Muhammad Waleed
20b-115-se (SE-B)
AI Lab#02
Sir Nasir Ud Deen

## Lab Tasks:

```
def printme(str):
    " This prints a passed string into this function "
    print(str)
    return


printme(" This is first call to the user defined function !")
printme(" Again second call to the same function ")
```

[69] ✓ 0.0s

```
This is first call to the user defined function !
Again second call to the same function
```

```
# Function definition is here
def changeme(mylist):
    " This changes a passed list into this function "
    print(" Values inside the function before change : ", mylist)
    mylist[2] = 50
    print(" Values inside the function after change : ", mylist)
    return


8  # Now you can call changeme function
mylist = [10, 20, 30]
changeme(mylist)
print(" Values outside the function : ", mylist)
```

[70] ✓ 0.0s

```
Values inside the function before change :  [10, 20, 30]
Values inside the function after change :  [10, 20, 50]
Values outside the function :  [10, 20, 50]
```

Muhammad Waleed
20b-115-se (SE-B)
AI Lab#02
Sir Nasir Ud Deen

```
# Function definition is here
def changeme(mylist):
    " This changes a passed list into this function "
    mylist = [1, 2, 3, 4]  # This would assign new reference in mylist
    print(" Values inside the function : ", mylist)
    return


# Now you can call changeme function
mylist = [10, 20, 30]
changeme(mylist)
print(" Values outside the function : ", mylist)
```

[71] ✓ 0.0s

```
Values inside the function :  [1, 2, 3, 4]
Values outside the function :  [10, 20, 30]
```

```
# Function definition is here
def sum(arg1, arg2):
    # Add both the parameters and return them ."
    total = arg1 + arg2
    print(" Inside the function : ", total)
    return total


# Now you can call sum function
total = sum(10, 20)
print(" Outside the function : ", total)
```

[72] ✓ 0.0s

```
Inside the function :  30
Outside the function :  30
```

Muhammad Waleed
20b-115-se (SE-B)
AI Lab#02
Sir Nasir Ud Deen

```
class Dog:
    # Class Attribute
    species = 'mammal'
# Initializer / Instance Attributes

    def __init__(self, name, age):
        self.name = name
        self.age = age
# instance method

    def description(self):
        return "{} is {} years old ". format(self . name, self . age)
        # instance method

    def speak(self, sound):
        return "{} says {}". format(self . name, sound)

    # Instantiate the Dog object
razer = Dog(" Razer ", 6)
# call our instance methods
print(razer . description())
print(razer . speak(" Woof Woof "))
```

[73] ✓ 0.0s

```
Razer  is 6 years old
Razer  says  Woof Woof
```

Muhammad Waleed
20b-115-se (SE-B)
AI Lab#02
Sir Nasir Ud Deen

```python
# Parent class
class Dog:

    # Class attribute
    species = 'mammal'
    # Initializer / Instance attributes

    def __init__(self, name, age):

        self . name = name
        self . age = age
    # instance method

    def description(self):
        return "{} is {} years old ". format(self . name, self . age)
        # instance method

    def speak(self, sound):
        return "{} says {}". format(self . name, sound)
    # Child class ( inherits from Dog class )


class RussellTerrier (Dog):
    def run(self, speed):
        return "{} runs {}". format(self . name, speed)
    # Child class ( inherits from Dog class )


class Bulldog (Dog):
    def run(self, speed):
        return "{} runs {}". format(self . name, speed)


# Child classes inherit attributes and
# behaviors from the parent class
thunder = Bulldog(" Thunder ", 9)
print(thunder . description())
# Child classes have specific attributes
# and behaviors as well
print(thunder . run(" slowly "))
spinter = Bulldog(" Spinter ", 12)
print(spinter . description())
print(spinter . run(" fast "))
roger = RussellTerrier(" Roger ", 5)
```

[74] ✓ 0.0s

```
Thunder  is 9 years old
Thunder  runs  slowly
Spinter  is 12 years old
Spinter  runs  fast
```

