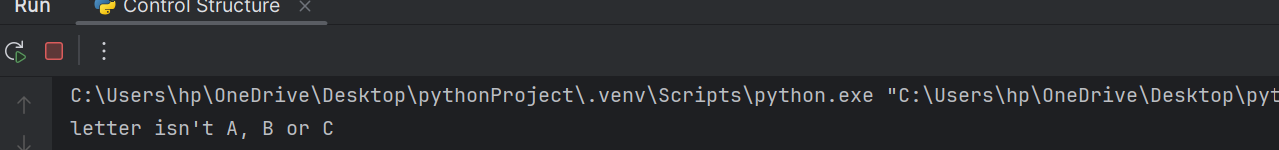
**Assignment 7**

**Control Structure**

**If,else,elif:**

letter = "A"  
if letter == "B":  
 print("letter is B")  
elif letter == "C":  
 print("letter is C")  
else:  
 print("letter isn't A, B or C")

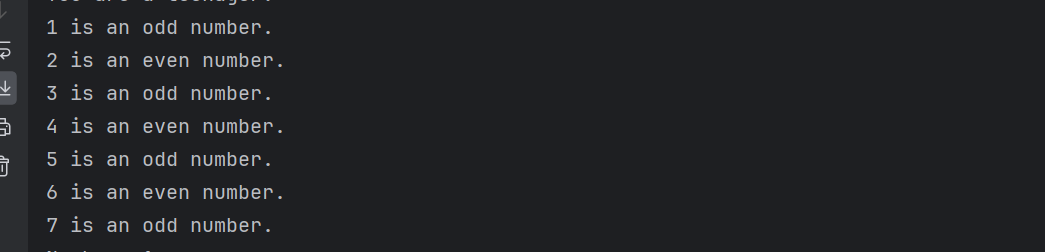
**Output:**



**For loop:**

for i in range(1, 8):  
 if i % 2 == 0:  
 print(f"{i} is an even number.")  
 else:  
 print(f"{i} is an odd number.")

**Output:**



**While loop:**

counter = 1  
while counter <= 5:  
 print("Number:", counter)  
 counter += 1

**Output:**

A black rectangle with white text

Description automatically generated

**Nested Loop:**

for i in range(3):  
 for j in range(5):  
 print(f"({i}, {j})", end=" ")  
 print()

**Output:**

A group of numbers on a black background

Description automatically generated

**Break,Continue,Pass:**

for i in range(1, 11):  
 if i % 2 == 0:  
 print(f"Even number: {i}")  
 continue  
 elif i == 7:  
 print("Breaking the loop at 7")  
 break  
 else:  
 pass  
  
 print(f"Number: {i}")  
print("Loop finished.")

