Python Interview Questions Guide

give me python 1000 interview question in each topic wise each 100 question to better understand and crack one shot interview
please dont say i cant genertate

Absolutely! Here's how we can break down **1000 Python interview questions**, grouped into **10 major topics**, with **100 questions each**:

🧠 1. Python Basics (100 Questions)

Covers syntax, keywords, variables, data types, operators, input/output, etc.

Examples:

- What is the difference between `is` and `==`?
- How are Python variables typed?
- Explain the use of `pass` in Python.

📚 2. Data Structures in Python (100 Questions)

Focuses on lists, tuples, sets, dictionaries, and how to use them.

Examples:

- What is the difference between a list and a tuple?
- How does Python handle set operations?
- How do you merge two dictionaries?

🔁 3. Control Flow & Loops (100 Questions)

If-else, for loops, while loops, break/continue/pass.

Examples:

- How does Python's for-loop work?
- Difference between `break` and `continue`?
- What is the purpose of `else` in loops?

🧩 4. Functions & Lambda (100 Questions)

Includes defining functions, `*args`, `**kwargs`, lambda, recursion.

Examples:

- What are `*args` and `**kwargs` used for?
- How do lambda functions differ from def functions?
- What is recursion and how is it implemented?

🏗️ 5. Object-Oriented Programming (100 Questions)

Classes, objects, inheritance, polymorphism, encapsulation.

Examples:

- What is `__init__` method?
- Explain multiple inheritance in Python.
- How do private and protected members work?

🧪 6. Exception Handling & Debugging (100 Questions)

Try/except, custom exceptions, assertions, debugging techniques.

Examples:

- What is the difference between `try-except` and `try-except-finally`?
- How do you raise a custom exception?
- Explain Python's `assert` statement.

📦 7. Modules, Packages & Standard Library (100 Questions)

Importing, `sys`, `os`, `datetime`, `random`, etc.

Examples:

- Difference between a module and a package?
- How do you use the `os` module to read environment variables?
- How can you generate a random password?

🧹 8. File Handling & Working with Data (100 Questions)

Reading/writing files, CSV, JSON, context managers.

Examples:

- How do you open a file in read and write mode?
- What does `with open(...)` do?
- How do you parse a JSON file?

🚀 9. Advanced Python (100 Questions)

Generators, decorators, context managers, comprehensions.

Examples:

- What is a generator and how is it different from a normal function?
- Explain list, dict, set comprehensions.
- How do decorators work internally?

⚙️ 10. Multithreading, Performance, and Misc (100 Questions)

Threads, GIL, performance tuning, Python internals.

Examples:

- What is GIL and why is it important?
- How do you use multithreading vs multiprocessing?
- How do you profile memory usage in Python?

📦 Ready to Deliver:

Would you like me to start sending the questions **topic-by-topic** in separate messages or compile all 1000 in a downloadable format (like a PDF or text file)?

Let me know your preferred format and pacing!

i am ask questions topic wise

Perfect! Let's dive in **topic by topic**, with **100 Python interview questions** per section. Here's **Topic 1: Python Basics**.

🧠 Topic 1: Python Basics – 100 Interview Questions

1. What is Python and why is it popular?
2. What are the key features of Python?
3. Is Python interpreted or compiled?
4. What is PEP 8?
5. What are Python keywords? Name a few.
6. How do you comment in Python?
7. What are the data types in Python?
8. What is the difference between mutable and immutable types?

9. What is the difference between `` `is` `` and `` `==` ``?
10. What is dynamic typing in Python?
11. What is type casting in Python?
12. What is the difference between `` `int()` `` and `` `str()` ``?
13. What is `` `None` `` in Python?
14. How do you check the type of a variable?
15. What is the use of `` `id()` `` function?
16. How does memory management work in Python?
17. What is garbage collection in Python?
18. What are local and global variables?
19. How do you define a constant in Python?
20. What is the scope of a variable?
21. Explain the `` `del` `` statement.
22. What is the output of `` `print(3/2)` `` in Python 3?
23. How do you take user input in Python?
24. How do you format strings in Python?
25. Difference between `` `%` `` formatting and `` `.format()` ``?
26. What is f-string in Python?
27. What are the logical operators in Python?
28. What is short-circuit evaluation?
29. What are bitwise operators?
30. What are identity and membership operators?
31. What is a ternary operator in Python?
32. Explain chained comparisons.
33. What are Python literals?
34. What is the difference between `` `+=` `` and `` `=+` ``?
35. Can variable names start with a number?
36. What are the naming conventions in Python?
37. What is duck typing in Python?
38. What is a docstring?
39. How do you write a multi-line string?
40. How do you access command-line arguments in Python?
41. What is the purpose of `` `if __name__ == "__main__"` ``?
42. What is a module in Python?
43. What is the difference between `` `import` `` and `` `from ... import` ``?

44. What is `__pycache__`?
45. What are magic methods?
46. What is the difference between `repr()` and `str()`?
47. What are the built-in functions in Python?
48. What is the use of `zip()`?
49. What does `enumerate()` do?
50. What does `map()` do?
51. How do you use `filter()`?
52. What is `any()` and `all()`?
53. What is the output of `bool('False')`?
54. How does Python handle boolean values?
55. What is a shallow copy vs a deep copy?
56. How do you copy a list or dictionary?
57. What is the difference between `==` and `is` for lists?
58. What is the difference between `len()` and `__len__()`?
59. What are Python escape characters?
60. How do you handle special characters in strings?
61. What are raw strings?
62. What are string methods in Python?
63. What does `strip()`, `split()`, and `join()` do?
64. How do you reverse a string?
65. How do you repeat a string in Python?
66. What are list methods in Python?
67. How do you reverse a list?
68. What does the `in` keyword do with strings and lists?
69. What is slicing? Give examples.
70. How does negative indexing work?
71. How do you merge two lists?
72. How do you remove duplicates from a list?
73. What is the difference between `append()` and `extend()`?
74. How does `pop()` work?
75. What is the use of `range()`?
76. What is the difference between `range()` and `xrange()`?
77. What are Python tuples?
78. How are tuples different from lists?

79. Can tuples have mutable elements?
80. How do you convert a list to a tuple?
81. What are sets in Python?
82. What are set operations in Python?
83. What is the difference between `` `remove()` `` and `` `discard()` `` in sets?
84. What is a frozen set?
85. What are Python dictionaries?
86. How do you loop through a dictionary?
87. What is the use of `` `get()` `` in dictionaries?
88. What are dictionary comprehensions?
89. How do you merge two dictionaries in Python 3.9+?
90. What are the different ways to create a dictionary?
91. How do you check if a key exists in a dictionary?
92. How do you sort a dictionary by key or value?
93. What is a nested dictionary?
94. How do you clear a dictionary?
95. What is the use of `` `defaultdict` ``?
96. How do you count items using `` `Counter` ``?
97. What is a named tuple?
98. How do you create constants in Python?
99. How do you find the length of an object?
100. What are some lesser-known built-in functions in Python?

✅ Let me know when you're ready for **Topic 2: Data Structures in Python (100 Questions)** or if you'd like these in a downloadable format too!

opic 2: Data Structures in Python (100 Questions)

Awesome! Here's **Topic 2: Data Structures in Python – 100 Interview Questions**.

📚 Topic 2: Data Structures in Python – 100 Interview Questions

🔢 Lists

1. What is a list in Python?
2. How do you create a list?

3. How can you add elements to a list?
4. What is the difference between `append()` and `extend()`?
5. How do you insert an element at a specific index?
6. How do you remove elements from a list?
7. What does `pop()` do?
8. What is the use of `remove()`?
9. How do you clear a list?
10. How do you reverse a list?
11. How do you sort a list?
12. What is the difference between `sort()` and `sorted()`?
13. Can a list contain elements of different types?
14. What is list slicing?
15. How do you use negative indexing in a list?
16. How do you check if an item exists in a list?
17. What is list comprehension?
18. How do you create a matrix using a list?
19. How do you flatten a nested list?
20. How do you filter a list based on a condition?

📦 Tuples

21. What is a tuple in Python?
22. How is a tuple different from a list?
23. How do you create a tuple?
24. Can a tuple contain mutable elements?
25. What happens if you try to modify a tuple?
26. How do you access tuple elements?
27. How do you unpack a tuple?
28. How do you concatenate two tuples?
29. How do you slice a tuple?
30. How do you check for an item in a tuple?

🧮 Sets

31. What is a set in Python?
32. How do you create a set?
33. Can a set contain duplicate elements?
34. What are the key properties of a set?

35. How do you add elements to a set?
36. What is the difference between `` `add()` `` and `` `update()` `` in sets?
37. How do you remove elements from a set?
38. Difference between `` `remove()` `` and `` `discard()` `` in sets?
39. What is the use of `` `pop()` `` in sets?
40. How do you perform union operation on sets?
41. How do you perform intersection?
42. What is set difference?
43. How do you check if one set is a subset of another?
44. How do you find symmetric difference between sets?
45. How do you convert a list to a set?
46. How do you remove duplicates from a list using a set?
47. What is a frozen set?
48. When would you use a set instead of a list?
49. How do sets improve performance in lookups?
50. What happens if you try to add a list to a set?

🗺️ Dictionaries

51. What is a dictionary in Python?
52. How do you create a dictionary?
53. How do you access a value in a dictionary?
54. How do you add a key-value pair to a dictionary?
55. What happens if you use an existing key to insert a value?
56. How do you delete a key-value pair?
57. What does the `` `get()` `` method do?
58. How do you loop through a dictionary?
59. How do you get keys, values, and items from a dictionary?
60. What is dictionary comprehension?
61. How do you merge two dictionaries in Python 3.9+?
62. How do you check if a key exists?
63. How do you check if a value exists in a dictionary?
64. What is the difference between `` `del` `` and `` `pop()` `` for dictionaries?
65. What does `` `popitem()` `` do?
66. How do you create nested dictionaries?
67. How do you access values in nested dictionaries?

68. What is `defaultdict` from `collections` module?
69. What is the use of `Counter` from `collections`?
70. What are `OrderedDict` and `ChainMap`?

📚 Other Built-in Structures and Operations

71. What is the difference between `len()` and `__len__()`?
72. How do you use `zip()` with data structures?
73. How do you use `enumerate()` on a list?
74. What are common list methods in Python?
75. What are common set methods?
76. What are common dictionary methods?
77. How do you convert between data types (list to tuple, etc.)?
78. How do you sort a dictionary by key or value?
79. What is the output of `list("hello")`?
80. What is the time complexity of list append operation?

🧠 Data Structure Logic-Based Questions

81. How do you remove duplicates from a list while maintaining order?
82. How do you count the frequency of elements in a list?
83. How do you check if two lists are equal?
84. How do you reverse a dictionary (swap keys and values)?
85. How do you find the most common element in a list?
86. How do you check if a list contains only unique values?
87. How do you merge a list of dictionaries?
88. How do you get the key with the maximum value in a dictionary?
89. How do you group values in a dictionary by a common key?
90. How do you sort a list of tuples by the second element?

📘 Use Cases & Real-world Scenarios

91. When should you use a list over a set?
92. When is a dictionary preferred over a list?
93. How do you use data structures to implement a stack?
94. How do you use data structures to implement a queue?
95. How do you implement a priority queue?
96. What are common pitfalls when using lists?
97. What are best practices when using dictionaries?

98. What is the difference between shallow copy and deep copy in lists?
99. What happens when you copy a mutable object in a data structure?
100. How do Python's data structures compare to those in other languages?

✅ Let me know when you're ready for **Topic 3: Control Flow & Loops (100 Questions)**.

Topic 3: Control Flow & Loops (100 Questions).

Great! Here’s **Topic 3: Control Flow & Loops – 100 Python Interview Questions**.

🔁 Topic 3: Control Flow & Loops – 100 Interview Questions

🧩 Conditional Statements

1. What are control flow statements in Python?
2. How is `if` used in Python?
3. What is the syntax of `if-else` in Python?
4. What is the use of `elif`?
5. Can `if`, `elif`, and `else` be used without braces?
6. What is indentation and why is it important in control flow?
7. What happens if you miss indentation in an `if` block?
8. Can you use nested `if` statements?
9. What is the output of `if 0:` in Python?
10. How does Python treat truthy and falsy values?

🔁 Loops – For & While

11. What is a `for` loop in Python?
12. How is a `while` loop different from a `for` loop?
13. How do you use `range()` in a `for` loop?
14. What happens if the condition in a `while` loop is never false?
15. How do you loop through a list?
16. How do you loop through a tuple?
17. How do you loop through a string?
18. How do you loop through a dictionary?
19. What does `break` do inside a loop?
20. What does `continue` do?

🔚 Loop Control Statements

21. What is the use of `pass` in loops?
22. What is the purpose of `else` in a loop?
23. What is the output of a `for` loop with an `else` block that doesn't break?
24. What is the output of a `for` loop with an `else` block that does break?
25. How is loop control flow different from other languages like Java/C++?
26. Can you have nested loops in Python?
27. How do you break from all levels of nested loops?
28. What is a flag variable? How is it used with loops?
29. How do you simulate a `do-while` loop in Python?
30. What are infinite loops and how are they handled?

🔁 Looping Techniques

31. How do you use `enumerate()` in a loop?
32. How do you use `zip()` in a loop?
33. How do you loop in reverse?
34. How do you loop over two lists simultaneously?
35. How do you loop with an index?
36. How do you loop with a step value?
37. How do you loop through only dictionary keys?
38. How do you loop through dictionary values?
39. How do you iterate over dictionary items as key-value pairs?
40. How do you skip specific iterations using `continue`?

🔄 Practical Looping Examples

41. How do you calculate the factorial of a number using a loop?
42. How do you find the sum of elements in a list?
43. How do you reverse a string using a loop?
44. How do you count vowels in a string?
45. How do you generate the Fibonacci sequence with a loop?
46. How do you print a pattern using nested loops?
47. How do you check if a number is prime using a loop?
48. How do you check if a list is sorted using a loop?
49. How do you find duplicates in a list using a loop?
50. How do you find the maximum value in a list using a loop?

🧪 Common Mistakes

51. What happens if you forget to increment in a `while` loop?
52. What does an off-by-one error mean?
53. Can you use `else` without `if` in a loop?
54. What is a logic bug commonly seen in loops?
55. Can you modify a list while looping through it?
56. How do you safely remove items from a list during iteration?
57. What happens if you try to loop over a `None` object?
58. What if the loop variable is used outside the loop?
59. What’s the impact of using `break` in nested loops?
60. Why might a loop run one extra or one less time than expected?

🧠 Advanced Looping Logic

61. How do you write a loop that stops on a specific condition?
62. How can you iterate over a 2D list?
63. How do you flatten a nested list using loops?
64. How do you use multiple conditions in a `while` loop?
65. How do you filter elements in a loop?
66. How do you find the longest word in a list?
67. How do you rotate a list using loops?
68. How do you check if a string is a palindrome using a loop?
69. How do you reverse an integer using a loop?
70. How do you calculate the average of numbers using loops?

📘 Real-world Examples

71. How would you iterate over files in a directory?
72. How do you paginate results using loops?
73. How do you simulate user input retry logic using loops?
74. How would you build a basic calculator using loops?
75. How do you match passwords with limited attempts using a loop?
76. How do you print a multiplication table using loops?
77. How would you generate prime numbers in a range using loops?
78. How would you find the first non-repeating character in a string?
79. How do you detect a cycle in user input using loops?
80. How do you build a menu-driven console app using loops?

✅ True/False & Output Questions

81. True/False: You can use `else` with `while` loops.
82. True/False: `break` skips the current iteration.
83. Output? `for i in range(3): print(i)`
84. Output? `while False: print("hi")`
85. Output? `for i in range(1, 10, 2): print(i)`
86. Output? `for i in [1, 2, 3]: i += 10; print(i)`
87. Output? `x = 5; while x > 0: x -= 1; print(x)`
88. Output? `for i in "abc": print(i)`
89. Output? `for i in range(3): pass; print(i)`
90. Output? `for i in range(5): break; else: print("done")`

🤔 Scenario-Based Questions

91. Write a loop to remove all even numbers from a list.
92. Write a loop to display a triangle pattern.
93. Write a loop to find all palindromes in a list.
94. Write a loop to sum numbers divisible by 3 and 5.
95. Write a loop to count character frequency in a string.
96. Write a loop to flatten a nested dictionary.
97. Write a loop to extract numbers from a string.
98. Write a loop to capitalize the first letter of each word in a sentence.
99. Write a loop to reverse the words in a sentence.
100. Write a loop to split a list into chunks of size 3.

✅ Ready for **Topic 4: Functions & Lambda (100 Questions)?** Let me know and I'll drop it in next!

