29/3/23

# ASSIGNMENT 2

PAGE NO.: 18
DATE: / /

Unit - 3 & 4 Control Structures Function & OOPS

**Q.1** What is decision making ? Why do we need Decision making in programming explain with an example.

—↓ Decision making is the process of choosing between different alternatives based on some criteria or objectives. In programming, decision making is a crucial concept that enables a computer program to make choices and execute different actions based on specific conditions or inputs.

The most common way to implement decision-making in programming is through conditional statements such as "if-else" or "Switch".

—↓ For example, consider a program that needs to determine whether a number is positive or negative.

```
1. number = -6
2. if number >= 0 :
3.         print ("The number is positive")
4. else :
5.         print ("The number is negative")
```

OUTPUT :- The number is negative

**Q.2** Explain if Statement with an example.

—↓ In programming, an "if" statement is a type of conditional statement that allows a program

to execute different block of codes depending on whether a certain condition is true or false.

example:
```
1 age = 20                    # You are an adult
2 if age >=18 :
      print ("You are an adult")
```

In this example, the variable "age" has a value of 20. The "if" statement here checks whether the value of "age" is greater than or equal to 18.

Q.3 Explain if else statement with an example
—4 In programming, an "if-else" statement is a type of conditional statement that allows a program to execute one block of code if a certain condition is True, and a different block of code if the condition is false.

example:
```
1 age = 16                    # You are not yet
                                an adult
2 if age >= 18 :
3       print ("You are an adult")
4 else :
      print ("You are not yet an adult")
```

In this example, the variable "age" has a value of 16. The 'if' statement here checks whether the value of 'age' is greater than or equal to 18, since 16 is less than 18, the condition is false so and the program will executes the code inside the 'else' block.

Q.4 Explain if... elif ... else statement with an example.

→ In Programming, an 'if-elif-else' statement is a type of conditional statement that allows a program to execute different blocks of code based on multiple conditions.

example:1
```
score = 75                                    # C
if score >= 90:
    print ("A")
elif score >= 80:
    print ("B")
elif score >= 70:
    print ("C")
elif score >= 60:
    print ("D")
else:
    print ("F")
```

In this example, we have a variable called 'score' having value of 75. Since 75 is between 70 and 80, the program executes the code inside the "elif score >=70" block, which ~~assigns~~ print the value "C", to

Q.5 Explain the Nested if statement with an example.

→ In programming, a "nested if" statement is a type of conditional statements that allows a program to check multiple conditions in a hierarchical manner. A nested if statement contains one or more if statements within another if statement

example:1
```
age = 20                                    # You are an adult & cannot
weight = 120                                  ride the roller coaster
if age >= 18 :
    if weight >= 100 :
        print ("You are an adult & can ride
        the roller coaster")
    else :
        print ("You are an adult & cannot ride
        the roller coaster")
else:
    print ("you are not an adult")
```

In this example, the program uses nested if statement to check whether the person is both an adult and meets the weight requirement to ride a roller coaster.

Q.6 Explain for Loop with an example

→ In programming, a "for loop" is a control flow statement that allows a program to repeatedly execute a block of code a certain number of times or iterate over a sequence of elements.

example:1
```
numbers = [2, 4, 6, 8]
for x in numbers :
    print (x)
```
```
# 2
# 4
# 6
# 8
```

In this example, we have a list of numbers that contains four elements: 2, 4, 6, 8. The program uses a loop to iterate over the elements in the list and print each one.

**Q7** Explain range() function with an example.

—↓ In programming, the 'range()' function is a built-in function in Python that generates a sequence of numbers. It can be used to create a range of integers, with different options specifying the start, stop & step size.

eg: 1. for i in range (1, 10, 3):
        print (i)

# 1
# 4
# 7

In this example, the 'range()' function creates a sequence of numbers starting from 1, up to but not including, 10 with a step size of 3.

**Q.8** Explain while loop Nested loop with an example.

—↓ In programming, a "while loop" is a control flow statement that allows a program to repeatedly execute a block of code as long as a certain condition is true. A "nested loop" is a loop that is contained within another loop, allowing a program to iterate over a set of values multiple times.

eg:
```
1. #While loop
2. count = 0
3. while count <=5:
4.     print (count)
5.     count += 1
```

# 0
# 1
# 2
# 3
# 4
# 5

eg:
```
1  # Nested loop                    # while: 1
2  i = 1                            # for: 1
3  while i<3:                       # for: 2
4      print("while:", i)           # while: 2
5      for j in range(1,3):         # for: 1
6          print("for:", j)         # for: 2
7      i+=1
```

Q.9 Explain Control Statements with Example

—↓ In programming, "Control Statements" are statements that change the flow of execution in a program. There are three types:

i) Conditional Statements: It allows a program to execute different code based on the value of a particular condition.

⇒
```
x = 15                              # x is greater than 10
if x > 10:
    print("x is greater than 10")
else:
    print("x is less than 10")
```

ii) Loop statements: It allows a program to repeat a block of code multiple times.