**Output:**

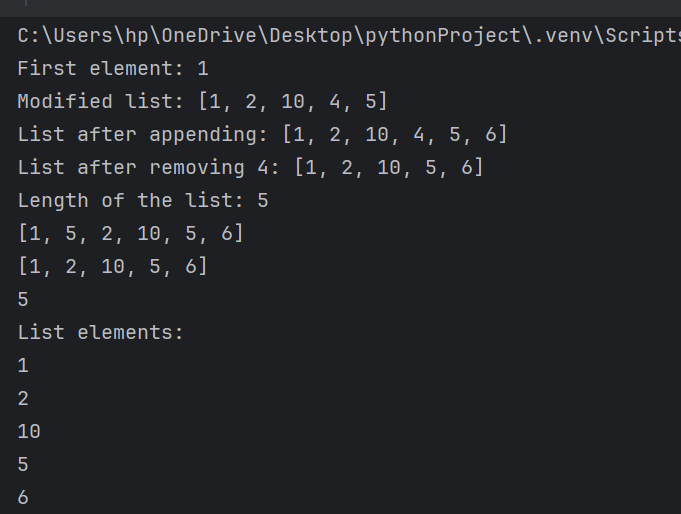
A screenshot of a computer

Description automatically generated

**Lists And It’s Methods:**

my\_list = [1, 2, 3, 4, 5]  
print("First element:", my\_list[0])  
my\_list[2] = 10  
print("Modified list:", my\_list)  
my\_list.append(6)  
print("List after appending:", my\_list)  
my\_list.remove(4)  
print("List after removing 4:", my\_list)  
print("Length of the list:", len(my\_list))  
my\_list.insert(1, 5)  
print(my\_list)  
popped\_value = my\_list.pop(1)  
print(my\_list)  
print(popped\_value)  
print("List elements:")  
for item in my\_list:  
 print(item)

**Output:**



**Slicing:**

mylist = [1, 2, 3, 4, 5,6]  
sublist = mylist[1:4]  
print(sublist)  
sublist\_with\_step = mylist[::2]  
print(sublist\_with\_step)  
last\_two\_elements = mylist[-2:]  
print(last\_two\_elements)

**Output:**

A blue rectangle with white text

Description automatically generated

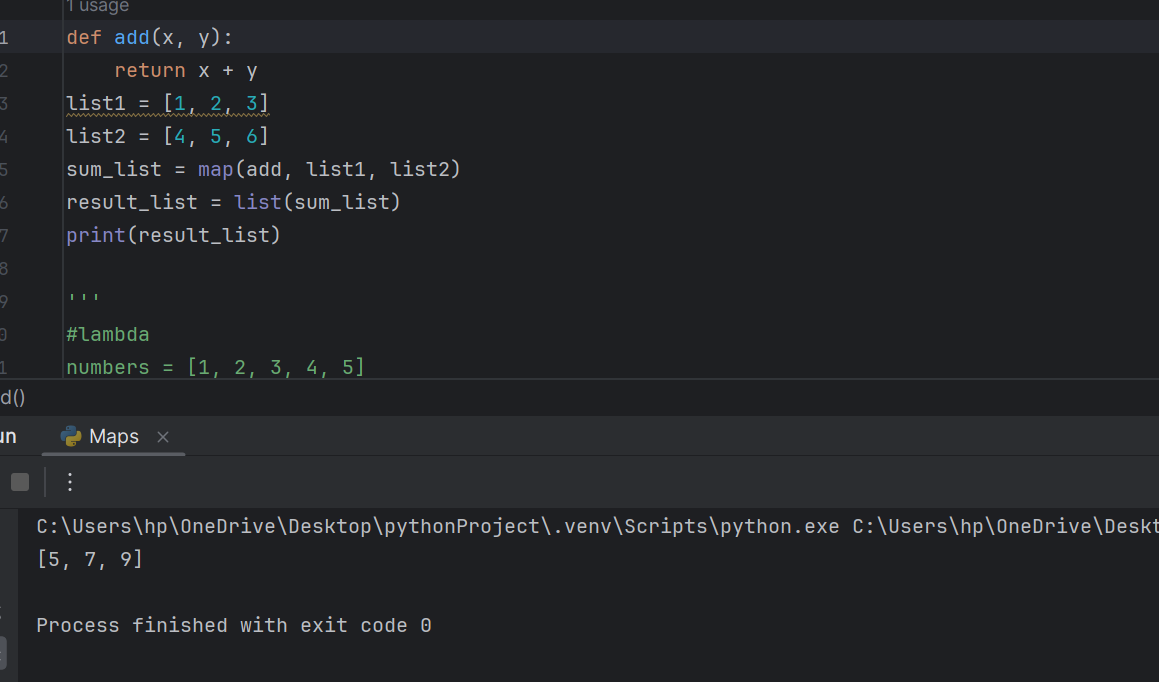
**Dictionary:**

my\_dict = {"name": "John", "age": 25, "city": "New York"}  
print("Name:", my\_dict["name"])  
print("Age:", my\_dict["age"])  
my\_dict["age"] = 26  
print("Updated age:", my\_dict["age"])  
my\_dict["occupation"] = "Engineer"  
print("Updated dictionary:", my\_dict)  
del my\_dict["city"]  
print("Dictionary after removing 'city':", my\_dict)  
  
