 'john', 123, 'john'].

What is the output of print list1 + list2, if list1 = ['abcd', 786 , 2.23, 'john', 70.2] and ist2 = [123, 'john']?

It will print concatenated lists. Output would be ['abcd', 786, 2.23, 'john', 70.2, 123, 'john']

What are tuples in Python?

A tuple is another sequence data type that is similar to the list. A tuple consists of a number of values separated by commas. Unlike lists, however, tuples are enclosed within parentheses.

What is the difference between tuples and lists in Python?

The main differences between lists and tuples are − Lists are enclosed in brackets ([ ]) and their elements and size can be changed, while tuples are enclosed in parentheses (( )) and cannot be updated. Tuples can be thought of as read-only lists.

What is the output of print tuple if tuple = ('abcd', 786 , 2.23, 'john', 70.2)?

It will print complete tuple. Output would be ('abcd', 786, 2.23, 'john', 70.200000000000003).

What is the output of print tuple[0] if tuple = ('abcd', 786 , 2.23, 'john', 70.2)?

It will print first element of the tuple. Output would be abcd.

What is the output of print tuple[1:3] if tuple = ('abcd', 786 , 2.23, 'john', 70.2)?

It will print elements starting from 2nd till 3rd. Output would be (786, 2.23).

What is the output of print tuple[2:] if tuple = ('abcd', 786 , 2.23, 'john', 70.2)?

It will print elements starting from 3rd element. Output would be (2.23, 'john', 70.200000000000003).

What is the output of print tinytuple * 2 if tinytuple = (123, 'john')?

It will print tuple two times. Output would be (123, 'john', 123, 'john').

What is the output of print tuple + tinytuple if tuple = ('abcd', 786 , 2.23, 'john', 70.2) and tinytuple = (123, 'john')?

It will print concatenated tuples. Output would be ('abcd', 786, 2.23, 'john', 70.200000000000003, 123, 'john').

What are Python's dictionaries?

Python's dictionaries are kind of hash table type. They work like associative arrays or hashes found in Perl and consist of key-value pairs. A dictionary key can be almost any Python type, but are usually numbers or strings. Values, on the other hand, can be any arbitrary Python object.

How will you create a dictionary in python?

Dictionaries are enclosed by curly braces ({ }) and values can be assigned and accessed using square braces ([]).

```
dict = {}

dict['one'] = "This is one"
```

dict[2]   = "This is two"

tinydict = {'name': 'john','code':6734, 'dept': 'sales'}

How will you get all the keys from the dictionary?

Using dictionary.keys() function, we can get all the keys from the dictionary object.

print dict.keys()   # Prints all the keys

How will you get all the values from the dictionary?

Using dictionary.values() function, we can get all the values from the dictionary object.

print dict.values()   # Prints all the values

How will you convert a string to an int in python?

int(x [,base]) - Converts x to an integer. base specifies the base if x is a string.

How will you convert a string to a long in python?

long(x [,base]) - Converts x to a long integer. base specifies the base if x is a string.

How will you convert a string to a float in python?

float(x) − Converts x to a floating-point number.

How will you convert a object to a string in python?

str(x) − Converts object x to a string representation.

How will you convert a object to a regular expression in python?

repr(x) − Converts object x to an expression string.

How will you convert a String to an object in python?

eval(str) − Evaluates a string and returns an object.

How will you convert a string to a tuple in python?

tuple(s) − Converts s to a tuple.

How will you convert a string to a list in python?

list(s) − Converts s to a list.

How will you convert a string to a set in python?

set(s) − Converts s to a set.

How will you create a dictionary using tuples in python?

dict(d) − Creates a dictionary. d must be a sequence of (key,value) tuples.

How will you convert a string to a frozen set in python?

frozenset(s) − Converts s to a frozen set.

How will you convert an integer to a character in python?

chr(x) − Converts an integer to a character.

How will you convert an integer to an unicode character in python?

unichr(x) − Converts an integer to a Unicode character.

