**What is the difference between a list and a tuple?**

**Answer:**

* **List:**
  + Mutable (can be modified).
  + Syntax: my\_list = [1, 2, 3].
  + Suitable for collections that might need changes.
* **Tuple:**
  + Immutable (cannot be modified after creation).
  + Syntax: my\_tuple = (1, 2, 3).
  + Used for fixed collections of items.

**4. What are Python’s built-in data structures?**

**Answer:**

* **List:** Ordered, mutable collection of elements.
* **Tuple:** Ordered, immutable collection.
* **Set:** Unordered, unique elements.
* **Dictionary:** Key-value pairs.

Example:

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7 | my\_list = [1, 2, 3]    my\_tuple = (1, 2, 3)    my\_set = {1, 2, 3}    my\_dict = {"key1": "value1", "key2": "value2"} |

**5. What are Python’s data types?**

**Answer:**

Python provides various data types for different purposes:

* **Numeric:** int, float, complex.
* **Text:** str.
* **Sequence:** list, tuple, range.
* **Set Types:** set, frozenset.
* **Mapping:** dict.
* **Boolean:** bool.

**6. What is PEP 8?**

**Answer:**

PEP 8 is Python’s style guide for writing clean and readable code. It includes conventions such as indentation (4 spaces per level), naming variables in snake\_case, and avoiding line length exceeding 79 characters. It ensures consistency across Python projects.

**7. How does Python manage memory?**

**Answer:**

Python uses **automatic memory management**, which involves:

* **Reference Counting:** Tracks the number of references to an object.
* **Garbage Collection:** Frees memory of unused objects automatically when the reference count reaches zero.

**8. What is a Python module?**

**Answer:**

A Python module is a file containing Python code (functions, classes, and variables) that can be imported into other programs. Modules help organize code and promote reusability.

Example:

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11 | # my\_module.py    def greet():        return "Hello, Python!"    # importing the module    import my\_module    print(my\_module.greet())  # Output: Hello, Python! |

**9. What is Python’s print() function?**

**Answer:**

The print() function is used to display output in the console. You can print text, variables, and formatted strings.

Example:

|  |  |
| --- | --- |
| 1  2  3 | name = "Python"    print("Welcome to", name)  # Output: Welcome to Python |

**10. What is Python’s type() function?**

**Answer:**

The type() function returns the data type of an object.

Example:

|  |  |
| --- | --- |
| 1  2  3 | print(type(5))        # Output: <class 'int'>    print(type("hello"))  # Output: <class 'str'> |

**11. What are Python’s loops?**

**Answer:**

Python provides two main types of loops:

* **for loop:** Iterates over a sequence.
* **while loop:** Runs as long as a condition is true.

Example:

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15 | # For loop    for i in range(3):        print(i)    # While loop    x = 0    while x < 3:        print(x)        x += 1 |

**12. What are Python’s conditional statements?**

**Answer:**

Python uses if, elif, and else to execute code based on conditions.

Example:

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13 | age = 18    if age < 18:        print("Minor")    elif age == 18:        print("Just turned adult")    else:        print("Adult") |

**13. What is Python’s range() function?**

**Answer:**

The range() function generates a sequence of numbers.

Example:

|  |  |
| --- | --- |
| 1  2  3 | for i in range(5):        print(i)  # Output: 0, 1, 2, 3, 4 |

**14. How does Python handle exceptions?**

**Answer:**

Python uses try-except blocks to catch and handle errors.

Example:

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7 | try:        print(10 / 0)    except ZeroDivisionError:        print("Cannot divide by zero!") |

**15. What is the difference between is and ==?**

**Answer:**

* is compares whether two objects reference the same memory location.
* == compares the values of the objects.

Example:

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7 | a = [1, 2, 3]    print(a == b)  # True (values are equal)    b = [1, 2, 3]    print(a is b)  # False (different memory locations) |

**Intermediate Python Interview Questions and Answers**

**16. What Are Dictionaries in Python?**

Dictionaries in Python are collections of key-value pairs where each key is unique and immutable (e.g., strings, numbers), and values can be of any type. They are mutable, allowing modification, addition, or deletion of elements.

**Example:**

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9 | person = {"name": "John", "age": 30}    print(person["name"])  # Output: John    person["city"] = "New York"  # Adding a new pair    del person["age"]  # Deleting a pair    print(person)  # Output: {'name': 'John', 'city': 'New York'} |

**Key Points:**

* **Mutable**: Can be changed after creation.
* **Efficient**: O(1) average lookup time.
* **Use Cases**: Mapping, counting, and lookup tables.

**17. Explain the difference between deep copy and shallow copy.**

**Answer:**

* **Shallow Copy:** Creates a new object but inserts references to the original objects’ contents. Changes in nested objects affect both copies.
* **Deep Copy:** Creates a new object and recursively copies all objects inside it. Changes in the original do not affect the deep copy.

Example:

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11 | import copy    list1 = [[1, 2], [3, 4]]    shallow = copy.copy(list1)    deep = copy.deepcopy(list1)    shallow[0][0] = 100  # Affects list1    deep[0][0] = 200  # Does not affect list1 |

**18. What are Python decorators?**

**Answer:**A design pattern in Python that helps you to modify the behavior of a function or method without changing its code is a decorator. They are implemented as higher order functions.

Example:

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24  25  26  27 | 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 |

**19. How does Python handle memory management?**

**Answer:**Python uses automatic memory management, employing techniques such as:

* **Reference Counting:** Tracks the number of references to objects.
* **Garbage Collection:** Frees up memory for objects no longer in use, using algorithms like generational garbage collection.

**20. What is a Python generator, and how is it different from a list?**

**Answer:**Generators are iterators that produce values one at a time using the yield keyword. Unlike lists, generators do not store all values in memory, making them memory-efficient.

Example:

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11 | def gen\_numbers():        for i in range(5):            yield i    gen = gen\_numbers()    print(next(gen))  # 0    print(next(gen))  # 1 |

**21. Explain Python’s with statement.**

**Answer:**The with statement is used for resource management, ensuring proper acquisition and release of resources like file handling.

Example:

|  |  |
| --- | --- |
| 1  2  3  4  5 | with open('example.txt', 'w') as f:        f.write('Hello, World!')    # Automatically closes the file after the block |

**22. What are list comprehensions, and why are they used?**

**Answer:**Creating lists with list comprehensions is concise. Traditional for loops are more readable and often faster than this.

Example:

|  |  |
| --- | --- |
| 1  2  3 | squares = [x\*\*2 for x in range(10)]    print(squares)  # [0, 1, 4, 9, 16, 25, 36, 49, 64, 81] |

**23. How do you manage dependencies in Python projects?**

**Answer:**Dependencies can be managed using:

* pip: For installing and managing packages.
* requirements.txt: Lists all dependencies for a project.
* Virtual environments: Tools like venv or virtualenv isolate dependencies.

