Top 100 Python Interview Questions and Answers for 2025 (Extended)

July 2025

Introduction

This document provides 100 new Python interview questions and answers for 2025, covering basic, intermediate, advanced, and coding topics. Designed for freshers and experienced developers, it complements earlier resources with fresh content, focusing on modern Python applications like data science, web development, and algorithms. Use this guide to prepare for technical interviews at top companies.

1 Basic Python Questions

1. **What is the difference between `bytes` and `str` types in Python?**
   Answer: The `str` type represents Unicode strings for text, while `bytes` represents a sequence of bytes for binary data. `str` is human-readable, while `bytes` is used for raw data like files or network protocols.

```
text = "hello"
binary = b"hello"
print(type(text))  # <class 'str'>
print(type(binary))  # <class 'bytes'>
```

2. **What is the `bool` type in Python?**
   Answer: The `bool` type represents Boolean values: `True` or `False`. Its a subclass of `int`, where `True` is 1 and `False` is 0.

```
print(True + 1)  # Output: 2
print(isinstance(True, int))  # Output: True
```

3. **What is a docstring in Python?**
   Answer: A docstring is a multiline string used to document a function, class, or module, placed immediately after the definition. Its accessible via `__doc__`.

```
def add(a, b):
    """Returns the sum of two numbers."""
    return a + b
print(add.__doc__)  # Output: Returns the sum of two numbers.
```

4. **What are Pythons built-in data types?**
   Answer: Pythons built-in data types include:
   - Numeric: `int`, `float`, `complex`
   - Sequence: `list`, `tuple`, `str`
   - Mapping: `dict`
   - Set: `set`, `frozenset`
   - Boolean: `bool`
   - Binary: `bytes`, `bytearray`
   - None: `NoneType`

5. **What is the `None` type?**
   Answer: `None` is a singleton object of `NoneType`, representing the absence of a value, often used as a default return value or placeholder.

```
def func():
    pass
print(func())  # Output: None
```

6. **What is the `type()` function?**
   Answer: The `type()` function returns the type of an object.

```
print(type(42))  # Output: <class 'int'>
print(type("hello"))  # Output: <class 'str'>
```

7. **What is the `id()` function?**
   Answer: The `id()` function returns a unique identifier for an object, typically its memory address in CPython.

```
x = 42
print(id(x))  # Output: Memory address (e.g.,
   140735674572896)
```

8. **What is the `print()` functions `end` parameter?**
   Answer: The `end` parameter specifies what to print at the end of the output, defaulting to a newline (`\n`).

```
print("Hello", end=" ")
print("World")  # Output: Hello World
```

9. **What is the `input()` function?**
   Answer: The `input()` function reads a line from standard input as a string.

```
name = input("Enter your name: ")
print(f"Hello, {name}")
```

10. **What is a Python namespace?**
    Answer: A namespace is a mapping from names to objects, used to manage variable scope (e.g., global, local, built-in namespaces).

```
x = 10  # Global namespace
def func():
    x = 20  # Local namespace
    print(x)
func()  # Output: 20
print(x)  # Output: 10
```

11. **What is the `global` keyword?**
Answer: The `global` keyword allows a function to modify a global variable.

```
x = 10
def modify():
    global x
    x = 20
modify()
print(x)  # Output: 20
```

12. **What is the `nonlocal` keyword?**
Answer: The `nonlocal` keyword allows a nested function to modify a variable in an enclosing functions scope.

```
def outer():
    x = 10
    def inner():
        nonlocal x
        x = 20
    inner()
    print(x)  # Output: 20
outer()
```

13. **What is the `isinstance()` function?**
Answer: The `isinstance()` function checks if an object is an instance of a specified class or tuple of classes.

```
x = 42
print(isinstance(x, int))  # Output: True
print(isinstance(x, (int, str)))  # Output: True
```

14. **What is the `issubclass()` function?**
Answer: The `issubclass()` function checks if a class is a subclass of another class or tuple of classes.

```
class A: pass
class B(A): pass
print(issubclass(B, A))  # Output: True
```

15. **What is the `any()` function?**
Answer: The `any()` function returns `True` if any element in an iterable is `True`.

```
print(any([False, True, False]))  # Output: True
print(any([False, False]))  # Output: False
```

16. **What is the `all()` function?**
Answer: The `all()` function returns `True` if all elements in an iterable are `True`.

```
print(all([True, True, True]))  # Output: True
print(all([True, False, True]))  # Output: False
```

17. **What is the `sorted()` function?**
Answer: The `sorted()` function returns a new sorted list from an iterable, optionally using a `key` function.

```
lst = [3, 1, 2]
print(sorted(lst))  # Output: [1, 2, 3]
print(sorted(lst, reverse=True))  # Output: [3, 2, 1]
```

18. **What is the `reversed()` function?**
Answer: The `reversed()` function returns an iterator of an iterable in reverse order.

