# Python Interview Questions

A rundown of habitually asked Python interview inquiries with responds to for freshers and experienced are given underneath.

## 1) What is Python?

Python is a high-level, interpreted programming language known for its simplicity and readability. Python was created by **Guido Van Rossum** and first released in **1991**.

It is an object-oriented high-level language that works equally well on Windows, Linux, UNIX, and Macintosh. its built-in high-level data structures, as well as dynamic binding and typing. Python is focused on code readability and expressiveness hence it attracts many developers to use it for multiple applications including web development, data analysis, artificial intelligence, scientific computing, etc.

It requires less code to develop applications and is simple to learn.

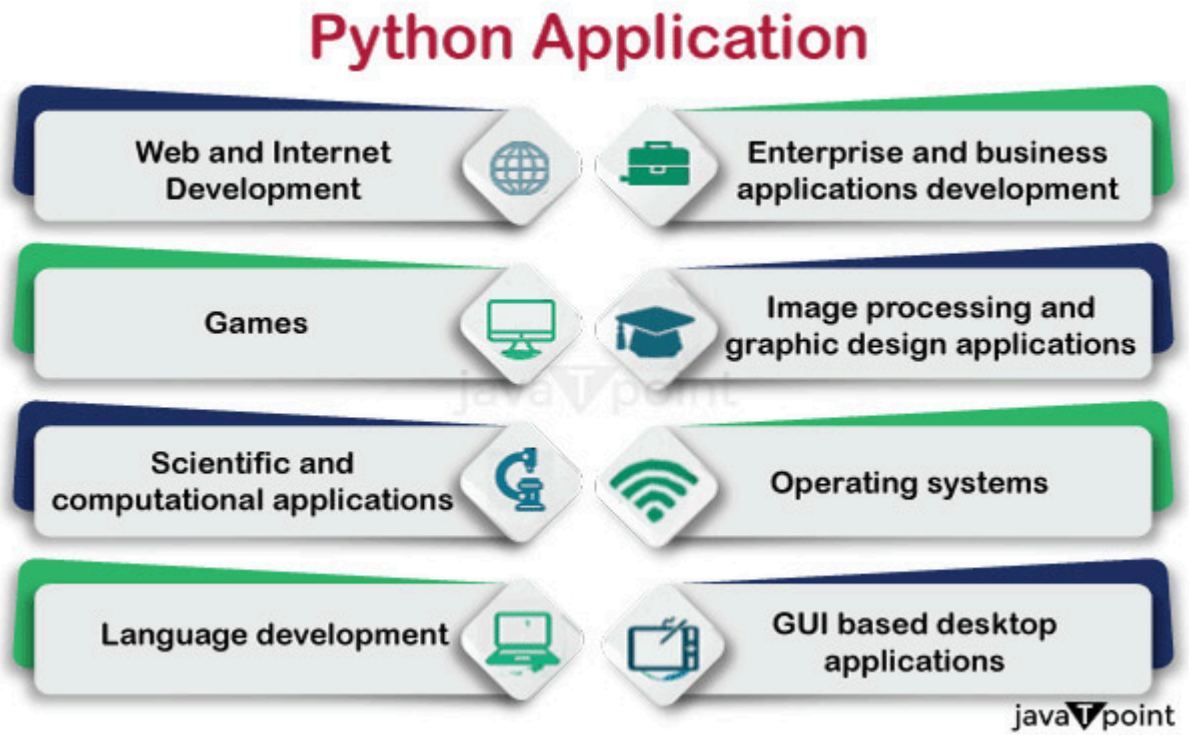

It is used extensively for:

- development of the web (server-side).
- Software creation.
- Mathematics.
- Scripting the system.
- Data Analysis and Visualization.
- Machine Learning and Artificial Intelligence.
- Game Development.
- Web Scraping.
- Desktop GUI Applications.

## 2) Why Python?

- Python is a high-level, interpreted programming language that is object-oriented and has dynamic semantics.
- Python works with Windows, Mac, Linux, Raspberry Pi, and other platforms.
- Compared to other programming languages, Python lets developers write programs with fewer lines.
- Big standard library provides less dependencies to external sources, thus development is simplified.
- Python has an active and supportive community which features an abundance of resources, advice and a huge ecosystem of third-party packages.
- The cross-platform compatibility of this software guarantees smooth functioning on different operating systems thus making it more accessible and usable.
- Python's syntax which is user-friendly, makes easy learning, making it an awesome choice for both beginners and advanced developer.
- Because Python runs on an interpreter system, the code can be run right after it is written. Providing a prototype quickly is helpful.
- Python can be depicted as a procedural way, an item orientated way or a utilitarian way.
- For all of the major platforms, the Python interpreter and the extensive standard library are free to distribute in binary or source form.

## 3) What kinds of applications can Python be used for?

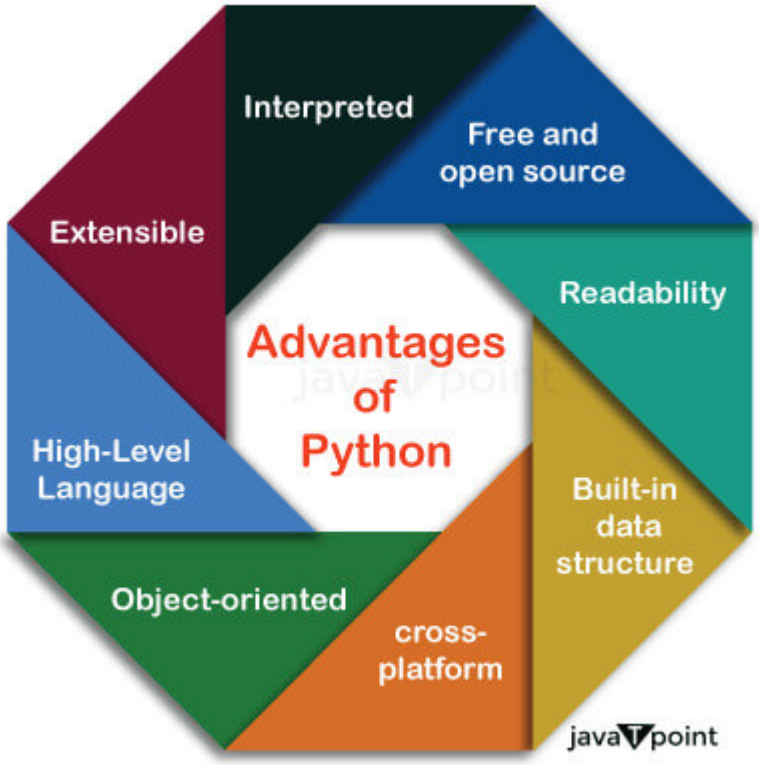



Python is utilized in several software domains, some of which are listed below.

- Games for the web and the internet Development of scientific and computational applications Language development
- Applications for image processing and graphic design Development of enterprise and business applications
- Development of operating systems graphical user interface (GUI) based desktop applications.

- Development of web applications Django, Pyramid, and Flask are the most well-known Python web frameworks.
- Processing e-mail, FTP, IMAP, and other Internet protocols are supported by Python's standard library.
- SciPy and NumPy, two Python modules, aid in the creation of scientific and computational applications.
- Python's Tkinter library supports to make a work area 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.

