**PYTHON Assignment**

**MODULE: 3 (Advanced Python Programming)**

1. What is File function in python? What are keywords to create and write file?

Ans. In Python, the open() function is used to interact with files. It is a built-in function that is used to open a file and return a file object that can be used to read, write, or manipulate the file.

***Syntax:***

open(`file_name`,`mode`)

To create and write in a file, we typically use the 'w' mode.

***Syntax:***

```
open(`file_name`,`w`)
file.write('Hello, this is a sample text.')
```

2. Explain Exception handling? What is an Error in Python?

Ans. Exception handling in Python refers to the mechanism that allows you to handle errors or exceptional situations in a program gracefully, preventing the program from crashing and providing a way to handle errors in a controlled manner.

In Python, errors are typically classified into two types:

**a) Syntax Errors:**

- It is also known as parsing errors and occurs when the Python interpreter encounters code that violates the language's syntax rules.
- These errors are detected during the parsing phase (when the interpreter reads the code).
- Examples of syntax errors include missing colons at the end of statements, mismatched parenthesis, or misspelled keywords.

**b) Exceptions:**

- Exceptions are errors that occur during the execution of a program. They can be handled in such a way to prevent the program from terminating abruptly.
- Common exceptions include Zero Division Error, File Not Found Error, Type Error, Value Error, and so on.
- Exception handling is implemented using the try, except, else, and finally blocks.

3. How many except statements can a try-except block have? Name some built-in exception classes.

Ans. A try block in Python can have multiple except blocks to handle different types of exceptions. This allows you to handle various exceptions differently based on their types.

- Here are some common built-in exception classes in Python:

a) **Exception:**
   The base class for all exceptions. It catches all exceptions that derive from it.
b) **Syntax Error:**
   Raised for syntax errors.
c) **Indentation Error:**
   Raised for indentation errors.
d) **Name Error:**
   Raised when a local or global name is not found.
e) **Type Error:**
   Raised when an operation or function is applied to an object of an inappropriate type.
f) **Value Error:**
   Raised when a built-in operation or function receives an argument of the correct type but an invalid value.
g) **Zero Division Error:**
   Raised when the second operand of a division or modulo operation is zero.
h) **File Not Found Error:**
   Raised when a file or directory is requested, but it cannot be found.
i) **IO Error:**
   Raised when an I/O operation (such as opening or reading a file) fails.
j) **Key Error:**
   Raised when a dictionary key is not found.
k) **Index Error:**
   Raised when a sequence subscript is out of range.
l) **Attribute Error:**
   Raised when an attribute reference or assignment fails.
m) **OS Error:**
   A base class for I/O and operating system-related errors.

4. When will the else part of try-except-else be executed?

Ans. The else block in a try-except-else construct is executed only if the try block does not raise any exceptions. The else block is optional and is executed when the code in the try block runs successfully without any exceptions being raised.

5. Can one block of except statements handle multiple exception?

Ans. Yes, a single except block can handle multiple exceptions by specifying them as a tuple. This allows you to catch and handle different types of exceptions using the same block of code.

***For example:***

```
try:
    x = int("abc")  # Raises a ValueError
except (ValueError, TypeError) as e:
    print(f"Error: {e}")
```

6. When is the finally block executed?

Ans. The finally block in a try-except-finally construct is executed regardless of whether an exception occurs or not. It ensures that certain code is always executed, whether an exception is raised in the try block or not.

7. What happens when “1”==1 is executed?

Ans. When the expression "1" == 1 is executed in Python, it evaluates to False. This is because the equality operator (==) compares the values on both sides of the operator for equality, but it does not perform data type coercion.

In this case:

The left side of the expression, "1", is a string.

The right side of the expression, 1, is an integer.

Since the data types are different, the equality comparison returns False. The string "1" is not equal to the integer

8. How Do You Handle Exceptions With Try/Except/Finally In Python? Explain with coding snippets.

Ans. In Python, the try, except, and finally blocks are used to handle exceptions. Here's an explanation with coding snippets for each part:

a) **Using try and except:**

The try block contains the code that may raise an exception. If an exception occurs, it is caught by the except block, and you can handle the exception or perform specific actions.

*For example:*

```
try:
    # Code that may raise an exception
    result = 10 / 0  # Raises a ZeroDivisionError
except ZeroDivisionError as e:
    # Handle the exception
    print(f"Error: {e}")
```

b) **Using try, except, and else:**

The else block is executed if no exception occurs in the try block. It's used for code that should run only when the try block is successful.

*For example:*

```
try:
    # Code that may raise an exception
    result = 10 / 2  # No exception here
except ZeroDivisionError as e:
    # Handle the exception
    print(f"Error: {e}")
else:
    # Code to be executed if no exception occurred in the try block
    print(f"Result: {result}")
```

c) **Using try, except, else, and finally:**

The finally block is optional and is executed regardless of whether an exception occurred or not. It's typically used for cleanup operations.

*For example:*

```
try:
    # Code that may raise an exception
    result = 10 / 2  # No exception here
```

```
except ZeroDivisionError as e:
        # Handle the exception
        print(f"Error: {e}")
else:
        # Code to be executed if no exception occurred in the try block
        print(f"Result: {result}")
finally:
        # Code to be executed regardless of whether an exception occurred or not
        print("Finally block executed.")
```

9. What are OOPs concepts? Is multiple inheritance supported in java?

Ans. OOP stands for **O**bject **O**riented **P**rogramming. In this type of programming the process can happen in random order. It creates objects that contain data and method, rather than writing procedures or methods that perform operations on the data. Examples of OOP languages are C++, Java, Pearl, Asp.net, etc.

