Python Interview Questions

1. What are Python's key features?

- **Interpreted** (no need for compilation)
- **Dynamically typed** (no need to declare variable types)
- **Object-oriented & functional programming**
- **Extensive libraries** (NumPy, Pandas, etc.)
- **Garbage collection** (automatic memory management)

---

2. What is the difference between Python 2 and Python 3?

Feature	Python 2	Python 3
Print statement	print "Hello"	print("Hello")
Integer division	5 / 2 = 2	5 / 2 = 2.5
Unicode	Not default	Default
Iteration	xrange()	range()

---

3. What are Python's built-in data types?

- **Numeric:** int, float, complex
- **Sequence:** list, tuple, range, str
- **Set:** set, frozenset
- **Mapping:** dict
- **Boolean:** bool
- **None:** NoneType

---

4. What is the difference between list, tuple, and set?

Feature	List	Tuple	Set
Mutable	✅ Yes	❌ No	✅ Yes
Ordered	✅ Yes	✅ Yes	❌ No

Feature	List	Tuple	Set
Allows duplicates	✅ Yes	✅ Yes	❌ No
Example	[1,2,3]	(1,2,3)	{1,2,3}

---

5. Explain Python's memory management.

- Uses **garbage collection** (removes unused objects).
- Uses **reference counting** (sys.getrefcount(obj)).
- Large objects go to the **heap**.

---

6. What are Python's control flow statements?

- **Conditional statements:** if, elif, else
- **Loops:** for, while
- **Loop control:** break, continue, pass

****7. What are *args and kwargs?**

- *args: Passes **multiple positional arguments** as a tuple.
- **kwargs: Passes **multiple keyword arguments** as a dictionary.

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


example(1, 2, 3, name="John", age=30)
# Output: (1, 2, 3) {'name': 'John', 'age': 30}
```

---

8. What is the difference between is and ==?

- == checks **value equality**.
- is checks **memory location (identity check)**.

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

```
print(a == c)  # True (same values)
print(a is c)  # False (different objects)
print(a is b)  # True (same object)
```

---

9. What are shallow copy and deep copy?

- **Shallow copy:** Copies object but not nested objects (copy.copy()).
- **Deep copy:** Copies object **and** nested objects (copy.deepcopy()).

```
import copy
list1 = [[1, 2], [3, 4]]
shallow = copy.copy(list1)
deep = copy.deepcopy(list1)

list1[0][0] = 99
print(shallow)  # [[99, 2], [3, 4]] (affected)
print(deep)  # [[1, 2], [3, 4]] (not affected)
```


---

10. What are Python decorators?

A decorator modifies the behavior of a function without changing its code.

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

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

say_hello()
```

💡 **Output:**

Before function execution

Hello!

After function execution

---

11. What is list comprehension?

A compact way to create lists.

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

---

12. What are Python generators?

Generators return **lazy** sequences using yield.

```
def count():
    for i in range(3):
        yield i


gen = count()
print(next(gen))  # 0
print(next(gen))  # 1
```


---

13. What are lambda functions?

Anonymous, one-line functions.

```
add = lambda x, y: x + y
print(add(2, 3))  # 5
```

---

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

- copy() creates a **shallow copy** (nested objects remain linked).
- deepcopy() creates a **deep copy** (independent objects).

---

15. How does Python handle memory allocation?

- Uses **private heap space** for memory management.

- Uses **reference counting** and **garbage collection**.
- Uses **memory pools** (like PyMalloc).

---

16. What are Python's built-in functions?

- len(), type(), id(), input(), print(), sorted(), etc.

---

17. What is the difference between mutable and immutable objects?

- **Mutable**: Can be modified (list, dict, set).
- **Immutable**: Cannot be modified (int, str, tuple).

---

18. How does exception handling work in Python?

Uses try, except, else, and finally.

```
try:
    x = 1 / 0
except ZeroDivisionError:
    print("Cannot divide by zero")
finally:
    print("Execution completed")
```


---

19. What is the difference between split() and join()?

- split() breaks a string into a list.
- join() combines a list into a string.

```
text = "hello world"
words = text.split()  # ['hello', 'world']
print("_".join(words))  # hello_world
```