- It is an **Object-oriented** programming language

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

**Readability:** Python focuses on readability and thus makes code maintenance less expensive due to its simple and clear syntax. There is then no need for developers to struggle with code, for the code is made easier to write and understand, and as such programs are developed faster and errors are eliminated in incorporating a wider spectrum of exercises and fun to its practice.

- High-Level Language
- Cross-platform

**Cross-Platform:** Python code can execute on different platforms and operating systems without change such as Windows, macOS, Linux, and Unix. This cross-platform compatibility favors the choice of Python in app development that needs to run across different operating systems.

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

## 5) What is PEP 8?

The Python Enhancement Proposal, also known as PEP 8, is a document that provides instructions on how to write Python code. In essence, it is a set of guidelines for formatting Python code for maximum readability. Guido van Rossum, Barry Warsaw, and Nick Coghlan wrote it in 2001.

PEP 8 covers various aspects of Python code formatting, including:

1. Indentation
2. Whitespace
3. Name Conventions
4. Line Lengths
5. Comments

6. Imports
7. Functions and Method Definition
8. Whitespaces in Expressions and Statements

## 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**

Text can be enclosed in either single or double quotes to create string literals. String literals, for instance, are values for strings.

**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
```

**Explanation:**

- In this code:
- The variable 'a' stores the integer literal 10.
- The variable 'b' stores the float literal 12.3.
- The variable 'x' is the complex literal 3.14j.
- Each variable when printed returns its value. If the complex literal x is concerned this stands for a complex number with a real part of 0.0 and an imaginary part of 3.14.

**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
```

**Explanation:**

- In the given Python code, p is True because 1 is Boolean type equivalent to True. q is False since 1 not equal to False. r = 4 and s = 7 as True is considered

as 1 and False is considered as 0 in numeric contexts.

**Special literals**

Python contains one unique exacting, or at least, 'None'. This exceptional strict is utilized for characterizing an invalid variable. In the event that 'None' is contrasted and whatever else other than a 'None', it will get back to false.

**Example:**

```
word = None
print(word)
```

**Output:**

```
None
```

## 7) Describe the Python Functions?

The Python functions are named blocks of code which perform a single operation. They are defined by the **`def`** keyword, which is followed by a function name, parameters inside parentheses, and a colon Within the body of the function, the code that gets executed when the function is invoked is indented. Functions can take input arguments, carry out operations and optionally return the results by using the **'return'** statement. They encourage code reusability, organization, and abstraction.

Functions fall into three categories:

Built-In Functions: duplicate(), len(), count() are the a few implicit capabilities.

User-defined Functions: User-defined functions are functions that are defined by a user.

Anonymous functions: Because they are not declared using the standard def keyword, these functions are also referred to as lambda functions.

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

```
def function_name(parameters list):
    #--- statements---
    return a_value
```

## 8) What is zip() capability in Python?

The zip() function in Python returns a zip object that maps an identical index across multiple containers. It takes an iterable, transforms it into an iterator, and then uses the passed iterables to combine the elements. It returns a tuple iterator.

Signature zip(iterator1, iterator2, iterator3, etc.) Parameters iterator1, iterator2, and iterator3: These are joined-together iterator objects.

Return It returns a iterator that is the product of two or more iterators.

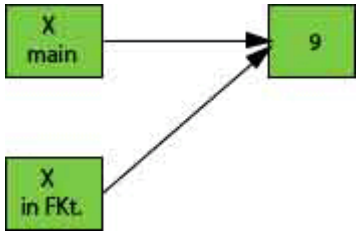
> Note: When the first list ends, zip stops making tuples if the given lists have different lengths. This indicates that two lists have three lengths, and a triple of five lengths will result.

## 9) What is Python's paramter passing system?

In Python, there are two mechanisms for passing parameters:

- Pass by references
- Pass by value

By default, all arguments (parameters) are passed to the functions "by reference." In this manner, in the event that you change the worth of the boundary inside a capability, the change is reflected in the calling capability too. It shows the first factor. For instance, if a variable is passed to a function with the declaration "a = 10" and its value is changed to "a = 20," The same value is represented by both variables.




The pass by esteem is that at whatever point we pass the contentions to the capability just qualities pass to the capability, no reference passes to the capability. It makes it unchangeable, or immutable. The original value of either variable remains the same despite being modified in the function.

Python has a default contention idea which assists with calling a technique utilizing an erratic number of contentions.

## 10) In Python, how do you overload methods or constructors?

In Python method and constructor overloading are supported like they are in languages like Java or C++. However, you can achieve similar functionality using various techniques like:

- By Using Default Arguments.
- By Using Variable Arguments.
- By Using Function Arguments.

**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:  [email protected]
```

**Explanation:**

- The code defines a class called **'Student'**.
- It tries to define two **__init__** methods within the class, but only the last one is taken because of Python's method overriding behavior.
- The last **'__init__'** method takes two parameters: **'name'** and **'email'**, assigning the instance attributes accordingly.
- When a Student object is created with both name and email arguments it initializes name and email variables.
- If an object is created with only the name argument, it throws an error because the class doesn't have a default constructor.
- To solve this you can use **default argument values or variable arguments** for constructor overloading-like behavior.

## 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)
list_1.remove(3)
print("After removal: ", list_1)
```

**Output:**

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

**Explanation:**

In this example, the first occurrence of 3 within the list list_1 is taken off at the index 0. The given list is modified after removal, which is subsequently printed out. The new list does not have value 3 at the beginning.

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

**Example:**

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

**Output:**

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

**Explanation:**

Here, the element at the '2nd index' (value 7) is used index[0] is deleted from list_1. After deleting the element with the index 2 value '7', the list is printed and it no longer contains the value '7'.

> 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 Python?

The **"swapcase()"** function in Python is a string method that creates a new string, having characters that are uppercase converted to lowercase and those that are lowercase converted to uppercase. In no other words, it switches the case of every character in the string. The original string would be unaltered.

Here's an example demonstrating the usage of 'swapcase()':

```
# Original string
original_string = "Hello World"
# Applying swapcase() function
swapped_string = original_string.swapcase()
# Output
print("Original string:", original_string)  # Output: Original string: Hello World
print("Swapped string:", swapped_string)    # Output: Swapped string: hELLO wORLD
```

**Output:**

```
Original String: Hello World
Swapped String: hELLO wORLD
```

**Explanation:**

Here the example shows that "swapcase()" function changes the uppercase letters to lowercase and lowercase letters to uppercase in the given string "Hello World" which becomes swapped string of "hELLO wORLD".

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

To eliminate the whitespaces and following spaces from the string, Python provides strip([str]) worked in capability. After removing any whitespaces that may be present, this function returns a copy of the string. If not, returns the original string.

**Example:**

```
string = "  javatpoint "
string2 = "    javatpoint        "
string3 = "       javatpoint"
print(string)
print(string2)
print(string3)
print("After stripping all have placed in a sequence:")
print(string.strip())
print(string2.strip())
print(string3.strip())
```

**Output:**

```
javatpoint
    javatpoint
       javatpoint
After stripping all have placed in a sequence:
Javatpoint
javatpoint
javatpoint
```

**Explanation:**

- In this example:
- Strings **'string', 'string2' and `'string3'**contain spaces before and/or after.
- The **'strip()'** method will chop these whitespace characters at the beginning and end of the strings off.
- After calling to 'strip()', the altered strings are printed, and they are all laid out in a sequence, and neither start nor end of string are surrounded by whitespace.

