**1. What are Python’s key features?**

* Interpreted language
* Dynamically typed
* Object-oriented
* Extensive standard library
* Platform-independent

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

* == compares **values**.
* is compares **object identities**.

a = [1, 2]; b = a

print(a == b) # True

print(a is b) # True

c = [1, 2]

print(a == c) # True

print(a is c) # False

**3. What are Python’s data types?**

* int, float, bool, str
* list, tuple, set, dict

**4. What is a Python list?**

* A mutable, ordered collection of items.

my\_list = [1, 2, 3]

**5. What are Python dictionaries?**

* Key-value pair mappings.

my\_dict = {"name": "Remya", "age": 30}

**6. What is list comprehension?**

A concise way to create lists.

squares = [x\*\*2 for x in range(5)]

**7. \*\*What are \*args and kwargs?**

* \*args → variable number of positional arguments.
* \*\*kwargs → variable number of keyword arguments.

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

print(args)

print(kwargs)

**8. What is the difference between a shallow copy and a deep copy?**

* **Shallow copy**: references nested objects.
* **Deep copy**: copies all nested objects.

import copy

shallow = copy.copy(obj)

deep = copy.deepcopy(obj)

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

| **Feature** | **List** | **Tuple** |
| --- | --- | --- |
| Mutability | Mutable | Immutable |
| Syntax | [] | () |
| Performance | Slower | Faster |

**10. What is a lambda function?**

An anonymous, one-line function.

add = lambda x, y: x + y

**11. What are decorators in Python?**

Functions that modify the behavior of another function.

def decorator(func):

def wrapper():

print("Before")

func()

print("After")

return wrapper

@decorator

def say\_hello():

print("Hello")

**12. Explain Python’s GIL (Global Interpreter Lock).**

* A mutex that allows only one thread to execute Python bytecode at a time.
* Affects multi-threading performance in CPU-bound tasks.

**13. What is a generator in Python?**

A function that uses yield to return values one at a time, saving state between calls.

def gen():

yield 1

yield 2

**14. Explain Python memory management.**

* Python uses reference counting and garbage collection (GC) for memory management.
* gc module is used for monitoring.

**15. What are Python modules and packages?**

* **Module**: a .py file with Python definitions.
* **Package**: a directory with an \_\_init\_\_.py file containing modules.

**16. How would you handle exceptions?**

try:

x = 1 / 0

except ZeroDivisionError:

print("Cannot divide by zero.")

finally:

print("Done")

**17. How to reverse a string in Python?**

s = "hello"

print(s[::-1])

**18. How to merge two dictionaries in Python 3.9+?**

d1 = {"a": 1}; d2 = {"b": 2}

merged = d1 | d2

**19. How do you debug a Python program?**

* Use pdb module.

python -m pdb script.py

**20. Explain how you’d improve Python code performance.**

* Use built-in functions.
* Use list comprehensions.
* Avoid global variables.
* Use generators.
* Profile with cProfile.

--------------------------------------------------------------------------------------------------------------