Example:

|  |  |
| --- | --- |
| 1  2  3 | bash    pip install -r requirements.txt |

**24. What is the difference between is and ==?**

**Answer:**

* **is:** Compares memory locations of objects.
* **==:** Compares the values of objects.

Example:

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7 | a = [1, 2, 3]    b = [1, 2, 3]    print(a == b)  # True (values are the same)    print(a is b)  # False (different memory locations) |

**25. What are Python’s magic methods?**

**Answer:**Magic methods are special methods surrounded by double underscores. Examples:

* \_\_init\_\_: Constructor for initializing objects.
* \_\_str\_\_: Provides a string representation of an object.
* \_\_add\_\_: Overloads the + operator.

Example:

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15 | class Point:        def \_\_init\_\_(self, x, y):            self.x = x            self.y = y        def \_\_str\_\_(self):            return f"Point({self.x}, {self.y})"    point = Point(1, 2)    print(point)  # Point(1, 2) |

**26. What is the purpose of the zip() function?**

**Answer:**The zip() function combines two or more iterables into tuples, stopping at the shortest iterable.

Example:

|  |  |
| --- | --- |
| 1  2  3  4  5 | a = [1, 2, 3]    b = ['x', 'y', 'z']    print(list(zip(a, b)))  # [(1, 'x'), (2, 'y'), (3, 'z')] |

**27. Explain Python’s enumerate() function.**

**Answer:**The enumerate() function adds an index to an iterable, returning tuples of the index and value.

Example:

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13 | fruits = ['apple', 'banana', 'cherry']    for index, fruit in enumerate(fruits):        print(index, fruit)    Output:    0 apple    1 banana    2 cherry |

To explore more, read this comprehensive blog on[Python enumerate()](https://www.mygreatlearning.com/blog/python-enumerate/).

**28. How do you handle exceptions in Python?**

**Answer:**Use try-except blocks for handling exceptions.

Example:

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11 | try:        result = 10 / 0    except ZeroDivisionError as e:        print(f"Error: {e}")    finally:        print("This always executes.") |

**29. What is the difference between mutable and immutable objects in Python?**

**Answer:**

* **Mutable:** Can be changed after creation (e.g., list, dict, set).
* **Immutable:** Cannot be changed after creation (e.g., int, tuple, str).

Example:

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7 | mutable\_list = [1, 2, 3]    mutable\_list[0] = 100  # Allowed    immutable\_tuple = (1, 2, 3)    # immutable\_tuple[0] = 100  # Raises TypeError |

**30. What are Python’s \*args and \*\*kwargs?**

**Answer:**

* \*args: Passes a variable number of positional arguments.
* \*\*kwargs: Passes a variable number of keyword arguments.

Example:

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11 | def greet(\*args, \*\*kwargs):        print(args)        print(kwargs)    greet("Hello", "World", name="Alice", age=25)    # ('Hello', 'World')    # {'name': 'Alice', 'age': 25} |

**Advanced Python Interview Questions and Answers**

**31. What is Python’s Global Interpreter Lock (GIL)?**

Answer:  
In Python’s CPython implementation, the GIL is a mutex that prevents multiple threads from running Python bytecode at the same time. It ensures memory management thread safe. But this doesn’t allow Python to make the most of multi core processors for CPU bound tasks. The workarounds include using multiprocessing or external libraries like NumPy for heavy computations.

**32. Explain Python’s memory management system.**

**Answer:**Python uses a combination of reference counting and garbage collection for memory management.

* **Reference Counting:** Tracks the number of references to an object. If the count drops to zero, the memory is freed.
* **Garbage Collection:** Handles circular references using generational garbage collection, categorizing objects by their lifespan for efficient cleanup.

**33. What are metaclasses in Python?**

**Answer:**Metaclasses are classes of classes that define how a class behaves. Classes are instances of metaclasses, just like objects are instances of classes. You can use metaclasses to customize class creation, enforce coding standards, or implement Singleton patterns.

Example:

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15 | class Meta(type):        def \_\_new\_\_(cls, name, bases, dct):            if "mandatory\_method" not in dct:                raise TypeError("Class must implement 'mandatory\_method'")            return super().\_\_new\_\_(cls, name, bases, dct)    class MyClass(metaclass=Meta):        def mandatory\_method(self):            pass |

**34. What is monkey patching in Python?**

**Answer:**Monkey patching refers to dynamically modifying or extending a module or class at runtime. It is often used to change behavior without altering the original source code.

Example:

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7 | import some\_module    def new\_method():        return "Patched!"    some\_module.original\_method = new\_method |

**35. Explain the difference between deep copy and shallow copy.**

**Answer:**

* **Shallow Copy:** Creates a new object but copies only references of nested objects. Changes in nested objects affect both copies.
* **Deep Copy:** Recursively copies all objects, creating independent copies.

Example:

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11 | import copy    original = [[1, 2], [3, 4]]    shallow = copy.copy(original)    deep = copy.deepcopy(original)    shallow[0][0] = 99  # Affects original    deep[0][0] = 88  # Does not affect original |

**36. What are Python’s magic methods?**

**Answer:**Magic methods are special methods with double underscores, like \_\_init\_\_, \_\_str\_\_, and \_\_len\_\_. They allow customization of object behavior for built-in operations like addition, comparison, or iteration.

Example:

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13 | class MyClass:        def \_\_init\_\_(self, value):            self.value = value        def \_\_str\_\_(self):            return f"MyClass({self.value})"        def \_\_add\_\_(self, other):            return self.value + other.value |

**37. What is the difference between @staticmethod and @classmethod?**

**Answer:**

* **@staticmethod:** Does not access the class or instance. It’s used for utility functions.
* **@classmethod:** Accesses the class itself as the first argument (cls). Used for alternative constructors.

Example:

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13 | class MyClass:        @staticmethod        def static\_method():            return "I don't use class data!"        @classmethod        def class\_method(cls):            return f"I am a method of {cls.\_\_name\_\_}" |

**38. How does Python implement multithreading?**

**Answer:**Python’s threading module provides support for threads, but due to the GIL, threads are not executed in parallel for CPU-bound tasks. They are effective for I/O-bound tasks. For true parallelism, use the multiprocessing module or external libraries like joblib.