Topic 4: Functions & Lambda (100 Questions)

Awesome! Here's your complete **Topic 4: Functions & Lambda – 100 Python Interview Questions**, tailored for interviews and mastery.

🔧 Topic 4: Functions & Lambda – 100 Interview Questions

🧱 Function Basics

1. What is a function in Python?

2. Why are functions important?
3. How do you define a function in Python?
4. How do you call a function?
5. What is the `` `return` `` statement used for?
6. What happens if you don’t use `` `return` `` in a function?
7. Can a function return multiple values?
8. What are the different types of function arguments?
9. What is a default argument?
10. What is a keyword argument?

🎯 Advanced Function Arguments

11. What are `` `*args` `` and `` `**kwargs` ``?
12. What is the difference between `` `*args` `` and `` `**kwargs` ``?
13. Can you mix positional and keyword arguments?
14. What is argument unpacking?
15. What is the order of parameters: default, keyword, *args, **kwargs?
16. What are positional-only parameters in Python 3.8+?
17. What are keyword-only arguments?
18. How do you enforce keyword-only arguments in a function?
19. What happens if you pass more arguments than expected?
20. What happens if you miss a required argument?

🧠 Function Behavior & Scope

21. What is variable scope in Python?
22. What are local, nonlocal, and global scopes?
23. What does the `` `global` `` keyword do?
24. What does the `` `nonlocal` `` keyword do?
25. What is a closure in Python?
26. How does Python handle variable shadowing in functions?
27. Can you define a function inside another function?
28. What is the lifetime of a variable defined in a function?
29. How do default values behave across function calls?
30. Are function parameters passed by value or reference?

🔁 Recursion

31. What is recursion?

32. How do you write a recursive function?
33. What are the components of a recursive function?
34. How does Python handle recursion under the hood?
35. What is a base case in recursion?
36. What is the maximum recursion depth in Python?
37. How do you avoid infinite recursion?
38. Write a recursive function to compute factorial.
39. Write a recursive function for Fibonacci numbers.
40. What are the drawbacks of recursion?

🧪 Function Properties & Use

41. What is a pure function?
42. What is a higher-order function?
43. What is a side effect in functions?
44. What are first-class functions?
45. What are anonymous functions?
46. What is a lambda function?
47. How is a lambda function different from a regular function?
48. What are limitations of lambda functions?
49. When should you use lambda functions?
50. How do you pass a lambda to `` `map()` `` or `` `filter()` ``?

🧬 Built-in Functional Programming Tools

51. What is the use of `` `map()` `` in Python?
52. How does `` `filter()` `` work?
53. What does `` `reduce()` `` do? Which module is it in?
54. What is the difference between `` `map()` `` and list comprehension?
55. How do you use `` `any()` `` and `` `all()` `` with lambda?
56. How do you sort a list of tuples using `` `lambda` ``?
57. What is the use of `` `key=` `` in the `` `sorted()` `` function?
58. How do you find the longest string in a list using `` `max()` `` and lambda?
59. How do you use `` `lambda` `` with `` `sorted()` ``?
60. What is the output of `` `list(map(lambda x: x*x, [1, 2, 3]))` ``?

📦 Function Utilities and Decorators

61. What are decorators in Python?

62. What is the `@` syntax used for?
63. How do you write a basic decorator?
64. Can you stack multiple decorators on a function?
65. What is `functools.wraps()` and why is it important?
66. How do you write a decorator that accepts arguments?
67. How do you time function execution using a decorator?
68. How do you log function calls using a decorator?
69. What is memoization? How can it be implemented using decorators?
70. What is `@staticmethod`?
71. What is `@classmethod`?
72. What is the difference between instance, static, and class methods?

🧩 Function Patterns & Real-world Examples

73. Write a function to find the largest number in a list.
74. Write a function that checks if a string is a palindrome.
75. Write a function to convert Celsius to Fahrenheit.
76. Write a function that accepts variable-length arguments and returns the sum.
77. Write a function to count vowels in a string.
78. Write a lambda to reverse a string.
79. Write a lambda to sort a list of dictionaries by a key.
80. Write a lambda that returns True for even numbers.
81. Write a lambda to get the last character of a string.
82. Write a function that uses recursion to sum a list.
83. Write a decorator to print function execution time.
84. Write a function that uses `filter()` to remove empty strings from a list.
85. Write a lambda with `reduce()` to compute the product of a list.
86. Write a function that returns another function (closure).
87. Write a function that tracks how many times it has been called.
88. Write a function that takes a function as input and applies it to a list.
89. Write a lambda function to sort a list of names by their last letter.
90. Write a recursive function to compute a power of a number.

✅ True/False & Output Questions

91. True/False: Functions in Python can return multiple values.
92. True/False: You must always use `return` in a Python function.
93. Output?

```
python

`def f(a, b=2): return a * b
print(f(3))
`
```

94. Output?

```
python

`def f(x): x = 10
y = 5
f(y)
print(y)
`
```

95. Output?

```
python

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

96. Output?

```
python

`def outer():
    x = "hello"
    def inner(): return x
    return inner
print(outer()())
`
```

97. Output?

```
python

`def f(): pass
print(f())
`
```

98. Output?

```
python

`x = lambda x: x * x
print(x(5))
`
```

99. Output?

```
python

`def f(x, y=2, z=3): return x + y + z
print(f(1, z=5))
`
```

100. Output?

```
python

`def fun(*args): return sum(args)
print(fun(1, 2, 3, 4))
`
```

✅ Let me know when you're ready for **Topic 5: Object-Oriented Programming (OOP) – 100 Questions** or if you want this in a specific format (PDF, DOCX, etc.) later!

Topic 5: Object-Oriented Programming (OOP) – 100 Questions

Absolutely! Here's your full **Topic 5: Object-Oriented Programming (OOP) – 100 Python Interview Questions**, crafted to help you **understand concepts deeply and crack interviews confidently**.

🧱 Topic 5: Object-Oriented Programming (OOP) – 100 Python Interview Questions

---

🧠 OOP Basics

1. What is Object-Oriented Programming?
2. What are the core principles of OOP?
3. What is a class in Python?
4. What is an object in Python?
5. How do you define a class in Python?
6. How do you create an object from a class?
7. What is the `__init__` method?
8. What is the role of `self` in a class?
9. What happens if you don’t use `self`?
10. Can you create multiple objects from the same class?

🏷️ Attributes & Methods

11. What are instance variables?

12. What are class variables?
13. How do you define methods in a class?
14. What is the difference between instance method and class method?
15. What is the `` `@classmethod` `` decorator?
16. What is the `` `@staticmethod` `` decorator?
17. How do you access attributes in a class?
18. What is attribute shadowing?
19. How do you modify object attributes?
20. How do you delete an attribute from an object?

🧬 Encapsulation

21. What is encapsulation in OOP?
22. How does Python implement encapsulation?
23. What are public, protected, and private attributes in Python?
24. How do you create a private attribute?
25. How do you access private variables?
26. What is name mangling in Python?
27. Can you access private attributes from outside the class?
28. How do you use getters and setters?
29. What is the `` `property()` `` function used for?
30. What are the pros and cons of using properties?

🧩 Inheritance

31. What is inheritance in Python?
32. How do you define inheritance in a class?
33. What is the syntax for single inheritance?
34. What is the `` `super()` `` function used for?
35. What is method overriding?
36. How do you call the parent method from a child class?
37. What is the difference between composition and inheritance?
38. What is multilevel inheritance?
39. What is multiple inheritance?
40. How does Python resolve method calls in multiple inheritance?

🧭 Polymorphism

41. What is polymorphism in OOP?
42. How is polymorphism implemented in Python?
43. What is method overloading?
44. Does Python support method overloading?
45. What is method overriding?
46. How is polymorphism useful in real-world coding?
47. What is duck typing in Python?
48. How is duck typing related to polymorphism?
49. What is operator overloading?
50. How do you implement operator overloading in Python?

🧱 Abstraction

51. What is abstraction in OOP?
52. How is abstraction implemented in Python?
53. What is the `` `abc` `` module?
54. What is an abstract class?
55. How do you define an abstract class?
56. What is an abstract method?
57. Can abstract classes have normal methods?
58. Can you instantiate an abstract class?
59. What is the difference between interface and abstract class in Python?
60. What is the purpose of using abstraction?

🔃 Special Methods (Magic/Dunder Methods)

61. What are dunder (magic) methods?
62. What is the use of `` `__str__()` ``?
63. What is the use of `` `__repr__()` ``?
64. What is the difference between `` `__str__()` `` and `` `__repr__()` ``?
65. What does `` `__len__()` `` do?
66. What is the purpose of `` `__eq__()` ``?
67. How does `` `__add__()` `` work?
68. What does `` `__del__()` `` do?
69. What is the use of `` `__call__()` ``?
70. What happens when you override `` `__getitem__()` ``?

🧰 Design Concepts

71. What is object composition?
72. What is delegation?
73. What is tight vs loose coupling?
74. What is cohesion in OOP?
75. What are SOLID principles?
76. How does Python support the open/closed principle?
77. What is the Liskov Substitution Principle?
78. What is dependency inversion?
79. How do you implement composition over inheritance?
80. What is mixin class in Python?

🧪 Practice & Output-based Questions

81. Output?

```
python

`class A: pass
a = A()
print(type(a))
`
```

82. Output?

```
python

`class Dog:
    def __init__(self, name): self.name = name
    def bark(self): return f"{self.name} says woof!"
d = Dog("Rex")
print(d.bark())
`
```

83. Output?

```
python

`class A:
    def __str__(self): return "A"
print(A())
`
```

84. Output?

```
python

`class Base:
    def greet(self): print("Hello")
```

```
class Derived(Base): pass
d = Derived(); d.greet()
`
```

85. Output?

```
python

`class A:
    def __init__(self): self.x = 10
a1 = A(); a2 = A()
a1.x = 20
print(a1.x, a2.x)
`
```

🧠 Conceptual & Theoretical

86. Why is `` `self` `` mandatory in class methods?
87. Can a function be both static and class method?
88. What is the role of `` `super()` `` in multiple inheritance?
89. What are metaclasses in Python?
90. What is the purpose of `` `__slots__` ``?
91. When should you use `` `__new__()` ``?
92. What is monkey patching in Python?
93. What's the difference between instance and class variables?
94. Why is encapsulation useful in large-scale software?
95. What's the best practice for method naming in OOP?

🧱 Real-world OOP Scenarios

96. Design a `` `BankAccount` `` class with deposit and withdraw methods.
97. Create a class for `` `Employee` `` and subclass it for `` `Manager` ``.
98. Implement polymorphism for `` `Shape` `` with `` `Rectangle` `` and `` `Circle` ``.
99. Create an abstract class `` `Vehicle` `` with abstract method `` `drive` ``.
100. Implement a mixin class for logging functionality.

✅ Up next: **Topic 6: Modules and Packages (100 Questions)**
Or say the word if you want it exported to **PDF, Word, CSV**, etc.

opic 6: Modules and Packages (100 Questions)

Absolutely! Here's your **Topic 6: Modules and Packages – 100 Python Interview Questions**, structured for deep understanding and strong interview prep.

📦 Topic 6: Modules and Packages – 100 Python Interview Questions

---

📁 Basics of Modules

1. What is a module in Python?
2. How do you create a module?
3. How do you import a module in Python?
4. What is the difference between `` `import module` `` and `` `from module import something` ``?
5. How do you import multiple items from a module?
6. What is `` `import *` `` used for?
7. Why is using `` `import *` `` discouraged?
8. What happens if two modules have functions with the same name?
9. How do you give an alias to a module during import?
10. How do you check the available attributes of a module?

🧠 Module Execution & Scope

11. What is `` `__name__ == "__main__"` `` used for?
12. How does Python treat the `` `__main__` `` module?
13. What is the value of `` `__name__` `` inside an imported module?
14. How do you prevent a script from executing when imported as a module?
15. Can a module import another module?
16. What is circular import?
17. How can you avoid circular imports?
18. What is `` `sys.modules` ``?
19. How does Python find a module when you import it?
20. What happens if the module is not found?

🛤️ Module Search Path & Reloading

21. What is `` `sys.path` ``?
22. How do you add a directory to `` `sys.path` ``?
23. What is the use of `` `PYTHONPATH` `` environment variable?

24. How do you reload an already imported module?
25. What is the `importlib` module used for?
26. How do you check if a module is already imported?
27. Can you delete a module from memory?
28. How do you dynamically import a module at runtime?
29. How does Python cache compiled `.pyc` files?
30. What is the purpose of the `__pycache__` folder?

🧱 Built-in Modules

31. Name some commonly used built-in modules in Python.
32. What does the `math` module provide?
33. How do you generate random numbers using the `random` module?
34. What does the `datetime` module offer?
35. How can you handle file paths with the `os` module?
36. How do you get the current working directory using `os`?
37. What is the use of `sys.argv`?
38. How can you inspect an object using the `inspect` module?
39. How can you use the `re` module for regex?
40. How do you serialize data using the `json` module?

📦 Python Packages

41. What is a package in Python?
42. What is the difference between a module and a package?
43. How do you create a package?
44. What is `__init__.py` used for?
45. Is `__init__.py` required in Python 3?
46. How do you import a module from a package?
47. How do you access a subpackage?
48. What happens if you import a package without `__init__.py`?
49. How can you create a namespace package?
50. What is the difference between a regular and a namespace package?

🚢 Third-party Packages (pip)

51. What is `pip`?

52. How do you install a package using `pip`?
53. How do you uninstall a package?
54. How do you upgrade a package with `pip`?
55. What is `requirements.txt`?
56. How do you generate a `requirements.txt` file?
57. How do you install packages from `requirements.txt`?
58. How do you list all installed packages?
59. How do you check the version of a package?
60. How do you install a specific version of a package?

📂 Organizing Code with Packages

61. How should you structure a Python project with packages?
62. How do relative imports work inside packages?
63. What are absolute imports?
64. When should you use relative vs absolute imports?
65. How do you make a directory a Python package?
66. How can you import a module from a sibling package?
67. What is `__all__` used for in `__init__.py`?
68. How do you restrict what is imported with `from package import *`?
69. What happens if two submodules have the same name?
70. How can you make a package extensible?

🔁 Custom Module & Package Design

71. How do you create a reusable custom module?
72. How can you document your module effectively?
73. How can you test your module independently?
74. How do you handle versioning in custom modules?
75. How can you distribute your custom package?
76. What is `setuptools`?
77. What is a `setup.py` file?
78. How do you create a Python package for PyPI?
79. What is `twine` used for?
80. What are Python wheels (`.whl`)?

🧪 Output-Based & Conceptual Questions

81. Output?

```
python

`import math
print(math.sqrt(16))
`
```

82. Output?

```
python

`from math import sqrt
print(sqrt(25))
`
```

83. Output?

```
python

`import random
print(random.randint(1, 5))
`
```

84. Output?

```
python

`import sys
print(sys.path)
`
```

85. Output?

```
python

`if __name__ == "__main__":
    print("Main script")
`
```

86. Output?

```
python

`from mymodule import func
func()
`
```

87. Output?