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

We can make use of the lstrip() function to get rid of leading characters from a string. It is a Python string function with an optional parameter of the char type. In the event that a boundary is given, it eliminates the person. Otherwise, it strips the string of all leading spaces.

**Example:**

```
string = "  javatpoint "
string2 = "    javatpoint        "
print(string)
print(string2)
print("After stripping all leading whitespaces:")
print(string.lstrip())
print(string2.lstrip())
```

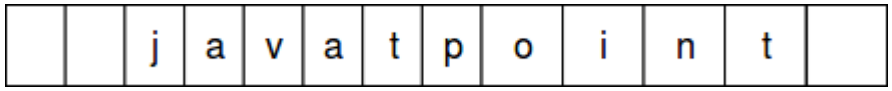
**Output:**

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

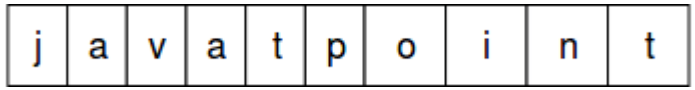
**Explanation:**

- In this example:
- Strings **'string', 'string2' and `'string3'**contain spaces before and/or after.
- The **'strip()'** method will chop these whitespace characters at the beginning and end of the strings off.

- After calling to 'strip()', the altered strings are printed, and they are all laid out in a sequence, and neither start nor end of string are surrounded by whitespace.




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
```

**Explanation:**

- In this example:
- The concatenation of each character of "Rohan" with each element of "ab".
- The string obtained is "aRohanb", the characters of "Rohan" lying in between the characters of "ab".

## 16) Give an example of shuffle() method?

The given string or array is shuffled using this technique. The items in the array become random as a result. The random module includes this method. Therefore, we must import it before calling the function. It rearranges components each time when the capability calls and creates different result.

**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:**

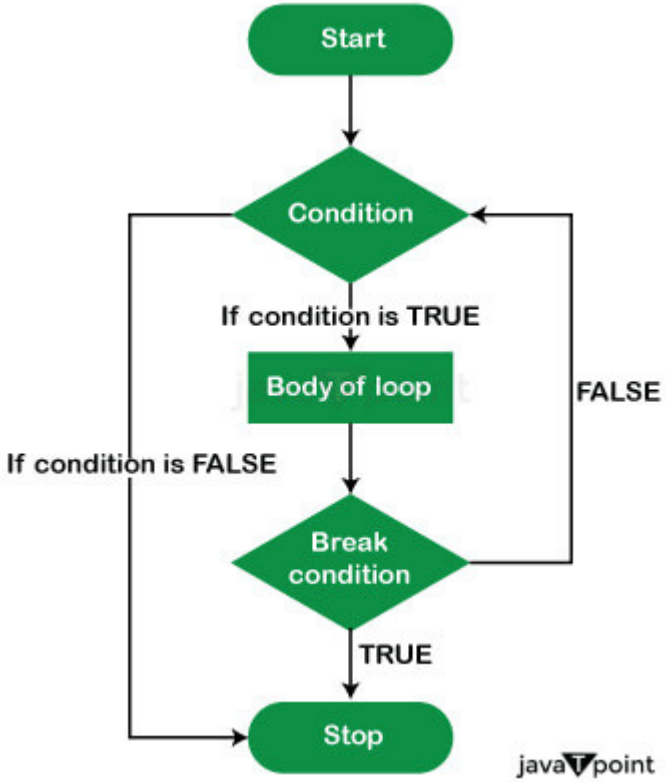
```
Original LIST1:
['Z', 'Y', 'X', 'W', 'V', 'U']

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

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

**Explanation:**

- In this example:
- The associated of the 'print' procedure performed on original 'sample_list1 is 'sample_list1.
- The function 'shuffle()' is used to shuffle the elements in the 'sample_list1'.
- The first shuffle results are printed.
- The 'shuffle()' function is used again to give a random order to the elements of the list.
- The list is printed one more time after the second shuffle occurs, each time showing a different permutation of elements.

## 17) What is the use of the break statement?

In Python, the break statement is used to exit or terminate a loop before a loop iteration is over when a specified condition is met. When used inside of a loop (eg "for" or "while"), the "break" statement causes a loop to terminate immediately, no matter what the condition of the loop or the sequence of iterations is.




The primary use cases for the 'break' statement include:

- **Early Termination:** Break statement is an essential element of the programming flow which helps break the loop when a specific condition is met

during the loop execution, unnecessary further iterations are avoided.

- **Exiting Infinite Loops:** The break statement is used to exit a loop when the loop is meant to run forever until a certain condition is successful.

**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:
      continue
   break
```

**Output:**

```
2
X 11
X 22
X 33
```

```
Y 11
Y 22
Y 33
BREAK
```

**Explanation:**

- The outer loop applies to the elements of 'list_1' with 'X' as the starting value.
- The internal loop run through elements of 'list_2' one after another for each value of 'i'.
- Between the loops, the 'i' j combination is printed first.
- When i is 'Y' and 'j' is 33, the condition if i == 'Y' and j == 33: ur satisfied.
- The code outputs 'BREAK' and continues with the outer loop by using a break statement.
- The break statement exits out the inner loop only and not the outer. As a result, the outer loop remains active. The else block under the inner definition is not reached due to the loop termination through break.
- The outer loop continues and hits the break, thus breaking the loop instantly.

## 18) What is tuple in python?

A built-in type of data collection is the tuple. It permits us to store values in a grouping. Because it cannot be changed, the original data do not reflect any changes. A tuple is created with () brackets rather than [] square brackets. We are unable to

remove any element, but we can locate it in the tuple. Indexing allows us to obtain elements. It likewise permits navigating components in switch request by utilizing negative ordering. There are a variety of Tuple methods, including Len(), max(), sum(), and sorted().

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])
```

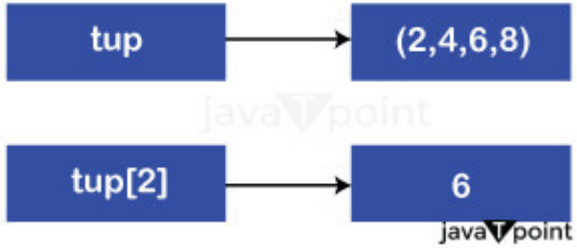
**Output:**

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

**Explanaton:**

- The pair **'tup'** of elements is 2, 4, 6, and 8.

- The tuple that consists of '(2, 4, 6, 8)' is displayed when we execute the function **'print(tup)'**.
- The **'tup[2]'** accessed a single value from the tuple using an index that gives 6 as the result, which is the element at index 2 in the tuple value.