**39. What is the purpose of Python’s \_\_slots\_\_?**

**Answer:**

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13 | \_\_slots\_\_ restricts the attributes of a class to predefined ones, saving memory by preventing the creation of \_\_dict\_\_.    Example:    class MyClass:        \_\_slots\_\_ = ['x', 'y']    obj = MyClass()    obj.x = 10  # Allowed    obj.z = 20  # Raises AttributeError |

**40. What are Python’s coroutines, and how are they used?**

**Answer:**Coroutines are functions that can pause and resume execution using the yield or await keywords. They are used in asynchronous programming to handle tasks like I/O without blocking execution.

Example:

|  |  |
| --- | --- |
| 1  2  3  4  5 | async def fetch\_data():        await asyncio.sleep(1)        return "Data fetched" |

**41. What is the difference between a process and a thread in Python?**

**Answer:**

* **Process:** Independent execution unit with its own memory space. Created using the multiprocessing module.
* **Thread:** Lightweight unit within a process sharing the same memory space. Managed using the threading module.

**42. How do you handle circular imports in Python?**

**Answer:**Circular imports occur when two modules depend on each other. Solutions include restructuring code to reduce interdependencies or using dynamic imports within functions.

**43. What are Python’s descriptors?**

**Answer:**Descriptors are objects that manage attribute access through methods like \_\_get\_\_, \_\_set\_\_, and \_\_delete\_\_. They are used in frameworks for property management, like Django models.

Example:

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9 | class Descriptor:        def \_\_get\_\_(self, instance, owner):            return "Accessed value"    class MyClass:        attr = Descriptor() |

**44. What is the difference between isinstance() and type()?**

**Answer:**

* **isinstance()** checks if an object is an instance of a class or its subclass.
* **type()** checks the exact type of an object.

Example:

|  |  |
| --- | --- |
| 1  2  3 | print(isinstance(5, int))  # True    print(type(5) is int)      # True |

**45. What is the purpose of Python’s asyncio module?**

**Answer:**asyncio provides tools for asynchronous programming, enabling non-blocking execution of I/O-bound tasks. It uses event loops to manage coroutines.

Example:

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9 | import asyncio    async def say\_hello():        await asyncio.sleep(1)        print("Hello!")    asyncio.run(say\_hello()) |

**Master Python for Free with**[**Great Learning**](https://www.mygreatlearning.com/)**!**

### What are Python lists and tuples?

Lists and tuples are fundamental Python data structures with distinct characteristics and use cases.

**List:**

* **Mutable:** Elements can be changed after creation.
* **Memory Usage:** Consumes more memory.
* **Performance:** Slower iteration compared to tuples but better for insertion and deletion operations.
* **Methods:** Offers various built-in methods for manipulation.

**Example:**

a\_list = ["Data", "Camp", "Tutorial"]

a\_list.append("Session")

print(a\_list) # Output: ['Data', 'Camp', 'Tutorial', 'Session']

[**Powered By**](https://www.datacamp.com/datalab)

**Tuple:**

* **Immutable:** Elements cannot be changed after creation.
* **Memory Usage:** Consumes less memory.
* **Performance:** Faster iteration compared to lists but lacks the flexibility of lists.
* **Methods:** Limited built-in methods.

**Example:**

a\_tuple = ("Data", "Camp", "Tutorial")

print(a\_tuple) # Output: ('Data', 'Camp', 'Tutorial')

[**Powered By**](https://www.datacamp.com/datalab)

Learn more in our [**Python Lists tutorial**](https://www.datacamp.com/blog/top-python-interview-questions-and-answers).

### 3. What is \_\_init\_\_() in Python?

The \_\_init\_\_() method is known as a constructor in object-oriented programming (OOP) terminology. It is used to initialize an object's state when it is created. This method is automatically called when a new instance of a class is instantiated.

**Purpose:**

* Assign values to object properties.
* Perform any initialization operations.

**Example**:

We have created a book\_shop class and added the constructor and book() function. The constructor will store the book title name and the book() function will print the book name.

To test our code we have initialized the b object with “Sandman” and executed the book() function.

class book\_shop:

# constructor

def \_\_init\_\_(self, title):

self.title = title

# Sample method

def book(self):

print('The tile of the book is', self.title)

b = book\_shop('Sandman')

b.book()

# The tile of the book is Sandman

[**Powered By**](https://www.datacamp.com/datalab)

### 4. What is the difference between a mutable data type and an immutable data type?

#### Mutable data types:

* **Definition:** Mutable data types are those that can be modified after their creation.
* **Examples:** List, Dictionary, Set.
* **Characteristics:** Elements can be added, removed, or changed.
* **Use Case:** Suitable for collections of items where frequent updates are needed.

**Example:**

# List Example

a\_list = [1, 2, 3]

a\_list.append(4)

print(a\_list) # Output: [1, 2, 3, 4]

# Dictionary Example

a\_dict = {'a': 1, 'b': 2}

a\_dict['c'] = 3

print(a\_dict) # Output: {'a': 1, 'b': 2, 'c': 3}

[**Powered By**](https://www.datacamp.com/datalab)

#### ****Immutable data types:****

* **Definition:** Immutable data types are those that cannot be modified after their creation.
* **Examples:** Numeric (int, float), String, Tuple.
* **Characteristics:** Elements cannot be changed once set; any operation that appears to modify an immutable object will create a new object.

**Example:**

# Numeric Example

a\_num = 10

a\_num = 20 # Creates a new integer object

print(a\_num) # Output: 20

# String Example

a\_str = "hello"

a\_str = "world" # Creates a new string object

print(a\_str) # Output: world

# Tuple Example

a\_tuple = (1, 2, 3)

# a\_tuple[0] = 4 # This will raise a TypeError

print(a\_tuple) # Output: (1, 2, 3)

[**Powered By**](https://www.datacamp.com/datalab)

### 5. Explain list, dictionary, and tuple comprehension with an example.

#### List

List comprehension offers one-liner syntax to create a new list based on the values of the existing list. You can use a for loop to replicate the same thing, but it will require you to write multiple lines, and sometimes it can get complex.

List comprehension eases the creation of the list based on existing iterable.

my\_list = [i for i in range(1, 10)]

my\_list

# [1, 2, 3, 4, 5, 6, 7, 8, 9]

[**Powered By**](https://www.datacamp.com/datalab)

#### Dictionary

Similar to a List comprehension, you can create a dictionary based on an existing table with a single line of code. You need to enclose the operation with curly brackets {}.

# Creating a dictionary using dictionary comprehension

my\_dict = {i: i\*\*2 for i in range(1, 10)}

# Output the dictionary

my\_dict

{1: 1, 2: 4, 3: 9, 4: 16, 5: 25, 6: 36, 7: 49, 8: 64, 9: 81}

[**Powered By**](https://www.datacamp.com/datalab)

#### Tuple

Unlike lists and dictionaries, there is no special “tuple comprehension.”

