Python Interview

(Question and Answer)

1. Python Basics

1. Syntax, variables, data types (int, float, string, bool)
2. Type casting
3. Input/output
4. Operators (arithmetic, logical, comparison)

Q1: Python kya hai aur iski khasiyat kya hai?

Python ek high-level, interpreted, dynamically typed aur general-purpose programming language hai. Iska design simple aur readable hai, is wajah se beginners aur professionals dono ke liye suitable hai. Python me indentation-based syntax hota hai jo code ko readable banata hai. Python ka use web development, data science, machine learning, automation aur scripting sab me hota hai.

Q2: Python me variables ko kaise declare karte hain?

Python dynamically typed language hai, isliye variable banate waqt data type likhne ki zaroorat nahi hoti. Variable ka type runtime par decide hota hai. Variable name case-sensitive hote hain aur hamesha alphabet ya underscore se start karna chahiye.

Q3: Python me commonly used data types kaunse hain?

Python me basic data types hote hain: integers (`int`), floating point numbers (`float`), strings (`str`), boolean values (`bool`) aur complex numbers (`complex`). Ye sab internally objects hote hain kyunki Python me har cheez object ke form me hoti hai.

Q4: Python me type casting kaise hoti hai?

Python me type casting do tarah se hoti hai: ek explicit jo programmer manually karta hai jaise `int()`, `float()`, `str()`, aur ek implicit jahan Python automatically ek type ko dusre me convert kar deta hai jaise integer ko float me.

Q5: Python me user input lene ka tareeqa kya hai?

Python me `input()` function se user input liya jata hai aur iska default type hamesha string hota hai. Agar input ko number ki form me use karna ho to usse casting karna padta hai, jaise `int(input())`.

Q6: Python me operators kaise kaam karte hain?

Python me arithmetic, comparison, logical aur assignment operators use hote hain. Ek khasiyat ye hai ki Python me operator overloading support hota hai. Jaise plus operator numbers ke liye addition karta hai aur strings ke liye concatenation.

Q7: Python me Boolean values condition me kaise behave karti hain?

Python me `True` aur `False` special objects hain jo integers se inherit karte hain. Condition check karte waqt `0`, `""`, `[]`, None false consider hote hain aur baaki sab true.

Q8: Python dynamically typed language hone ka fayda aur nuksan kya hai?

Fayda ye hai ki development fast hoti hai aur kam code likhna padta hai. Lekin nuksan ye hai ki type mismatch ki wajah se runtime errors ka chance zyada hota hai.

Q9: Python me `is` aur == me kya fark hai?

== values ko compare karta hai jabki `is` memory location (object identity) ko compare karta hai. Matlab do alag-alag objects same values rakhte ho sakte hain lekin unka identity alag hoga.

Q10: Python me constants kaise banaye jate hain?

Python me built-in constants ka system nahi hai jaise C ya Java me hota hai. Lekin convention ke taur pe uppercase variable names ko constant treat kiya jata hai. Python 3.8 se `typing.Final` ka use karke constant ko enforce karne ka option bhi milta hai.

2. Control Flow

1. If/else statements
2. Loops (for, while)
3. Loop control (break, continue, pass)

Q11: Python me if/else ladder aur nested if ke beech ka difference kya hai, aur performance par iska kya asar hota hai?

If/else ladder sequentially har condition ko evaluate karta hai, jab tak ek true condition na mil jaye. Nested if ek condition ke andar dusri condition check karta hai. Performance ke liye ladder better hota hai jab multiple independent conditions ho, lekin nested if tab useful hota hai jab ek condition ke andar hi dusra logical check required ho. Agar unnecessary nested if banaye jayein to readability aur debugging mushkil ho jaata hai.

Q12: Python me if condition me non-boolean values (jaise list, dict, string) kaise evaluate hoti hain?

Python truthy aur falsy concept use karta hai. Non-empty strings, lists, dictionaries truthy hoti hain, jabke empty values aur 0 falsy hote hain. Ye feature conditional checks ko concise banata hai.

Example

```
if []:
    print("True")
else:
    print("False")   # output: False
```

Q13: Python me "ternary operator" ya conditional expression ka use kyu hota hai?

Ternary operator ek single line if/else hota hai jo concise aur readable code ke liye use hota hai. Ye especially list comprehensions ya lambdas me kaam aata hai.
Example:

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

Q14: For loop aur While loop me performance aur use-cases me kya farq hai?

For loop fixed iteration count ya iterable par iterate karne ke liye best hai (lists, tuples, range). While loop tab useful hota hai jab iterations unknown ho aur ek condition ke basis par terminate karna ho. Performance wise dono similar hain, lekin for loop optimized hai iterable traversal ke liye.