1. **What is Python?**
   * A high-level, interpreted, dynamically typed programming language known for readability and versatility.
2. **Is Python case-sensitive?**
   * Yes, Variable and variable are different.
3. **What are the key features of Python?**
   * Interpreted, object-oriented, dynamically typed, large standard library, cross-platform.
4. **What is PEP 8?**
   * A style guide for writing readable Python code.
5. **How do you write comments in Python?**
   * Single-line: # comment, Multi-line: triple quotes ('''comment''').
6. **What is the difference between = and ==?**
   * = is assignment, == is equality comparison.
7. **What is a variable?**
   * A named reference to a value stored in memory.
8. **What are Python keywords?**
   * Reserved words like if, for, True, etc.
9. **How to take input from the user?**
   * input("Enter name: ")
10. **What is indentation in Python?**
    * Defines code blocks, typically 4 spaces.
11. **How to print in Python without newline?**
    * print("Hello", end="")
12. **What is a literal?**
    * Raw data assigned to a variable, e.g., numbers, strings.
13. **What is the difference between None, False, and 0?**
    * All are falsy, but None is a special null object.
14. **What is the difference between Python 2 and 3?**
    * Syntax, print function, division, Unicode support, etc.
15. **What is typecasting?**
    * Converting one data type into another, e.g., int("5").
16. **List common data types.**
    * int, float, str, bool, list, tuple, set, dict.
17. **What is the difference between list and tuple?**
    * List is mutable, tuple is immutable.
18. **What is a dictionary in Python?**
    * An unordered collection of key-value pairs.
19. **How to convert a list to a set?**
    * set(my\_list)
20. **How to check the type of a variable?**
    * type(variable)
21. **What is the difference between is and ==?**
    * is: identity; ==: value equality.
22. **How to convert a string to a list?**
    * list("hello") → ['h','e','l','l','o']
23. **How to get the length of a list or string?**
    * len(obj)
24. **How do you create a range of numbers?**
    * range(start, stop[, step])
25. **What is a frozen set?**
    * An immutable set: frozenset([1, 2])
26. **What is a boolean in Python?**
    * True or False, subclass of int.
27. **How to slice a list or string?**
    * my\_list[start:end:step]
28. **What are identity and membership operators?**
    * is, is not; in, not in
29. **What is unpacking in Python?**
    * Assigning elements from a sequence to variables:  
      a, b = [1, 2]
30. **What are mutable and immutable types?**
    * Mutable: list, dict; Immutable: int, str, tuple.
31. **What are conditional statements?**
    * if, elif, else
32. **What are loops in Python?**
    * for and while loops
33. **What are loop control statements?**
    * break, continue, pass
34. **What is the difference between break and continue?**
    * break exits loop, continue skips to next iteration.
35. **What is the pass statement?**
    * A placeholder for future code.
36. **How do you iterate over a list?**
    * for i in my\_list:
37. **How do you iterate with index?**
    * for i, val in enumerate(my\_list):
38. **How to loop over a dictionary?**

for k, v in my\_dict.items():

print(k, v)

1. **What is the ternary operator in Python?**
   * x if condition else y
2. **How to use else with a loop?**
   * Runs if the loop completes without break.
3. **How to define a function in Python?**
   * def function\_name():
4. **What is the use of return?**
   * To return a value from a function.
5. \*\*What are \*args and **kwargs?**
   * Accept variable number of arguments.
6. **What is a lambda function?**
   * An anonymous one-liner function.
7. **What is recursion?**
   * A function calling itself.
8. **What is a docstring?**
   * A string to describe a function/module/class.
9. **What is the difference between local and global variables?**
   * Local: inside function; Global: outside function.
10. **How to make a variable global?**
    * Use global keyword.
11. **Can functions return multiple values?**
    * Yes, as a tuple.
12. **What is the scope of a variable?**
    * LEGB: Local, Enclosing, Global, Built-in
13. **What is function overloading?**
    * Not supported natively; can be mimicked using default args.
14. **What is map() in Python?**
    * Applies a function to each item in iterable.
15. **What is filter() in Python?**
    * Filters items based on a condition.
16. **What is reduce() in Python?**
    * Aggregates items into a single value.
17. **What are higher-order functions?**
    * Functions that take other functions as arguments.
18. **What is OOP?**
    * Object-Oriented Programming
19. **What are classes and objects?**
    * Class: blueprint; Object: instance of class
20. **What is \_\_init\_\_()?**
    * Constructor method.
21. **What is self?**
    * Refers to the instance calling the method.
22. **What is inheritance?**
    * Acquiring properties of a parent class.
23. **What is multiple inheritance?**
    * A class inheriting from more than one class.
24. **What are super() and its use?**
    * Calls parent class methods.
25. **What is encapsulation?**
    * Hiding data using private/protected variables.
26. **What is polymorphism?**
    * Same method name behaving differently based on context.
27. **What are @staticmethod and @classmethod?**
    * Static: no self/cls; Class: receives cls.
28. **How to make private attributes?**
    * Prefix with \_\_: self.\_\_data
29. **What is method overloading?**
    * Not directly supported; use default args.
30. **What is method overriding?**
    * Child class defines same method as parent.
31. **What is data hiding?**
    * Restricting access to attributes using name mangling.
32. **What is duck typing in Python?**
    * “If it walks like a duck...” → type is determined by behavior.
33. **What is exception handling?**
    * Mechanism to handle runtime errors using try-except.
34. **What are common exceptions?**
    * ZeroDivisionError, TypeError, KeyError, ValueError
35. **What is finally block?**
    * Executes regardless of exception.
36. **How to raise an exception?**
    * raise ValueError("Invalid value")
37. **How to read from a file?**

with open("file.txt", "r") as f:

data = f.read()

1. **How to write to a file?**

with open("file.txt", "w") as f:

f.write("Hello")