**When you use parentheses with a comprehension, Python actually creates a generator expression, not a tuple. To get a tuple, you must either convert the generator with** tuple() **or define a tuple literal directly.**

# Generator expression (not a tuple)

my\_gen = (i for i in range(1, 10))

my\_gen

# <generator object <genexpr> ...>

# Converting generator to tuple

my\_tuple = tuple(i for i in range(1, 10))

my\_tuple

# (1, 2, 3, 4, 5, 6, 7, 8, 9)

# Or simply define a tuple directly

literal\_tuple = (1, 2, 3)

literal\_tuple

# (1, 2, 3)

[**Powered By**](https://www.datacamp.com/datalab)

You can learn more about it in our [**Python Tuples tutorial**](https://www.datacamp.com/tutorial/python-tuples).

### 6. What is the Global Interpreter Lock (GIL) in Python, and why is it important?

The Global Interpreter Lock (GIL) is a mutex in CPython (the reference Python implementation) that ensures only one native thread executes Python bytecode at a time. It simplifies memory management by protecting internal data structures like reference counts, but it also restricts true parallelism in CPU-bound tasks, making multithreading less effective for computational workloads. However, it works well for I/O-bound tasks, where threads spend time waiting on network, file, or database operations.

Note**: Python 3.13 introduced an experimental no-GIL build (PEP 703), and Python 3.14 adds documented free-threaded support. Some C extensions and libraries may not yet be fully compatible.**

## Intermediate Python Interview Questions

Here are some of the questions you might encounter during an intermediate-level Python interview.

### 7. Can you e****xplain common searching and graph traversal algorithms in Python?****

Python has a number of different powerful algorithms for searching and graph traversal, and each one deals with different data structures and solves different problems. I can them here:

* **Binary Search**: If you need to quickly find an item in a sorted list, [**binary search**](https://www.datacamp.com/tutorial/binary-search-python) is your go-to. It works by repeatedly dividing the search range in half until the target is found.
* **AVL Tree**: An [**AVL tree**](https://www.datacamp.com/tutorial/avl-tree) keeps things balanced, which is a big advantage if you’re frequently inserting or deleting items in a tree. This self-balancing binary search tree structure keeps searches fast by making sure the tree never gets too skewed.
* **Breadth-First Search (BFS)**: [**BFS**](https://www.datacamp.com/tutorial/breadth-first-search-in-python) is all about exploring a graph level by level. It’s especially useful if you’re trying to find the shortest path in an unweighted graph since it checks all possible moves from each node before going deeper.
* **Depth-First Search (DFS)**: [**DFS**](https://www.datacamp.com/tutorial/depth-first-search-in-python) takes a different approach by exploring as far as it can down each branch before backtracking. It’s great for tasks like maze-solving or tree traversal.
* **A Algorithm\***: The [**A\* algorithm**](https://www.datacamp.com/tutorial/a-star-algorithm) is a bit more advanced and combines the best of both BFS and DFS by using heuristics to find the shortest path efficiently. It’s commonly used in pathfinding for maps and games.

### 8. What is a KeyError in Python, and how can you handle it?

A KeyError in Python occurs when you try to access a key that doesn’t exist in a dictionary. This error is raised because Python expects every key you look up to be present in the dictionary, and when it isn’t, it throws a KeyError.

For example, if you have a dictionary of student scores and try to access a student who isn’t in the dictionary, you’ll get a KeyError. To handle this error, you have a few options:

* **Use the .get() method**: This method returns None (or a specified default value) instead of throwing an error if the key isn’t found.
* **Use a try-except block**: Wrapping your code in try-except allows you to catch the KeyError and handle it gracefully.
* **Check for the key with in:** You can check if a key exists in the dictionary using if key in dictionary before trying to access it.

To learn more, read our full tutorial: [**Python KeyError Exceptions and How to Fix Them**](https://www.datacamp.com/tutorial/python-keyerror).

### 9. How does Python handle memory management, and what role does garbage collection play?

Python manages memory allocation and deallocation automatically using a private heap, where all objects and data structures are stored. The memory management process is handled by Python’s memory manager, which optimizes memory usage, and the garbage collector, which deals with unused or unreferenced objects to free up memory.

[**Garbage collection in Python**](https://www.datacamp.com/tutorial/python-garbage-collection) uses reference counting as well as a cyclic garbage collector to detect and collect unused data. When an object has no more references, it becomes eligible for garbage collection. The gc module in Python allows you to interact with the garbage collector directly, providing functions to enable or disable garbage collection, as well as to perform manual collection.

### 10. What is the difference between shallow copy and deep copy in Python, and when would you use each?

In Python, shallow and deep copies are used to duplicate objects, but they handle nested structures differently.

* **Shallow Copy**: A shallow copy creates a new object but inserts references to the objects found in the original. So, if the original object contains other mutable objects (like lists within lists), the shallow copy will reference the same inner objects. This can lead to unexpected changes if you modify one of those inner objects in either the original or copied structure. You can create a shallow copy using the copy() method or the copy module’s copy() function.
* **Deep Copy**: A deep copy creates a new object and recursively copies all objects found within the original. This means that even nested structures get duplicated, so changes in one copy don’t affect the other. To create a deep copy, you can use the copy module’s deepcopy() function.

**Example Usage**: A shallow copy is suitable when the object contains only immutable items or when you want changes in nested structures to reflect in both copies. A deep copy is ideal when working with complex, nested objects where you want a completely independent duplicate. Read our [**Python Copy List: What You Should Know**](https://www.datacamp.com/tutorial/python-copy-list) tutorial to learn more. This tutorial includes a whole section on the difference between shallow copy and deep copy.

### 11. How can you use Python’s collections module to simplify common tasks?

The collections module in Python provides specialized data structures like defaultdict, Counter, deque, and OrderedDict to simplify various tasks. For instance, Counter is ideal for counting elements in an iterable, while defaultdict can initialize dictionary values without explicit checks.

Example:

from collections import Counter

data = ['a', 'b', 'c', 'a', 'b', 'a']

count = Counter(data)

print(count) # Output: Counter({'a': 3, 'b': 2, 'c': 1})

[**Powered By**](https://www.datacamp.com/datalab)

## Advanced Python Interview Questions

These interview questions are for more experienced Python practitioners.

### 12. What is monkey patching in Python?

Monkey patching in Python is a dynamic technique that can change the behavior of the code at run-time. In short, you can modify a class or module at run-time.

**Example**:

Let’s learn monkey patching with an example.

1. We have created a class monkey with a patch() function. We have also created a monk\_p function outside the class.
2. We will now replace the patch with the monk\_p function by assigning monkey.patch to monk\_p.
3. In the end, we will test the modification by creating the object using the monkey class and running the patch() function.