Q15: Python me for loop ke andar `else` clause ka practical use kya hai?

For loop ka else tab execute hota hai jab loop normally terminate ho (break ke bina). Ye feature unique hai Python me aur generally search operations me use hota hai jahan agar element na mile to else run ho.
Example:

```
for i in [1,2,3]:
    if i == 5:
        break
else:
    print("Not Found")  # executes since loop completed
```

Q16: Python me `break`, `continue`, aur `pass` ke real-world scenarios kya ho sakte hain?

- **break**: Jab ek loop ko immediately terminate karna ho (e.g., password match milte hi stop).
- **continue**: Jab current iteration skip karke agla continue karna ho (e.g., skip invalid entries).
- **pass**: Placeholder for future code ya abstract structure maintain karne ke liye.

Q17: Nested loops ke andar `break` sirf inner loop todta hai ya outer loop bhi? Agar outer loop todna ho to kya karte hain?

Default me `break` sirf current (inner) loop terminate karta hai. Outer loop todne ke liye flags ya exceptions use karte hain. Advanced approach me functions ya custom logic likh kar handle kiya jata hai.

Q18: Infinite loop detect karna aur usse bachna Python me kaise hota hai?

Infinite loop tab hota hai jab termination condition kabhi false na ho. Isse bachne ke liye:

- Condition clearly define karna
- Counters/limits lagana
- Debugging ke liye timeouts use karna (threading ya signal modules ke through)

Q19: Python me loop ke andar resource-heavy operations optimize karne ke techniques kya hain?

- Loop ke andar constant calculations bahar shift karna
- List comprehension use karna instead of explicit loops
- Built-in functions (map, filter, sum, any, all) ka use karna jo C optimized hain
- Generator expressions use karke memory efficiency lana

Q20: Python me list comprehensions aur traditional for loops ke beech kya difference hai aur kis situation me comprehension use karna chahiye?

List comprehension concise aur fast hote hain kyunki internally optimized C implementation use karte hain. Ye simple transformations aur filtering ke liye best hote hain. Lekin agar complex logic, multiple statements ya debugging zaroori ho to traditional for loop zyada readable hota hai.
Example:
```
squares = [x*x for x in range(10) if x % 2 == 0]
```

3. Data Structures

1. Lists, Tuples
2. Dictionaries, Sets
3. List comprehensions

Q21. List aur tuple ke beech immutability ka kya difference hai, aur kis case me tuple prefer karna chahiye?

- **List** mutable hoti hai (update, append, delete possible).
- **Tuple** immutable hoti hai (ek bar bana to change nahi kar sakte).
- Tuple ka use tab karte hain jab **data constant hona chahiye** (e.g., configuration values, dictionary keys). Tuple ka access fast hota hai kyunki hashing aur immutability ke wajah se optimization hota hai.

Q22. Python list ka internal memory structure (dynamic array resizing) kaise kaam karta hai?

Python list internally **dynamic array** hoti hai.

- Jab space khatam hota hai, list automatically apna size **resize** karti hai (usually 1.125x ya 1.25x badh jata hai).
- Isse insertion **amortized O(1)** hota hai.
- Lekin kabhi-kabhi resize hone par O(n) cost lagta hai kyunki memory copy karni padti hai.

Q23. Dictionary me hashing ka kya role hai aur collision handling kaise hoti hai?

- Dictionary internally **hash table** use karti hai.
- Key ka hash banake index decide hota hai.
- Agar **collision** hota hai (same index par do key aa jayein) to Python **open addressing** (probing) ka use karta hai.
- Isi wajah se dictionary ka access average **O(1)** hota hai.

Q24. Set aur frozenset me kya difference hai, aur frozenset kab useful hota hai?

- **Set** mutable hai (elements add/remove kar sakte hain).
- **Frozenset** immutable hai (banne ke baad change nahi hota).
- Frozenset ka use tab hota hai jab hume **set ko dictionary key** ya ek aur set ke element ki tarah store karna ho (kyunki mutable types hashable nahi hote).

Q25. List comprehension vs generator expression – performance aur memory usage me kya farq hai?

- **List comprehension** ek pura list bana deta hai memory me (O(n) space).
- **Generator expression** ek iterator banata hai jo values lazily generate karta hai (O(1) space).
- Agar large dataset hai aur hume ek-ek element sequentially chahiye, to generator better hai.

Q26. Dictionary me insertion order maintain hota hai? Agar haan, to Python ke kis version se?

