A blue and yellow snake logo

Description automatically generated

Python is one of the most popular programming languages in the world today. It is known for its simple and readable syntax, making it a great choice for beginners as well as experienced developers. Python is used in many areas of technology, including web development, data science, machine learning, automation, artificial intelligence, game development, and more.

Because of its wide usage, Python is a common topic in job interviews for software developers, data analysts, machine learning engineers, and other tech roles. Whether you’re a fresher preparing for your first job, or a professional looking to switch careers, understanding the most common Python interview questions can help you feel more confident and perform better.

In this guide, we have gathered the **top 100 Python interview questions and answers**, starting from the basics and moving to more advanced topics. Each answer is written in simple words and explained clearly with examples so that anyone can understand — even if you’re just starting your Python journey.

Let’s dive into the questions and get ready to ace your Python interview!

**1. What is Python?**

Python is a popular, high-level programming language that is easy to read and write. It was created by Guido van Rossum and released in 1991. One of the main reasons why Python is so widely used is because of its simple syntax that looks like English. This makes it a great choice for beginners. Python is also versatile — you can use it for web development, data science, artificial intelligence, machine learning, automation, scripting, and more. It is free to use and open-source, which means anyone can download and use it. Big companies like Google, Instagram, and Netflix use Python in their systems.

**2. What are the key features of Python?**

Python has many features that make it a popular choice among developers. First, it is easy to learn and use. The syntax is simple, which helps beginners write code quickly. Second, it is an interpreted language, meaning Python executes code line by line, which makes debugging easier. Python is cross-platform, so it works on Windows, Mac, and Linux. It also supports object-oriented and functional programming. Another great feature is its huge standard library — it includes many tools and modules to do different tasks without writing everything from scratch. Python also has a large community, so getting help and learning resources is easy.

**3. What are variables in Python?**

Variables in Python are used to store information that you want to use later. Think of a variable as a box where you can keep something, like a number or a name. You don’t need to mention the data type when creating a variable — Python figures it out. For example:

name = "Alice"  
age = 25

Here, name stores a string and age stores an integer. You can also change the value of a variable anytime. Variables make it easy to write flexible programs where data can change. You just use the variable name wherever you need that value in your code.

**4. What are data types in Python?**

Data types tell Python what kind of value a variable holds. For example, int is for whole numbers, float is for decimal numbers, and str is for text. Python also has bool for True or False values, list for a collection of items, tuple for an unchangeable group of items, dict for key-value pairs, and set for unique items. These types help Python decide what kind of operations can be done with the data. You don’t have to define the type—Python does it automatically. Knowing data types is important to avoid errors and to use functions correctly in your code.

**5. What is a list in Python?**

A list in Python is a collection of items that are ordered and changeable. Lists are written with square brackets like this:

fruits = ["apple", "banana", "cherry"]

You can add, remove, or change items in a list using functions like append(), remove(), or by using indexes like fruits[1] = "orange". Lists can contain different data types—numbers, strings, even other lists. They are useful when you want to store multiple items in one variable. You can also loop through a list using a for loop. Lists are one of the most used data structures in Python because they are simple and powerful.

**6. What is a tuple in Python?**

A tuple is similar to a list, but the key difference is that you cannot change the items in a tuple once it is created. Tuples are **immutable**, which means unchangeable. They are written with parentheses like this:

colors = ("red", "green", "blue")

You can access items in a tuple using an index like colors[0]. Tuples are useful when you want to make sure that the data does not change. They are also slightly faster than lists. Tuples can be used as keys in dictionaries, but lists cannot because they are mutable. So, if you want to store fixed data, tuples are a good choice.

**7. What is a dictionary in Python?**

A dictionary in Python stores data in key-value pairs. Each key is linked to a value, like a word and its meaning in a real dictionary. Dictionaries are written using curly braces:

person = {"name": "Alice", "age": 30}

You can access the value by using the key: person["name"] returns "Alice". You can also add new key-value pairs or update existing ones. Dictionaries are very useful when you need to store related information, like user details or configuration settings. Keys must be unique and immutable, like strings or numbers. Dictionaries are fast and great for looking up data by name or key.

**8. What is a set in Python?**

A set in Python is a collection of unique items. It is unordered, which means the items do not have a fixed position. Sets are written using curly braces:

numbers = {1, 2, 3, 2}

The set automatically removes duplicates, so the output will be {1, 2, 3}. Sets are useful when you want to remove duplicates or check for common values between groups using operations like union, intersection, and difference. Since sets are unordered, you cannot access items using indexes. Sets are also faster than lists when checking if an item exists. They are simple but powerful for handling unique data.

**9. What is the difference between list and tuple in Python?**

Both lists and tuples can store multiple items, but they have some key differences. Lists are **mutable**, which means you can change, add, or remove items after the list is created. Tuples are **immutable**, which means you cannot change them after they are made. Lists are written with square brackets [ ], while tuples use parentheses ( ). Because of immutability, tuples are faster and can be used as keys in dictionaries, while lists cannot. Use a list when your data might change. Use a tuple when your data should stay the same. Both are useful in different situations.

**10. How do you write comments in Python?**

Comments in Python are used to explain what your code is doing. Python ignores comments — they are just for humans reading the code. To write a single-line comment, start the line with #:

# This is a single-line comment

You can also write multi-line comments using triple quotes:

'''  
This is a  
multi-line comment  
'''

Good comments help others (and yourself) understand your code, especially when it’s long or complex. They make your code more readable and easier to maintain. Writing clear comments is considered a good programming habit in any language.

**11. How is Python interpreted?**

Python is an interpreted language, which means the code is not compiled before running. Instead, it is executed line-by-line by the Python interpreter. When you write Python code and run it, the interpreter reads each line, translates it into machine language, and then executes it immediately. This is different from compiled languages like C or Java, where the code is turned into a complete executable file before it runs. Being interpreted makes it easier to test and debug Python code. If there’s an error, Python will stop at that line and show an error message. This feature is great for beginners and for rapid development.

**12. What is indentation in Python and why is it important?**

Indentation in Python refers to the spaces or tabs at the beginning of a line. Unlike many other programming languages, Python uses indentation to define blocks of code. For example, in loops, conditionals, and functions, the indented lines belong to the same code block. If indentation is not done correctly, Python will raise an error and stop running. This makes Python code more readable and clean. Here is a simple example:

if 5 > 2:  
 print("Five is greater than two")

In the above code, the print statement is indented, which shows that it belongs to the if block. Proper indentation is very important in Python programming.

**13. What are Python functions?**

A function in Python is a block of reusable code that performs a specific task. You can define your own functions using the def keyword, or you can use built-in functions like print(), len(), and type(). Functions help you avoid repeating code and make your programs more organized. Here's an example of a simple function:

def greet(name):  
 print("Hello, " + name)

You can call this function like greet("Alice"). Functions can take parameters and can also return values using the return keyword. Using functions makes your code shorter, cleaner, and easier to manage.

**14. What is the difference between a function and a method in Python?**

In Python, a **function** is a block of code that performs a task and is defined using the def keyword. It can be used on its own, outside of any class. For example:

def add(a, b):  
 return a + b

A **method**, on the other hand, is a function that is associated with an object. Methods are defined inside classes and are called using dot notation. For example:

name = "Alice"  
print(name.upper()) # upper() is a method

In simple terms, all methods are functions, but not all functions are methods. Methods always belong to an object or class, while functions do not.

**15. What are arguments and parameters in Python functions?**

**Parameters** are the names you define in a function when you write it. **Arguments** are the actual values you pass to the function when calling it. For example:

def greet(name): # 'name' is a parameter  
 print("Hello, " + name)

greet("Alice") # "Alice" is the argument

In this case, name is a parameter, and "Alice" is the argument passed to the function. Python also supports default arguments, keyword arguments, and variable-length arguments. Understanding how parameters and arguments work is important for writing flexible and reusable functions.

**16. What is the use of the return statement in Python?**

The return statement in Python is used in a function to send a value back to the place where the function was called. It ends the function and passes the result to the caller. Here's an example:

def add(a, b):  
 return a + b

result = add(5, 3)  
print(result) # Output: 8

In this case, the function add returns the sum of a and b, which is then stored in the variable result. If there is no return statement, the function will return None by default. Using return makes functions more useful because they can provide output to be used later.

**17. What is the difference between**del**and**remove()**in Python?**

In Python, both del and remove() are used to delete elements, but they operate differently and are used in different contexts.

* The del statement is a language construct used to delete an item at a specific index from a list or to delete entire variables or slices. It works with all types of objects, including lists, dictionaries, and variables. For example:

nums = [1, 2, 3, 4] del nums[1] # removes the item at index 1 (value 2)

* The remove() method, on the other hand, is a list method that removes the **first occurrence** of a specific value from the list. It raises a ValueError if the item is not found:

nums = [1, 2, 3, 2]   
nums.remove(2) # removes the first 2

In summary, use del when you know the index or want to delete a variable. Use remove() when you want to delete a known value from a list.

**18. What is the difference between for and while loops in Python?**

The for loop and while loop are both used to repeat actions, but they are used in different situations. A for loop is best when you know in advance how many times you want to repeat something. It works well with lists, strings, and ranges:

for i in range(5):  
 print(i)

A while loop is better when you don't know how many times you’ll repeat and want to continue until a certain condition is false:

while i < 5:  
 print(i)  
 i += 1

So, use for when looping through known items, and use while when you need to loop based on a condition.

**19. What is a conditional statement in Python?**

Conditional statements are used to run certain blocks of code only when specific conditions are met. Python uses if, elif, and else for this:

age = 18  
if age >= 18:  
 print("You are an adult")  
else:  
 print("You are a minor")

You can also use elif (short for "else if") to check multiple conditions:

if score >= 90:  
 print("A grade")  
elif score >= 75:  
 print("B grade")  
else:  
 print("C grade")

Conditional statements help your programs make decisions and behave differently based on input or data.

**20. What is the use of the**break**,**continue**, and**pass**statements in Python?**

These three statements control how loops behave:

* break: Stops the loop entirely and exits:

for i in range(5):  
 if i == 3:  
 break  
 print(i) # Prints 0, 1, 2

* continue: Skips the current loop cycle and moves to the next one:

for i in range(5):  
 if i == 3:  
 continue  
 print(i) # Skips 3

* pass: Does nothing. It’s used as a placeholder where code is needed later:

for i in range(5):  
 pass # To be implemented later

These are useful for controlling loops more precisely based on your program’s needs.

**21. What are Python lists and how do you use them?**

A list in Python is a collection of items that can hold different types of values like numbers, strings, or even other lists. Lists are **ordered** and **changeable**, meaning you can update, add, or remove items. You define a list using square brackets:

fruits = ["apple", "banana", "cherry"]

You can access list items by index, like fruits[0] which gives "apple". You can also change values, like fruits[1] = "orange". Python lists have many useful functions like append() to add an item, remove() to delete an item, and sort() to sort the list. Lists are one of the most used data types in Python.

**22. What is the difference between a list and a tuple in Python?**

Both lists and tuples are used to store multiple items, but the **main difference** is that lists are **mutable** (changeable), while tuples are **immutable** (cannot be changed). You create a list with square brackets [], and a tuple with parentheses ():

my\_list = [1, 2, 3]  
my\_tuple = (1, 2, 3)

You can change my\_list[0] = 10, but you cannot change my\_tuple[0]. Tuples are faster and take up less memory than lists. Use tuples when your data should not change, such as coordinates or fixed settings. Lists are better when you need to update, sort, or modify the data.