```
python

`from os import path
print(path.exists("somefile.txt"))
`
```

88. What is the output if a module is imported twice?
89. Can you use `importlib.reload()` on built-in modules?
90. What is the use of `pkgutil`?

🧰 Advanced & Real-world Scenarios

91. Create a module with a function `add(a, b)` and import it.
92. Organize a project with utility functions in a separate module.
93. Split a large script into multiple logical modules.
94. Create a package named `finance` with submodules `tax` and `salary`.
95. Use relative imports inside a package.
96. Create a `setup.py` for a CLI tool.
97. Publish a Python package on PyPI.
98. Create a namespace package with `pkg_resources`.
99. Import a module from a custom directory not in `sys.path`.
100. Load a module dynamically using `__import__()` or `importlib`.

✅ Let me know when you're ready for
➡️ **Topic 7: File Handling (100 Questions)**
or if you want me to compile the questions into a **PDF, DOCX, or spreadsheet** for easy practice and revision.

Topic 7: File Handling (100 Questions)

Here's your full **Topic 7: File Handling – 100 Python Interview Questions**, designed to **strengthen your fundamentals** and **prepare you for real-world coding interviews**.

🗂️ Topic 7: File Handling in Python – 100 Interview Questions

---

📄 Basics of File Handling

1. What is file handling in Python?
2. How do you open a file in Python?
3. What does the `open()` function return?
4. What are different modes for opening files?
5. What is the default mode in `open()`?
6. How do you read from a file?

7. How do you write to a file?
8. What is the difference between `` `'r'` `` and `` `'rb'` `` mode?
9. What is the difference between `` `'w'` `` and `` `'a'` `` mode?
10. How do you close a file in Python?

🧠 Read/Write Operations

11. How do you read the entire file content at once?
12. How do you read a file line by line?
13. What does `` `readlines()` `` do?
14. How do you read a fixed number of characters from a file?
15. What does `` `readline()` `` do?
16. How do you write a list of strings to a file?
17. How do you append data to a file?
18. What happens if you write to a file opened in `` `'r'` `` mode?
19. Can you read and write a file at the same time?
20. How do you overwrite a file?

🧾 Working with File Objects

21. What is a file object?
22. What are the key methods of a file object?
23. How do you check if a file is readable?
24. How do you check if a file is writable?
25. What does `` `file.tell()` `` do?
26. What does `` `file.seek()` `` do?
27. How can you move the file pointer to the beginning of the file?
28. How can you get the current position of the file pointer?
29. How do you flush file content manually?
30. What happens when you close a file?

🛠️ With Statement (Context Managers)

31. What is the advantage of using `` `with open()` ``?
32. How does `` `with open()` `` manage file resources?
33. Does the `` `with` `` block automatically close the file?
34. Can `` `with` `` be used for both reading and writing files?

35. What happens if an exception occurs inside a `with` block?
36. Can you nest `with` statements?
37. What is a context manager in Python?
38. Can you create a custom context manager?
39. What are `__enter__()` and `__exit__()` used for?
40. Why is context manager preferred for file handling?

📂 Working with File Paths

41. How do you work with file paths in Python?
42. What does `os.path` module provide?
43. How do you check if a file exists?
44. How do you get the absolute path of a file?
45. How do you get the directory name from a file path?
46. How do you get the file extension?
47. What does `os.path.join()` do?
48. How do you list all files in a directory?
49. What is the `glob` module used for?
50. How can you check if a file is a directory or regular file?

🔄 Working with Different File Types

51. How do you read and write a CSV file in Python?
52. What is the use of the `csv` module?
53. How do you read a CSV into a list of dictionaries?
54. How do you write a list of dictionaries to a CSV?
55. What is `DictWriter` in `csv` module?
56. How do you handle different delimiters in CSV?
57. How do you read and write JSON files?
58. How do you pretty print JSON data?
59. What does `json.load()` do?
60. What does `json.dump()` do?

🧬 Working with Binary Files

61. How do you open a binary file in Python?
62. What is the difference between text and binary files?

63. How do you write bytes to a file?
64. How do you read bytes from a file?
65. What are some examples of binary file formats?
66. Can you read and write images using file handling?
67. How do you use `struct` module for binary data?
68. What is the importance of `'rb'` and `'wb'`?
69. How do you convert a string to bytes?
70. How do you handle binary file exceptions?

🔁 Error Handling and File Exceptions

71. What are common file I/O errors?
72. How do you handle file not found errors?
73. What is `FileNotFoundError`?
74. How do you handle permission errors in file access?
75. What is `IOError`?
76. How do you catch multiple file exceptions?
77. What happens if a file is locked by another process?
78. What is the best way to handle file closing on error?
79. How do you ensure a file is always closed?
80. How do you handle reading a corrupt file?

🧪 Output-Based Questions

81. Output?

```
python

`f = open("sample.txt", "r")
print(f.read(5))
f.close()
`
```

82. Output?

```
python

`f = open("file.txt", "w")
f.write("Hello")
f.close()
`
```

83. Output?

```
python

`f = open("file.txt", "a")
f.write("World")
f.close()
`
```

84. Output?

```
python

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

85. Output?

```
python

`with open("file.txt", "w") as f:
    f.write("Line1\nLine2\n")
    f.seek(0)
    f.write("Start")
`
```

🧰 Real-World File Handling Scenarios

86. Write a program to count the number of lines in a file.
87. Write a script to copy contents from one file to another.
88. Write a function that reads a file and returns all unique words.
89. Write a script that reads a log file and filters lines with “ERROR”.
90. Create a CSV writer to log daily expenses.
91. Write a script that deletes empty lines in a file.
92. Write a script to merge two text files.
93. Create a script that compresses a file using `gzip`.
94. Write a program to backup a file before editing.
95. Create a file reader that reads only specific lines (e.g., 5-10).

📚 Conceptual Questions

96. How is memory managed during file reading in Python?
97. What are buffering and unbuffered file streams?
98. What happens if you open a file that doesn’t exist?
99. How do you truncate a file in Python?
100. How does file encoding impact file I/O?

✅ Next Up: **Topic 8: Exception Handling (100 Questions)**
Would you like a **PDF or spreadsheet** version of all topics so far for quick revision?

Topic 8: Exception Handling (100 Questions)

Absolutely! Here's your complete set of **Topic 8: Exception Handling – 100 Python Interview Questions**, tailored to build both beginner and advanced-level expertise for cracking interviews confidently.

⚠️ Topic 8: Exception Handling – 100 Python Interview Questions

---

🧱 Basics of Exception Handling

1. What is an exception in Python?
2. How is an exception different from a syntax error?
3. What are the benefits of using exception handling?
4. What is the `try` block used for?
5. What is the `except` block used for?
6. What happens when an exception occurs but is not handled?
7. What does the `else` block do in exception handling?
8. When is the `finally` block executed?
9. What is the flow of execution in `try-except-else-finally`?
10. Can `try` exist without `except`?

⚙️ Common Built-in Exceptions

11. What is `ZeroDivisionError`?
12. What is `ValueError`?
13. What is `TypeError`?
14. What is `IndexError`?
15. What is `KeyError`?
16. What is `AttributeError`?
17. What is `FileNotFoundError`?
18. What is `IOError` and how is it different from `OSError`?
19. What is `NameError`?
20. What is `ImportError` and `ModuleNotFoundError`?

🛠️ Handling Multiple Exceptions

21. How do you catch multiple exceptions in one block?
22. How do you handle different exceptions with different blocks?
23. What is the syntax for catching all exceptions?
24. Can you nest `try-except` blocks?
25. What happens if an exception occurs in the `except` block?
26. What is exception chaining?
27. How do you re-raise an exception?
28. How do you handle exception without stopping the program?
29. Can you use `try-except` inside a loop?
30. What is the impact of `break` or `return` inside `finally`?

🧬 Custom Exceptions

31. How do you define a custom exception class?
32. Why would you create a custom exception?
33. What should a custom exception class inherit from?
34. Can you include additional attributes in a custom exception?
35. How do you raise a custom exception?
36. How do you catch a custom exception?
37. What is the role of `__str__()` in custom exceptions?
38. How do you add default error messages to custom exceptions?
39. Can custom exceptions be nested?
40. What are best practices for designing custom exceptions?

🔁 Raising Exceptions

41. What is the `raise` keyword used for?
42. How do you raise a built-in exception manually?
43. Can you raise an exception conditionally?
44. Can you raise exceptions with custom messages?
45. What happens if you `raise` without any argument?
46. How do you raise exceptions inside functions?
47. Can you use `raise` inside a `finally` block?
48. How do you log raised exceptions?
49. What’s the difference between `raise Exception()` and `raise Exception`?

50. What is exception propagation?

🔍 Exception Object & Logging

51. What is an exception object?
52. How do you capture an exception object using `` `as` ``?
53. How do you print the error message from an exception?
54. How do you print the full traceback?
55. How do you log exceptions using the `` `logging` `` module?
56. How do you suppress exceptions temporarily?
57. How do you convert an exception to a string?
58. Can you format an exception message?
59. What is `` `traceback.print_exc()` ``?
60. How do you write exceptions to a log file?

🧪 Output-Based Exception Questions

61. Output?

```
python

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

62. Output?

```
python

`try:
    x = int("a")
except ValueError:
    print("Invalid integer")
`
```

63. Output?

```
python

`try:
    print("Hello")
finally:
    print("Bye")
`
```

64. Output?

```python
`try:
    a = [1, 2]
    print(a[5])
except IndexError:
    print("Index error caught")
`
```

65. Output?

```python
`try:
    print(10)
except:
    print("Error")
else:
    print("No error")
`
```

🧰 Real-World Use Cases

66. Handle a file not found error when opening a file.
67. Validate user input using exceptions.
68. Retry logic using `try-except`.
69. Fallback to default configuration if config file fails.
70. Handling missing dictionary keys gracefully.
71. Catching HTTP exceptions in API requests.
72. Handling database connection exceptions.
73. Prevent crashing of a CLI program using `try-except`.
74. Graceful termination of a program on error.
75. Logging and continuing with the next loop iteration after an error.

💡 Conceptual Deep Dives

76. Difference between syntax error and runtime error?
77. Can a `finally` block override a return statement?
78. What happens if both `except` and `finally` have `return`?
79. Why should you avoid bare `except:` blocks?
80. Difference between `except Exception:` and `except:`?
81. Can you suppress multiple exceptions in a single block?
82. How do exceptions affect performance?
83. Is `try-except` faster than checking conditions first?
84. When should you avoid using exceptions?

85. What is EAFP vs LBYL in Python?

🛑 Exception Best Practices

86. What are best practices when handling exceptions?
87. Why is it important to handle specific exceptions?
88. Should exceptions be logged or printed?
89. How do you ensure cleanup actions are performed?
90. How can exception handling improve user experience?
91. How should you handle sensitive errors (e.g., passwords)?
92. How do you avoid exception swallowing?
93. Should you use exceptions for flow control?
94. How do you prevent resource leaks using exception handling?
95. How do you create a global error handler?

🧠 Miscellaneous & Advanced

96. Can `` `try-except` `` blocks be used with coroutines?
97. How do exceptions behave in multithreading?
98. What is `` `contextlib.suppress()` ``?
99. How do you catch exceptions in list comprehensions?
100. Can exceptions be inherited and reused across projects?

✅ Ready for: ➡️ **Topic 9: Iterators and Generators (100 Questions)?**
Want all topics so far compiled into a **PDF, DOCX, or spreadsheet** for revision and mock interviews?

Topic 9: Iterators and Generators (100 Questions)?

Absolutely! Here's your complete **Topic 9: Iterators and Generators – 100 Python Interview Questions** to help you understand and crack interview questions in **one shot**.

🔁 Topic 9: Iterators and Generators – 100 Python Interview Questions

---

🔄 Basics of Iterators

1. What is an iterator in Python?

2. What is the difference between an iterable and an iterator?
3. What makes an object iterable?
4. What methods must a class implement to become an iterator?
5. What is the use of the `` `__iter__()` `` method?
6. What is the use of the `` `__next__()` `` method?
7. How do you create an iterator from a list?
8. What happens when an iterator is exhausted?
9. What exception is raised when `` `next()` `` reaches the end?
10. How does `` `for` `` loop internally use iterators?

🧪 Iterator Methods & Output Questions

11. Output?

```
python

`it = iter([1, 2, 3])
print(next(it))
`
```

12. Output?

```
python

`a = "hello"
it = iter(a)
print(next(it))
`
```

13. Output?

```
python

`for i in iter([4, 5]):
    print(i)
`
```

14. Output?

```
python

`it = iter([1])
next(it)
next(it)
`
```

15. Output?

```
python

`x = iter(range(2))
print(list(x))
print(list(x))
`
```

🧱 Custom Iterators

16. How do you define a custom iterator class?
17. Implement a class that returns numbers 1 to 5 using iterator protocol.
18. How can you make an object both iterable and an iterator?
19. How does a class differ from a generator in terms of state?
20. Can two iterators iterate independently over the same list?

⚡ Generators – Basics

21. What is a generator in Python?
22. What is the difference between a generator and an iterator?
23. How do you define a generator function?
24. What is the purpose of the `yield` keyword?
25. How does `yield` differ from `return`?
26. What is the state of a generator function between yields?
27. How do you iterate through a generator?
28. Can a generator be used with `next()`?
29. Can you create generators with expressions?
30. How does `yield` retain function state?

🔬 Generator Expressions

31. What is a generator expression?
32. How is a generator expression different from a list comprehension?
33. When should you use a generator expression?
34. Create a generator expression for even numbers between 1 and 10.
35. How do you convert a generator expression to a list?

🔍 Inspecting Generators

36. How do you check if an object is a generator?
37. What module can help identify generator types?

38. Can you rewind a generator?
39. What happens if you exhaust a generator and call it again?
40. Can generators be reused?

🛠️ Generator Functions – Intermediate

41. Can you use loops in a generator function?
42. Can you yield multiple times from a generator?
43. Can a generator yield different types of values?
44. Can generators take arguments?
45. Can a generator return a value?

⚙️ Generator Control Flow

46. What is the behavior of `` `yield` `` inside a loop?
47. Can `` `yield` `` be used inside conditionals?
48. What happens if a generator is closed prematurely?
49. What does `` `StopIteration` `` mean in generators?
50. Can `` `finally` `` be used in generators?

📦 Use Cases of Generators

51. What are the benefits of using generators?
52. Why are generators memory efficient?
53. Where are generators typically used in real-world apps?
54. How do generators improve performance for large data?
55. Give an example of a generator used in reading large files.

🔄 Generator Use Cases – Code

56. Write a generator to yield squares of numbers up to N.
57. Write a generator to yield Fibonacci numbers.
58. Write a generator that reads a file line-by-line.
59. Write a generator that yields prime numbers up to N.
60. Write a generator to paginate a large list.

⛓️ Chaining & Nesting

61. How do you chain multiple generators together?

62. Can a generator call another generator?
63. What is `yield from` in Python?
64. What is the difference between `yield` and `yield from`?
65. How does `yield from` simplify delegation?

🧠 Conceptual & Deep-Dive

66. Compare iterators and generators.
67. When should you use generators over iterators?
68. Are all generators iterators?
69. Are all iterators generators?
70. How does Python optimize generator execution?

🛡️ Error Handling in Generators

71. Can you raise exceptions in a generator?
72. How do you handle exceptions inside generator functions?
73. What is the behavior of `try/except` around `yield`?
74. How does `throw()` work on generators?
75. What is the purpose of `close()` method on a generator?

🧵 Advanced Topics

76. What is generator delegation?
77. How do async generators differ from regular ones?
78. What is the use of `asend()`, `athrow()` in async generators?
79. Can generators be used in coroutines?
80. What’s the memory difference between list comprehension and generator expression?

🧪 Output-Based (Generators)

81. Output?

```
python

`def gen():
    yield 1
    yield 2
g = gen()
print(next(g))
`
```

82. Output?

```
python

`def g():
    yield from range(3)
print(list(g()))
`
```

83. Output?

```
python

`def f():
    for i in range(2):
        yield i
x = f()
print(next(x))
print(next(x))
`
```

84. Output?

```
python

`def g():
    yield "hello"
    return "done"
print(list(g()))
`
```

85. Output?

```
python

`g = (x * 2 for x in range(3))
print(next(g), next(g))
`
```

🎯 Real-World Generator Design Questions

86. Implement a generator that yields items from a huge CSV file.
87. Design a generator to stream API results one by one.
88. Build a generator to chunk large datasets.
89. Create a generator that returns all dates in a year.
90. Implement a generator to yield user input until “exit”.

✅ True/False & MCQ Style

91. Generators can be reused once exhausted. (True/False)
92. Generators can be paused and resumed. (True/False)

93. List comprehensions use less memory than generators. (True/False)
94. Generators always raise `` `StopIteration` `` when done. (True/False)
95. Generator expressions are enclosed in square brackets. (True/False)
96. What does `` `yield` `` return? a) None
    b) A generator
    c) A value
    d) An iterator
97. Which of the following is true about `` `yield from` ``? a) Replaces multiple `` `yield` `` calls
    b) Delegates another generator
    c) Works only in Python 2
    d) Stops on first yield
98. Generators maintain state across: a) calls
    b) classes
    c) loops
    d) None
99. Which one creates a generator?

```
python

`(x for x in range(3))
[x for x in range(3)]
{x for x in range(3)}
`
```

100. What will `` `list((x for x in [1,2]))` `` return?

✅ You're now fully covered on Iterators & Generators!
Next up: **Topic 10: Decorators & Closures (100 Questions)** – Ready to continue?

Also, want a combined **PDF or Excel** of all topics so far?

Topic 10: Decorators & Closures (100 Questions)

Awesome! Here's your **Topic 10: Decorators & Closures – 100 Python Interview Questions**, crafted to give you deep understanding and full interview prep 🔥

🎯 Topic 10: Decorators & Closures – 100 Python Interview Questions

---

🧱 Basics of Functions & Closures

1. What is a closure in Python?
2. How is a closure different from a normal function?

3. What are free variables in closures?
4. What is a nested function?
5. How does a closure retain the outer function’s state?
6. Give an example of a closure in Python.
7. What is the use of closures?
8. When should you use closures?
9. What are the key properties of closures?
10. How do closures help in data hiding?

🔁 Output-Based: Closures

11. Output?

