Python Interview Questions and Answers for Freshers and Advanced Level Experienced

Basic Level Python Interview Questions for Freshers and Beginners

Q1: Explain Python?

Answer: Python is a highly comprehensive, interactive, and object-oriented scriptwriting language. It is specifically developed with the purpose of making the content highly readable among the net surfers. Python makes use of various English keywords other than just punctuations. It also has lesser syntactical constructions like in other languages.

Q2: What are the distinct features of Python?

Answer: The distinct features of Python include the following.

1. Structured and functional programming's are supported.
2. It can be compiled to byte-code for creating larger applications.
3. Develops high-level dynamic data types.
4. Supports checking of dynamic data types.
5. Applies automated garbage collection.
6. It could be used effectively along with Java, COBRA, C, C++, ActiveX, and COM.

Q3: What is Pythonpath?

Answer: ***Pythonpath*** is an environment variable which you can set to add additional directories where python will look for modules and packages**.** A ***Pythonpath*** tells the Python interpreter to locate the module files that can be imported into the program. It includes the Python source library directory and source code directory.

Q4: Can we preset Pythonpath?

Answer: Yes, we can preset ***Pythonpath*** as a Python installer.

Q5: Why do we use Pythonstartup environment variable?

Answer: We use the ***Pythonstartup*** environment variable because it consists of the path in which the initialization file carrying Python source code can be executed to start the interpreter. If this is the name of a readable file, the Python commands in that file are executed before the first prompt is displayed in interactive mode. The file is executed in the same namespace where interactive commands are executed so that objects defined or imported in it can be used without qualification in the interactive session.

Q6: What is the Pythoncaseok environment variable?

Answer: ***Pythoncaseok*** environment variable is applied in Windows with the purpose to direct Python to find the first case insensitive match in an import statement. The purpose of ***Pythoncaseok*** is to enable finding module files on filesystems which are case-insensitive such as `FAT`, or which behave in a case-insensitive manner from point of view of the programmer, such as `NTFS` on Windows. It exists to support code written for case-insensitive filesystems before case-sensitivity became the default behaviour when searching for modules in python 2.1. The detailed explanation for the change is available in PEP 235 .

Q7: What are the supported standard data types in Python?

Answer: The supported standard data types in Python include the following.

1. List.
2. Number.
3. String.
4. Dictionary.
5. Tuples.

Q8: Define tuples in Python?

Answer: ***Tuples*** is a sequence data type in Python. The number of values in ***tuples*** are separated by commas. ***Tuples*** are used to store multiple items in a single variable. A ***tuple*** is a collection which is ordered and **unchangeable**. ***Tuple*** items can be of any data type:

```
tuple1 = ("apple", "banana", "cherry")
tuple2 = (1, 5, 7, 9, 3)
tuple3 = (True, False, False)
```

Q9: What is the major difference between tuples and lists in Python?

Answer: There are several major differences between ***tuples*** and ***lists*** in Python, which include the following:

Tuples	**Lists**
Tuples are similar to a list, but they are enclosed within parenthesis, unlike the list.	The ***list*** is used to create a sequence.
The element and size can be changed.	The element and size cannot be changed.
They cannot be updated.	They can be updated.
They act as read-only lists.	They act as a changeable list.
Tuples are surrounded by ( )	***Lists*** are surrounded by [ ]
Example of ***Tuple*** Code is, tup = (1, "a", "string", 1+2)	Example of ***Lists*** Code is, L = [1, "a" , "string" , 1+2]

Q10: 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.

Q11: 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.

Q12: 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.

```
Example 1: def Function:
               pass

Example 2:
n = 10
for i in range(n):

  # pass can be used as placeholder
  # when code is to added later
  Pass

Example 3:
a = 10
b = 20

if(a<b):
  pass
else:
  print("b<a")
```

Q13: 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.

Q14: 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***. Print “Hello World!”, then the runtime error would be the missing parenthesis that is required by print ( ).

