Top 50 Python Interview Questions and Answers for 2025

July 2025

Introduction

This document provides 50 Python interview questions and answers, curated for 2025 interviews. It covers basic, intermediate, advanced, and coding-related topics, suitable for freshers and experienced candidates aiming for roles in software development, data science, and more. Use this guide to prepare effectively and ace your Python interview.

1 Basic Python Questions

1. **What is Python, and what are its key features?**
   Answer: Python is a high-level, interpreted, general-purpose programming language created by Guido van Rossum in 1991. Its key features include:

   - Simple, readable syntax
   - Dynamic typing and binding
   - Extensive standard library
   - Support for multiple paradigms (OOP, functional, procedural)
   - Cross-platform compatibility
   - Large community and third-party libraries (e.g., NumPy, Pandas)

   Python is widely used in web development, data science, AI, and automation.

2. **What is the difference between a list and a tuple?**
   Answer: Lists are mutable (can be modified) and defined with square brackets `[]`. Tuples are immutable (cannot be modified) and defined with parentheses `()`. Lists are suitable for dynamic data, while tuples are used for fixed data or as dictionary keys due to their hashability.

```
lst = [1, 2, 3]
lst[0] = 100  # Works
tup = (1, 2, 3)
tup[0] = 100  # Raises TypeError
```

3. **What is PEP 8?**
   Answer: PEP 8 is Pythons style guide for writing clean, readable code. It recommends 4-space indentation, limiting lines to 79 characters, and using `snake_case` for variables. It ensures code consistency and maintainability.

4. **What is the difference between == and is operators?**
   Answer: The == operator checks for value equality, while `is` checks for identity (same memory location).

```python
a = [1, 2, 3]
b = [1, 2, 3]
print(a == b)  # True (same values)
print(a is b)  # False (different objects)
```

5. **What is a lambda function?**
   Answer: A lambda function is an anonymous, single-expression function created using the `lambda` keyword, often used for short operations.

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

6. **What are `*args` and `**kwargs`?**
   Answer: `*args` allows a function to accept any number of positional arguments as a tuple. `**kwargs` allows any number of keyword arguments as a dictionary.

```python
def func(*args, **kwargs):
    print(args, kwargs)
func(1, 2, name="Alice")  # Output: (1, 2) {'name': 'Alice'}
```

7. **What is the purpose of `if __name__ == "__main__":`?**
   Answer: It checks if a Python module is run directly or imported. Code inside this block executes only when the script is run directly, not when imported.

```python
if __name__ == "__main__":
    print("Running directly")
```

8. **How does Python handle memory management?**
   Answer: Python uses:

   - **Reference Counting**: Tracks references to an object; deallocates when count reaches zero.
   - **Garbage Collection**: Handles cyclic references using generational garbage collection.
   - **Memory Manager**: Allocates heap space for objects.

9. **What is slicing in Python?**
   Answer: Slicing extracts a portion of a sequence (e.g., list, string, tuple) using `sequence[start:stop:step]`.

```python
lst = [1, 2, 3, 4, 5]
print(lst[1:4])  # Output: [2, 3, 4]
print(lst[-3:])  # Output: [3, 4, 5]
```

10. **What is the difference between `append()` and `extend()`?**
Answer: `append()` adds a single element to a list. `extend()` adds all elements from an iterable.

```
lst = [1, 2]
lst.append([3, 4])  # [1, 2, [3, 4]]
lst = [1, 2]
lst.extend([3, 4])  # [1, 2, 3, 4]
```

11. **What is a list comprehension?**
Answer: A concise way to create lists using a single line of code, with an expression and optional conditions.

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

12. **What is the difference between `range()` and `xrange()`?**
Answer: In Python 2, `range()` creates a list, while `xrange()` creates a generator-like object for memory efficiency. In Python 3, `range()` behaves like `xrange()`, and `xrange()` is removed.

```
for i in range(5):  # Memory-efficient in Python 3
    print(i)  # Output: 0, 1, 2, 3, 4
```

13. **What is the `pass` statement?**
Answer: The `pass` statement is a null operation used as a placeholder when no action is needed but syntax requires a statement.

```
def func():
    pass  # Placeholder for future code
```

14. **What is the difference between a module and a package?**
Answer: A module is a single Python file containing code (functions, classes, variables). A package is a directory of modules with an `__init__.py` file, enabling modular programming.

```
# mymodule.py
def greet():
    print("Hello")
# mypackage/__init__.py can import mymodule
```