- Python 3.6 me insertion order **implementation detail** tha.
- Python 3.7+ se officially guarantee ho gaya hai ki dictionary insertion order maintain karegi.
- Matlab agar aap keys insert karte hain `{"a":1, "b":2, "c":3}`, to iteration hamesha usi order me milega.

Q27. Python me nested list flatten karne ke 3 different tarike batayein (iteration, recursion, itertools).

Iteration:

```
nested = [[1, 2], [3, 4], [5]]
flat = [x for sublist in nested for x in sublist]
print(flat)  # [1, 2, 3, 4, 5]
```

Recursion:

```
def flatten(lst):
    result = []
    for i in lst:
        if isinstance(i, list):
            result.extend(flatten(i))
        else:
            result.append(i)
    return result

print(flatten([[1, [2, 3]], 4]))  # [1, 2, 3, 4]
```

✅ **itertools.chain:**`import itertools`

```
nested = [[1, 2], [3, 4], [5]]
flat = list(itertools.chain.from_iterable(nested))
print(flat)  # [1, 2, 3, 4, 5]
```

4. Functions

1. Defining functions
2. \args and *kwargs
3. Lambda functions
4. Recursion

Q28: Python me function ek "first-class citizen" kyon hai?

Answer: Python me functions ko variables me assign kar sakte ho, dusre function ko argument ke taur pe pass kar sakte ho, aur return bhi kar sakte ho. Isi wajah se function first-class object hai.

```
def greet(name): return f"Hello {name}"
say = greet
print(say("Aman"))  # Hello Aman
```

Q29: *args aur **kwargs me kya farq hai aur inka use kab hota hai?**
Answer:

- `*args` multiple positional arguments ko tuple ke form me leta hai.
- `**kwargs` keyword arguments ko dictionary ke form me leta hai.
  Useful jab function me unknown number of inputs handle karne ho.

Q30: Lambda function aur normal function me performance aur readability ke terms me kya difference hai?
Answer: Lambda ek anonymous one-liner function hota hai, readability ke liye use hota hai (jaise `map`, `filter`). Performance almost same hota hai as normal `def`, but readability ke liye simple cases me lambda prefer hota hai. Complex logic ke liye normal function better hai.

Q31: Python recursion me maximum depth limit kya hai aur usko kaise handle karte hain?
Answer: Default recursion limit ~1000 hoti hai (`sys.getrecursionlimit()`). Deep recursion me `RecursionError` aa sakta hai. Handle karne ke liye:

- Iterative approach use karo
- Ya `sys.setrecursionlimit(new_limit)` set kar sakte ho (lekin risky).

Q32: Python me closures kya hote hain?
Answer: Closure ek aisa function hai jo apne enclosing scope ke variables ko yaad rakhta hai, even after outer function finish ho jata hai.

```
def outer(x):
    def inner(y):
        return x + y
    return inner

add5 = outer(5)
```

```
print(add5(10))  # 15
```

Q33: Higher-order functions kya hote hain?

Higher-order function wo hote hain jo ek function ko argument ke roop me lete hain ya ek function return karte hain. Example: `map`, `filter`, `sorted(key=func)`.

Q34: Python me tail recursion optimization kyun nahi hota?
CPython me tail recursion optimization intentionally implement nahi kiya gaya, kyunki debugging aur stack trace readability preserve karna chahte the. Isliye deeply recursive problems me iterative ya generator approach prefer hoti hai.

5. Object-Oriented Programming (OOP)

1. Classes & Objects
2. Constructors (__*init*__)
3. Inheritance, Polymorphism
4. Encapsulation, Abstraction

Q35: Python me `self` ka role kya hai?
Answer: `self` ek instance ko represent karta hai aur iski properties/methods ko access karne ke liye use hota hai. Python automatically object pass karta hai as first argument in instance methods.

Q36: `__init__` aur `__new__` me kya farq hai?
Answer:

- `__new__` ek new object banata hai (memory allocation).
- `__init__` object ko initialize karta hai (values set karna).
  Usually sirf `__init__` override karte hain, `__new__` tab use hota hai jab immutable classes (jaise tuple, str) customize karni ho.

Q37: Python me multiple inheritance me ambiguity kaise resolve hoti hai?
Answer: Method Resolution Order (MRO) ke through — Python C3 linearization algorithm use karta hai. Is order ko `ClassName.__mro__` ya `ClassName.mro()` se check kar sakte ho.

Q38: Polymorphism Python me kaise achieve hota hai?
Answer: Duck typing se — agar object ke pass required method/attribute hai, to uska behavior chalega irrespective of class type. Example: `len()` string, list, tuple sab pe kaam karta hai.

