```
print(greet("Bob"))    # Output: Hello, Bob!
```

Classes:

1. **Class for Modeling a Car:**

   Classes are used for modeling complex data structures with attributes and behaviors.

```python
class Car:
    def __init__(self, make, model):
        self.make = make
        self.model = model
        self.speed = 0

    def accelerate(self):
        self.speed += 10

    def brake(self):
        self.speed -= 5

    def get_speed(self):
        return self.speed

my_car = Car("Toyota", "Camry")
my_car.accelerate()
print(f"My {my_car.make} {my_car.model} is going at {my_car.get_spe
```

1. **Inheritance and Polymorphism:**

   Classes can be used to create hierarchies and support polymorphism.

```python
class Animal:
    def speak(self):
        pass

class Dog(Animal):
    def speak(self):
```

- **When to Use**:
  - For small, isolated tasks with a well-defined purpose.
  - When you want to reuse a piece of code in multiple places.
  - To create modular and maintainable code.

2. **Classes**:
   - **Purpose**: Classes are used for modeling complex systems, creating objects with attributes and behaviors, and supporting Object-Oriented Programming (OOP) principles.
   - **When to Use**:
     - When you need to represent real-world entities with attributes and behaviors.
     - For maintaining state across multiple method calls.
     - To support inheritance, polymorphism, and abstraction.

   Now, let's explore 15 examples to illustrate when and how to use functions and classes.

   Examples of Using Functions:

1. **Simple Calculator Function**:
   - **Purpose**: A basic function for addition.
   - **Explanation**: Functions are suitable for encapsulating specific tasks.

python Copy code
```python
def add(a, b):
    return a + b

result = add(3, 4)
# Result: 7
```

1. **String Manipulation Function**:
   - **Purpose**: Function to capitalize and reverse a string.
   - **Explanation**: Functions are great for code reuse.

python Copy code
```python
def manipulate_string(text):
    return text.upper(), text[::-1]

upper_text, reversed_text = manipulate_string("hello")
# upper_text: 'HELLO', reversed_text: 'olleh'
```

1. **List Filtering Function**:

- **Purpose**: Function to filter even numbers from a list.
- **Explanation**: Functions help modularize code.

python Copy code

```python
def filter_even_numbers(numbers):
    return [num for num in numbers if num % 2 == 0]

evens = filter_even_numbers([1, 2, 3, 4, 5, 6])
# evens: [2, 4, 6]
```

1. **Factorial Calculation Function**:
   - **Purpose**: Function to calculate the factorial of a number.
   - **Explanation**: Functions are suitable for mathematical computations.

python Copy code

```python
def factorial(n):
    if n == 0:
        return 1
    else:
        return n * factorial(n - 1)

fact_5 = factorial(5)
# fact_5: 120
```

1. **File Reading Function**:
   - **Purpose**: Function to read and process data from a file.
   - **Explanation**: Functions help organize I/O operations.

python Copy code

```python
def process_file(filename):
    with open(filename, 'r') as file:
        data = file.read()
        # Process data here
    return data

file_data = process_file('data.txt')
```

```
# file_data: Content of the 'data.txt' file
```

Examples of Using Classes:

1. **Class for a Bank Account**:
   - **Purpose**: Class to model a bank account with attributes and methods.
   - **Explanation**: Classes are ideal for modeling real-world entities.

python Copy code

```python
class BankAccount:
    def __init__(self, account_number, balance):
        self.account_number = account_number
        self.balance = balance

    def deposit(self, amount):
        self.balance += amount

    def withdraw(self, amount):
        if self.balance >= amount:
            self.balance -= amount
        else:
            print("Insufficient balance.")

    def get_balance(self):
        return self.balance

acc1 = BankAccount('12345', 1000)
acc1.deposit(500)
acc1.withdraw(200)
# Current balance: 1300
```

1. **Inheritance with Animals**:
   - **Purpose**: Base class `Animal` and derived classes for specific animals.
   - **Explanation**: Classes support inheritance and polymorphism.

python Copy code

```python
class Animal:
    def speak(self):
        pass

class Dog(Animal):
    def speak(self):
        return "Woof!"

class Cat(Animal):
    def speak(self):
        return "Meow!"

def animal_sound(animal):
    return animal.speak()

dog = Dog()
cat = Cat()
sound1 = animal_sound(dog)  # "Woof!"
sound2 = animal_sound(cat)  # "Meow!"
```

1. **Class for a ToDo List**:
   - **Purpose**: Class for managing a ToDo list.
   - **Explanation**: Classes are helpful for maintaining state.

python Copy code

```python
class ToDoList:
    def __init__(self):
        self.tasks = []

    def add_task(self, task):
        self.tasks.append(task)

    def get_tasks(self):
        return self.tasks

my_list = ToDoList()
my_list.add_task("Buy groceries")
my_list.add_task("Finish homework")
tasks = my_list.get_tasks()
# tasks: ['Buy groceries', 'Finish homework']
```