15. **What is the `len()` function?**
Answer: The `len()` function returns the number of items in an object (e.g., list, string, dictionary).

```
lst = [1, 2, 3]
print(len(lst))  # Output: 3
```

2 Intermediate Python Questions

16. **What is a decorator?**
Answer: A decorator is a higher-order function that modifies another functions

behavior without changing its code, often used for logging or access control.

```python
def decorator(func):
    def wrapper():
        print("Before function call")
        func()
        print("After function call")
    return wrapper
@decorator
def greet():
    print("Hello")
greet()
```

Output:
Before function call
Hello
After function call

17. **What is the Global Interpreter Lock (GIL)?**
Answer: The GIL is a mutex in CPython that prevents multiple threads from executing Python bytecode simultaneously, ensuring thread-safe memory management. It limits multi-core utilization for CPU-bound tasks but is less impactful for I/O-bound tasks.

18. **What is the difference between shallow and deep copy?**
Answer: A shallow copy (`copy.copy()`) creates a new object but references nested objects. A deep copy (`copy.deepcopy()`) creates a fully independent copy, including nested objects.

```python
import copy
lst1 = [[1, 2], [3, 4]]
shallow = copy.copy(lst1)
deep = copy.deepcopy(lst1)
shallow[0][0] = 100  # Affects lst1
deep[0][0] = 200     # Does not affect lst1
```

19. **What is a generator, and how is it different from a list?**
Answer: A generator is an iterator that yields values one at a time using `yield`, saving memory by not storing the entire sequence. Unlike lists, generators are lazy-evaluated and single-use.

```python
def my_gen():
    for i in range(3):
        yield i
for num in my_gen():
    print(num)  # Output: 0, 1, 2
```

20. **What is duck typing?**
Answer: Duck typing focuses on an objects behavior (methods/properties) rather than its type. If it walks like a duck and quacks like a duck, its a duck.

```python
class Duck:
    def quack(self):
        print("Quack!")
class Person:
    def quack(self):
        print("I'm quacking!")
def make_quack(obj):
    obj.quack()
make_quack(Duck())     # Quack!
make_quack(Person())  # I'm quacking!
```

21. **How do you handle exceptions in Python?**
    Answer: Use `try-except` blocks to catch exceptions. `finally` executes regardless, and `else` runs if no exception occurs.

```python
try:
    result = 10 / 0
except ZeroDivisionError as e:
    print(f"Error: {e}")
else:
    print("No error")
finally:
    print("Always executes")
```

Output:
Error: division by zero
Always executes

22. **What is the `super()` function?**
    Answer: The `super()` function calls a parent class method, often used in inheritance to initialize parent attributes.

```python
class Parent:
    def __init__(self):
        self.value = "Parent"
class Child(Parent):
    def __init__(self):
        super().__init__()
        print(self.value)
Child()  # Output: Parent
```

23. **What is the difference between `split()` and `partition()`?**
    Answer: `split()` divides a string into a list based on a delimiter, splitting on all occurrences. `partition()` splits a string into a tuple of three parts: before, delimiter, and after, splitting only on the first occurrence.

```python
s = "a,b,c"
print(s.split(","))       # ['a', 'b', 'c']
print(s.partition(","))  # ('a', ',', 'b,c')
```

24. **What is a dictionary comprehension?**
    Answer: A concise way to create dictionaries using a single line, similar to list comprehensions.

```python
squares = {x: x**2 for x in range(5)}
print(squares)  # Output: {0: 0, 1: 1, 2: 4, 3: 9, 4: 16}
```

25. **What is the `zip()` function?**
    Answer: The `zip()` function pairs elements from multiple iterables into tuples, useful for parallel iteration.

```python
names = ["Alice", "Bob"]
ages = [25, 30]
print(list(zip(names, ages)))  # Output: [('Alice', 25),
    ('Bob', 30)]
```

26. **What is the `assert` statement used for?**
    Answer: The `assert` statement tests a condition, raising an `AssertionError` if false. Its used for debugging and internal checks.

```python
x = 5
assert x > 0, "x must be positive"
assert x < 0, "x must be negative"  # Raises AssertionError
```

27. **What is the `enumerate()` function?**
    Answer: The `enumerate()` function adds a counter to an iterable, returning index-value pairs.

```python
lst = ['a', 'b', 'c']
for i, val in enumerate(lst):
    print(f"Index {i}: {val}")
```

