Assignment-7

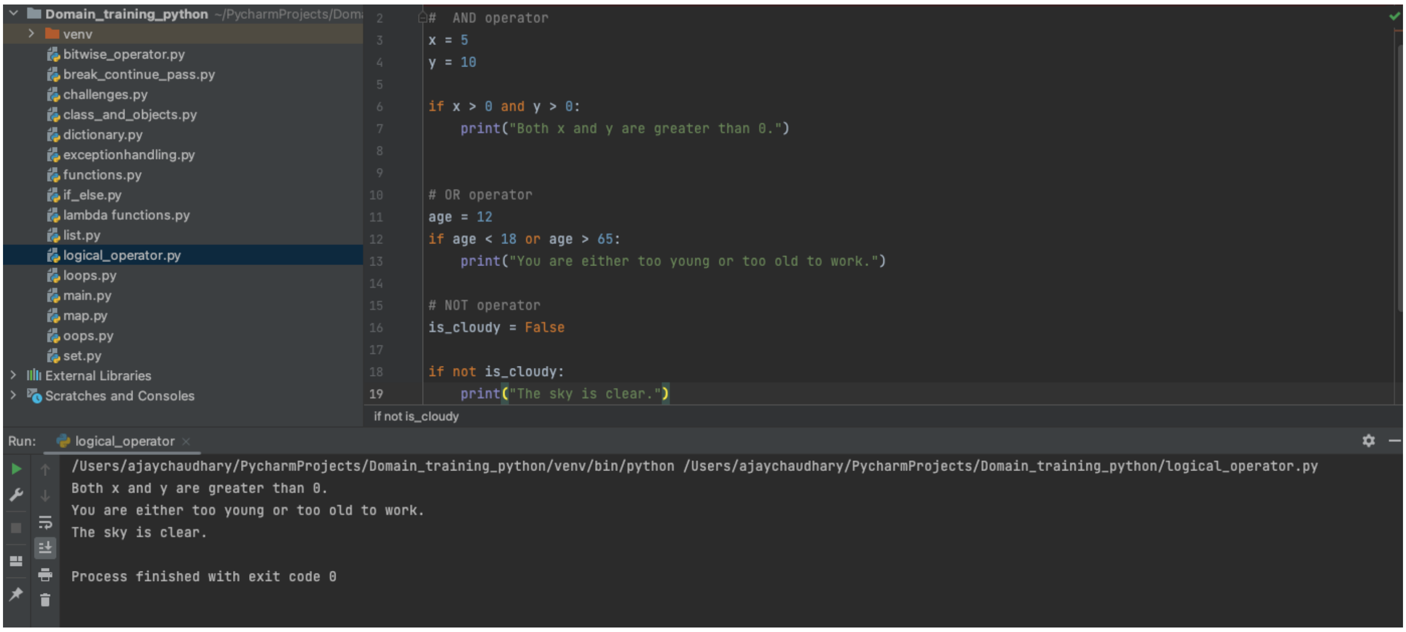
Name- Ajay Chaudhary
Batch- Data Engineering (Batch-01)

Python Logical operator

AND operator
The **and** operator returns **True** if both conditions are true, otherwise, it returns **False**.

OR operator
The **or** operator returns **True** if at least one condition is true, otherwise, it returns **False**.

NOT operator
The **not** operator is a unary operator that returns **True** if the condition is false and vice versa.



```python
#  AND operator
x = 5
y = 10

if x > 0 and y > 0:
    print("Both x and y are greater than 0.")


# OR operator
age = 12
if age < 18 or age > 65:
    print("You are either too young or too old to work.")

# NOT operator
is_cloudy = False

if not is_cloudy:
    print("The sky is clear.")
```

```
/Users/ajaychaudhary/PycharmProjects/Domain_training_python/venv/bin/python /Users/ajaychaudhary/PycharmProjects/Domain_training_python/logical_operator.py
Both x and y are greater than 0.
You are either too young or too old to work.
The sky is clear.

Process finished with exit code 0
```

Python bitwise operator

1. Bitwise AND (&):
    - Sets each bit to 1 if both corresponding bits are 1.

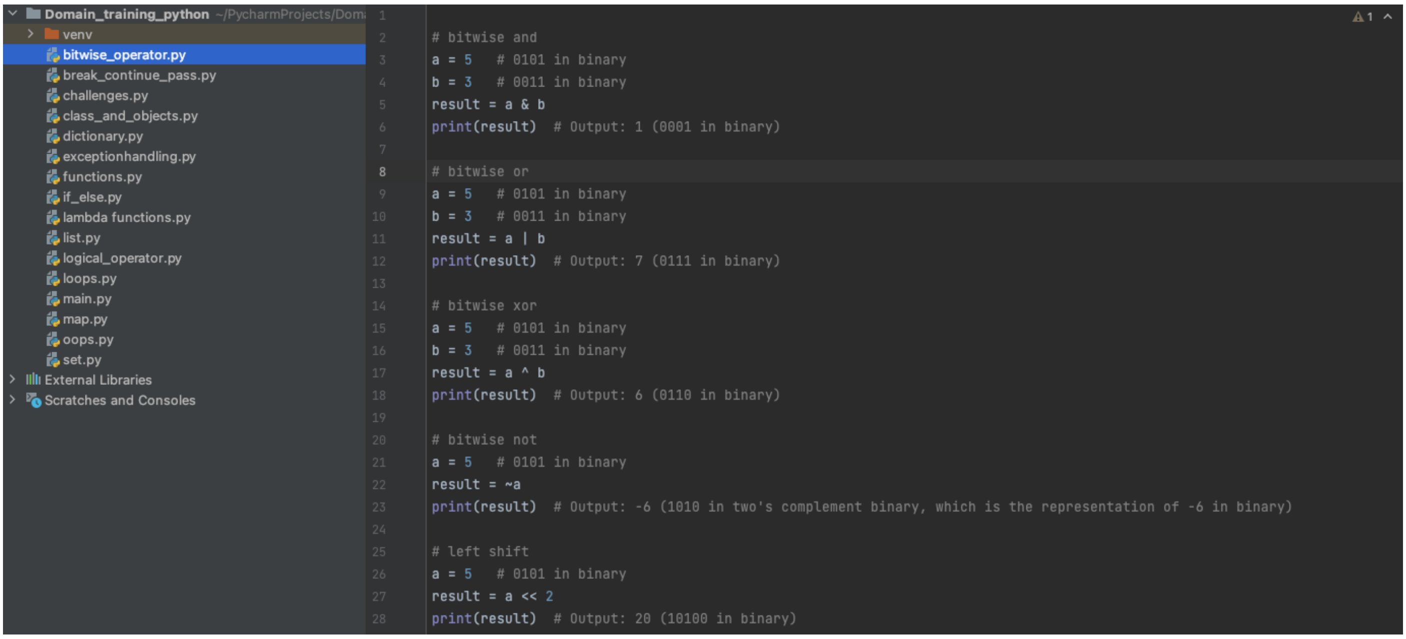
2. Bitwise OR (|):
    - Sets each bit to 1 if at least one of the corresponding bits is 1.

3. Bitwise XOR (^):
    - Sets each bit to 1 if exactly one of the corresponding bits is 1.

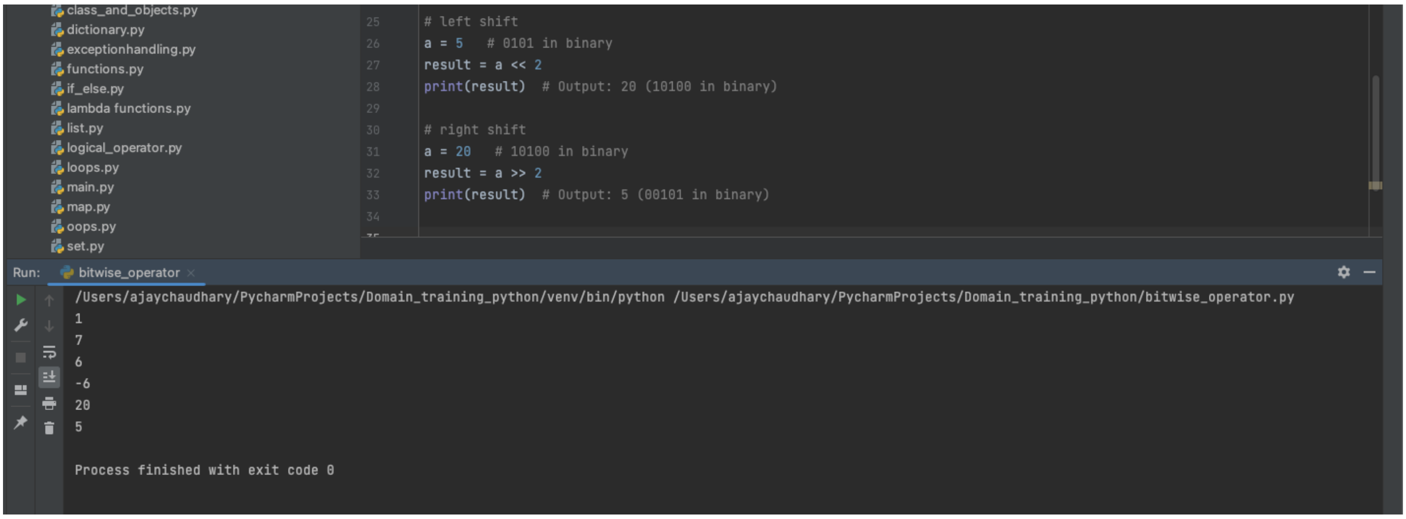
4. Bitwise NOT (~):
    - Inverts each bit; 0 becomes 1 and 1 becomes 0.

5. Left Shift (<<):
    - Shifts the bits to the left by a specified number of positions, filling the vacant positions with 0.

6. Right Shift (>>):
    - Shifts the bits to the right by a specified number of positions, filling the vacant positions based on the sign bit for signed integers.



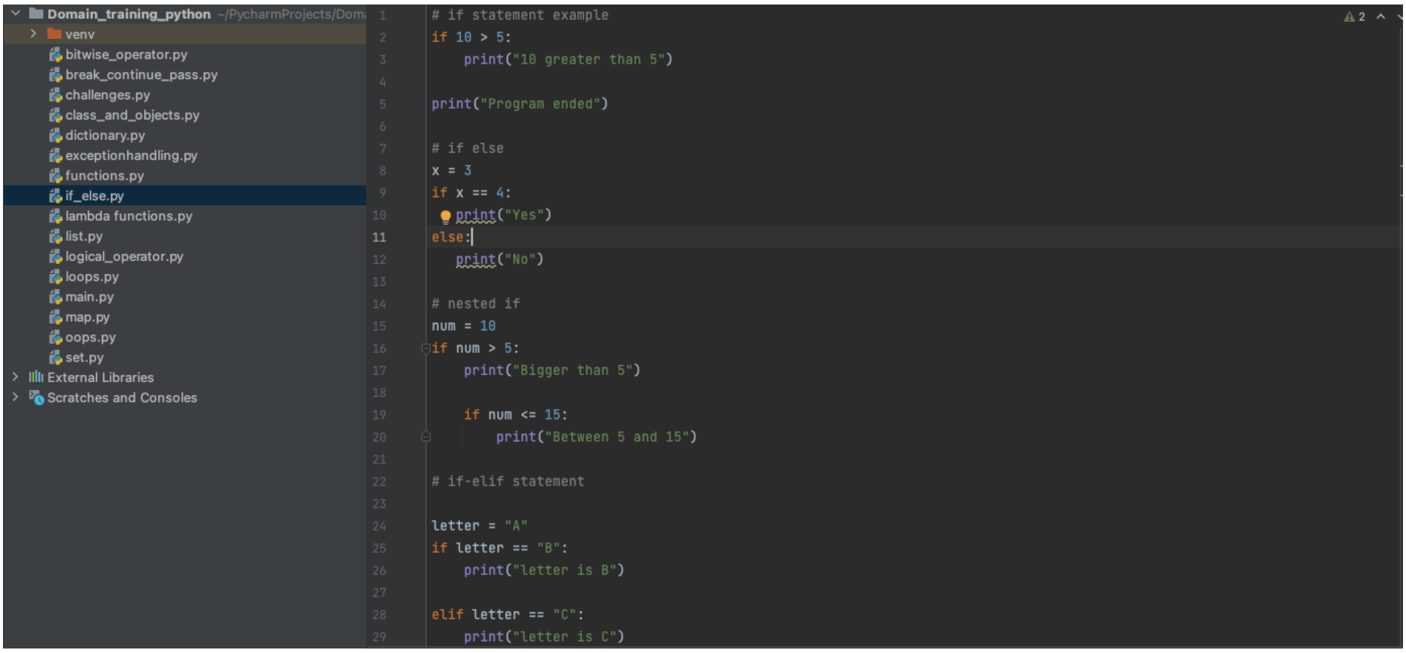
```python

# bitwise and
a = 5   # 0101 in binary
b = 3   # 0011 in binary
result = a & b
print(result)  # Output: 1 (0001 in binary)

# bitwise or
a = 5   # 0101 in binary
b = 3   # 0011 in binary
result = a | b
print(result)  # Output: 7 (0111 in binary)

# bitwise xor
a = 5   # 0101 in binary
b = 3   # 0011 in binary
result = a ^ b
print(result)  # Output: 6 (0110 in binary)

# bitwise not
a = 5   # 0101 in binary
result = ~a
print(result)  # Output: -6 (1010 in two's complement binary, which is the representation of -6 in binary)

# left shift
a = 5   # 0101 in binary
result = a << 2
print(result)  # Output: 20 (10100 in binary)
```



```python
# left shift
a = 5   # 0101 in binary
result = a << 2
print(result)  # Output: 20 (10100 in binary)

# right shift
a = 20   # 10100 in binary
result = a >> 2
print(result)  # Output: 5 (00101 in binary)
```

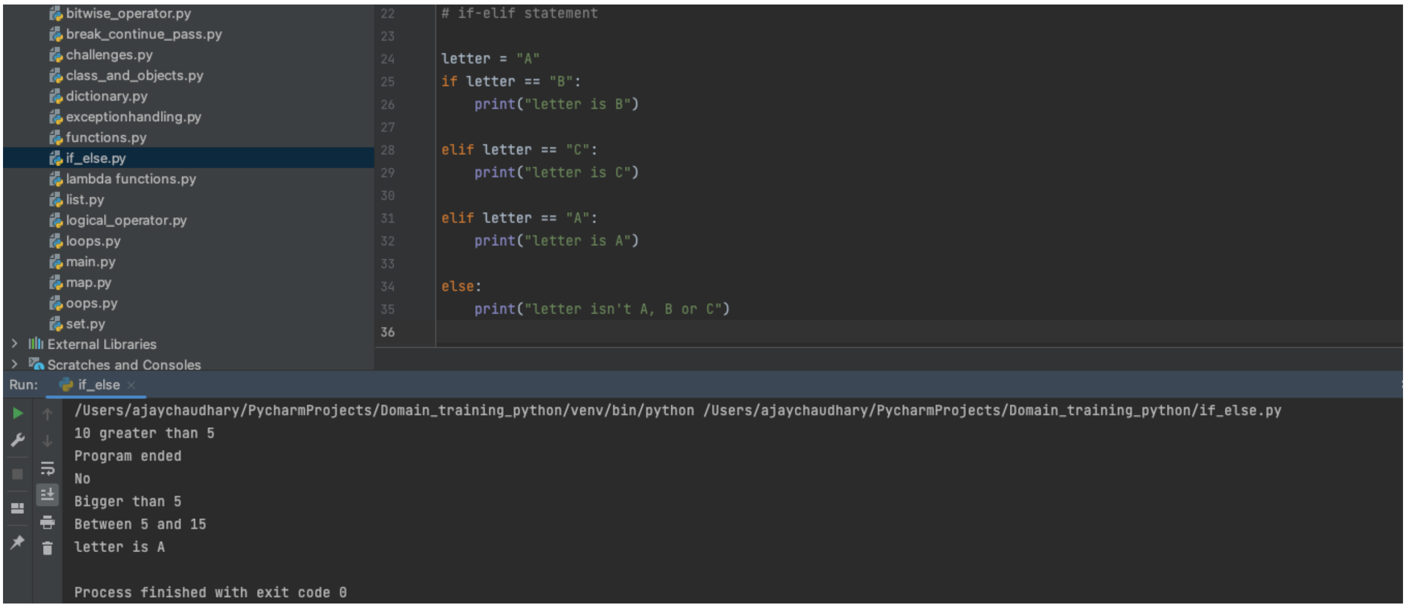
```
/Users/ajaychaudhary/PycharmProjects/Domain_training_python/venv/bin/python /Users/ajaychaudhary/PycharmProjects/Domain_training_python/bitwise_operator.py
1
7
6
-6
20
5

Process finished with exit code 0
```