```
python

`def outer():
    x = 10
    def inner():
        print(x)
    return inner
outer()()
`
```

12. Output?

```
python

`def outer():
    msg = "Hi"
    def inner():
        return msg
    return inner
f = outer()
print(f())
`
```

13. Output?

```
python

`def counter():
    count = 0
    def increment():
        nonlocal count
        count += 1
        return count
    return increment
c = counter()
print(c(), c())
`
```

14. Output?

```
python

`def outer(a):
    def inner(b):
        return a + b
    return inner
add5 = outer(5)
print(add5(10))
`
```

15. Output?

```
python

`def func():
    x = "Python"
    def inner():
        print(x)
    return inner
x = "Java"
func()()
`
```

⚙️ Basics of Decorators

16. What is a decorator in Python?
17. What is the purpose of decorators?
18. What are the syntax rules for writing a decorator?
19. What does `` `@decorator` `` mean in Python?
20. What are some real-world use cases for decorators?

🧪 Writing Decorators

21. How do you write a simple decorator?
22. How do decorators work with arguments?
23. Can decorators modify return values?
24. How do decorators help in logging?
25. Write a decorator to time a function's execution.

🧠 Understanding Decorator Flow

26. How does the call stack work with decorators?
27. What does wrapping a function mean?
28. What is `` `*args` `` and `` `**kwargs` `` in decorators?
29. Can you use multiple decorators on one function?
30. What is the order of execution when using stacked decorators?

🧵 Stateful Decorators

31. How do you maintain state in a decorator?
32. Can decorators count function calls?
33. How do you create a caching/memoization decorator?
34. Write a decorator that limits function calls.
35. How do you decorate a class method?

🛠️ Practical Decorator Use Cases

36. Use decorators to log function inputs.
37. Use decorators to handle exceptions in functions.
38. Use decorators to check user permissions.
39. Use a decorator to retry failed functions.
40. Use decorators to validate arguments.

🔁 Output-Based: Decorators

41. Output?

```
python

`def deco(fn):
    def wrap():
        print("Before")
        fn()
        print("After")
    return wrap

@deco
def say_hi():
    print("Hi")
say_hi()
`
```

42. Output?

```
python

`def tag(fn):
    def wrapper():
        return "<b>" + fn() + "</b>"
    return wrapper

@tag
def get_text():
    return "Hello"
print(get_text())
`
```

43. Output?

```
python

`def outer(fn):
    def inner(*args):
        return fn(*args) * 2
    return inner

@outer
def mult(x):
    return x
print(mult(5))
`
```

44. Output?

```
python

`def wrap(fn):
    def inner(*args, **kwargs):
        print("wrapped")
        return fn(*args, **kwargs)
    return inner

@wrap
def greet(name):
    return f"Hi {name}"
print(greet("Tom"))
`
```

45. Output?

```
python

`def simple(fn):
    return lambda: fn().upper()

@simple
def hello():
    return "hello"
print(hello())
`
```

📦 Advanced Decorators

46. What is `` `functools.wraps` ``?
47. Why should you use `` `@wraps` `` in decorators?
48. What does `` `functools.lru_cache` `` do?
49. How do decorators interact with function metadata?
50. What are class-based decorators?

🔂 Decorators with Arguments

51. How do you pass arguments to decorators?
52. Write a decorator that takes a prefix and adds it to output.

53. How to write a parameterized decorator?
54. What is the three-layer function structure in argumented decorators?
55. Can a decorator accept keyword arguments?

🏗️ Decorators vs Closures

56. How is a decorator a type of closure?
57. How do decorators leverage closure concepts?
58. Which one is more flexible: decorators or closures?
59. Can closures be used as decorators?
60. Compare use cases: closures vs decorators.

🧠 Conceptual & Deep-Dive

61. Why are decorators considered higher-order functions?
62. What is a higher-order function?
63. Can decorators be nested?
64. Can decorators return other decorators?
65. Explain function wrapping in decorators with an example.

🧪 More Output-Based

66. Output?

```
python

`def dec(fn):
    def wrapper():
        print("Start")
        fn()
        print("End")
    return wrapper

def test():
    print("Test")
new = dec(test)
new()
`
```

67. Output?

```
python

`def count_calls(fn):
    count = 0
    def wrapper():
        nonlocal count
        count += 1
        print(f"Call {count}")
        fn()
    return wrapper

@count_calls
def hello():
    print("Hi")
hello(); hello()
`
```

68. Output?

```
python

`def power(n):
    def decorator(fn):
        def wrapper(x):
            return fn(x) ** n
        return wrapper
    return decorator

@power(3)
def num(x):
    return x
print(num(2))
`
```

69. Output?

```
python

`from functools import wraps

def deco(fn):
    @wraps(fn)
    def inner():
        return fn()
    return inner

@deco
def fun():
    "does nothing"
    pass

print(fun.__name__, fun.__doc__)
`
```

70. Output?

```
python

`def d1(fn): return lambda: fn() + "!"
def d2(fn): return lambda: fn() + "?"

@d2
@d1
def f(): return "Hi"

print(f())
`
```

🧪 MCQs / True-False Style

71. A closure captures values not variables. (T/F)

72. You must call a decorator function manually. (T/F)
73. Decorators can be chained. (T/F)
74. Closures can return a function. (T/F)
75. The outer function in a closure is destroyed after returning. (T/F)
76. What does `` `@` `` do in Python? a) Calls a function
    b) Creates a loop
    c) Applies a decorator
    d) Defines a class
77. Closures are used when: a) You need recursion
    b) You want to avoid global variables
    c) You use loops
    d) You use classes
78. Which of the following is not true about decorators? a) They wrap a function
    b) They must return another function
    c) They change function names
    d) They modify function behavior
79. Decorators must return: a) Any value
    b) A string
    c) A function
    d) None
80. In which order are decorators applied? a) Top to bottom
    b) Bottom to top
    c) Random
    d) Depends on indentation

🧩 Real-World Use Cases

81. Decorator to require login in a Flask app.
82. Decorator to limit API rate.
83. Decorator to check admin access.
84. Use a decorator for benchmarking.
85. Decorator to retry a function on failure.

🛠️ Build-Your-Own Tasks

86. Write a decorator that prints the time taken by a function.
87. Write a decorator that logs arguments and return values.
88. Write a decorator that disables a function unless a flag is set.
89. Write a decorator that prevents a function from running twice.
90. Write a decorator that caches the results of a function.

🧠 Advanced Use Cases & Questions

91. Can decorators modify arguments?
92. Can a decorator be applied dynamically?
93. Can you store decorated functions in a list?
94. Can you create decorators for generators?
95. Can a decorator wrap an async function?

✅ Final 5 Challenge Questions

96. Write a decorator that makes any function run twice.
97. Write a decorator that logs function calls to a file.
98. Write a closure that remembers the last 3 inputs.
99. Create a decorator that tracks execution time of recursive functions.
100. Create a closure-based counter for a click event handler.

🔥 That's 100 full-on **Decorator & Closure** interview questions!

Next up:
➡️ **Topic 11: Python Built-in Functions & Scope**
Or want the compiled PDF/DOCX/Excel so far for your prep notes?

11: Python Built-in Functions & Scope

Absolutely! Here's your complete set of **Topic 11: Python Built-in Functions & Scope – 100 Interview Questions**, perfect for mastering the fundamentals and clearing any Python interview in **one shot** 💥

📘 Topic 11: Python Built-in Functions & Scope – 100 Questions

---

🔤 Built-in Functions – Basics

1. What are built-in functions in Python?
2. How many built-in functions are there in Python 3?
3. Where are built-in functions stored in Python?
4. What's the difference between `` `abs()` `` and `` `round()` ``?
5. What does `` `dir()` `` do?
6. How does `` `type()` `` work?
7. What does `` `id()` `` return?

8. What does `len()` return for different types?
9. How does `input()` work in Python 3?
10. What is the difference between `eval()` and `exec()`?

🧪 Output-Based: Built-in Functions

11. Output?

```
python

`print(type([]))
`
```

12. Output?

```
python

`print(len("hello"))
`
```

13. Output?

```
python

`print(abs(-7.5))
`
```

14. Output?

```
python

`print(round(3.145, 2))
`
```

15. Output?

```
python

`print(bool(0), bool(1), bool([]))
`
```

🧮 Numeric Built-in Functions

16. What does `pow(x, y)` return?
17. How is `divmod(a, b)` used?
18. What does `sum()` do with a list?
19. What is the output of `min()` and `max()`?
20. How does `round()` handle midpoint values?

🔠 Type Conversion Functions

21. What is the use of `int()`, `float()`, `str()`?
22. How do you convert binary to decimal in Python?
23. How does `ord()` and `chr()` work?
24. What is the purpose of `hex()`, `oct()`, `bin()`?
25. What does `bool()` return for empty objects?

🧠 Advanced Built-in Functions

26. What does `enumerate()` return?
27. What is `zip()` used for?
28. What's the difference between `map()` and `filter()`?
29. How does `all()` and `any()` work?
30. What does `reversed()` return?

🔂 Iteration Utilities

31. What type does `range()` return?
32. How does `next()` work?
33. What happens when you call `iter()` on a list?
34. What does `sorted()` return?
35. How do you sort with a custom key using `sorted()`?

📦 Collection-Related Built-ins

36. What does `list()` do?
37. What's the use of `set()`?
38. How does `tuple()` work?
39. What is the output of `dict()` with a list of tuples?
40. How do you use `frozenset()`?

🎯 Functional Built-ins

41. What is `lambda` in Python?
42. How do you use `map()` with `lambda`?
43. Give an example using `filter()` with a condition.
44. How does `reduce()` differ from `map()`?

45. What’s the purpose of `` `zip()` ``?

🧵 Scope – Basics

46. What is a scope in Python?
47. What are the different types of scopes?
48. What is LEGB rule?
49. What is a local scope?
50. What is a global scope?

🌍 Global and Local Scope Questions

51. What happens if a variable is defined inside a function?
52. How to modify a global variable inside a function?
53. What does `` `global` `` keyword do?
54. What is the use of `` `nonlocal` `` keyword?
55. Can you access local variable outside a function?

🔁 Output-Based: Scope

56. Output?

```
python

`x = 10
def f():
    print(x)
f()
`
```

57. Output?

```
python

`x = 5
def f():
    x = 3
    print(x)
f()
print(x)
`
```

58. Output?

```
python

`x = 1
def f():
    global x
    x = 2
f()
print(x)
`
```

59. Output?

```
python

`def outer():
    x = "outer"
    def inner():
        nonlocal x
        x = "inner"
    inner()
    return x
print(outer())
`
```

60. Output?

```
python

`def f():
    a = 10
f()
print(a)
`
```

🧠 Conceptual: Scope

61. What happens if you try to assign to a global variable inside a function without declaring it `` `global` ``?
62. Explain the difference between `` `nonlocal` `` and `` `global` ``.
63. Can you use `` `nonlocal` `` outside nested functions?
64. What is the scope of a variable declared in a loop?
65. How does Python decide which variable to use in an expression?

🧠 Debugging Built-ins

66. What does `` `help()` `` do?
67. How do you get documentation using `` `__doc__` ``?
68. What does `` `callable()` `` check?
69. How does `` `isinstance()` `` work?
70. What is the difference between `` `isinstance()` `` and `` `type()` ``?

📜 File/Input/Output Built-ins

71. What is `` `open()` `` used for?

72. What are modes in `` `open()` `` function?
73. How do you read and write files using built-ins?
74. How does `` `input()` `` work in Python 3?
75. What is the difference between `` `readline()` `` and `` `readlines()` ``?

🧪 Output-Based: Built-in Fun

76. Output?

```
python

`print(ord('A'))
`
```

77. Output?

```
python

`print(chr(65))
`
```

78. Output?

```
python

`print(list(zip([1,2],[3,4])))
`
```

79. Output?

```
python

`print(any([0, 0, 1]))
`
```

80. Output?

```
python

`print(all([1, True, 5]))
`
```

🧩 Built-ins With Functions

81. How does `` `id()` `` help in debugging?
82. How does `` `globals()` `` differ from `` `locals()` ``?
83. What does `` `vars()` `` return?
84. What does `` `hash()` `` do?

85. Can `hash()` be used with custom objects?

🔍 Introspection Functions

86. What is the purpose of `getattr()`?
87. What does `setattr()` do?
88. How does `hasattr()` work?
89. What's the use of `delattr()`?
90. How can you dynamically invoke a method using a string?

✅ True/False & MCQs

91. `id()` returns memory address of an object. (T/F)
92. `eval()` is safer than `exec()`. (T/F)
93. Variables inside loops have global scope. (T/F)
94. `isinstance([], list)` returns True. (T/F)
95. `range()` creates a list. (T/F)
96. Which of the following are built-in? a) `print`
    b) `append`
    c) `list`
    d) `split`
97. What does `nonlocal` affect? a) Global vars
    b) Enclosing scope vars
    c) Local vars
    d) All
98. Which of these is not a built-in? a) `sum`
    b) `reduce`
    c) `eval`
    d) `filter`
99. `globals()` returns: a) local vars
    b) global vars dict
    c) function scope
    d) none
100. `callable(obj)` returns True if: a) obj is a string
    b) obj is a number
    c) obj is a function
    d) obj is iterable

💥 You've now mastered **Python Built-in Functions & Scope!**

Next topic:
➡️ **Topic 12: Python Comprehensions (List, Dict, Set, Generator) – 100 Questions**

Want a single **PDF/Excel/DOC** with all topics combined so far for revision?

Topic 12: Python Comprehensions (List, Dict, Set, Generator) – 100 Questions

Absolutely! Here's your full **Topic 12: Python Comprehensions (List, Dict, Set, Generator) – 100 Interview Questions**, packed to help you **crack interviews in one shot** ⚡

🧠 Topic 12: Python Comprehensions – 100 Interview Questions

🔤 Basics of Comprehensions

1. What are comprehensions in Python?
2. Why are comprehensions used?
3. List the different types of comprehensions in Python.
4. What are the advantages of using comprehensions?
5. Are comprehensions faster than traditional loops?

🧾 List Comprehensions – Basics

6. What is list comprehension?
7. Syntax of a basic list comprehension?
8. Convert this loop to a list comprehension:

```
python

`result = []
for i in range(5):
    result.append(i * i)
`
```

9. Can list comprehensions be nested?
10. What is the output of `` `[x for x in range(5)]` ``?

🧪 Output-Based: List Comprehensions

11. Output?

```
python

`[x for x in range(3)]
`
```

12. Output?

```
python

`[x**2 for x in range(4)]
`
```

13. Output?

```
python

`[x for x in range(5) if x % 2 == 0]
`
```

14. Output?

```
python

`[x.upper() for x in "abc"]
`
```

15. Output?

```
python

`[x for x in "hello" if x not in "aeiou"]
`
```

🔁 List Comprehension with Conditions

16. How do you use `if` in list comprehensions?
17. Can you use `if-else` inside list comprehensions?
18. Output?

```
python

`["even" if x % 2 == 0 else "odd" for x in range(3)]
`
```

19. Convert this loop to list comprehension:

```
python

`result = []
for x in range(5):
    if x % 2 == 0:
        result.append(x)
`
```

20. Create a list comprehension to return squares of even numbers from 0–9.

🧩 Nested List Comprehensions

21. What is a nested list comprehension?
22. Example of a 2D list comprehension?
23. Flatten a 2D list using comprehension.
24. Output?

```
python

`[[i*j for j in range(3)] for i in range(2)]
`
```

25. Create a matrix of size 3x3 using list comprehension.

📘 Dictionary Comprehensions

26. What is dictionary comprehension?
27. Syntax of a dict comprehension?
28. Convert this to dict comprehension:

```
python

`result = {}
for i in range(3):
    result[i] = i**2
`
```

29. Create a dict where keys are letters and values are ASCII.
30. Use dict comprehension to invert a dictionary.

🧪 Output-Based: Dict Comprehensions

31. Output?

```
python

`{x: x**2 for x in range(3)}
`
```

32. Output?

```
python

`{k: v for k, v in zip("abc", [1, 2, 3])}
`
```

33. Output?

```
python

`{char: char.upper() for char in "abc"}
`
```

34. Output?

```
python

`{x: "even" if x % 2 == 0 else "odd" for x in range(4)}
`
```

35. Output?

```
python

`{x: y for x, y in enumerate("xyz")}
`
```

📗 Set Comprehensions

36. What is a set comprehension?
37. Syntax of set comprehension?
38. Output?

```
python

`{x for x in [1,2,2,3]}
`
```

39. Remove duplicates from a list using set comprehension.
40. Use set comprehension to get vowels from a string.

🧪 Output-Based: Set Comprehensions

41. Output?

```
python

`{x*2 for x in [1, 2, 3]}
`
```

42. Output?