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

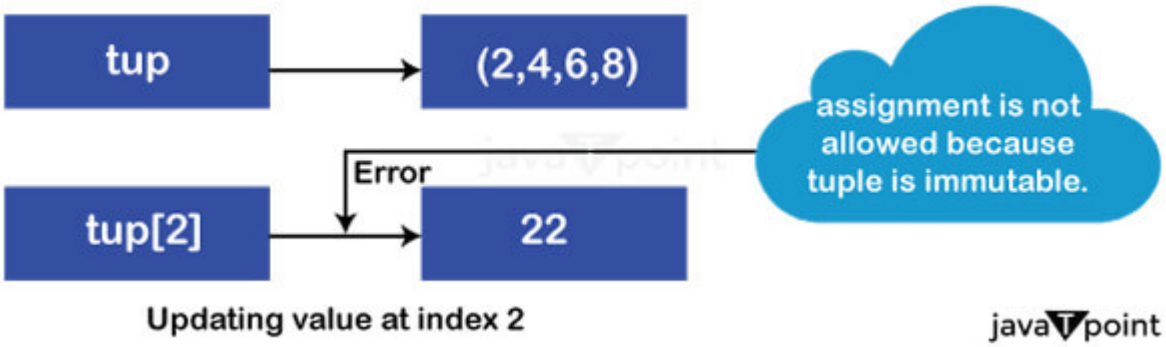
**Example:**

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

# Displaying Single value
print(tup[2])

# Updating by assigning new value
tup[2]=22
# Displaying Single value
```

```
print(tup[2])
```

**Output:**

```
tup[2]=22
TypeError: 'tuple' object does not support item assignment
(2, 4, 6, 8)
```




**Explanation:**

- When **print(tup)** and **tup[2]** are called, they give the tuple and value of the index (6 at index 2), then the code switches index 2 to 22 by assigning the assignment **tup[2] = 22**.
- Yet, this example generates a **'TypeError'** since of not-modifiable nature of tuples, item elements cannot be changed as soon as they are created.

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

You can manipulate binary and text files on the file system with the help of Python's libraries and modules. It makes it easier to create, edit, 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.

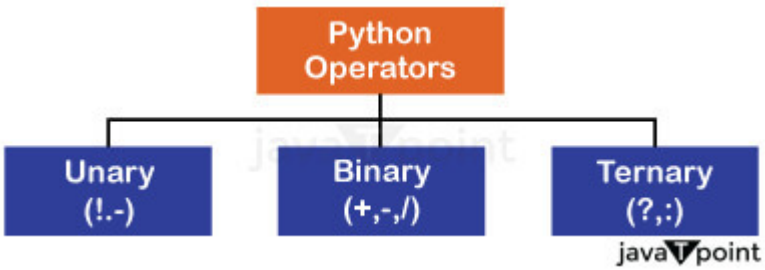
## 20) What are the different file processing modes supported by Python?

There are four ways to open files in Python. The read-write (rw), write-only (w), append (a), and read-only (r) modes. '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 both read-only and write-only modes; and 'a' is used to open a file in append mode. In the event that the mode isn't determined, of course the document opens in read-just mode.

- Read-only (r) mode: Read a file by opening it. It's the default setting.
- Only write mode (w): Open a document for composing. On the off chance that the record contains information, information would be lost. A brand-new file is also created.
- Read-Write (rw) mode: In write mode, open a file for reading. It implies refreshing mode.
- Addition mode (a): If the file exists, open it for writing and append it to the end.

## 21) What is an operator in Python?

An operator is a specific symbol that is applied to some values and results in an output. Operands are the work of an operator. Operands are literals or variables with numbers that have some values. Administrators can be unary, double, or ternary. A ternary operator, a binary operator, a ternary operator, and a unary operator are all examples of operators that require three or more operands, respectively.




**Example:**

```
# Unary Operator
A = 12
B = -(A)
print (B)
# Binary Operator
A = 12
B = 13
print (A + B)
print (B * A)
```

```
#Ternary Operator
A = 12
B = 13
min = A if A < B else B

print(min)
```

**Output:**

```
# Unary Operator
-12
# Binary Operator
25
156
# Ternary Operator
12
```

**Explanation:**

- The Unary Operator- Assigns 'A' a negative value to 'B', setting 'B' to -12.
- The binary operator performs sum multiplication operations on 'A' and 'B'. 'A + B 25', 'B * A 156', however.
- Ternary operator - Checks if 'A' is less than 'B'. If true, a 'min_value' value of 'A' is given; Otherwise, a 'B' value is assigned. In this case, 'min_value' is 12 because 'A' is less than 'B'.

## 22) What are the different types of operators in Python?

The Python language has operators that can be classified broadly according to their functions. These comprise:

1. Arithmetic Operators
2. Assignment Operators
3. Relational Operators/Comparison Operator
4. Logical Operators
5. Membership Operators
6. Bitwise Operators
7. Identity Operators

**1. Arithmetic Operators:** Perform mathematical operations which include addition(+), subtraction(-), multiplication(*), division(/), modulus(%).

**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
```

**Explanation:**

- The + operator adds the values 12 and 23, resulting the value of 35.
- The - operator subtracts 12 from 23, resulting the value of -11.
- The * operator multiplies the 12 with 23 and we obtain 276 as a result.
- The / operator divides the 12 by 2, resulting the value of 0.5217391304347826.

**2. Relational Operators:** are used to compare values. These operators test the conditions and then return a boolean value either True or False.

**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
```

**Explanation:**

- a== b checks if a is same to b, which evaluates to False.
- a< b checks if a is much less than b, which evaluates to True.
- a <= b tests if a is much less than or identical to b, which evaluates to True.
- a!= b exams if a is not equal to b, which evaluates to True.

**3. Assignment Operators** Assign values to variables and modify their values. These include easy project (=) as well as compound project operators like =, -=, *=, /=, and others.

**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
```

**Explanation:**

- a = 2 increments the fee of a with the aid of 2, resulting in a being 14.
- a-= 2 decrements the cost of a by 2, resulting in a being 12.
- a *= 2 multiplies the fee of a through 2, resulting in a being 24.
- a**= 2 increases the value of a to the energy of two, resulting in a being 576.

**4. 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
```

**5. Bitwise Operators:** Perform operations on binary representations of integers, manipulating individual bits. See the example below:

**Example:**

```
a = 10  # Binary: 1010
b = 12  # Binary: 1100
```

```
# Bitwise AND
print(a & b)  # Output: 8 (Binary: 1000)
# Bitwise OR
print(a | b)  # Output: 14 (Binary: 1110)
# Bitwise XOR
print(a ^ b)  # Output: 6 (Binary: 0110)
# Bitwise NOT
print(~a)     # Output: -11 (Binary: -1011)
```

**Output:**