Instead of displaying patch() is being called, it has displayed monk\_p() is being called.

class monkey:

def patch(self):

print ("patch() is being called")

def monk\_p(self):

print ("monk\_p() is being called")

# replacing address of "patch" with "monk\_p"

monkey.patch = monk\_p

obj = monkey()

obj.patch()

# monk\_p() is being called

[**Powered By**](https://www.datacamp.com/datalab)

**Caution: Use these sparingly; monkey patching can make your code harder to read and may surprise others working with your code or tests.**

### 13. What is the Python “with” statement designed for?

The with statement is used for exception handling to make code cleaner and simpler. It is generally used for the management of common resources like creating, editing, and saving a file.

**Example**:

Instead of writing multiple lines of open, try, finally, and close, you can create and write a text file using the with statement. It is simple.

# using with statement

with open('myfile.txt', 'w') as file:

file.write('DataCamp Black Friday Sale!!!')

[**Powered By**](https://www.datacamp.com/datalab)

### 14. Why use else in try/except construct in Python?

try: and except: are commonly known for exceptional handling in Python, so where does else: come in handy? else: will be triggered when no exception is raised.

**Example**:

Let’s learn more about else: with a couple of examples.

1. On the first try, we entered 2 as the numerator and d as the denominator. Which is incorrect, and except: was triggered with “Invalid input!”.
2. On the second try, we entered 2 as the numerator and 1 as the denominator and got the result 2. No exception was raised, so it triggered the else: printing the message Division is successful.

try:

num1 = int(input('Enter Numerator: '))

num2 = int(input('Enter Denominator: '))

division = num1/num2

print(f'Result is: {division}')

except:

print('Invalid input!')

else:

print('Division is successful.')

## Try 1 ##

# Enter Numerator: 2

# Enter Denominator: d

# Invalid input!

## Try 2 ##

# Enter Numerator: 2

# Enter Denominator: 1

# Result is: 2.0

# Division is successful.

[**Powered By**](https://www.datacamp.com/datalab)

Take the [**Python Fundamentals**](https://www.datacamp.com/tracks/python-fundamentals) skill track to gain the foundational skills you need to become a Python programmer.

### 15. What are decorators in Python?

Decorators in Python are a design pattern that allows you to add new functionality to an existing object without modifying its structure. They are commonly used to extend the behavior of functions or methods. You can read more about [**how to use Python decorators**](https://www.datacamp.com/tutorial/decorators-python) in a separate guide.

**Example:**

import functools

def my\_decorator(func):

@functools.wraps(func) # preserves \_\_name\_\_, \_\_doc\_\_, etc.

def wrapper(\*args, \*\*kwargs):

print("Something is happening before the function is called.")

result = func(\*args, \*\*kwargs)

print("Something is happening after the function is called.")

return result

return wrapper

@my\_decorator

def say\_hello():

print("Hello!")

say\_hello()

# Output:

# Something is happening before the function is called.

# Hello!

# Something is happening after the function is called.

[**Powered By**](https://www.datacamp.com/datalab)

### ****16. What are context managers in Python, and how are they implemented?****

Context managers in Python are used to manage resources, ensuring that they are properly acquired and released. The most common use of context managers is the with statement.

**Example:**

class FileManager:

def \_\_init\_\_(self, filename, mode):

self.filename = filename

self.mode = mode

def \_\_enter\_\_(self):

self.file = open(self.filename, self.mode)

return self.file

def \_\_exit\_\_(self, exc\_type, exc\_value, traceback):

self.file.close()

with FileManager('test.txt', 'w') as f:

f.write('Hello, world!')

[**Powered By**](https://www.datacamp.com/datalab)

In this example, the FileManager class is a context manager that ensures the file is properly closed after it is used within the with statement.

### 17. What are metaclasses in Python, and how do they differ from regular classes?

Metaclasses are classes of classes. They define how classes behave and are created. While regular classes create objects, metaclasses create classes. By using metaclasses, you can modify class definitions, enforce rules, or add functionality during class creation.

Example:

class Meta(type):

def \_\_new\_\_(cls, name, bases, dct):

print(f"Creating class {name}")

return super().\_\_new\_\_(cls, name, bases, dct)

class MyClass(metaclass=Meta):

pass

# Output: Creating class MyClass

[**Powered By**](https://www.datacamp.com/datalab)

## Python Data Science Interview Questions

For those focused more on data science applications of Python, these are some questions you may encounter.

### 18. What are the advantages of NumPy over regular Python lists?

There are several advantages of NumPy over regular Python lists, such as:

* **Memory**: NumPy arrays are more memory-efficient than Python lists because they store elements of the same type in contiguous blocks. (Exact memory use depends on element type and system, but you can check with sys.getsizeof or array.nbytes.)
* **Speed**: NumPy uses optimized C implementations, so operations on large arrays are much faster than with lists.
* **Versatility: NumPy supports vectorized operations (e.g., addition, multiplication) and provides many built-in mathematical functions that Python lists don’t support.**

### 19. What is the difference between merge, join, and concatenate?

#### Merge

Merge two DataFrames named series objects using the unique column identifier.

It requires two DataFrame, a common column in both DataFrame, and “how” you want to join them together. You can left, right, outer, inner, and cross join two data DataFrames. By default, it is an inner join.

pd.merge(df1, df2, how='outer', on='Id')

[**Powered By**](https://www.datacamp.com/datalab)

#### Join

[**Join the DataFrames**](https://www.datacamp.com/tutorial/joining-dataframes-pandas) using the unique index. It requires an optional on argument that can be a column or multiple column names. By default, the join function performs a left join.

df1.join(df2)

[**Powered By**](https://www.datacamp.com/datalab)

#### Concatenate

[**Concatenate**](https://www.datacamp.com/tutorial/python-concatenate-strings) joins two or multiple DataFrames along a particular axis (rows or columns). It doesn't require an on argument.

pd.concat(df1,df2)

[**Powered By**](https://www.datacamp.com/datalab)

* **join()**: combines two DataFrames by index.
* **merge()**: combines two DataFrames by the column or columns you specify.
* **concat()**: combines two or more DataFrames vertically or horizontally.

### 20. How do you identify and deal with missing values?

#### Identifying missing values

We can identify missing values in the DataFrame by using the isnull() function and then applying sum(). Isnull() will return boolean values, and the sum will give you the number of missing values in each column.

In the example, we have created a dictionary of lists and converted it into a pandas DataFrame. After that, we used isnull().sum() to get the number of missing values in each column.

import pandas as pd

import numpy as np

# dictionary of lists

dict = {'id':[1, 4, np.nan, 9],

'Age': [30, 45, np.nan, np.nan],

'Score':[np.nan, 140, 180, 198]}

# creating a DataFrame

df = pd.DataFrame(dict)

df.isnull().sum()

# id 1

# Age 2

# Score 1

[**Powered By**](https://www.datacamp.com/datalab)

#### Dealing with missing values

There are various [**ways of dealing with missing values in Python**](https://www.datacamp.com/tutorial/python-nan-missing-values-in-python).