If else statements, nested if-



```python
# if statement example
if 10 > 5:
    print("10 greater than 5")

print("Program ended")

# if else
x = 3
if x == 4:
   print("Yes")
else:
   print("No")

# nested if
num = 10
if num > 5:
    print("Bigger than 5")

    if num <= 15:
        print("Between 5 and 15")

# if-elif statement

letter = "A"
if letter == "B":
    print("letter is B")

elif letter == "C":
    print("letter is C")
```

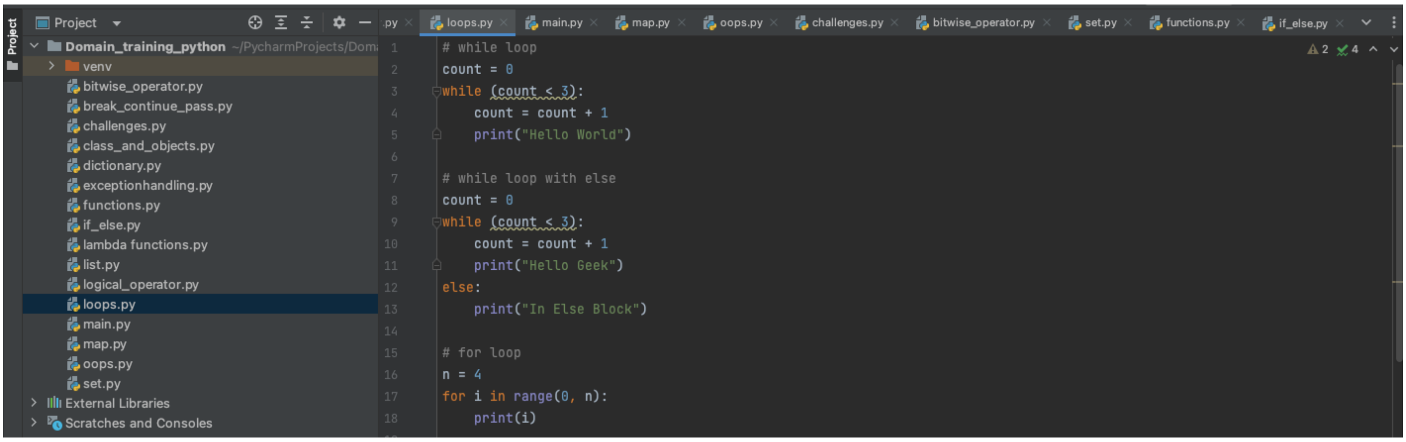
If-elif statement



```python
# if-elif statement

letter = "A"
if letter == "B":
    print("letter is B")

elif letter == "C":
    print("letter is C")

elif letter == "A":
    print("letter is A")

else:
    print("letter isn't A, B or C")
```

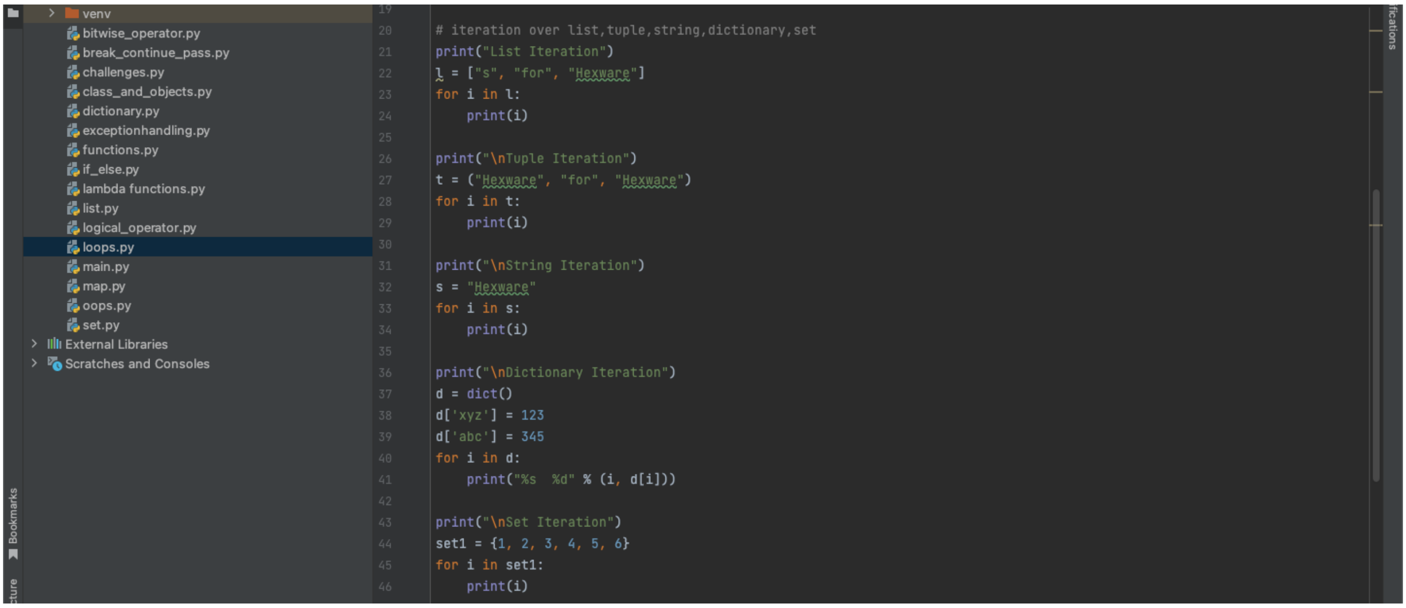
```
/Users/ajaychaudhary/PycharmProjects/Domain_training_python/venv/bin/python /Users/ajaychaudhary/PycharmProjects/Domain_training_python/if_else.py
10 greater than 5
Program ended
No
Bigger than 5
Between 5 and 15
letter is A

Process finished with exit code 0
```

Loops(while,while loop with else & for loop)



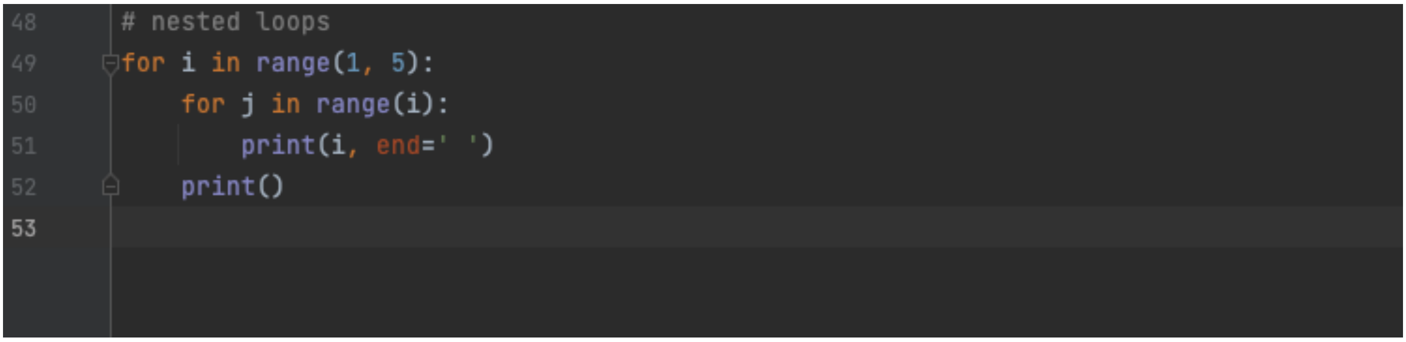
```python
# while loop
count = 0
while (count < 3):
    count = count + 1
    print("Hello World")

# while loop with else
count = 0
while (count < 3):
    count = count + 1
    print("Hello Geek")
else:
    print("In Else Block")

# for loop
n = 4
for i in range(0, n):
    print(i)
```

Iteration over list, tuple, dictionary, set



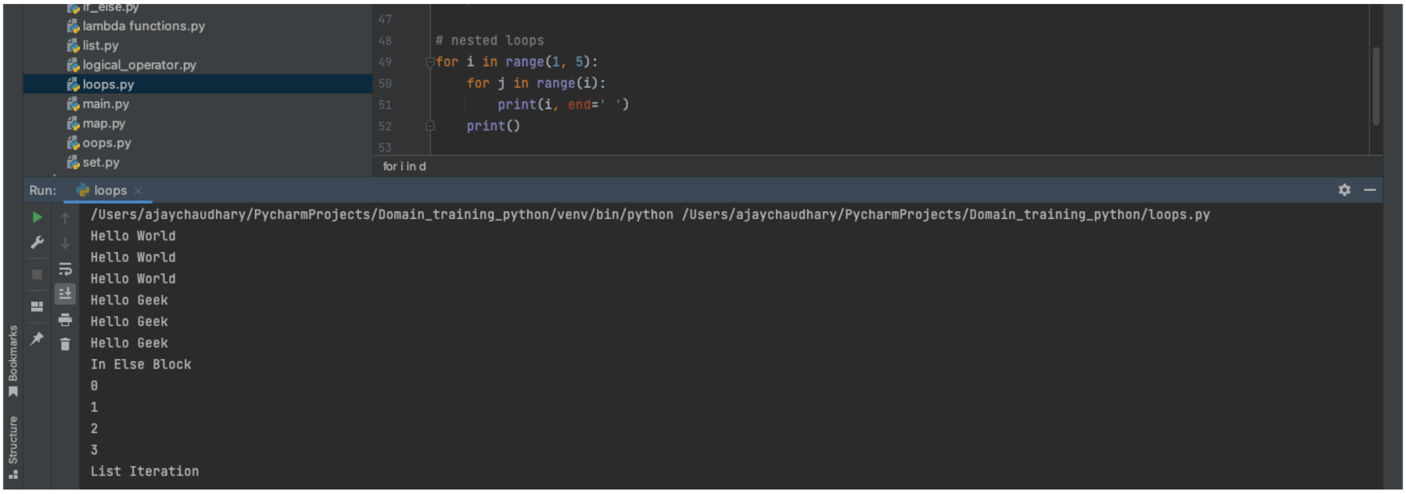
```python
# iteration over list,tuple,string,dictionary,set
print("List Iteration")
l = ["s", "for", "Hexware"]
for i in l:
    print(i)

print("\nTuple Iteration")
t = ("Hexware", "for", "Hexware")
for i in t:
    print(i)

print("\nString Iteration")
s = "Hexware"
for i in s:
    print(i)

print("\nDictionary Iteration")
d = dict()
d['xyz'] = 123
d['abc'] = 345
for i in d:
    print("%s  %d" % (i, d[i]))

print("\nSet Iteration")
set1 = {1, 2, 3, 4, 5, 6}
for i in set1:
    print(i)
```

Nested loops



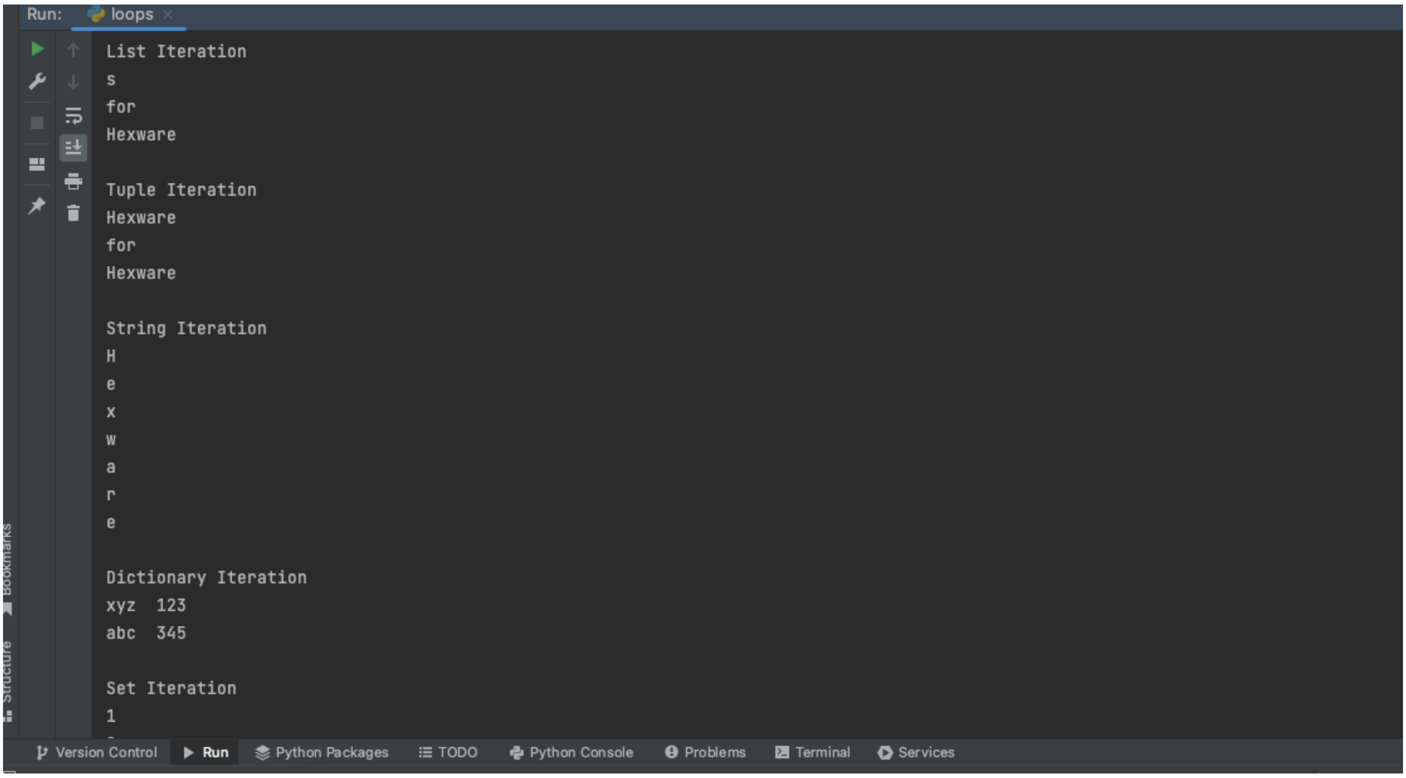
```python
# nested loops
for i in range(1, 5):
    for j in range(i):
        print(i, end=' ')
    print()
```

Outputs:



```python
# nested loops
for i in range(1, 5):
    for j in range(i):
        print(i, end=' ')
    print()
```

