Tab 1

# Python

🔹 Basic Python Questions

Q1. What are Python's key features?
Answer:

- Easy-to-read syntax
- Dynamically typed
- Interpreted language
- Extensive standard library
- Supports multiple programming paradigms
- Portable and open source

Q2. What is the difference between a list and a tuple?
Answer:

- List is mutable (`[1, 2, 3]`)
- Tuple is immutable (`(1, 2, 3)`)
- Lists have more built-in methods

Q3. What is the difference between `is` and `==` in Python?
Answer:

- `==` checks if values are equal
- `is` checks if two variables point to the same object in memory

Q4. What are Python's data types?
Answer:

- int, float, str, bool, list, tuple, dict, set, NoneType, bytes

Q5. What is a Python dictionary? How do you access and modify its elements?
Answer:

- A dictionary is a key-value data structure
- Example:

```python
d = {'name': 'John'}
print(d['name'])        # Access
d['age'] = 30           # Modify
```

Q6. What are *args and kwargs in Python?
Answer:

- `*args`: variable number of positional arguments
- `**kwargs`: variable number of keyword arguments
- Example:

```python
def func(*args, **kwargs): pass
```

Q7. What is the purpose of `self` in a class?
Answer:

- Refers to the instance of the class
- Used to access instance variables and methods

Q8. What is the difference between `range()` and `xrange()`?
Answer:

- `range()` returns a list (in Python 2) or a range object (Python 3)
- `xrange()` existed in Python 2, returns generator-like object
- In Python 3, only `range()` exists and behaves like `xrange`

Q9. Explain mutable and immutable objects.
Answer:

- **Mutable**: can be changed (`list`, `dict`, `set`)
- **Immutable**: cannot be changed (`int`, `str`, `tuple`)

Q10. What are Python's built-in data structures?
Answer:

- List, Tuple, Dictionary, Set

🔹 OOPs Concepts in Python

Q11. What are the four pillars of OOP?
Answer:

1. Encapsulation
2. Abstraction
3. Inheritance
4. Polymorphism

Q12. Explain inheritance and its types in Python.
Answer:

- Inheritance allows one class to acquire properties of another
- Types:
  - Single
  - Multiple
  - Multilevel
  - Hierarchical
  - Hybrid

Q13. What is the difference between `@staticmethod`, `@classmethod`, and instance methods?
Answer:

In Python, both `@staticmethod` and `@classmethod` are decorators used to define methods inside a class, but they differ in **how they interact with the class and its instances**:

🔹 `@staticmethod`

- Does **not take `self` or `cls`** as the first argument.
- It behaves like a regular function, just placed inside a class for organizational purposes.
- Cannot access or modify class or instance state.

```python
class MyClass:
    @staticmethod
    def greet(name):
        return f"Hello, {name}!"
MyClass.greet("Rushikesh")  # ✅ Works
```

🔹 `@classmethod`

- Takes **`cls` as the first argument**, referring to the class itself.
- Can access and modify **class variables or other class methods**.
- Useful for **factory methods** or when behavior depends on the class.

```python
class MyClass:
    value = 0
    @classmethod
    def set_value(cls, val):
        cls.value = val

MyClass.set_value(10)
print(MyClass.value)  # 👉 10
```

Q14. What is method overloading and overriding?
Answer:

- **Overloading**: Same method name, different arguments (not natively supported in Python)
- **Overriding**: Redefining a method in a child class that exists in the parent

Q15. What is a constructor in Python?
Answer:

- `__init__()` is the constructor
- Called automatically when object is created

🔹 Intermediate Python Concepts

Q16. What is list comprehension? Give examples.
Answer:

- Compact way to create a list

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

Q17. Explain how memory is managed in Python.
Answer:

- Automatic garbage collection
- Reference counting + cyclic garbage collector
- Memory managed by Python memory manager

Q18. What is the difference between deep copy and shallow copy?
Answer:

- **Shallow copy**: copies only references of nested objects
- **Deep copy**: creates new objects recursively

```python
import copy
shallow = copy.copy(obj)
deep = copy.deepcopy(obj)
```

Q19. What are generators and iterators?
Answer:

- **Generators**: functions using `yield` to produce values lazily
- **Iterators**: objects with `__iter__()` and `__next__()`

Q20. What is a lambda function? Where would you use one?
Answer:

- Anonymous function

```python
f = lambda x: x*2
```

- Used for short functions in `map()`, `filter()`, etc.