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

19. **What is the `slice()` function?**
Answer: The `slice()` function creates a slice object for slicing sequences.

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

20. **What is the `ord()` function?**
Answer: The `ord()` function returns the Unicode code point of a single character.

```
print(ord('A'))  # Output: 65
```

21. **What is the `chr()` function?**
Answer: The `chr()` function returns the character corresponding to a Unicode code point.

```
print(chr(65))  # Output: A
```

22. **What is the `divmod()` function?**
Answer: The `divmod()` function returns a tuple of quotient and remainder for integer division.

```
print(divmod(10, 3))  # Output: (3, 1)
```

23. **What is the `pow()` function?**
Answer: The `pow()` function computes a number raised to a power, optionally with a modulus.

```
print(pow(2, 3))  # Output: 8
print(pow(2, 3, 5))  # Output: 3 (2^3 % 5)
```

24. **What is the `round()` function?**
Answer: The `round()` function rounds a number to a specified number of decimal places.

```
print(round(3.14159, 2))  # Output: 3.14
```

25. **What is the `format()` method for strings?**
Answer: The `format()` method formats strings using placeholders or named arguments.

```
print("Hello, {}".format("Alice"))  # Output: Hello, Alice
print("{name} is {age}".format(name="Bob", age=30))  #
    Output: Bob is 30
```

2 Intermediate Python Questions

26. **What is the `collections.namedtuple`?**
Answer: `namedtuple` creates tuple subclasses with named fields, improving readability.

```
from collections import namedtuple
Point = namedtuple('Point', ['x', 'y'])
p = Point(1, 2)
print(p.x, p.y)  # Output: 1 2
```

27. **What is the `collections.defaultdict`?**
Answer: `defaultdict` is a dictionary that provides a default value for missing keys.

```
from collections import defaultdict
d = defaultdict(int)
d['a'] += 1
print(d['a'])  # Output: 1
print(d['b'])  # Output: 0
```

28. **What is the `collections.deque`?**
Answer: `deque` is a double-ended queue for efficient appends and pops from both ends.

```
from collections import deque
d = deque([1, 2, 3])
d.appendleft(0)
print(d)  # Output: deque([0, 1, 2, 3])
```

29. **What is the `itertools` module?**
Answer: The `itertools` module provides tools for efficient iteration, like `combinations`, `permutations`, and `product`.

```
from itertools import combinations
print(list(combinations([1, 2, 3], 2)))  # Output: [(1, 2),
    (1, 3), (2, 3)]
```

30. **What is the `contextlib` module?**
Answer: The `contextlib` module provides utilities for context managers, like `contextmanager` for creating custom `with` statements.

```
from contextlib import contextmanager
@contextmanager
def temp_value():
    print("Enter")
    yield "temp"
    print("Exit")
with temp_value() as val:
    print(val)
# Output: Enter
#         temp
#         Exit
```

31. **What is the `functools.partial` function?**
Answer: `functools.partial` creates a new function with some arguments pre-filled.

```
from functools import partial
def multiply(x, y):
    return x * y
double = partial(multiply, 2)
print(double(5))  # Output: 10
```

32. **What is the `operator` module?**
Answer: The `operator` module provides functions for built-in operators, useful for functional programming.

```
from operator import add
print(add(2, 3))  # Output: 5
```

33. **What is the `re` module for regular expressions?**
Answer: The `re` module provides functions for pattern matching and string manipulation using regular expressions.

```
import re
print(re.findall(r'\d+', "abc123def456"))  # Output: ['123',
    '456']
```

34. **What is the `datetime` module?**
Answer: The `datetime` module handles dates and times, including formatting and arithmetic.

```
from datetime import datetime
now = datetime.now()
print(now.strftime("%Y-%m-%d"))  # Output: Current date
    (e.g., 2025-07-27)
```