if "city" in my\_dict:  
 print("City:", my\_dict["city"])  
else:  
 print("City not found in the dictionary.")  
keys = my\_dict.keys()  
print("Keys:", keys)  
values = my\_dict.values()  
print("Values:", values)  
items = my\_dict.items()  
print("Items:", items)  
occupation = my\_dict.pop("occupation")  
print("Removed occupation:", occupation)  
print("Updated dictionary:", my\_dict)  
new\_data = {"country": "USA", "age": 27}  
my\_dict.update(new\_data)  
print("Updated dictionary:", my\_dict)

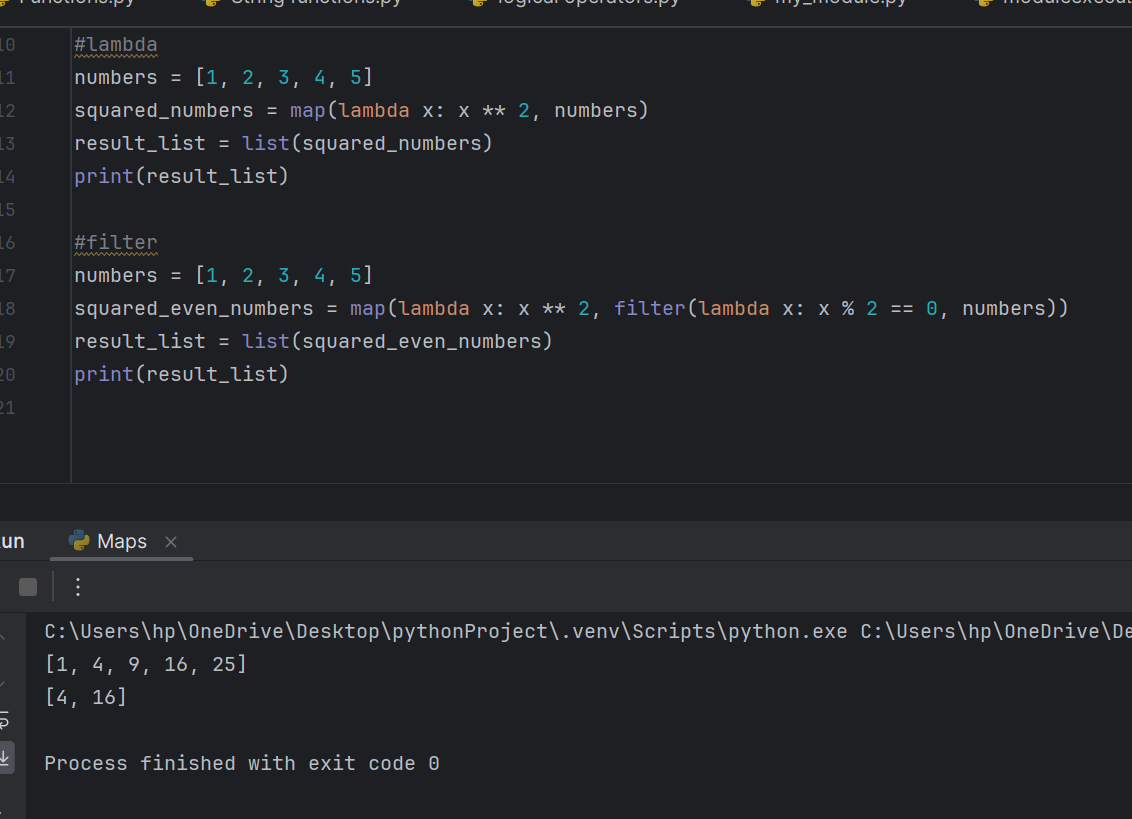
**Output:**



**MAPS**



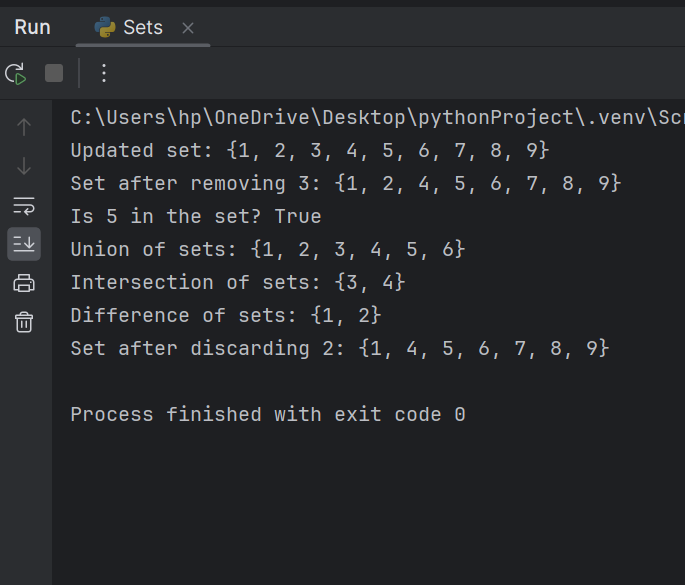
**Maps with Lambda and Filter:**



**Set And It’s Methods:**

my\_set = {1, 2, 3, 4, 5}  
  
# Adding elements to the set  
my\_set.add(6)  
my\_set.update({7, 8, 9})  
print("Updated set:", my\_set)  
  
# Removing elements from the set  
my\_set.remove(3)  
print("Set after removing 3:", my\_set)  
  