---

20. What is the difference between classmethod, staticmethod, and instance methods?

- **Instance Method:** Works with object instance (self).
- **Class Method:** Works with class (cls, @classmethod).
- **Static Method:** No self or cls, acts like a normal function.

```
class Demo:
    @staticmethod
    def static():
        print("Static method")

    @classmethod
    def class_method(cls):
        print("Class method")

    def instance_method(self):
        print("Instance method")

Demo.static()
Demo.class_method()
Demo().instance_method()
```


21. What is the difference between isinstance() and type()?

- isinstance(obj, class): Checks if obj belongs to a class or subclass.
- type(obj) == class: Checks if obj belongs **only** to that class.

```
class A: pass
class B(A): pass

b = B()
print(isinstance(b, A))  # True (B is a subclass of A)
print(type(b) == A)  # False (b is an instance of B, not A)
```

---

22. What is the difference between dir() and vars()?

- dir(obj): Lists all attributes of an object.
- vars(obj): Returns the __dict__ attribute (only instance variables).

```
class Demo:
    def __init__(self):
```

```
        self.x = 10
print(dir(Demo()))  # Shows all attributes
print(vars(Demo()))  # Shows {'x': 10}
```

---

23. What is Python's Global Interpreter Lock (GIL)?

- **GIL** prevents multiple threads from executing Python bytecode **simultaneously**.
- This limits Python's true multithreading capabilities.

💡 **Solution:** Use **multiprocessing** instead of multithreading for CPU-bound tasks.

---

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

- copy.copy(obj): **Shallow copy** (copies top-level structure, keeps nested references).
- copy.deepcopy(obj): **Deep copy** (copies everything, including nested objects).

```
import copy
list1 = [[1, 2], [3, 4]]
shallow = copy.copy(list1)
deep = copy.deepcopy(list1)
```



```
list1[0][0] = 99
print(shallow)  # [[99, 2], [3, 4]]
print(deep)  # [[1, 2], [3, 4]]
```

---

25. What is metaprogramming in Python?

Metaprogramming is writing code that **modifies code** at runtime (e.g., metaclasses).

```
class Meta(type):
    def __new__(cls, name, bases, dct):
        print(f"Creating class: {name}")
        return super().__new__(cls, name, bases, dct)

class Demo(metaclass=Meta):
    pass
```

```
# Output: Creating class: Demo
```

---

26. What is monkey patching in Python?

Monkey patching is modifying a class **at runtime**.

```
class A:
    def hello(self):
        return "Hello"

def new_hello(self):
    return "Hi"

A.hello = new_hello  # Modifying class at runtime
print(A().hello())  # Hi
```


---

27. What are Python's built-in magic methods (__init__, __str__)?

- __init__(): Constructor
- __str__(): Returns a string representation
- __len__(): Defines len(obj)
- __getitem__(): Enables indexing obj[index]

```
class Demo:
    def __init__(self, val):
        self.val = val
    def __str__(self):
        return f"Value: {self.val}"

obj = Demo(10)
print(obj)  # Value: 10
```

---

28. What is the difference between @staticmethod and @classmethod?

- **Static method**: No self, acts like a normal function.
- **Class method**: Uses cls and works on the class level.

```
class Demo:
    @staticmethod
    def static():
        print("Static method")

    @classmethod
    def class_method(cls):
        print("Class method")

Demo.static()
Demo.class_method()
```

---

29. How do you handle file operations in Python?

Using open(), read(), write(), and with.

```
with open("file.txt", "w") as f:
    f.write("Hello, World!")

with open("file.txt", "r") as f:
    print(f.read())  # Hello, World!
```

---

30. What are Python's built-in collections?

- Counter – Counts elements
- deque – Fast double-ended queue
- defaultdict – Dictionary with default values
- OrderedDict – Keeps order of insertion
- namedtuple – Immutable tuple with named fields

```
from collections import Counter
```

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

---

31. What are Python's memory views?

- memoryview() allows **direct access to buffer memory**.
- Used for efficient large data handling.

```
b = bytearray("hello", "utf-8")
m = memoryview(b)
print(m[0])  # 104 (ASCII of 'h')
```

---

32. What is the with statement in Python?

- It ensures proper resource management (auto-closing files).

```
with open("data.txt", "r") as f:
    data = f.read()  # File auto-closes after block ends
```


---

33. What is enumerate() in Python?

- Adds index to an iterable.

```
names = ["Alice", "Bob"]
for index, name in enumerate(names):
    print(index, name)
```