1. **How to append to a file?**
   * Use mode "a".
2. **What is the difference between read(), readline(), readlines()?**
   * Entire content, one line, all lines as list.
3. **What is the purpose of with statement?**
   * Automatically handles file closing.
4. **How to check if a file exists?**
   * os.path.exists(filename)
5. **What is a generator?**
   * Function that yields values one at a time using yield.
6. **What is a coroutine?**
   * A generator that can consume values with send().
7. **What is a decorator?**
   * A function that wraps another function to modify its behavior.
8. **What are Python modules and packages?**
   * Module: .py file; Package: folder with \_\_init\_\_.py
9. **What is the Python standard library?**
   * Built-in modules like math, json, re, etc.
10. **What is pip?**
    * Python package installer.
11. **What is virtualenv?**
    * Tool to create isolated Python environments.
12. **What is the GIL (Global Interpreter Lock)?**
    * Prevents multiple native threads from executing Python bytecode simultaneously.
13. **What is multithreading?**
    * Concurrent execution of threads.
14. **What is multiprocessing?**
    * Runs multiple processes to bypass GIL.
15. **What is monkey patching?**
    * Changing code behavior at runtime.
16. **What is the difference between deepcopy() and copy()?**
    * deepcopy copies nested objects, copy does not.
17. **What is \_\_name\_\_ == '\_\_main\_\_' for?**
    * Ensures code is only run when script is executed directly.
18. **What is memory management in Python?**
    * Done via reference counting and garbage collection.
19. **What is a context manager?**
    * Uses \_\_enter\_\_ and \_\_exit\_\_ methods. e.g., with statement.
20. **What is the difference between del and remove()?**
    * del deletes by index/reference, remove() by value.
21. **What is a metaclass?**
    * A class of a class; defines class behavior.
22. **What are dunder (magic) methods?**
    * Special methods like \_\_init\_\_, \_\_str\_\_, \_\_len\_\_, etc.
23. **How to serialize objects in Python?**
    * Using pickle or json.
24. **What are common Python libraries used in real-world projects?**  
    - NumPy, Pandas, Matplotlib, Flask, Django, Requests, SQLAlchemy, Scikit-learn
25. **What is Python and why is it popular?**  
    Python is a high-level, interpreted language known for its simplicity, readability, and wide ecosystem of libraries.
26. **What is the difference between an interpreter and a compiler?**  
    An interpreter executes code line-by-line, while a compiler translates the whole code into machine language before execution.
27. **What is Python syntax?**  
    It's the set of rules that define how a Python program is written and interpreted (e.g., indentation, colons).
28. **What is dynamic typing in Python?**  
    You don’t declare variable types explicitly; Python assigns them at runtime.
29. **How does Python handle memory management?**  
    Through reference counting and garbage collection.
30. **What are Python's core data types?**  
    int, float, bool, str, list, tuple, set, dict.
31. **What are namespaces?**  
    They are containers that map names to objects, helping avoid naming conflicts.
32. **How are comments used in Python?**  
    Single-line: #, Multi-line: triple quotes ''' '''.
33. **What are operators in Python?**  
    Arithmetic (+, -, \*, /), Logical (and, or, not), Comparison (==, !=, >).
34. **How is user input handled?**  
    Using input() function, which always returns a string.
35. **What is the difference between a list and a tuple?**  
    Lists are mutable; tuples are immutable.
36. **What are sets in Python?**  
    Unordered collections of unique elements.
37. **How is type casting done?**  
    Using functions like int(), float(), str(), etc.
38. **What is a string in Python?**  
    Immutable sequence of Unicode characters.
39. **How do you generate a sequence of numbers?**  
    Using range(start, stop, step).
40. **What is Gitignore used for?**  
    To ignore files and folders from being tracked by Git.
41. **What are binary types?**  
    bytes, bytearray, and memoryview.
42. **How do booleans work in Python?**  
    True and False are special cases of integers 1 and 0.
43. **How to declare a dictionary?**  
    d = {'key': 'value'}
44. **How to check if a key exists in a dictionary?**  
    Use in: 'key' in dict.
45. **What are numbers in Python?**  
    Numeric data types: int, float, complex.
46. **What is the output of bool([])?**  
    False, since empty containers are falsy.
47. **How do you convert a list to a set?**  
    set(my\_list)
48. **What is the difference between None and 0?**  
    None means "no value", while 0 is a numeric value.
49. **What is the type of range(5)?**  
    range object (an iterable).
50. **How do you define a function?**  
    def my\_function():
51. **What is a lambda function?**  
    Anonymous one-line function: lambda x: x\*2
52. **How do you pass arguments to functions?**  
    Positional, keyword, default, \*args, \*\*kwargs.
53. **What is the difference between \*args and \*\*kwargs?**  
    \*args for non-keyworded args; \*\*kwargs for keyworded args.
54. **How do you return multiple values from a function?**  
    As a tuple: return a, b
55. **What is a docstring?**  
    First statement in a function/module/class to describe it.
56. **What is a Python array?**  
    From the array module; stores elements of the same type.
57. **Difference between list and array?**  
    Lists can store mixed types; arrays are more memory-efficient but type-restricted.
58. **What is the use of enumerate()?**  
    Adds counter to an iterable.
59. **How to create default arguments in functions?**  
    def greet(name="Guest"): assigns default if not passed.
60. **What is a class in Python?**  
    A blueprint for creating objects.
61. **What is self?**  
    Refers to the instance calling the method.
62. **What is inheritance?**  
    A class can inherit properties from another class.
63. **How to create a class?**