Following are the concepts of OOP:

1. **Class**

- It is considered to be the building block of OOP.
- It is collection of data member and member function.
- It is a blueprint for creating objects.

2. **Object**

- It is an instance of a class
- It encapsulates data (attributes) and behavior (methods)

3. **Encapsulation/Data Binding**

- It refers to binding of data and methods that operate as a single unit called class.
- It restricts direct access to some of the object's component to prevent unintended interface and misuse.

4. **Access Specifier/Access Modifier/Data Hiding**

- It refers to hiding of data.
- It allows programmers to focus on relevant aspects of an object while hiding unnecessary details.

5. **Inheritance**

- It is a mechanism by which a new class, called a subclass or derived class, inherits properties of existing class, called a superclass or base class.
- It increases reusability of the code and creates a hierarchical relation between classes.

- In python, there are three major forms of inheritance used widely:

a) Single Inheritance
b) Multiple Inheritance
c) Multi Level Inheritance

**6. <u>Polymorphism</u>**

- It allows objects of different types to be treated as objects of common type.
- It enables to use same item in multiple ways.
- This can be achieved through method overloading (same class, same method name with different parameters) and method overriding (same method name, same parameter with different class).

❖ JAVA does not support multiple inheritance, but a form of it is supported called interfaces. An interface in JAVA is a collection of abstract methods, and a class can implement multiple interfaces.

10. How to Define a Class in Python? What Is Self? Give An Example Of A Python Class

Ans. In Python, you can define a class using the class keyword. A class is a blueprint for creating objects, and it defines attributes and methods that the objects will have. The `self` parameter is used inside methods to refer to the instance of the class.

- ***<u>For example:</u>***

```
# To create a class named rectangle
class Rectangle:
   # To get user input for length of rectangle
   l=float(input("Enter value for length of a rectangle:"))
   # To get user input for width of a rectangle
   w=float(input("Enter value for width of a rectangle:"))
   # To create UDF for calculating area of rectangle
   def compute(self):
      print("Area of rectangle is:", self.l*self.w)
# To create object for class rectangle
Area=Rectangle()
# To call UDF to compute the area
Area.compute()
```

11. Explain Inheritance in Python with an example? What is init? Or What Is A Constructor In Python?

Ans. Inheritance allows a new class to inherit attributes and methods from an existing class. The existing class is referred to as the parent class or base class, and the new class is called the child class or derived class. This enables code reuse and the creation of a hierarchical structure among classes.

The __init__ method is a special method in Python called a constructor. It is automatically called when an object is created from the class.

- ***For example:***

```
class Animal:
    def __init__(self, name):
        self.name = name
def make_sound(self):
        pass  # Placeholder method, to be overridden by child classes

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

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

# Creating instances of the child classes
my_dog = Dog(name="Buddy")
my_cat = Cat(name="Whiskers")

# Accessing attributes and methods
print(f"{my_dog.name} says {my_dog.make_sound()}")
print(f"{my_cat.name} says {my_cat.make_sound()}")
```

- ***Explanation of code:***

*The Animal class is the parent class, and it has an __init__ method (constructor) that initializes the name attribute. It also has a make_sound method, which is meant to be overridden by child classes.*

*The Dog and Cat classes are child classes that inherit from the Animal class. They provide their own implementation of the make_sound method.*
*In this example, the constructor initializes the name attribute.When instances of Dog and Cat are created, they inherit the name attribute from the Animal class and provide their own implementation of the make_sound method. This is an example of how inheritance allows for code reuse and helps create a more organized and modular code structure.*

12. What is Instantiation in terms of OOP terminology?

Ans. The process of creating an instance of a class is called instantiation. This involves allocating memory for the object, initializing its attributes (if any), and returning a reference to the newly created instance.

13. What is used to check whether an object o is an instance of class A?

Ans. In Python, you can use the isinstance() function to check whether an object is an instance of a particular class.

- The syntax for isinstance() is as follows:
  isinstance(object, classinfo)

Here, object is the object you want to check, and classinfo is the class (or a tuple of classes) against which you want to check the instance.

- To check whether an object o is an instance of class A, you can use the following:

```
result = isinstance(o, A)
if result:
        print("Object o is an instance of class A.")
else:
        print("Object o is not an instance of class A.")
```

The isinstance() function will return True if o is an instance of class A, and False otherwise.

14. What relationship is appropriate for Course and Faculty?

Ans. The relationship between a Course and a Faculty can be modeled using various associations depending on the business requirements and the nature of the interaction between the two entities.

Common relationship types in this context are:

a) Many-to-Many Relationship:
   If a course can have multiple faculties and a faculty can be associated with multiple courses, a many-to-many relationship might be suitable.

b) One-to-Many Relationship:
   If each course is associated with one faculty but a faculty can teach multiple courses, a one-to-many relationship may be more appropriate.

15. What relationship is appropriate for Student and Person?

Ans. The relationship between Student and Person can be modeled using the concept of inheritance, as students are a specialized type of person. This relationship is commonly known as an "is-a" relationship, where a Student is a specific type of Person. In object-oriented programming, this is achieved through inheritance.