**23. What are Python dictionaries and how are they used?**

A dictionary in Python is a collection of **key-value pairs**. Each key is unique and maps to a value. You create a dictionary using curly braces {}:

person = {"name": "Alice", "age": 25}

You can access values using keys, like person["name"] which gives "Alice". You can also add or update values, like person["age"] = 30. Dictionaries are useful when you want to store and retrieve data using names or identifiers instead of positions. Some helpful functions include keys(), values(), and items(). They are powerful for storing structured data, like JSON responses or configuration settings.

**24. What are Python sets and what are they used for?**

A set is a collection of **unique** items. It is **unordered**, so the items do not have a fixed position and cannot be accessed by index. Sets are defined using curly braces {}:

my\_set = {1, 2, 3}

If you try to add a duplicate, it will be ignored. Sets are useful for checking membership and removing duplicates. You can use add() to insert elements and remove() to delete them. Python also supports set operations like union (|), intersection (&), and difference (-). Sets are great when you need fast lookups or want to ensure no duplicates exist.

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

A class in Python is a blueprint for creating objects. It defines the structure and behavior (methods and variables) of an object. You create a class using the class keyword:

class Person:  
 def \_\_init\_\_(self, name):  
 self.name = name

def greet(self):  
 print("Hello, my name is " + self.name)

The \_\_init\_\_ method is the constructor and runs when a new object is created. You can create an object like p1 = Person("Alice") and call its method using p1.greet(). Classes help in object-oriented programming and allow you to create reusable code.

**26. What is an object in Python?**

An object in Python is an instance of a class. When you create a class, you’re just defining the structure. But when you create an object using that class, you get a working version with real values. For example:

class Dog:  
 def \_\_init\_\_(self, name):  
 self.name = name

dog1 = Dog("Buddy")

Here, dog1 is an object of the Dog class. It has its own copy of data and can use class methods. In Python, almost everything is an object—strings, lists, functions, and even classes. Objects make code modular, reusable, and organized.

**27. What is inheritance in Python?**

Inheritance in Python allows one class (called a **child** or **subclass**) to get features from another class (called a **parent** or **base class**). It helps in reusing code and building relationships between classes. Here’s a basic example:

class Animal:  
 def speak(self):  
 print("Animal speaks")

class Dog(Animal):  
 def bark(self):  
 print("Dog barks")d = Dog()  
d.speak()  
d.bark()

In this example, the Dog class inherits from Animal, so it can use the speak() method. Inheritance supports code reuse and helps organize code better when working with related classes.

**28. What is polymorphism in Python?**

Polymorphism means “many forms”. In Python, polymorphism allows different classes to have methods with the same name, but different behavior. For example, if two classes have a method named speak(), you can call speak() on any object, and it will behave according to its class:

class Dog:  
 def speak(self):  
 return "Bark"

class Cat:  
 def speak(self):  
 return "Meow"

animals = [Dog(), Cat()]  
for animal in animals:  
 print(animal.speak())

Each object knows how to perform its version of the method. Polymorphism makes code flexible and helps when writing functions that can work with multiple types of objects.

**29. What is encapsulation in Python?**

Encapsulation is a concept in object-oriented programming that hides the internal details of a class and protects data from outside access. In Python, we use **private** variables (with a single or double underscore \_ or \_\_) to indicate that they should not be accessed directly:

class Person:  
 def \_\_init\_\_(self, name):  
 self.\_\_name = name # private variable

def get\_name(self):  
 return self.\_\_name

In this example, \_\_name is private. We access it using the get\_name() method. Encapsulation helps keep your data safe, and only allows access through defined methods, making your code more secure and easier to maintain.

**30. What is abstraction in Python?**

Abstraction means showing only the essential features and hiding the unnecessary details. It helps reduce complexity and allows you to focus on what an object does, not how it does it. In Python, abstraction is often implemented using **abstract classes** and **methods** from the abc module:

from abc import ABC, abstractmethod

class Animal(ABC):  
 @abstractmethod  
 def make\_sound(self):  
 pass

class Dog(Animal):  
 def make\_sound(self):  
 print("Bark")

Here, Animal is an abstract class, and make\_sound() must be implemented in any child class. Abstraction helps in designing clean interfaces and focusing on high-level functionality.

**31. What is the difference between**\*args**and**\*\*kwargs**in Python?**

* \*args allows a function to accept **any number of positional arguments**, packed as a tuple.
* \*\*kwargs allows a function to accept **any number of keyword arguments**, packed as a dictionary.

Example:

def demo(\*args, \*\*kwargs):  
 print(args)  
 print(kwargs)  
  
demo(1, 2, a=3, b=4)

These are useful for creating flexible functions, wrappers, and decorators.

**32. What is the**if \_\_name\_\_ == "\_\_main\_\_"**statement used for?**

The if \_\_name\_\_ == "\_\_main\_\_" statement is used to control the execution of code in Python scripts. When a Python file is run directly, the special built-in variable \_\_name\_\_ is set to "\_\_main\_\_". However, when that file is imported as a module into another script, \_\_name\_\_ is set to the module's name instead.

This allows developers to write code that acts differently depending on whether it’s run directly or imported. It’s commonly used to encapsulate the script’s entry point:

def main():  
 print("Running as a script")  
  
if \_\_name\_\_ == "\_\_main\_\_":  
 main()

This is particularly useful in larger applications and during unit testing, as it allows for better organization and reuse of code.

**33. What is a**defaultdict**in Python?**

A defaultdict is a subclass of the built-in dict class that provides a default value for non-existent keys. This prevents KeyError exceptions and is particularly useful when counting items or grouping data.

It requires a factory function to specify the default value:

from collections import defaultdict  
dd = defaultdict(int)  
dd["apple"] += 1 # no error even though "apple" didn't exist

This is much cleaner than checking for key existence manually using get() or if statements. Common factories include int, list, and set.

**34. What is list comprehension in Python?**

List comprehension is a concise way to create lists in Python. Instead of using loops, you can use a single line of code. It improves readability and performance. Here’s a basic example:

squares = [x\*x for x in range(5)]

This creates a list [0, 1, 4, 9, 16]. You can also add conditions:

even = [x for x in range(10) if x % 2 == 0]

List comprehensions are useful when you want to transform or filter data quickly. They make your code shorter, cleaner, and easier to understand than using a full for loop.

**35. What is a higher-order function in Python?**

A higher-order function is any function that either accepts another function as an argument, or returns a function as its result. Python supports higher-order functions natively, which makes it a flexible language for functional-style programming.

Examples of built-in higher-order functions in Python include map(), filter(), and sorted().

Example:

def apply\_twice(func, value):  
 return func(func(value))  
  
def square(x):  
 return x \* x  
print(apply\_twice(square, 2)) # Output: 16

Higher-order functions promote reusability and abstraction, allowing more expressive and concise code.

**36. How does Python handle memory management?**

Python manages memory automatically using a system called **garbage collection**. This system keeps track of all objects and frees up memory that is no longer being used. Python also uses reference counting — each object keeps a count of how many references point to it. When that count reaches zero, the memory is released. Python’s memory management is done by the **Python memory manager**, and it includes private heap space where all objects and data structures are stored. As a developer, you don’t usually need to manage memory directly, but understanding how it works can help write better, more efficient code.

**37. What is multiple inheritance in Python?**

Multiple inheritance is when a class inherits from more than one parent class. Python fully supports this, which allows a class to combine functionality from multiple sources.

Example:

class A:  
 def greet(self):  
 print("Hello from A")  
class B:  
 def greet(self):  
 print("Hello from B")  
class C(A, B):  
 pass  
c = C()  
c.greet() # Uses A's method due to MRO

While powerful, multiple inheritance can make the class hierarchy hard to manage, so it should be used with care. The super() function and MRO help mitigate the complexity.

**38. What is a package in Python?**

A **package** in Python is a collection of modules organized in directories. It allows you to group related modules together. A package is a folder that contains an \_\_init\_\_.py file, which tells Python that the folder should be treated as a package. Example structure:

mypackage/  
│  
├── **\_\_init\_\_**.py  
├── module1.py  
└── module2.py

You can import modules from the package using dot notation:

from mypackage import module1

Packages help in organizing large applications and reusing code across projects. Python also has many third-party packages that can be installed using tools like pip.

**39. What is the purpose of**\_\_init\_\_.py**in Python packages?**

The \_\_init\_\_.py file in Python marks a directory as a package so that its modules can be imported. Without this file, Python won’t recognize the folder as a package in older versions (though in modern Python, it’s optional). This file can be empty or contain initialization code for the package. For example, you might import key modules or define variables inside it:

# \_\_init\_\_.py  
from .module1 import function1

With this, users can simply do from package import function1 instead of importing the whole module. It helps organize imports and controls how packages behave during import.

**40. How do you handle exceptions in Python?**

In Python, you handle errors and exceptions using try, except, and optionally finally. This allows your program to continue running even if something goes wrong. Here's an example:

try:  
 num = int(input("Enter a number: "))  
 result = 10 / num  
except ZeroDivisionError:  
 print("Cannot divide by zero!")  
except ValueError:  
 print("Please enter a valid number.")  
finally:  
 print("This always runs.")

The try block runs the risky code. If there’s an error, Python checks for a matching except block. The finally block always runs, whether an error occurred or not. Exception handling makes your programs more robust and user-friendly.

**41. What is the difference between**break**,**continue**, and**pass**in Python?**

In Python, break, continue, and pass are control flow statements, but each one serves a different purpose.

* break: It is used to exit a loop completely, even if the loop condition is still true. Once break is encountered, the loop stops running.

for i in range(5):  
 if i == 3:  
 break  
 print(i)

* continue: It skips the current iteration and moves to the next one without stopping the loop.

for i in range(5):  
 if i == 3:  
 continue  
 print(i)

* pass: It does nothing and is used as a placeholder where code is required syntactically but no action is needed.

for i in range(5):  
 if i == 3:  
 pass  
 print(i)

These are useful for controlling how loops and conditionals behave during execution.

**42. What are Python decorators?**

A **decorator** in Python is a function that takes another function as input and adds extra functionality to it, without changing its original structure. It is often used to modify the behavior of a function or method dynamically.

def decorator(func):  
 def wrapper():  
 print("Before function call")  
 func()  
 print("After function call")  
 return wrapper

@decorator  
def greet():  
 print("Hello!")greet()

Here, @decorator wraps the greet() function and adds extra code before and after it. Decorators are used often in logging, authentication, timing, and access control. They help you write cleaner, reusable, and more readable code.

**43. What is a**Counter**in Python?**

Counter is a class from the collections module that helps count occurrences of elements in an iterable. It returns a dictionary-like object where elements are stored as keys and counts as values.

Example:

from collections import Counter  
c = Counter("banana")  
print(c) # Counter({'a': 3, 'n': 2, 'b': 1})

It supports most dictionary operations and provides extra methods like .most\_common() and .elements() for retrieving data in specific formats. It is extremely useful for statistics, frequency analysis, and text processing.

**44. What are closures in Python?**

Closures are functions that remember the values of variables from their enclosing lexical scope even after that scope has finished executing. In other words, a closure allows a function to access variables from an outer function that has already returned.