Q15: 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
```

Q16: 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.

Q17: 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.

Q18: 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.

Q19: Does Python support an intrinsic do-while loop?

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

Q20: 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

Q21: 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.

Q22: What is the purpose of relational operators in Python?

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

Python Relational Operator

- Less than → used with <
- Greater than → used with >
- Equal to → used with ==
- Not equal to → used with !=
- Less than or equal to → used with <=
- Greater than or equal to → used with >=

Q23: What are assignment operators in Python?

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

Operator	Description assignment operators	Syntax
=	Assign value of right side of expression to left side operand	x = y + z
+=	Add and Assign: Add right side operand with left side operand and then assign to left operand	a += b
-=	Subtract AND: Subtract right operand from left operand and then assign to left operand: True if both operands are equal	a -= b
*=	Multiply AND: Multiply right operand with left operand and then assign to left operand	a *= b
/=	Divide AND: Divide left operand with right operand and then assign to left operand	a /= b
%=	Modulus AND: Takes modulus using left and right operands and assign result to left operand	a %= b
//=	Divide(floor) AND: Divide left operand with right operand and then assign the value(floor) to left operand	a //= b
**=	Exponent AND: Calculate exponent(raise power) value using operands and assign value to left operand	a **= b
&=	Performs Bitwise AND on operands and assign value to left operand	a &= b
|=	Performs Bitwise OR on operands and assign value to left operand	a |= b
^=	Performs Bitwise xOR on operands and assign value to left operand	a ^= b
>>=	Performs Bitwise right shift on operands and assign value to left operand	a >>= b
<<=	Performs Bitwise left shift on operands and assign value to left operand	a <<= b

Q24: 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. A ***membership operator*** in Python can be defined as being an operator that is used to validate the membership of a value. This operator is used to test memberships in variables such as strings, integers as well as tuples.

Operator	Description of membership operators	Example
in	Evaluates to true if it finds a variable in the specified sequence and false otherwise.	x in y, here in results in a 1 if x is a member of sequence y.
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.

Q25: 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.

Q26: 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.

Q27: 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.

Q28: 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.

Q29: Explain how is it possible to Get the Google cache age of any URL or webpage using Python.

Answer: In order to Get the Google cache age of any URL or webpage using Python, the following URL format is used:

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

Simply replace URLGOESHERE with the web address of the website or webpage whose cache you need to retrieve and see in Python.

Q30: Give a detailed explanation about setting up the database in Django.

Answer: The process of ***setting up a database*** is initiated by using the command edit mysite/setting.py. This is a normal Python module with a module-level representation of Django settings. Django relies on SQLite by default, which is easy to be used as it doesn't require any other installation.

SQLite stores data as a single file in the filesystem. Now, you need to tell Django how to use the database. For this, the project's setting.py file needs to be used. Following code must be added to the file for making the database workable with the Django project:

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

If you need to use a database server other than the SQLite, such as MS SQL, MySQL, and PostgreSQL, then you need to use the database's administration tools to create a brand new database for your Django project.

You have to modify the following keys in the DATABASE ‘default’ item to make the new database work with the Django project:

- ENGINE – For example, when working with a MySQL database replace ‘django.db.backends.sqlite3’ with ‘django.db.backends.mysql’
- NAME – Whether using SQLite or some other database management system, the database is typically a file on the system. The NAME should contain the full path to the file, including the name of that particular file.

NOTE: - Settings like Host, Password, and User needs to be added when not choosing SQLite as the database.

Advanced Level Python Interview Questions for Experienced

Q1: How will you differentiate between deep copy and shallow copy?

Answer: We use a ***shallow copy*** when a new instance type gets created. It keeps the values that are copied in the new instance. Just like it copies the values, the ***shallow copy*** also copies the reference pointers.

Reference points copied in the ***shallow copy*** reference to the original objects. Any changes made in any member of the class affect the original copy of the same. ***Shallow copy*** enables faster execution of the program.

Deep copy is used for storing values that are already copied. Unlike ***shallow copy***, it doesn't copy the reference pointers to the objects. ***Deep copy*** makes the reference to an object in addition to storing the new object that is pointed by some other object.

Changes made to the original copy will not affect any other copy that makes use of the referenced or stored object. Contrary to the shallow copy, ***deep copy*** makes execution of a program slower. This is due to the fact that it makes some copies for each object that is called.

Q2: How will you distinguish between NumPy and SciPy?

Answer: Typically, ***NumPy*** contains nothing but the array data type and the most basic operations, such as basic element-wise functions, indexing, reshaping, and sorting. All the numerical code resides in ***SciPy***.

As one of ***NumPy's*** most important goals is compatibility, the library tries to retain all features supported by either of its predecessors. Hence, ***NumPy*** contains a few linear algebra functions despite the fact that these more appropriately belong to the ***SciPy*** library.

SciPy contains fully-featured versions of the linear algebra modules available to ***NumPy*** in addition to several other numerical algorithms.

Q3: Observe the following code:

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

Write down the output of the code.

Answer:

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

Q4: Python has something called the dictionary. Explain using an example.

Answer: A ***dictionary*** in Python programming language is an unordered collection of data values such as a map. ***Dictionary*** holds key:value pair. It helps in defining a one-to-one relationship between keys and values. Indexed by keys, a typical ***dictionary*** contains a pair of keys and corresponding values.

Let us take an example with two keys, namely Language, and Offering. Their corresponding values are Python, and Tutorials. The code for the example will be:

```
dict={‘Language’:‘Python’:‘Offering’:‘Tutorials’}
print dict[Language] #Prints Python
print dict[Offering] #Prints Tutorials
```

Q5: Python supports negative indexes. What are they and why are they used?

Answer: The sequences in Python are indexed. It consists of ***positive*** and ***negative*** numbers. ***Positive*** numbers use 0 as the first index, 1 as the second index, and so on. Hence, any index for a positive number n is n-1.

Unlike ***positive*** numbers, index numbering for the negative numbers start from -1 and it represents the last index in the sequence. Likewise, -2 represents the penultimate index. These are known as ***negative*** indexes. ***Negative*** indexes are used for:

- Removing any new-line spaces from the string, thus allowing the string to except the last character, represented as S[:-1]
- Showing the index to representing the string in the correct order

Q6: Suppose you need to collect and print data from IMDb top 250 Movies page. Write a program in Python for doing so. (NOTE: - You can limit the displayed information for 3 fields; namely movie name, release year, and rating.)

Answer:

```
from bs4 import BeautifulSoup
import requests
import sys
url = 'http://www.imdb.com/chart/top'
response = requests.get(url)
soup = BeautifulSoup(response.text)
tr = soup.findChildren("tr")
tr = iter(tr)
next(tr)
for movie in tr:
```

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

Q7: Take a look at the following code:

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

What will be the output?

Answer: The output of the program will be “invalid code.” This is because a new exception class must inherit from a BaseException.

Q8: What do you understand by monkey patching in Python?

Answer: The dynamic modifications made to a class or module at runtime are termed as ***monkey patching*** in Python. Consider the following code snippet:

```
class MyClass:
def f(self):
print "f()"
```

We can ***monkey-patch*** the program something like this:

```
import m
def monkey_f(self):
print "monkey_f()"
m.MyClass.f = monkey_f
obj = m.MyClass()
obj.f()
```

The output for the program will be monkey_f().

The examples demonstrate changes made in the behavior of f() in MyClass using the function we defined i.e. monkey_f() outside of the module m.

Q9: What do you understand by the process of compilation and linking in Python?

Answer: In order to compile new extensions without any error, compiling and linking is used in Python. ***Linking*** initiates only and only when the ***compilation*** is complete.

In the case of dynamic loading, the process of ***compilation*** and ***linking*** depends on the style that is provided with the concerned system. In order to provide dynamic loading of the configuration setup files and rebuilding the interpreter, the Python interpreter is used.

Q10: What is Flask and what are the benefits of using it?

Answer: Flask is a web microframework for Python. Flask depends on the Jinja template engine and the Werkzeug WSGI toolkit. As such, it has some notable advantages:

- Flask has little to no dependencies on external libraries
- Because there is a little external dependency to update and fewer security bugs, the web microframework is lightweight to use.
- Features an inbuilt development server and a fast debugger.

Q11: What is the map() function used for in Python?

Answer: The ***map()*** function applies a given function to each item of an iterable. It then returns a list of the results. The value returned from the ***map()*** function can then be passed on to functions to the likes of the list() and set().

Typically, the given function is the first argument and the iterable is available as the second argument to a ***map()*** function. Several tables are given if the function takes in more than one arguments.

Q12: What is Pickling and Unpickling in Python?

Answer: The Pickle module in Python allows accepting any object and then converting it into a string representation. It then dumps the same into a file by means of the dump function. This process is known as pickling.

The reverse process of ***pickling*** is known as ***unpickling*** i.e. retrieving original Python objects from a stored string representation.

Q13: Whenever Python exits, all the memory isn't deallocated. Why is it so?

Answer: Upon exiting, Python's built-in effective clean up mechanism comes into play and try to deallocate or destroy every other object.

However, Python modules that are having circular references to other objects or the objects that are referenced from the global namespaces aren't always ***deallocated*** or destroyed.

This is because it is not possible to ***deallocate*** those portions of the ***memory*** that are reserved by the C library.

Q14: Write a program in Python for getting indices of N maximum values in a NumPy array.

Answer:

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

Q15: Write code to show randomizing the items of a list in place in Python along with the output.

Answer:

```
from random import shuffle
x = ['Learning', 'Python']
shuffle(x)
print(x)
Output:
['Python', 'Learning']
```

Q16: Explain memory managed in Python?

Answer: Python private heap space takes place of ***memory management*** in Python. It contains all Python objects and data structures. The interpreter is responsible to take care of this private heap and the programmer does not have access to it. The Python ***memory manager*** is responsible for the allocation of Python heap space for Python objects. The programmer may access some tools for the code with the help of the core API. Python also provides an inbuilt garbage collector, which recycles all the unused memory and frees the memory and makes it available to heap space.

Q17: What is the lambda function?

Answer: ***Lambda*** comes from the ***Lambda Calculus*** and refers to anonymous functions in programming. ***Lambda function***, also referred to as 'Anonymous function' is same as a regular python function but can be defined without a name. While normal functions are defined using the def keyword, anonymous functions are defined using the ***lambda*** keyword.

Why is this cool? It allows you to write quick throw away functions without naming them. It also provides a nice way to write closures. With that power you can do things like this.

```
def adder(x):
    return lambda y: x + y
add5 = adder(5)
add5(1)
6
```

As you can see from the snippet of Python, the function adder takes in an argument x, and returns an anonymous function, or lambda, that takes another argument y. That anonymous function allows you to create functions from functions. This is a simple example, but it should convey the power lambdas and closures have.

Q18: What are Python decorators?

Answer: A ***decorator*** is a design pattern in Python that allows a user to add new functionality to an existing object without modifying its structure. ***Decorators*** are usually called before the definition of a function you want to decorate.

```
Example 1: Define decorator function

def uppercase_decorator(function):
    def wrapper():
        func = function()
```

```
        make_uppercase = func.upper()
        return make_uppercase

    return wrapper

Example 2: Use decorator

@uppercase_decorator
def say_hi():
    return 'hello there'

say_hi()

Output: 'HELLO THERE'
```

Q19: Differentiate between list and tuple.

Answer: ***Tuple*** is not mutable it can be hashed eg. key for dictionaries. On the other hand, ***lists*** are mutable.

Q20: How are arguments passed in Python? By value or by reference?

Answer: All of the Python is an object and all variables hold references to the object. The reference values are according to the functions; as a result, the value of the reference cannot be changed.

Q21: What are the built-in types provided by the Python?

Answer:

Mutable built-in types:

- Lists
- Sets
- Dictionaries

Immutable built-in types:

- Strings
- Tuples
- Numbers

Q22: How a file is deleted in Python?

Answer: The file can be deleted by either of these commands:

```
os.remove(filename)
os.unlink(filename)
```

Q23: What are Python modules?

Answer: A file containing Python code like functions and variables is a Python ***module***. A Python ***module*** is an executable file with a .py extension.

Python has built-in modules some of which are:

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

Q24: What is the // operator? What is its use?

Answer: The // is a Floor ***division operator*** 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.

Q25: 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.

```
Example:

#!/usr/bin/python3

str = "this is string example....wow!!!"
print (str.split( ))
print (str.split('i',1))
print (str.split('w'))

Output:

['this', 'is', 'string', 'example....wow!!!']
['th', 's is string example....wow!!!']
['this is string example....', 'o', '!!!']
```

Q26: 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.

Q27: 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.

Q28: Is Python a case sensitive language?

Answer: Yes. Python is a ***case sensitive language***.

Q29: Define slicing in Python.

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

Q30: What are docstring?

Answer: ***Docstring*** is a Python documentation string, it is a way of documenting Python functions, classes and modules. Python ***docstrings*** are the string literals that appear right after the definition of a function, method, class, or module.

```
def square(n):
    '''Takes in a number n, returns the square of n'''
    return n**2
```

Q31: 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]
```

Q32: Define Python Iterators.

Answer: Group of elements, containers or objects that can be traversed. Iterators are everywhere in Python.

Q33: 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)

Q34: 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

Q35: 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.

Q36: How are files deleted in Python?

Answer: To delete a file in Python:

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

Q37: 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 *
```

Q38: What is monkey patching?

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

Q39: Does Python supports multiple inheritances?

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

Q40: 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.

Q41: 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.

Q42: 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.

Q43: 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.

Q44: 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.

Q45: 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

Used resources: https://bit.ly/3bOPEhE

Python preparation books: https://bit.ly/2YtfaGq