1. Drop the entire row or the columns if it consists of missing values using dropna(). This method is not recommended, as you will lose important information.
2. Fill the missing values with the constant, average, backward fill, and forward fill using the fillna() function.
3. Replace missing values with a constant String, Integer, or Float using the replace() function.
4. Fill in the missing values using an interpolation method.

**Note**: make sure you are working with a larger dataset while using the dropna() function.

# drop missing values

df.dropna(axis = 0, how ='any')

#fillna

df.fillna(method ='bfill')

#replace null values with -999

df.replace(to\_replace = np.nan, value = -999)

# Interpolate

df.interpolate(method ='linear', limit\_direction ='forward')

[**Powered By**](https://www.datacamp.com/datalab)

Become a professional data scientist by taking the [**Data Scientist with Python**](https://www.datacamp.com/tracks/associate-data-scientist-in-python) career track. It includes 25 courses and six projects to help you learn all the fundamentals of data science with the help of Python libraries.

### 21. Which all Python libraries have you used for visualization?

Data visualization is the most important part of data analysis. You get to see your data in action, and it helps you find hidden patterns.

The most popular Python data visualization libraries are:

1. Matplotlib
2. Seaborn
3. Plotly
4. Bokeh

In Python, we generally use **Matplotlib** and **seaborn** to display all types of data visualization. With a few lines of code, you can use it to display scatter plot, line plot, box plot, bar chart, and many more.

For interactive and more complex applications, we use **Plotly**. You can use it to create colorful interactive graphs with a few lines of code. You can zoom, apply animation, and even add control functions. Plotly provides more than 40 unique types of charts, and we can even use them to create a web application or dashboard.

**Bokeh** is used for detailed graphics with a high level of interactivity across large datasets.

### 22. How would you normalize or standardize a dataset in Python?

Normalization scales data to a specific range, usually [0, 1], while standardization transforms it to have a mean of 0 and a standard deviation of 1. Both techniques are essential for preparing data for machine learning models.

Example:

from sklearn.preprocessing import MinMaxScaler, StandardScaler

import numpy as np

data = np.array([[1, 2], [3, 4], [5, 6]])

# Normalize

normalizer = MinMaxScaler()

normalized = normalizer.fit\_transform(data)

print(normalized)

# Standardize

scaler = StandardScaler()

standardized = scaler.fit\_transform(data)

print(standardized)

[**Powered By**](https://www.datacamp.com/datalab)

## Python Coding Interview Questions

If you have a Python coding interview coming up, preparing questions similar to these can help you impress the interviewer.

### 23. How can you replace string space with a given character in Python?

It is a simple string manipulation challenge. You have to replace the space with a specific character.

**Example 1**: A user has provided the string l vey u and the character o, and the output will be loveyou.

**Example 2**: A user has provided the string D t C mpBl ckFrid yS le and the character a, and the output will be DataCampBlackFridaySale.

**The simplest way is to use the built-in** str.replace() **method to directly replace spaces with the given character.**

def str\_replace(text, ch):

return text.replace(" ", ch)

text = "D t C mpBl ckFrid yS le"

ch = "a"

str\_replace(text, ch)

# 'DataCampBlackFridaySale'

[**Powered By**](https://www.datacamp.com/datalab)

### 24. Given a positive integer num, write a function that returns True if num is a perfect square else False.

This has a relatively straightforward solution. You can check if the number has a perfect square root by:

* Using math.isqrt(num) to get the integer square root exactly.
* Squaring it and checking if it equals the original number.
* Returning the result as a boolean.

#### Test 1

We have provided number 10 to the valid\_square() function:

1. By taking the integer square root of the number, we get 3.
2. Then, take the square of 3 and get 9.
3. 9 is not equal to the number, so the function will return False.

#### Test 2

We have provided number 36 to the valid\_square() function:

1. By taking the integer square root of the number, we get 6.
2. Then, take the square of 6 and get 36.
3. 36 is equal to the number, so the function will return True.

import math

def valid\_square(num):

if num < 0:

return False

square = math.isqrt(num)

return square \* square == num

valid\_square(10)

# False

valid\_square(36)

# True

[**Powered By**](https://www.datacamp.com/datalab)

### 25. Given an integer n, return the number of trailing zeroes in n factorial n!

To pass this challenge, you have to first calculate n factorial (n!) and then calculate the number of training zeros.

#### Finding factorial

In the first step, we will use a while loop to iterate over the n factorial and stop when the n is equal to 1.

#### Calculating trailing zeros

In the second step, we will calculate the trailing zero, not the total number of zeros. There is a huge difference.

7! = 5040

[**Powered By**](https://www.datacamp.com/datalab)

The seven factorials have a total of two zeros and only one trailing zero, so our solution should return 1.

1. Convert the factorial number to a string.
2. Read it back and apply for a loop.
3. If the number is 0, add +1 to the result, otherwise break the loop.
4. Returns the result.

The solution is elegant but requires attention to detail.

def factorial\_trailing\_zeros(n):

fact = n

while n > 1:

fact \*= n - 1

n -= 1

result = 0

for i in str(fact)[::-1]:

if i == "0":

result += 1

else:

break

return result

factorial\_trailing\_zeros(10)

# 2

factorial\_trailing\_zeros(18)

# 3

[**Powered By**](https://www.datacamp.com/datalab)

Take the essential [**practicing coding interview questions**](https://www.datacamp.com/courses/practicing-coding-interview-questions-in-python?hl=GB) course to prepare for your next coding interviews in Python.