Here’s a simple example:

def outer(msg):  
 def inner():  
 print(msg)  
 return inner

greet = outer("Hello")  
greet() # prints "Hello"

In the above code, the function inner() forms a closure—it remembers the variable msg from its enclosing function outer(). Closures are useful for building function factories, decorators, and keeping state in a clean and elegant way.

**45. What are \*args and kwargs in Python?**

In Python, \*args and \*\*kwargs are used in function definitions to allow the function to accept a variable number of arguments.

* \*args collects extra **positional** arguments into a tuple.
* \*\*kwargs collects extra **keyword** arguments into a dictionary.

def example(\*args, \*\*kwargs):  
 print(args)  
 print(kwargs)

example(1, 2, 3, name="Alice", age=25)

This would print:

(1, 2, 3)  
{'name': 'Alice', 'age': 25}

These features make your functions flexible and reusable. You can call them with different numbers of parameters without changing the function definition.

**46. What is the LEGB rule in Python?**

LEGB stands for **Local → Enclosing → Global → Built-in**, and it defines the order in which Python searches for variables:

1. **Local**: Names inside the current function.
2. **Enclosing**: Names in outer (but not global) functions if nested.
3. **Global**: Names defined at the top-level of a script or module.
4. **Built-in**: Python’s predefined names like len() or str.

Example:

x = "global"  
def outer():  
 x = "enclosing"  
 def inner():  
 x = "local"  
 print(x)  
 inner()  
outer() # prints "local"

Understanding LEGB is crucial for working with nested functions, closures, and scoping bugs.

**47. How can you handle file operations in Python?**

Python provides easy ways to work with files using built-in functions like open(), read(), write(), and close(). Here's a simple example of reading a file:

with open("example.txt", "r") as file:  
 content = file.read()  
 print(content)

The with statement automatically closes the file. You can also write to a file:

with open("example.txt", "w") as file:  
 file.write("Hello, Python!")

Use modes like "r" for reading, "w" for writing, and "a" for appending. File operations are useful for data storage, configuration, and logging in applications.

**48. What are Python’s built-in data types?**

Python has several built-in data types that are categorized into different groups:

* **Numeric types**: int, float, complex
* **Sequence types**: list, tuple, range
* **Text type**: str
* **Set types**: set, frozenset
* **Mapping type**: dict
* **Boolean type**: bool
* **Binary types**: bytes, bytearray, memoryview

Each type serves different purposes. For example, use int for whole numbers, str for text, list for ordered groups, and dict for key-value pairs. Understanding data types helps you store and manipulate data correctly in your programs.

**49. What is the difference between mutable and immutable types in Python?**

**Mutable** types can be changed after creation. Examples include list, dict, and set. You can add, remove, or change elements in these types.

**Immutable** types cannot be changed once created. Examples include int, float, str, and tuple. If you try to modify them, Python creates a new object instead.

x = "hello"  
x = x + " world" # Creates a new string

Knowing which types are mutable and which are not helps you avoid bugs, especially when passing variables into functions. It also affects performance and how memory is managed.

**50. How do you define and use a function in Python?**

To define a function in Python, you use the def keyword followed by the function name and parentheses. You can pass arguments into the function and return values using the return statement.

def greet(name):  
 return f"Hello, {name}!"

message = greet("Alice")  
print(message)

Functions make your code reusable and organized. Instead of repeating the same logic, you put it in a function and call it whenever needed. You can also have default arguments, variable-length arguments, and even functions inside functions.

**51. What is a Python metaclass?**

In Python, a metaclass is a class of a class — it defines how classes behave. While classes define how objects behave, metaclasses define how classes themselves are created.

By default, all classes in Python are instances of type, the default metaclass. However, you can define custom metaclasses by inheriting from type, and use them to automatically modify class attributes or behavior at the time of class creation.

Use cases include:

* Enforcing coding conventions (e.g., attribute naming rules)
* Automatically registering classes
* Singleton pattern implementations

Although powerful, metaclasses are an advanced feature and should be used with care to avoid unnecessary complexity.

**52. What is duck punching (monkey patching) in Python?**

Monkey patching (or duck punching) is the practice of changing or extending the behavior of libraries, classes, or modules at runtime.

Example:

import math  
math.sqrt = lambda x: "No square roots allowed"  
print(math.sqrt(9)) # Outputs: No square roots allowed

This can be useful for testing or hotfixes but is generally discouraged in production code because it can lead to unpredictable behavior and maintenance challenges.

**53. What is a lambda function in Python?**

A **lambda function** is a small anonymous function in Python. It’s used when you need a simple function for a short period and don’t want to formally define it using def. Lambda functions can take any number of arguments but only contain one expression.

Basic syntax:

lambda arguments: expression

Example:

add = lambda x, y: x + y  
print(add(3, 5)) # Output: 8

Lambdas are useful in places where a quick function is needed, like with map(), filter(), and sorted().

nums = [5, 2, 9]  
sorted\_nums = sorted(nums, key=lambda x: -x)

Although powerful, for complex operations, using regular functions is more readable.

**54. What is method resolution order (MRO) in Python?**

MRO is the order in which Python looks for methods in a class hierarchy. It determines which method gets called when multiple inheritance is involved. Python uses the C3 linearization algorithm to compute this order.

You can view a class’s MRO using:

print(ClassName.\_\_mro\_\_)

class A: pass  
class B(A): pass  
class C(A): pass  
class D(B, C): pass  
  
print(D.\_\_mro\_\_)

This is especially useful in understanding complex inheritance and avoiding bugs due to unexpected method overrides.

**55. What is exception handling in Python?**

**Exception handling** is used in Python to catch and respond to errors that occur during program execution. Instead of crashing the program, you can handle the error gracefully using try, except, else, and finally blocks.

Here’s how it works:

try:  
 result = 10 / 0  
except ZeroDivisionError:  
 print("You can't divide by zero!")  
else:  
 print("No errors occurred.")  
finally:  
 print("This will always run.")

* try block contains code that might raise an error.
* except block handles the error.
* else runs if no error occurs.
* finally always runs, useful for cleanup.

Using exceptions helps in building robust applications that can recover from unexpected situations.

**56. What are Python’s magic methods?**

**Magic methods** in Python are special methods with double underscores at the beginning and end of their names. They’re also known as **dunder methods**. Python uses these methods to perform operator overloading and other behaviors internally.

Get Shirsh Shukla’s stories in your inbox

Join Medium for free to get updates from this writer.



Subscribe

Some common magic methods are:

* \_\_init\_\_: Constructor, called when an object is created.
* \_\_str\_\_: Returns a string representation of the object.
* \_\_len\_\_: Returns the length using len().
* \_\_add\_\_: Defines behavior for +.

Example:

class Book:  
 def \_\_init\_\_(self, title):  
 self.title = title

def \_\_str\_\_(self):  
 return f"Book: {self.title}"book = Book("Python 101")  
print(book) # Output: Book: Python 101

Magic methods make classes act like built-in types, improving readability and flexibility.

**57. What is the difference between**is**and**==**in Python?**

In Python:

* == checks if **values** of two variables are equal.
* is checks if two variables point to the **same object in memory**.

Example:

a = [1, 2, 3]  
b = [1, 2, 3]

print(a == b) # True, because contents are same  
print(a is b) # False, because they are two different objects

is is often used when comparing objects like None:

if my\_var is None:  
 print("Value is None")

Use == for equality of content and is when you care about identity, such as comparing singletons or cached objects.

**58. What is the difference between a tuple and a list in Python?**

Both tuples and lists are used to store multiple items, but:

* **List** is mutable: you can add, remove, or change items.
* **Tuple** is immutable: once created, it cannot be changed.

my\_list = [1, 2, 3]  
my\_tuple = (1, 2, 3)

Lists are defined with square brackets [], and tuples with parentheses ().

Lists are commonly used when data needs to change during runtime, while tuples are ideal for fixed data or when you want to ensure the data stays unchanged. Also, because tuples are immutable, they can be used as dictionary keys, unlike lists.

**59. What is duck typing in Python?**

**Duck typing** is a concept from dynamic typing. In Python, it means that the type or class of an object is less important than the methods or operations it supports. The idea is: *“If it walks like a duck and quacks like a duck, it’s a duck.”*

Example:

class Duck:  
 def quack(self):  
 print("Quack!")

class Person:  
 def quack(self):  
 print("I can quack too!")def make\_quack(thing):  
 thing.quack()

make\_quack(Duck())  
make\_quack(Person())

Even though Duck and Person are different classes, both can be passed because they have a quack() method. This flexibility makes Python code more reusable and simple.

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

Python uses a combination of **reference counting** and a **garbage collector** to manage memory.

* Each object has a reference count: when the count reaches zero, it is deleted.
* Python also uses a garbage collector to handle **circular references**, where two objects refer to each other but are no longer used.

You don’t usually need to manage memory manually. Python handles it for you. But you can still check or influence it using the gc module:

import gc  
gc.collect()

Also, memory is allocated in private heaps managed by the Python interpreter. Efficient memory handling is part of Python’s design to help developers focus on logic instead of low-level resource management.

**61. What is the purpose of the**gc**module in Python?**

The gc (garbage collection) module provides access to Python's automatic memory management. It allows you to manually trigger garbage collection, monitor objects that aren’t being collected, and control collection thresholds.

For example:

import gc  
gc.collect() # Manually triggers garbage collection

It’s particularly useful when dealing with circular references — where two objects refer to each other, preventing their reference count from dropping to zero. The gc module helps identify and clean these unreachable objects.

**62. What are Python iterators?**

An **iterator** in Python is an object that allows you to loop over its elements, one at a time. It must implement the \_\_iter\_\_() and \_\_next\_\_() methods. You can get an iterator from any iterable (like lists, tuples, sets) using the iter() function. Then, you use next() to get the next item.

Example:

my\_list = [10, 20, 30]  
it = iter(my\_list)  
print(next(it)) # 10  
print(next(it)) # 20

When there are no more items, next() raises a StopIteration error. Iterators are memory-efficient because they don’t store the whole sequence in memory, which is helpful for large datasets or file reading.

**63. What is the difference between**deepcopy()**and**copy()**?**

In Python, copy() and deepcopy() both come from the copy module and are used to duplicate objects, but they behave differently:

* copy.copy() creates a **shallow copy** of an object. It copies the outer object, but not the nested objects inside it. So changes to inner objects affect both copies.
* copy.deepcopy() creates a **completely independent clone**, including all nested objects. Changes to one object do not affect the other.

Example:

import copy

original = [[1, 2], [3, 4]]  
shallow = copy.copy(original)  
deep = copy.deepcopy(original)original[0][0] = 99  
print(shallow[0][0]) # 99 (affected)  
print(deep[0][0]) # 1 (not affected)

Use deepcopy() when you need a completely separate object structure.

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

**Slicing** in Python lets you extract a portion of a list, string, or tuple using a range of indexes. The basic syntax is object[start:stop:step].

Example:

my\_list = [0, 1, 2, 3, 4, 5]  
print(my\_list[1:4]) # [1, 2, 3]  
print(my\_list[::2]) # [0, 2, 4]  
print(my\_list[::-1]) # [5, 4, 3, 2, 1, 0]

* start is the index to begin from (inclusive),
* stop is where to end (exclusive)
* step tells how many items to skip.