⇒
```
for i in range(3):                  # 0
    print(i)                        # 1
                                    # 2
```

iii) Jump Statements: It allows a program to interrupt the normal flow of execution & jump to a different point in the program.

⇒
```
for i in range(5):                  # 0
    if i == 3:                      # 1
        break                       # 2
    print(i)
```

Q.10 Write a program to find GCD of 2 numbers in Python.

→
```
a = int(input("Enter first number: "))
b = int(input("Enter second number: "))
while b!=0:
    temp = b
    b = a
    a = temp
print("GCD: ", a)
```

Output:
```
Enter first number: 5
Enter second number: 55
GCD: 5
```

Q.11 Write a program that prints the number from 1 to 20. But for multiples of three print "Fizz" instead of the number and for the multiples print "Buzz". For numbers which are multiple of both three and five print "FizzBuzz".

→
```
for i in range(1,20):
    if i%3==0 and i%5==0:
        print("FizzBuzz")
    elif i%3==0:
        print("Fizz")
    elif i%5==0:
        print("Buzz")
    else:
        print(i)
```

Output:

| | | | | | | |
|---|---|---|---|---|---|---|
| 1 | 4 | 7 | Buzz | 13 | 16 | 19 |
| 2 | Buzz | 8 | 11 | 14 | 17 | Buzz |
| Fizz | Fizz | Fizz | Fizz | FizzBuzz | Fizz | |

Q.12 Write a program to count the number of vowels in python.

→
```
string = input ("Enter a String: ")
vowels = ['a', 'e', 'i', 'o', 'u']
count = 0
for i in string:
    if i.lower() in vowels:
        count += 1
print ("The number of vowels: ", count)
```

Output:
```
Enter a String: Hello
The number of vowels: 2
```

Q.13 Write a program to find the sum of all items in a list.

→
```
list = [1, 2, 3, 4, 5]
sum = 0
for i in list:
    sum += i
print ("Sum: ", sum)
```

Output:
```
Sum: 15
```

Q.14 Explain the function with example

→ A function is a block of organized, reusable code that performs a specific task. Functions help make code more modular, easier to read, and easier to maintain. In we can define a function using the 'def' keyword.

eg:
```
def area (l, b):
    a = l * b
```

```
3.  return a
4.  result = area(5,10)
5.  print("The area of the rectangle: ", result)
```

Output: The area of the rectangle: 50

Q.15 Explain the function type with example

→ There are several types of functions, including

i) Built-in functions

These are functions that are already defined in the programming language and can be used directly without any additional setup or configuration. Examples of this function includes print(), len() & range().

ii) User-defined functions

These are functions that are created by the programmer to perform specific tasks.

eg:
```
def sum(a,b):
    return a+b
```

iii) Anonymous functions

Also known as Lambda functions, these are small, one-line functions that do not have a name and are used for simple tasks.

eg:
```
result = lambda x, y : x * y
```

iv) Recursive functions

These are functions that call themselves, either directly or indirectly, to solve a problem. Recursive functions are often used for problems that can be broken down into smaller sub-problems.

eg:
```
def factorial (n):
    if n == 0:
        return 1
    else:
        return n * factorial (n-1)
```

Q.16 Explain function arguments with examples.

→ Function arguments are the inputs that a function receives when it is called. Arguments allow functions to take input data and perform operations on that data. There are different types of function arguments, which are:

i) Positional Arguments
ii) Keyword Arguments
iii) Default Arguments
iv) Variable - length Arguments

eg:
```
def add (a, b):
    return a + b
result = add (2, 3)
print (result)          # Output : 5
```

Q.17 Explain Anonymous Functions (Lambda Functions) with an example.

→ This is a function that is defined without a name and are often used for simple tasks and are created using the Lambda keyword.

eg:
```
add_numbers = lambda a, b: a + b
result = add_numbers (2, 3)
print (result)          # Output 5
```

→ Lambda function can be used with other built-in

functions like map() and filter().

eg:
```
numbers = [1, 2, 3, 4, 5]
squared_numbers = list(map(lambda x: x**2, numbers))
print(squared_numbers)

# Output : [1, 4, 9, 16, 25]
```

Q.18 → Explain Recursive Function with an Example.

A recursive function is a function that calls itself its execution. It can be used to solve problems that can be broken down into smaller, sompler versions of the same problem. Each recursive function works on a smaller portion of the problem until a base case is reached, at which point the function stops calling itself and returns a final result

eg:
```
def factorial(n):
    if n == 1:
        return 1
    else:
        return n * factorial(n-1)

result = factorial(5)
print(result)

# Output : 120
```

This process continues recursively until the base case is reached, at which point the final result (120) is returned.

Q.19 Write a python function to check whether a number is even or odd.

—↓

```
def check(num):
    if num % 2 == 0:
        return "Even"
    else:
        return "Odd"

print(check(5))          # Output: Odd.
```

Q.20 Write a python program to display Fibonacci series using recursion.