35. **What is the `os` module?**
Answer: The `os` module provides functions for interacting with the operating system, like file and directory operations.

```python
import os
print(os.getcwd())  # Output: Current working directory
```

36. **What is the `sys` module?**
Answer: The `sys` module provides system-specific parameters and functions, like command-line arguments.

```python
import sys
print(sys.argv)  # Output: List of command-line arguments
```

37. **What is the `json` module?**
Answer: The `json` module handles JSON encoding and decoding.

```python
import json
data = {"name": "Alice"}
print(json.dumps(data))  # Output: {"name": "Alice"}
```

38. **What is the `csv` module?**
Answer: The `csv` module provides functions to read and write CSV files.

```python
import csv
with open('data.csv', 'w', newline='') as f:
    writer = csv.writer(f)
    writer.writerow(['name', 'age'])
    writer.writerow(['Alice', 25])
```

39. **What is the `math` module?**
Answer: The `math` module provides mathematical functions and constants.

```python
import math
print(math.sqrt(16))  # Output: 4.0
print(math.pi)  # Output: 3.141592653589793
```

40. **What is the `random` module?**
Answer: The `random` module provides functions for generating random numbers and selections.

```python
import random
print(random.randint(1, 10))  # Output: Random integer
   between 1 and 10
```

41. **What is a Python virtual environment?**
Answer: A virtual environment isolates Python packages for a project, preventing conflicts between dependencies.

```python
# Create: python -m venv myenv
# Activate (Unix): source myenv/bin/activate
# Activate (Windows): myenv\Scripts\activate
```

42. **What is the `pip` tool?**
Answer: `pip` is Pythons package manager for installing and managing third-party libraries.

```
# Install a package
# pip install requests
```

43. **What is a Python iterator?**
Answer: An iterator is an object that implements `__iter__()` and `__next__()` for sequential access.

```
lst = [1, 2, 3]
it = iter(lst)
print(next(it))  # Output: 1
```

44. **What is a Python iterable?**
Answer: An iterable is an object that can be iterated over, implementing `__iter__()` or `__getitem__()`.

```
for x in [1, 2, 3]:
    print(x)  # Output: 1, 2, 3
```

45. **What is the `__init__` method?**
Answer: The `__init__` method is a constructor called when an object is created to initialize its attributes.

```
class Person:
    def __init__(self, name):
        self.name = name
p = Person("Alice")
print(p.name)  # Output: Alice
```

46. **What is method resolution order (MRO)?**
Answer: MRO determines the order in which base classes are searched for a method in inheritance, using C3 linearization.

```
class A: pass
class B(A): pass
print(B.__mro__)  # Output: (<class '__main__.B'>, <class
   '__main__.A'>, <class 'object'>)
```

47. **What is a classmethod in Python?**
Answer: A `@classmethod` takes the class as its first argument (`cls`), used for alternative constructors.

```
class MyClass:
    @classmethod
    def from_string(cls, s):
        return cls(s)
```

48. **What is a staticmethod in Python?**
Answer: A `@staticmethod` is a method that doesnt take `self` or `cls`, used for utility functions within a class.

```python
class MyClass:
    @staticmethod
    def utility():
        return "Utility"
print(MyClass.utility())  # Output: Utility
```

49. **What is the `property` decorator?**
Answer: The `@property` decorator creates getter, setter, and deleter methods for class attributes.

```python
class Person:
    def __init__(self, name):
        self._name = name
    @property
    def name(self):
        return self._name
p = Person("Alice")
print(p.name)  # Output: Alice
```

50. **What is the `threading` module?**
Answer: The `threading` module provides tools for creating and managing threads, suitable for I/O-bound tasks.

```python
import threading
def task():
    print("Running")
t = threading.Thread(target=task)
t.start()
t.join()
```

51. **What is the `asyncio` module?**
Answer: The `asyncio` module enables asynchronous programming using coroutines for concurrent I/O-bound tasks.

```python
import asyncio
async def say_hello():
    print("Hello")
    await asyncio.sleep(1)
    print("World")
asyncio.run(say_hello())
```