Slicing is powerful for reversing sequences, picking even-indexed items, or extracting parts of text. It works with strings too:

text = "Python"  
print(text[1:4]) # "yth"

**65. What is the difference between**append()**and**extend()**in lists?**

Both append() and extend() are used to add elements to a list, but they do it differently:

* append() adds a **single element** to the end of the list, even if it’s another list.
* extend() takes an **iterable** and adds **each element** from it to the list.

Example:

a = [1, 2]  
a.append([3, 4])  
print(a) # [1, 2, [3, 4]]

b = [1, 2]  
b.extend([3, 4])  
print(b) # [1, 2, 3, 4]

So, append() adds the whole object, while extend() breaks it apart and adds each item individually. Use append() when you want to keep an item together; use extend() when you want to expand the list.

**66. What is the purpose of**enumerate()**in Python?**

The enumerate() function is used when you need both the **index and value** while looping over an iterable. It adds a counter to the iterable and returns it as an enumerate object, which you can convert into a list or use in a loop.

Example:

fruits = ['apple', 'banana', 'cherry']  
for index, fruit in enumerate(fruits):  
 print(index, fruit)

Output:

0 apple   
1 banana   
2 cherry

This is much cleaner than using range(len(fruits)). You can also start the index at a custom number by using enumerate(fruits, start=1). It’s helpful when you’re processing lists, keeping track of positions, or labeling items in a report or UI.

**67. What is the difference between**filter()**,**map()**, and**reduce()**?**

All three are functional programming tools in Python:

* map() applies a function to **every item** in an iterable and returns a new iterable.
* filter() applies a function that returns True or False, and only keeps items where the function returns True.
* reduce() repeatedly applies a function to **accumulate** a result (from functools module).

Example:

from functools import reduce

nums = [1, 2, 3, 4]print(list(map(lambda x: x \* 2, nums))) # [2, 4, 6, 8]  
print(list(filter(lambda x: x % 2 == 0, nums))) # [2, 4]  
print(reduce(lambda x, y: x + y, nums)) # 10

map() transforms data, filter() selects data, and reduce() combines data. They help write clean, short code for data processing.

**68. What is the use of**zip()**in Python?**

The zip() function is used to combine two or more iterables into a single iterable of tuples, where the i-th tuple contains the i-th element from each iterable. It stops at the shortest input length.

Example:

names = ['Alice', 'Bob']  
scores = [85, 90]  
zipped = zip(names, scores)  
print(list(zipped)) # [('Alice', 85), ('Bob', 90)]

It’s great for looping over multiple lists at once:

for name, score in zip(names, scores):  
 print(f"{name} scored {score}")

If the input lists are of unequal length, the extra items are ignored. You can also unzip using zip(\*zipped\_data).

**69. What is**\*args**and**\*\*kwargs**in Python?**

In Python:

* \*args lets you pass a **variable number of positional arguments** to a function.
* \*\*kwargs lets you pass a **variable number of keyword arguments** (as a dictionary).

Example:

def demo(\*args, \*\*kwargs):  
 print("Args:", args)  
 print("Kwargs:", kwargs)

demo(1, 2, 3, a=4, b=5)

Output:

Args: (1, 2, 3)  
Kwargs: {'a': 4, 'b': 5}

These are useful when writing flexible functions that can accept any number of inputs. It helps when wrapping other functions or when you don’t know in advance how many parameters might be passed.

**70. What is recursion in Python?**

**Recursion** is when a function calls itself to solve a smaller part of a problem until it reaches a base case. It’s commonly used in problems like calculating factorials, Fibonacci numbers, and tree traversals.

Example:

def factorial(n):  
 if n == 0:  
 return 1  
 return n \* factorial(n - 1)

print(factorial(5)) # 120

Every recursive function must have a base case to prevent infinite recursion. Python limits recursion depth by default (about 1000 calls). You can check it using sys.getrecursionlimit().

Recursion makes some problems easier to solve, though it may use more memory than loops.

**71. What is a Python module?**

A **Python module** is a file containing Python code, usually saved with a .py extension. Modules help organize and reuse code across multiple programs. You can define functions, variables, and classes inside a module and then import them into other files using the import statement.

For example, if you create a file math\_utils.py with a function:

def add(a, b):  
 return a + b

You can use it in another file like this:

import math\_utils  
print(math\_utils.add(2, 3))

Python has many built-in modules like math, random, and datetime. You can also create your own custom modules to keep your code clean, organized, and maintainable, especially in large projects.

**72. What are weak references in Python?**

Weak references allow Python to refer to an object without increasing its reference count. This means the object can still be garbage-collected, which is useful when managing caches or graphs where circular references can occur.

Python provides weakref module:

import weakref  
  
class MyClass:  
 pass  
obj = MyClass()  
r = weakref.ref(obj)  
print(r()) # Returns the object  
del obj  
print(r()) # Returns None (object is collected)

This mechanism helps avoid memory leaks in large or long-running applications.

**73. What is the**super()**function in Python?**

The super() function returns a proxy object that delegates method calls to a parent or sibling class. It’s commonly used in class constructors (\_\_init\_\_) to ensure proper initialization in inheritance chains.

Example:

class Parent:  
 def \_\_init\_\_(self):  
 print("Parent initialized")  
  
class Child(Parent):  
 def \_\_init\_\_(self):  
 super().\_\_init\_\_()  
 print("Child initialized")  
c = Child()

Using super() helps support maintainable, extensible code and is critical in multiple inheritance where calling the correct parent method matters.

**74. What is the purpose of**\_\_init\_\_.py**?**

The \_\_init\_\_.py file is used to mark a directory as a **Python package**. Without it, Python doesn’t recognize the folder as part of the package system. Even if it's empty, it’s still needed (especially in older Python versions).

For example, in this structure:

my*\_package/* ***\_\_init\_\_****.py  
 module1.py*

You can now do:

from my\_package import module1

Besides identifying packages, \_\_init\_\_.py can also run initialization code or expose specific classes/functions when the package is imported. For instance, you can import functions from other modules inside \_\_init\_\_.py, making them accessible directly from the package. It helps organize your project structure for modular and maintainable code.

**75. What is**collections.deque**and why is it better than a list for some use cases?**

collections.deque (double-ended queue) is a part of Python’s standard collections module. It is a specialized list-like container optimized for fast appends and pops from both ends.

While Python lists are efficient for append and pop operations at the **end**, they are slow when inserting or deleting items at the **beginning** because all elements must be shifted.

deque solves this by using a doubly linked list internally, allowing O(1) operations on both ends:

from collections import deque  
dq = deque([1, 2, 3])  
dq.appendleft(0) # fast insert at the beginning  
dq.pop() # fast removal from end

It is ideal for queue and stack implementations, sliding windows, and breadth-first searches.

**76. What are**namedtuple**s and why are they useful?**

A namedtuple is a factory function in the collections module that returns a subclass of tuple with named fields. This makes your code more readable and self-documenting, while still being as memory-efficient as a regular tuple.

Example:

from collections import namedtuple  
Point = namedtuple("Point", ["x", "y"])  
p = Point(1, 2)  
print(p.x, p.y) # Access by name instead of index

Use namedtuple when you need immutable data structures with readable fields—ideal for data modeling, configuration, and return values.

**77. What are Python comprehensions?**

Python **comprehensions** are a concise way to create lists, sets, or dictionaries using a single line of code. They are often used as a more readable alternative to loops.

Example of list comprehension:

squares = [x\*x for x in range(5)]

This creates [0, 1, 4, 9, 16]. Similarly, you can use:

* **Set comprehension**: {x\*x for x in range(5)}
* **Dict comprehension**: {x: x\*x for x in range(5)}

You can also add conditions:

even\_squares = [x\*x for x in range(10) if x % 2 == 0]

Comprehensions make the code cleaner and easier to read compared to using traditional loops, especially when the operation is simple and directly maps from input to output.

**78. What is the difference between**@staticmethod**and**@classmethod**in Python?**

Both @staticmethod and @classmethod are decorators in Python, but they serve different purposes. A @staticmethod is a function within a class that does not access instance (self) or class (cls) variables. It is used when the logic relates to the class but doesn’t need its state. A @classmethod, on the other hand, takes the class itself as the first parameter (cls) and can modify class-level data. It’s useful for alternative constructors or utility functions that operate on class variables. Use @staticmethod for general utility functions, and @classmethod when you need access to class properties.

**79. What are Python annotations?**

Python annotations are a way to add **type hints** to function parameters and return types. Introduced in PEP 484, they help improve code readability, allow static type checkers like mypy to analyze your code, and assist with IDE autocompletion.

Annotations do not affect runtime behavior by default; they are just metadata. For example:

def greet(name: str) -> str:  
 return f"Hello, {name}"

Here, name is expected to be a string, and the function returns a string. You can also access annotations using the \_\_annotations\_\_ attribute.

While annotations are optional, they are increasingly used in production code for documentation, validation, and tooling support.

**80. What are type hints and why are they useful in Python?**

Type hints (also called type annotations) allow developers to specify the expected data types of function parameters and return values. Introduced in Python 3.5 via PEP 484, they don’t change how code runs but help with readability, autocompletion, and static type checking.

Example:

def add(a: int, b: int) -> int:  
 return a + b

Tools like mypy can then analyze your code for type consistency without running it. Type hints are valuable in large codebases and collaborative projects.

**81. What is a namespace in Python?**

A **namespace** in Python refers to a space where names are mapped to objects. Think of it as a dictionary where the keys are variable names and the values are the objects those names refer to. Python uses namespaces to keep track of all the names in your program, like variable names, function names, class names, etc.

There are four types of namespaces in Python:

* **Built-in**: Contains built-in functions like print() and len().
* **Global**: Contains names defined at the top-level of a script or module.
* **Enclosing**: Relevant for nested functions (outer function scope).
* **Local**: Inside a function or block.

For example:

x = 10 # Global namespace

def func():  
 y = 5 # Local namespace

Namespaces prevent naming conflicts and help Python know which variable you’re referring to in different parts of the program. You can access namespaces using functions like globals() and locals().

**82. What is the difference between global and local variables?**

A **local variable** is defined inside a function and can only be used within that function. A **global variable**, on the other hand, is defined outside any function and can be accessed from anywhere in the code, including inside functions.

Example:

x = 5 # Global

def my\_func():  
 x = 10 # Local  
 print(x)

my\_func() # Prints 10  
print(x) # Prints 5

If you want to change the global variable inside a function, you must use the global keyword:

def my\_func():  
 global x  
 x = 20

Local variables help keep functions independent, while global variables can be accessed across functions but may lead to unexpected behavior if not managed properly.

**83. What is a Python set and how is it different from a list?**

A **set** in Python is an unordered collection of unique elements. It is defined using curly braces {} or the set() function. Unlike lists, sets do not allow duplicate values, and they are not indexed, meaning you cannot access items using indexes like my\_set[0].

Example:

my\_list = [1, 2, 2, 3]  
my\_set = set(my\_list) # {1, 2, 3}

Key differences:

* **Uniqueness**: Sets automatically remove duplicates.
* **Order**: Lists maintain order; sets do not.
* **Mutability**: Both are mutable, but sets only contain immutable elements.
* **Operations**: Sets support mathematical operations like union, intersection, and difference.

Use sets when you need to store unique values or perform set-based operations efficiently.

**84. How does exception handling work in Python?**

Exception handling in Python is done using try, except, finally, and else blocks. It helps your program handle unexpected errors without crashing. You wrap the code that might throw an error in a try block, then use except to handle specific or general exceptions.