### 26. Can the String Be Split into Dictionary Words?

**You are provided with a large string and a dictionary of the words. You have to find if the input string can be segmented into words using the dictionary or not.**

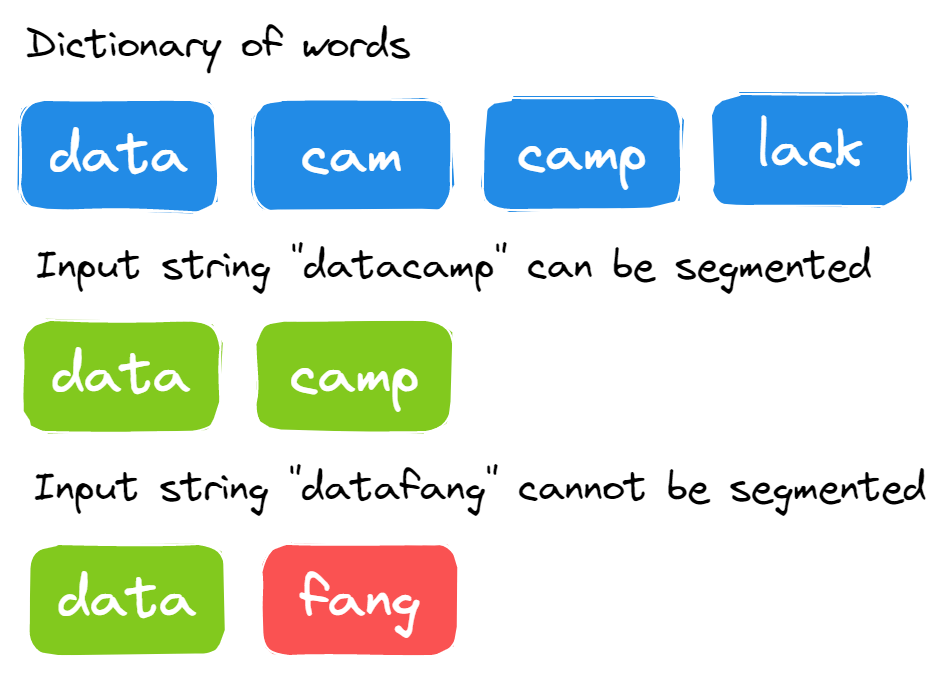


Image by Author

The solution is reasonably straightforward. You have to segment a large string at each point and check if the string can be segmented to the words in the dictionary.

1. Run the loop using the length of the large string.
2. We will create two substrings.
3. The first substring will check each point in the large string from s[0:i].
4. If the first substring is not in the dictionary, it will return False.
5. If the first substring is in the dictionary, it will create the second substring using s[i:].
6. If the second substring is in the dictionary or the second substring is of zero length, then return True. Recursively call can\_segment\_str() with the second substring and return True if it can be segmented.
7. To make the solution efficient for longer strings, we add memoization so substrings are not recomputed again and again.

def can\_segment\_str(s, dictionary, memo=None):

if memo is None:

memo = {}

if s in memo:

return memo[s]

if not s:

return True

for i in range(1, len(s) + 1):

first\_str = s[0:i]

if first\_str in dictionary:

second\_str = s[i:]

if (

not second\_str

or second\_str in dictionary

or can\_segment\_str(second\_str, dictionary, memo)

):

memo[s] = True

return True

memo[s] = False

return False

s = "datacamp"

dictionary = ["data", "camp", "cam", "lack"]

can\_segment\_str(s, dictionary)

# True

[**Powered By**](https://www.datacamp.com/datalab)

### 27. Can you remove duplicates from a sorted array?

**Given an integer sorted array in increasing order, remove duplicates so each unique element appears only once. Because Python lists don’t change length in-place for this problem, place the results in the first k positions of the same array and return k (the new length). Only the first k elements are valid after the call; elements beyond k are stale.**

Image from **[LeetCode](https://leetcode.com/problems/remove-duplicates-from-sorted-array/solution/" \t "_blank)**

**Example 1**: input array is [1,1,2,2], the function should return 2.

**Example 2**: input array is [1,1,2,3,3], the function should return 3.

Solution:

1. Run a loop from index 1 to the end. Compare the current element with the previous unique element; when different, write it at insertIndex and increment insertIndex. Return insertIndex.
2. Return insertIndex as it is the k.

This question is relatively straightforward once you know how. If you put more time into understanding the statement, you can easily come up with a solution.

def removeDuplicates(array):

size = len(array)

if size == 0:

return 0

insertIndex = 1

for i in range(1, size):

if array[i - 1] != array[i]:

array[insertIndex] = array[i]

insertIndex += 1

return insertIndex

array\_1 = [1, 2, 2, 3, 3, 4]

k1 = removeDuplicates(array\_1)

# 4; array\_1[:k1] -> [1, 2, 3, 4]

array\_2 = [1, 1, 3, 4, 5, 6, 6]

k2 = removeDuplicates(array\_2)

# 5; array\_2[:k2] -> [1, 3, 4, 5, 6]

[**Powered By**](https://www.datacamp.com/datalab)

### 28. Can you find the missing number in the array?

You have been provided with the list of positive integers from 1 to n. All the numbers from 1 to n are present except x, and you must find x.

**Example**:

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| **4** | **5** | **3** | **2** | **8** | **1** | **6** |

* n = 8
* missing number = 7

This question is a simple math problem.

1. Find the sum of all elements in the list.
2. By using arithmetic series sum formula, we will find the expected sum of the first n numbers.
3. Return the difference between the expected sum and the sum of the elements.

def find\_missing(input\_list):

sum\_of\_elements = sum(input\_list)

# There is exactly 1 number missing

n = len(input\_list) + 1

actual\_sum = (n \* ( n + 1 ) ) / 2

return int(actual\_sum - sum\_of\_elements)

list\_1 = [1,5,6,3,4]

find\_missing(list\_1)

# 2

[**Powered By**](https://www.datacamp.com/datalab)

### 29. Write a Python function to determine if a given string is a palindrome.

A string is a palindrome if it reads the same forward and backward.

Example:

def is\_palindrome(s):

s = ''.join(e for e in s if e.isalnum()).lower() # Remove non-alphanumeric and convert to lowercase

return s == s[::-1]

print(is\_palindrome("A man, a plan, a canal: Panama")) # Output: True

print(is\_palindrome("hello")) # Output: False

[**Powered By**](https://www.datacamp.com/datalab)

## Python Interview Questions for Facebook, Amazon, Apple, Netflix, and Google

Below, we’ve picked out some of the questions you might expect from the most sought-after roles in the industries, those at Meta, Amazon, Google, and the like.

#### Facebook/Meta Python interview questions

The exact questions you’ll encounter at Meta depend largely on the role. However, you might expect some of the following:

### 30. Can you find the maximum single sell profit?

You are provided with the list of stock prices, and you have to return the buy and sell price to make the highest profit.

**Note**: We have to make maximum profit from a single buy/sell, and if we can’t make a profit, we have to reduce our losses.

**Example 1**: stock\_price = [8, 4, 12, 9, 20, 1], buy = 4, and sell = 20. Maximizing the profit.

**Example 2**: stock\_price = [8, 6, 5, 4, 3, 2, 1], buy = 6, and sell = 5. Minimizing the loss.

**Solution**:

1. We will calculate the global profit by subtracting global sell (the first element in the list) from current buy (the second element in the list).
2. Run the loop for the range of 1 to the length of the list.
3. Within the loop, calculate the current profit using list elements and current buy value.
4. If the current profit is greater than the global profit, change the global profit with the current profit and global sell to the i element of the list.
5. If the current buy is greater than the current element of the list, change the current buy with the current element of the list.
6. In the end, we will return global buy and sell value. To get global buy value, we will subtract global sell from global profit.