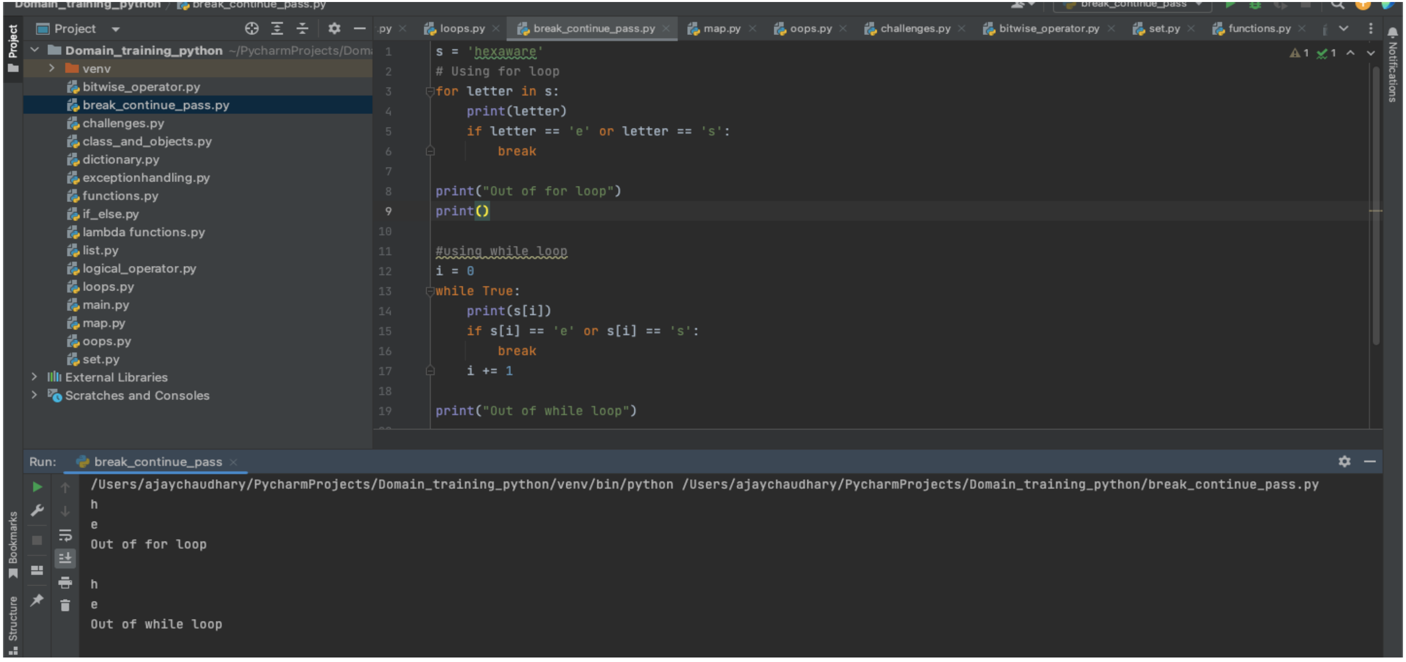
```
/Users/ajaychaudhary/PycharmProjects/Domain_training_python/venv/bin/python /Users/ajaychaudhary/PycharmProjects/Domain_training_python/loops.py
Hello World
Hello World
Hello World
Hello Geek
Hello Geek
Hello Geek
In Else Block
0
1
2
3
List Iteration
```



```
List Iteration
s
for
Hexware

Tuple Iteration
Hexware
for
Hexware

String Iteration
H
e
x
w
a
r
e

Dictionary Iteration
xyz  123
abc  345

Set Iteration
1
```

break Statement:

- The break statement is used to exit a loop prematurely. When encountered within a loop, it terminates the loop and transfers control to the next statement after the loop.

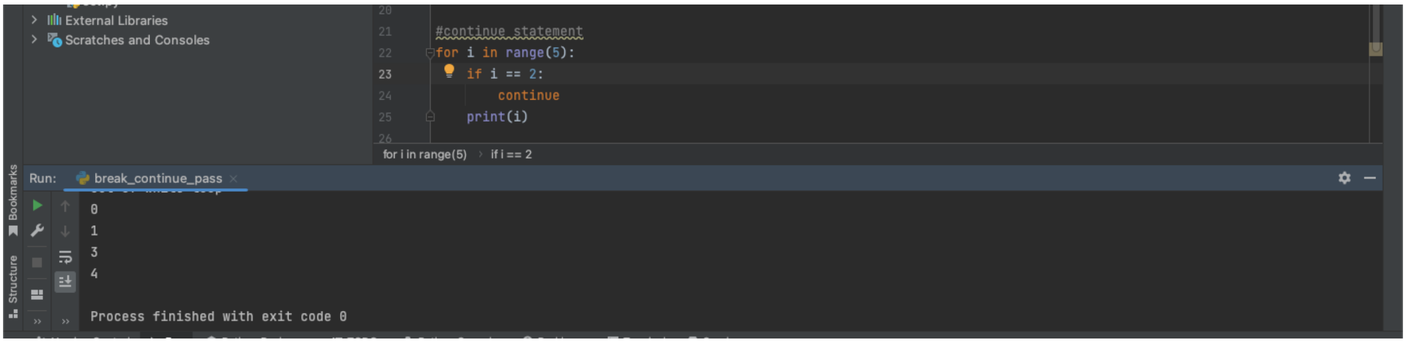


```python
s = 'hexaware'
# Using for loop
for letter in s:
    print(letter)
    if letter == 'e' or letter == 's':
        break

print("Out of for loop")
print()

#using while loop
i = 0
while True:
    print(s[i])
    if s[i] == 'e' or s[i] == 's':
        break
    i += 1

print("Out of while loop")
```

```
/Users/ajaychaudhary/PycharmProjects/Domain_training_python/venv/bin/python /Users/ajaychaudhary/PycharmProjects/Domain_training_python/break_continue_pass.py
h
e
Out of for loop

h
e
Out of while loop
```

continue Statement:

- The continue statement is used to skip the rest of the code inside a loop for the current iteration and move to the next iteration.

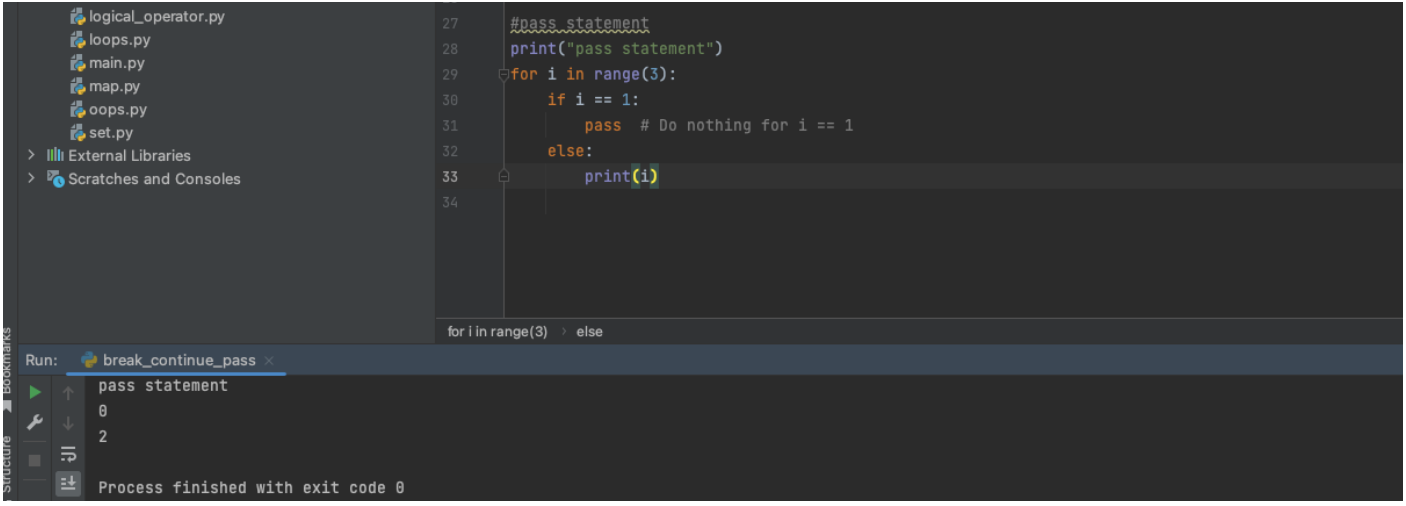


```python
#continue statement
for i in range(5):
    if i == 2:
        continue
    print(i)
```

```
0
1
3
4

Process finished with exit code 0
```

pass Statement:

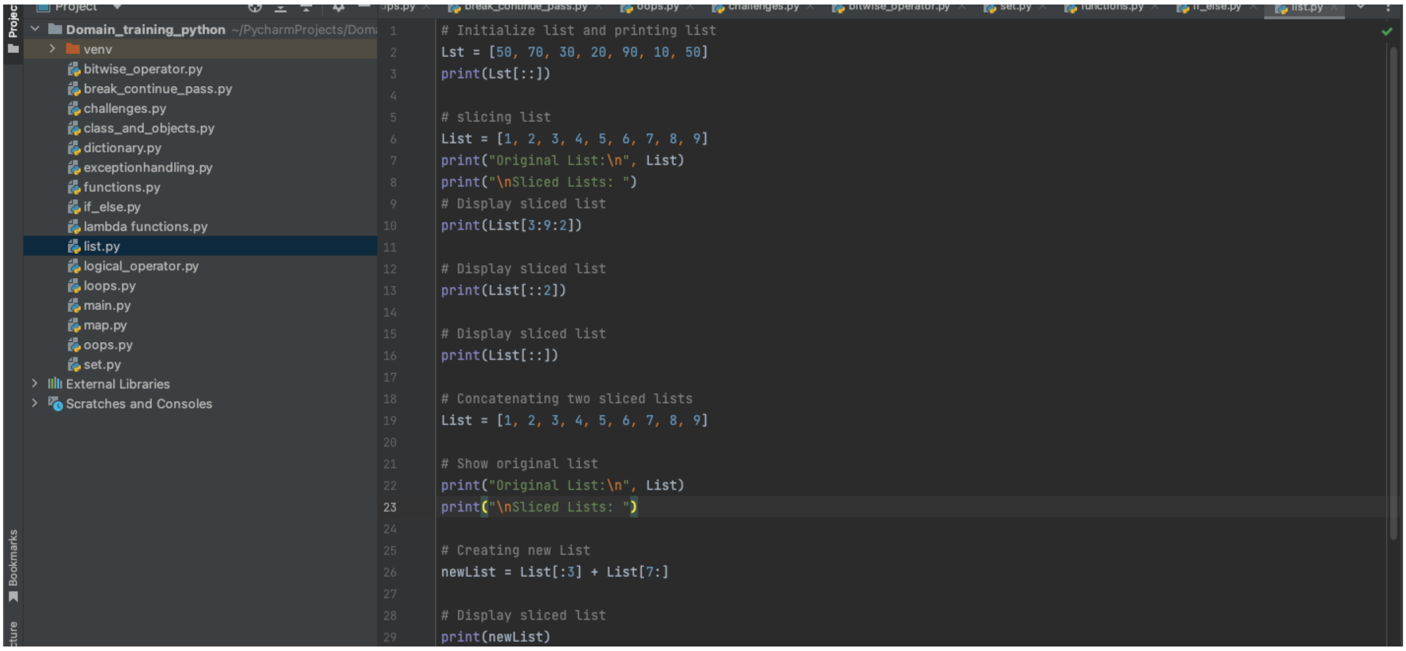
- The pass statement is a no-operation statement. It serves as a placeholder where syntactically some code is required but no action is desired or necessary. It essentially does nothing.



```python
#pass statement
print("pass statement")
for i in range(3):
    if i == 1:
        pass  # Do nothing for i == 1
    else:
        print(i)
```

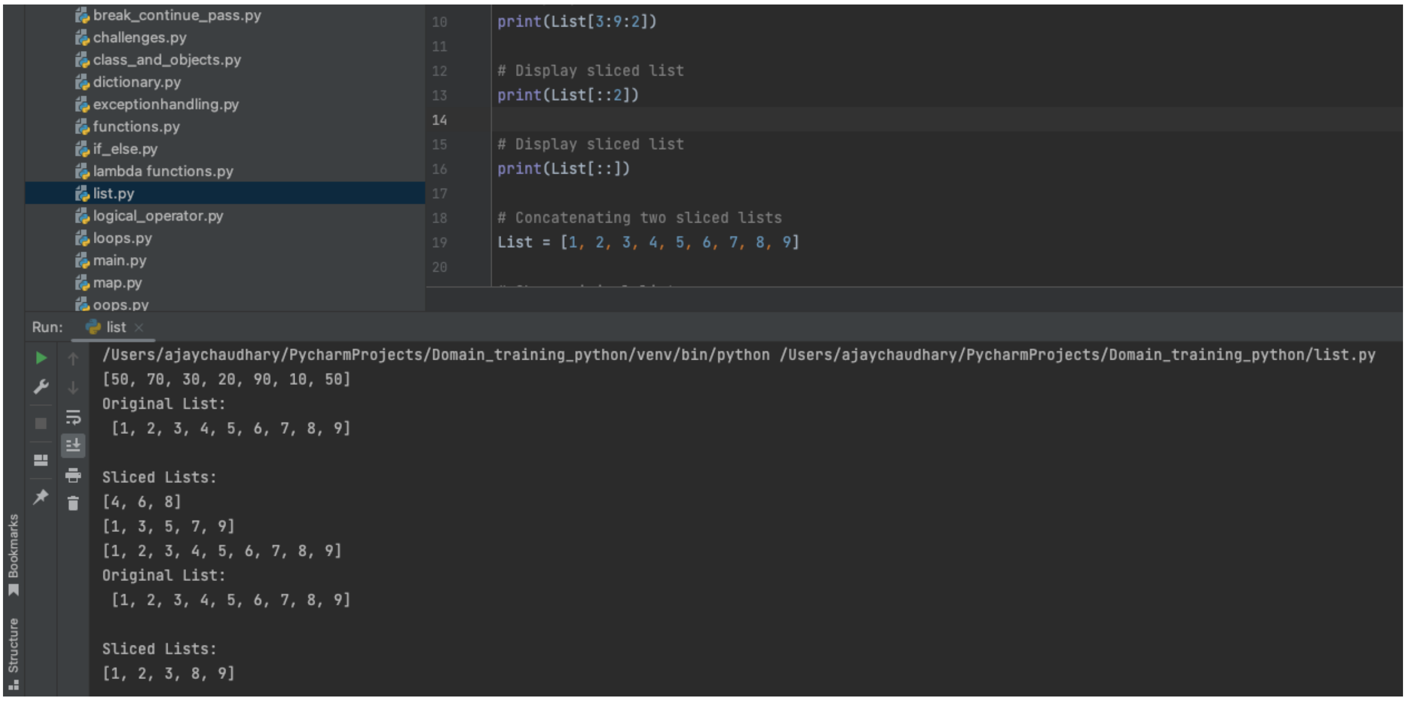
```
pass statement
0
2

Process finished with exit code 0
```

List and its methods



```python
# Initialize list and printing list
Lst = [50, 70, 30, 20, 90, 10, 50]
print(Lst[::])

# slicing list
List = [1, 2, 3, 4, 5, 6, 7, 8, 9]
print("Original List:\n", List)
print("\nSliced Lists: ")
# Display sliced list
print(List[3:9:2])

# Display sliced list
print(List[::2])

# Display sliced list
print(List[::])

# Concatenating two sliced lists
List = [1, 2, 3, 4, 5, 6, 7, 8, 9]

# Show original list
print("Original List:\n", List)
print("\nSliced Lists: ")

# Creating new List
newList = List[:3] + List[7:]

# Display sliced list
print(newList)
```