52. **What is a coroutine in Python?**
Answer: A coroutine is a function defined with `async def` that can be paused and resumed using `await`.

```python
import asyncio
async def example():
    await asyncio.sleep(1)
    return "Done"
asyncio.run(example())
```

53. **What is the `logging` module?**
Answer: The `logging` module provides flexible logging of messages, supporting different severity levels.

```
import logging
logging.basicConfig(level=logging.INFO)
logging.info("This is an info message")
```

54. **What is the `unittest` module?**
Answer: The `unittest` module is a built-in framework for writing and running unit tests.

```
import unittest
class TestMath(unittest.TestCase):
    def test_add(self):
        self.assertEqual(1 + 1, 2)
if __name__ == "__main__":
    unittest.main()
```

55. **What is the `pytest` framework?**
Answer: `pytest` is a third-party testing framework that simplifies writing and running tests with powerful features like fixtures.

```
# test_example.py
def test_add():
    assert 1 + 1 == 2
# Run: pytest test_example.py
```

3 Advanced Python Questions

56. **What is the `weakref` module?**
Answer: The `weakref` module creates weak references to objects, allowing them to be garbage-collected if no strong references exist.

```
import weakref
obj = [1, 2, 3]
ref = weakref.ref(obj)
print(ref())  # Output: [1, 2, 3]
del obj
print(ref())  # Output: None
```

57. **What is the `gc` module?**
Answer: The `gc` module provides an interface to Pythons garbage collector, allowing manual control and debugging.

```
import gc
print(gc.isenabled())  # Output: True
gc.collect()  # Force garbage collection
```

58. **What is the `dis` module?**
Answer: The `dis` module disassembles Python bytecode, useful for understanding how code is executed.

```python
import dis
def add(a, b):
    return a + b
dis.dis(add)
```

59. **What is the `inspect` module?**
Answer: The `inspect` module provides functions to inspect live objects, like source code or signatures.

```python
import inspect
def example():
    pass
print(inspect.getsource(example))
```

60. **What is the `abc` module?**
Answer: The `abc` module provides tools for defining abstract base classes using `ABC` and `abstractmethod`.

```python
from abc import ABC, abstractmethod
class Animal(ABC):
    @abstractmethod
    def speak(self):
        pass
```

61. **What is the `dataclasses` module?**
Answer: The `dataclasses` module simplifies class creation with automatic methods like `__init__` and `__repr__`.

```python
from dataclasses import dataclass
@dataclass
class Person:
    name: str
    age: int
p = Person("Alice", 25)
print(p)  # Output: Person(name='Alice', age=25)
```

62. **What is the `typing` module?**
Answer: The `typing` module provides type hints for static type checking, improving code clarity.

```python
from typing import List
def process(items: List[int]) -> int:
    return sum(items)
print(process([1, 2, 3]))  # Output: 6
```

63. **What is the `pathlib` module?**
Answer: The `pathlib` module provides an object-oriented interface for filesystem paths.

```python
from pathlib import Path
p = Path("example.txt")
print(p.exists())  # Output: True or False
```

64. **What is the `mmap` module?**
Answer: The `mmap` module allows memory-mapped file I/O, efficient for large files.

```
import mmap
with open("example.txt", "r") as f:
    with mmap.mmap(f.fileno(), 0, access=mmap.ACCESS_READ)
       as m:
        print(m.read(10))  # Read first 10 bytes
```

65. **What is the `argparse` module?**
Answer: The `argparse` module parses command-line arguments, providing a user-friendly interface.

```
import argparse
parser = argparse.ArgumentParser()
parser.add_argument("--name")
args = parser.parse_args()
print(args.name)
```

66. **What is the `enum` module?**
Answer: The `enum` module provides support for enumerations, creating named constants.

```
from enum import Enum
class Color(Enum):
    RED = 1
print(Color.RED)  # Output: Color.RED
```

67. **What is the `hashlib` module?**
Answer: The `hashlib` module provides cryptographic hashing functions like MD5 or SHA.

```
import hashlib
print(hashlib.md5(b"hello").hexdigest())  # Output:
   5d41402abc4b2a76b9719d911017c592
```

68. **What is the `shutil` module?**
Answer: The `shutil` module provides high-level file operations like copying and moving.

```
import shutil
shutil.copy("source.txt", "dest.txt")
```