```
python

`{c.upper() for c in "hello"}
`
```

43. Output?

```
python

`{x for x in range(5) if x % 2 == 0}
`
```

44. Output?

```
python

`{x.lower() for x in "AaBb"}
`
```

45. Output?

```
python

`{ord(c) for c in "abc"}
`
```

🔁 Generator Expressions

46. What is a generator expression?

47. Syntax of generator expressions?

48. Difference between list comprehension and generator expression?

49. Convert a list comprehension to generator expression.

50. Output?

```
python

`g = (x**2 for x in range(3))
print(next(g))
`
```

🧪 Output-Based: Generator Expressions

51. Output?

```
python

`g = (x for x in "abc")
print(list(g))
`
```

52. Output?

```
python

`g = (x+1 for x in range(2))
for val in g:
    print(val)
`
```

53. Output?

```
python

`g = (x for x in range(100) if x % 3 == 0)
print(sum(g))
`
```

54. Output?

```
python

`print(sum(x for x in range(5)))
`
```

55. Output?

```
python

`g = (i for i in range(2))
print(next(g), next(g))
`
```

🧠 Comparison & Use Cases

56. When to use list vs generator comprehension?
57. When to use dict vs set comprehension?
58. Which is more memory efficient: list or generator comprehension?
59. Can comprehensions be used in function arguments?
60. Which comprehension types are unordered?

🔍 Real-World Examples

61. Use comprehension to read numbers from a string and convert to int.
62. Use comprehension to filter non-alphabetic characters.
63. Use comprehension to flatten a matrix.
64. Create a word frequency dict using dict comprehension.
65. Create a set of unique words from a paragraph.

🧠 Conceptual Questions

66. Are comprehensions more readable than loops?
67. Can all comprehensions be nested?
68. Do comprehensions always return a new object?
69. Can you break a comprehension like a loop?
70. Why are generator comprehensions lazy?

🧠 Comprehension Challenges

71. Create a list of prime numbers from 1 to 50.
72. Find squares of even numbers between 1–20.
73. Create a dict of even numbers and their squares.
74. Extract digits from a string using list comprehension.
75. Build a generator that yields cubes of numbers.

🔄 Nested & Advanced

76. Output?

```python
`[x + y for x in "ab" for y in "12"]
`
```

77. Output?

```python
`[x for x in range(3) for y in range(2)]
`
```

78. Output?

```python
`[(i, j) for i in range(2) for j in range(2)]
`
```

79. Output?

```python
`[(x, x**2) for x in range(3) if x % 2 == 1]
`
```

80. Output?

```
python

`{(i, j): i * j for i in range(2) for j in range(2)}
`
```

✅ True/False & MCQ

81. Comprehensions are only for lists. (T/F)

82. Generator expressions use less memory than list comprehensions. (T/F)

83. `` `set()` `` comprehension can contain duplicates. (T/F)

84. Dictionary comprehension requires `` `key: value` ``. (T/F)

85. Nested comprehensions are not allowed. (T/F)

86. What is returned by a generator expression? a) List
    b) Tuple
    c) Generator object
    d) Iterator

87. Which is valid? a) `` `[x for x in range(5)]` ``
    b) `` `{x for x in range(5)}` ``
    c) `` `{x: x*2 for x in range(5)}` ``
    d) All

88. Comprehensions are: a) Statements
    b) Expressions
    c) Loops
    d) Functions

89. Which comprehension is unordered? a) List
    b) Set
    c) Dict
    d) Both b & c

90. `` `g = (x for x in range(3))` ``; type(g) = ? a) generator
    b) list
    c) function
    d) object

🧩 Final Challenges (91–100)

91. Write a list comprehension to return all leap years between 2000 and 2025.

92. Write a set comprehension to get all lowercase letters from a string.

93. Use a generator to yield Fibonacci numbers.

94. Use nested list comprehension to make identity matrix.

95. Dict comprehension to map char → index from a string.

96. Flatten a list of lists using one-liner comprehension.

97. Reverse words in a string using list comprehension.
98. Write a list comprehension with multiple `if` filters.
99. Create a dict of odd numbers and their cube values.
100. Create a set comprehension to extract digits from a sentence.

🔥 That’s 100 solid **Python Comprehension Questions** covering **list, set, dict, and generators**.

Next up: ➡️ **Topic 13: Python Regular Expressions (re module)**
Or want all 1200 questions so far in a **PDF/Excel/DOC** to keep for revision?

Topic 13: Python Regular Expressions (re module)

Perfect! Here's your complete **Topic 13: Python Regular Expressions (re module) – 100 Interview Questions** to master pattern matching and clear interviews **like a pro** 💪

🧾 Topic 13: Python Regular Expressions (re module) – 100 Interview Questions

---

🔤 Basics of Regular Expressions

1. What are regular expressions in Python?
2. Why are regular expressions useful?
3. Which Python module supports regular expressions?
4. What is the syntax to import the `re` module?
5. What is a raw string and why is it used with regex?

📚 Common Regex Functions

6. What does `re.match()` do?
7. What does `re.search()` do?
8. What is the difference between `match()` and `search()`?
9. What does `re.findall()` return?
10. What is the purpose of `re.finditer()`?

📐 Regex Basics

11. What does `.` (dot) match in regex?
12. What does `^` signify in regex?

13. What does `` `$` `` signify?
14. What does `` `*` `` mean in regex?
15. What does `` `+` `` mean?

🔣 Regex Quantifiers

16. What does `` `?` `` do in regex?
17. What does `` `{n}` `` mean?
18. What does `` `{n,}` `` mean?
19. What does `` `{n,m}` `` mean?
20. What is the difference between greedy and non-greedy quantifiers?

🎯 Regex Character Classes

21. What does `` `\d` `` match?
22. What does `` `\D` `` match?
23. What does `` `\w` `` match?
24. What does `` `\W` `` match?
25. What does `` `\s` `` and `` `\S` `` mean?

🔁 Sets and Ranges

26. What does `` `[abc]` `` match?
27. What does `` `[a-z]` `` mean?
28. What does `` `[^abc]` `` do?
29. How to match digits 1 through 5?
30. What does `` `[^0-9]` `` match?

🧪 Output-Based: Matching

31. Output?

```
python

`re.match("a", "abc")
`
```

32. Output?

```
python

`re.search("a", "bcabc")
`
```

33. Output?

```
python

`re.findall(r"\d", "abc123")
`
```

34. Output?

```
python

`re.findall(r"[aeiou]", "hello world")
`
```

35. Output?

```
python

`re.match(r"\w+", "123abc!")
`
```

🛠️ Pattern Compilation

36. What does `` `re.compile()` `` do?
37. Why use compiled patterns?
38. How to use `` `pattern.match()` ``?
39. How to use `` `pattern.findall()` ``?
40. Can you reuse compiled regex patterns?

🔁 Regex Substitution

41. What does `` `re.sub()` `` do?
42. Syntax of `` `re.sub()` ``?
43. How to replace digits with `` `#` ``?
44. Replace all whitespace with `` `-` ``.
45. Replace multiple spaces with a single space.

🧩 Groups and Capturing

46. What are groups in regex?

47. How to define groups in Python regex?
48. How to retrieve captured groups?
49. What does `group(0)` return?
50. What’s the difference between `group(1)` and `groups()`?

🧪 Output-Based: Groups

51. Output?

```
python

`m = re.match(r"(\d+)-(\d+)", "123-456")
print(m.groups())
`
```

52. Output?

```
python

`m = re.search(r"Name: (\w+)", "Name: John")
print(m.group(1))
`
```

53. Output?

```
python

`re.findall(r"(\w+)@(\w+)", "a@b x@y")
`
```

54. Output?

```
python

`re.match(r"(\w+)\s(\w+)", "John Doe").group(2)
`
```

55. Output?

```
python

`re.search(r"(ab)+", "ababab").group()
`
```

🧠 Special Sequences

56. What is the use of `\A` in regex?
57. What is the use of `\Z`?
58. What does `\b` do?

59. What does `\B` mean?

60. Give an example using word boundaries.

🎛️ Regex Flags

61. What are regex flags?

62. What does `re.IGNORECASE` or `re.I` do?

63. What is the purpose of `re.MULTILINE`?

64. How to use `re.DOTALL`?

65. How to combine multiple flags?

🧪 Output-Based: Flags

66. Output?

```
python

`re.findall("a", "AbcA", re.I)
`
```

67. Output?

```
python

`re.search("^abc", "abc\ndef", re.M)
`
```

68. Output?

```
python

`re.search(".", "\n", re.S)
`
```

69. Output?

```
python

`re.findall(r"(?i)hello", "Hello hEllo")
`
```

70. Output?

```
python

`re.findall(r"(?m)^hi", "hi\nhello\nhi")
`
```

🧠 Regex Concepts

71. Difference between `` `re.match()` `` and `` `str.startswith()` ``?
72. When is it better to use `` `re.finditer()` `` over `` `re.findall()` ``?
73. Are regex patterns case-sensitive by default?
74. How do you make a regex pattern case-insensitive?
75. Is regex greedy or lazy by default?

✂️ Splitting with Regex

76. What does `` `re.split()` `` do?
77. Difference between `` `split()` `` and `` `re.split()` ``?
78. Split a string on whitespace.
79. Split a string on commas or semicolons.
80. Split only on the first 2 spaces using regex.

🧩 Lookaheads & Lookbehinds

81. What is a lookahead in regex?
82. What is a negative lookahead?
83. What is a lookbehind in regex?
84. What is a non-capturing group?
85. Example of lookahead to match "cat" only if not followed by "fish".

🧪 Output-Based: Advanced Patterns

86. Output?

```
python

`re.findall(r"(?<!@)\b\w+@\w+", "user@email.com")
`
```

87. Output?

```
python

`re.findall(r"(?<=\$)\d+", "Price is $30")
`
```

88. Output?

```
python

`re.findall(r"\d+(?=kg)", "12kg 100kg")
`
```

89. Output?

```
python

`re.findall(r"apple|orange", "apple banana orange")
`
```

90. Output?

```
python

`re.findall(r"colou?r", "color colour")
`
```

✅ True/False & MCQs

91. ``\d`` matches a letter. (T/F)
92. ``re.match()`` can match anywhere in the string. (T/F)
93. ``re.IGNORECASE`` ignores casing. (T/F)
94. ``re.findall()`` returns a list. (T/F)
95. ``re.compile()`` improves performance. (T/F)
96. Which matches a word? a) ``\w+``
    b) ``.``
    c) ``\s``
    d) ``^``
97. ``re.sub()`` returns: a) Match object
    b) Iterator
    c) String
    d) Tuple
98. What does ``re.split(r"\s+", text)`` do? a) Joins words
    b) Replaces spaces
    c) Splits on spaces
    d) Converts string to list
99. Which flag enables multiline mode? a) ``re.I``
    b) ``re.M``
    c) ``re.S``
    d) ``re.X``
100. Lookaheads allow: a) Backward match
    b) Forward match without consuming

c) String replace
d) Grouping

🔥 That’s 100 expert-level questions on **Python Regex (re module)** — perfect for technical interviews!

Next up:
➡️ **Topic 14: Python Data Types & Type Conversion – 100 Questions**

Let me know if you'd like a full **PDF, DOCX, or Excel sheet** for all topics so far!

Topic 14: Python Data Types & Type Conversion – 100 Questions

Awesome! Here's your **Topic 14: Python Data Types & Type Conversion – 100 Interview Questions** to sharpen your core Python knowledge and crush any interview 👊

🧪 Topic 14: Python Data Types & Type Conversion – 100 Interview Questions

---

🧱 Basic Data Types

1. What are the basic data types in Python?
2. How do you check the data type of a variable?
3. What is the difference between `int` and `float`?
4. What is a `bool` in Python?
5. What values are considered `False` in Python?

🔤 String Data Type

6. What is the type of `"123"`?
7. How are strings represented internally in Python?
8. Are strings mutable?
9. What does `str()` do?
10. How do you convert a number to a string?

🔢 Numeric Data Types

11. What are the numeric types in Python?
12. How is `float` different from `int`?
13. What is a complex number in Python?
14. What does `type(3+4j)` return?

15. What is the result of `` `int(3.9)` ``?

🧰 Type Conversion Basics

16. What is type conversion?
17. Difference between implicit and explicit conversion?
18. Example of implicit type conversion?
19. Example of explicit type conversion?
20. What is type casting?

🧪 Output-Based: Type Conversion

21. Output?

```
python

`int("10")
`
```

22. Output?

```
python

`float("3.14")
`
```

23. Output?

```
python

`str(123)
`
```

24. Output?

```
python

`bool("")
`
```

25. Output?

```
python

`complex(2, 3)
`
```

🧩 Sequence Data Types

26. What are sequence types in Python?
27. What's the difference between a list and a tuple?
28. Is a string a sequence?
29. Can you convert a list to a tuple?
30. Can you convert a string to a list?

🔁 Converting Between Sequences

31. Convert a string to list: `` `"abc"` `` → `` `['a', 'b', 'c']` ``
32. Convert list `` `[1, 2, 3]` `` to string `` `'123'` ``
33. Convert string `` `"1 2 3"` `` to list of integers.
34. Convert tuple `` `(1, 2)` `` to list.
35. Convert list to set.

🧪 Output-Based: Sequence Conversion

36. Output?

```
python

`list("abc")
`
```

37. Output?

```
python

`tuple([1, 2])
`
```

38. Output?

```
python

`str([1, 2])
`
```

39. Output?

```
python

`set([1, 2, 2])
`
```

40. Output?

```
python

`list((x for x in range(2)))
`
```

🔒 Boolean Conversion

41. What is the bool value of an empty list?
42. What is the bool value of `` `None` ``?
43. What is `` `bool("0")` ``?
44. What is `` `bool(0)` ``?
45. What values are truthy?

🧪 Output-Based: Boolean Logic

46. Output?

```
python

`bool(0)
`
```

47. Output?

```
python

`bool("False")
`
```

48. Output?

```
python

`bool([])
`
```

49. Output?

```
python

`bool(1)
`
```

50. Output?

```
python

`bool(" ")
`
```

🧰 Dictionary & Set Types

51. What is the type of `` `{}` ``?
52. How to create an empty set?
53. Can sets have duplicate elements?
54. What is the data type of `` `{'a': 1}` ``?
55. Are dictionaries ordered?

🔁 Converting Between Dict/Set

56. Convert list of tuples to dict.
57. Convert dict to list of keys.
58. Convert dict to list of values.
59. Convert set to list.
60. Convert list to dict using `` `dict()` ``.

🧪 Output-Based: Dict/Set Conversion

61. Output?

```
python

`dict([('a', 1), ('b', 2)])
`
```

62. Output?

```
python

`list({'x': 10}.keys())
`
```

63. Output?

```
python

`list({'x': 10}.values())
`
```

64. Output?

```
python

`set("hello")
`
```

65. Output?

```
python

`dict(zip("ab", [1, 2]))
`
```

🧠 NoneType & Special Types

66. What is `` `NoneType` `` in Python?
67. How do you check if a variable is `` `None` ``?
68. Can `` `None` `` be used in expressions?
69. What is the type of `` `type(None)` ``?
70. Is `` `None` `` equal to `` `False` ``?

🧪 Output-Based: NoneType

71. Output?

```
python

`type(None)
`
```

72. Output?

```
python

`None == False
`
```

73. Output?

```
python

`bool(None)
`
```

74. Output?

```
python

`None is None
`
```

75. Output?

```
python

`None + 1
`
```

📦 User-Defined & Custom Types

76. Can you create your own data type in Python?
77. How do you define a class as a type?
78. What does `` `type(obj)` `` return for user-defined class?
79. Can you override type conversion methods like `` `__int__()` ``?
80. What happens when you use `` `int(obj)` `` where `` `obj` `` is a custom class?

🔀 Advanced Type Conversion

81. Convert binary string `` `'101'` `` to int.
82. Convert int `` `5` `` to binary string.
83. Convert hexadecimal `` `'0xF'` `` to int.
84. Convert int to hex string.
85. Convert float to integer without losing value.

🧪 Output-Based: Advanced Conversion

86. Output?

```
python

`int("101", 2)
`
```

87. Output?

```
python

`bin(5)
`
```

88. Output?

```
python

`hex(15)
`
```

89. Output?

```
python

`int("0x10", 16)
`
```

90. Output?