# Checking membership  
print("Is 5 in the set?", 5 in my\_set)  
  
# Set operations  
set1 = {1, 2, 3, 4}  
set2 = {3, 4, 5, 6}  
  
# Union  
union\_set = set1.union(set2)  
print("Union of sets:", union\_set)  
  
# Intersection  
intersection\_set = set1.intersection(set2)  
print("Intersection of sets:", intersection\_set)  
  
# Difference  
difference\_set = set1.difference(set2)  
print("Difference of sets:", difference\_set)  
  
my\_set.discard(2)  
print("Set after discarding 2:", my\_set)

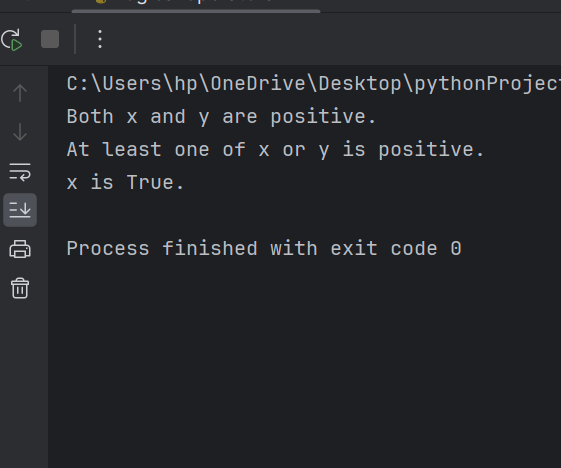
**Output:**



**And,or,not:**

# Example using 'and' operator  
x = 5  
y = 10  
  
if x > 0 and y > 0:  
 print("Both x and y are positive.")  
else:  
 print("At least one of x or y is not positive.")  
  
# Example using 'or' operator  
x = -5  
y = 10  
  
if x > 0 or y > 0:  
 print("At least one of x or y is positive.")  
else:  
 print("Neither x nor y is positive.")  
  