69. **What is the `zipfile` module?**
Answer: The `zipfile` module handles creation and extraction of ZIP archives.

```
import zipfile
with zipfile.ZipFile("archive.zip", "w") as zf:
    zf.write("example.txt")
```

70. **What is the `tempfile` module?**
Answer: The `tempfile` module creates temporary files and directories.

```python
import tempfile
with tempfile.TemporaryFile() as f:
    f.write(b"Hello")
```

71. **What is the `subprocess` module?**
Answer: The `subprocess` module runs external commands and interacts with their input/output.

```python
import subprocess
subprocess.run(["echo", "Hello"], capture_output=True)
```

72. **What is the `socket` module?**
Answer: The `socket` module provides low-level networking interfaces for client-server communication.

```python
import socket
s = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
```

73. **What is the `requests` library?**
Answer: The `requests` library simplifies HTTP requests for APIs and web scraping.

```python
import requests
response = requests.get("https://api.example.com")
print(response.status_code)  # Output: 200
```

74. **What is the `beautifulsoup` library?**
Answer: The `beautifulsoup` library parses HTML/XML for web scraping.

```python
from bs4 import BeautifulSoup
html = "<p>Hello</p>"
soup = BeautifulSoup(html, "html.parser")
print(soup.p.text)  # Output: Hello
```

75. **What is the `numpy` library?**
Answer: The `numpy` library provides support for numerical arrays and mathematical operations.

```python
import numpy as np
arr = np.array([1, 2, 3])
print(arr * 2)  # Output: [2 4 6]
```

76. **What is the `pandas` library?**
Answer: The `pandas` library provides data structures like DataFrames for data manipulation and analysis.

```python
import pandas as pd
df = pd.DataFrame({"name": ["Alice"], "age": [25]})
print(df["name"])  # Output: 0    Alice
```

77. **What is the `matplotlib` library?**
Answer: The `matplotlib` library creates visualizations like plots and charts.

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

78. **What is the `seaborn` library?**
Answer: The `seaborn` library enhances `matplotlib` for statistical data visualization.

```
import seaborn as sns
tips = sns.load_dataset("tips")
sns.scatterplot(data=tips, x="total_bill", y="tip")
plt.show()
```

79. **What is the `scikit-learn` library?**
Answer: The `scikit-learn` library provides tools for machine learning, like classification and regression.

```
from sklearn.linear_model import LinearRegression
model = LinearRegression()
```

80. **What is the `tensorflow` library?**
Answer: The `tensorflow` library is used for machine learning and deep learning, supporting neural networks.

```
import tensorflow as tf
model = tf.keras.Sequential()
```

4 Coding and Practical Questions

81. **Write a function to check if two strings are anagrams.**
Answer: Compare sorted strings or use a character count.

```
def is_anagram(s1, s2):
    return sorted(s1) == sorted(s2)
print(is_anagram("listen", "silent"))  # Output: True
```

82. **Write a function to find the longest common prefix in a list of strings.**
Answer: Compare characters from all strings.

```
def longest_common_prefix(strs):
    if not strs:
        return ""
    for i, char in enumerate(strs[0]):
        for s in strs[1:]:
            if i >= len(s) or s[i] != char:
                return strs[0][:i]
    return strs[0]
print(longest_common_prefix(["flower", "flow", "flight"]))
  # Output: fl
```

83. **Write a function to reverse a linked list.**
Answer: Use iterative pointer manipulation.

```
class ListNode:
    def __init__(self, val=0, next=None):
        self.val = val
        self.next = next
def reverse_list(head):
    prev = None
    curr = head
    while curr:
        next_node = curr.next
        curr.next = prev
        prev = curr
        curr = next_node
    return prev
```

84. **Write a function to merge k sorted lists.**
Answer: Use a min-heap to merge efficiently.

```
from heapq import heappush, heappop
def merge_k_lists(lists):
    heap = []
    for i, lst in enumerate(lists):
        if lst:
            heappush(heap, (lst.val, i, lst))
    dummy = ListNode(0)
    curr = dummy
    while heap:
        val, i, node = heappop(heap)
        curr.next = ListNode(val)
        curr = curr.next
        if node.next:
            heappush(heap, (node.next.val, i, node.next))
    return dummy.next
```