class MyClass:

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

self.name = "Remya"

1. **What are instance variables?**  
   Belong to an object, not a class.
2. **What is polymorphism?**  
   Methods with the same name behaving differently based on context.
3. **What is encapsulation?**  
   Hiding internal states using private/protected members.
4. **What are iterators?**  
   Objects with \_\_iter\_\_() and \_\_next\_\_() methods.
5. **How do you create a custom iterator?**  
   Define a class with \_\_iter\_\_() and \_\_next\_\_().
6. **What is pylint used for?**  
   Static code analysis tool for Python.
7. **What is a module in Python?**  
   A .py file containing Python code.
8. **How to import a module?**  
   import math or from math import sqrt
9. **What is the use of pip?**  
   Python package manager to install libraries.
10. **How to install a package using pip?**  
    pip install package-name
11. **What is the math module?**  
    Built-in module with mathematical functions.
12. **How to calculate square root using math module?**  
    math.sqrt(9)
13. **What is the use of json module?**  
    To encode and decode JSON.
14. **How to convert a dictionary to JSON?**  
    json.dumps(dictionary)
15. **What is regex in Python?**  
    Regular expression for pattern matching.
16. **How to use regex in Python?**  
    Using re module: re.search(pattern, string)
17. **What are collections modules?**  
    Advanced data structures like deque, Counter, defaultdict.
18. **What is Counter used for?**  
    Count occurrences of elements.
19. **What is defaultdict?**  
    A dictionary that returns a default value if key is missing.
20. **What is namedtuple?**  
    Tuple with named fields.
21. **Difference between import module and from module import \*?**  
    First is explicit; second imports everything (not recommended).
22. **How do you open a file in Python?**  
    open("filename.txt", "r")
23. **How do you read a file?**  
    file.read() or file.readlines()
24. **How do you write to a file?**  
    file.write("Hello")
25. **What does "w" mode do in open()?**  
    Write mode; overwrites existing content.
26. **What is with open() used for?**  
    Ensures the file is automatically closed.
27. **How to delete a file?**  
    os.remove("file.txt")
28. **What is exception handling?**  
    Catching runtime errors using try, except.
29. **How do you raise an exception?**  
    raise ValueError("Invalid input")
30. **What is the use of finally block?**  
    Executes code regardless of exceptions.
31. **What happens when no exception is raised in try block?**  
    except is skipped; finally is executed.
32. **How to catch multiple exceptions?**

except (TypeError, ValueError) as e:

1. **What is a custom exception?**  
   User-defined exception using class inheriting Exception.
2. **What is assert in Python?**  
   Used to debug by testing conditions: assert x > 0
3. **Difference between syntax error and exception?**  
   Syntax error: compile-time; Exception: runtime.
4. **What is the base class for exceptions?**  
   BaseException
5. **What is NumPy used for?**  
   Numerical computations and array processing.
6. **How to create a NumPy array?**  
   np.array([1, 2, 3])
7. **What is the shape of a NumPy array?**  
   Tuple showing dimensions: array.shape
8. **What is Pandas used for?**  
   Data analysis and manipulation.
9. **What is a DataFrame in Pandas?**  
   2D labeled data structure.
10. **How to read a CSV in Pandas?**  
    pd.read\_csv("file.csv")
11. **How to get the first 5 rows of DataFrame?**  
    df.head()
12. **How to filter rows in Pandas?**  
    df[df['column'] > 10]
13. **What is Matplotlib used for?**  
    Data visualization and plotting.
14. **How to plot a line graph?**

plt.plot(x, y)

plt.show()