How will you convert a single character to its integer value in python?

ord(x) − Converts a single character to its integer value.

How will you convert an integer to hexadecimal string in python?

hex(x) − Converts an integer to a hexadecimal string.

How will you convert an integer to octal string in python?

oct(x) − Converts an integer to an octal string.

What is the purpose of ** operator?

** Exponent − Performs exponential (power) calculation on operators. a**b = 10 to the power 20 if a = 10 and b = 20.

What is the purpose of // operator?

// Floor Division − The division of operands where the result is the quotient in which the digits after the decimal point are removed.

What is the purpose of is operator?

is − Evaluates to true if the variables on either side of the operator point to the same object and false otherwise. x is y, here is results in 1 if id(x) equals id(y).

What is the purpose of not in operator?

not in − Evaluates to true if it does not finds a variable in the specified sequence and false otherwise. x not in y, here not in results in a 1 if x is not a member of sequence y.

What is the purpose break statement in python?

break statement − Terminates the loop statement and transfers execution to the statement immediately following the loop.

What is the purpose continue statement in python?

continue statement − Causes the loop to skip the remainder of its body and immediately retest its condition prior to reiterating.

What is the purpose pass statement in python?

pass statement − The pass statement in Python is used when a statement is required syntactically but you do not want any command or code to execute.

How can you pick a random item from a list or tuple?

choice(seq) − Returns a random item from a list, tuple, or string.

How can you pick a random item from a range?

randrange ([start,] stop [,step]) − returns a randomly selected element from range(start, stop, step).

How can you get a random number in python?

random() − returns a random float r, such that 0 is less than or equal to r and r is less than 1.

How will you set the starting value in generating random numbers?

seed([x]) − Sets the integer starting value used in generating random numbers. Call this function before calling any other random module function. Returns None.

How will you randomizes the items of a list in place?

shuffle(lst) − Randomizes the items of a list in place. Returns None.

How will you capitalizes first letter of string?

capitalize() − Capitalizes first letter of string.

How will you check in a string that all characters are alphanumeric?

isalnum() − Returns true if string has at least 1 character and all characters are alphanumeric and false otherwise.

How will you check in a string that all characters are digits?

isdigit() − Returns true if string contains only digits and false otherwise.

How will you check in a string that all characters are in lowercase?

islower() − Returns true if string has at least 1 cased character and all cased characters are in lowercase and false otherwise.

How will you check in a string that all characters are numerics?

isnumeric() − Returns true if a unicode string contains only numeric characters and false otherwise.

How will you check in a string that all characters are whitespaces?

isspace() − Returns true if string contains only whitespace characters and false otherwise.

How will you check in a string that it is properly titlecased?

istitle() − Returns true if string is properly "titlecased" and false otherwise.

How will you check in a string that all characters are in uppercase?

isupper() − Returns true if string has at least one cased character and all cased characters are in uppercase and false otherwise.

How will you merge elements in a sequence?

join(seq) − Merges (concatenates) the string representations of elements in sequence seq into a string, with separator string.

How will you get the length of the string?

len(string) − Returns the length of the string.

How will you get a space-padded string with the original string left-justified to a total of width columns?

ljust(width[, fillchar]) − Returns a space-padded string with the original string left-justified to a total of width columns.

How will you convert a string to all lowercase?

lower() − Converts all uppercase letters in string to lowercase.

How will you remove all leading whitespace in string?

lstrip() − Removes all leading whitespace in string.

How will you get the max alphabetical character from the string?

max(str) − Returns the max alphabetical character from the string str.

How will you get the min alphabetical character from the string?

min(str) − Returns the min alphabetical character from the string str.

How will you replaces all occurrences of old substring in string with new string?

replace(old, new [, max]) − Replaces all occurrences of old in string with new or at most max occurrences if max given.

How will you remove all leading and trailing whitespace in string?

strip([chars]) − Performs both lstrip() and rstrip() on string.

