Date 20/02/25
Page 01

2300311016l1051

ASSIGNMENT - 3

Ques. Explain the difference between if, ~~else~~ elif, and else statements in Python. Provide a practical example.

Soln: The 'if' statement in Python is used to check a condition, and if it evaluates to True, the correspond-ing block of code executes. If the condition is False, the program moves on.

The 'elif' statement, short for "else-if", provides additional conditions to check when the previous if conditions fails, allowing multiple conditional branches.

The 'else' statement serves as a fallback, executing when none of the preceding conditions are met, ensuring that there is always a defined outcome.

Ex(i) Here's a small example to check if a number is even or odd :-

```
num = int(input("Enter a number: "))
if num % 2 == 0:
    print("Even Number")
else:
    print("Odd Number")
```

Ex(ii): Another example of using if-elif, else to check if a number is positive, negative or zero.

2300311016111051

```
num = int(input("Enter a number: "))
if num > 0:
    print("Positive number")
elif num < 0:
    print("Negative number")
else:
    print("Zero")
```

Ques. write a Python program that uses a for loop to iterate through a list of numbers and prints each numbers and its square.

Soln:→

```
numbers = [2, 3, 4, 5]
for num in numbers:
    print(f"Number: {num} & Square: {num**2}")
```

output:

Number: 2 Square: 4
Number 3 Square: 9
Number: 4 Square: 16
Number: 5 Square: 25

Ques. What is the purpose of breaking statements in loops? Provide an example scenario where it would be appropriately used.

Soln: Breaking statements or Jumping statements, such as break, are used in loops to prematurely terminate

2300311016l1051

their execution when a specific condition is met. Instead of running through all iterations, the loop stops immediately, improving efficiency and preventing unecessary processing. This is particularly useful when searching for an item in a list, handling user input, or stopping execution when a condition is satisfied. without 'break', the loop would continue running, even when the desired result is already found.

∴ Example:- (code to check whether a number exists in a list or not):-

```
numbers = [10, 20, 30, 40, 50, 60, 70, 80, 90, 100]
check = 40
for num in numbers:
    if num == check:
        print(f"Number {num} found in list!")
        break
```

Output:

Number 40 found in list!

Ques. Describe the importance of functions in Python. How do they help organize and manage code?

Soln:→ Functions in Python help to organize code by breaking complex tasks into smaller, reusable blocks. Instead

of repeating code, functions allow programmers to define operations once and use them multiple times, improving efficiency and readability. They follow the DRY (Don't Repeat Yourself) principle, reducing redundancy and making code easier to maintain.

Functions also make debugging simpler, as issues can be fixed in one place without affecting the entire program. They enhance collaboration in team projects, as different team numbers can work on different functions independently. Additionally, functions improve abstraction by hiding complex logic, keeping the main program clean and concise.

In large applications, functions can be grouped into modules for better organization. Overall, they are essential for writing 'efficient', 'reusable', and 'maintainable' code, making programming easier and more structured.

Quer. Develop a Python module containing functions for basic arithmetic operations (add, subtract, multiply, divide). write a program to import this module and use its functions.

Soln:→ arithmetic_module.py

```
def add(a,b):
    return a+b
```

2300311016l1051

```
def subtract(a,b):
    return a-b
def multiply(a,b):
    return a*b
def divide(a,b):
    return a/b
```

∴ main.py

```
import arithmetic_module
def main():
    print("Basic Arithmetic operations using
                Module:-")
    x=10
    y=5
    print(f"{x} + {y} = {arithmetic_module.
                add(x,y)}")
    print(f"{x} - {y} = {arithmetic_module.
                subtract(x,y)}")
    print(f"{x} * {y} = {arithmetic_module.
                multiply(x,y)}")
    print(f"{x} / {y} = {arithmetic_module.
                divide(x,y)}")
```

∴ Output:-

```
10 + 5 = 15
10 - 5 = 5
10 * 5 = 50
10 / 5 = 2
```

Date 20/02/25
Page 06

2300311016111051

Ques. write a Python script that uses a lambda function combined with the map() and filter() functions to process a list of numbers. The script should square each number in the list and then filter out the squares that are even. Print the final list of odd squares.

Soln:→
```
numlist = [1,2,3,4,5,6,7,8,9,10]
odd_squares = list(filter(lambda x: x%2!=0,
                          map(lambda x: x**2,
                              numlist)))
print("Odd squares:", odd_squares)
```

Output: odd squares: [1, 9, 25, 49, 81]

Ques. Implement a recursive function that generates the nth Fibonacci number. Then, write a script that uses this function to print the first 15 Fibonacci numbers.

Soln:→
```
def fibonacci(n):
    if n==0:
        return 0
    elif n==1:
        return 1
    else:
        return fibonacci(n-1) + fibonacci(n-2)
for i in range(15):
    print(fibonacci(i), end=' ')
```

Output: 0 1 1 2 3 5 8 13 21 34 55 89 144 233 377