```
8 (Binary: 1000)
14 (Binary: 1110)
6 (Binary: 0110)
```

**Explanation:**

- a & b: Performs bitwise AND operations on the binary representations of a and b.
- a | B: Performs bitwise OR operations on the binary representations of a and b.
- a^b: is a bitwise XOR operation on the binary representations of a and b.
- ~a: Performs a bitwise NOT operation on a, which shifts all bits and adds 1 to the result due to the complement representation of the two, resulting in -11.

## 23) How to create a Unicode string in Python?

In Python 3, the old Unicode type has been replaced by "str" type, and the string is treated as Unicode of course. Using the art.title.encode("utf-8") function, we can create a Unicode string.

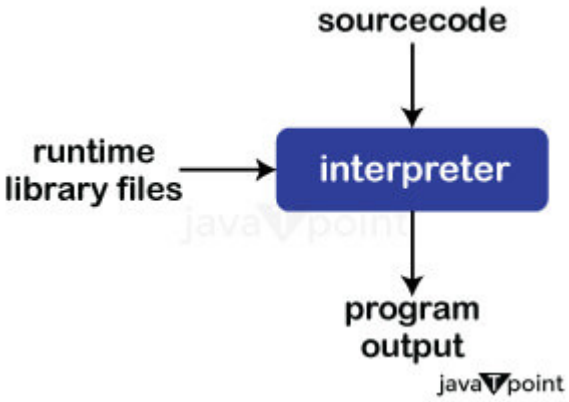
**Example:**

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

**Output:**

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

## 24) is Python interpreted language?




Yes, python is an interpreted language. This means the code executed line by line by the python interpreter rather than being compiled into a machine code beforehand.

Python does not require compilation before running, unlike Java and C.

## 25) How is memory managed in Python?

Memory management in Python is handled by the Python runtime environment and involves several key components and techniques:

- **Automatic Memory Management:** The Python is able to keeps the account for memory allocation and re-use through the automatic memory management or garbage collection. Therefore, memory allocation and release

happen automatically by the Python interpreter at runtime, which is actually a feature.

- **Reference Counting:** By using reference count python keeps trace of number of references to the each objects. In each Python object a reference count is there which is the count of current references to the object and is increased by new references and decreased when a reference goes out of scope or by explicit deletion. When the referencing count falls to zero an object loses its memory allocated in this way because the count is zero.
- **Garbage Collection:** Besides regarding of the reference counting concept, Python uses garbage collection for recovering of the object memory area of not-fully-referenced objects such as circular dependencies or cyclic references. Garbage collection scans the heap frequently to identify objects discarded and before the area is cleared to serve as a buffer for current objects.
- **Memory Pool:** Python enforces a memory pool to perform optimally in terms of allocating and releasing memory for little objects. The memory pool allows memory allocations and deallocations to not frequently happen by reusing this pool's memory block.
- **Dynamic Allocation:** Objects belong to the data types such as integers, strings, and more, which are called as data types. Python is a dynamic language that has the ability to allocate memory as needed during program execution. Memory, stating that it was acquired for the process heap, is handled by the Python runtime environment.
- **Optimizations:** What Python's code optimization can offer is manyfold. Firstly, Python interns small integers and strings, which optimizes the memory usage and improves the performance, thereby disposing the duplicate objects from

being created in the memory. Secondly, Python senses the reachable and unreachable objects and reclaims memory occupied by the unreachable ones.

## 26) What is the Python decorator?

Decorators is a useful Python tool that allows programmers to add functionality to existing code. They are very powerful. Because a component of the program attempts to modify another component at compile time, this is also known as metaprogramming. It permits the client to wrap one more capability to expand the way of behaving of the wrapped capability, without forever changing 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.

**Explanation:**

- 'function_is_called()', the function is defined and presented.
- In bounds of 'function_is_called()', there is another function that is to be 'function_is_returned()' defined.
- After the nested function 'function_is_returned()' is used, the JavaTpoint is displayed.
- 'function_is_called()' function contains the 'function_is_returned()' function that is returned.
- The returned function is pass 'new_1'.
- function new_1() is called which later results in calling the 'function_is_returned()' function finally displaying the message "JavaTpoint".
- Thus, after you type 'new_1', the console prints "JavaTpoint".

## 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.

We must explicitly declare a variable as "global" whenever a new value is given to it within the function because it is implicitly local. The global keyword must be used to declare a variable in order to make it global.

Any function can access and alter the value of global variables, which are accessible anywhere in the program.

**Example:**

```
A = "JavaTpoint"
def my_function():
  print(A)
my_function()
```

**Output:**

```
JavaTpoint
```

**Explanation:**

- The variable 'A' is assigned the value "JavaTpoint".
- The function 'my_function()' is defined, which prints the value of the variable 'A'.
- The function 'my_function()' is called.

- The value of variable A is printed in the function 'my_function()', which is "JavaTpoint".
- So, when you run the code, it will print "JavaTpoint" to the console.

**Local Variables:**

A local variable is any variable declared within a function. The local space, not the global space, contains this variable.

In the event that a variable is relegated to another worth anyplace inside the capability's body, being a local is expected.

Only the local body can access local variables.

**Example:**

```
def my_function2():
    K = "JavaTpoint Local"
    print(K)
my_function2()
```

**Output:**

```
JavaTpoint Local
```

**Explanation:**

- The function 'my_function2()' is defined.
- A local variable 'K' is defined with value "JavaTpoint Local" in 'my_function2()'.
- The 'K' value of the local variable is printed.
- The function 'my_function2()' is called.
- During the function call, the value of the local variable K is printed, which is "JavaTpoint Local".

## 28) What is the namespace in Python?

The namespace is a fundamental concept for organizing and structuring code that is more useful in large projects. However, if you're new to programming, it might be a little hard to understand. Subsequently, we attempted to make namespaces somewhat more obvious.

A namespace is characterized as a basic framework to control the names in a program. It ensures that there will be no conflict and that each name will be unique.

Additionally, Python executes namespaces as word references and keeps up with name-to-protest planning where names go about as keys and the articles as values.

## 29) What are iterators in Python?

In python, Iterators are objects that represent a flow of data and which can be used to begin a process repeatedly. They extract a way for an iteration over elements which may be contained in pairs of things, tuple, dictionary, or string in one item at a time.

Iterators implement the iterator protocol, which consists of two main methods:

**__iter__():** This, too, will be the iterator object and is referred to as the init method for instance when an iterator is initialized.

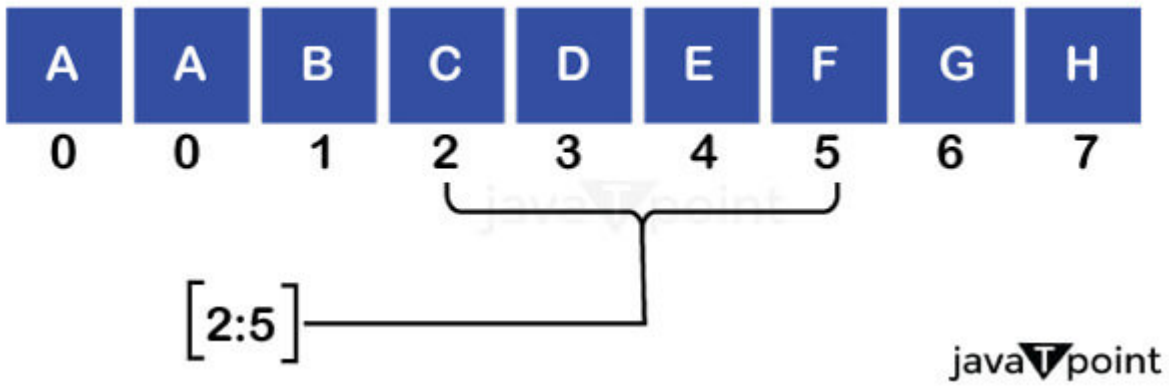
**__next__():** This method therefore gives the other element in the collection. It terminates the code further with a **StopIteration exception**(when there are no more items to return or the flow is disrupted).

## 30) What is a generator in Python?

In Python, the generator is a way that determines how to execute iterators. Except for the fact that it produces expression in the function, it is a normal function. It eliminates the __itr__ and next() methods and reduces additional overheads.

On the off chance that a capability contains essentially a yield explanation, it turns into a generator. By saving its states, the yield keyword pauses the current execution and allows it to be resumed whenever necessary.

## 31) What is slicing in Python?




Slicing is a technique that can be used in Python to cut out part of a sequence (such as a list, tuple or string) using a range of indices. Slicing will provide you with the ability to compose a new sequence, consisting of the elements extracted from the original one, starting at a prescribed start index and before the suggested end index.

**Example:**