Q39: Encapsulation aur abstraction Python me practically kaise implement karte hain?
Answer:

- Encapsulation: `_protected` aur `__private` naming convention se access restrict karte hain.
- Abstraction: `abc` module ke through abstract classes banate hain jisme sirf method signatures hote hain, implementation derived class me hota hai.

Q40: Python me staticmethod aur classmethod me kya farq hai?
Answer:

- `@staticmethod`: instance aur class dono ka reference nahi leta; simple utility functions ke liye.
- `@classmethod`: class ka reference (`cls`) leta hai, aur alternate constructors banane ke liye useful hota hai.

Q41: Python me composition vs inheritance me kya difference hai?
Answer:

- Inheritance "is-a" relationship dikhata hai (Car is a Vehicle).
- Composition "has-a" relationship dikhata hai (Car has an Engine).
  Composition zyada flexible hai aur overuse of inheritance avoid karta hai.

6. Modules & Packages

1. import, built-in modules
2. Creating your own module
3. pip & virtual environments

Q42: Python me module aur package me kya difference hai?
Answer:

- Module = ek single `.py` file.
- Package = ek folder jisme multiple modules ho aur ek `__init__.py` file ho.

Q43: Built-in module `math` aur `random` ka ek advanced use-case batao.
Answer:

- `math.isclose(a,b)` → floating point precision compare karna.
- `random.choices(population, weights, k)` → weighted random sampling.

Q44: Python me `import` statement internally kaise kaam karta hai?
Answer: Jab import hota hai, Python `sys.modules` dictionary check karta hai. Agar module already loaded hai to wahi reference return karta hai; otherwise naya load karta hai aur cache me daal deta hai.

Q45: Apna khud ka module banane ke liye minimum requirements kya hain?
Answer:

1. Ek `.py` file banani hai jisme functions/classes define ho.
2. Us file ko importable location pe rakho (same folder ya `PYTHONPATH` me).

Q46: Python me relative aur absolute import me kya farq hai?
Answer:

- Absolute import: project root se poora path likhte hain (`from package.module import func`).
- Relative import: dot notation use hota hai (`from .module import func`).

Q47: Virtual environment ka use kyon important hai?
Answer: Virtual env isolated environment provide karta hai jisme project-specific dependencies install kar sakte ho bina system-wide libraries ko affect kiye. Isse version conflicts avoid hote hain.

Q48: pip aur conda me kya difference hai?
Answer:

- pip → Python-only packages install karta hai from PyPI.
- conda → language-agnostic hai; Python ke saath-saath C libraries, compilers, aur environment management bhi karta hai.

7. Error Handling

1. Try, except, finally
2. Custom exceptions

Q49: Python me `try-except-finally` block ka kaam kya hai?
Answer:

- `try`: risky code run karta hai
- `except`: exception handle karta hai
- `finally`: hamesha execute hota hai (cleanup, closing files).

```
try:
    f = open("data.txt")
    data = f.read()
except FileNotFoundError:
    print("File not found!")
finally:
    print("Cleanup done.")
```

Q50: Agar multiple `except` blocks ho to Python kaise decide karta hai kaunsa chalega?
Answer: Sabse pehle jo matching exception class milegi wahi chalegi. Order matter karta hai (specific → general order).

Q51: Exception chaining (`raise ... from ...`) ka use kya hai?
Answer: Isse ek custom error ko original error ke context ke sath raise kar sakte ho. Debugging ke liye helpful hai.

```
try:
    int("abc")
except ValueError as e:
    raise RuntimeError("Conversion failed") from e
```

Q52: Python me apna custom exception kaise banate hain?
Answer: Bas `Exception` class inherit karte hain.

```
class InvalidAgeError(Exception):
    pass
```

Q53: `assert` statement aur exception handling me kya difference hai?
Answer:

- `assert` mainly debugging ke liye hota hai. Agar condition false ho to `AssertionError`.
- Exception handling production-level errors handle karta hai.

Q54: Agar `try` block me return ho aur `finally` block bhi ho to kaunsa execute hoga?
Answer: Pehle `finally` chalega, phir return. Example:

```
def test():
    try:
        return 1
    finally:
        print("Finally executed")
print(test())  # Finally executed \n 1
```

Q55: Checked vs unchecked exceptions ka Python me kya concept hai?
Answer: Python me Java jaise "checked exceptions" nahi hote. Sare exceptions runtime pe handle hote hain.

8. File Handling

1. Reading/Writing files
2. Working with CSV/JSON

Q56: Python me file read karte waqt `with open()` kyun prefer karte ho?
Answer: `with` block automatically file close kar deta hai, even agar error aa jaye to.