How will you change case for all letters in string?

swapcase() − Inverts case for all letters in string.

How will you get titlecased version of string?

title() − Returns "titlecased" version of string, that is, all words begin with uppercase and the rest are lowercase.

How will you convert a string to all uppercase?

upper() − Converts all lowercase letters in string to uppercase.

How will you check in a string that all characters are decimal?

isdecimal() − Returns true if a unicode string contains only decimal characters and false otherwise.

What is the difference between del() and remove() methods of list?

To remove a list element, you can use either the del statement if you know exactly which element(s) you are deleting or the remove() method if you do not know.

What is the output of len([1, 2, 3])?

3.

What is the output of [1, 2, 3] + [4, 5, 6]?

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

What is the output of ['Hi!'] * 4?

['Hi!', 'Hi!', 'Hi!', 'Hi!']

What is the output of 3 in [1, 2, 3]?

True

What is the output of for x in [1, 2, 3]: print x?

1

2

3

What is the output of L[2] if L = [1,2,3]?

3, Offsets start at zero.

What is the output of L[-2] if L = [1,2,3]?

1, Negative: count from the right.

What is the output of L[1:] if L = [1,2,3]?

2, 3, Slicing fetches sections.

How will you compare two lists?

cmp(list1, list2) − Compares elements of both lists.

How will you get the length of a list?

len(list) − Gives the total length of the list.

How will you get the max valued item of a list?

max(list) − Returns item from the list with max value.

How will you get the min valued item of a list?

min(list) − Returns item from the list with min value.

How will you get the index of an object in a list?

list.index(obj) − Returns the lowest index in list that obj appears.

How will you insert an object at given index in a list?

list.insert(index, obj) − Inserts object obj into list at offset index.

How will you remove last object from a list?

list.pop(obj=list[-1]) − Removes and returns last object or obj from list.

How will you remove an object from a list?

list.remove(obj) − Removes object obj from list.

How will you reverse a list?

list.reverse() − Reverses objects of list in place.

How will you sort a list?

list.sort([func]) − Sorts objects of list, use compare func if given..

What we call a function which is incomplete version of a function?

Stub.

When a function is defined then the system stores parameters and local variables in an area of memory. What this memory is known as?

Stack.

A canvas can have a foreground color? (Yes/No)

Yes.

Is Python platform independent?

No

There are some modules and functions in python that can only run on certain platforms.

Do you think Python has a complier?

Yes

Yes it has a complier which works automatically so we don’t notice the compiler of python.

What are the applications of Python?

Django (Web framework of Python).

2. Micro Frame work such as Flask and Bottle.

3. Plone and Django CMS for advanced content Management.

What is the basic difference between Python version 2 and Python version 3?

Table below explains the difference between Python version 2 and Python version 3.

S.No	Section	Python Version2	Python Version3
1.	Print Function		

Print command can be used without parentheses.

Python 3 needs parentheses to print any string. It will raise error without parentheses.

2. Unicode

ASCII str() types and separate Unicode() but there is no byte type code in Python 2.

Unicode (utf-8) and it has two byte classes −

Byte

Bytearray S.

3. Exceptions

Python 2 accepts both new and old notations of syntax.

Python 3 raises a SyntaxError in turn when we don't enclose the exception argument in parentheses.

4. Comparing Unorderable

It does not raise any error.

It raises 'TypeError' as warning if we try to compare unorderable types.

Which programming Language is an implementation of Python programming language designed to run on Java Platform?

Jython

(Jython is successor of Jpython.)

Is there any double data type in Python?

No

Is String in Python are immutable? (Yes/No)

Yes.

Can True = False be possible in Python?

No.

When does a new block begin in python?

A block begins when the line is intended by 4 spaces.

Name the python Library used for Machine learning.

Scikit-learn python Library used for Machine learning

What does pass operation do?

Pass indicates that nothing is to be done i.e. it signifies a no operation.

Name the tools which python uses to find bugs (if any).

Pylint and pychecker.