Example:

try:  
 x = 1 / 0  
except ZeroDivisionError:  
 print("Cannot divide by zero.")  
finally:  
 print("This will always run.")

You can also use else to run code if no exception occurs:

try:  
 x = 5  
except:  
 print("Error")  
else:  
 print("No error")

This system helps in writing robust and error-tolerant applications. Always catch specific exceptions instead of using a broad except:.

**85. What is the difference between**repr()**and**str()**in Python?**

Both repr() and str() are used to get string representations of objects, but for different purposes:

* str() is for **user-friendly** display (e.g., print() output).
* repr() is for **developer-oriented** output that ideally could be used to recreate the object.

Example:

s = "Hello"  
print(str(s)) # Output: Hello  
print(repr(s)) # Output: 'Hello'

You can override \_\_str\_\_() and \_\_repr\_\_() in custom classes for better control over their behavior.

**86. What is None in Python?**

None is a special constant in Python that represents the **absence of a value** or a **null value**. It is often used to indicate that a variable doesn’t have any meaningful data yet. It’s an object of its own datatype — NoneType.

Example:

x = None  
if x is None:  
 print("x has no value")

Functions that don’t explicitly return anything will return None by default:

def greet():  
 print("Hello")

print(greet()) # Outputs "Hello" and then "None"

None is not the same as 0, False, or an empty string. It’s used in comparisons, default arguments, and when checking if a variable was assigned any value or not.

**87. What is the use of the**id()**function in Python?**

The id() function in Python returns the **unique identifier** of an object, which is its memory address in CPython (the standard Python implementation). This is useful for checking whether two variables refer to the same object in memory.

Example:

a = [1, 2, 3]  
b = a  
print(id(a), id(b)) # Same id

Even if two objects have the same value, they may not have the same id:

x = [1, 2]  
y = [1, 2]  
print(id(x) == id(y)) # False

You can combine id() with the is keyword to check identity. This is especially useful when dealing with mutable and immutable types and understanding object references.

**88. What is a recursive lambda function?**

A recursive lambda is a lambda function that calls itself to solve problems. However, since lambda functions in Python are anonymous and can’t directly refer to themselves, we must assign them to a variable.

Example:

factorial = lambda n: 1 if n == 0 else n \* factorial(n - 1)  
print(factorial(5)) # Output: 120

This is a simple, elegant way to define recursion in functional style, though it’s typically more readable using a def function for more complex logic.

**89. What is a ternary operator in Python?**

The **ternary operator** in Python is a way to write simple if-else statements in one line. It’s also called a **conditional expression**.

Syntax:

x = value\_if\_true if condition else value\_if\_false

Example:

age = 18  
status = "Adult" if age >= 18 else "Minor"

It makes code more concise and readable for small decisions. But avoid using it for complex logic, as it can reduce readability. It’s commonly used when assigning values based on a quick condition.

**90. What are Python assertions?**

**Assertions** are a debugging tool used to test if a condition is true. If the condition is false, the program will raise an AssertionError. They help catch bugs early by checking if your assumptions in the code hold true.

Example:

x = 5  
assert x > 0, "x must be positive"

If x is less than or equal to 0, the assertion will fail. You can provide a custom error message after the comma.

Assertions are mostly used in development and testing phases. In production, they can be disabled with the -O (optimize) switch. They are not a replacement for proper error handling but are very useful in spotting bugs quickly.

**91. What is the use of**zip()**in Python?**

The zip() function in Python is used to combine two or more iterables (like lists or tuples) into a single iterable of tuples. It pairs the elements from each iterable based on their position (index). If the iterables are of different lengths, zip() stops at the shortest one.

Example:

names = ["Alice", "Bob", "Charlie"]  
scores = [85, 90, 78]  
zipped = zip(names, scores)  
print(list(zipped)) # [('Alice', 85), ('Bob', 90), ('Charlie', 78)]

You can use zip() for many tasks like combining data, iterating over multiple sequences at once, or transposing rows and columns in matrices. It's efficient and easy to use, often seen in data processing, file merging, or where multiple iterables need to be processed in parallel.

**92. What is the purpose of**dir()**in Python?**

The dir() function is a built-in utility that returns a list of all attributes and methods (including inherited ones) available for a given object. It’s widely used for introspection, helping developers understand what functionality is available on an object without reading the source code or documentation.

For example:

print(dir([])) # Shows all methods and attributes for a list

When called with no arguments, dir() returns the list of names in the current local scope.

This tool is especially useful in REPL environments or debugging sessions, helping you explore objects dynamically and understand the Python object model better.

**93. What is the**map()**function in Python?**

The map() function applies a given function to all items in an iterable and returns a map object (which is an iterator). It’s a clean way to apply transformations to a list or tuple without writing a loop.

Syntax:

map(function, iterable)

Example:

numbers = [1, 2, 3, 4]  
squared = map(lambda x: x\*\*2, numbers)  
print(list(squared)) # [1, 4, 9, 16]

map() is often used when you need to apply the same function to every element. It's efficient and can be combined with other functional programming tools like filter() or reduce(). It keeps code short and readable.

**94. What is the**filter()**function in Python?**

The filter() function is used to filter elements from an iterable based on a condition. It returns an iterator containing only the elements for which the function returns True.

Syntax:

filter(function, iterable)

Example:

numbers = [1, 2, 3, 4, 5]  
even = filter(lambda x: x % 2 == 0, numbers)  
print(list(even)) # [2, 4]

It is useful when you want to keep elements that meet certain criteria and discard the rest. filter() makes your code more expressive and avoids manual loops with conditional checks.

**95. What is the**reduce()**function in Python?**

The reduce() function from the functools module applies a function to the items of a sequence and reduces it to a single value. It processes the sequence pairwise.

Syntax:

from functools import reduce  
reduce(function, iterable)

Example:

from functools import reduce  
numbers = [1, 2, 3, 4]  
result = reduce(lambda x, y: x + y, numbers)  
print(result) # 10

You can use reduce() for operations like summing, multiplying, or combining values. It's powerful, but sometimes less readable than a loop or sum(). Use it when you need to process a list into a single result step by step.

**96. What is a**frozenset**in Python?**

A frozenset is the immutable version of a Python set. It cannot be modified after creation, which means you cannot add or remove elements. This immutability makes it hashable, allowing it to be used as a key in dictionaries or as elements in other sets.

Example:

fs = frozenset([1, 2, 3])

It supports all set operations like union, intersection, and difference. Use frozenset when you need to ensure the contents of a set do not change, especially when dealing with caching, memoization, or set-based keys.

**97. What is the difference between**yield**and**return**in Python?**

Both yield and return are used in functions, but they work very differently. return ends a function and sends back a value. Once return is called, the function ends. On the other hand, yield pauses the function and sends a value but keeps the function state alive for the next call.

With yield, the function becomes a generator. You can use next() to get the next value from it.

Example:

def gen():  
 yield 1  
 yield 2

g = gen()  
print(next(g)) # 1  
print(next(g)) # 2

yield is memory-efficient and great for iterating over large or infinite data.

**98. What is the Global Interpreter Lock (GIL) in Python?**

The Global Interpreter Lock (GIL) is a mechanism in the CPython interpreter that prevents multiple native threads from executing Python bytecodes at the same time. It exists to protect access to Python objects, ensuring thread safety. Because of the GIL, multi-threaded Python programs may not achieve true parallelism on multi-core CPUs for CPU-bound tasks. However, for I/O-bound operations like file handling or network requests, threading can still be beneficial. If you need real parallelism in CPU-heavy operations, consider using multiprocessing or external tools like NumPy or C extensions.

**99. How do you handle memory management in Python?**

Python handles memory management automatically using a technique called **garbage collection**. The Python interpreter keeps track of objects and their references using a reference counting system. When an object’s reference count drops to zero, it is deleted automatically.

In addition to reference counting, Python has a **cyclic garbage collector** to clean up objects involved in reference cycles. You can also manage memory manually using the gc module, but usually, it's not needed.

Example:

import gc  
gc.collect() # Triggers garbage collection

Good practices like avoiding unnecessary global variables and closing files and connections help manage memory better in Python.

**100. What is the**with**statement in Python and why is it used?**

The with statement in Python is used for **resource management**, such as working with files or network connections. It ensures that resources are properly closed after they are used, even if an error occurs during their use.

Example:

with open("file.txt", "r") as file:  
 data = file.read()

Here, file is automatically closed when the block ends. This is better than manually calling file.close() because it handles exceptions safely.

The with statement works with context managers, which define \_\_enter\_\_() and \_\_exit\_\_() methods. It makes your code cleaner, safer, and easier to maintain.

[PreparePrepare](https://prepinsta.com/interview-preparation/technical-interview-questions/python/" \o "Prepare)

[CoursesCourses](https://prepinsta.com/interview-preparation/technical-interview-questions/python/" \o "Courses)

[Micro CertsMicro Certs](https://prepinsta.com/interview-preparation/technical-interview-questions/python/" \o "Micro Certs)

[Job Assistance programJob Assistance program](https://prepinsta.com/interview-preparation/technical-interview-questions/python/" \o "Job Assistance program)

[OffCampusOffCampus](https://prepinsta.com/interview-preparation/technical-interview-questions/python/" \o "OffCampus)



* [**Login**](https://prepinsta.com/interview-preparation/technical-interview-questions/python/)
* [**Get Prime**](https://prepinstaprime.com/?utm_source=prepinsta&utm_medium=top+menu)

**Technical Interview Preparation**

* [Technical Interview Questions](https://prepinsta.com/interview-preparation/technical-interview-questions/)
* [HTML Interview Questions](https://prepinsta.com/interview-preparation/technical-interview-questions/html/)
* [CSS Interview Questions](https://prepinsta.com/interview-preparation/technical-interview-questions/css/)
* [JavaScript Interview Questions](https://prepinsta.com/interview-preparation/technical-interview-questions/javascript/)
* [React JS Interview Questions](https://prepinsta.com/interview-preparation/technical-interview-questions/react-js/)
* [Node.js interview questions](https://prepinsta.com/interview-preparation/technical-interview-questions/node-js/)
* [MERN Stack Interview Questions](https://prepinsta.com/interview-preparation/technical-interview-questions/mern-stack/)
* [Data Analytics Interview Questions](https://prepinsta.com/data-analyst-interview-questions-and-answers/)
* [C++ Technical Interview Questions](https://prepinsta.com/interview-preparation/technical-interview-questions/c-plus-plus/)
* [Python Interview Questions](https://prepinsta.com/interview-preparation/technical-interview-questions/python/)
* [Java Interview Question](https://prepinsta.com/interview-preparation/technical-interview-questions/java/)
* [Most Asked Coding Question](https://prepinsta.com/interview-preparation/technical-interview-questions/most-asked-coding-questions-in-placements/)
* [DSA interview questions](https://prepinsta.com/interview-preparation/technical-interview-questions/dsa-interview-questions-and-answers-for-freshers/)
* [Computer Network Interview Question](https://prepinsta.com/interview-preparation/technical-interview-questions/computer-network/)
* [Cloud Computing Interview Question](https://prepinsta.com/interview-preparation/technical-interview-questions/cloud-computing/)
* [Other Related Links](https://prepinsta.com/interview-preparation/technical-interview-questions/python/#1613778)

 Apply For Jobs

Get Hiring Updates right in your inbox from PrepInsta

[Get Hiring Updates](javascript:void(0))

# Python Interview Questions

## Top Python Technical Interview Questions with Answers

Here, Python Technical Interview Questions page will help you to get the Python-based Theoretical and Coding Questions that are frequently asked in Technical Interviews if you have opted for Python as your preferred language.