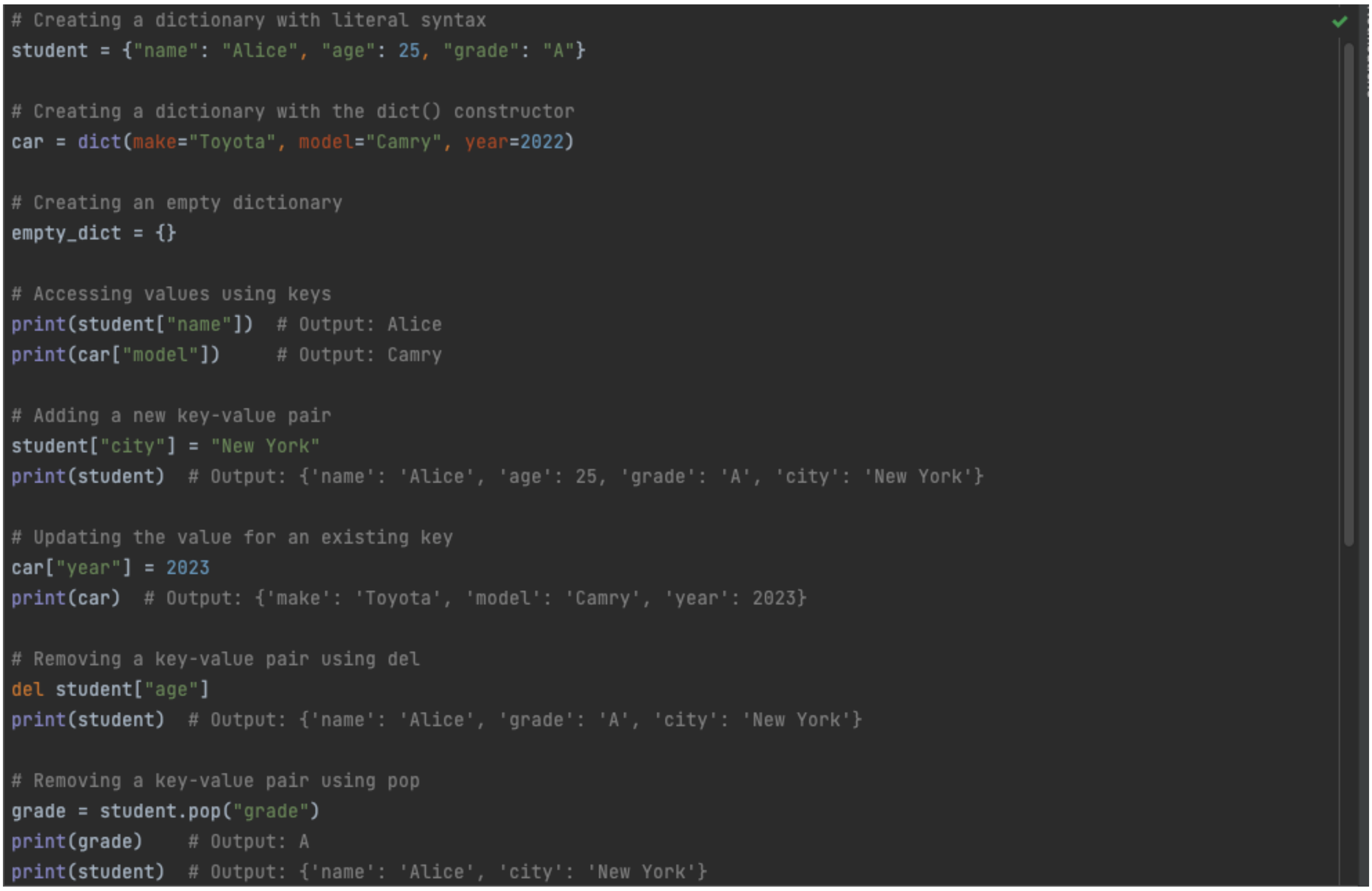
Slicing



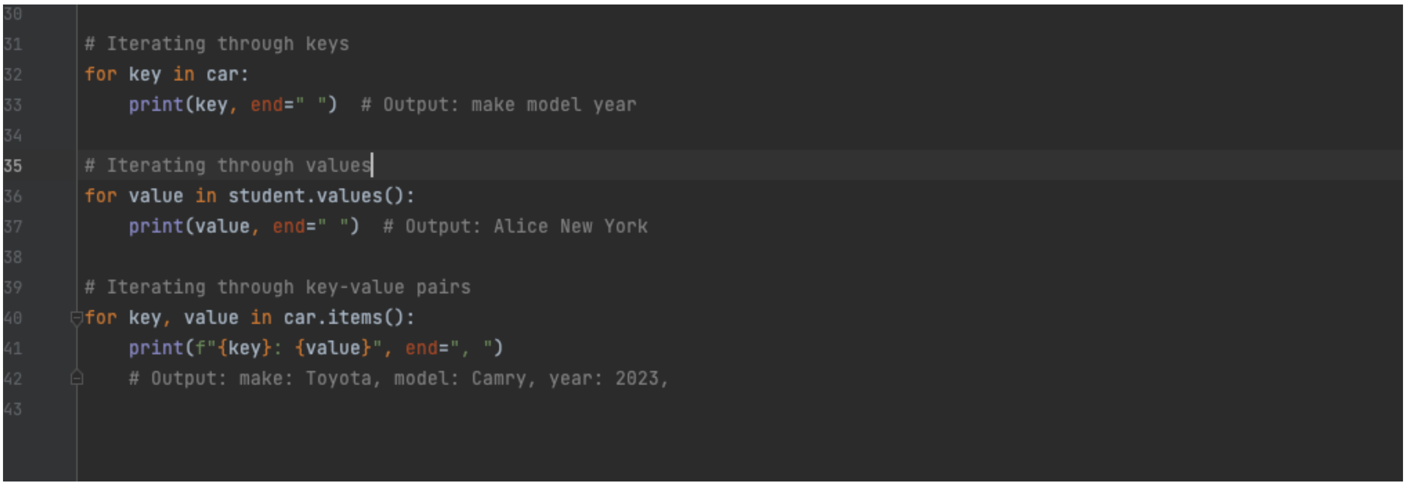
```python
print(List[3:9:2])

# Display sliced list
print(List[::2])

# Display sliced list
print(List[::])

# Concatenating two sliced lists
List = [1, 2, 3, 4, 5, 6, 7, 8, 9]
```

```
/Users/ajaychaudhary/PycharmProjects/Domain_training_python/venv/bin/python /Users/ajaychaudhary/PycharmProjects/Domain_training_python/list.py
[50, 70, 30, 20, 90, 10, 50]
Original List:
 [1, 2, 3, 4, 5, 6, 7, 8, 9]

Sliced Lists:
[4, 6, 8]
[1, 3, 5, 7, 9]
[1, 2, 3, 4, 5, 6, 7, 8, 9]
Original List:
 [1, 2, 3, 4, 5, 6, 7, 8, 9]

Sliced Lists:
[1, 2, 3, 8, 9]
```

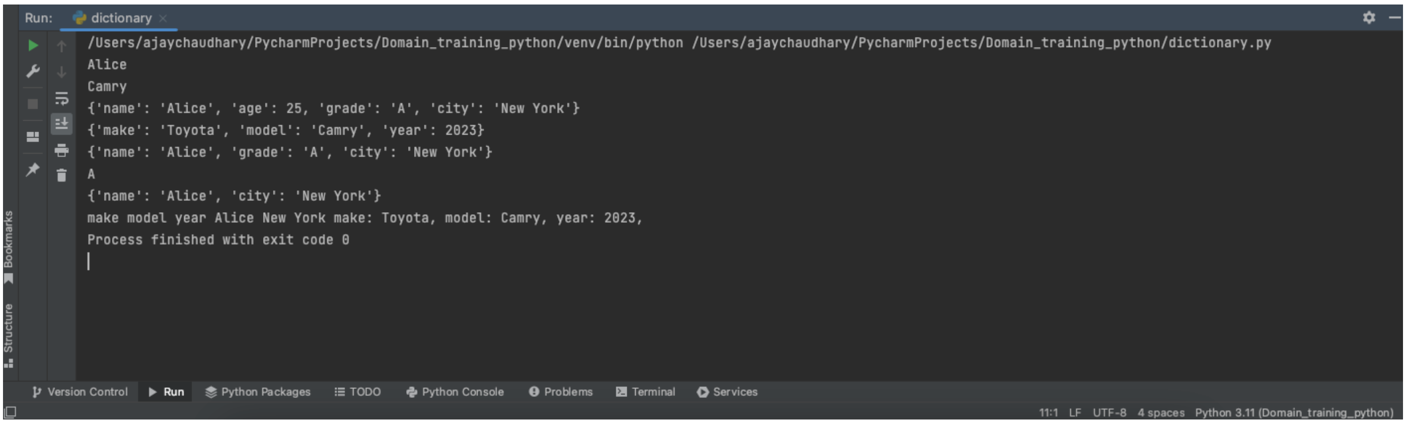
Dictionary and its method



```
# Creating a dictionary with literal syntax
student = {"name": "Alice", "age": 25, "grade": "A"}

# Creating a dictionary with the dict() constructor
car = dict(make="Toyota", model="Camry", year=2022)

# Creating an empty dictionary
empty_dict = {}

# Accessing values using keys
print(student["name"])  # Output: Alice
print(car["model"])     # Output: Camry

# Adding a new key-value pair
student["city"] = "New York"
print(student)  # Output: {'name': 'Alice', 'age': 25, 'grade': 'A', 'city': 'New York'}

# Updating the value for an existing key
car["year"] = 2023
print(car)  # Output: {'make': 'Toyota', 'model': 'Camry', 'year': 2023}

# Removing a key-value pair using del
del student["age"]
print(student)  # Output: {'name': 'Alice', 'grade': 'A', 'city': 'New York'}

# Removing a key-value pair using pop
grade = student.pop("grade")
print(grade)    # Output: A
print(student)  # Output: {'name': 'Alice', 'city': 'New York'}
```

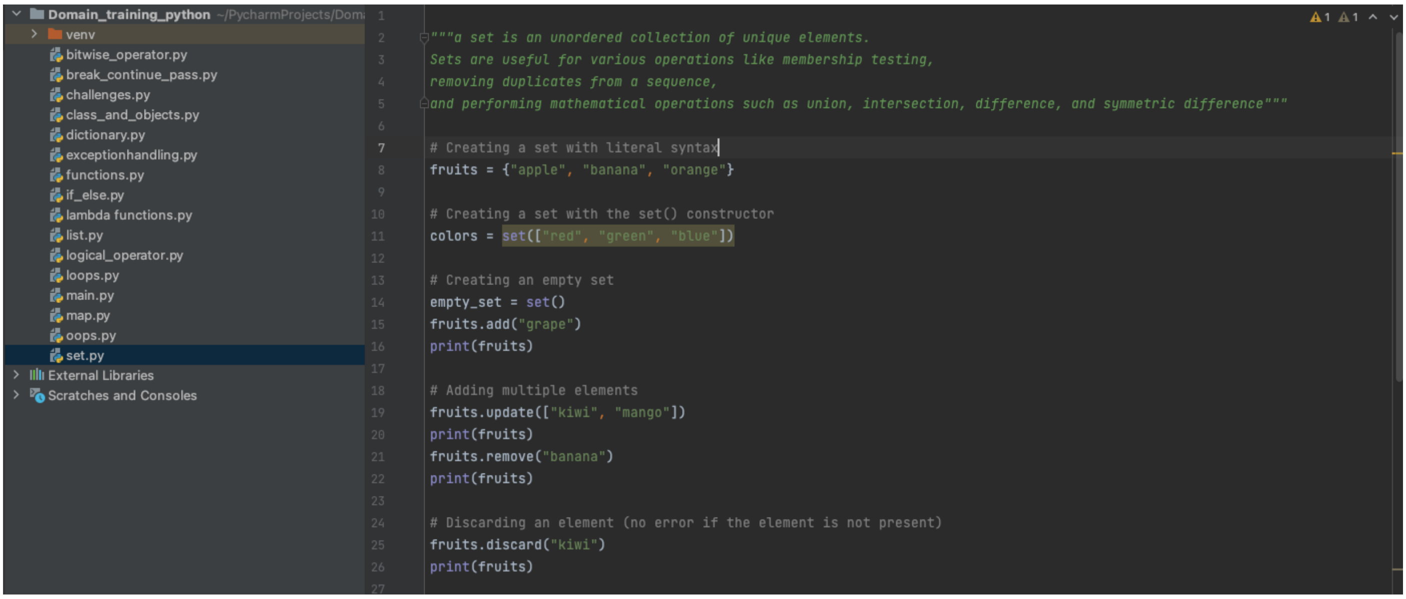


```

# Iterating through keys
for key in car:
    print(key, end=" ")  # Output: make model year

# Iterating through values
for value in student.values():
    print(value, end=" ")  # Output: Alice New York

# Iterating through key-value pairs
for key, value in car.items():
    print(f"{key}: {value}", end=", ")
    # Output: make: Toyota, model: Camry, year: 2023,

```

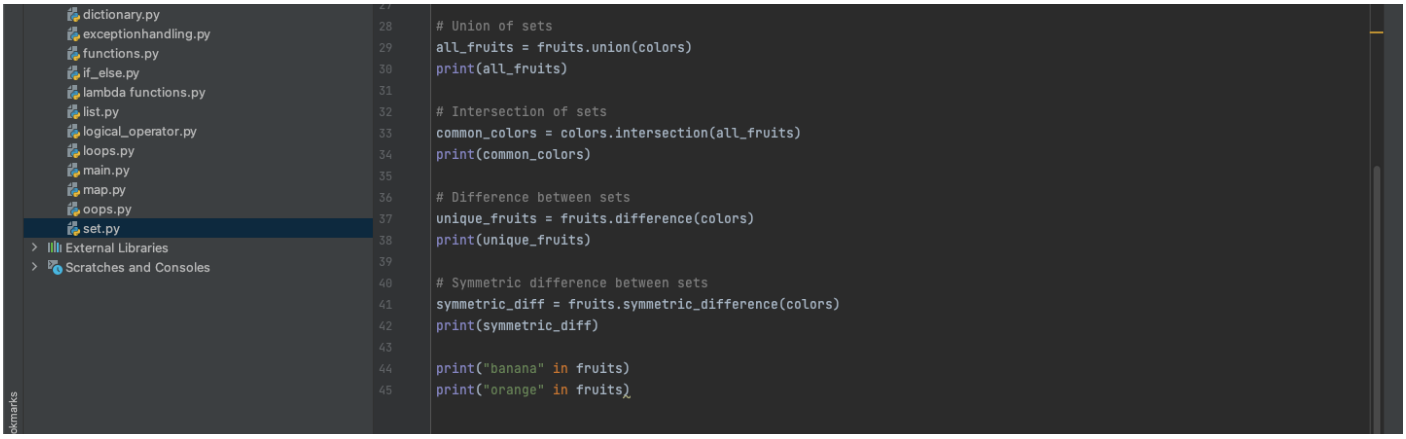


```
/Users/ajaychaudhary/PycharmProjects/Domain_training_python/venv/bin/python /Users/ajaychaudhary/PycharmProjects/Domain_training_python/dictionary.py
Alice
Camry
{'name': 'Alice', 'age': 25, 'grade': 'A', 'city': 'New York'}
{'make': 'Toyota', 'model': 'Camry', 'year': 2023}
{'name': 'Alice', 'grade': 'A', 'city': 'New York'}
A
{'name': 'Alice', 'city': 'New York'}
make model year Alice New York make: Toyota, model: Camry, year: 2023,
Process finished with exit code 0
```

