Atharva Yadav S23-127

Written Assignment 2

PAGE No. / DATE / /

Q] Exception handling in Python

Exception handling in python is managed through the use of try, except, else and finally blocks. These blocks enable developers to anticipate and manage errors that may occur during the execution of a program, allowing for more robust and error resistant code.

* try block: code that might cause an exception is placed inside a try block.

except block: If an exception occurs in the try block, the flow of exception moves to an except block where the exception can be handled. Multiple except blocks can cater different types of exception.

else block: If no exceptions occur within the try block, the else block is executed.

finally block: code within the finally block executes regardless of whether an exception occurred, making it ideal for cleaning up resources or executing code that must run no matter what.

Example:

Output: You can't divide by zero!

```
python
try:
    result = 10/0
except ZeroDivisionError:
    print("You can't divide by zero!")
else:
    print("Division successfull")
finally:
    Print("Execution complete")
```

PAGE No.
DATE / /

6] Polymorphism in Python:

Polymorphism in programming refers to the ability of different objects to respond, in their own way, to the same method call. In Python, polymorphism can be demonstrated in several ways:

Method overriding: When a subclass provides a specific implementation for a method that is already defined in its superclass.

Duck typing: Python's approach to polymorphism where the class of an object is less important than the methods/attributes the objects has. If an object can "quack" like a duck, Python allows treating it as a duck.

Example of Method ~~Ov~~ overriding:

```
python.
class Bird:
    def fly(self):
        print("Some birds can fly.")
class Parrot(Bird):
    bird.fly()

test_bird(Bird())
test_bird(Parrot())
```

Output:

Some birds can fly

parrots can fly

PAGE No.
DATE / /

Lambda function in Python:-

Lambda function in Python are small, anonymous functions defined by the keyword lambda Lambda functions can have any number of arguments but only one expression. The expression is evaluated and returned. Lambda functions are often used when a simple function is passed as an argument to higher order functions:

Example:

```
Python
multiply = lamda x, y: x * y
print(multiply(5,6)) # output: 30
```

Output : 30

Q] Multiple inheritance in python.

Multiple inheritance is supported in python. allowing a class to inherit from more than one parent class. This feature enables the child class to access attributes and methods of all the parent classes.

Example:

```
class Father:
    def gardening (self):
        print("I Enjoy gardening").
class Mother:
    def cooking (self):
        print ("I enjoy cooking")
```

PAGE No.
DATE / /

```
class child(father, mother):
    def sports(self):
        print("I enjoy sports").

c = child()                    Output: I enjoy gardening
c.gardening()                          I Love cooking.
c.cooking()
```

8] Creating Arrays in Python.

Arrays in python can be created using the array module for 1D arrays or the numpy library for 1D, 2D and 3D arrays, offering more functionality and efficiency for large datasets.

1D Array:

```
python
import numpy as np
array_1d = np.array([1, 2, 3])
```

2D Array:

```
Python
array_2d = np.array([[1,2,3], [4, 5, 6]])
```

3D Array:

```
python.
array_3d = np.array([[[1, 2],[3,3]], [[5, 6], [7,8]]])
```

These steps and concepts illustrate basic to advanced functionalities in python, demonstrating its versatility from handling errors gracefully to supporting complex inheritance

patterns and accommodating various data structures.

operating overloading in Python allows you to define how operators behave for user defined objects. ~~This is~~ This is achieved by defining special methods in your class that are called when certain operators are used on instances of the class. Here is an example of operator overloading in Python:

```
class vector:
    def __init__(self, x, y):
        self.x = x
        self.y = y

    def add(self, other):
        return vector(self.x + other.x, self.y +
                                        other.y)

    def __mul__(self, scaler):
        return vector(self.x * scaler, self.y * scaler)

    def __str__(self):
        return f"vector({self.x}, {self.y})"

v1 = vector(1, 2)
v2 = vector(3, 4)
result_addition = v1 + v2
print("Addition:", result_addition)
result_multiplication = v1 * 2
print("Multiplication:", result_multiplication)
```

PAGE No.
DATE / /

Output

Addition : vector (4, 6)

Multiplication : vector (2, 4)

Anusha
26/3/24