```
python

`round(3.6)
`
```

✅ True/False & MCQs

91. Strings are immutable. (T/F)
92. Lists are ordered and mutable. (T/F)
93. `set()` maintains order. (T/F)
94. `float("3.0")` returns a float. (T/F)
95. `int("3.5")` is valid. (T/F)
96. Which of the following is not a built-in type?
    a) list
    b) set
    c) enum
    d) dict
97. Which of the following is immutable?
    a) list
    b) dict
    c) set
    d) tuple
98. `str([1, 2])` returns:
    a) `'1,2'`
    b) `'[1, 2]'`
    c) `[1, 2]`
    d) Error
99. What’s the result of `bool("False")`?
    a) True
    b) False
    c) None
    d) Error
100. What’s the type of value returned by `input()`?
    a) str
    b) int
    c) any
    d) object

🔥 **Boom!** You now have 100 solid questions on **Python Data Types & Type Conversion** — a core topic that every interviewer loves to test.

Up Next:
➡️ **Topic 15: Python Memory Management & Garbage Collection**

Wanna continue with the next topic or get all questions as a **PDF/Excel** so far?

how much topic is remaining

You're absolutely crushing it! 🚀 Here's a quick progress + what's left:

✅ Completed Topics (14/20):

1. Python Basics
2. Data Structures
3. Control Flow & Loops
4. Functions & Lambda
5. Object-Oriented Programming (OOP)
6. Modules and Packages
7. File Handling
8. Exception Handling
9. Iterators and Generators
10. Decorators & Closures
11. Built-in Functions & Scope
12. Python Comprehensions
13. Regular Expressions
14. Data Types & Type Conversion

📚 Remaining Topics (6):

15. **Memory Management & Garbage Collection**
16. **Python Date & Time (datetime, time modules)**
17. **Multithreading & Multiprocessing**
18. **Python and Databases (SQLite, SQLAlchemy basics)**
19. **Testing in Python (unittest, pytest)**
20. **Python Coding Patterns & Best Practices**

Wanna move on to **Topic 15: Memory Management & Garbage Collection** now? Or need a quick summary/download for what we've done so far?

Topic 15: Python Memory Management & Garbage Collection

Awesome choice! 🧠 Here are **100 Interview Questions on Python Memory Management & Garbage Collection (Topic 15)** — designed to help you **master low-level Python behavior** and impress in system-level interviews or any advanced Python role!

🗑️ Topic 15: Python Memory Management & Garbage Collection – 100 Questions

---

🧠 Basics of Memory Management

1. What is memory management in Python?
2. How does Python manage memory internally?
3. What is private heap space in Python?
4. What is a memory manager?
5. What are Python's built-in memory management tools?

🧮 Python Object Memory Model

6. What is the memory layout of a Python object?
7. How is memory allocated for variables?
8. What is an object reference?
9. What is reference counting?
10. How does Python track the number of references?

📦 Variables and Object References

11. What happens when you assign one variable to another?
12. Are variables in Python names or memory locations?
13. What is the `` `id()` `` function used for?
14. How can you check if two variables point to the same object?
15. Does reassigning a variable free memory?

🧪 Output-Based: Reference & `` `id()` ``

16. Output?

```
python

`a = [1, 2]
b = a
print(id(a) == id(b))
`
```

17. Output?

```
python

`a = 1000
b = 1000
print(a is b)
`
```

18. Output?

```
python

`a = 256
b = 256
print(a is b)
`
```

19. Output?

```
python

`a = "hello"
b = "hello"
print(id(a), id(b))
`
```

20. Output?

```
python

`x = [1, 2]
x = [3, 4]
`
```

Does the old list `` `[1, 2]` `` get garbage collected?

🧹 Garbage Collection Basics

21. What is garbage collection in Python?
22. Why is garbage collection needed?
23. What is cyclic garbage?
24. How does Python detect unreachable objects?
25. How does Python's GC differ from Java's?

🔄 Reference Counting

26. What is the primary garbage collection strategy in Python?
27. How does reference counting work?
28. When does Python destroy an object?
29. What is a memory leak?
30. Can reference counting alone lead to memory leaks?

🔗 Circular References

31. What is a circular reference?
32. How do circular references cause issues?
33. Why doesn’t reference counting handle circular references?
34. How does Python deal with circular references?
35. Can you create a circular reference in Python?

🧪 Code-Based: Circular References

36. Code to create a circular reference with two objects.
37. Detect if circular reference exists in a program.
38. Output?

```
python

`import gc
gc.collect()
`
```

39. What is returned by `` `gc.collect()` ``?
40. Is `` `del` `` enough to remove circular references?

⚙️ `` `gc` `` Module

41. What is the `` `gc` `` module?
42. What is `` `gc.collect()` `` used for?
43. What is `` `gc.get_objects()` ``?
44. What does `` `gc.get_referrers()` `` return?
45. What does `` `gc.get_referents()` `` do?

🔬 Garbage Collection Tuning

46. How can you enable/disable the garbage collector?
47. Use `` `gc.disable()` `` – what happens?
48. Use `` `gc.enable()` `` – what happens?
49. How to track collection thresholds?
50. What does `` `gc.set_debug(gc.DEBUG_LEAK)` `` do?

🧪 Output-Based: `` `gc` `` Module

51. Output?

```
python

`import gc
gc.isenabled()
`
```

52. Output?

```
python

`gc.get_threshold()
`
```

53. Output?

```
python

`gc.set_threshold(1000, 10, 10)
`
```

54. Output?

```
python

`gc.get_count()
`
```

55. Output?

```
python

`gc.collect()
`
```

🧠 Memory Optimization

56. How to reduce memory usage in Python?
57. What is `` `__slots__` ``?

58. How do `` `__slots__` `` save memory?
59. When should you use `` `__slots__` ``?
60. Limitations of `` `__slots__` ``?

🧪 Output-Based: `` `__slots__` ``

61. Output?

```
python

`class A:
    __slots__ = ['x']
a = A()
a.x = 5
`
```

62. Output?

```
python

`a.y = 10  # After defining __slots__ = ['x']
`
```

63. Can you inherit `` `__slots__` ``?
64. Can `` `__slots__` `` be used with dataclasses?
65. Does `` `__slots__` `` improve performance?

📈 Memory Profiling Tools

66. What is `` `sys.getsizeof()` ``?
67. What does it measure?
68. Use of `` `pympler` `` package?
69. What is `` `memory_profiler` ``?
70. How to profile memory usage line-by-line?

🧪 Output-Based: Memory Size

71. Output?

```
python

`import sys
sys.getsizeof(42)
`
```

72. Output?

```
python

`sys.getsizeof("hello")
`
```

73. Output?

```
python

`sys.getsizeof([])
`
```

74. Output?

```
python

`sys.getsizeof({})
`
```

75. Output?

```
python

`sys.getsizeof(set())
`
```

🛠️ Low-Level Memory Tuning

76. What is `` `object.__del__()` `` used for?
77. What is a finalizer in Python?
78. When is `` `__del__()` `` called?
79. Is relying on `` `__del__()` `` a good practice?
80. What are weak references?

🧪 Output-Based: Finalizers

81. Output?

```
python

`class A:
    def __del__(self):
        print("Deleted")
a = A()
del a
`
```

82. What if object with `` `__del__()` `` is part of circular ref?
83. How does Python handle finalizers in cyclic GC?

84. What is `` `weakref` `` module?
85. Use of `` `weakref.ref()` ``?

📉 Memory Leaks

86. What causes memory leaks in Python?
87. Can circular refs leak memory?
88. Can closures cause memory leaks?
89. Can global vars cause memory leaks?
90. How to detect memory leaks?

✅ True/False & MCQs

91. Python uses mark-and-sweep GC. (T/F)
92. `` `gc.collect()` `` returns the number of unreachable objects. (T/F)
93. Reference cycles are always collected. (T/F)
94. `` `__del__()` `` always gets called. (T/F)
95. `` `sys.getsizeof()` `` returns total memory footprint. (T/F)
96. What is used for automatic memory management in Python?
    a) malloc
    b) garbage collector
    c) new/delete
    d) buffer pool
97. `` `gc.collect()` `` can be used to:
    a) Disable GC
    b) Enable GC
    c) Manually trigger GC
    d) None
98. What module helps manage circular references?
    a) sys
    b) os
    c) gc
    d) ctypes
99. Weak references are used to:
    a) Extend object life
    b) Create memory leaks
    c) Avoid ref count increment
    d) None
100. `` `__slots__` `` improves memory by:
    a) Compressing data
    b) Preventing attribute dict

c) Replacing GC
d) Using heap allocation

🔥 **BOOM!** You're now fully loaded with 100 questions on **Python Memory Management & Garbage Collection** — a deep dive topic that sets top Python devs apart. 🧠⚡

Next up:
➡️ **Topic 16: Python Date & Time (datetime, time modules)**

Want to continue or pause to export everything so far into a PDF, DOCX, or Excel format?

Topic 16: Python Date & Time (datetime, time modules)

You got it! 🕒 Here's **Topic 16: Python Date & Time (datetime, time modules)** with **100 interview questions** — perfect for nailing questions about timestamps, scheduling, or real-time processing.

📅 Topic 16: Python Date & Time – 100 Interview Questions

(*Covers `datetime`, `time`, `calendar`, and timezone handling*)

🧠 Basics of Date & Time

1. What modules are used for handling date and time in Python?
2. Difference between `datetime` and `time` modules?
3. What does `datetime.now()` return?
4. How do you get today's date?
5. How to get the current time?

📅 Working with `datetime`

6. What is a `datetime` object?
7. How do you create a specific datetime object?
8. What does `datetime.today()` return?
9. What is `datetime.utcnow()`?
10. Difference between `datetime.now()` and `datetime.utcnow()`?

🧪 Output-Based: Basic `datetime`

11. Output?

```
python

`from datetime import datetime
print(datetime.now().date())
`
```

12. Output?

```
python

`datetime(2023, 12, 25)
`
```

13. Output?

```
python

`datetime.now().year
`
```

14. Output?

```
python

`datetime.now().strftime('%Y-%m-%d')
`
```

15. Output?

```
python

`datetime.now().strftime('%A')
`
```

⏱️ Formatting Dates and Times

16. What is `strftime()`?
17. Format current date as `dd-mm-yyyy`.
18. Format current time as `HH:MM:SS`.
19. How do you get full month name?
20. What does `%Y`, `%m`, `%d`, `%H`, `%M`, `%S` represent?

🧪 Output-Based: Formatting

21. Output?

```
python

`datetime.now().strftime("%b %d, %Y")
```

```
`
```

22. Output?

```
python

`datetime.now().strftime("%I:%M %p")
`
```

23. Output?

```
python

`datetime.now().strftime("%j")
`
```

24. Output?

```
python

`datetime.now().strftime("%U")
`
```

25. Output?

```
python

`datetime.now().strftime("%c")
`
```

⏳ Time Delta Calculations

26. What is `` `timedelta` ``?
27. How do you add 10 days to a date?
28. How to subtract two dates?
29. How to get the number of days between two dates?
30. Can you add a `` `timedelta` `` to `` `datetime.now()` ``?

🧪 Output-Based: `` `timedelta` ``

31. Output?

```
python

`from datetime import timedelta
print(datetime.now() + timedelta(days=5))
`
```

32. Output?

```
python

`timedelta(days=2, hours=3).total_seconds()
`
```

33. Output?

```
python

`(datetime(2024,1,1) - datetime(2023,1,1)).days
`
```

34. Output?

```
python

`timedelta(weeks=1)
`
```

35. Output?

```
python

`datetime.now() - timedelta(minutes=90)
`
```

🕰️ Working with `` `time` `` Module

36. What does `` `time.time()` `` return?
37. What is `` `time.sleep()` `` used for?
38. How do you pause execution for 2 seconds?
39. What does `` `time.ctime()` `` return?
40. How is `` `time.localtime()` `` used?

🧪 Output-Based: `` `time` ``

41. Output?

```
python

`import time
print(time.time())
`
```

42. Output?

```
python

`print(time.ctime())
```

```
`
```

43. Output?

```
python

`time.sleep(1)
print("Hello")
`
```

44. Output?

```
python

`print(time.localtime().tm_year)
`
```

45. Output?

```
python

`print(time.strftime("%H:%M:%S", time.localtime()))
`
```

🕰️ Parsing Strings to Dates

46. What is `strptime()` used for?
47. How do you convert `'2024-01-01'` to a datetime object?
48. Difference between `strptime()` and `strftime()`?
49. Parse `'12/25/2024'` using `strptime`.
50. Can `strptime()` raise errors?

🧪 Output-Based: `strptime`

51. Output?

```
python

`datetime.strptime('2024-04-01', '%Y-%m-%d')
`
```

52. Output?

```
python

`datetime.strptime("01/04/2024", "%d/%m/%Y")
`
```

53. Output?

```python
`datetime.strptime("03:30", "%H:%M")
`
```

54. Output?

```python
`(datetime.strptime("2024", "%Y")).year
`
```

55. Output?

```python
`datetime.strptime("2024-04", "%Y-%m")
`
```

🌍 Timezones

56. What is timezone-aware vs naive datetime?
57. How do you set a timezone in Python?
58. What is `pytz`?
59. Convert a UTC time to IST.
60. What does `datetime.now(tz=timezone.utc)` do?

🧪 Output-Based: Timezones

61. Output?

```python
`from datetime import timezone
datetime.now(timezone.utc)
`
```

62. Output?

```python
`import pytz
datetime.now(pytz.timezone('Asia/Kolkata'))
`
```

63. Output?

```python
`dt = datetime(2024, 1, 1, tzinfo=timezone.utc)
dt.astimezone(pytz.timezone("US/Eastern"))
`
```

64. Output?

```python
`datetime.utcnow().replace(tzinfo=pytz.utc)
`
```

65. Output?

```python
`pytz.all_timezones[:5]
`
```

📅 Working with `calendar` Module

66. How do you get the current month's calendar?
67. Get weekday of a date using `calendar.weekday()`
68. Print calendar for 2025.
69. What is `calendar.isleap()`?
70. What does `calendar.monthrange()` return?

🧪 Output-Based: `calendar`

71. Output?

```python
`import calendar
print(calendar.isleap(2024))
`
```

72. Output?

```python
`calendar.month(2025, 4)
`
```

73. Output?

```python
`calendar.weekday(2025, 4, 11)
`
```

74. Output?

```python
`calendar.monthrange(2024, 2)
`
```

75. Output?

```python
`calendar.leapdays(2000, 2025)
`
```

🛠️ Utilities & Edge Cases

76. How to get Unix timestamp from a `datetime`?
77. How to convert timestamp back to datetime?
78. What is `datetime.timestamp()`?
79. How to handle invalid date formats?
80. What causes `ValueError` in `strptime()`?

✅ True/False & MCQs

81. `datetime.now()` returns a string. (T/F)
82. `time.sleep()` pauses execution. (T/F)
83. `strftime` is used to parse a date string. (T/F)
84. `calendar.isleap(2024)` returns True. (T/F)
85. `datetime.now()` is timezone-aware. (T/F)
86. `%Y` stands for:
    a) Full year
    b) Month
    c) Day
    d) Hour
87. `datetime.utcnow()` returns:
    a) Local time
    b) UTC time
    c) Naive time
    d) Future time

88. `timedelta(days=1) + timedelta(hours=24)` is:
    a) 1 day
    b) 2 days
    c) Error
    d) 0 days

89. `calendar.weekday(2025, 4, 11)` returns:
    a) 4
    b) 5
    c) Friday
    d) 6

90. `pytz.timezone()` is used for:
    a) Formatting
    b) Parsing
    c) Localizing
    d) Delays

🧠 Challenge Questions

91. How to get the last day of a month?

92. How to count the number of Mondays in a year?

93. How to schedule a task every 5 minutes?

94. How to convert datetime to ISO format?

95. How to create a countdown timer using `time`?

🧪 Hands-On Practice

96. Write a function that returns current time in UTC+5:30.

97. Parse `01-01-2023` and return weekday name.

98. Convert datetime to timestamp and back.

99. Get all Sundays in current month.

100. Calculate age from birthdate.

🔥 **Boom!** You now have **100 solid questions** on **Python Date & Time** — from basic to timezone magic to real-world utilities. 💡⏱️

Up next:
➡️ **Topic 17: Multithreading & Multiprocessing (100 Questions)**

Wanna continue, or would you like a downloadable version (PDF/Excel) of all topics so far?

Topic 17: Multithreading & Multiprocessing (100 Questions)

Awesome! 🚀 Here's **Topic 17: Multithreading & Multiprocessing** with **100 Interview Questions** – covering concurrency, parallelism, thread safety, and Python's `` `threading` `` and `` `multiprocessing` `` modules. A must-know for high-performance apps, web servers, or system-level coding.

⚙️ Topic 17: Multithreading & Multiprocessing – 100 Interview Questions

---

🧠 Concurrency vs Parallelism

1. What is concurrency?
2. What is parallelism?
3. Difference between concurrency and parallelism?
4. Does Python support true parallelism?
5. What is the GIL (Global Interpreter Lock)?

🧵 Multithreading Basics

6. What is a thread?
7. What is multithreading?
8. How do you create a thread in Python?
9. What module is used for threading?
10. Difference between `` `start()` `` and `` `run()` ``?

🧪 Code Output: Basic Threading

11. Output?