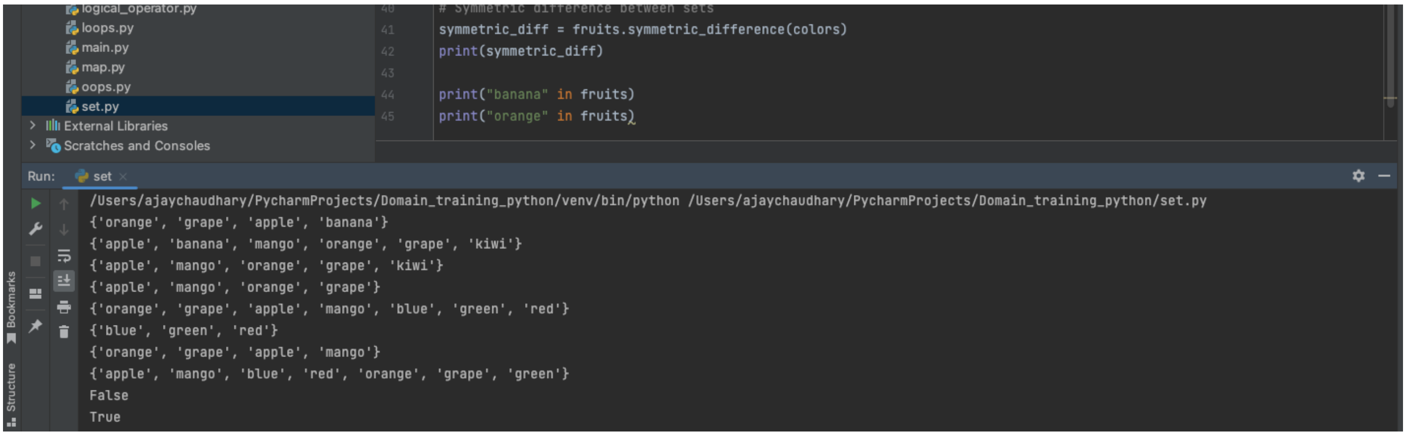
Set and its methods



```python

"""a set is an unordered collection of unique elements.
Sets are useful for various operations like membership testing,
removing duplicates from a sequence,
and performing mathematical operations such as union, intersection, difference, and symmetric difference"""

# Creating a set with literal syntax
fruits = {"apple", "banana", "orange"}

# Creating a set with the set() constructor
colors = set(["red", "green", "blue"])

# Creating an empty set
empty_set = set()
fruits.add("grape")
print(fruits)

# Adding multiple elements
fruits.update(["kiwi", "mango"])
print(fruits)
fruits.remove("banana")
print(fruits)

# Discarding an element (no error if the element is not present)
fruits.discard("kiwi")
print(fruits)

```



```python
# Union of sets
all_fruits = fruits.union(colors)
print(all_fruits)

# Intersection of sets
common_colors = colors.intersection(all_fruits)
print(common_colors)

# Difference between sets
unique_fruits = fruits.difference(colors)
print(unique_fruits)

# Symmetric difference between sets
symmetric_diff = fruits.symmetric_difference(colors)
print(symmetric_diff)

print("banana" in fruits)
print("orange" in fruits)
```



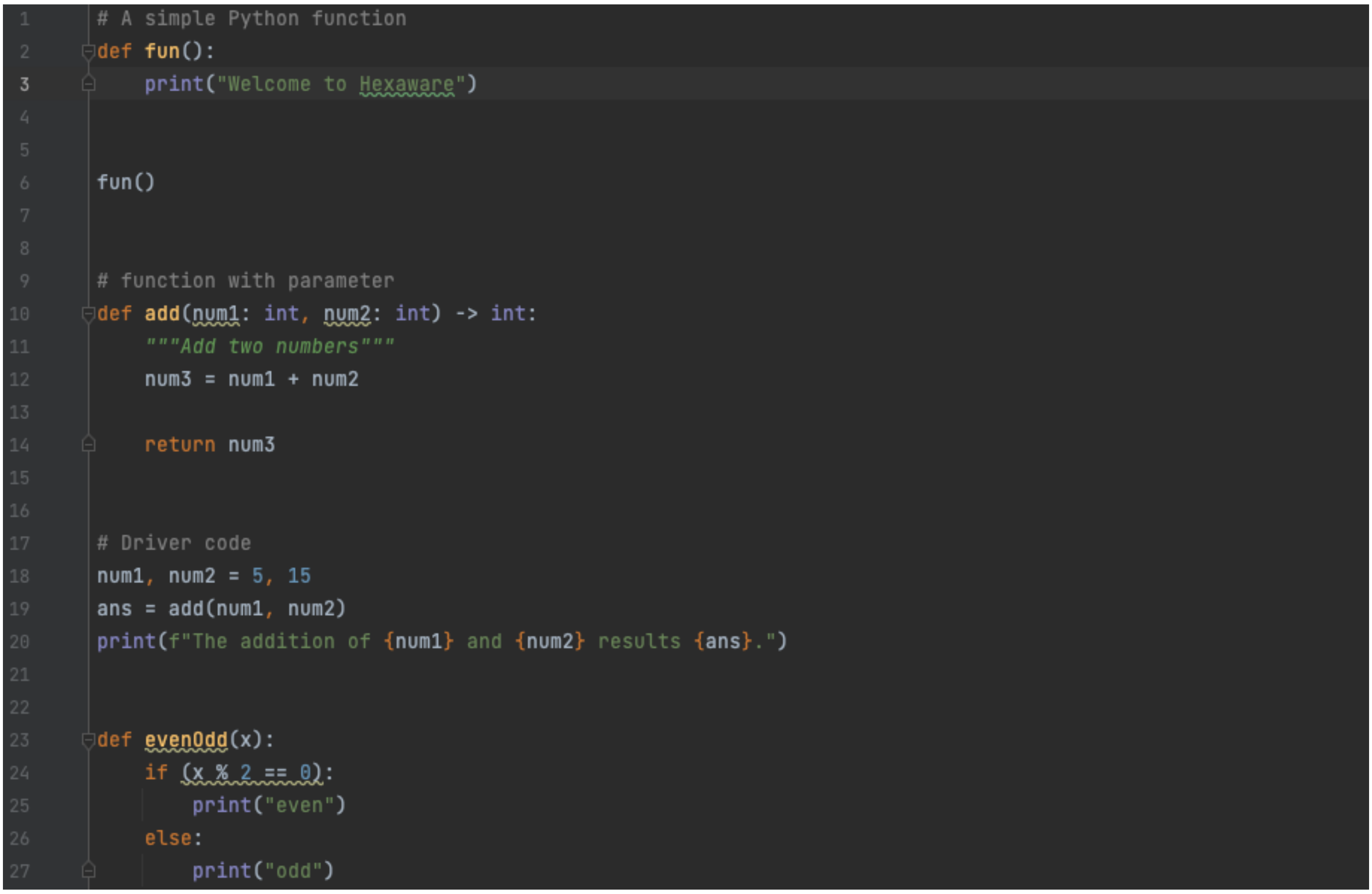
```python
# Symmetric difference between sets
symmetric_diff = fruits.symmetric_difference(colors)
print(symmetric_diff)

print("banana" in fruits)
print("orange" in fruits)
```

```
/Users/ajaychaudhary/PycharmProjects/Domain_training_python/venv/bin/python /Users/ajaychaudhary/PycharmProjects/Domain_training_python/set.py
{'orange', 'grape', 'apple', 'banana'}
{'apple', 'banana', 'mango', 'orange', 'grape', 'kiwi'}
{'apple', 'mango', 'orange', 'grape', 'kiwi'}
{'apple', 'mango', 'orange', 'grape'}
{'orange', 'grape', 'apple', 'mango', 'blue', 'green', 'red'}
{'blue', 'green', 'red'}
{'orange', 'grape', 'apple', 'mango'}
{'apple', 'mango', 'blue', 'red', 'orange', 'grape', 'green'}
False
True
```

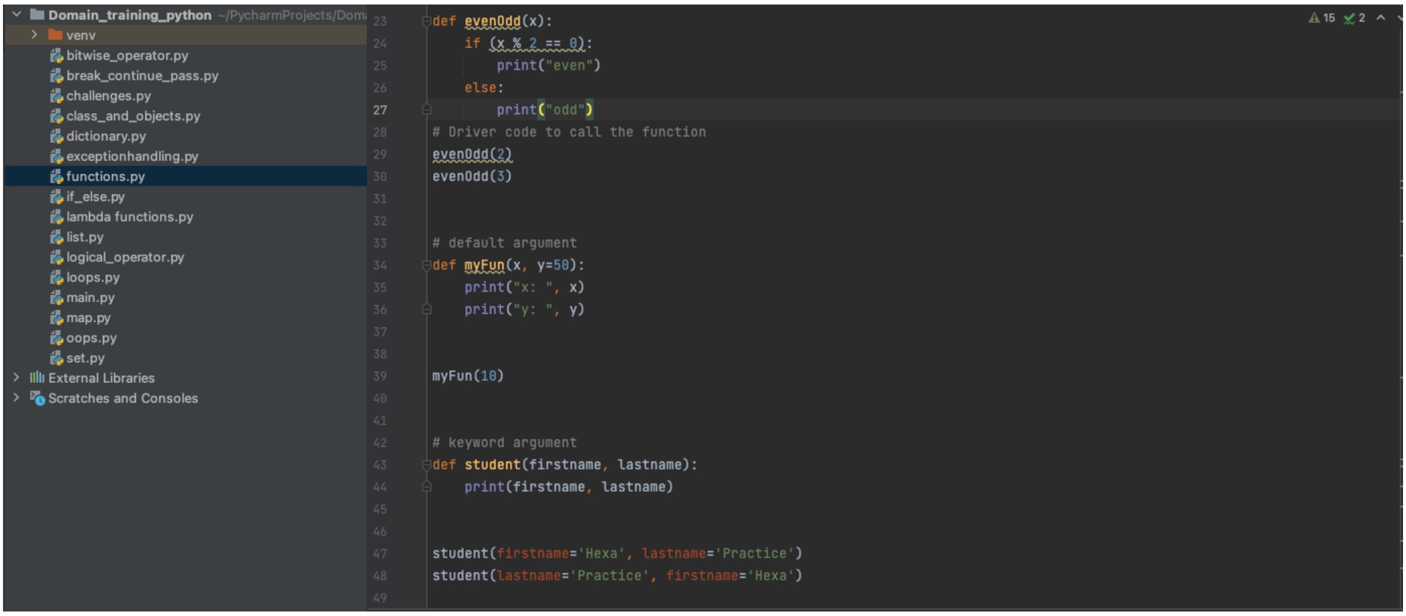
Functions

Function Definition:

- A function is defined using the def keyword, followed by the function name and a pair of parentheses. The function body is indented and contains the code that the function will execute.



```python
# A simple Python function
def fun():
    print("Welcome to Hexaware")


fun()


# function with parameter
def add(num1: int, num2: int) -> int:
    """Add two numbers"""
    num3 = num1 + num2

    return num3


# Driver code
num1, num2 = 5, 15
ans = add(num1, num2)
print(f"The addition of {num1} and {num2} results {ans}.")


def evenOdd(x):
    if (x % 2 == 0):
        print("even")
    else:
        print("odd")
```



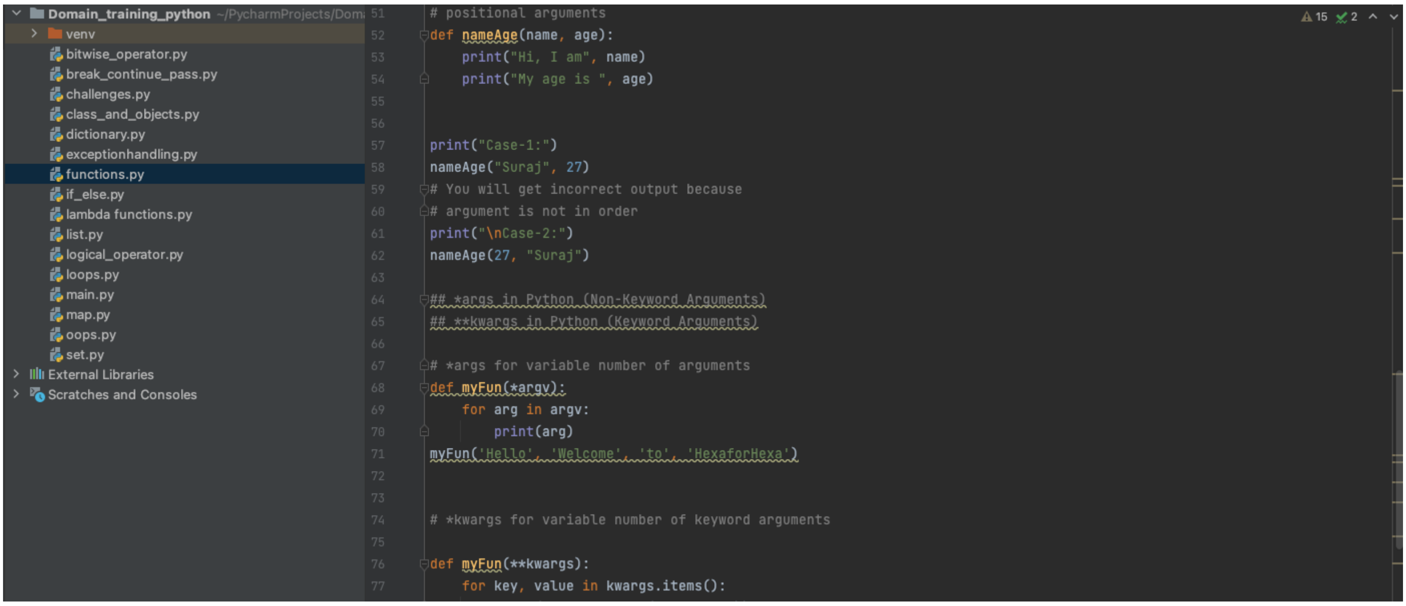
```python
def evenOdd(x):
    if (x % 2 == 0):
        print("even")
    else:
        print("odd")
# Driver code to call the function
evenOdd(2)
evenOdd(3)


# default argument
def myFun(x, y=50):
    print("x: ", x)
    print("y: ", y)


myFun(10)


# keyword argument
def student(firstname, lastname):
    print(firstname, lastname)


student(firstname='Hexa', lastname='Practice')
student(lastname='Practice', firstname='Hexa')
```

1. args:

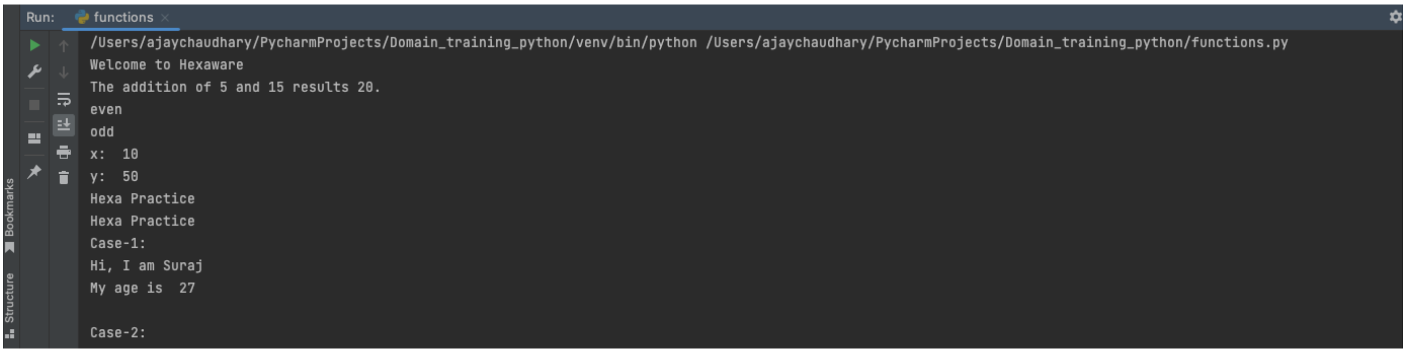
- args is a convention (the name can be different) used to represent a tuple of positional arguments in a function definition. It allows a function to accept a variable number of positional arguments.