# Example using 'not' operator  
x = True  
if not x:  
 print("x is False.")  
else:  
 print("x is True.")

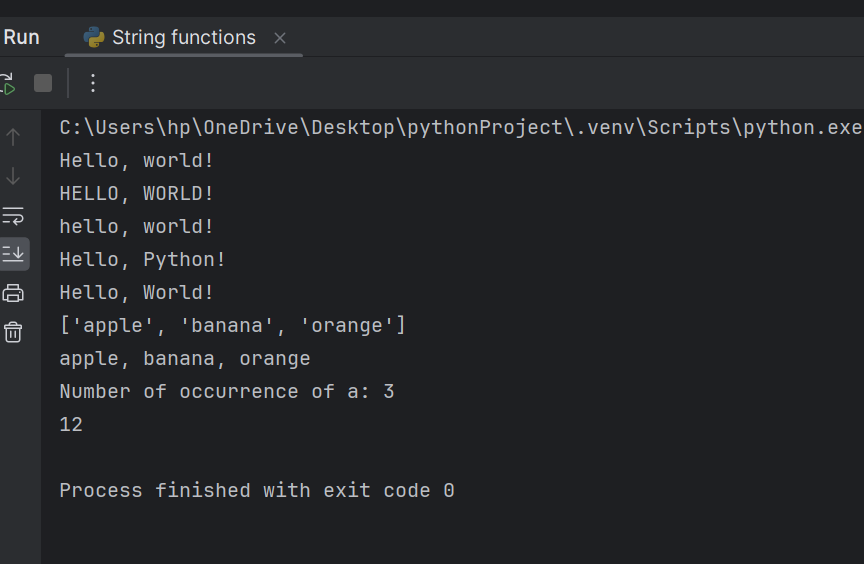
**Output:**



**String Functions:**

#capitalise  
text = "hello, world!"  
print(text.capitalize())  
#upper  
text = "hello, world!"  
print(text.upper())  
#lower  
text = "Hello, World!"  
print(text.lower())  
#replace  
text = "Hello, World!"  
new\_text = text.replace("World", "Python")  
print(new\_text)  
#strip  
text = " Hello, World! "  
stripped\_text = text.strip()  
print(stripped\_text)  
#split  
text = "apple,banana,orange"  
fruits = text.split(',')  
print(fruits)  
#join  
fruits = ['apple', 'banana', 'orange']  
text = ', '.join(fruits)  
print(text) # Output: 'apple, banana, orange'  
#count  
message = 'Have a Good Day'  
print('Number of occurrence of a:', message.count('a'))  
#find  
message = 'Have a Good Day'  
print(message.find('Day'))