1. **What is unit testing?**  
   Testing individual units or functions of code.
2. **What is Pytest?**  
   Python testing framework to write simple test cases.
3. **How to write a test case in Pytest?**  
   Function name must start with test\_.
4. **What is a fixture in Pytest?**  
   Reusable setup code using @pytest.fixture.
5. **How do you run tests using Pytest?**  
   pytest test\_file.py
6. **How to assert in Pytest?**  
   Use Python’s assert statement.
7. **What is test discovery in Pytest?**  
   Pytest auto-discovers files with test\_\*.py.
8. **What is mocking in testing?**  
   Replacing real objects with mock ones.
9. **What is TDD (Test-Driven Development)?**  
   Writing tests before code.
10. **What is the unittest module?**  
    Built-in module for unit testing in Python.
11. **Difference between unittest and Pytest?**  
    Pytest is simpler, more flexible than unittest.
12. **How to test exceptions in Pytest?**

with pytest.raises(ValueError):

function()

1. **How to skip a test in Pytest?**  
   Use @pytest.mark.skip.
2. **How to run only failed tests in Pytest?**  
   pytest --lf
3. **How to generate test report in Pytest?**  
   pytest --html=report.html

SCENARIO

1. **Scenario:** You have a list of numbers. Remove duplicates while preserving order.  
   **Solution:**

seen = set()

result = []

for num in [1, 2, 2, 3, 1]:

if num not in seen:

seen.add(num)

result.append(num)

1. **Scenario:** Convert two lists into a dictionary.  
   **Solution:**

keys = ['a', 'b']

values = [1, 2]

my\_dict = dict(zip(keys, values))

1. **Scenario:** Flatten a nested list: [[1,2],[3,4]] → [1,2,3,4]  
   **Solution:**

flat = [num for sublist in [[1, 2], [3, 4]] for num in sublist]

1. **Scenario:** Reverse a string without using slicing.  
   **Solution:**

def reverse\_string(s):

return ''.join(reversed(s))

1. **Scenario:** Count occurrences of each character in a string.  
   **Solution:**

from collections import Counter

Counter("hello")

1. **Scenario:** Print only even numbers from a list using list comprehension.  
   **Solution:**

evens = [x for x in range(10) if x % 2 == 0]

1. **Scenario:** Find the first repeated element in a list.  
   **Solution:**

seen = set()

for x in [1, 2, 3, 2, 4]:

if x in seen:

print(x)

break

seen.add(x)

1. **Scenario:** Sum all values in a nested dictionary.  
   **Solution:**

data = {'a': 1, 'b': {'c': 2, 'd': 3}}

total = data['a'] + sum(data['b'].values())

1. **Scenario:** Print a pattern using nested loops.  
   **Solution:**

for i in range(1, 5):

print('\*' \* i)

1. **Scenario:** Loop through a string and skip vowels.  
   **Solution:**

for ch in "python":

if ch in 'aeiou':

continue

print(ch)

1. **Scenario:** Create a function that takes a list and returns only odd numbers.  
   **Solution:**

def get\_odds(lst):

return [x for x in lst if x % 2 != 0]

1. **Scenario:** Write a lambda that squares a number.  
   **Solution:**

square = lambda x: x\*\*2

1. **Scenario:** Use map() to double each element in a list.  
   **Solution:**

list(map(lambda x: x \* 2, [1, 2, 3]))

1. **Scenario:** Write a function that returns both sum and product of two numbers.  
   **Solution:**

def compute(a, b):

return a + b, a \* b

1. **Scenario:** Use filter() to remove empty strings from a list.  
   **Solution:**

list(filter(None, ['a', '', 'b']))

1. **Scenario:** Count lines in a text file.  
   **Solution:**

with open('file.txt') as f:

line\_count = len(f.readlines())

1. **Scenario:** Read only the first 10 characters from a file.  
   **Solution:**with open('file.txt') as f:

print(f.read(10))

1. **Scenario:** Write a list of strings to a file.  
   **Solution:**

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

f.writelines(['line1\n', 'line2\n'])

1. **Scenario:** Append new content to an existing file.  
   **Solution:**

with open('file.txt', 'a') as f:

f.write('new line\n')

1. **Scenario:** Check if a file exists before reading.  
   **Solution:**

import os

if os.path.exists("file.txt"):

with open("file.txt") as f:

print(f.read())

1. **Scenario:** Handle division by zero error.  
   **Solution:**

try:

x = 5 / 0

except ZeroDivisionError:

print("Cannot divide by zero.")