2. kwargs:

- kwargs is a convention (the name can be different) used to represent a dictionary of keyword arguments in a function definition. It allows a function to accept a variable number of keyword arguments.



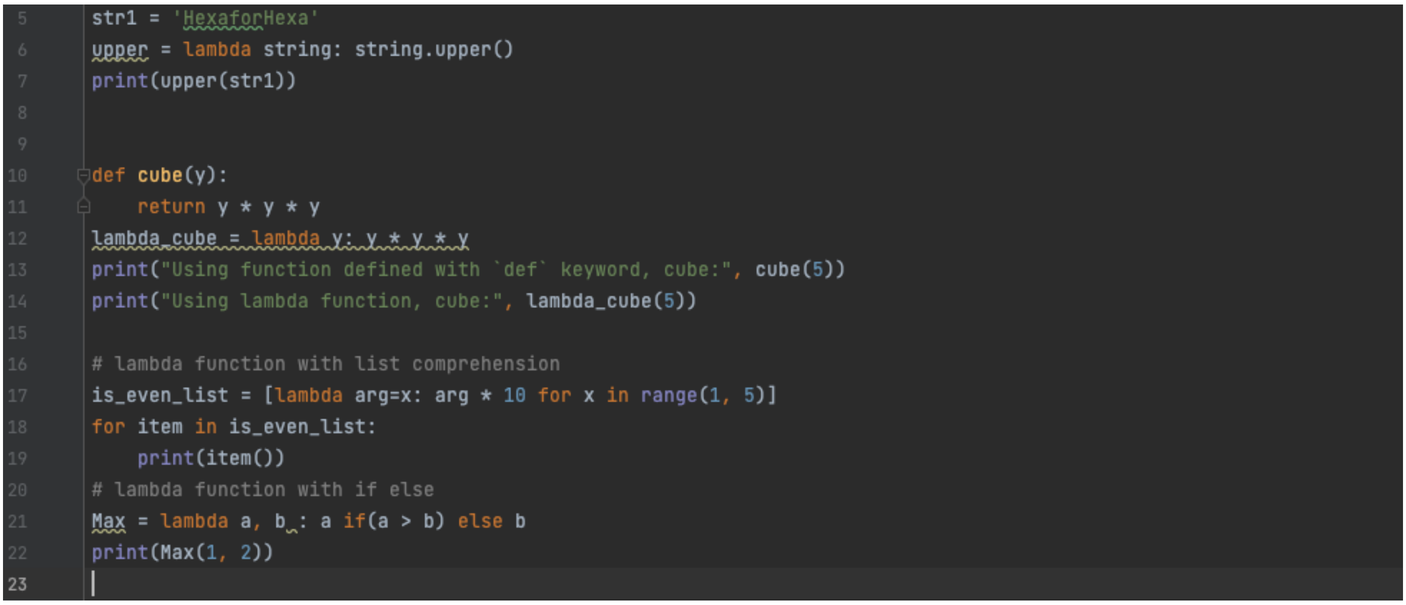
```python
# positional arguments
def nameAge(name, age):
    print("Hi, I am", name)
    print("My age is ", age)


print("Case-1:")
nameAge("Suraj", 27)
# You will get incorrect output because
# argument is not in order
print("\nCase-2:")
nameAge(27, "Suraj")

## *args in Python (Non-Keyword Arguments)
## **kwargs in Python (Keyword Arguments)

# *args for variable number of arguments
def myFun(*argv):
    for arg in argv:
        print(arg)
myFun('Hello', 'Welcome', 'to', 'HexaforHexa')


# *kwargs for variable number of keyword arguments

def myFun(**kwargs):
    for key, value in kwargs.items():
```



```
/Users/ajaychaudhary/PycharmProjects/Domain_training_python/venv/bin/python /Users/ajaychaudhary/PycharmProjects/Domain_training_python/functions.py
Welcome to Hexaware
The addition of 5 and 15 results 20.
even
odd
x:  10
y:  50
Hexa Practice
Hexa Practice
Case-1:
Hi, I am Suraj
My age is  27

Case-2:
```

Lambda function-

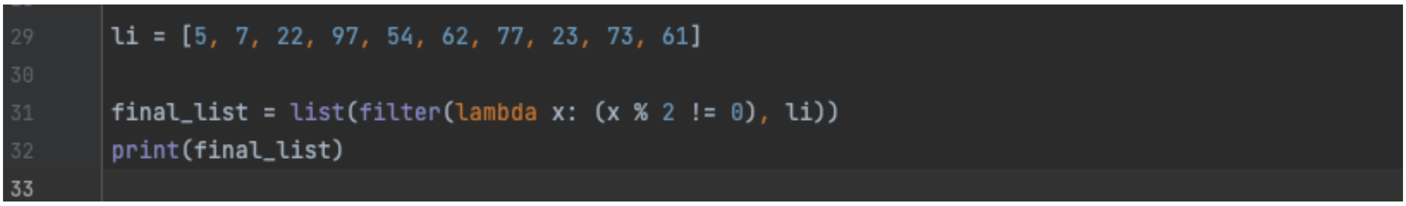
Python Lambda Functions are anonymous functions means that the function is without a name.
This function can have any number of arguments but only one expression, which is evaluated and returned.
One is free to use lambda functions wherever function objects are required.
You need to keep in your knowledge that lambda functions are syntactically restricted to a single expression.



```python
str1 = 'HexaforHexa'
upper = lambda string: string.upper()
print(upper(str1))


def cube(y):
    return y * y * y
lambda_cube = lambda y: y * y * y
print("Using function defined with `def` keyword, cube:", cube(5))
print("Using lambda function, cube:", lambda_cube(5))

# lambda function with list comprehension
is_even_list = [lambda arg=x: arg * 10 for x in range(1, 5)]
for item in is_even_list:
    print(item())
# lambda function with if else
Max = lambda a, b : a if(a > b) else b
print(Max(1, 2))

```

Filter function

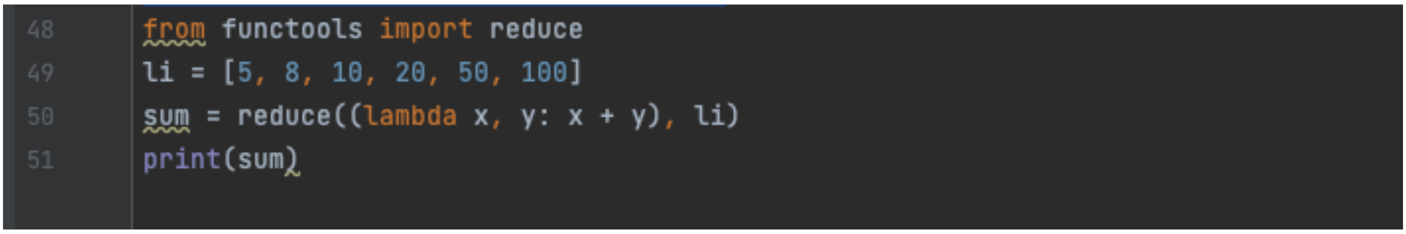
The filter() function in Python takes in a function and a list as arguments.
This offers an elegant way to filter out all the elements of a sequence “sequence”,
for which the function returns True.



```python
li = [5, 7, 22, 97, 54, 62, 77, 23, 73, 61]

final_list = list(filter(lambda x: (x % 2 != 0), li))
print(final_list)

```

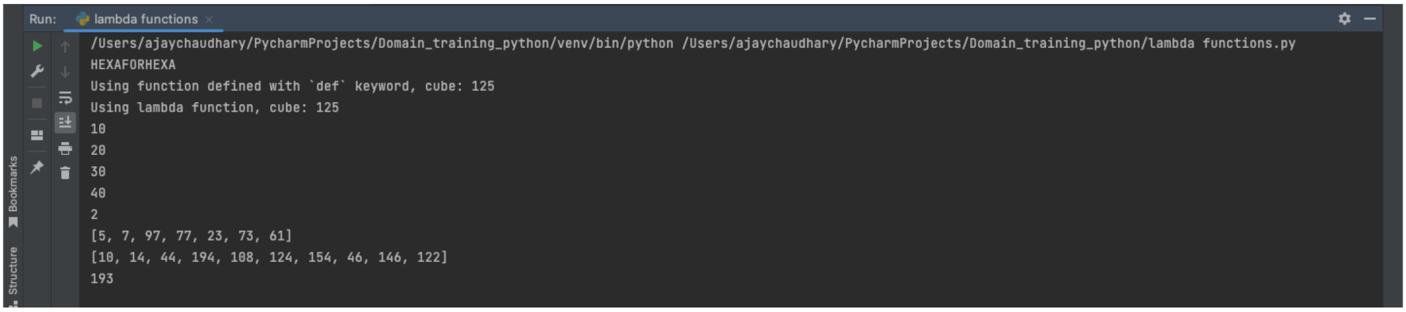
Reduce function

The reduce() function in Python takes in a function and a list as an argument.
The function is called with a lambda function and an iterable
and a new reduced result is returned.



```python
from functools import reduce
li = [5, 8, 10, 20, 50, 100]
sum = reduce((lambda x, y: x + y), li)
print(sum)
```

Outputs:

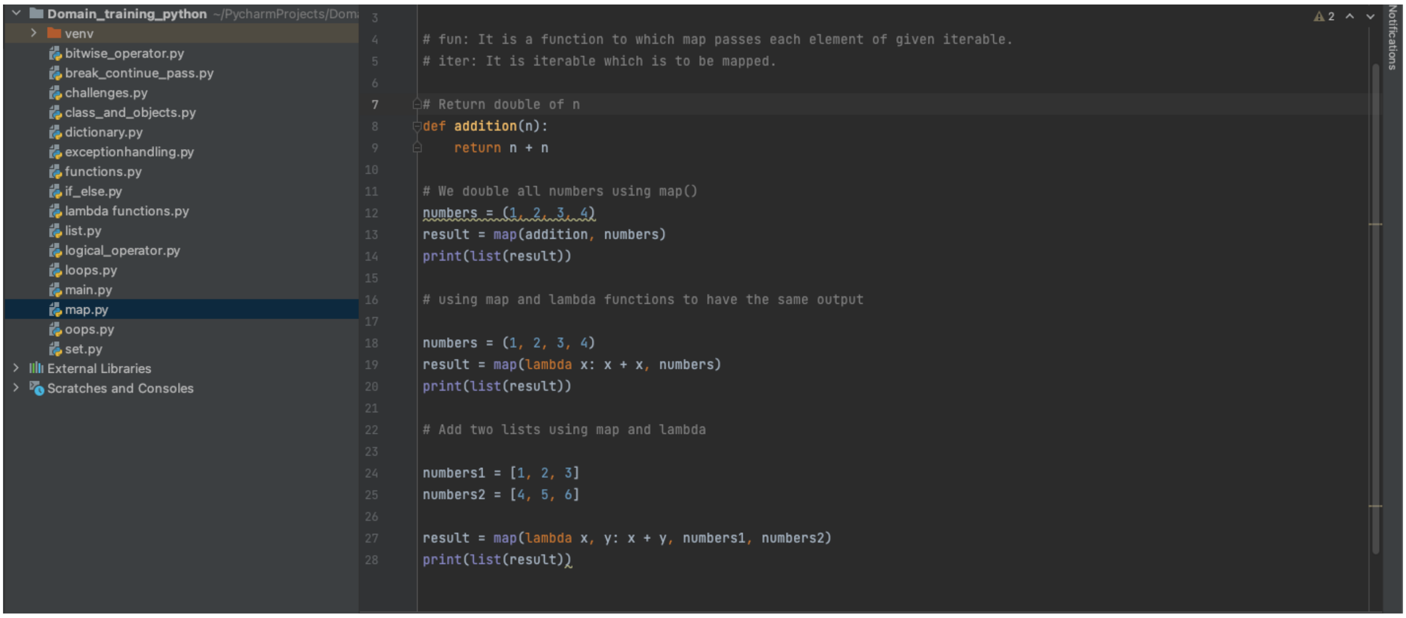


```
/Users/ajaychaudhary/PycharmProjects/Domain_training_python/venv/bin/python /Users/ajaychaudhary/PycharmProjects/Domain_training_python/lambda functions.py
HEXAFORHEXA
Using function defined with `def` keyword, cube: 125
Using lambda function, cube: 125
10
20
30
40
2
[5, 7, 97, 77, 23, 73, 61]
[10, 14, 44, 194, 108, 124, 154, 46, 146, 122]
193
```

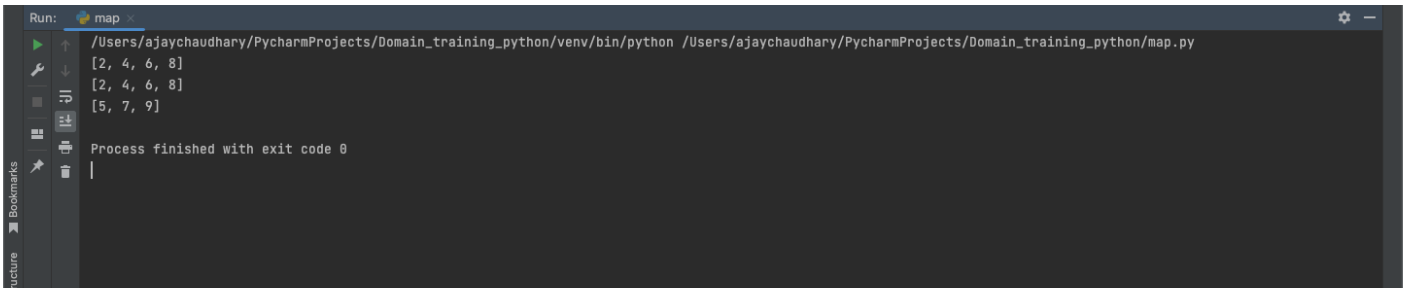
Map function

- function: This is the function that you want to apply to each item in the iterable. It could be a built-in function or a custom-defined function.
- iterable: This is the iterable (e.g., list, tuple, set) whose elements will be processed by the function.

