**1. Positional & Default Arguments**

**Case Study:**

Create a function greet that takes a name (positional argument) and a greeting (default argument, default value: "Hello") and returns a greeting message.

**Solution:**

def greet(name, greeting="Hello"):

return f"{greeting}, {name}!"

# Test cases

print(greet("Alice")) # Output: Hello, Alice!

print(greet("Bob", "Good morning")) # Output: Good morning, Bob!

**2. Variable Positional Arguments (\*args)**

**Case Study:**

Create a function sum\_numbers that accepts a variable number of arguments and returns their sum.

**Solution:**

def sum\_numbers(\*args):

return sum(args)

# Test cases

print(sum\_numbers(1, 2, 3)) # Output: 6

print(sum\_numbers(4, 5, 6, 7, 8)) # Output: 30

**3. Variable Keyword Arguments (\*\*kwargs)**

**Case Study:**

Create a function person\_details that accepts any number of keyword arguments and returns a formatted string.

**Solution:**

def person\_details(\*\*kwargs):

return ", ".join(f"{key}: {value}" for key, value in kwargs.items())

# Test cases

print(person\_details(name="Alice", age=25, city="New York"))

# Output: "name: Alice, age: 25, city: New York"

**4. Keyword-Only Arguments**

**Case Study:**

Create a function student\_info that accepts name as a positional argument and age as a keyword-only argument.

**Solution:**

def student\_info(name, \*, age):

return f"Student: {name}, Age: {age}"

# Test cases

print(student\_info("John", age=22)) # Output: "Student: John, Age: 22"

**5. Passing Values in Different Ways**

**Case Study:**

Demonstrate different ways to pass values to a function (positional, keyword, tuple unpacking, dictionary unpacking).

**Solution:**

def display(a, b, c):

return f"a={a}, b={b}, c={c}"

# Positional

print(display(1, 2, 3))

# Keyword

print(display(a=10, b=20, c=30))

# Tuple Unpacking

tup = (100, 200, 300)

print(display(\*tup))

# Dictionary Unpacking

d = {"a": 7, "b": 8, "c": 9}

print(display(\*\*d))

**6. Function Returning Another Function**

**Case Study:**

Create a function multiplier that returns a function to multiply numbers.

**Solution:**

def multiplier(factor):

def multiply(number):

return number \* factor

return multiply

double = multiplier(2)

print(double(5)) # Output: 10

**7. Recursion - Factorial**

**Case Study:**

Write a recursive function factorial to calculate the factorial of a number.

**Solution:**

def factorial(n):

if n == 0 or n == 1:

return 1

return n \* factorial(n - 1)

print(factorial(5)) # Output: 120

**8. Recursion - Fibonacci**

**Case Study:**

Write a recursive function to find the nth Fibonacci number.

**Solution:**

def fibonacci(n):

if n <= 1:

return n

return fibonacci(n-1) + fibonacci(n-2)

print(fibonacci(6)) # Output: 8

**9. Iterator Functions**

**Case Study:**

Create an iterator for even numbers up to n.

**Solution:**

class EvenNumbers:

def \_\_init\_\_(self, max\_num):

self.max\_num = max\_num

self.num = 0

def \_\_iter\_\_(self):

return self

def \_\_next\_\_(self):

if self.num > self.max\_num:

raise StopIteration

self.num += 2

return self.num - 2

evens = EvenNumbers(10)

for num in evens:

print(num)

**10. Generator Function**

**Case Study:**

Create a generator for prime numbers up to n.

**Solution:**

def prime\_generator(n):

for num in range(2, n + 1):

if all(num % i != 0 for i in range(2, int(num\*\*0.5) + 1)):

yield num

print(list(prime\_generator(10))) # Output: [2, 3, 5, 7]

**11. Closure Example**

**Case Study:**

Create a closure to generate functions for exponentiation.

**Solution:**

def power(exponent):

def inner(base):

return base \*\* exponent

return inner

square = power(2)

print(square(5)) # Output: 25

**12. Decorator Example**

**Case Study:**

Create a decorator that logs function calls.

**Solution:**

def log\_decorator(func):

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

print(f"Calling {func.\_\_name\_\_} with {args} {kwargs}")

return func(\*args, \*\*kwargs)

return wrapper

@log\_decorator

def add(a, b):

return a + b

print(add(3, 4)) # Output: Log message and result

**13. Lambda Function - Simple**

**Case Study:**

Use a lambda function to calculate the square of a number.

**Solution:**

square = lambda x: x \* x

print(square(6)) # Output: 36

**14. Lambda Function with map**

**Case Study:**

Use map to square a list of numbers.

**Solution:**

nums = [1, 2, 3, 4]

squared = list(map(lambda x: x \*\* 2, nums))

print(squared) # Output: [1, 4, 9, 16]

**15. Lambda Function with filter**

**Case Study:**

Use filter to get even numbers.

**Solution:**

nums = [1, 2, 3, 4, 5, 6]

evens = list(filter(lambda x: x % 2 == 0, nums))

print(evens) # Output: [2, 4, 6]