Output:
Index 0: a
Index 1: b
Index 2: c

28. **What are f-strings?**
    Answer: F-strings (formatted string literals) are strings prefixed with `f` that embed expressions inside curly braces, evaluated at runtime.

```python
name = "Alice"
print(f"Hello, {name}!")  # Output: Hello, Alice!
```

29. **What is the difference between `set` and `frozenset`?**
    Answer: A `set` is mutable and allows adding/removing elements. A `frozenset` is immutable and hashable, suitable as dictionary keys.

```python
s = set([1, 2, 3])
s.add(4)  # Works
fs = frozenset([1, 2, 3])
fs.add(4)  # Raises AttributeError
```

30. **What is the `map()` function?**
    Answer: The `map()` function applies a function to all items in an iterable, returning an iterator of results.

```python
nums = [1, 2, 3]
squares = map(lambda x: x**2, nums)
print(list(squares))  # Output: [1, 4, 9]
```

3 Advanced Python Questions

31. **How do you find the length of the longest substring without repeating characters?**
    Answer: Use a sliding window with a dictionary to track character positions.

```python
def lengthOfLongestSubstring(s):
    seen = {}
    max_length = start = 0
    for end, char in enumerate(s):
        if char in seen and seen[char] >= start:
            start = seen[char] + 1
        else:
            max_length = max(max_length, end - start + 1)
        seen[char] = end
    return max_length
print(lengthOfLongestSubstring("abcabcbb"))  # Output: 3
```

32. **How do you check if a linked list is a palindrome?**
    Answer: Find the middle using two pointers, reverse the second half, and compare both halves.

```python
class ListNode:
    def __init__(self, val=0, next=None):
        self.val = val
        self.next = next
def isPalindrome(head):
    if not head or not head.next:
        return True
    slow = fast = head
    while fast.next and fast.next.next:
        slow = slow.next
        fast = fast.next.next
    second_half = reverse(slow.next)
    first_half = head
    while second_half:
        if first_half.val != second_half.val:
            return False
        first_half = first_half.next
        second_half = second_half.next
    return True
def reverse(node):
    prev = None
    while node:
        next_node = node.next
        node.next = prev
        prev = node
        node = next_node
    return prev
```

33. **What is the `collections` module?**
Answer: The `collections` module provides specialized container data types like `namedtuple`, `deque`, `Counter`, `OrderedDict`, and `defaultdict`.

```
from collections import Counter
print(Counter("hello"))  # Output: Counter({'l': 2, 'h': 1,
   'e': 1, 'o': 1})
```

34. **What is the `multiprocessing` module used for?**
Answer: The `multiprocessing` module bypasses the GIL by using separate processes, enabling true parallelism for CPU-bound tasks.

```
from multiprocessing import Process
def print_square(num):
    print(num * num)
if __name__ == "__main__":
    p = Process(target=print_square, args=(5,))
    p.start()
    p.join()
```

Output: 25

35. **How do you implement a binary search?**
Answer: Binary search finds an element in a sorted list by dividing the search interval in half.

```
def binary_search(arr, target):
    left, right = 0, len(arr) - 1
    while left <= right:
        mid = (left + right) // 2
        if arr[mid] == target:
            return mid
        elif arr[mid] < target:
            left = mid + 1
        else:
            right = mid - 1
    return -1
print(binary_search([1, 2, 3, 4, 5], 3))  # Output: 2
```

36. **What is the `pickle` module?**
Answer: The `pickle` module serializes Python objects into a byte stream (`pickling`) and deserializes them (`unpickling`).

```
import pickle
data = {"name": "Alice"}
with open("data.pkl", "wb") as f:
    pickle.dump(data, f)
with open("data.pkl", "rb") as f:
    print(pickle.load(f))  # Output: {'name': 'Alice'}
```

37. **What is the difference between `__str__` and `__repr__`?**
Answer: `__str__` provides a readable string representation of an object, while `__repr__` provides an unambiguous representation for debugging.

```
class Person:
    def __init__(self, name):
        self.name = name
    def __str__(self):
        return f"Person: {self.name}"
    def __repr__(self):
        return f"Person('{self.name}')"
p = Person("Alice")
print(str(p))  # Output: Person: Alice
print(repr(p)) # Output: Person('Alice')
```