1. **Geometry Shapes with Classes**:
   - **Purpose**: Classes for modeling geometric shapes.
   - **Explanation**: Classes help create complex systems.

python Copy code

```python
class Shape:
    def area(self):
        pass

class Circle(Shape):
    def __init__(self, radius):
        self.radius = radius

    def area(self):
        return 3.14 * self.radius * self.radius

class Rectangle(Shape):
    def __init__(self, length, width):
        self.length = length
        self.width = width

    def area(self):
        return self.length * self.width

circle = Circle(5)
rect = Rectangle(4, 6)
area_circle = circle.area()     # 78.5
area_rectangle = rect.area()   # 24
```

1. **Class for a Social Media Post**:
   - **Purpose**: Class for representing a social media post.
   - **Explanation**: Classes encapsulate attributes and behaviors.

python Copy code

```python
class SocialMediaPost:
    def __init__(self, user, content):
        self.user = user
        self.content = content
        self.likes = 0

    def like(self):
        self.likes += 1

    def get_likes(self):
        return self.likes

post = SocialMediaPost("Alice", "Having a great day!")
post.like()
likes = post.get_likes()
# likes: 1
```

1. **Class for a Music Player**:
   - **Purpose**: Class for a music player with play and pause functionality.
   - **Explanation**: Classes help abstract complex functionality.

python Copy code

```python
class MusicPlayer:
    def __init__(self):
        self.playing = False

    def play(self):
        self.playing = True

    def pause(self):
        self.playing = False

    def is_playing(self):
        return self.playing
```

```
player = MusicPlayer()
player.play()
playing = player.is_playing()
# playing: True
```

1. **Class for a Product Catalog**:
   - **Purpose**: Class for managing a product catalog.
   - **Explanation**: Classes provide structure for data and behavior.

python Copy code

```python
class Product:
    def __init__(self, name, price):
        self.name = name
        self.price = price

class ProductCatalog:
    def __init__(self):
        self.products = []

    def add_product(self, product):
        self.products.append(product)

    def get_products(self):
        return self.products

product1 = Product("Phone", 599)
product2 = Product("Laptop", 1299)
catalog = ProductCatalog()
catalog.add_product(product1)
catalog.add_product(product2)
products = catalog.get_products()
# products: [Product(name='Phone', price=599), Product(name='Laptop
```

1. **Class for a Stack Data Structure**:
   - **Purpose**: Class for a stack data structure.

- **Explanation**: Classes are useful for abstract data structures.

python Copy code

```python
class Stack:
    def __init__(self):
        self.items = []

    def push(self, item):
        self.items.append(item)

    def pop(self):
        if not self.is_empty():
            return self.items.pop()
        else:
            print("Stack is empty.")

    def is_empty(self):
        return len(self.items) == 0

stack = Stack()
stack.push(1)
stack.push(2)
stack.push(3)
popped_item = stack.pop()  # 3
```

1. **Class for a Recipe Book**:
   - **Purpose**: Class for managing a collection of recipes.
   - **Explanation**: Classes are suitable for organizing and managing data.

python Copy code

```python
class Recipe:
    def __init__(self, name, ingredients):
        self.name = name
        self.ingredients = ingredients

class RecipeBook:
    def __init__(self):
        self.recipes = []

    def add_recipe(self, recipe):
        self.recipes.append(recipe)

    def get_recipes(self):
        return self.recipes

recipe1 = Recipe("Pasta", ["Pasta", "Tomato Sauce", "Cheese"])
recipe2 = Recipe("Salad", ["Lettuce", "Tomato", "Cucumber"])
book = RecipeBook()
book.add_recipe(recipe1)
book.add_recipe(recipe2)
recipes = book.get_recipes()
# recipes: [Recipe(name='Pasta', ingredients=['Pasta', 'Tomato Sauc
```

1. **Class for a Timer**:
   - **Purpose**: Class for a timer with start and stop functionality.
   - **Explanation**: Classes are used to encapsulate behavior.

python Copy code

```python
import time

class Timer:
    def __init__(self):
        self.start_time = None
        self.end_time = None

    def start(self):
        self.start_time = time.time()

    def stop(self):
        self.end_time = time.time()
        return self.end_time - self.start_time
```

```
timer = Timer()
timer.start()
# Perform some task...
elapsed_time = timer.stop()
# elapsed_time: Time in seconds
```

In these examples, you can see how functions are used for encapsulating specific tasks and code reuse, while classes are employed for modeling complex entities, supporting inheritance, maintaining state, and organizing data and behavior. The choice between functions and classes depends on the nature of your code and the problem you're solving.