The map() function returns a map object (an iterator). To get the result as a list, you can use the list() function to convert it.



```python

# fun: It is a function to which map passes each element of given iterable.
# iter: It is iterable which is to be mapped.

# Return double of n
def addition(n):
    return n + n

# We double all numbers using map()
numbers = (1, 2, 3, 4)
result = map(addition, numbers)
print(list(result))

# using map and lambda functions to have the same output

numbers = (1, 2, 3, 4)
result = map(lambda x: x + x, numbers)
print(list(result))

# Add two lists using map and lambda

numbers1 = [1, 2, 3]
numbers2 = [4, 5, 6]

result = map(lambda x, y: x + y, numbers1, numbers2)
print(list(result))
```



```
/Users/ajaychaudhary/PycharmProjects/Domain_training_python/venv/bin/python /Users/ajaychaudhary/PycharmProjects/Domain_training_python/map.py
[2, 4, 6, 8]
[2, 4, 6, 8]
[5, 7, 9]

Process finished with exit code 0
```

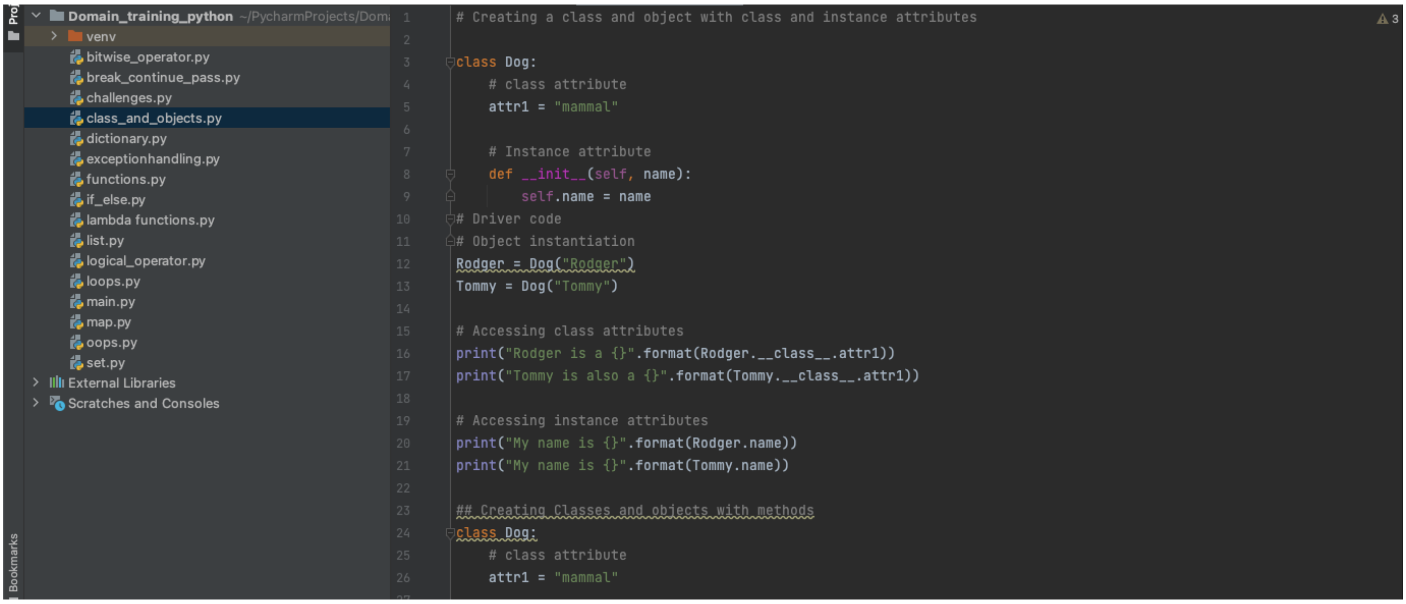
Class and Objects-

Class:

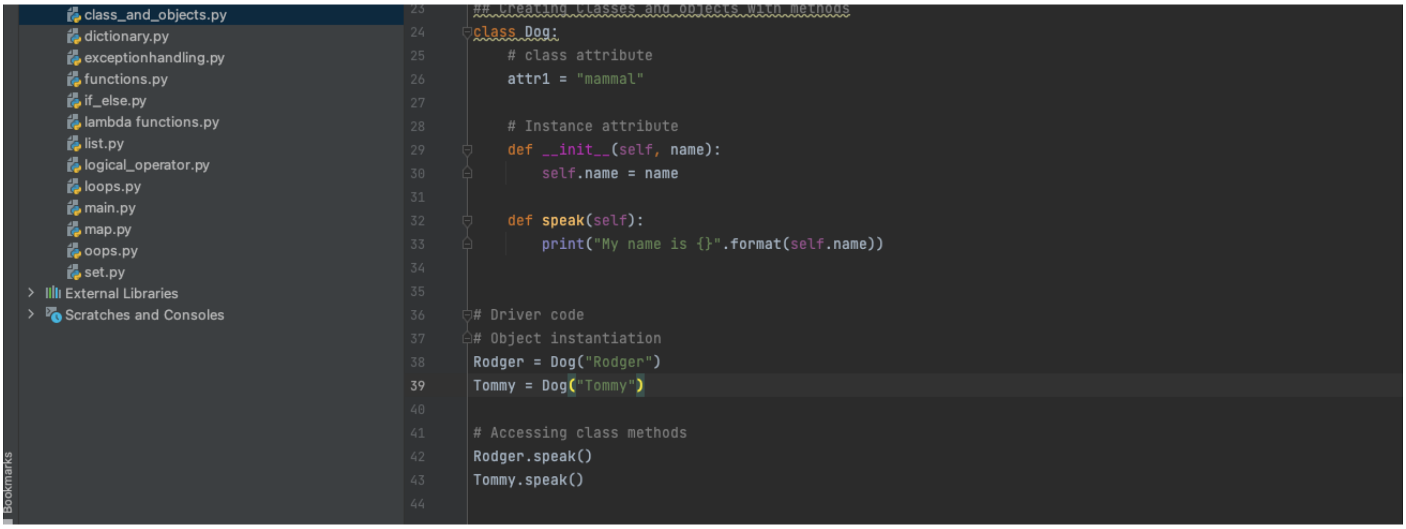
- A class is a user-defined data type in object-oriented programming.
- It serves as a blueprint for creating objects.
- It defines a set of attributes (properties) and methods (functions) that characterize the objects created from the class.
- The attributes are variables that store data, and the methods are functions that perform actions related to the class.

Object:

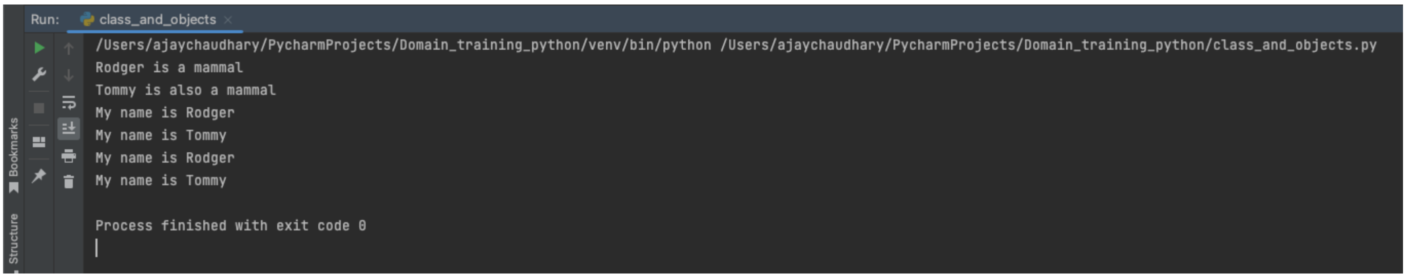
- An object is an instance of a class, created from the class blueprint.
- Objects have their own unique set of attributes, which are defined by the class, and can have their own state.
- Objects can perform actions or operations through the methods defined in the class



```
# Creating a class and object with class and instance attributes

class Dog:
    # class attribute
    attr1 = "mammal"

    # Instance attribute
    def __init__(self, name):
        self.name = name
# Driver code
# Object instantiation
Rodger = Dog("Rodger")
Tommy = Dog("Tommy")

# Accessing class attributes
print("Rodger is a {}".format(Rodger.__class__.attr1))
print("Tommy is also a {}".format(Tommy.__class__.attr1))

# Accessing instance attributes
print("My name is {}".format(Rodger.name))
print("My name is {}".format(Tommy.name))

## Creating Classes and objects with methods
class Dog:
    # class attribute
    attr1 = "mammal"
```



```
## Creating Classes and objects with methods
class Dog:
    # class attribute
    attr1 = "mammal"

    # Instance attribute
    def __init__(self, name):
        self.name = name

    def speak(self):
        print("My name is {}".format(self.name))


# Driver code
# Object instantiation
Rodger = Dog("Rodger")
Tommy = Dog("Tommy")

# Accessing class methods
Rodger.speak()
Tommy.speak()
```

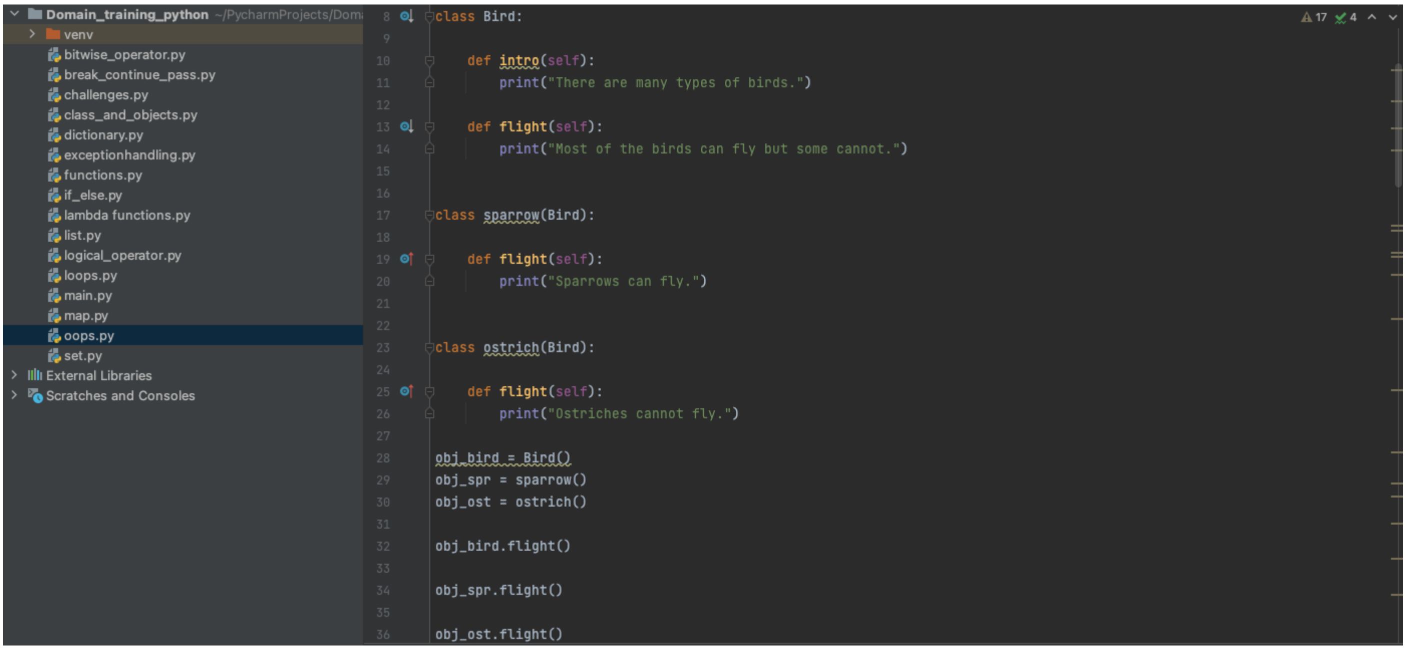


```
/Users/ajaychaudhary/PycharmProjects/Domain_training_python/venv/bin/python /Users/ajaychaudhary/PycharmProjects/Domain_training_python/class_and_objects.py
Rodger is a mammal
Tommy is also a mammal
My name is Rodger
My name is Tommy
My name is Rodger
My name is Tommy

Process finished with exit code 0
```

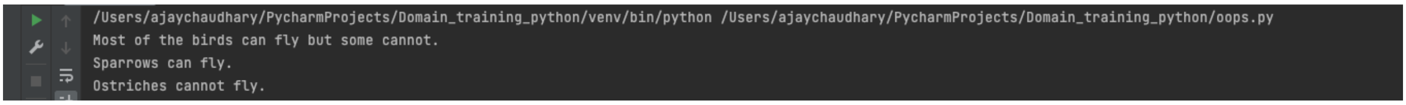
Object oriented programming concepts-

Polymorphism-

Polymorphism simply means having many forms.
For example, we need to determine if the given species of birds fly or not, using polymorphism we can do this using a single function.



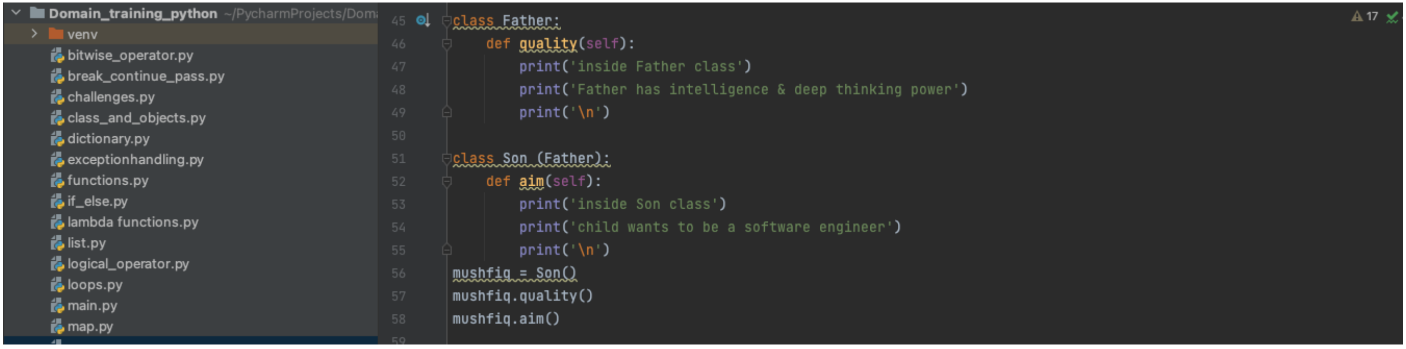
```
class Bird:

    def intro(self):
        print("There are many types of birds.")

    def flight(self):
        print("Most of the birds can fly but some cannot.")


class sparrow(Bird):

    def flight(self):
        print("Sparrows can fly.")


class ostrich(Bird):

    def flight(self):
        print("Ostriches cannot fly.")

obj_bird = Bird()
obj_spr = sparrow()
obj_ost = ostrich()

obj_bird.flight()

obj_spr.flight()

obj_ost.flight()
```



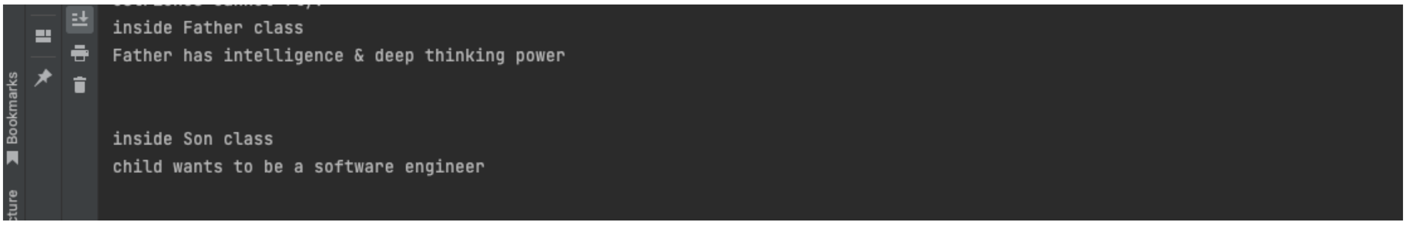
```
/Users/ajaychaudhary/PycharmProjects/Domain_training_python/venv/bin/python /Users/ajaychaudhary/PycharmProjects/Domain_training_python/oops.py
Most of the birds can fly but some cannot.
Sparrows can fly.
Ostriches cannot fly.
```

Inheritance-

It is permitted in python to build a class derived from one or more other classes. This class is referred to as child class or subclass. The attributes, methods, and other members of parent class are inherited by the child class. The methods of parent class are overridden in child class.