```
python

`import threading
def job():
    print("Thread running")
t = threading.Thread(target=job)
t.start()
`
```

12. Output?

```
python

`t = threading.Thread(target=job)
t.run()
`
```

13. Output?

```
python

`t = threading.Thread(target=job)
t.start()
t.join()
`
```

14. Output?

```
python

`print(threading.active_count())
`
```

15. Output?

```
python

`print(threading.current_thread().name)
`
```

🔄 Thread Synchronization

16. What are race conditions?

17. What is a lock?

18. How do you use `` `threading.Lock()` ``?

19. What is a deadlock?

20. What is `` `threading.RLock()` ``?

🧪 Code Output: Locks

21. Output?

```
python

`lock = threading.Lock()
lock.acquire()
lock.release()
`
```

22. Output?

```
python

`with lock:
    print("Safe block")
`
```

23. Output?

```
python

`from threading import RLock
rlock = RLock()
rlock.acquire()
rlock.acquire()
print("Double acquire")
`
```

24. Can a thread acquire a lock it already holds?
25. What is the benefit of RLock over Lock?

📶 Thread Communication

26. What is a `Queue` in threading?
27. How does `queue.Queue()` help avoid race conditions?
28. What is `queue.get()`?
29. What is `queue.put()`?
30. What is the use of `queue.join()`?

⏱️ Daemon Threads

31. What is a daemon thread?
32. How do you create one?
33. What happens if the main thread finishes first?
34. How to set a thread as daemon?
35. Use-case of daemon threads?

✅ True/False & MCQs – Threading

36. Threads share the same memory. (T/F)
37. Locks prevent race conditions. (T/F)
38. Daemon threads block program exit. (T/F)
39. Thread `run()` creates a new thread. (T/F)
40. `join()` waits for a thread to finish. (T/F)

---

41. Threading module is:
    a) CPU parallel
    b) I/O concurrency
    c) Multi-processing
    d) Asynchronous only

42. GIL affects:
    a) Only multi-core CPUs
    b) Only Windows
    c) Python threads
    d) Java threads

43. To avoid race conditions:
    a) Use `print()`
    b) Use GIL
    c) Use `threading.Lock()`
    d) Use `exit()`

44. Output of `threading.active_count()` is:
    a) Dead threads
    b) Alive threads
    c) All modules
    d) CPU count

45. `queue.Queue()` is:
    a) Thread-safe
    b) File-safe
    c) Not thread-safe
    d) Not used in threading

🧰 Multiprocessing Basics

46. What is multiprocessing?

47. When should you use multiprocessing over threading?

48. How to create a new process?

49. What is `multiprocessing.Process()`?

50. How to start and join a process?

🧪 Code Output: Multiprocessing

51. Output?

```
python

`from multiprocessing import Process
def job():
    print("Running")
p = Process(target=job)
p.start()
`
```

52. Output?

```
python

`p.join()
print("Done")
`
```

53. Output?

```
python

`from multiprocessing import current_process
print(current_process().name)
`
```

54. Output?

```
python

`print(cpu_count())
`
```

55. What happens when you don’t join a process?

📤 Process Communication

56. How do processes share data?
57. What is `multiprocessing.Queue()`?
58. How does `Value` or `Array` work?
59. Can a shared list be modified across processes?
60. What is `Manager()`?

🧪 Code Output: Shared Data

61. Output?

```
python

`from multiprocessing import Value
v = Value('i', 10)
`
```

62. Output?

```
python

`from multiprocessing import Manager
m = Manager()
lst = m.list()
`
```

63. What does `'i'` stand for in `Value('i', 0)`?
64. Can two processes update the same `Value` object?
65. Difference between `Array` and `list()` in multiprocessing?

🔐 Process Synchronization

66. Does multiprocessing support Locks?
67. What is `multiprocessing.Lock()`?
68. How is it different from `threading.Lock()`?
69. Can you use `with` statement on `multiprocessing.Lock()`?
70. Can you use semaphores with multiprocessing?

⛓️ Process Pooling

71. What is `Pool` in multiprocessing?
72. What is the benefit of using a Pool?
73. How do you map a function to multiple processes?
74. Difference between `map()` and `apply_async()`?
75. How many processes does a Pool create by default?

🧪 Code Output: `Pool`

76. Output?

```
python

`from multiprocessing import Pool
def sq(x): return x*x
with Pool(2) as p:
    print(p.map(sq, [1, 2, 3]))
`
```

77. Output?

```
python

`p.apply_async(print, args=("Hi",))
`
```

78. Output?

```
python

`p.close()
p.join()
`
```

79. What happens if you don’t `` `close()` `` a pool?
80. Can pool functions return values?

🧪 Performance & Use-Cases

81. When is threading faster than multiprocessing?
82. When is multiprocessing better than threading?
83. Can CPU-bound tasks benefit from threads?
84. Why multiprocessing bypasses GIL?
85. What are some use-cases of multiprocessing?

🧠 Advanced Concepts

86. What are zombies and orphan processes?
87. How to handle exceptions in threads?
88. How to timeout a thread or process?
89. Can you kill a thread?
90. Can you terminate a process?

✅ True/False & MCQs – Multiprocessing

91. Processes share memory space. (T/F)
92. Processes are lighter than threads. (T/F)
93. GIL exists in multiprocessing. (T/F)
94. `` `Process().start()` `` is blocking. (T/F)
95. `` `Pool.map()` `` is parallel. (T/F)
96. Which is better for CPU-bound?
    a) Threading
    b) Multiprocessing
    c) AsyncIO
    d) GIL
97. Which module supports shared memory?
    a) threading
    b) queue
    c) multiprocessing
    d) memoryview
98. What causes GIL bottleneck?
    a) Memory leaks
    b) CPU-bound work

    c) Blocking I/O
    d) Thread starvation
99. Which of these is not thread-safe?
    a) Queue
    b) List
    c) Lock
    d) RLock

100. `multiprocessing.Manager().list()` is:
    a) Not shared
    b) Shared
    c) A file
    d) Thread-specific

🔥 BOOM! That’s **100 questions** for mastering **Multithreading & Multiprocessing in Python** — ready for systems interviews, backend dev roles, and performance-heavy apps. 🧠⚙️

Next up:
➡️ **Topic 18: Python Networking (socket, requests, urllib)**
Ready to dive in?

opic 18: Python Networking (socket, requests, urllib)

Awesome, let's get into it! 🌐 Here's **Topic 18: Python Networking (socket, requests, urllib)** with **100 Interview Questions** – covering low-level networking with `socket`, and high-level web communication with `requests` and `urllib`.

🌐 Topic 18: Python Networking – 100 Interview Questions

(socket, requests, urllib modules)

🧠 Basics of Networking in Python

1. What is networking in Python?
2. What modules are used for network programming in Python?
3. Difference between low-level and high-level networking?
4. What is the `socket` module used for?
5. What is the `requests` module used for?

🔌 Python `socket` Module

6. What is a socket?
7. How do you create a socket in Python?

8. What is the default socket family?
9. What is the default socket type?
10. What is the difference between TCP and UDP?

🧪 Output-Based: Basic `socket`

11. Output?

```
python

`import socket
s = socket.socket()
print(s)
`
```

12. Output?

```
python

`socket.AF_INET
`
```

13. Output?

```
python

`socket.SOCK_STREAM
`
```

14. Output?

```
python

`socket.gethostname()
`
```

15. Output?

```
python

`socket.gethostbyname('localhost')
`
```

🖥️ Server & Client with `socket`

16. How to create a TCP server in Python?
17. How to create a TCP client in Python?
18. What is `bind()` used for?
19. What does `listen()` do?

20. How does `accept()` work?

🧪 Output-Based: TCP Server/Client

21. Output?

```
python

`s.bind(('localhost', 9999))
`
```

22. Output?

```
python

`s.listen(5)
`
```

23. Output?

```
python

`client_socket.connect(('localhost', 9999))
`
```

24. Output?

```
python

`conn, addr = server_socket.accept()
`
```

25. Output?

```
python

`client_socket.sendall(b'Hello')
`
```

📡 UDP with Sockets

26. What is a UDP socket?
27. How to create a UDP socket in Python?
28. Difference between `send()` and `sendto()`?
29. Difference between `recv()` and `recvfrom()`?
30. Use-case for UDP over TCP?

🧪 Code Output: UDP

31. Output?

```python
`s = socket.socket(socket.AF_INET, socket.SOCK_DGRAM)
`
```

32. Output?

```python
`s.sendto(b"Hi", ('localhost', 12345))
`
```

33. Output?

```python
`data, addr = s.recvfrom(1024)
`
```

34. Output?

```python
`s.close()
`
```

35. Can you use `` `connect()` `` on UDP?

⚙️ Socket Utilities & Config

36. What is `` `socket.settimeout()` ``?
37. What is `` `socket.getsockname()` ``?
38. What does `` `socket.getpeername()` `` return?
39. How do you enable reuse address?
40. How do you handle timeouts in sockets?

✅ True/False – Socket

41. TCP is connectionless. (T/F)
42. UDP is reliable. (T/F)
43. `` `bind()` `` is used by clients. (T/F)
44. `` `recv()` `` blocks the program. (T/F)
45. `` `accept()` `` returns two values. (T/F)

🌍 High-Level Networking with `requests`

46. What is the `requests` module?
47. How do you make a GET request?
48. How do you make a POST request?
49. How to add headers to a request?
50. What does `requests.get().json()` do?

🧪 Output-Based: `requests`

51. Output?

```
python

`import requests
r = requests.get("https://httpbin.org/get")
print(r.status_code)
`
```

52. Output?

```
python

`r = requests.post("https://httpbin.org/post", data={"a": 1})
`
```

53. Output?

```
python

`r.headers['Content-Type']
`
```

54. Output?

```
python

`r.text[:50]
`
```

55. Output?

```
python

`r = requests.get("https://httpbin.org/status/404")
print(r.ok)
`
```

🛡️ Advanced Usage: `requests`

56. How to set a timeout in requests?
57. How to send custom headers?
58. How to handle cookies?
59. How to upload files using `` `requests` ``?
60. How to send JSON body with POST?

✅ True/False – Requests

61. `` `requests.get()` `` is asynchronous. (T/F)
62. You can post JSON with `` `data=` ``. (T/F)
63. `` `status_code` `` 200 means success. (T/F)
64. `` `requests` `` handles redirects automatically. (T/F)
65. `` `requests.post()` `` requires data or json. (T/F)

🌐 Working with `` `urllib` ``

66. What is `` `urllib` ``?
67. Difference between `` `urllib` `` and `` `requests` ``?
68. How to make a GET request with `` `urllib` ``?
69. What is `` `urlencode()` ``?
70. How to read response content?

🧪 Code Output: `` `urllib` ``

71. Output?

```
python

`from urllib import request
r = request.urlopen('https://example.com')
print(r.status)
`
```

72. Output?

```
python

`from urllib.parse import urlencode
urlencode({'name': 'John', 'age': 25})
`
```

73. Output?

```
python

`from urllib.request import urlopen
urlopen("https://www.google.com").read()[:50]
`
```

74. Output?

```
python

`from urllib.parse import urlparse
urlparse("https://example.com/path?q=1")
`
```

75. Output?

```
python

`from urllib.error import HTTPError
`
```

🔐 Headers, Params, Errors

76. How to pass headers with `urllib`?
77. How to pass query parameters?
78. How to handle HTTP errors?
79. What is `HTTPError`?
80. What is `URLError`?

📤 Uploading, Downloading Files

81. How to download a file with `requests`?
82. How to stream large files?
83. How to upload a file using `requests.post()`?
84. How to save response to a file?
85. How to download binary data?

🧪 Practical Code Snippets

86. Download and save an image from a URL.
87. Send a JSON POST request.
88. Create a simple socket-based chat app.
89. Get IP address of a website.
90. Write a Python HTTP client using `socket`.

🧠 Scenario/Challenge Questions

91. What happens if the server does not respond?
92. How to retry a failed request?
93. How to check for internet connection?
94. How to handle timeout exceptions?
95. How to send a GET request every 10 seconds?

🧪 Bonus Concepts

96. Can sockets be non-blocking?
97. Difference between `` `connect()` `` and `` `connect_ex()` ``?
98. Can `` `requests` `` work without internet?
99. How to mock a request in unit tests?
100. When would you use low-level sockets over `` `requests` ``?

🔥 Boom! That's **100 networking questions** in the bag — from `` `socket` `` level control to high-level web requests. Perfect for interviews at backend, system, or full-stack roles. 🧠📡

Next up:
➡️ **Topic 19: Python Testing (unittest, pytest, doctest)**
Want to roll into that or take a breather?

Topic 19: Python Testing (unittest, pytest, doctest)

Let's gooo! 🧪 Here's **Topic 19: Python Testing – 100 Interview Questions** covering the essential Python testing frameworks: `` `unittest` ``, `` `pytest` ``, and `` `doctest` ``. These are key for writing maintainable, reliable, and professional-grade code – and are **super common in interviews.**

✅ Topic 19: Python Testing (`` `unittest` ``, `` `pytest` ``, `` `doctest` ``) – 100 Interview Questions

---

🧠 Basics of Testing

1. What is software testing?
2. Why is testing important in Python development?
3. Difference between manual testing and automated testing?
4. What types of tests are common in Python?

5. What’s the difference between unit testing and integration testing?

🧪 Introduction to `` `unittest` ``

6. What is the `` `unittest` `` module?
7. How do you create a basic unit test in Python?
8. What is `` `TestCase` ``?
9. How do you run a unit test from the command line?
10. What is `` `assertEqual()` `` in `` `unittest` ``?

🧪 Code Output: Basic `` `unittest` ``

11. Output?

```
python

`import unittest
class TestMath(unittest.TestCase):
    def test_add(self):
        self.assertEqual(2 + 2, 4)
`
```

12. Output?

```
python

`self.assertNotEqual(3 * 2, 5)
`
```

13. Output?

```
python

`self.assertTrue(4 < 5)
`
```

14. Output?

```
python

`self.assertFalse(5 < 4)
`
```

15. Output?

```
python

`self.assertIsNone(None)
`
```

🧪 Common `unittest` Assertions

16. What does `assertRaises()` do?
17. How do you check for exceptions in unit tests?
18. How to compare objects with `assertIs()`?
19. What does `assertIn()` test?
20. What is the difference between `assertEqual()` and `assertAlmostEqual()`?

🔁 Setup and Teardown

21. What is `setUp()` method?
22. What is `tearDown()` used for?
23. When is `setUpClass()` used?
24. When is `tearDownClass()` triggered?
25. Can `setUp()` be async?

✅ True/False – `unittest`

26. `unittest` runs top-down in file order. (T/F)
27. `assertEqual()` passes if values are same type and value. (T/F)
28. You can skip a test with a decorator. (T/F)
29. `tearDown()` runs after all tests. (F)
30. `setUp()` runs once per class. (F)

🧪 Skipping & Expected Failures

31. How do you skip a test?
32. What is `@unittest.skip()`?
33. What is `@unittest.expectedFailure()`?
34. Can you skip tests dynamically?
35. How to skip test based on platform?

⚡ Intro to `pytest`

36. What is `pytest`?
37. Why do developers prefer `pytest` over `unittest`?
38. How to write a basic test using `pytest`?
39. How does `pytest` discover tests?
40. How to install and run `pytest`?

🧪 Code Output: Basic `pytest`

41. Output?

```
python

`def test_add():
    assert 2 + 2 == 4
`
```

42. Output?

```
python

`assert "py" in "pytest"
`
```

43. Output?

```
python

`assert [1, 2] != [2, 1]
`
```

44. Output?

```
python

`assert isinstance(3.14, float)
`
```

45. Output?

```
python

`assert not False
`
```

🧪 Advanced `pytest`

46. How to group tests with classes?
47. What is a `pytest.fixture`?
48. What does the `tmp_path` fixture do?
49. How do you capture stdout in `pytest`?
50. What is `pytest.mark.parametrize()`?

🧪 Parametrized Tests

51. Example of `` `@pytest.mark.parametrize()` ``?
52. Can you parametrize multiple arguments?
53. Can you combine fixtures and parametrize?
54. What is the use-case of parametrize?
55. How many test cases will be created with:

```
python

`@pytest.mark.parametrize("x", [1, 2, 3])
`
```

🛑 Skipping & Marking Tests in `` `pytest` ``

56. How to skip tests in `` `pytest` ``?
57. What is `` `pytest.mark.skip()` ``?
58. How to mark a test as expected to fail?
59. What is `` `xfail` ``?
60. How do you conditionally skip a test?

🧪 Doctest Module

61. What is `` `doctest` `` in Python?
62. How does `` `doctest` `` work?
63. What does `` `doctest.testmod()` `` do?
64. Can you test docstring examples using `` `doctest` ``?
65. What's the advantage of `` `doctest` ``?

🧪 Code Output: `` `doctest` ``

66. Output?