Go through this page to get all the sample questions for preparing for Python Technical Interview.

## Introduction To Python

**Python** is a high-level, interpreted programming language known for its simplicity and readability. Created by **Guido van Rossum** in the late 1980s, Python  supports code readability and a clear syntax that allows programmers to express concepts in fewer lines of code as compared to other languages.

##### Here are some key features of Python:

1. **Easy to Learn and Use:** Python has a clean, simple, and small syntax that makes it easy for beginners to understand and write code.
2. **Versatile and Cross-Platform:** Python can be used for a wide range of applications, including web development, data analysis, scientific computing, artificial intelligence, machine learning, automation, and more.
3. **Interpreted and Interactive:** Python is an interpreted language, which means that the code is executed line by line without the need for explicit compilation.
4. **Large Standard Library:** Python comes with a comprehensive standard library that provides a wide range of modules and functions for common tasks. It offers ready-to-use modules for file I/O, networking, regular expressions, database access, and much more, reducing the need for external dependencies.
5. **Third-Party Ecosystem:** Python has a thriving ecosystem of third-party libraries and frameworks maintained by the community. These libraries extend Python’s capabilities and offer specialized tools for specific domains, such as Django and Flask for web development, NumPy and Pandas for data manipulation, TensorFlow and PyTorch for machine learning, and many others.
6. **Dynamic Typing:** Python uses dynamic typing, which means that you don’t need to explicitly declare variable types. The type of a variable is determined at runtime based on the assigned value. This allows for flexibility and faster development but requires attention to type-related issues.

## Python Interview Questions for Freshers

#### **Question 1: Tell something about Python ?**

**Answer:** Python is a general-purpose, high-level, and interpreted programming language. Python supports objects, modules, exception-handling, threads, and automatic memory management which help in solving real-world problems with less coding.

#### **Question 2: What are the benefits of using Python?**

**Answer:** Python benefits:

1. Readable and simple syntax.
2. Large standard library.
3. Wide variety of third-party libraries and frameworks.
4. Easy integration with other languages.
5. Powerful for data analysis and scientific computing.
6. Ideal for automation and scripting.

#### **Question 3: Name some Libraries of Python Programing language and their application?**

**Answer:**

**NumPy:** It provides operations on multi-dimensional arrays, mathematical functions, and tools for working with large datasets. In simple words it enables us to perform complex mathematical operations.

NumPy is mainly used in data analysis, scientific calculations, and machine learning.

**Pandas:** This library is used for data manipulation and analysis. Pandas offers data structures (such as DataFrames) and functions for cleaning, transforming and exploring structured data.

**Matplotlib:** This library for creating static, animated, and interactive visualizations in Python from the data provided by the user. Matplotlib enables the creation of various plots, charts, and graphs, making it a go-to choice for data visualization tasks.

**TensorFlow:** An open-source library for machine learning and deep learning. TensorFlow provides a flexible ecosystem for building and deploying machine learning models, especially neural networks. It is widely used in research, production-grade applications, and AI development.

**Scikit-learn:** A machine learning library that offers a range of algorithms and tools for data mining, classification, regression, clustering, and dimensionality reduction. Scikit-learn simplifies the implementation of machine learning models and pipelines.

**Beautiful Soup:** A library for web scraping and parsing HTML/XML documents. Beautiful Soup makes it easy to extract data from web pages, navigate the HTML/XML structure, and scrape information for various applications.

**PyTorch:** PyTorch is designed to take advantage of GPUs for fast computation in deep learning tasks. It provides a tensor library that enables efficient data storage and manipulation on GPUs. PyTorch is widely used in developing and training neural networks.

#### **Question 4: What is the difference between Compiled Languages and Interpreted Languages?**

**Answer:**

In the case of **compiled languages**, the source code is translated entirely into machine code by a compiler before execution. The resulting compiled program can be directly executed by the computer’s processor, providing fast and efficient performance.  
**Ex –** C, C++, and Rust.

And in the case of **Interpreted language,** an interpreter reads and executes the source code line by line at runtime. The interpreter translates each line of code into machine code while the program is running.

**Ex-** Python, JavaScript, and Ruby.

#### **Question 5: What is a module in Python with example?**

**Answer:** Module is a file containing Python definitions, functions, classes, and variables. It acts as a container for related code and can be imported into other Python programs using the import statement. Modules help in modularizing code and promote code reuse.

“**Import**” is used to get all functionalities of any particular module.

Ex-  
Python provides a wide range of built-in modules that offer various functionalities like:

1. **math**: Provides mathematical functions and constants.
2. **random**: Offers functions for generating pseudo-random numbers.
3. **datetime**: Enables manipulation of dates, times, and time intervals.
4. **os**: Allows interaction with the operating system, providing functions for file operations, directory handling, etc.
5. **sys**: Provides access to system-specific parameters and functions.
6. **json**: Enables encoding and decoding of JSON data.
7. **re**: Provides regular expression matching operations.
8. **csv**: Offers functionality for reading and writing CSV files.
9. **urllib**: Allows making HTTP requests and working with URLs.
10. **sqlite3**: Provides a simple and lightweight database interface for SQLite database

**Example:**

Run

import math

value = math.pi

print("Value of Pi = : ",value)

#### **Question 6: In python, arguments are passed by reference or value ?**

**Answer:**

In python, arguments are passed through reference. This means that when a function is called and an argument is passed, a reference to the object is passed in place of the value itself.

#### **Question 7: What is the use of Floor Division (//) in python?**

**Answer:**  Floor division in Python is denoted by the double forward slash operator //. It performs division between two numbers and returns the largest integer less than or equal to the quotient. **Code Ex –**

Result = 10 // 3

print(Result)

**Output:** 3

#### **Question 8: What is the use Range function in python ?**

**Answer:**

**range()** function in Python allows to create a sequence of numbers that can be used for iteration. It can be used to create a range of numbers with a specified start, stop, and step value. **[**i.e. **range( start, stop, steps) ]**

**Code Ex –** 

Run

for i in range(10):

print(i) #Output: 0 1 2 3 4 5 6 7 8 9

for j in range(0, 10, 2):

print(j) #Output: 0 2 4 6 8

#### **Question 9: What is the difference between data structures like list, tuples, dictionaries and sets in python ?**

| **List** | **Tuples** | **Dictionaries** | **Sets** |
| --- | --- | --- | --- |
| Ordered collection of items. | Ordered collection of items. | Collection of key-value pairs. And Unordered (no specific order of elements). | Unordered collection of unique elements. |
| Mutable (can be modified after creation). | Immutable (cannot be modified after creation). | Mutable | Mutable |
| Allows duplicate elements. | Allows duplicate elements. | Keys are unique, and values can be duplicated. | Doesn’t allow duplicate elements. |
| Elements are enclosed in square brackets [ ]. | Elements are enclosed in parentheses ( ) or can be without any enclosure. | Elements are enclosed in curly braces { }, with each item consisting of a key and its value separated by a colon **:** | Elements are enclosed in curly braces { }. |
| Ex- fruits = [“apple”, “banana”, “orange”, “grape”] | Ex- numbers = (“Twenty”, 20, “Thirty”) | Ex- assassin = {“name”: “John Wick”, “age”: 28, “major”: “Self Defence & Martial Art”} | Ex- numbers = {1, 2, 3, 4, 5} |

#### **Question 10: What is a Negative Index in Python? Give an Example.**

**Answer:**

Negative indexing allows us to access elements from the end of a sequence by using negative numbers as their indices.  
Last element has an index of -1  
Second last has an index of -2  
Third last has an index of -3 and so on.

#### **Question 11: What are \*args and \*\*kwargs in python?**

**Answer:**

**\*args (pronounced “star args”)** allows you to pass multiple number of arguments to a function without specifying their names in advance. It collects all the arguments into a tuple, which you can access within the function.

Run

def prac\_function(\*args):

for ARG in args:

print(ARG)

prac\_function(1,2,3,4,5,6) #Output: 1 2 3 4 5 6

**\*\*kwargs (pronounced “star star kwargs”)** allows you to pass multiple number of keyword arguments to a function. Keyword arguments are like labeled items you put into the bag. \*\*kwargs collects these labeled items into a dictionary, where the labels (keywords) become the keys and the values are the corresponding items.

Run

def prac\_function(\*\*kwargs):

for key, value in kwargs.items():

print(key, value)

prac\_function(Name='Viraj', Age=22, City='Lucknow')

#### **Question 12: What is Slicing ?**

**Answer:** Slicing is a technique used to extract a part or a subsequence of a sequence, such as a string, list, or tuple. It allows you to retrieve a specific range of elements or specific element from the sequence based on indices.

Basic syntax for slicing = “**sequence\_name[start:stop:step]**”  
**Code Ex-**

Run

my\_list = [1, 2, 3, 4, 5, 6, 7, 8, 9, 10]

sliced = my\_list[2:6]

print(sliced)                     #Output: [3, 4, 5, 6]

# Slicing with negative indices to get the last three elements

sliced = my\_list[-3:]

print(sliced)                     #Output: [8, 9, 10]

#### **Question 13: What is the Lambda expressions in Python ? Explain with an example.**

**Answer:**

Lambda expressions are a way to quickly create what are known as anonymous functions, basically just, one-time-use functions that you don’t even really name. You just use them one time and then never reference them again.  
**Code Ex-**

Run

add = lambda x, y: x + y

result = add(20,15)

print(result) #Output: 35

#### **Question 14: What is PEP 8 ?**

**Answer:**

PEP 8 is a guide that helps in writing clean, consistent, and maintainable Python code that is easy to read and understand. It stands for Python Enhancement Proposal, it specifically provides guidelines and recommendations on how to format and structure Python code to enhance readability and reliability of the code.  
Some key points of PEP 8 include:

1. Indentation: Use 4 spaces for indentation to improve code readability.
2. Line Length: Limit lines to a maximum of 79 characters to ensure readability, although it can be extended up to 120 characters in certain cases.
3. Naming Conventions
4. Function and Variable Names
5. Imports: Import modules on separate lines and follow a specific ordering convention (standard library modules, third-party modules, local modules).
6. It also provides numerous other guidelines covering various aspects of coding style, including whitespace, blank lines, operator spacing, and more.

#### **Question 15: What are .py and .pyc files ?**

**Answer:**

**.py files:**

1. These files contain Python source code written in plain text.
2. They are human-readable and editable using a text editor or integrated development environment (IDE).
3. Python source code is typically saved with the .py extension.

**.pyc files:**

1. These files are compiled bytecode files generated by the Python interpreter.
2. They are not human-readable and cannot be edited directly.
3. The Python interpreter compiles the .py source code into bytecode and saves it as a .pyc file for efficient execution in subsequent runs.
4. The .pyc files speed up the loading and execution of Python programs since the interpreter can directly execute the bytecode without the need for recompilation.