Q57: Python me file modes `"r"`, `"w"`, `"a"`, `"x"` me difference kya hai?
Answer:

- `"r"` = read (error if not exists)
- `"w"` = write (overwrite)
- `"a"` = append
- `"x"` = create new file (error if already exists)

Q58: Agar large file read karni ho to memory-efficient approach kya hai?
Answer: `readline()` ya iterate line by line instead of `read()`.

```
with open("big.txt") as f:
    for line in f:
        print(line.strip())
```

Q59: Python me CSV file kaise read/write karte hain?
Answer: `csv` module use hota hai.

```
import csv
with open("data.csv") as f:
    reader = csv.reader(f)
    for row in reader:
        print(row)
```

Q60: JSON me `dumps()` aur `dump()` me kya farq hai?
Answer:

- `dumps()` → Python object ko JSON string me convert karta hai.
- `dump()` → JSON directly file me likhta hai.

9. Decorators & Generators (Basics)

1. @decorator
2. yield & generator functions

Q61: Python me decorator kya hota hai?
Answer: Function jo dusre function ko modify karta hai without changing its code. Mostly @ syntax se lagate hain.

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

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

Q62: Higher-order functions aur decorators me kya relation hai?
Answer: Decorator ek special case hai higher-order function ka, jo ek function ko input lekar modified function return karta hai.

Q63: Generator function aur normal function me kya difference hai?
Answer: Normal function ek hi value return karke khatam ho jata hai. Generator `yield` use karke ek-ek value produce karta hai without losing state.

Q64: Generator ka ek real-world use case batao.
Answer: Large data stream (jaise log file, API response) ko lazily process karna without loading entire data in memory.

Q65: Generator expressions aur list comprehensions me kya farq hai?
Answer:

- `[x*x for x in range(1000)]` → list comprehension → memory heavy.
- `(x*x for x in range(1000))` → generator expression → memory efficient (lazy).

10. Advanced Concepts (Intro)

1. Iterators
2. Context Managers (with)
3. Regular Expressions (re)
4. Multithreading & Multiprocessing (basic)

Q66: Python me iterator aur iterable me kya farq hai?
Answer:

- Iterable = object jisko loop kar sakte ho (list, tuple, string).
- Iterator = object jo `__iter__()` aur `__next__()` implement karta hai.

Q67: Agar khud ka iterator banana ho to kaise karte ho?
Answer:

```
class Counter:
    def __init__(self, n):
        self.n, self.current = n, 0
    def __iter__(self):
        return self
    def __next__(self):
        if self.current < self.n:
            self.current += 1
            return self.current
        raise StopIteration
```

Q68: Context manager (`with` statement) internally kaise kaam karta hai?
Answer: Class ke andar `__enter__()` aur `__exit__()` methods hote hain jo resource allocate aur release karte hain.

Q69: Python me `re.match()` aur `re.search()` me kya farq hai?
Answer:

- `match()` → sirf string ke start me check karta hai.
- `search()` → poore string me check karta hai.

Q70: `re.findall()` aur `re.finditer()` me kya farq hai?
Answer:

- `findall()` → list return karta hai (all matches).
- `finditer()` → iterator return karta hai (efficient for large data).

Q71: Python me multithreading GIL ki wajah se limited kyun hoti hai?
Answer: CPython me Global Interpreter Lock ek samay pe sirf ek thread ko Python bytecode execute karne deta hai. CPU-bound tasks (heavy computation) me threads slow ho jate hain.

Q72: Multithreading aur multiprocessing me kya difference hai?
Answer:

- Multithreading → threads ek hi memory share karte

Q73. How does Python internally implement Iterators and what happens when `StopIteration` is raised?
Ans: Iterator wo object hai jo `__iter__()` aur `__next__()` methods implement karta hai. Jab data finish ho jata hai to `__next__()` `StopIteration` raise karta hai, jise `for` loop internally handle karta hai aur loop khatam kar deta hai.

Q74. How do Context Managers ensure proper resource cleanup even if an exception occurs inside the block?
Ans: Context manager `__enter__()` aur `__exit__()` methods use karta hai. Agar `with` block ke andar error bhi aa jaye to bhi `__exit__()` hamesha run hota hai jisse resources (file close, lock release, rollback) properly clean ho jate hain.

Q75. What is the difference between `re.match()`, `re.search()` and `re.findall()` in Python?
Ans:

- `re.match()` → sirf string ke start se pattern check karta hai.
- `re.search()` → puri string scan karke pehla match return karta hai.
- `re.findall()` → saare matches list ke form mein return karta hai.