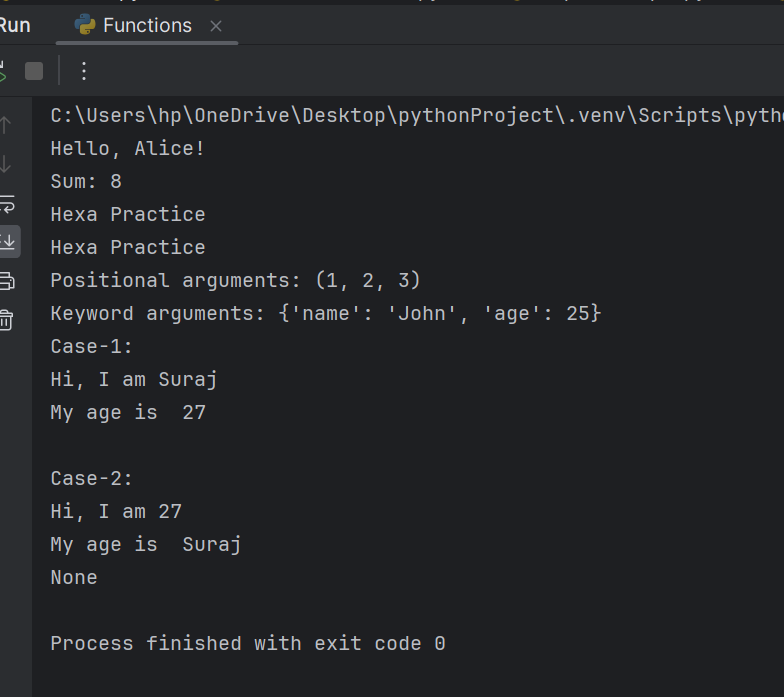
**Output:**



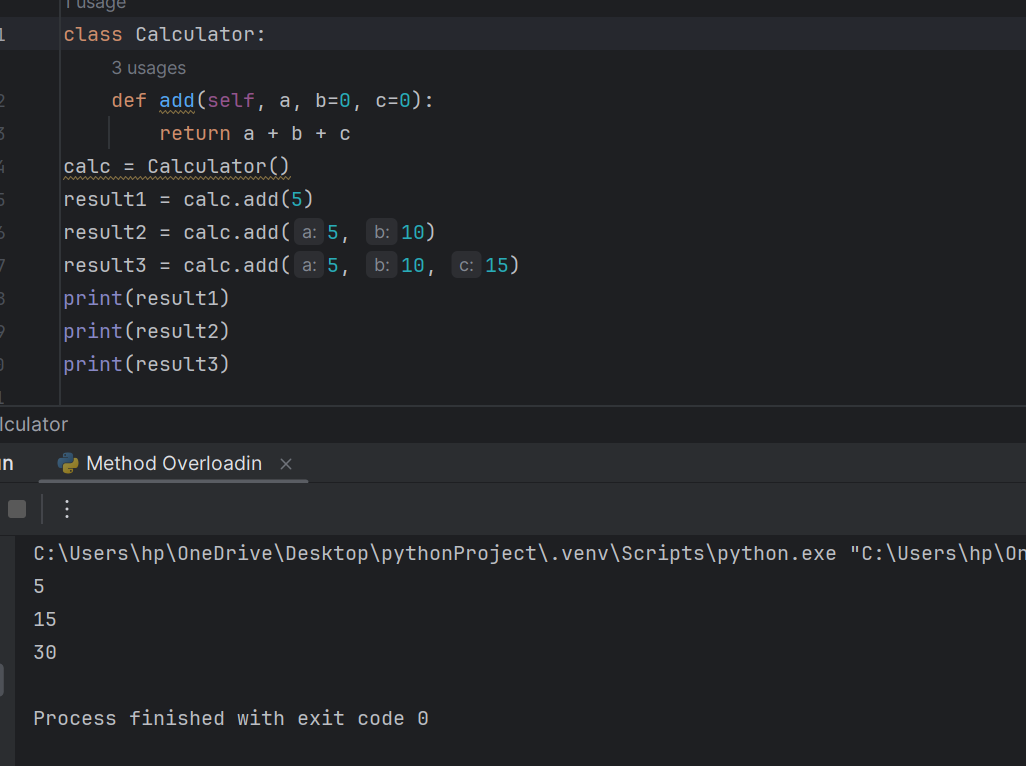
**Functions:**

def greet(name):  
 print(f"Hello, {name}!")  
greet("Alice")  
  
#function with parameters  
def add\_numbers(a, b):  
 return a + b  
result = add\_numbers(5, 3)  
print("Sum:", result)  
  
#default arguments  
def myFun(x, y=50):  
 print("x: ", x)  
 print("y: ", y)  
  
#Keyword Arguments  
def student(firstname, lastname):  
 print(firstname, lastname)  
student(firstname='Hexa', lastname='Practice')  
student(lastname='Practice', firstname='Hexa')  
  
#variable-length arguments  
def print\_args(\*args, \*\*kwargs):  
 print("Positional arguments:", args)  
 print("Keyword arguments:", kwargs)  
  
print\_args(1, 2, 3, name="John", age=25)  
  