#### **Question 16: What are the types of literals in Python?**

**Answer:**

1. **Numeric Literal**: Numeric literals can be floating-point values, integers, or complex numbers.
2. **Character Literal:** A character literal consists of a single character enclosed in double quotes.
3. **Boolean Literal:** The boolean literals are True or False.
4. **Literal Collections:** There are four types of literal collections, including list literals, tuple literals, set literals, and dictionary literals.
5. **String Literal:** String literal is created by assigning text to a variable using single or double quotes. Multiline literals can be formed by enclosing text within triple quotes.

#### **Question 17: What are some built in data types in python ?**

**Answer:**

**Numeric Types:**

1. **int:** Represents integer values (e.g., 1, 5, -10).
2. **float:** Represents floating-point numbers with decimal values (e.g., 3.14, -0.5).

**Sequence Types:**

1. **str:** Represents a sequence of characters, also known as strings (e.g., “hello”, ‘world’).
2. **list:**Represents an ordered collection of items (e.g., [1, 2, 3], [‘apple’, ‘banana’]).
3. **tuple:**Represents an ordered, immutable collection of items (e.g., (1, 2, 3), (‘a’, ‘b’, ‘c’)).

**Mapping Type:**

dict: Represents a collection of key-value pairs (e.g., {‘name’: ‘John’, ‘age’: 25}).

**Set Types:**

set: Represents an unordered collection of unique elements (e.g., {1, 2, 3}).

**Boolean Type:**

bool: Represents the truth values True and False.

**None Type:**

None: Represents the absence of a value or the null value.

#### **Question 18: What is pickling and Unpickling in Python ?**

**Answer:**

**Pickling:**

1. Pickling is the process of converting a Python object hierarchy into a byte stream.
2. It allows you to save the state of an object or a collection of objects as a file or transfer it over a network.
3. The resulting byte stream can be stored persistently or transmitted between different systems.
4. Pickling is commonly used for tasks like caching, serialization, and data persistence.

**Unpickling:**

1. Unpickling is the process of reconstructing a Python object hierarchy from a byte stream.
2. It is the reverse operation of pickling and allows you to restore the state of the objects.
3. By unpickling, you can retrieve the original object or data structure that was pickled.
4. Unpickling is essential when you want to retrieve and utilize the saved data or objects.

#### **Question 19: How memory is managed in python programming language ?**

**Answer:**

In Python, memory management operates in the following manner:

1. Memory management in Python is handled by a private heap space. All Python objects and data structures are stored within this private heap, which remains inaccessible to programmers. The responsibility of managing this private heap lies with the Python interpreter.
2. The allocation of heap space for Python objects is handled by Python’s memory manager. Although programmers do not have direct access to this process, Python’s core API provides certain tools that can be utilized.
3. Python incorporates an internal garbage collector that is responsible for reclaiming unused memory. This ensures that memory becomes available within the heap space for future utilization.

#### **Question 20: What are Deep Copy and Shallow Copy ?**

**Answer:**

**Shallow Copy:**

* Shallow copy creates a new object and then copies the references of the original object’s elements into the new object.
* The new object and the original object share the same elements (references), so changes made to one object may affect the other.
* In a shallow copy, the top-level elements are copied, but the nested objects within them are not duplicated.
* Shallow copies are created using methods like slicing, the `copy()` method, or the `copy` module.

**Deep Copy:**

* Deep copy creates a new object and recursively copies all the elements and nested objects of the original object.
* The new object is completely independent of the original object, and any changes made to one object do not affect the other.
* Deep copies ensure that all levels of the object hierarchy are duplicated, including nested objects and their references.
* Deep copies are created using the `copy.deepcopy()` function from the `copy` module.

## Python OOPS Interview Questions

#### **Question 21: List some common Python interpreters.**

**Answer:**

1. **CPython:** The default and most widely used Python interpreter. It is written in C and serves as the reference implementation for the Python language.
2. **Jython:** An implementation of Python that runs on the Java Virtual Machine (JVM). It allows seamless integration with Java code and libraries.
3. **IronPython:** An implementation of Python targeting the .NET framework. It provides integration with the .NET ecosystem and allows Python code to interact with .NET languages and libraries.
4. **PyPy:** A fast and highly optimized implementation of Python. It utilizes a Just-in-Time (JIT) compiler to improve execution speed.
5. **Stackless Python:** A variant of CPython that provides support for micro threads, allowing lightweight concurrency without the need for traditional operating system threads.
6. **MicroPython:** A lightweight implementation of Python specifically designed for microcontrollers and embedded systems. It provides a reduced subset of the Python language to optimize for limited resources.

#### **Question 22: Write a program to produce the Fibonacci series in Python.**

**Answer:**

Run

def fibonacci(n):

series = [ ]

a, b = 0, 1

while len(series) < n:

series.append(a)

a, b = b, a + b

return series

n = int(input("Enter number of terms in the Fibonacci series: "))

fibonacci\_series = fibonacci(n)

print("Fibonacci series:", fibonacci\_series)

#### **Question 23: Differentiate between Pyramid, Django, and Flask.**

**Answer:**

| **Django** | **Pyramid** | **Flask** |
| --- | --- | --- |
| Django is a high-level, full-featured web framework that follows the batteries included methodology. | Pyramid is a flexible, minimalist web framework that follows the “pay only for what you need”. | Flask is a lightweight web framework that has simplicity and extensibility. |
| It provides a huge set of built-in features, such as an ORM (Object-Relational Mapping), authentication, routing, templating, and admin interface. | It provides a core set of tools for building web applications, allowing developers to choose and add additional components as needed. | It provides the resources for building web applications but keeps the core framework minimalistic. |
| Django is suitable for building complex, database-driven web applications with less code and rapid development. | Pyramid is highly customizable and adaptable, making it suitable for a wide range of applications, from simple to complex. | Flask allows developers to choose extensions based on their specific requirements, making it highly modular. |
| It enforces a specific project structure and follows the Model-View-Controller (MVC) architectural pattern. | It follows a “traversal” routing system and supports various templating engines. | It follows a route-decorator approach and supports various templating engines. |

#### **Question 24: What is Dynamically typed and Statically typed languages ?**

**Answer:**

**Dynamically Typed**: In dynamically typed languages, variable types are determined at runtime, meaning that type checking occurs during program execution. Variables can be assigned values of different types at different points in the program.

**Code Ex-**

Run

# Dynamically typed language example (Python)

x = 10 # x is an integer

print(x)

x = "Ten" # x is now a string

print(x)

x = [10, 20, 30] # x is now a list

print(x)

**Statically Typed:** In statically typed languages, variable types are checked and resolved during compile-time, before the program is executed. Variables must be explicitly declared with their types, and type checking is performed at compile-time.

**Code Ex-**

Run

#include<bits/stdc++.h>

using namespace std;

int main ()

{

int x = 5; // x is an integer

cout << x << endl;

x = "Hello";

cout << x << endl;

return 0;

}

// Error: invalid conversion from 'const char\*' to 'int'

#### **Question 25: What is OOPs Concepts ? Explain all of them with examples.**

**Answer:**

**1. Encapsulation:**

Encapsulation is the process of hiding the internal implementation details of an object and exposing only the necessary information. It helps in achieving data security and code maintainability.

**Ex-**

Run

class Car:

def \_\_init\_\_(self, brand, model):

self.brand = brand

self.model = model

def start\_engine(self):

print("Engine started.")

my\_car = Car("Ford", "Mustang")

print(my\_car.brand)

Output: Ford

my\_car.start\_engine() #Output: Engine started

**2. Inheritance:**

Inheritance is a mechanism where a class inherits properties and methods from a parent class. It allows code reuse and the creation of specialized classes based on more general classes.

**Ex-**

Run

class Animal:

def \_\_init\_\_(self, name):

self.name = name

def speak(self):

print("Animal speaks.")

class Dog(Animal):

def speak(self):

print("Woof!")

my\_dog = Dog("Buddy")

print(my\_dog.name)

Output: Buddy

my\_dog.speak() #Output: Woof!

**3. Polymorphism:**

Polymorphism allows objects of different classes to be treated as objects of a common base class. It provides flexibility and extensibility in handling objects of different types.

**Ex-**

Run

class Shape:

def area(self):

pass

class Rectangle(Shape):

def \_\_init\_\_(self, width, height):

self.width = width

self.height = height

def area(self):

return self.width \* self.height

class Circle(Shape):

def \_\_init\_\_(self, radius):

self.radius = radius

def area(self):

return 3.14 \* self.radius \*\* 2

shapes = [Rectangle(4, 5), Circle(3)]

for shape in shapes:

print(shape.area())

**4. Abstraction:**

Abstraction involves representing essential features of an object while hiding the unnecessary details. It allows programmers to work with high-level concepts without worrying about implementation specifics.

**Ex-**

Run

from abc import ABC, abstractmethod

class Shape(ABC):

def area(self):

pass

class Rectangle(Shape):

def \_\_init\_\_(self, width, height):

self.width = width

self.height = height

def area(self):

return self.width \* self.height

class Circle(Shape):

def \_\_init\_\_(self, radius):

self.radius = radius

def area(self):

return 3.14 \* self.radius \*\* 2

my\_rectangle = Rectangle(4, 5)

my\_circle = Circle(3)

print(my\_rectangle.area()) #Output: 20

print(my\_circle.area()) #Output: 28.26

#### **Question 26: Write a python program to checks if a sequence is a Palindrome or not.**

Run

sequence = input ("Enter a sequence = ")

reversed\_sequence = sequence[::-1]

if sequence == reverse\_sequence:

print("Palindrome Sequence.")

else:

print("Not a Palindrome.")

#### **Question 27: What is PIP ? Name some common PIP Command.**

PIP (**Python Package Installer**) is the default package manager for Python. It is a command-line tool that allows you to easily install, manage, and uninstall Python packages from the Python Package Index (PyPI) or other package repositories.

Some Common PIP Command are:

1. **pip install package\_name:** Installs a Python package.
2. **pip uninstall package\_name:** Uninstalls a Python package.
3. **pip list:** Lists all installed packages.
4. **pip freeze > requirements.txt:** Exports a list of installed packages and their versions to a requirements.txt file.
5. **pip install -r requirements.txt:** Installs packages listed in a requirements.txt file.

#### **Question 28: Elaborate the concept of Dictionaries with the help of Example.**

Run

# Creating a dictionary

student = {

"name": "John Wick",

"age": 28,

"major": "Martial Arts and Self Defence.",

"university": "Shaolin University"

}

# Accessing dictionary values

print("Name:", student["name"])

print("Age:", student["age"])

print("Major:", student["major"])

print("University:", student["university"])

#### **Question 29: What is docstring in Python?**

**Answer:**

Docstring is a string literal used to document modules, classes, functions, and methods. It serves as a documentation tool to describe the purpose, behavior, parameters, return values, and other important details of the code.

A docstring is enclosed in triple quotes (single or double) and is typically placed as the first line after the definition of a module, class, function, or method. It can span multiple lines and supports both single-line and multi-line docstrings.

Run

def hello(name):

"""

This function says hello to the person with the given name.

Parameters: name (str): The name of the person to be greeted.

Returns: str: A greeting message.

"""

return "Hello, " + name + " !!!!!"

print(hello("Isabelle"))

#### **Question 30: What are python namespaces ? What are their applications ?**

**Answer:**

Namespace is a mapping from names (identifiers) to objects. It serves as a system to organize and provide a unique context for names in a Python program. Namespaces help prevent naming conflicts and provide a way to access objects in a structured manner.