💡 **Output:**

```
0 Alice
1 Bob
```

---

34. What is zip() in Python?

Combines multiple iterables element-wise.

```
a = [1, 2, 3]
b = ["a", "b", "c"]
```

```
print(list(zip(a, b)))  # [(1, 'a'), (2, 'b'), (3, 'c')]
```

---

35. What is map(), filter(), and reduce()?

- map(func, iter): Applies func to all elements.
- filter(func, iter): Filters elements where func returns True.
- reduce(func, iter): Applies cumulative function (import from functools).

```
from functools import reduce

nums = [1, 2, 3, 4]
print(list(map(lambda x: x**2, nums)))  # [1, 4, 9, 16]
print(list(filter(lambda x: x % 2 == 0, nums)))  # [2, 4]
print(reduce(lambda x, y: x + y, nums))  # 10
```

---

36. What is the purpose of globals() and locals()?

- globals(): Returns global variables dictionary.
- locals(): Returns local variables dictionary.

```
x = 10
print(globals())  # Shows all global variables
def test():
    y = 20
    print(locals())  # {'y': 20}
test()
```

---

37. What is Python's assert statement?

- Used for debugging and unit testing.

```
x = 10
assert x > 5  # No error
assert x < 5  # AssertionError
```

---

38. What is a Python frozen set?

- **Immutable version** of a set.

```
fs = frozenset([1, 2, 3])
# fs.add(4)  # Error! Cannot modify
print(fs)  # frozenset({1, 2, 3})
```

39. What is the difference between mutable and immutable objects in Python?

- **Mutable**: Can be changed after creation (list, dict, set).
- **Immutable**: Cannot be changed after creation (int, float, str, tuple).

```
a = [1, 2, 3]
a[0] = 100  # ✅ Allowed (Mutable)
print(a)  # [100, 2, 3]

b = (1, 2, 3)
# b[0] = 100  # ❌ TypeError (Immutable)
```


---

40. What is a Python decorator?

- A decorator **modifies the behavior of a function** without changing its code.

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

@decorator
def hello():
    print("Hello!")
```

```
hello()
```

💡 **Output:**

```
Before function call
Hello!
After function call
```

---

41. What is a lambda function?

- **Anonymous function** using lambda.

```
add = lambda x, y: x + y
print(add(5, 10))  # 15
```

---

42. What is the difference between *args and **kwargs?

- *args: Variable-length **positional** arguments.
- **kwargs: Variable-length **keyword** arguments.

```
def demo(*args, **kwargs):
    print(args)  # Tuple of positional args
    print(kwargs)  # Dict of keyword args

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


---

43. What is a generator in Python?

- Uses yield instead of return to create **iterators** lazily.

```
def gen():
    yield 1
    yield 2
    yield 3

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

---

44. How does Python handle memory management?

- **Reference counting**
- **Garbage collection** (gc module)

```
import gc
gc.collect()  # Manually trigger garbage collection
```

---

45. What is the difference between del and None in Python?

- del: Deletes a variable from memory.
- None: Assigns None, but object may still exist.

```
a = [1, 2, 3]
del a  # Variable `a` is deleted

b = [4, 5, 6]
b = None  # Object exists, but `b` does not reference it
```


---

46. What is the difference between a shallow copy and a deep copy?

- **Shallow copy (copy.copy())**: Copies references.
- **Deep copy (copy.deepcopy())**: Copies entire structure.

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

lst[0][0] = 99
print(shallow)  # [[99, 2], [3, 4]]
print(deep)  # [[1, 2], [3, 4]]
```

---

47. How does Python's __name__ == "__main__" work?

- Used to check if a script is **executed directly** or **imported**.

```
if __name__ == "__main__":
    print("Executed directly")
else:
    print("Imported as module")
```

---

48. How do you implement a singleton class in Python?

A singleton ensures only **one instance** of a class exists.

```
class Singleton:
    _instance = None
    def __new__(cls):
        if cls._instance is None:
            cls._instance = super().__new__(cls)
        return cls._instance

s1 = Singleton()
s2 = Singleton()
print(s1 is s2)  # True
```


---

49. What is duck typing in Python?

- If an object **behaves like a type**, it **is** that type.

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

def make_quack(obj):
    obj.quack()

d = Duck()
make_quack(d)  # Works even if `Duck` is not explicitly a "duck"
```

---

50. What are Python's built-in data types?

1. **Numeric**: int, float, complex
2. **Sequence**: list, tuple, range, str
3. **Mapping**: dict
4. **Set**: set, frozenset
5. **Boolean**: bool
6. **Binary**: bytes, bytearray, memoryview

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

- **Lists** are **mutable** (can be changed).
- **Tuples** are **immutable** (cannot be changed).