```
my_list=[1,2,3,4,5]
# Get a slice from index 1 to index 3 (exclusive)
print(my_list[1:3])
# Get a slice from index 2 to the end of the list
print(my_list[2:])
# Get a slice from the beginning to index 3 (exclusive)
print(my_list[:3])
# Get a slice of every second element
```

```
print(my_list[::2])
```

**Output:**

```
2 3
3 4 5
1 2
1 3 5
```

**Explanation:**

- we begin with a list named **'my_list'** which holds the elements [1, 2, 3, 4, 5].
- The first 'print()' statement using slicing extracts elements from index 1 to index 3, prints and returns a new list.
- The second 'print()' statement slices a part of the list from the element of index 2 to the end of the list using slicing and then prints the result.
- The fourth 'print()' statement that slices the beginning elements of the list up to index 3 (exclusive) and print the result is 'List[:3]'.
- The last 'print()' function excludes the second value of the list and uses slicing with a step size of 2 in order to print the result.

## 32) What is a dictionary in Python?

A built-in data type is the Python dictionary. A one-to-one relationship between keys and values is established by this. A pair of keys and their values are contained in dictionaries. It stores components in key and worth matches. Values can be duplicated, whereas keys are unique. The dictionary elements are accessed by the key.

Keys index dictionaries.

**Example:**

The following example contains some keys Country Hero & Cartoon. Their corresponding values are India, Modi, and Rahul respectively.

```
dict = {'Country': 'India', 'Hero': 'Modi', 'Cartoon': 'Rahul'}
print ("Country: ", dict['Country'])
print ("Hero: ", dict['Hero'])
print ("Cartoon: ", dict['Cartoon'])
```

**Output:**

```
Country:  India
Hero:  Modi
Cartoon:  Rahul
```

**Explanation:**

- A dictionary named **'dict'** is shown as a dictionary with **key-value pairs**.
- The 'print()' function is utilized to display the values for the keys in various dictionaries.
- **dict['Country']** gives the value of the key 'Country' and prints it on console.
- **dict['Hero']** returns the value connected with the key 'Hero' and the output is printed
- **dict['Cartoon']** retrieves the value associated with the key 'Cartoon' and displays it.
- The outlet will display the values matching the keys 'Country', 'Hero', and 'Cartoon' in the dictionary.

## 33) What is Pass in Python?

Pass specifies an operation-free Python statement. It is in a compound statement as a placeholder. To make a vacant class or works, the pass catchphrase assists with passing the control without error.

**Example:**

```
class Student:
    pass # Passing class
class Student:
    def info():
```

```
    pass # Passing function
```

**Explanation:**

Two class definitions are given under the name 'Student'. The first class is null and acts as a placeholder. The second class contains an empty method named 'info()', which is defined by the keyword 'def' followed by 'pass'. Both can be extended with materials and techniques if needed.

## 34) Explain docstring in Python?

The first statement in a module, function, class, or method definition is the Python docstring, a string literal. It makes it easier to link the documents together.

"Attribute docstrings" are string literals that occur immediately after a straightforward assignment at the top.

"Additional docstrings" are string literals that occur immediately after another docstring.

Despite fitting on a single line, docstrings are created by Python using triple quotes.

The phrase in docstring ends with a period (.) and may include several lines. It may include special characters like spaces.

**Example:**

```
# One-line docstrings
def hello():
    """A function to greet."""
    return "hello"
```

**Explanation:**

In the provided Python's function code, a single-line docstring is defined for the function **'hello()'**. Inside the docstring, a single quoted text "A function to greet." is located directly above the function definition. It presents a short description of the functionality. The docstring accessor can be obtained with **'__doc__'** attribute.

## 35) What is a negative index in Python and why are they used?

Python's sequences are indexed and contain both positive and negative numbers. The numbers that are positive purposes '0' that is utilizes as first record and '1' as the subsequent file and the cycle go on that way.

The negative number's index begins with '-1,' which denotes the sequence's final index, and ends with '-2,' which denotes the sequence's penultimate index.

The negative index is used to get rid of any new-line spaces in a string and make it possible for the string to contain all but the last character, S[:-1]. The negative index is also used to show the correct order in which the index represents the string.

## 36) What is pickling and unpickling in Python?

A module known as the Python pickle is one that transforms any Python object into a string representation. Utilizing the dump function, it copies the Python object to a file; this cycle is called Pickling.

Unpickling is the procedure of obtaining the original Python objects from the stored string representation.

## 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?

The 'help()' and 'dir()' functions in Python are used for introspection and documentation purposes:

- **help():** 'help()' is the function that usually gives you access to built-in Python documentation (docstrings) for modules, classes, functions, and methods. Called without arguments it starts an interactive help session Considering an object as the argument, it displays its documentation.
- **dir():** The 'dir()' call returns a list of all valid attributes that can be applied to the object given. It is the way to explore the attributes and methods an object has, including the built-in attributes - '__doc__' and '__name__' - and any user-defned attributes.

Both of them functions have benefits for the usage on the exploration and comprehension of Python objects, the modules, and the libraries. In fact, they are great tools for learning and debugging

## 39) What are the differences between Python 2.x and Python 3.x?

Python 2.x is a previous version. Python 3.x is the most recent release. Python 2.x is inheritance now. This language is both now and in the future Python 3.x.

The most noticeable contrast somewhere in the range of Python2 and Python3 is on paper explanation (capability). It looks like print "Hello" in Python 2, and it looks like print ("Hello") in Python 3.

String in Python2 is ASCII verifiably, and in Python3 it is Unicode.

The xrange() technique has eliminated from Python 3 variant. Error handling introduces a brand-new keyword.

## 40) How Python does Compile-time and Run-time code checking?

The majority of the checking for things like type, name, and so on is done at compile time in Python. are deferred until code execution. As a result, the Python code will compile successfully if it refers to a user-defined function that does not exist. With one exception, the Python code will fail when the execution path is missing.

## 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:**

```
with open("FILE NAME", "r") as fp:
    fileData = fp.read()
# To print the contents of the file
print(fileData)
```

**Output:**

```
"The data of the file will be printed."
```

**Explanation:**

- The code of this command will make the target file named "FILE NAME" as read mode inside a context manager.
- It reads the entire file content including the header information and stores it in variable f**ileData**.
- The last task is to print the content of the file **(fileData)** stored in the console.

> Note: You should replace the "FILE NAME" with actual name (and path name if necessary) of the file you want to read.

## 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:**

```
list_1 = ["A","B","C"]
s_1 = "Javatpoint"
```

```
# creating enumerate objects
object_1 = enumerate(list_1)
object_2 = enumerate(s_1)

print ("Return type:",type(object_1))
print (list(enumerate(list_1)))
print (list(enumerate(s_1)))
```

**Output:**

```
Return type: <class 'enumerate'>
[(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')]
```

**Explanation:**

- Two sequences are defined: 'list_1' list and 's_1' string.
- **enumerate()** function makes object_1 and object_2 for the list and the string, respectively.
- The **'type()'** function helps to figure out the type of the enumerate objects, which are printed into the console.
- The **'list()'** function is employed to convert the enumerate objects into the lists while these lists are printed in the console.

- The output indicates the creation of the elements, displayed in the list and in the string by pairs, where each element is indexed.

## 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?

Python has the smtplib and email modules for sending emails. Import these modules into the made mail script and send letters by confirming a client.

It has a strategy SMTP(smtp-server, port). It requires two boundaries to lay out 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)
```

**Explanation:**

- The **smtplib** module is imported to help in sending emails.
- Initialize a **'s'** SMTP object by defining the GMail SMTP server **('smtp.gmail.com') and port ('587')**.
- Encryption the network security using **'starttls()'** for beginning TLS.
- User authentication via **'login()'** is required in the sender's email account using the email address and password of the sender.
- Define the message content which is to be sent.
- Send the email using **'sendmail()'**, mentioning the sender's email address, recipient's email address and a message.
- Close the connection with the help of **'quit()'**.