Q76. Why is Python Multithreading not suitable for CPU-bound tasks?
Ans: Python mein Global Interpreter Lock (GIL) hota hai jo ek waqt mein sirf ek thread ko Python bytecode chalane deta hai. Is wajah se multithreading CPU-bound tasks ke liye fast nahi hota, lekin I/O-bound tasks ke liye useful hai.

Q77. How does Multiprocessing bypass the Global Interpreter Lock (GIL)?
Ans: Multiprocessing har process ko alag memory aur interpreter deta hai. GIL process-specific hota hai, isliye har process apne CPU core par parallel chal sakta hai aur GIL ka limitation remove ho jata hai.

Q78. In terms of memory and communication, how do Multithreading and Multiprocessing differ?
Ans:

- **Multithreading:** Threads ek hi memory share karte hain, isliye fast hota hai lekin race conditions ka risk hota hai (synchronization chahiye).

- **Multiprocessing:** Processes alag memory use karte hain, data share karne ke liye IPC (pipes/queues) use hota hai jo slow hai but safe hai.

11. Python Internals

1. How Python executes code (bytecode, CPython vs PyPy)
2. Memory management
3. Global Interpreter Lock (GIL)

Q79: Python code ka execution process kaisa hota hai (bytecode, CPython vs PyPy)?

Answer:
Python code pehle source code se bytecode me convert hota hai (intermediate representation). CPython interpreter bytecode ko execute karta hai. PyPy ek alternative interpreter hai jo JIT compilation provide karta hai aur faster execution deta hai.

Q80: Python me memory management kaise hota hai?

Answer:
Python automatic memory management provide karta hai. Garbage collector unused objects ko remove karta hai aur memory free karta hai. Reference counting aur cyclic garbage collection use hota hai.

Q81: Global Interpreter Lock (GIL) kya hai?

Answer:
GIL ek mutex hai jo Python interpreter me ek time me sirf ek thread ko CPU execute karne

deta hai. Ye multithreading me thread-level parallelism ko restrict karta hai, lekin memory safety ensure karta hai.

12. Advanced OOP in Python

1. MRO (Method Resolution Order)
2. Dunder methods (*str, **repr, *_eq_, etc.)
3. @property, setters/getters
4. Abstract Base Classes (ABC module)

Q82: Method Resolution Order (MRO) kya hai?

Answer:
MRO decide karta hai ki multiple inheritance me method ya attribute kis class se pick hoga. Python me C3 linearization algorithm use hoti hai.

Q83: Dunder methods kya hote hain aur kyun important hain?

Answer:
Dunder methods special methods hote hain jaise `__str__`, `__repr__`, `__eq__`. Ye Python objects ke behavior customize karte hain jaise printing, equality comparison, operator overloading.

Q84: @property decorator ka use kyun karte hain?

Answer:
`@property` getter aur setter define karne ke liye use hota hai. Isse encapsulation maintain hoti hai aur class attributes ko method ki tarah access kar sakte hain.

Q85: Abstract Base Classes (ABC) ka use kya hai?

Answer:
ABC module se abstract classes create hoti hain jo blueprint provide karti hain. Concrete classes ko unke abstract methods implement karne padte hain.

13. Metaprogramming

1. type() vs class
2. Metaclass
3. eval(), exec()

Q86: `type()` aur `class` me kya difference hai?

Answer:
`class` se normal class banate hain. `type()` dynamically runtime me class create karne ke liye use hota hai.

Q87: Metaclass kya hoti hai aur kyun use hoti hai?

Answer:
Metaclass class ke liye class hoti hai. Ye runtime me class creation customize karne ke liye use hoti hai. Frameworks jaise Django me validation ke liye use hoti hai.

Q88: Python me `eval()` aur `exec()` ka use kya hai?

Answer:
`eval()` single expression evaluate karta hai aur result return karta hai.

`exec()` Python code ko dynamically execute karta hai (statements) aur return value nahi deta.

Most Important Example:

```
x = 10
expr = "x + 5"
result = eval(expr)  # 15
```

14. Decorators (Deep Dive)

1. Nested decorators
2. Parameterized decorators
3. Class-based decorators

Q89: Nested decorators kya hote hain aur kab use hote hain?

Answer:
Nested decorators ek function par multiple decorators apply karne ka tarika hai. Ye function ke behavior ko sequentially modify karte hain.

Q90: Parameterized decorators kya hote hain?

Answer:
Ye decorators arguments accept kar sakte hain. Function ke behavior ko customize karne ke liye use hote hain.

Most Important Example:

```
def repeat(n):
    def decorator(func):
        def wrapper():
            for _ in range(n):
                func()
        return wrapper
    return decorator

@repeat(3)
def greet():
    print("Hello Aman")
```

Q91: Class-based decorators kya hote hain?