—↓

```
def fibonacci(n):
    if n <= 1:
        return n
    else:
        return fibonacci(n-1) + fibonacci(n-2)

terms = int(input("Enter the number of terms you want to display: "))

if terms <= 0:
    print("Please enter a positive integer: ")
else:
    print("Fibonacci Sequence: ")
    for i in range(terms):
        print(fibonacci(i))
```

# Output: Enter the number of terms you want to display: 5

0 2
1 3
1

Q.21 What is class, what is object what is method explain with an example

—> A class is a blueprint for creating objects that define a set of attributes and methods that the objects will have.

—> An object is an instance of a class and it contains data and methods defined in the class.

—> A method is a function defined in a class that can be called on an object of that class.

eg:
```
class Person:
    def __init__(self, name, age):
        self.name = name
        self.age = age

    def greet(self):
        print("Hello, my name is", self.name, "and I am", self.age, "years old.")

person1 = Person("Alice", 25)
person1.greet()
```

Q.22 What is data encapsulation explain with an example

—> Data Encapsulation is the practice of hiding the implementation detail of a class from the outside world and only exposing a public interface through which the class can be used.

eg:
```
class BankAccount :
    def __init__(self, acc_num, balance):
        self.__acc_num = acc_num
        self.__balance = balance

    def deposit(self, amount):
        self.__balance += amount

    def withdraw(self, amount):
        if amount <= self.__balance:
            self.__balance -= amount
        else:
            print("Insufficient funds.")

    def get_balance(self):
        return self.__balance

acc1 = BankAccount("1234567890", 1000)

print(acc1.__acc_num)
print(acc1.__balance)
acc1.deposit(500)
acc1.balance(200)
print("Balance:", acc1.get_balance())
```

Q.23 What is inheritance explain with an example

→ Inheritance is a concept where a class can inherit attributes and methods from another class, called the superclass or parent class. The subclass can then add or modify these attributes and methods as needed, while still retaining the ones inherited from the parent class.

eg:
```
class Animal:
    def __init__(self, name, sound):
        self.name = name
        self.sound = sound

    def make_sound(self):
        print(self.sound)

class Dog(Animal):
    def __init__(self, name):
        super().__init__(name, "Woof")

dog1 = Dog("Rufus")
dog1.make_sound()   # Output: "Woof"
```

Q.24 Explain different types of inheritance with examples.

→ i) Single Inheritance

A class inherits attributes and methods from a single parent class.

eg:
```
class animal():
    def speak(self):
        print("Animal Speaking")

class Dog(animal):
    def bark(self):
        print("Dog barking")

d = Dog()
d.speak()   # Output: Animal Speaking.
d.bark()    # Output: Dog Barking.
```

ii) Multiple Inheritance

A class can inherit attributes and methods from multiple parent classes.

eg:
```
class Base1:
    def speak(self):
        print("Base1 Speaking")

class Base2:
    def talk(self):
        print("Base2 & Talking")

class Derived(Base1, Base2):
    def listen(self):
        print("Derived Listening")

d = Derived()
d.speak()      # Base1 Speaking
d.talk()       # Base2 Talking
d.listen()     # Derived Listening.
```

iii) Multi-level Inheritance

A class inherits attributes and methods from a parent class, which itself inherits from another parent class.

eg:
```
class Animal():
    def speak(self):
        print("Speak")

class Dog(Animal):
    def bark(self):
        print("Bark")
```

```
class Labrador(Dog):
    def breed(self):
        print("Labrado")

l = Labrador()
l.speak()      # Speak
l.bark()       # Bark
l.breed()      # Labrador
```

ii) ~~Hire~~ Hierarchical Inheritance.
Multiple classes inherit from a single parent class.

eg:
```
class Animal:
    def speak(self):
        print("Speaking")

class Dog(Animal):
    def bark(self):
        print("Barking")

class Cat(Animal):
    def meow(self):
        print("Meowing")

d = Dog()
d.speak()      # Speaking
d.bark()       # Barking
c = Cat()
c.speak()      # Speaking
c.meow()       # Meowing
```

Q.25 Explain Polymorphism with an example. Explain method overriding and operator overloading with an example

→ Polymorphism is the ability of objects to take on multiple forms or meanings.

→ Method Overriding is a feature of inheritance where a subclass provides a specific implementation of a method that is already provided by its superclass. When a method is overridden in the subclass, the implementation in the subclass.

eg:
```
class Animal:
    def speak(self):
        print("Speaking")

class Dog(Animal):
    def speak(self):
        print("Barking")

d = Dog()
d.speak()    # Output: Barking.
```

→ Operator ~~overri~~ Overloading is a feature where the behaviour of an operator is defined for a user-defined data type.

eg:
```
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)

V1 = Vector(2,4)
V2 = Vector(1,3)
V3 = V1 + V2
print(V3.x, V3.y) # Output: 3 7
```

~~~~ x ~~~~ x ~~~~

Twinkle
~~~~