## 46) What is the difference between Python Arrays and lists?

In Python, arrays and lists are both used to store collections of elements, but they have some differences:

**1. Data Types:**

- Lists can have the objects of very different data types (e.g., integer, string, float).

- A typical array contains same type of elements and stored in contagious location. In Python, arrays are from the array module, and the data type of the elements should be specified when array is created.

**2. Functionality:**

- Lists in Python are implemented as dynamic arrays, implying that they scales up and down when elements are added or removed. They, also, possess a large variety of built-in methods for manipulation like **'append()', 'remove()', 'pop()'**, and so on.
- Arrays not only have the ability to perform fewer operations, but this is true than for lists. They provide fundamental procedures such as array accessing, insertion, deletion, and iteration. Other functions such as sorting and searching that need the usage of **'numpy'** functions also are there.

**3. Efficiency:**

- Arrays typically use lesser memory and are faster, although most notable when handling large datasets of homogeneous data types. This is because arrays have a fixed size, which is allocated for elements of a particular data type, whereas lists dynamically create space when needed.
- Lists offer more flexibility and increased convenience thanks to their various features and usability, but they may be less efficient for the particular operations because of dynamic resizing and support of heterogeneous data types.

## 47) What is lambda function in Python?

In Python, a lambda function is a small function that is defined anonymously by the keyword 'lambda'. It can have any number of arguments and it can only have one expression. Lambda functions are frequently used instead of the full 'def' statement when a function definition is very simple and not complicated enough.

**Syntax:**

```
lambda arguments: expression
```

## 48) Why do lambda forms in Python not have the statements?

Since the statement is used to create the new function object and return it at runtime, lambda forms in Python do not include it.

## 49) What are functions in Python?

In Python, functions are the self-contained, independent, and modular blocks of code that do one job. They take two inputs (arguments), perform those operations, and then return an output. Functions contribute to the code organization in separate units that are reusable, which increase the readability, debugging, and maintaining.

**Example:**

```
def New_func():
    print ("Hi, Welcome to JavaTpoint")
New_func() #calling the function
```

**Output:**

```
Hi, Welcome to JavaTpoint
```

**Explanation:**

- The 'def' keyword defines the function as 'New_func'.
- Inside the function the single statement is specified to display the message 'Hi, Welcome to JavaTpoint '.
- After the function name is defined, then the function is called by a function name followed by parenthesis, that is, '(New_func())'.
- The function that executes the statement inside the body when it is called is executed.

## 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:**

```
class Employee_1:
    def __init__(self, name, age,salary):
        self.name = name
        self.age = age
        self.salary = 20000
E_1 = Employee_1("pqr", 20, 25000)
# E1 is the instance of class Employee.
#__init__ allocates memory for E1.
print(E_1.name)
print(E_1.age)
print(E_1.salary)
```

**Output:**

```
pqr
20
25000
```

**Explanation:**

- The **Employee_1** class contains an __init__ this encapsulates new instances with attributes name, age, and salary.
- An instance 'E_1' of the 'Employee_1' class is created with the provided values: "pqr", 20 and 2500, respectively.

- '__init__' method as memory allocation and attribute initialization are performed when 'E_1' is created.
- Printing the components of 'E_1' instance casts out the values for name and age provided, yet salary remains '20000' as stored within init method.

## 51) What is self in Python?

Self is a class's instance or object. This is explicitly set as the first parameter in Python. Be that as it may, this isn't true in Java where it's discretionary. It assists with separating between the techniques and properties of a class with neighborhood factors.

The newly created object is referred to as the self-variable in the init method, whereas the object whose method was called is referred to in other methods.

## 52) In Python, how can you generate random numbers?

Irregular module is the standard module that is utilized to create an irregular number. The term "method" refers to:

```
import random
random.random
```

The statement **random.random()** strategy returns the drifting point number that is in the scope of [0, 1). Float numbers are generated at random by the function. The bound methods of the hidden instances are the ones that are utilized with the random classes. The Random instances can be used to demonstrate multi-threading applications that generate distinct thread instances. The following are additional random generators utilized in this:

- **range (a, b):** it picks a number and characterizes the in the middle between [a, b). It returns the components by choosing it arbitrarily from the reach that is determined. It does not create an object of range.
- **uniform, a and b:** It selects a floating-point number from the [a,b] range and returns the following floating-point number: normalvariate(mean, sdev) It is utilized in the normal distribution, where the mu represents the mean and the sdev represents the standard deviation.

Multiple independent random number generators are created by the Random class, which is used and instantiated.

## 53) What is PYTHONPATH?

PYTHONPATH is a enviroment variable which is utilized when a module is imported. PYTHONPATH is also looked up whenever a module is imported to see if the imported modules are in different directories. It is used by the interpreter to choose which module to load.

## 54) What are modules in Python? Name a few regularly utilized worked in modules in Python?

Modules in Python are files with the Python code that create functions, classes and variables. Allowing code organization, reusability, and maintainability by dividing related functionalities into individual files is their main property. Modules can be imported by any Python script or module using the 'import' statement.

Some of the commonly used built-in modules are:

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

## 55) What is the difference between range & xrange?

In most respects, the functionality of xrange and range is identical. You can use them both to generate a list of integers in any way you like. The main distinction is that reach returns a Python list item and x reach returns a xrange object.

This indicates that unlike range, xrange does not actually produce a static list at runtime. It makes the qualities as you want them with a unique strategy called yielding. Generators are a type of object that can benefit from this approach. This

indicates that the function to use is xrange if you want to create a list for a really large range, such as one billion.

This is especially true when working with a memory-sensitive system like a mobile phone because range will use as much memory as it can to create your integer array, which could cause a Memory Error and cause your program to crash. It's a beast that needs memories.

## 56) What benefits do NumPy exhibits offer over (nested) Python records?

Lists in Python work well as general-purpose containers. Due to Python's list comprehensions, they are simple to construct and manipulate, and they support insertion, deletion, appending, and concatenation in a fairly efficient manner.

