**Lab Exercises on Python Functions**

**1. Defining a Function with Positional Arguments**

**Exercise:**  
Write a function add\_numbers(a, b) that takes two numbers as positional arguments and returns their sum.

def add\_numbers(a, b):

return a + b

# Example usage

print(add\_numbers(5, 10)) # Output: 15

**2. Function with Default Arguments**

**Exercise:**  
Create a function greet(name, message="Hello") that prints a greeting message.

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

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

# Example usage

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

print(greet("Bob", "Hi")) # Output: Hi, Bob!

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

**Exercise:**  
Write a function sum\_all(\*numbers) that takes any number of arguments and returns their sum.

def sum\_all(\*numbers):

return sum(numbers)

# Example usage

print(sum\_all(1, 2, 3, 4)) # Output: 10

print(sum\_all(5, 10)) # Output: 15

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

**Exercise:**  
Create a function display\_info(\*\*info) that prints key-value pairs.

def display\_info(\*\*info):

for key, value in info.items():

print(f"{key}: {value}")

# Example usage

display\_info(name="Alice", age=25, city="New York")

**5. Keyword-Only Arguments**

**Exercise:**  
Write a function student\_info(name, \*, age, grade) that takes keyword-only arguments.

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

return f"{name} is {age} years old and in grade {grade}."

# Example usage

print(student\_info("John", age=15, grade=10))

**6. Different Ways to Pass Values**

**Exercise:**  
Write a function area(length, width) and call it using positional and keyword arguments.

def area(length, width):

return length \* width

# Example usage

print(area(5, 10)) # Positional

print(area(length=5, width=10)) # Keyword

print(area(10, width=5)) # Mixed

**7. Function Returning Another Function**

**Exercise:**  
Create a function multiplier(n) that returns another function that multiplies a number by n.

def multiplier(n):

def multiply(x):

return x \* n

return multiply

double = multiplier(2)

print(double(5)) # Output: 10

**8. Recursion: Factorial**

**Exercise:**  
Write a recursive function factorial(n).

def factorial(n):

if n == 0:

return 1

return n \* factorial(n - 1)

# Example usage

print(factorial(5)) # Output: 120

**9. Recursion: Fibonacci**

**Exercise:**  
Write a recursive function to compute Fibonacci numbers.

def fibonacci(n):

if n <= 1:

return n

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

# Example usage

print(fibonacci(6)) # Output: 8

**10. Creating an Iterator Class**

**Exercise:**  
Implement an iterator class for even numbers.

class EvenNumbers:

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

self.limit = limit

self.num = 0

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

return self

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

if self.num > self.limit:

raise StopIteration

self.num += 2

return self.num - 2

# Example usage

evens = EvenNumbers(10)

for num in evens:

print(num)

**11. Generator Function for Fibonacci**

**Exercise:**  
Write a generator function to generate Fibonacci numbers.

def fibonacci\_gen(n):

a, b = 0, 1

for \_ in range(n):

yield a

a, b = b, a + b

# Example usage

for num in fibonacci\_gen(6):

print(num)

**12. Generator Expression**

**Exercise:**  
Create a generator expression for squares of numbers.

squares = (x \* x for x in range(5))

for sq in squares:

print(sq)

**13. Closure Function**

**Exercise:**  
Write a closure function that keeps track of how many times it has been called.

def counter():

count = 0

def inner():

nonlocal count

count += 1

return count

return inner

counter\_fn = counter()

print(counter\_fn()) # Output: 1

print(counter\_fn()) # Output: 2

**14. Simple Decorator**

**Exercise:**  
Write a decorator that prints "Function is being called" before execution.

def announce(func):

def wrapper():

print("Function is being called")

return func()

return wrapper

@announce

def greet():

print("Hello!")

greet()

**15. Decorator with Arguments**

**Exercise:**  
Modify the above decorator to work with arguments.

def announce(func):

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

print("Function is being called")

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

return wrapper

@announce

def greet(name):

print(f"Hello, {name}!")

greet("Alice")

**16. Chaining Multiple Decorators**

**Exercise:**  
Write two decorators and apply them to the same function.

def uppercase(func):

def wrapper():

return func().upper()

return wrapper

def greet\_decorator(func):

def wrapper():

return f"Hello, {func()}!"

return wrapper

@uppercase

@greet\_decorator

def name():

return "Alice"

print(name()) # Output: HELLO, ALICE!

**17. Lambda Function for Addition**

**Exercise:**  
Write a lambda function to add two numbers.

add = lambda x, y: x + y

print(add(3, 7)) # Output: 10

**18. Lambda with map()**

**Exercise:**  
Use map() and a lambda function to double a list of numbers.

numbers = [1, 2, 3, 4]

doubled = list(map(lambda x: x \* 2, numbers))

print(doubled) # Output: [2, 4, 6, 8]

**19. Lambda with filter()**

**Exercise:**  
Use filter() to filter even numbers.

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

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

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

**20. Lambda with reduce()**

**Exercise:**  
Use reduce() to compute the product of numbers.

from functools import reduce

numbers = [1, 2, 3, 4]

product = reduce(lambda x, y: x \* y, numbers)

print(product) # Output: 24