```
lst = [1, 2, 3]
lst[0] = 100  # ✅ Allowed

tup = (1, 2, 3)
# tup[0] = 100  # ❌ TypeError: 'tuple' object does not support item assignment
```


---

52. How do you remove duplicates from a list in Python?

- Convert it to a **set** and back to a list.

```
lst = [1, 2, 2, 3, 4, 4, 5]
unique_lst = list(set(lst))
print(unique_lst)  # [1, 2, 3, 4, 5]
```

---

53. How do you reverse a list in Python?

- Using [::-1] or reverse().

```
lst = [1, 2, 3, 4]
print(lst[::-1])  # [4, 3, 2, 1]

lst.reverse()
print(lst)  # [4, 3, 2, 1]
```

---

54. How do you sort a list in Python?

- **Ascending order:** sorted(lst)
- **Descending order:** sorted(lst, reverse=True)
- **In-place sorting:** lst.sort()

```
lst = [5, 1, 4, 2]
print(sorted(lst))  # [1, 2, 4, 5]
lst.sort()
print(lst)  # [1, 2, 4, 5]
```

---

55. How do you find the index of an element in a list?

```
lst = ['apple', 'banana', 'cherry']
print(lst.index('banana'))  # 1
```

Tuple Questions

56. Why are tuples faster than lists in Python?

- **Tuples** are **immutable**, meaning Python can optimize their storage better than lists.
- Lists require extra memory for dynamic resizing.

---

57. How do you create a tuple with a single element?

- Add a **comma** after the element.

```
tup = (5,)  # ✅ Tuple
not_a_tuple = (5)  # ❌ Just an integer
```

---

58. How do you convert a list to a tuple?

```
lst = [1, 2, 3]
tup = tuple(lst)
print(tup)  # (1, 2, 3)
```

---

59. Can a tuple contain mutable objects?

- **Yes,** a tuple itself is immutable, but it can hold mutable objects like **lists**.

```
tup = ([1, 2], [3, 4])
tup[0].append(5)
print(tup)  # ([1, 2, 5], [3, 4])
```

---

60. How do you unpack values from a tuple?

```
tup = (10, 20, 30)
a, b, c = tup
print(a, b, c)  # 10 20 30
```

Set Questions

61. What are the key characteristics of a Python set?

- **Unordered** collection
- **No duplicate elements**
- **Mutable**

---

62. How do you create an empty set in Python?

- **Use set()**, not {} (which creates an empty dictionary).

```
s = set()  # ✅ Empty set
d = {}  # ❌ Empty dictionary
```

---

63. How do you add and remove elements from a set?

```
s = {1, 2, 3}
s.add(4)  # Adds 4
s.remove(2)  # Removes 2
print(s)  # {1, 3, 4}
```

---

64. How do you find the union and intersection of two sets?

```
a = {1, 2, 3}
b = {3, 4, 5}

print(a | b)  # Union: {1, 2, 3, 4, 5}
```

```
print(a & b)  # Intersection: {3}
```

---

65. How do you check if a set is a subset of another set?

```
a = {1, 2}
b = {1, 2, 3, 4}

print(a.issubset(b))  # True
print(b.issuperset(a))  # True
```

Dictionary Questions

66. What is a dictionary in Python?

- A dictionary stores **key-value pairs**.

```
d = {"name": "Alice", "age": 25}
print(d["name"])  # Alice
```

---

67. How do you iterate over keys and values in a dictionary?


```
d = {"a": 1, "b": 2}
for key, value in d.items():
    print(key, value)
```

---

68. What is the difference between get() and direct key access in a dictionary?

- get() **avoids KeyError** if the key is missing.

```
d = {"x": 10}
print(d.get("x"))  # 10
print(d.get("y", "Not Found"))  # Not Found
```

---

69. How do you merge two dictionaries in Python?

```
d1 = {"a": 1, "b": 2}
d2 = {"c": 3, "d": 4}

merged = {**d1, **d2}
```

```
print(merged)  # {'a': 1, 'b': 2, 'c': 3, 'd': 4}
```

---

70. How do you delete a key from a dictionary?

```
d = {"name": "Alice", "age": 25}

del d["age"]

print(d)  # {'name': 'Alice'}
```