38. **What is a metaclass?**
Answer: A metaclass is a class of a class that defines how a class behaves. Its used to customize class creation.

```
class Meta(type):
    def __new__(cls, name, bases, attrs):
        attrs["custom"] = "value"
        return super().__new__(cls, name, bases, attrs)
class MyClass(metaclass=Meta):
    pass
print(MyClass.custom)  # Output: value
```

39. **How do you reverse a string in Python?**
Answer: Use slicing with a step of -1.

```
s = "hello"
print(s[::-1])  # Output: olleh
```

40. **What is the walrus operator?**
Answer: The walrus operator (:=), introduced in Python 3.8, assigns a value to a variable within an expression.

```
if (n := len("hello")) > 3:
    print(f"Length {n} is greater than 3")
```

Output: Length 5 is greater than 3

4 Coding and Practical Questions

41. **Write a function to check if a number is prime.**
Answer:

```
def is_prime(n):
    if n < 2:
```

```python
        return False
    for i in range(2, int(n**0.5) + 1):
        if n % i == 0:
            return False
    return True
print(is_prime(17))  # Output: True
```

42. **Write a function to compute the factorial of a number.**
    Answer:

```python
def factorial(n):
    if n == 0:
        return 1
    return n * factorial(n - 1)
print(factorial(5))  # Output: 120
```

43. **Write a function to find the Fibonacci sequence up to n terms.**
    Answer:

```python
def fibonacci(n):
    n0, n1 = 0, 1
    for _ in range(n):
        print(n0, end=", ")
        n0, n1 = n1, n0 + n1
fibonacci(6)  # Output: 0, 1, 1, 2, 3, 5,
```

44. **How do you merge two sorted lists?**
    Answer: Use a two-pointer approach to compare and merge elements.

```python
def merge_sorted_lists(list1, list2):
    result = []
    i, j = 0, 0
    while i < len(list1) and j < len(list2):
        if list1[i] <= list2[j]:
            result.append(list1[i])
            i += 1
        else:
            result.append(list2[j])
            j += 1
    result.extend(list1[i:])
    result.extend(list2[j:])
    return result
print(merge_sorted_lists([1, 3, 5], [2, 4, 6]))  # Output:
    [1, 2, 3, 4, 5, 6]
```

45. **How do you find the intersection of two lists?**
    Answer: Use sets for efficient intersection.

```python
def intersection(list1, list2):
    return list(set(list1) & set(list2))
print(intersection([1, 2, 3], [2, 3, 4]))  # Output: [2, 3]
```

46. **How do you remove duplicates from a list while maintaining order?**
Answer: Use a dictionary or set to track seen items.

```python
def remove_duplicates(lst):
    seen = set()
    return [x for x in lst if not (x in seen or seen.add(x))]
print(remove_duplicates([1, 2, 2, 3, 1]))  # Output: [1, 2,
    3]
```

47. **Write a function to check if a string is a palindrome.**
Answer:

```python
def is_palindrome(s):
    s = s.lower().replace(" ", "")
    return s == s[::-1]
print(is_palindrome("A man a plan a canal Panama"))  #
    Output: True
```

48. **How do you read a CSV file using Pandas?**
Answer: Use `pandas.read_csv()`.

```python
import pandas as pd
df = pd.read_csv("data.csv")
print(df.head())
```

49. **How do you implement a queue using two stacks?**
Answer: Use one stack for enqueue and another for dequeue, reversing order when needed.

```python
class Queue:
    def __init__(self):
        self.s1 = []
        self.s2 = []
    def enqueue(self, x):
        self.s1.append(x)
    def dequeue(self):
        if not self.s2:
            while self.s1:
                self.s2.append(self.s1.pop())
        return self.s2.pop() if self.s2 else None
q = Queue()
q.enqueue(1)
q.enqueue(2)
print(q.dequeue())  # Output: 1
```

50. **How do you find the most frequent element in a list?**
Answer: Use `collections.Counter`.

```python
from collections import Counter
def most_frequent(lst):
    return Counter(lst).most_common(1)[0][0]
print(most_frequent([1, 2, 2, 3, 2]))  # Output: 2
```

Conclusion

Practice these 50 questions to build confidence for your Python interview. Focus on understanding concepts, writing clean code, and explaining your thought process. For further practice, explore platforms like LeetCode, HackerRank, or DataCamp.