**Data Analytics**

Course Objectives:

1. Learning about the Importance of Data and its importance.
   1. Data, it is gained by processing the knowledge, for every instance of work we need data regarding it.
   2. Data ensures the workflow with its necessity
2. Knowing Python fundamentals and Pandas essentials.
   1. Python, it is a programing language which could be helpful in Analysing data
   2. Python consists of a huge set of libraries, around 700+ where pandas is a part of it.
   3. Pandas, the library majorly used for prediction the outcome based on huge data
3. Learning the Principles of Probability and Sampling Methods.
4. Getting knowledge about formulating and testing hypotheses.
5. Learning and analytical comparison with ANOVA methods.
6. Learning about Performance indicators using ROC methods.

* **UNIT I**

**Introduction to Data Analytics:**  Data and its importance, data and its types, importance of data analysis.

**Python Fundamentals:**  Python language Basics, Jupyter Notebook, Introduction to pandas, Data Structures, Essential Functionality.

**Central Tendency and Dispersion:**  Visual Representation of the Data, Measures of Central Tendency Dispersion

**Program 1:Write a Python Program to swap two numbers**

#Swapping of Two Numbers

a = 10

b = 20

print("Before Swapping of Elements")

print("a =", a)

print("b =", b)

# Swapping logic

print("After Swapping of Elements")

a = a + b

b = a - b

a = a - b

print("a =", a)

**Initial Setup**

**python**

**Copy code**

**a = 10**

**b = 20**

1. **a is initialized with 10 and b is initialized with 20.**

**Before Swapping**

**python**

**Copy code**

**print("Before Swapping of Elements")**

**print("a =", a)**

**print("b =", b)**

1. **The program prints the values of a and b before the swap:**

**Output:  
css  
Copy code  
Before Swapping of Elements**

**a = 10**

**b = 20**

**Swapping Logic**

**Step 1: Calculate the Sum and Assign to a**

**python**

**Copy code**

**a = a + b**

1. **The first operation adds a and b, and the result is assigned to a. Now, a holds the sum of the original a and b:**
   * **a = 10 + 20 = 30**
   * **After this step:**
     + **a = 30**
     + **b = 20**

**Step 2: Assign the Original Value of a to b**

**python**

**Copy code**

**b = a - b**

1. **The second operation calculates the difference between the new value of a (which is 30) and the current value of b (which is 20), and assigns the result to b. This gives us the original value of a.**
   * **b = 30 - 20 = 10**
   * **After this step:**
     + **a = 30**
     + **b = 10**

**Step 3: Assign the Original Value of b to a**

**python**

**Copy code**

**a = a - b**

1. **The final step calculates the difference between the current value of a (which is 30) and the new value of b (which is 10), and assigns the result to a. This gives us the original value of b.**
   * **a = 30 - 10 = 20**
   * **After this step:**
     + **a = 20**
     + **b = 10**

**After Swapping**

**python**

**Copy code**

**print("After Swapping of Elements")**

**print("a =", a)**

**print("b =", b)**

1. **The program prints the values of a and b after the swap:**

**Output:  
css  
Copy code  
After Swapping of Elements**

**a = 20**

**b = 10**

**Summary of the Swap Process**

1. **Before the Swap:**
   * **a = 10, b = 20**
2. **After Step 1 (a = a + b):**
   * **a = 30, b = 20**
3. **After Step 2 (b = a - b):**
   * **a = 30, b = 10**
4. **After Step 3 (a = a - b):**
   * **a = 20, b = 10**

**Thus, a and b are successfully swapped using arithmetic operations without needing a temporary variable.**

**2 Program: Write a Python Program for Fibonacci series**

 # Fibonacci series

n = int(input("Enter the number of terms: "))

# Validate input

if n <= 0:

    print("Please enter a positive integer.")

elif n == 1:

    print("Fibonacci series:")

    print(0)  # For n = 1, only the first term is printed

else:

    print("Fibonacci series:")

    a, b = 0, 1

    for \_ in range(n):

        print(a, end=" ")  # Print the current term

        a, b = b, a + b    # Update the terms

**Explanation:**

1. Input the number of terms n for the Fibonacci sequence.
2. Initialize the first two terms as a = 0 and b = 1.
3. Use a for loop to iterate n times:
   * Print the current value of a.
   * Update a and b for the next term in the sequence.

**Example Walkthrough**

Let's assume n = 5:

* Initial values: a = 0, b = 1.

**Iteration 1**:

* Print: a = 0.
* Update: a, b = b, a + b → a = 1, b = 1.

**Iteration 2**:

* Print: a = 1.
* Update: a, b = b, a + b → a = 1, b = 2.

**Iteration 3**:

* Print: a = 1.
* Update: a, b = b, a + b → a = 2, b = 3.

**Iteration 4**:

* Print: a = 2.
* Update: a, b = b, a + b → a = 3, b = 5.

**Iteration 5**:

* Print: a = 3.
* Update: a, b = b, a + b → a = 5, b = 8.

**3.Program**

**Write a Python Program to  find whether given number is prime or not**

# Input: Number to check if it's prime

num = int(input("Enter a number: "))

# A prime number is greater than 1 and has no divisors other than 1 and itself

if num <= 1:

    print(f"{num} is not a prime number.")

else:

    # Check divisors from 2 to the square root of the number

    for i in range(2, int(num\*\*0.5) + 1):

        if num % i == 0:

            print(f"{num} is not a prime number.")

            break

    else:

        print(f"{num} is a prime number.")

**Explanation:**

1. **Prime Number Definition**:
   * A prime number is greater than 1 and is divisible only by 1 and itself.
2. **Handling Edge Cases**:
   * Numbers ≤ 1 are not prime, so they are handled first.
3. **Efficient Check**:
   * Instead of checking divisors up to num - 1, check only up to the square root of the number (num\*\*0.5) for efficiency. If a number has a divisor larger than its square root, the smaller divisor would have already been found.
4. **Loop with else**:
   * The else part of the loop runs only if the loop completes without finding a divisor, confirming the number is prime.

**Explanation:**

1. **Prime Number Definition**:
   * A prime number is greater than 1 and is divisible only by 1 and itself.
2. **Handling Edge Cases**:
   * Numbers ≤ 1 are not prime, so they are handled first.
3. **Efficient Check**:
   * Instead of checking divisors up to num - 1, check only up to the square root of the number (num\*\*0.5) for efficiency. If a number has a divisor larger than its square root, the smaller divisor would have already been found.
4. **Loop with else**:
   * The else part of the loop runs only if the loop completes without finding a divisor, confirming the number is prime.