**PYTHON INTERVIEW QUESTIONS**

1. What are Python’s key features? Why is it called an interpreted language?

Python is a high-level, object-oriented programming language known for its **simplicity and readability**.  
Key features include:

* **Interpreted**: Executes code line-by-line using an interpreter, making debugging easier.
* **Dynamically typed**: No need to declare variable types.
* **Extensive libraries**: Rich standard and third-party libraries.
* **Cross-platform**: Runs on Windows, Linux, macOS.
* **Supports multiple paradigms**: Procedural, object-oriented, and functional.
* **Open-source**: Freely available and community-driven.

**2. Difference between Python 2 and Python 3**

* Python 2 (legacy, discontinued in 2020) vs Python 3 (current, supported).
* Major differences:
  + **Print**: print "Hello" (Py2) vs print("Hello") (Py3).
  + **Division**: 5/2 = 2 (Py2 integer division) vs 5/2 = 2.5 (Py3 true division).
  + **Unicode**: Strings are ASCII by default in Py2, Unicode by default in Py3.
  + **Community Support**: Only Python 3 is maintained.

**3.Explain indentation in Python. What happens if indentation is incorrect?**

* Python uses **indentation (spaces/tabs)** to define code blocks (instead of {} like Java/C).
* Example:

if True:

print("Yes")

* Incorrect indentation → **IndentationError** or **unexpected behavior**.

**4. What are Python keywords? Can you use them as variable names?**

* **Keywords** are reserved words that have predefined meaning (e.g., if, for, class, def, return).
* No, they **cannot** be used as variable names. Example:

class = 10 # ❌ SyntaxError

**5. Difference between list, tuple, set, and dictionary**

* List → Ordered, mutable, allows duplicates.

l = [1, 2, 2, 3]

* Tuple → Ordered, immutable, allows duplicates.

t = (1, 2, 2, 3)

* Set → Unordered, unique elements only.

s = {1, 2, 3}

* Dictionary → Key-value pairs, keys unique.

d = {"a": 1, "b": 2}

**6. Difference between mutable and immutable data types**

* Mutable: Can be changed after creation.  
  Examples: list, dict, set.
* Immutable: Cannot be changed after creation.  
  Examples: tuple, string, int.

**7. Explain == vs is operator in Python**

* == → Compares values.
* is → Compares memory reference (identity).

a = [1, 2]

b = [1, 2]

print(a == b) # True (same values)

print(a is b) # False (different objects in memory)

**8. Difference between append() vs extend() in lists**

* append(x) → Adds one element to the list.
* extend(iterable) → Adds all elements of an iterable.

l = [1, 2]

l.append([3, 4]) # [1, 2, [3, 4]]

l = [1, 2]

l.extend([3, 4]) # [1, 2, 3, 4]

**9. Shallow copy vs deep copy**

* Shallow copy → Creates a new object but references inner objects.
* Deep copy → Creates new object + recursively copies inner objects.
* import copy

a = [[1, 2], [3, 4]]

shallow = copy.copy(a)

deep = copy.deepcopy(a)

a[0][0] = 99

print(shallow) # [[99, 2], [3, 4]]

print(deep) # [[1, 2], [3, 4]]

**10. How does Python handle memory management (Garbage collection)?**

* Python uses automatic memory management:
  + Reference counting: Keeps track of references to objects.
  + Garbage Collector (GC): Removes cyclic references (via gc module).
* Unreferenced objects are collected automatically, freeing memory.

**11. Functions in Python; Difference between \*args and \*\*kwargs**

A function in Python is a reusable block of code that performs a specific task. It is defined using the def keyword and can accept parameters and return values.

\*args: Accepts a variable number of **positional arguments** as a tuple.

\*\*kwargs: Accepts a variable number of **keyword arguments** as a dictionary.

**12. Explain decorators in Python with an example**

A **decorator** is a function that modifies the behavior of another function without changing its code. It’s commonly used for logging, access control, and timing.

**13.What are generators and yield?**

A **generator** is a special type of function that returns an iterator and produces values one at a time using the yield keyword. It supports **lazy evaluation**, which is memory-efficient.

**14.Difference between iterable, iterator, and generator**

- Iterable : an object that can be looped over (eg., list, tuple). Has \_\_iter\_\_() method.

- Iterator: an object with \_\_next\_\_() method that returns items one by one.

- Generator: a type of iterator created using Yield. Automatically implements \_\_iter\_\_() and \_\_next\_\_()

**15.Explain list comprehension with an example**

**List comprehension** is a concise way to create lists using a single line of code.

squares = [x\*\*2 for x in range(5)] print(squares) # Output: [0, 1, 4, 9, 16]

**16.Difference between classmethod, staticmethod, and instance method**

**Instance Method**: Operates on object (self).

**Class Method**: Operates on class (cls). Uses @classmethod.

**Static Method**: No access to class or object. Uses @staticmethod.

class Example: def instance\_method(self): pass @classmethod def class\_method(cls): pass @staticmethod def static\_method(): pass

**17.Python’s OOP concepts: inheritance, polymorphism, encapsulation**

* **Inheritance**: Enables a class to inherit attributes and methods from another class.
* **Polymorphism**: Allows different classes to define methods with the same name but different behavior.
* **Encapsulation**: Restricts access to internal data using private variables (\_var, \_\_var).

**18. What are magic/dunder methods in Python? Examples**

**Magic methods** (or **dunder methods**) are special methods with double underscores used to customize class behavior.

Examples:

* \_\_init\_\_: Constructor method
* \_\_str\_\_: String representation
* \_\_len\_\_, \_\_add\_\_, \_\_eq\_\_, etc.

class Person: def \_\_init\_\_(self, name): self.name = name def \_\_str\_\_(self): return f"My name is {self.name}"

**19. Explain Python’s Global Interpreter Lock (GIL)**

The **GIL** is a mutex in CPython that allows only one thread to execute Python bytecode at a time.  
It ensures **thread safety** during memory management using reference counting.  
While it simplifies single-threaded execution, it limits true parallelism in **CPU-bound** multi-threaded programs.  
For **I/O-bound** tasks, the GIL is released during operations, so threading can still be effective.

**20.Difference between deepcopy vs copy.copy() in Python**

Copy.copy() -> Shallow copy: Copies the object but **not nested objects**.

Copy.deepcopy() -> -> Deep copy: Copies the object and **all nested objects recursively**.

import copy a = [[1, 2], [3, 4]] shallow = copy.copy(a) deep = copy.deepcopy(a)

Modifying a[0][0] affects shallow, but not deep.

Let me know if you want to rehearse these answers in mock Q&A format or get feedback on your delivery style—I can simulate that next!