Answer:
Class-based decorators class ke `__call__` method ko implement kar ke function ko decorate karte hain. Ye stateful decorators ke liye useful hain.

Q92: Decorators ka use code me maintainability aur readability ke liye kaise hota hai?

Answer:
Decorators repetitive code ko reduce karte hain aur cross-cutting concerns (logging, authentication) ko centralized handle karte hain.

Q93: Multiple decorators ek function pe kaise behave karte hain?

Answer:
Multiple decorators top-to-bottom apply hote hain. Pehla decorator sabse upar execute hota hai aur last decorator sabse neeche execute.

15. Generators & Iterators (Advanced Use)

1. Chaining generators
2. Custom iterator classes
3. Coroutine basics (yield from)

Q94: Chaining generators kya hota hai?

Answer:
Ek generator ka output dusre generator me input ban sakta hai, jise generator chaining kehte hain. Ye memory-efficient pipeline create karta hai.

Q95: Custom iterator class kaise banate hain?

Answer:
`__iter__()` aur `__next__()` methods implement karke custom iterator banate hain. Ye sequence elements ko controlled access provide karta hai.

Q96: `yield from` ka use kya hai?

Answer:
`yield from` ek generator ke nested generator ko flatten kar deta hai aur values directly parent generator me yield hoti hain.

Q97: Generator expressions aur list comprehensions me kya difference hai?

Answer:
Generator expressions lazy evaluation provide karte hain aur memory efficient hote hain, list comprehensions poori list memory me store kar lete hain.

Q98: Iterators aur iterables me kya difference hai?

Answer:
Iterable object se `iter()` function se iterator create hota hai. Iterator ek object hai jo next element return karta hai aur exhausted hone par StopIteration raise karta hai.

16. Context Managers (Advanced)

1. Custom context managers (*enter, *exit*)
2. contextlib module

Q99: Custom context manager kaise banate hain?

Answer:
`__enter__()` aur `__exit__()` methods implement karke custom context manager banate hain jo resource allocation aur cleanup handle kare.

Most Important Example:

```
class MyFile:
    def __init__(self, filename):
        self.filename = filename
    def __enter__(self):
        self.file = open(self.filename, 'r')
        return self.file
    def __exit__(self, exc_type, exc_value, traceback):
        self.file.close()

with MyFile('file.txt') as f:
    data = f.read()
```

Q100: `contextlib` module ka use kya hai?

Answer:
`contextlib` simple aur reusable context managers create karne ke liye utilities provide karta hai jaise `contextmanager` decorator.

Q101: Context manager ka main fayda kya hai?

Answer:
Automatic resource management, exceptions hone par bhi cleanup ensure karna, aur code readability improve karna.

Q102: Multiple context managers ek sath kaise use karte hain?

Answer:
`with` statement me multiple managers comma se separate karke use kar sakte hain.

```
with open('f1.txt') as f1, open('f2.txt') as f2:
    pass
```

Q103: Context manager ka __exit__ parameters kya represent karte hain?

Answer:
`exc_type`, `exc_value`, `traceback` exceptions ke type, value aur stack trace ko represent karte hain. Agar exception handle karna ho to True return karna padta hai.

17. Concurrency in Python

1. Threading vs Multiprocessing
2. asyncio, event loop
3. Futures & Tasks

Q104: Threading aur multiprocessing me difference kya hai?

Answer:
Threading → same memory space, CPU-bound me limited due to GIL
Multiprocessing → alag memory space, CPU-bound me true parallelism

Q105: asyncio aur event loop ka use kya hai?

Answer:
Asyncio asynchronous I/O handle karta hai aur event loop tasks ko schedule karta hai. CPU-bound tasks me nahi, I/O-bound tasks me efficient.

Q106: Future aur Task object kya hai asyncio me?

Answer:
Future object ek promise hai ki value future me available hogi. Task ek coroutine ko schedule karne aur run karne ka handle provide karta hai.

Q107: Python me deadlock kya hota hai aur kaise avoid karte hain?

Answer:
Deadlock multiple threads ya processes resource ke liye indefinitely wait karte hain. Avoidance: timeouts, proper locking order, aur `threading.Lock` ya `multiprocessing.Lock` ka careful use.

Q108: Thread-safe code ka matlab kya hai?

Answer:
Thread-safe code wo hota hai jo multiple threads me simultaneously execute hone par bhi correct result de aur data corruption na ho.

18. Descriptors

1. *get, **set, *_delete_
2. Use in frameworks like Django

Q109: Descriptors kya hote hain Python me?