The question is a bit tricky, and you can come up with your unique algorithm to solve the problems.

def buy\_sell\_stock\_prices(stock\_prices):

current\_buy = stock\_prices[0]

global\_sell = stock\_prices[1]

global\_profit = global\_sell - current\_buy

for i in range(1, len(stock\_prices)):

current\_profit = stock\_prices[i] - current\_buy

if current\_profit > global\_profit:

global\_profit = current\_profit

global\_sell = stock\_prices[i]

if current\_buy > stock\_prices[i]:

current\_buy = stock\_prices[i]

return global\_sell - global\_profit, global\_sell

stock\_prices\_1 = [10,9,16,17,19,23]

buy\_sell\_stock\_prices(stock\_prices\_1)

# (9, 23)

stock\_prices\_2 = [8, 6, 5, 4, 3, 2, 1]

buy\_sell\_stock\_prices(stock\_prices\_2)

# (6, 5)

[**Powered By**](https://www.datacamp.com/datalab)

#### Amazon Python interview questions

Amazon Python interview questions can vary greatly but could include:

### 31. Can you find a Pythagorean triplet in an array?

Write a function that returns True if there is a Pythagorean triplet that satisfies a2+ b2 = c2.

**Example**:

|  |  |
| --- | --- |
| **Input** | **Output** |
| [3, 1, 4, 6, 5] | True |
| [10, 4, 6, 12, 5] | False |

**Solution**:

1. Square all the elements in the array.
2. Sort the array in increasing order.
3. Run two loops. The outer loop starts from the last index of the array to 1, and the inner loop starts from (outer\_loop\_index - 1) to the start.
4. Create set() to store the elements between outer loop index and inner loop index.
5. Check if there is a number present in the set which is equal to (array[outerLoopIndex] – array[innerLoopIndex]). If yes, return True, else False.

def checkTriplet(array):

n = len(array)

for i in range(n):

array[i] = array[i]\*\*2

array.sort()

for i in range(n - 1, 1, -1):

s = set()

for j in range(i - 1, -1, -1):

if (array[i] - array[j]) in s:

return True

s.add(array[j])

return False

arr = [3, 2, 4, 6, 5]

checkTriplet(arr)

# True

[**Powered By**](https://www.datacamp.com/datalab)

### 32. How many ways can you make change with coins and a total amount?

We need to create a function that takes a list of coin denominations and a total amount and returns the number of ways we can make the change.

**In the example, we have provided coin denominations [1, 2, 5] and the total amount of 5. In return, we get four ways to make the change.**

Image by Author

**Solution**:

1. We will create the list of size amount + 1. Additional space is added to store the solution for a zero amount.
2. We will initiate a solution list with solution[0] = 1.
3. We will run two loops. The outer loop iterates over the denominations, and the inner loop runs from the current denomination value to amount + 1.
4. The results of different denominations are stored in the array solution. solution[i] = solution[i] + solution[i - den].

**The process will be repeated for all the elements in the denomination list, and at the last element of the solution list, we will have our number.**

def solve\_coin\_change(denominations, amount):

solution = [0] \* (amount + 1)

solution[0] = 1

for den in denominations:

for i in range(den, amount + 1):

solution[i] += solution[i - den]

return solution[amount]

denominations = [1, 2, 5]

amount = 5

solve\_coin\_change(denominations, amount)

# 4

[**Powered By**](https://www.datacamp.com/datalab)

#### Google Python interview questions

As with the other companies mentioned, Google Python interview questions will depend on the role and level of experience. However, some common questions include:

### 33. Define a lambda function, an iterator, and a generator in Python.

The Lambda function is also known as an anonymous function. You can add any number of parameters but with only one statement.

An iterator is an object that we can use to iterate over iterable objects like lists, dictionaries, tuples, and sets.

The generator is a function similar to a normal function, but it generates a value using the yield keyword instead of return. If the function body contains yield, it automatically becomes a generator.

Read more about [**Python iterators and generators**](https://www.datacamp.com/tutorial/python-iterators-generators-tutorial) in our full tutorial.

### 34. Given an array arr[], find the maximum j – i such that arr[j] > arr[i]

This question is quite straightforward but requires special attention to detail. We are provided with an array of positive integers. We have to find the maximum difference between j-i where array[j] > array[i].

**Examples**:

1. Input: [20, 70, 40, 50, 12, 38, 98], Output: 6  (j = 6, i = 0)
2. Input: [10, 3, 2, 4, 5, 6, 7, 8, 18, 0], Output: 8 ( j = 8, i = 0)

**Solution**:

1. Calculate the length of the array and initiate max difference with -1.
2. Run two loops. The outer loop picks elements from the left, and the inner loop compares the picked elements with elements starting from the right side.
3. Stop the inner loop when the element is greater than the picked element and keep updating the maximum difference using j - I.

def max\_index\_diff(array):

n = len(array)

max\_diff = -1

for i in range(0, n):

j = n - 1

while(j > i):

if array[j] > array[i] and max\_diff < (j - i):

max\_diff = j - i

j -= 1

return max\_diff

array\_1 = [20,70,40,50,12,38,98]

max\_index\_diff(array\_1)

# 6

[**Powered By**](https://www.datacamp.com/datalab)

### 35. How would you use the ternary operators in Python?

Ternary operators are also known as conditional expressions. They are operators that evaluate expression based on conditions being True and False.

You can write conditional expressions in a single line instead of writing using multiple lines of if-else statements. It allows you to write clean and compact code.

For example, we can convert nested if-else statements into one line, as shown below.

If-else statement

score = 75

if score < 70:

if score < 50:

print('Fail')

else:

print('Merit')

else:

print('Distinction')

# Distinction

[**Powered By**](https://www.datacamp.com/datalab)

Nested Ternary Operator

print('Fail' if score < 50 else 'Merit' if score < 70 else 'Distinction')

# Distinction

[**Powered By**](https://www.datacamp.com/datalab)

### ****36. How would you implement an LRU Cache in Python?****

Python provides a built-in functools.lru\_cache decorator to implement an LRU (Least Recently Used) cache. Alternatively, you can create one manually using the OrderedDict from collections.

Example using functools:

from functools import lru\_cache

@lru\_cache(maxsize=3)

def add(a, b):

return a + b

print(add(1, 2)) # Calculates and caches result

print(add(1, 2)) # Retrieves result from cache

[**Powered By**](https://www.datacamp.com/datalab)

## ****Upskilling Your Team with Python****