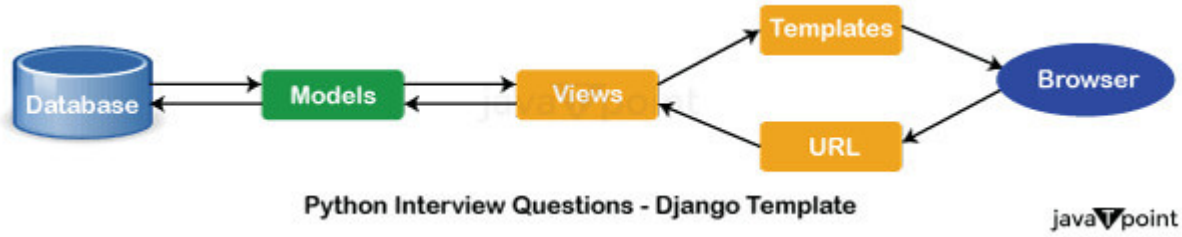
They are constrained in a few ways: Because they can contain objects of different types, Python must store type information for each element and execute type dispatching code when operating on each element because they do not support "vectorized" operations like addition and multiplication by elements.

Not only is NumPy more effective, but also additionally more helpful. Numerous free vector and matrix operations enable us to sometimes avoid unnecessary work. Additionally, they are effectively implemented.

NumPy cluster is quicker and we get a ton worked in with NumPy, FFTs, convolutions, quick looking, essential measurements, straight polynomial math, histograms, and so on.

## 57) Mention what the Django templates consist of.

A simple text file serves as the template. Any text-based format, including XML, CSV, and HTML, can be created by it. A layout contains factors that get supplanted with values when the format is assessed and labels (% tag %) that control the rationale of the layout.

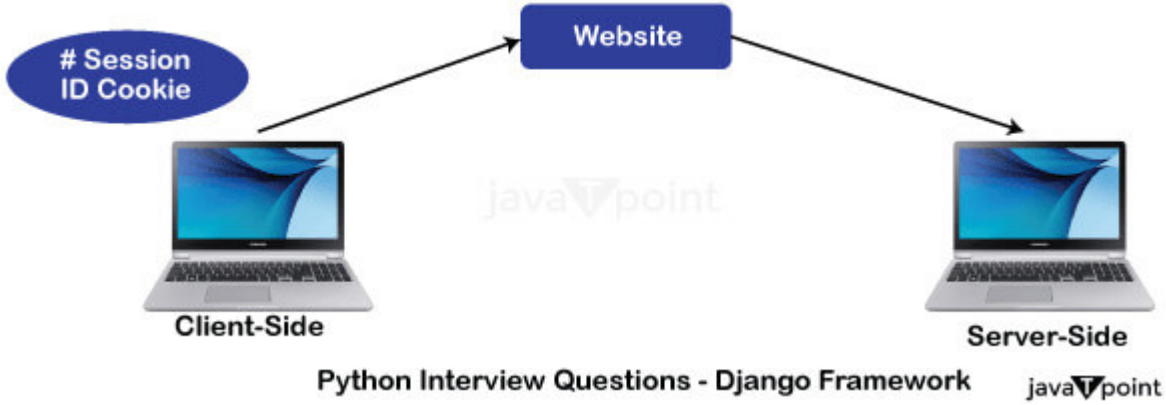



Python Interview Questions - Django Template

javatpoint

## 58) Explain the use of session in Django framework?

Django gives a meeting that allows the client to store and recover information on a for each site-guest premise. By placing a session ID cookie on the client side and storing all relevant data on the server side, Django abstracts the sending and receiving of cookies.




So, the data itself is not stored on the client side. This is good from a security perspective.

## Interview based Multiple Choice Questions on Python

1) Which of the accompanying Statements is/are Valid in regard of the Python programming language?

Statement 1: Python is a high-level, general-purpose, interpreted programming language.

Statement 2: For rapid application development and deployment, Python offers dynamic binding and typing in addition to high-level data structures.

Statement 3: Python is a programming language with statically typed code.

**Options:**

a. Only Statement 1

b. Statement 1 and 2

c. Statement 1 and 3

d. All Statements are Correct

**Show Answer** **Workspace**

2) What is the full form of PEP?

**Options:**

a. Python Enhancement Proposal

b. Python Enchantment Proposal

c. Programming Enhancement Proposition

d. Python Enrichment Program

**Show Answer** **Workspace**

3) Which of the accompanying Statements is/are NOT right in regard to Memory The executives in Python?

First Statement: Python's memory management is handled by Python Memory Manager.

2nd Statement: As the garbage collection method for Python, the CMS (Concurrent Mark Sweep) approach is utilized.

Statement 3: In order to recycle the free memory for the private heap space, Python provides a core garbage collection feature.

**Options:**

a. Only Statement 3

b. Statement 1 and 3

c. Statement 2 and 3

d. Only Statement 2

**Show Answer** **Workspace**

4) Which of the following statements about Python namespaces is true or false?

First statement: Python carries out the namespace as Exhibit.

2nd Statement: There are three types of Python namespaces: local, global, and built-in.

Statement 3: A Python namespace guarantees that each object's name is unique and can be used without being inconsistent.

**Options:**

a. Only Statement 1

b. Only Statement 3

c. Statement 1 and 2

d. Statement 1 and 3

**Show Answer** **Workspace**

5) Which of the following is invalid in terms of Variable Names?

**Options:**

a. _mystr = "Hello World!"

b. __mystr = "Hello World!"

c. __mystr__ = "Hello World!"

d. None of the mentioned

**Show Answer** **Workspace**

6) In respect to the scope in Python, which of the following statements is/are TRUE?

**Statement 1:** A variable created within a function belongs to the local scope and can be used outside that function.

**Statement 2:** A local scope is referred to the object present through the execution of code since its inception.

**Statement 3:** A local scope is referred to the local object present in the current function.

**Options:**

a. Only Statement 2

b. Statement 1 and 3

c. Only Statement 3

d. All Statements are True

**Show Answer** **Workspace**

7) What will the following snippet of code print?

**Code:**

```
# assigning a variable
myint = 10

# defining a function
def myfunction():
    # reassigning a variable
    myint = 20

# calling the function
myfunction()

# printing the value of the variable
```

```
print(myint)
```

**Options:**

a. 10

b. 20

c. 0

d. Traceback Error

**Show Answer** **Workspace**

8) What will the following snippet of code yield?

**Code:**

```
# assigning a variable
myint = 21

# using if False statement
if False:
    # reassigning a variable
    myint = 34

# defining a function
def myfunction():
```

```
    if True:
        # reassigning a variable
        myint = 65

# calling the function
myfunction()

# printing the value of the variable
print(myint)
```

**Options:**

a. 65

b. Traceback Error

c. 21

d. 34

**Show Answer** **Workspace**

9) Among the following statements based on the difference between lists and tuples, which one statement is TRUE?

**Statement 1:** List is a sequence data structure, whereas Tuple is not.

**Statement 2:** Lists are immutable; however, Tuples are mutable.

**Statement 3:** Tuple is a sequence data structure, whereas List is not.

**Statement 4:** Tuples are immutable; however, Lists are mutable.

**Options:**

a. Statement 1

b. Statement 4

c. Statement 2

d. Statement 3

**Show Answer** **Workspace**

10) What will be the output of the following snippet of code?

**Code:**

```
# defining a list
my_list = [7, 9, 8, 2, 5, 0, 1, 3, 6]

# using the pop() function
my_list.pop(2)

# printing the final list
print(my_list)
```