Muhammad Waleed
20b-115-se (SE-B)
AI Lab#02
Sir Nasir Ud Deen

```python
# Parent class
class Dog :
    # Class attribute
    species = 'mammal'
# Initializer / Instance attributes
    def __init__ ( self , name , age ) :
        self . name = name
        self . age = age
# instance method
    def description ( self ) :
        return "{} is {} years old ". format ( self . name , self . age )
# instance method
    def speak ( self , sound ) :
        return "{} says {}". format ( self . name , sound )
 # Child class ( inherits from Dog () class )
class RussellTerrier ( Dog ) :
    def run ( self , speed ) :
        return "{} runs {}". format ( self . name , speed )
# Child class ( inherits from Dog () class )
class Bulldog ( Dog ) :
    def run ( self , speed ) :
        return "{} runs {}". format ( self . name , speed )
 # Child classes inherit attributes and
 # behaviors from the parent class
thunder = Bulldog (" Thunder ", 9)
print ( thunder . description () )
# Child classes have specific attributes
# and behaviors as well
print ( thunder . run (" slowly ") )
# Is thunder an instance of Dog ()?
print ( isinstance ( thunder , Dog ) )
# Is thunder_kid an instance of Dog ()?
thunder_kid = Dog (" ThunderKid ", 2)
print ( isinstance ( thunder , Dog ) )
# Is Kate an instance of Bulldog ()
Kate = RussellTerrier (" Kate ", 4)
print ( isinstance ( Kate , Dog ) )
print ( isinstance ( thunder_kid , Kate ) )

print (" Thanks for understanding the concept of OOPs ")
```

[75] ⊗ 0.0s

```
 Thunder  is 9 years old
 Thunder  runs  slowly
True
True
True
```

Muhammad Waleed
20b-115-se (SE-B)
AI Lab#02
Sir Nasir Ud Deen

```python
import math

class ComplexAPI:
    def __init__(self, real, imag):
        self.real = real
        self.imag = imag

    def theta(self):
        return math.atan2(self.imag, self.real)

    def conjugate(self):
        return ComplexAPI(self.real, -self.imag)

    def __sub__(self, other):
        return ComplexAPI(self.real - other.real, self.imag - other.imag)

    def __truediv__(self, other):
        if isinstance(other, ComplexAPI):
            conj = other.conjugate()
            numerator = self * conj
            denominator = other * conj
            return ComplexAPI(numerator.real / denominator.real, numerator.imag / denominator.real)
        elif isinstance(other, complex):
            conj = ComplexAPI(other.real, -other.imag)
            numerator = self * conj
            denominator = other * conj
            return ComplexAPI(numerator.real / denominator.real, numerator.imag / denominator.real)
        else:
            raise TypeError(f"unsupported operand type(s) for /: 'ComplexAPI' and '{type(other)}'")

    def __mul__(self, other):
        if isinstance(other, ComplexAPI):
            real = self.real * other.real - self.imag * other.imag
            imag = self.real * other.imag + self.imag * other.real
            return ComplexAPI(real, imag)
        elif isinstance(other, complex):
            real = self.real * other.real - self.imag * other.imag
            imag = self.real * other.imag + self.imag * other.real
            return complex(real, imag)
        else:
            raise TypeError(f"unsupported operand type(s) for *: 'ComplexAPI' and '{type(other)}'")

    def __pow__(self, power):
        r = math.hypot(self.real, self.imag)
        theta = self.theta()
        new_r = r ** power
        new_theta = theta * power
        real = new_r * math.cos(new_theta)
        imag = new_r * math.sin(new_theta)
        return ComplexAPI(real, imag)


a = ComplexAPI(1, 2)
b = ComplexAPI(3, 4)
c = 2 + 3j

print(a - b)
print(a / b)
```

```
-2 + -2j
0.44 + 0.08j
```

Muhammad Waleed
20b-115-se (SE-B)
AI Lab#02
Sir Nasir Ud Deen

## Home Task:

```python
class Polynomial:
    def __init__(self, *args):
        self.coefficients = args
    def __str__(self):
        return ' + '.join([str(c) + 'x^' + str(i) for i, c in enumerate(self.coefficients)])
    def __add__(self, other):
        return Polynomial(*[c1 + c2 for c1, c2 in zip(self.coefficients, other.coefficients)])
    def __sub__(self, other):
        return Polynomial(*[c1 - c2 for c1, c2 in zip(self.coefficients, other.coefficients)])
    def __mul__(self, other):
        return Polynomial(*[c1 * c2 for c1, c2 in zip(self.coefficients, other.coefficients)])
    def __eq__(self, other):
        return self.coefficients == other.coefficients
    def degree(self):
        return len(self.coefficients) - 1
    def evaluate(self, x):
        return sum([c * x ** i for i, c in enumerate(self.coefficients)])

p1 = Polynomial(1, 2, 3)
p2 = Polynomial(3, 2, 1)

print(p1)

print(p1 + p2)

print(p1 - p2)

print(p1 * p2)

print(p1 == p2)

print(p1.degree())

print(p1.evaluate(2))
```

[1] ✓ 0.0s

```
1x^0 + 2x^1 + 3x^2
4x^0 + 4x^1 + 4x^2
-2x^0 + 0x^1 + 2x^2
3x^0 + 4x^1 + 3x^2
False
2
17
```