Q21. Explain Python's GIL (Global Interpreter Lock).
Answer:

- GIL allows only one thread to execute Python bytecode at a time
- Affects multithreaded performance

Q22. What is the difference between `map()`, `filter()`, and `reduce()`?
Answer:

- `map(func, iterable)`: transforms elements
- `filter(func, iterable)`: filters elements
- `reduce(func, iterable)`: reduces to single value

```python
from functools import reduce
reduce(lambda x, y: x+y, [1,2,3])
```

🔹 Error Handling & File I/O

Q23. What is exception handling in Python? How is it done?
Answer:

```python
try:
    x = 1 / 0
except ZeroDivisionError:
    print("Can't divide by 0")
finally:
    print("Done")
```

Q24. Difference between `try`, `except`, `finally` blocks.
Answer:

- `try`: block where error might occur
- `except`: handles the error
- `finally`: executes no matter what

Q25. What are some common built-in exceptions?
Answer:

- `TypeError`
- `ValueError`
- `IndexError`

- `KeyError`
- `ZeroDivisionError`
- `IOError`

Q26. How do you read and write files in Python?
Answer:

```python
with open('file.txt', 'r') as f:
    data = f.read()

with open('file.txt', 'w') as f:
    f.write('Hello')
```

🔹 Advanced Python

Q27. What are decorators? How and why are they used?
Answer:

- A decorator is a function that modifies the behavior of another function (or method or class) without changing its source code.
- Think of it like "wrapping" a function with extra functionality.
- A decorator is a function that takes another function as input and returns a modified function.

⚙️ Example:

```python
def decorator_function(original_function):
    def wrapper():
        print("Before the function call")
        original_function()
        print("After the function call")
    return wrapper

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

say_hello()
```

Output:

```None
Before the function call
Hello!
After the function call
```

- `@decorator_function` is the decorator syntax (shortcut for `say_hello = decorator_function(say_hello)`)

📦 Use Cases of Decorators:

- Logging
- Access control / authentication
- Measuring execution time
- Caching
- Input validation

🔐 Built-in Decorators:

- `@staticmethod`
- `@classmethod`
- `@property`

Q28. What is the difference between `@staticmethod` and `@classmethod`?
Answer:

- `staticmethod`: no access to class/instance
- `classmethod`: takes `cls` parameter, can modify class state

Q29. What is a context manager? (Explain with `with` statement)
Answer:

- A **context manager** is a construct that **sets something up, lets you use it, and then automatically cleans it up** — all using the `with` statement.
- A **context manager** manages resources like files, locks, database connections, etc., and ensures proper acquisition and release of those resources.

📂 Most Common Example – File Handling

```python
with open('example.txt', 'r') as file:
    content = file.read()
```

- `open()` returns a context manager.
- `with` ensures the file is **automatically closed**, even if an error occurs inside the block.

✅ Benefits of Context Managers:

- Cleaner and safer resource handling
- Automatically handles exceptions
- Great for managing setup/cleanup logic

Q30. What is multithreading vs multiprocessing in Python?
Answer:

- Multithreading: multiple threads, shared memory, affected by GIL
- Multiprocessing: multiple processes, true parallelism

Q31. How does Python handle memory management and garbage collection?
Answer:

- Reference counting
- `gc` module for cyclic GC
- Automatically done by Python runtime

🔹 Python for Data Engineering (if applicable)

Q32. How do you connect to a database using Python?
Answer:

```python
import sqlalchemy
engine =
sqlalchemy.create_engine("mysql+pymysql://user:pass@host/db")
```

Q33. What are some libraries you've used for ETL (e.g., `pandas`, `pyodbc`, `sqlalchemy`)?
Answer:

- `pandas`
- `sqlalchemy`
- `Pyodbc`
- `psycopg2`
- `pyarrow`
- `bigquery` libraries

Q34. How do you read large datasets efficiently in Python?
Answer:

- Use `pandas.read_csv()` with `chunksize`
- Use Dask or PySpark for large-scale processing

Q35. How do you process and transform data using `pandas` or `PySpark`?
Answer:

- **`pandas`:**

```python
df = pd.read_csv('file.csv')
df['new_col'] = df['col1'] + df['col2']
```

- **`PySpark`:**

```python
df = spark.read.csv('file.csv', header=True)
df = df.withColumn('new_col', df.col1 + df.col2)
```

🔹 Common Coding Questions

1. **Reverse a string:**

```python
s[::-1]
```

2. **Check palindrome:**

Python
```python
def is_palindrome(s): return s == s[::-1]
```

3. **Second largest in a list:**

Python
```python
lst = list(set(lst))
lst.sort()
second_largest = lst[-2]
```

4. **Character frequency:**

Python
```python
from collections import Counter
Counter("hello")
```

5. **Find duplicates in list:**

Python
```python
[x for x in set(lst) if lst.count(x) > 1]
```

6. **Fibonacci:**

Python
```python
def fib(n):
    a, b = 0, 1
    for _ in range(n):
        print(a, end=' ')
        a, b = b, a + b
```

7. **Flatten nested list:**

Python
```python
def flatten(lst):
    return [item for sublist in lst for item in sublist]
```

8. **Intersection of lists:**

```python
list(set(a) & set(b))
```