Python uses namespaces to determine the scope of names. When you use a variable, function, or any other object, Python looks for that name within the available namespaces to resolve it.

**Here are a few types of namespaces in Python:**

1. **Built-in Namespace:** It contains the names of built-in functions, exceptions, and objects that are available by default in Python. Examples include print(), len(), str, etc.
2. **Global Namespace:** It refers to the names defined at the top level of a module or declared as global within a function. These names are accessible throughout the module or function.
3. **Local Namespace:** It represents the names defined within a function. These names are accessible only within the function’s scope.
4. **Class Namespace:** It contains the names defined within a class. These names are accessible within the class and can be accessed using the class name.

## Python Libraries Interview Questions

#### **Question 31: Write a program in Python to check if a number is prime.**

Run

a = int(input("Enter a number = "))

if a == 1:

print("Not a prime number.")

else:

for x in range(2, a):

if (a % x) == 0:

print("Not a prime number.")

break

else:

print("Prime Number.")

#### **Question 32 : Python program to print following ‘\*’ pattern: \* \*\* \*\*\* \*\*\*\* \*\*\*\*\***

Run

def pattern\_print(rows):

for i in range(1, rows + 1):

print("\*" \* i)

# Take input from the user

rows = int(input("Enter the number of rows: "))

# Call the function to print the pattern

pattern\_print(rows)

#### **Question 33 : Python program to print following ‘\*’ pattern: \*\*\*\* \*\*\* \*\* \***

Run

def pattern(rows):

for i in range(rows, 0, -1):

print("\*" \* i)

# Take input from the user

num\_rows = int(input("Enter the number of rows: "))

# Call the function to print the pattern

pattern(num\_rows)

#### **Question 34 : Write a python program to perform bubble sort for given array.**

Run

def bubble\_sort(arr): # Bubble Sort Function

n = len(arr)

for i in range(n - 1):

for j in range(n - i - 1):

if arr[j] > arr[j + 1]:

arr[j], arr[j + 1] = arr[j + 1], arr[j]

numbers = input("Enter the List of numbers: ").split()

numbers = [int(num) for num in numbers]

bubble\_sort(numbers) # Calling bubble sort function

print("Sorted list:", numbers) # Sorted List

#### **Question 35 : Write a python program to perform merge sort for given array.**

Run

def merge\_sort(arr): # Merge Sort Function

if len(arr) > 1:

mid = len(arr) // 2

left\_part = arr[:mid]

right\_part = arr[mid:]

merge\_sort(left\_part)

merge\_sort(right\_part)

i = j = k = 0

while i < len(left\_part) and j < len(right\_part):

if left\_part[i] < right\_part[j]:

arr[k] = left\_part[i]

i += 1

else:

arr[k] = right\_part[j]

j += 1

k += 1

while i < len(left\_part):

arr[k] = left\_part[i]

i += 1

k += 1

while j < len(right\_part):

arr[k] = right\_part[j]

j += 1

k += 1

numbers = input("Enter the List of numbers= ").split()

numbers = [int(num) for num in numbers]

merge\_sort(numbers)

print("Sorted list =", numbers)

#### **Question 36 : Write a python program to perform merge sort for given array.**

Run

def merge\_sort(arr): # Merge Sort Function

if len(arr) > 1:

mid = len(arr) // 2

left\_part = arr[:mid]

right\_part = arr[mid:]

merge\_sort(left\_part)

merge\_sort(right\_part)

i = j = k = 0

while i < len(left\_part) and j < len(right\_part):

if left\_part[i] < right\_part[j]:

arr[k] = left\_part[i]

i += 1

else:

arr[k] = right\_part[j]

j += 1

k += 1

while i < len(left\_part):

arr[k] = left\_part[i]

i += 1

k += 1

while j < len(right\_part):

arr[k] = right\_part[j]

j += 1

k += 1

numbers = input("Enter the List of numbers= ").split()

numbers = [int(num) for num in numbers]

merge\_sort(numbers)

print("Sorted list =", numbers)

#### **Question 37 : Write a python program to print Star (\*) Triangle:**

#### \*    \*\*   \*\*\*  \*\*\*\* \*\*\*\*\*

Run

def star\_triangle(rows):

for i in range(1, rows + 1):

print(" " \* (rows - i), end="")

print("\*" \* (2\*i - 1))

num\_rows = int(input("Enter the number of ROWS: "))

star\_triangle(num\_rows)

#### **Question 38: What is PythonPath ?**

PYTHONPATH is an environment variable in Python that tells the interpreter where to look for Python modules and packages.  
It is a list of directory paths separated by colons (on Unix-based systems) or semicolons (on Windows). When you import a module or package, Python searches for it in the directories listed in PYTHONPATH. It allows you to specify additional directories outside the default ones where your Python code resides, enabling easy access to custom modules or packages.

#### **Question 39: What are Global and Local Variable ?**

**Global variables** are declared outside any function or block and can be accessed from anywhere within the program. They have a global scope, meaning they are visible to all functions and blocks within the program.

**Ex:**

x = 10

def print\_global():

print(x)

print\_global()

# Output: 10

**Local variables** are declared within a function or block and can only be accessed within that specific function or block. They have a local scope, meaning they are visible and accessible only within their respective function or block.

**Ex:**

def print\_local():

y = 20

print(y)

print\_local()

# Output: 20

#### **Question 40: Make a Binary search program in Python**

Run

def binary\_search(arr, target):

low = 0

high = len(arr) - 1

while low <= high:

mid = (low + high) // 2

mid\_value = arr[mid]

if mid\_value == target:

return mid

elif mid\_value < target:

low = mid + 1

else:

high = mid - 1

return -1

elements = input("Enter a list of numbers = ").split()

target = int(input("Enter the number to be searched = "))

arr = [int(element) for element in elements]

arr.sort()

print("Sorted List =",arr)

index = binary\_search(arr, target)

if index != -1:

print(f"{target} is at index {index}")

else:

print(f"{target} is not present in the list")

## Python Interview Questions for Experienced

#### **Question 41: Make a linear search program in python.**

Run

def linear\_search(arr, target):

for i in range(len(arr)):

if arr[i] == target:

return i

return -1

elements = input("Enter the list of numbers = ").split()

target = int(input("Enter the number to be searched = "))

arr = [int(element) for element in elements]

arr = sorted(arr)

print("Sorted Array = ",arr)

index = linear\_search(arr, target)

if index != -1:

print(f"{target} is at index {index}")

else:

print(f"{target} is not present in list")

#### **Question 42: What is map() function ?**

**map() function**in Python is a built-in function that allows you to apply a specified function to every item in one or more iterable objects, such as lists, tuples, or strings. It takes in two or more arguments: the function to be applied and the iterable(s) on which the function should operate.

#### **Question 43: How does continue, break, and pass work?**

**Continue, break, and pass** are three control flow statements in Python that allow you to change the flow of your program under certain conditions.

**Continue statement:**When continue is used inside a loop, it tells Python to skip the current iteration and move on to the next one. Any code after the continue statement within the loop for that iteration will be ignored, and the loop will continue with the next iteration.

**Break statement:** When break is used inside a loop, it tells Python to immediately exit the loop, regardless of any remaining iterations. Any code after the break statement within the loop will be skipped, and the program will continue executing from the next statement after the loop.

**Pass statement:** pass is used as a placeholder when you need to have a statement for syntactic reasons, but you don’t want to do anything in that part of the code. It doesn’t do anything and is mainly used to avoid syntax errors when you’re still working on implementing certain parts of your code.

#### **Question 44: What are Classes and Objects ?**

**Class** is a blueprint or a template that defines the structure and behavior of objects. It is like a blueprint for creating multiple instances of similar objects with shared characteristics and functionalities.

It contains data (in the form of attributes or properties) and behaviors (in the form of methods or functions) that define the objects’ characteristics and actions. It provides a way to organize related data and functions into a single unit.

**An object,** on the other hand, is an instance of a class. It represents a specific entity or item created based on the class definition. Objects have their own unique state and can interact with other objects or perform operations defined within the class.

#### **Question 45: Write a python code to illustrate classes and object.**

Run

class assassin:

def \_\_init\_\_(self, name, age):

self.name = name

self.age = age

a1 = assassin("John Wick", 28)

print(a1.name)

print(a1.age)

#### **Question 46: How to add values or remove values to a python array?**

**Adding values:**

1. **append():** Adds an element to the end of the array.
2. **extend():** Appends multiple elements from an iterable to the end of the array.
3. **insert():** Inserts an element at a specific index within the array.

**Removing values:**

1. **remove():** Removes the first occurrence of a specific element from the array.
2. **pop():** Removes and returns an element at a specified index from the array.
3. **del** statement: Deletes an element or a slice of elements from the array.

#### **Question 47: Which python framework is best flask or django ?**

**Django** provides a wide range of built-in features and components, reducing the need for external libraries or packages.

* **Rapid development:** Django’s high-level abstractions and conventions simplify the development process, allowing you to build applications quickly.
* **Scalability:** Django has proven to be scalable and has been used successfully in handling high-traffic websites and complex applications.
* **Larger Community:** Django has a large and active community, offering extensive documentation, tutorials, and reusable packages.
* **Better Security Module:** Django incorporates security features by default, including protection against common vulnerabilities.

**Flask,**on the other hand, is a lightweight micro-framework that offers more flexibility and customization options.

It is suitable for smaller projects, APIs, or situations where simplicity and control are desired. Flask allows developers to have more control over the architecture and components they use but lacks some of the built-in functionalities provided by Django.

Conclusion is that Django’s comprehensive feature set, strong community support, and emphasis on convention-over-configuration make it a preferred choice for many web development projects.

#### **Question 48: What is multi threading and how it can be achieved ?**

**Multithreading** is a programming technique that allows multiple threads of execution to run concurrently within a single process. A thread is a lightweight unit of execution within a program that can perform tasks independently.

Multithreading can be achieved in various programming languages, including Python, by utilizing the operating system’s threading capabilities or using libraries that provide threading functionality.

Here are the basic steps to achieve multithreading in Python:

1. **Import the threading module:** In Python, multithreading is facilitated by the built-in threading module. Import the module to gain access to its classes and functions.
2. **Define a task or function:** Create a function or task that you want each thread to execute concurrently. This function represents the work that will be performed by each thread.
3. **Create thread objects:** Instantiate thread objects from the Thread class provided by the threading module. Specify the target function or task to be executed by each thread. You can also pass any required arguments to the target function.
4. **Start the threads:** Call the start() method on each thread object to start the execution of the threads. Each thread will begin running concurrently.

**Wait for thread completion:** If needed, use the join() method on each thread to wait for its completion. This ensures that the main program doesn’t proceed until all threads have finished their execution.

#### **Question 49: Write the python code to perform Write and Read operation in Python ?**

Run

# Open the file in write

file = open("file.txt", "w")

file.write("Hello Prepsters.\n")

file.write("We will assist you in getting IT Job.\n")

file.write("Here, You’ll learn various skills.")

file.close()

# Opening the file in read mode.

file = open("file.txt", "r")

contents = file.read()

print(contents)

# file.close() # Closing the file

#### **Question 50: Write the python code to append operation in existing file ?**

Run

#Appending operation

file = open("file.txt", "a")

file.write("\nHello we are PrepInsta.")

file.write("\nPrepInsta and PrepInsta Prime.")

contents = file.read()

print(contents)

file.close()