1. **Scenario:** Handle missing file error.  
   **Solution:**

try:

open('nofile.txt')

except FileNotFoundError:

print("File not found.")

1. **Scenario:** Use finally to close a file.  
   **Solution:**

try:

f = open('file.txt')

finally:

f.close()

1. **Scenario:** Raise a custom exception for negative numbers.  
   **Solution:**

if x < 0:

raise ValueError("Negative not allowed")

1. **Scenario:** Catch multiple exception types.  
   **Solution:**

try:

# some code

except (ValueError, ZeroDivisionError):

pass

1. **Scenario:** Create a class Person with name and age.  
   **Solution:**

class Person:

def \_\_init\_\_(self, name, age):

self.name = name

self.age = age

1. **Scenario:** Add a method to return full details.  
   **Solution:**

def details(self):

return f"{self.name} is {self.age} years old"

1. **Scenario:** Inherit from Person and create a Student class.  
   **Solution:**

class Student(Person):

def \_\_init\_\_(self, name, age, course):

super().\_\_init\_\_(name, age)

self.course = course

1. **Scenario:** Override a method in child class.  
   **Solution:**

def details(self):

return f"{self.name} studies {self.course}"

1. **Scenario:** Count number of objects created from a class.  
   **Solution:**

class MyClass:

count = 0

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

MyClass.count += 1

1. **Scenario:** Create a DataFrame from a list of dicts.  
   **Solution:**

import pandas as pd

df = pd.DataFrame([{'a': 1}, {'a': 2}])

1. **Scenario:** Filter rows where column age > 30.  
   **Solution:**

df[df['age'] > 30]

1. **Scenario:** Read a CSV file and display top 5 rows.  
   **Solution:**

df = pd.read\_csv('data.csv')

print(df.head())

1. **Scenario:** Replace NaN values with 0.  
   **Solution:**

df.fillna(0, inplace=True)

1. **Scenario:** Create a NumPy array and reshape it.  
   **Solution:**

import numpy as np

a = np.arange(6).reshape(2, 3)

1. **Scenario:** Plot a bar chart using Matplotlib.  
   **Solution:**

import matplotlib.pyplot as plt

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

plt.show()

1. **Scenario:** Drop duplicates in Pandas DataFrame.  
   **Solution:**

df.drop\_duplicates(inplace=True)

1. **Scenario:** Sort DataFrame by a column.  
   **Solution:**

df.sort\_values(by='age')

1. **Scenario:** Apply a function to a DataFrame column.  
   **Solution:**

df['salary'] = df['salary'].apply(lambda x: x \* 1.1)

1. **Scenario:** Get basic stats of DataFrame.  
   **Solution:**

df.describe()

1. **Scenario:** Write a basic Pytest function.  
   **Solution:**

def add(x, y): return x + y

def test\_add(): assert add(2, 3) == 5

1. **Scenario:** Mark a test to skip.  
   **Solution:**

import pytest

@pytest.mark.skip

def test\_skip(): assert True

1. **Scenario:** Test for exceptions in Pytest.  
   **Solution:**

with pytest.raises(ZeroDivisionError):

1 / 0

1. **Scenario:** Parametrize test inputs.  
   **Solution:**

@pytest.mark.parametrize("a,b", [(1,2), (3,4)])

def test\_add(a,b): assert a+b > 0

1. **Scenario:** Test a function that reads a file.  
   **Solution:**  
   Use mocking with unittest.mock.
2. **Scenario:** Get current date and time.  
   **Solution:**

from datetime import datetime

print(datetime.now())

1. **Scenario:** Convert list of strings to integers.  
   **Solution:**

list(map(int, ['1', '2']))

1. **Scenario:** Zip two lists into pairs.  
   **Solution:**

list(zip([1, 2], ['a', 'b']))

1. **Scenario:** Create a timer using time.sleep().  
   **Solution:**

import time

time.sleep(2)

1. **Scenario:** Get count of each word in a sentence.  
   **Solution:**

Counter("this is a test this".split())