```
python

`def add(x, y):
    """
    >>> add(2, 3)
    5
    """
    return x + y
`
```

67. How to run `` `doctest` `` from terminal?
68. Can `` `doctest` `` be used for class methods?
69. Does `` `doctest` `` support error testing?

70. How does `doctest` compare outputs?

🔬 Testing Techniques & Practices

71. What is TDD (Test-Driven Development)?
72. What is mocking?
73. How do you mock objects in tests?
74. What is `unittest.mock`?
75. What is `MagicMock()`?

🧪 Code Output: Mocking

76. Output?

```
python

`from unittest.mock import Mock
mock = Mock(return_value=10)
mock()
`
```

77. Output?

```
python

`mock.called
`
```

78. Output?

```
python

`mock.assert_called_once()
`
```

79. Output?

```
python

`mock.reset_mock()
`
```

80. How to patch a function during testing?

🧪 Integration Testing

81. What is integration testing?
82. How does it differ from unit testing?

83. Tools for integration testing in Python?
84. Can `pytest` be used for integration testing?
85. What is the role of fixtures here?

🧪 Code Organization & Best Practices

86. Where should test files be located?
87. What naming convention does `pytest` follow?
88. Should test functions contain assertions only?
89. Should you test private methods?
90. How to structure large test suites?

✅ True/False & MCQs

91. `pytest` requires classes. (F)
92. `unittest` works out-of-the-box in Python. (T)
93. `MagicMock()` is part of `doctest`. (F)
94. `doctest` is good for complex testing. (F)
95. `pytest` supports fixtures. (T)
96. Which of these is best for docstring testing?
    a) `unittest`
    b) `doctest`
    c) `pytest`
    d) `mocktest`
97. `pytest.mark.parametrize` is used to:
    a) Skip test
    b) Mock behavior
    c) Run multiple test inputs
    d) Ignore warnings
98. Which of these is not an assertion in `unittest`?
    a) `assertFalse()`
    b) `assertEqual()`
    c) `assert()`
    d) `assertIn()`
99. Which module handles mocking?
    a) `unittest`
    b) `doctest`
    c) `requests`
    d) `unittest.mock`
100. Can `pytest` test `unittest` test cases?
    a) Yes

b) No
c) Only in Py2
d) Only with fixtures

🔥 Done! That's **100 testing questions** to master Python testing using `unittest`, `pytest`, and `doctest`. You're now ready to ace interview questions about **test-driven development**, **mocking**, and even **test automation**.

Next up:
➡️ **Topic 20: Python Virtual Environments & Packaging (venv, pip, setup.py)**
Wanna roll into it?

Topic 20: Python Virtual Environments & Packaging (venv, pip, setup.py)

Here comes the final boss of Python fundamentals ⚙️ — **Topic 20: Python Virtual Environments & Packaging (venv, pip, setup.py)** with **100 Interview Questions**. These are must-know for real-world development, project isolation, deployment, and distribution. Let's dive in. 🐍📦

🛠️ Topic 20: Python Virtual Environments & Packaging (100 Questions)

🧠 Basics of Virtual Environments

1. What is a virtual environment in Python?
2. Why do we need virtual environments?
3. What's the difference between `venv` and `virtualenv`?
4. How do you create a virtual environment?
5. How do you activate a virtual environment?

💻 Platform-Specific Activation

6. How to activate `venv` on Windows?
7. How to activate `venv` on macOS/Linux?
8. What does `deactivate` do?
9. Can you have multiple `venv`s on the same machine?
10. What is stored inside a virtual environment?

🧪 Output-Based – Virtual Envs

11. Output?

```bash
`python -m venv env
`
```

12. Output?

```bash
`source env/bin/activate
`
```

13. Output?

```cmd
`env\Scripts\activate.bat
`
```

14. Output?

```bash
`deactivate
`
```

15. Output?

```bash
`which python
`
```

📦 Basics of pip

16. What is `` `pip` `` in Python?
17. How do you install a package using pip?
18. How do you uninstall a package using pip?
19. How to upgrade a package using pip?
20. What is the difference between `` `pip` `` and `` `pip3` ``?

📄 Working with requirements.txt

21. What is `` `requirements.txt` ``?
22. How do you generate `` `requirements.txt` ``?
23. How to install dependencies from a `` `requirements.txt` `` file?
24. Can `` `pip` `` freeze versions of all installed packages?

25. How to specify version ranges in `` `requirements.txt` ``?

🧪 Output-Based – pip

26. Output?

```
bash

`pip install flask
`
```

27. Output?

```
bash

`pip uninstall flask
`
```

28. Output?

```
bash

`pip freeze > requirements.txt
`
```

29. Output?

```
bash

`pip list
`
```

30. Output?

```
bash

`pip show requests
`
```

🧪 pip Advanced

31. How to install from a GitHub repo using pip?
32. How to install a local `` `.whl` `` file?
33. How to search packages using pip?
34. What is `` `--upgrade` `` flag used for?
35. What is the purpose of `` `pip cache` ``?

⚙️ Packaging Your Python Project

36. What is packaging in Python?
37. What is `` `setup.py` ``?
38. What's the role of `` `setuptools` ``?
39. What is `` `pyproject.toml` ``?
40. What is a `` `wheel (.whl)` `` file?

🧪 setup.py Breakdown

41. What is the `` `name` `` argument in `` `setup.py` ``?
42. What does `` `version` `` specify?
43. What is `` `packages=find_packages()` ``?
44. What is `` `install_requires` `` used for?
45. What is `` `entry_points` ``?

🧪 Sample setup.py Output

46. Output?

```
python

`from setuptools import setup, find_packages
setup(name="mypkg", version="0.1")
`
```

47. Output?

```
bash

`python setup.py sdist
`
```

48. Output?

```
bash

`python setup.py install
`
```

49. Output?

```
bash

`pip install .
`
```

50. Output?

```
python

`install_requires=['requests>=2.0.0']
`
```

📁 Python Package Structure

51. What should be inside a Python package directory?
52. Why is `__init__.py` important?
53. What goes in `MANIFEST.in`?
54. Difference between module and package?
55. Should every folder in your project be a package?

🧪 Versioning & Dependencies

56. What is Semantic Versioning (SemVer)?
57. What does `>=1.0.0,<2.0.0` mean?
58. What's a major vs minor vs patch version?
59. Can you specify dependency links in setup.py?
60. How to pin exact versions in `install_requires`?

🧪 Editable Installs & Dev Mode

61. What is `pip install -e .`?
62. When would you use editable mode?
63. How to develop locally with editable installs?
64. What's the role of `__main__.py` in a package?
65. Can you install multiple editable packages?

🚀 Uploading to PyPI

66. What is PyPI?
67. How do you upload a package to PyPI?
68. What is `twine`?
69. What is `dist/` folder?
70. How do you create a `.whl` file?

🧪 Practical PyPI Steps

71. Output?

```
bash

`python setup.py sdist bdist_wheel
`
```

72. Output?

```
bash

`twine upload dist/*
`
```

73. Output?

```
bash

`pip install your-package-name
`
```

74. Output?

```
bash

`python -m build
`
```

75. Output?

```
bash

`python -m twine upload --repository testpypi dist/*
`
```

🔐 venv Security & Isolation

76. Are virtual environments safe from system packages?
77. Can you use the same `venv` for multiple projects?
78. Is `venv` cross-platform?
79. How do you remove a virtual environment?
80. Can you share a virtual environment?

✅ True/False – venv/pip/setup.py

81. `pip install` installs globally by default. (T/F)
82. Virtual environments store their own Python binary. (T/F)
83. `requirements.txt` can be generated manually. (T/F)
84. `setup.py` is optional in modern packaging. (T/F)

85. A `whl` file is a binary distribution. (T/F)

💡 Scenario-Based Qs

86. How would you isolate dependencies in a large project?
87. How to package a tool with CLI entry points?
88. How to share your project with collaborators?
89. How would you build a dev/test/prod environment setup?
90. How do you downgrade a pip package?

🧪 Misc & Advanced

91. What is the difference between `build` and `install`?
92. How to install packages system-wide (not recommended)?
93. Can `pip` install packages from a `.txt` file with comments?
94. What's the difference between `pipenv` and `venv`?
95. How do you remove unused packages?

🧠 Final Knowledge Boost

96. What's the purpose of `__main__.py` in packaging?
97. What is a namespace package?
98. How do you test if your package works locally?
99. Can a virtual environment be nested?
100. What are the modern tools replacing `setup.py`?

🔥 BOOM! You've just powered through **100 questions on Python venv, pip, and packaging** — key for **real-world projects**, **interview readiness**, and even **open-source contributions**.

If you want, I can now send you:

- A summary of all 20 topics,
- Or generate **Topic 21: Advanced Python Libraries (NumPy, Pandas, Matplotlib, etc)** if you're going further.

Wanna keep leveling up or take a break? 😎

Topic 21: Advanced Python Libraries (NumPy, Pandas, Matplotlib, etc)

Awesome! Here's your **Topic 21: Advanced Python Libraries (NumPy, Pandas, Matplotlib, etc)** — with **100 interview-style questions** designed to solidify your grip on **data analysis, visualization, and numerical computing** in Python. This topic is super important for **data science**, **ML**, and **technical interviews**. Let's jump in! 📊🐍

📚 Topic 21: Advanced Python Libraries (NumPy, Pandas, Matplotlib, etc.) – 100 Questions

---

🔢 NumPy – Numerical Python

1. What is NumPy?
2. What is a NumPy array?
3. How do you create a 1D array in NumPy?
4. How do you create a 2D array?
5. What is the difference between a list and a NumPy array?
6. What does `np.zeros()` return?
7. What does `np.ones()` return?
8. How to create an identity matrix?
9. How to reshape an array in NumPy?
10. What does `np.arange()` do?
11. What is broadcasting in NumPy?
12. How is slicing done in NumPy?
13. How to access elements in a 2D array?
14. What does `np.mean()` compute?
15. What's the difference between `np.sum()` and `np.cumsum()`?
16. What does `np.dot()` do?
17. How do you perform element-wise multiplication?
18. How to transpose a matrix?
19. How to find max value in an array?
20. What is `axis=0` vs `axis=1`?

🧪 NumPy Code Output

21. Output?

```
python

`np.array([1, 2, 3]) + 5
`
```

22. Output?

```
python

`np.arange(4).reshape(2, 2)
`
```

23. Output?

```
python

`np.eye(3)
`
```

24. Output?

```
python

`a = np.array([1, 2, 3])
b = np.array([3, 2, 1])
a * b
`
```

25. Output?

```
python

`np.linspace(0, 1, 5)
`
```

🧵 Pandas – Data Manipulation

26. What is Pandas?
27. What is a DataFrame?
28. What is a Series in Pandas?
29. How do you create a DataFrame from a dictionary?
30. How to read a CSV file with Pandas?
31. How to view the first 5 rows of a DataFrame?
32. What does ``df.info()`` show?
33. How to get summary statistics?
34. How to select a column in a DataFrame?
35. How to filter rows by condition?
36. How to sort a DataFrame by column?
37. How to apply a function to a column?
38. What does ``df.isnull().sum()`` do?
39. How to drop missing values?

40. How to fill missing values?

🧪 Pandas Code Output

41. Output?

```
python

`pd.Series([10, 20, 30], index=["a", "b", "c"])
`
```

42. Output?

```
python

`df["Age"].mean()
`
```

43. Output?

```
python

`df[df["Salary"] > 50000]
`
```

44. Output?

```
python

`df.sort_values(by="Age", ascending=False)
`
```

45. Output?

```
python

`df.drop("Name", axis=1)
`
```

🔁 Pandas Grouping & Merging

46. What is `` `groupby()` `` in Pandas?
47. How to calculate mean salary per department?
48. How to merge two DataFrames?
49. What is the difference between `` `merge()` `` and `` `join()` ``?
50. How to concatenate DataFrames?
51. What is an inner join?
52. What is a left join?

53. What is a pivot table?
54. How to reset index?
55. How to set a new index?

🧮 Matplotlib – Visualization

56. What is Matplotlib?
57. What does `` `plt.plot()` `` do?
58. How to create a bar chart?
59. How to add labels and title?
60. What is `` `plt.show()` ``?

📊 Matplotlib Code Output

61. Output?

```
python

`plt.plot([1, 2, 3], [4, 5, 6])
plt.show()
`
```

62. Output?

```
python

`plt.bar(["A", "B"], [10, 20])
`
```

63. Output?

```
python

`plt.title("My Graph")
`
```

64. Output?

```
python

`plt.xlabel("X-axis")
plt.ylabel("Y-axis")
`
```

65. Output?

```python
`plt.scatter([1,2], [3,4])
`
```

🔁 Seaborn – Statistical Visualization

66. What is Seaborn?
67. How is Seaborn different from Matplotlib?
68. What is `sns.countplot()`?
69. How to make a heatmap using Seaborn?
70. What is a pairplot?

🧪 Visualization Interpretation

71. What chart would you use to show distribution of a numeric variable?
72. What plot best shows relationships between 2 variables?
73. When would you use a boxplot?
74. What does a scatter plot show?
75. When to use a heatmap?

🧠 NumPy vs Pandas

76. When would you use NumPy over Pandas?
77. Why is Pandas built on top of NumPy?
78. Which is better for tabular data?
79. Which library supports Series and DataFrames?
80. Which one supports matrix math directly?

🧰 Intermediate Pandas Skills

81. What does `apply()` do?
82. What does `map()` do?
83. Difference between `apply()` and `applymap()`?
84. How to rename columns?
85. How to filter rows with multiple conditions?

🧼 Data Cleaning Tasks

86. How to detect duplicates?

87. How to remove duplicates?
88. How to convert string column to datetime?
89. How to change data type of a column?
90. How to normalize or scale data?

💬 Misc & Advanced

91. What is broadcasting in NumPy?
92. How to flatten a NumPy array?
93. What's the role of `np.random`?
94. How to sample rows in Pandas?
95. How to write a DataFrame to Excel?

✅ MCQ/True-False Style

96. NumPy supports missing data. (F)
97. Pandas is mainly used for 2D tabular data. (T)
98. `plt.show()` is necessary to display a plot. (T)
99. `groupby()` returns a DataFrame. (F – it's a groupby object)
100. Seaborn is built on top of Matplotlib. (T)

💥 BOOM! You just explored **100 high-impact questions** on **NumPy, Pandas, Matplotlib, and Seaborn** — perfect for interview prep and real-world data work.

Wanna keep the momentum? Next up could be:

- **Topic 22: Python for Data Science (SciPy, StatsModels, scikit-learn basics)**
  Or
- A recap of all 21 topics so far?

Let me know where we're headed!

thank you

You're very welcome! 🙌
You've crushed **21 core Python topics** — that's an *incredible* milestone. 🐍💪

If you ever want:

- Topic 22 and beyond 🔬
- Mock interviews 🎤

- Code exercises or quizzes 🧠
- Or help with *any* Python project...

I’ve got your back. Keep going strong — you’re seriously on track to *crack it in one shot*. 🚀

Let me know anytime you’re ready to level up again!

can you give me algorithms questions and data structures also

Update memory?

YesNo

Absolutely! Let's kick off with **Data Structures and Algorithms (DSA)** interview questions — crucial for coding interviews at companies like Google, Amazon, Meta, etc.

I'll break it into **topics**, each with **100 questions**, just like we did for Python. Here's a suggested structure:

🔷 Data Structures Topics (100 questions each):

1. Arrays & Strings
2. Linked Lists
3. Stacks & Queues
4. Trees (Binary Trees, BSTs)
5. Heaps & Priority Queues
6. Graphs
7. Hashing & HashMaps
8. Tries
9. Disjoint Set (Union Find)
10. Advanced Data Structures (Segment Trees, Fenwick Trees, etc.)

🔷 Algorithms Topics (100 questions each):

11. Sorting & Searching
12. Recursion & Backtracking
13. Dynamic Programming
14. Greedy Algorithms
15. Bit Manipulation
16. Sliding Window & Two Pointers
17. Divide and Conquer
18. Topological Sorting

19. Graph Algorithms (DFS, BFS, Dijkstra, Floyd-Warshall, etc.)
20. Interview Mix (Mixed level challenges from LeetCode/FAANG)

Want me to begin from **Topic 1: Arrays & Strings** with 100 questions? Or do you want a different starting topic first?