Answer:
Descriptors ek object-oriented protocol hai jo attribute access ko customize karta hai. Ye objects me `__get__`, `__set__`, aur `__delete__` methods implement karte hain.

Q110: `__get__`, `__set__`, aur `__delete__` ka kya role hai?

Answer:
`__get__` → attribute read karte waqt call hota hai
`__set__` → attribute assign karte waqt call hota hai
`__delete__` → attribute delete karte waqt call hota hai

Q111: Python me descriptors ka main use kya hai?

Answer:
Validation, type checking, lazy evaluation, aur computed properties implement karne me.

Q112: Data descriptors aur non-data descriptors me kya difference hai?

Answer:
Data descriptors → `__set__` ya `__delete__` implement karte hain, attribute assignment ko control karte hain
Non-data descriptors → sirf `__get__` implement karte hain, assignment ko override nahi karte

Q113: Descriptors Django me kaise use hote hain?

Answer:
Django ORM me fields (IntegerField, CharField) descriptors ka use karke validation aur database access manage karte hain.

19. Typing & Static Analysis

1. Type hints (List[int], Dict[str, Any], etc.)
2. mypy, pyright

Q114: Type hints kya hote hain aur unka use kya hai?

Answer:
Type hints variables aur functions ke expected types specify karte hain. Ye runtime error nahi rokta, lekin static analysis tools me errors detect karne me help karta hai.

Q115: List[int], Dict[str, Any] ka matlab kya hai?

Answer:
List[int] → integer elements wali list
Dict[str, Any] → keys string aur values kisi bhi type ke ho sakte hain

Q116: mypy aur pyright ka use kya hai?

Answer:
Ye static type checkers hain jo code me type hints ke hisaab se possible type errors detect karte hain aur code quality improve karte hain.

Q117: Type hints ke fayde kya hain?

Answer:
Code readability, error detection, IDE autocomplete support, aur maintainability improve karte hain.

Q118: Optional aur Union type hints ka use kaise hota hai?

Answer:
Optional → variable ya function argument None ya given type ho sakta hai
Union → variable multiple types accept kar sakta hai

```
from typing import Optional, Union

def greet(name: Optional[str] = None):
    if name:
        print(f"Hello {name}")
    else:
        print("Hello Guest")
```

20. Packaging & Distributing Python Code

1. Create your own pip-installable packages
2. setup.py, pyproject.toml
3. Versioning, dependencies

Q119: Pip-installable package kaise create karte hain?

Answer:
Python project ko package me organize karke `setup.py` ya `pyproject.toml` define karte hain aur pip se installable banate hain.

Q120: `setup.py` aur `pyproject.toml` me kya difference hai?

Answer:
`setup.py` traditional build script hai
`pyproject.toml` modern standard, build system aur dependencies specify karta hai

Q121: Versioning aur dependencies kaise manage karte hain?

Answer:
Semantic versioning (major.minor.patch) follow karte hain. Dependencies ko `install_requires` me list karte hain.

Q122: Python package distribute karne ka standard process kya hai?

Answer:

1. Package structure create karna
2. Metadata define karna (`setup.py`)
3. Build (`python -m build`)
4. Upload (`twine upload dist/*`)

Q123: Local development aur virtual environment ka use kyu zaruri hai?

Answer:
Dependencies isolated rakhne ke liye, system packages me conflict avoid karne ke liye aur project reproducibility ke liye zaruri hai.

21. Memory & Performance Optimization

1. sys, gc modules
2. Profiling code (cProfile, timeit)

3. Writing memory-efficient code (e.g. generators vs lists)

Q124: sys aur gc module ka use kya hai?

Answer:
`sys` → memory usage, interpreter info, path management
`gc` → garbage collection control aur memory leaks detect karne ke liye

Q125: Code profiling ka matlab aur kaise karte hain?

Answer:
Profiling se code ke bottlenecks detect karte hain. Tools: `cProfile`, `timeit`

Q126: Generators vs Lists me memory efficiency kaise compare karte hain?

Answer:
Generators lazy evaluation karte hain, list memory me saari values store kar leti hai → large datasets me generators memory efficient hain

Most Important Example:

```
# generator
gen = (x*x for x in range(1000000))
```

Q127: Memory-efficient code likhne ke aur kya techniques hain?

Answer:

- `__slots__` ka use karke class memory reduce karna
- Lazy evaluation
- Efficient data structures (sets vs lists, dict views)

Q128: Python me multithreading CPU-bound tasks ke liye kyun limited hai?

Answer:
Global Interpreter Lock (GIL) ek time me sirf ek thread ko Python bytecode execute karne deta hai. Isliye CPU-bound tasks me multithreading parallelism nahi provide karta.