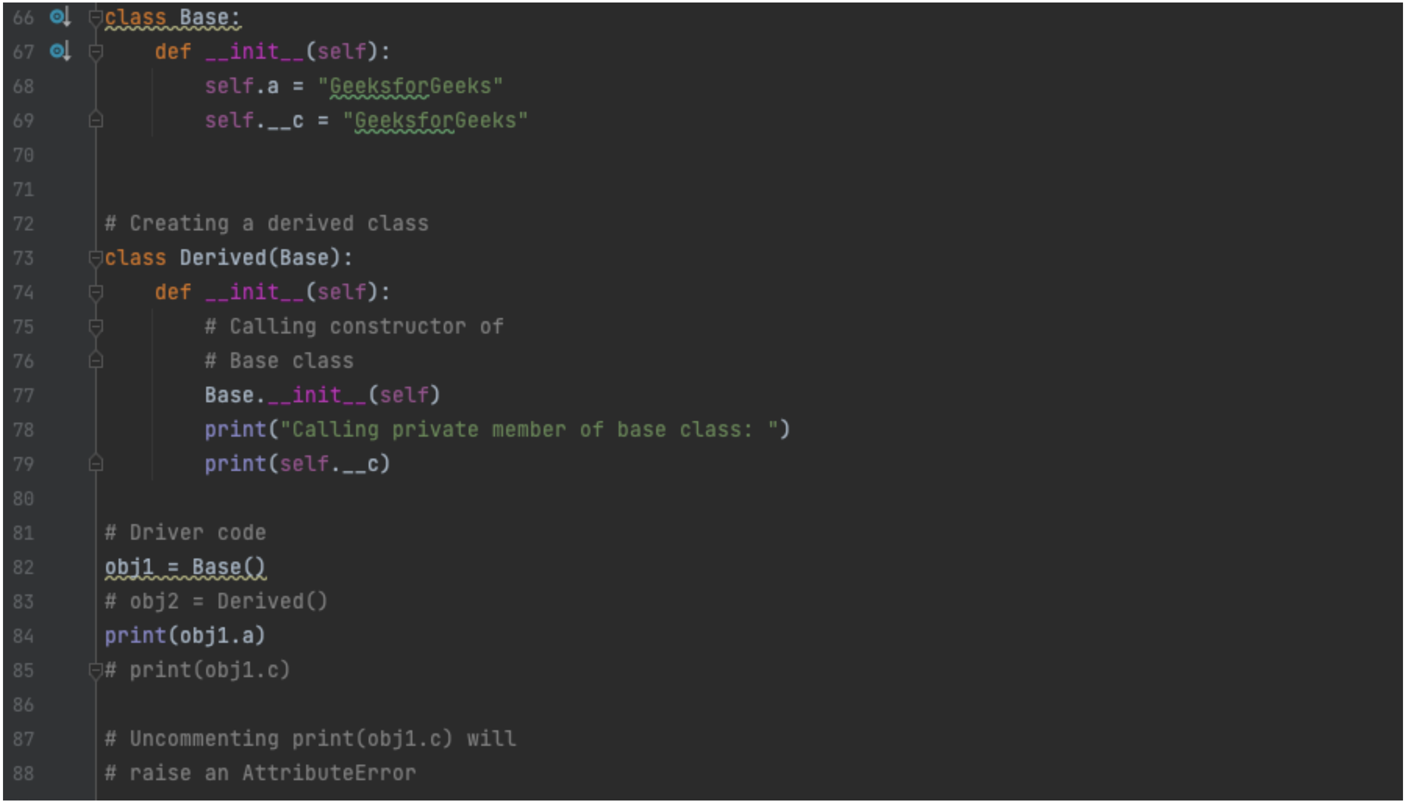
```
class Father:
    def quality(self):
        print('inside Father class')
        print('Father has intelligence & deep thinking power')
        print('\n')

class Son (Father):
    def aim(self):
        print('inside Son class')
        print('child wants to be a software engineer')
        print('\n')
mushfiq = Son()
mushfiq.quality()
mushfiq.aim()
```



```
inside Father class
Father has intelligence & deep thinking power


inside Son class
child wants to be a software engineer
```

Encapsulation-

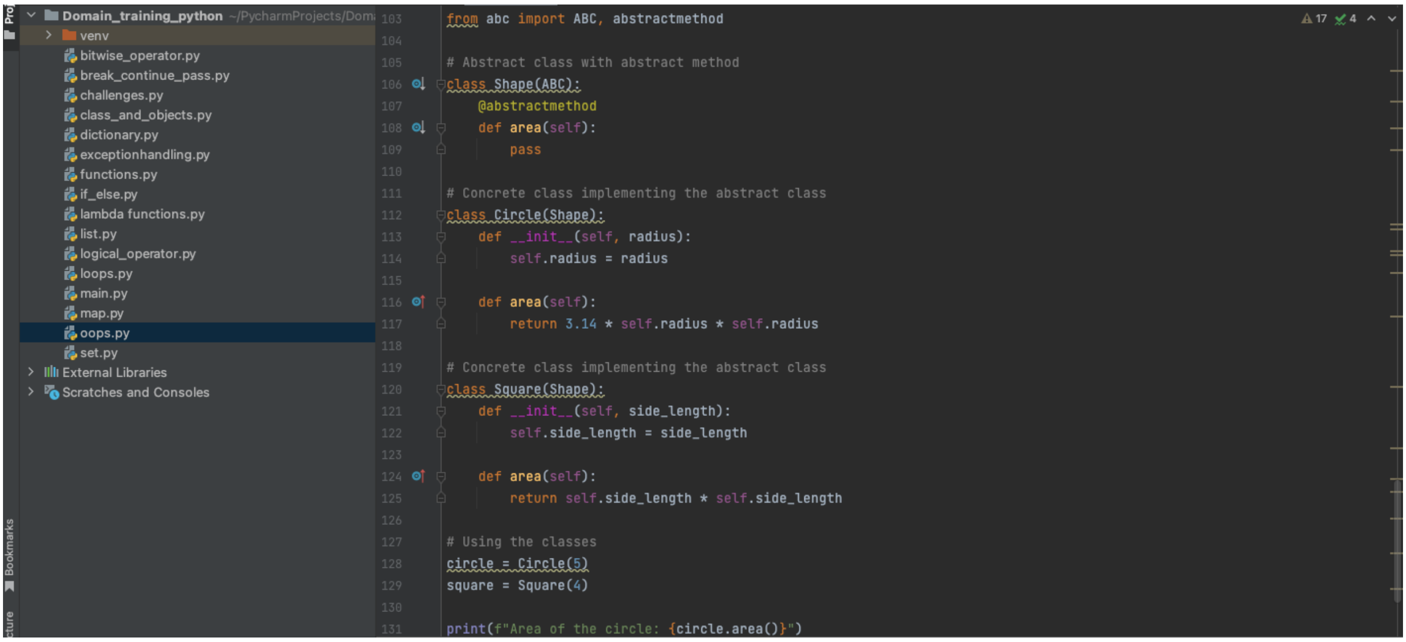
It describes the idea of wrapping data and the methods that work on data within one unit.
This puts restrictions on accessing variables and methods directly
and can prevent the accidental modification of data



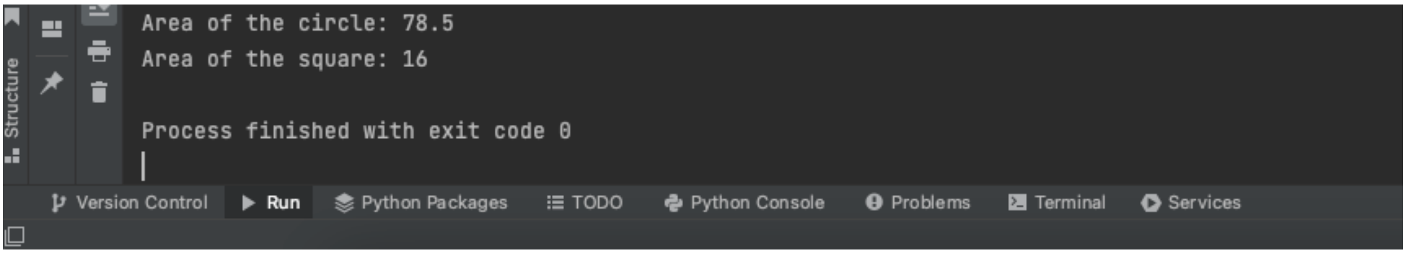
```python
class Base:
    def __init__(self):
        self.a = "GeeksforGeeks"
        self.__c = "GeeksforGeeks"


# Creating a derived class
class Derived(Base):
    def __init__(self):
        # Calling constructor of
        # Base class
        Base.__init__(self)
        print("Calling private member of base class: ")
        print(self.__c)

# Driver code
obj1 = Base()
# obj2 = Derived()
print(obj1.a)
# print(obj1.c)

# Uncommenting print(obj1.c) will
# raise an AttributeError
```

Abstraction-

It hides unnecessary code details from the user.
Also, when we do not want to give out sensitive parts of our code implementation
and this is where data abstraction came.
Data Abstraction in Python can be achieved by creating abstract classes.

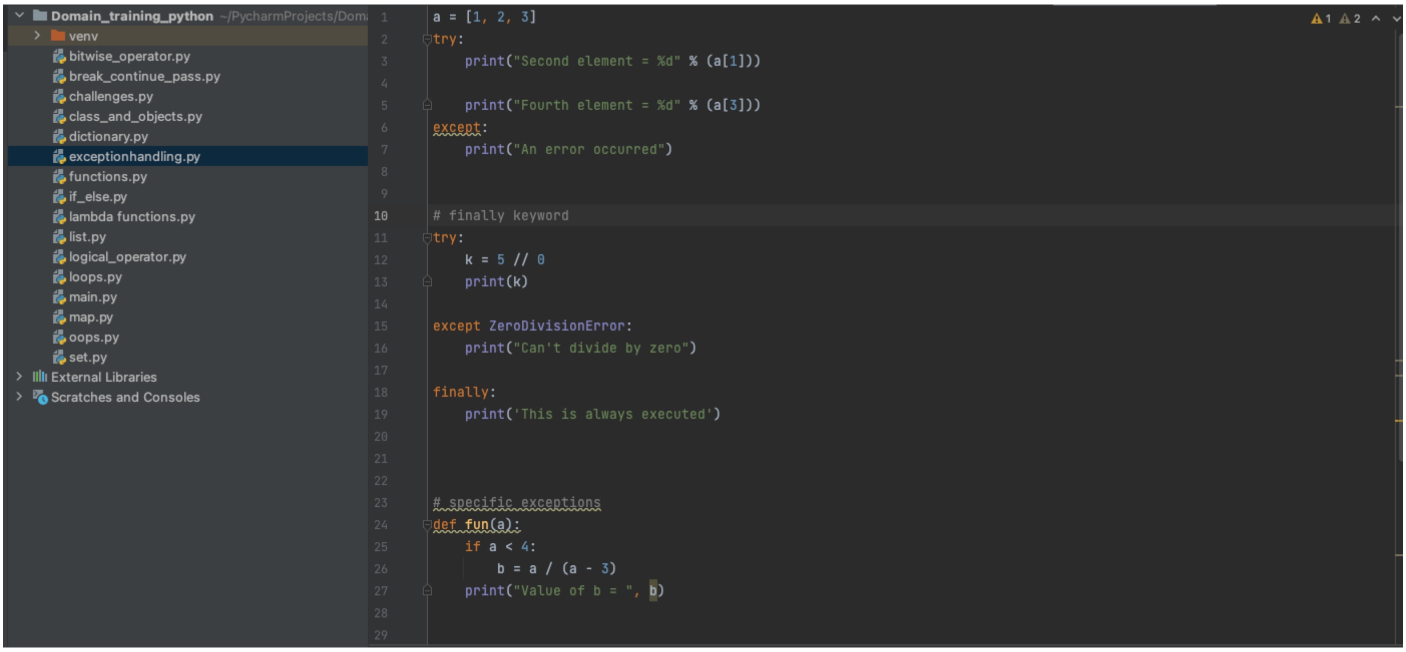


```python
from abc import ABC, abstractmethod

# Abstract class with abstract method
class Shape(ABC):
    @abstractmethod
    def area(self):
        pass

# Concrete class implementing the abstract class
class Circle(Shape):
    def __init__(self, radius):
        self.radius = radius

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

# Concrete class implementing the abstract class
class Square(Shape):
    def __init__(self, side_length):
        self.side_length = side_length

    def area(self):
        return self.side_length * self.side_length

# Using the classes
circle = Circle(5)
square = Square(4)

print(f"Area of the circle: {circle.area()}")
```

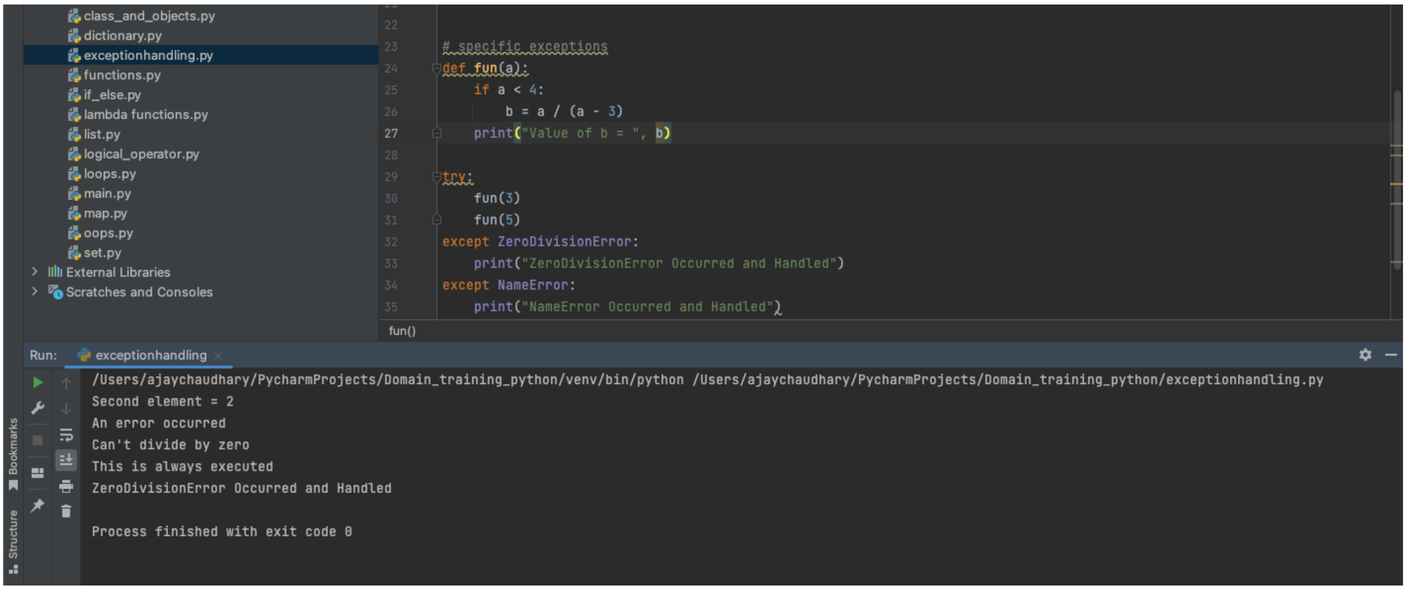


```
Area of the circle: 78.5
Area of the square: 16

Process finished with exit code 0
```

Exception handling



```python
a = [1, 2, 3]
try:
    print("Second element = %d" % (a[1]))

    print("Fourth element = %d" % (a[3]))
except:
    print("An error occurred")


# finally keyword
try:
    k = 5 // 0
    print(k)

except ZeroDivisionError:
    print("Can't divide by zero")

finally:
    print('This is always executed')



# specific exceptions
def fun(a):
    if a < 4:
        b = a / (a - 3)
    print("Value of b = ", b)
```



```python

# specific exceptions
def fun(a):
    if a < 4:
        b = a / (a - 3)
    print("Value of b = ", b)

try:
    fun(3)
    fun(5)
except ZeroDivisionError:
    print("ZeroDivisionError Occurred and Handled")
except NameError:
    print("NameError Occurred and Handled")
```

```
/Users/ajaychaudhary/PycharmProjects/Domain_training_python/venv/bin/python /Users/ajaychaudhary/PycharmProjects/Domain_training_python/exceptionhandling.py
Second element = 2
An error occurred
Can't divide by zero
This is always executed
ZeroDivisionError Occurred and Handled

Process finished with exit code 0
```