85. **Write a function to find the median of two sorted arrays.**
Answer: Use binary search to partition arrays.

```
def find_median_sorted_arrays(nums1, nums2):
    if len(nums1) > len(nums2):
        nums1, nums2 = nums2, nums1
    x, y = len(nums1), len(nums2)
    left, right = 0, x
    while left <= right:
        px = (left + right) // 2
        py = (x + y + 1) // 2 - px
        left_x = float("-inf") if px == 0 else nums1[px - 1]
        right_x = float("inf") if px == x else nums1[px]
        left_y = float("-inf") if py == 0 else nums2[py - 1]
        right_y = float("inf") if py == y else nums2[py]
        if left_x <= right_y and left_y <= right_x:
```

```
            if (x + y) % 2 == 0:
                return (max(left_x, left_y) + min(right_x,
                    right_y)) / 2
            return max(left_x, left_y)
        elif left_x > right_y:
            right = px - 1
        else:
            left = px + 1
```

86. **Write a function to validate a binary search tree.**
Answer: Check if nodes are within valid ranges.

```
class TreeNode:
    def __init__(self, val=0, left=None, right=None):
        self.val = val
        self.left = left
        self.right = right
def is_valid_bst(root):
    def validate(node, min_val, max_val):
        if not node:
            return True
        if node.val <= min_val or node.val >= max_val:
            return False
        return validate(node.left, min_val, node.val) and
            validate(node.right, node.val, max_val)
    return validate(root, float("-inf"), float("inf"))
```

87. **Write a function to implement a stack using two queues.**
Answer: Use one queue for push and another for pop.

```
from collections import deque
class Stack:
    def __init__(self):
        self.q1 = deque()
        self.q2 = deque()
    def push(self, x):
        self.q1.append(x)
    def pop(self):
        while len(self.q1) > 1:
            self.q2.append(self.q1.popleft())
        result = self.q1.popleft()
        self.q1, self.q2 = self.q2, self.q1
        return result
```

88. **Write a function to find the intersection of two sorted arrays.**
Answer: Use two pointers to compare elements.

```
def intersect_sorted(arr1, arr2):
    result = []
    i = j = 0
    while i < len(arr1) and j < len(arr2):
        if arr1[i] == arr2[j]:
```

```
            result.append(arr1[i])
            i += 1
            j += 1
        elif arr1[i] < arr2[j]:
            i += 1
        else:
            j += 1
    return result
print(intersect_sorted([1, 2, 2, 3], [2, 2, 4]))  # Output:
   [2, 2]
```

89. **Write a function to rotate a matrix 90 degrees.**
Answer: Transpose and reverse rows.

```
def rotate_matrix(matrix):
    n = len(matrix)
    for i in range(n):
        for j in range(i, n):
            matrix[i][j], matrix[j][i] = matrix[j][i],
               matrix[i][j]
    for i in range(n):
        matrix[i].reverse()
    return matrix
print(rotate_matrix([[1, 2, 3], [4, 5, 6], [7, 8, 9]]))
# Output: [[7, 4, 1], [8, 5, 2], [9, 6, 3]]
```

90. **Write a function to find the longest palindromic substring.**
Answer: Use expand-around-center technique.

```
def longest_palindromic_substring(s):
    def expand(left, right):
        while left >= 0 and right < len(s) and s[left] ==
           s[right]:
            left -= 1
            right += 1
        return s[left + 1:right]
    result = ""
    for i in range(len(s)):
        palindrome1 = expand(i, i)
        palindrome2 = expand(i, i + 1)
        result = max(result, palindrome1, palindrome2,
           key=len)
    return result
print(longest_palindromic_substring("babad"))  # Output: bab
   or aba
```

91. **Write a function to group anagrams in a list of strings.**
Answer: Use sorted strings as dictionary keys.

```
def group_anagrams(strs):
    d = {}
    for s in strs:
```

```
        key = "".join(sorted(s))
        d.setdefault(key, []).append(s)
    return list(d.values())
print(group_anagrams(["eat", "tea", "tan", "ate", "nat",
    "bat"]))
# Output: [['eat', 'tea', 'ate'], ['tan', 'nat'], ['bat']]
```