  
# Positional argument  
def nameAge(name, age):  
 print("Hi, I am", name)  
 print("My age is ", age)  
print("Case-1:")  
nameAge("Suraj", 27)  
# You will get incorrect output because  
# argument is not in order  
print("\nCase-2:")  
nameAge(27, "Suraj")  
  
#use of docstring  
def evenOdd(x):  
 if (x % 2 == 0):  
 print("even")  
 else:  
 print("odd")  
print(evenOdd.\_\_doc\_\_)

**Output:**



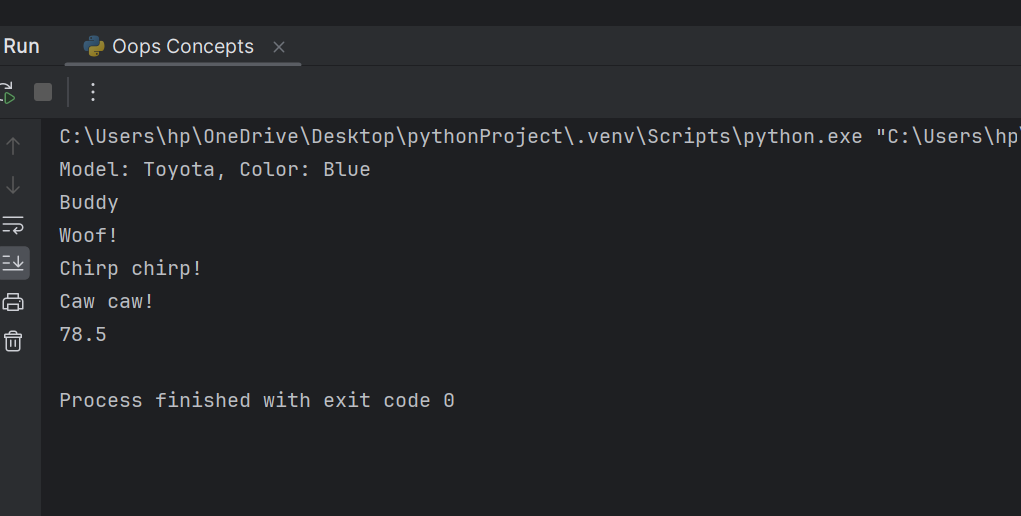
**Method Overloading**



**Oops Concepts:**

#Encapsulation  
class Car:  
 def \_\_init\_\_(self, model, color):  
 self.model = model  
 self.color = color  
  
 def display\_info(self):  
 print(f"Model: {self.model}, Color: {self.color}")  
  
my\_car = Car("Toyota", "Blue")  
my\_car.display\_info()  
#Inheritance  
class Animal:  
 def \_\_init\_\_(self, name):  
 self.name = name  
  
class Dog(Animal):  
 def bark(self):  
 print("Woof!")  
  
my\_dog = Dog("Buddy")  
print(my\_dog.name)  
my\_dog.bark()  
#polymorphism  
class Bird:  
 def make\_sound(self):  
 pass  
  
class Sparrow(Bird):  
 def make\_sound(self):  
 print("Chirp chirp!")  
  
class Crow(Bird):  
 def make\_sound(self):  
 print("Caw caw!")  
  
def make\_bird\_sound(bird):  
 bird.make\_sound()  
  
sparrow = Sparrow()  
crow = Crow()  
  
make\_bird\_sound(sparrow)  
make\_bird\_sound(crow)  
  
#Abstraction  
from abc import ABC, abstractmethod  
  
class Shape(ABC):  
 @abstractmethod  
 def area(self):  
 pass  
  
class Circle(Shape):  
 def \_\_init\_\_(self, radius):  
 self.radius = radius  
  
 def area(self):  
 return 3.14 \* self.radius \* self.radius  
  
circle = Circle(5)  
print(circle.area())

**Output:**



**Modules**

A screen shot of a computer

Description automatically generated

A screenshot of a computer program

Description automatically generated