92. **Write a function to implement LRU cache.**
Answer: Use a dictionary and doubly linked list.

```
from collections import OrderedDict
class LRUCache:
    def __init__(self, capacity):
        self.cache = OrderedDict()
        self.capacity = capacity
    def get(self, key):
        if key not in self.cache:
            return -1
        self.cache.move_to_end(key)
        return self.cache[key]
    def put(self, key, value):
        if key in self.cache:
            self.cache.move_to_end(key)
        self.cache[key] = value
        if len(self.cache) > self.capacity:
            self.cache.popitem(last=False)
```

93. **Write a function to find the kth largest element in an array.**
Answer: Use a min-heap.

```
from heapq import heappush, heappop
def find_kth_largest(nums, k):
    heap = []
    for num in nums:
        heappush(heap, num)
        if len(heap) > k:
            heappop(heap)
    return heappop(heap)
print(find_kth_largest([3, 2, 1, 5, 6, 4], 2))  # Output: 5
```

94. **Write a function to check if a number is a power of two.**
Answer: Use bitwise operations.

```
def is_power_of_two(n):
    return n > 0 and (n & (n - 1)) == 0
print(is_power_of_two(16))  # Output: True
print(is_power_of_two(18))  # Output: False
```

95. **Write a function to compute the square root of a number.**
Answer: Use Newtons method.

```
def sqrt(n):
```

```
    if n < 0:
        return -1
    x = n
    for _ in range(20):
        x = 0.5 * (x + n / x)
    return x
print(sqrt(16))  # Output: 4.0
```

96. **Write a function to find the longest consecutive sequence in an array.**
Answer: Use a set for O(1) lookups.

```
def longest_consecutive(nums):
    num_set = set(nums)
    longest = 0
    for num in num_set:
        if num - 1 not in num_set:
            length = 1
            while num + length in num_set:
                length += 1
            longest = max(longest, length)
    return longest
print(longest_consecutive([100, 4, 200, 1, 3, 2]))  # Output: 4
```

97. **Write a function to implement a trie for prefix search.**
Answer: Use a nested dictionary for the trie.

```
class Trie:
    def __init__(self):
        self.root = {}
        self.end = '*'
    def insert(self, word):
        node = self.root
        for char in word:
            node = node.setdefault(char, {})
        node[self.end] = True
    def search(self, word):
        node = self.root
        for char in word:
            if char not in node:
                return False
            node = node[char]
        return self.end in node
```

98. **Write a function to find the shortest path in a graph (Dijkstras algorithm).**
Answer: Use a priority queue.

```
from heapq import heappush, heappop
def dijkstra(graph, start):
    distances = {node: float("inf") for node in graph}
    distances[start] = 0
```

```
    pq = [(0, start)]
    while pq:
        dist, node = heappop(pq)
        if dist > distances[node]:
            continue
        for neighbor, weight in graph[node].items():
            new_dist = dist + weight
            if new_dist < distances[neighbor]:
                distances[neighbor] = new_dist
                heappush(pq, (new_dist, neighbor))
    return distances
graph = {'A': {'B': 1, 'C': 4}, 'B': {'C': 2}, 'C': {}}
print(dijkstra(graph, 'A'))  # Output: {'A': 0, 'B': 1, 'C':
   3}
```

99. **Write a function to perform a depth-first search on a graph.**
Answer: Use recursion or a stack.

```
def dfs(graph, start):
    visited = set()
    def dfs_recursive(node):
        visited.add(node)
        print(node, end=" ")
        for neighbor in graph[node]:
            if neighbor not in visited:
                dfs_recursive(neighbor)
    dfs_recursive(start)
graph = {'A': ['B', 'C'], 'B': ['D'], 'C': ['D'], 'D': []}
dfs(graph, 'A')  # Output: A B D C
```

100. **Write a function to perform a breadth-first search on a graph.**
Answer: Use a queue.

```
from collections import deque
def bfs(graph, start):
    visited = set()
    queue = deque([start])
    while queue:
        node = queue.popleft()
        if node not in visited:
            print(node, end=" ")
            visited.add(node)
            queue.extend(n for n in graph[node] if n not in
               visited)
graph = {'A': ['B', 'C'], 'B': ['D'], 'C': ['D'], 'D': []}
bfs(graph, 'A')  # Output: A B C D
```

Conclusion

These 100 questions cover a wide range of Python topics, from basics to advanced algorithms and libraries. Practice writing and explaining the code to build confidence for

your 2025 Python interview. Use platforms like LeetCode or HackerRank for additional practice.