Semester III Python Programming 2025 New Syllabus SEP

Practice Programs:

1. Python program to solve the Fibonacci sequence using recursion.
2. Python function to check whether a number is perfect or not.
3. Python program to converting an Integer to a String in any base.
4. Python program that uses List Comprehension to perform any 3

a. Create an output list which contains only the even numbers From the input list

b. Create an output list which contains squares of all the numbers from 1 to 9.

c. Create an output list which extracts all the numbers from an input string .

5. Python program to count the number of lines in a text file.
6. Python program to copy the contents of a file to another file.
7. Python GUl program to draw various shapes on Canvas.

PART A

1. Python function to calculate the factorial of a number (a non-negative integer). The fun accepts the number as an argument.

2. Python function that takes a list and returns a new list with unique elements of the first list.

3. Python program of recursion list sum.

4. Python program to get the sum of digits of a non-negative integer.

5. Python program to demonstrate any 5 string and List operations.

6. Create an output tuple that converts the words to uppercase from the input tuple of words.

7. Python program to demonstrate any 5 operations performed on dictionary.

8. Python program to create a module Calculation.py that contains functions to perform arithmetic operations. Demonstrate importing the module.

PART B

1. Python program to demonstrate modification of an existing table data from MySQL database.

2. Python class named Circle constructed by a radius and two methods which will compute the ; and the perimeter of a circle.

3. Python class named Rectangle constructed by a length and width and a method which, compute the area and perimeter of rectangle. Inherit a class Box that contains additional meth volume. Override the perimeter method to compute perimeter of a Box.

4. Python program to show use of Regular expressions with match(), search. findall), sub) :

split).

5. python program to demonstrate Exception handling using 'try', 'except'. 'finally' and 'else block.

6. Python program to read a file line by line store it into an array.

7. Python GUI program to design Student Registration Form using any 5 widgets.

PART A

1. Python function to calculate the factorial of a number (a non-negative integer). The fun accepts the number as an argument.

```
def factorial(n):
  if n < 0:
    return "Factorial is not defined for negative numbers."
  elif n == 0 or n == 1:
    return 1
  else:
    result = 1
    for i in range(2, n + 1):
      result *= i
    return result

# Main Program
number = int(input("Enter a non-negative integer: "))
result = factorial(number)
print(f"The factorial of {number} is: {result}")
```

OUTPUT

1. Enter a non-negative integer: 5

The factorial of 5 is: 120

2. Enter a non-negative integer: -4

Factorial is not defined for negative numbers.

2. Python function that takes a list and returns a new list with unique elements of the first list.

```
def get_unique_elements(input_list):
  unique_list = []
  for item in input_list:
    if item not in unique_list:
      unique_list.append(item)
  return unique_list

# Main Program
original_list = [1, 2, 2, 3, 4, 4, 5, 1, 6]
print("Original List:", original_list)

unique_list = get_unique_elements(original_list)
print("List with Unique Elements:", unique_list)
```

OUTPUT :

```
Original List: [1, 2, 2, 3, 4, 4, 5, 1, 6]
List with Unique Elements: [1, 2, 3, 4, 5, 6]
```

3. Python program of recursion list sum.

```
def recursive_list_sum(lst):
  if not lst:                    #  checks if the list is empty
    return 0      #  This is the base case of the recursion ,the condition that stops the recursion.
  else:
    return lst[0] + recursive_list_sum(lst[1:])

# Main Program
my_list = [5, 3, 7, 1, 9]
print("Original List:", my_list)
total = recursive_list_sum(my_list)
print("Sum of elements (using recursion):", total)
```

OUTPUT :

```
Original List: [5, 3, 7, 1, 9]
Sum of elements (using recursion): 25
```

4. Python program to get the sum of digits of a non-negative integer.

```
def sum_of_digits(n):
  total = 0
  while n > 0:
    digit = n % 10
    total += digit
    n = n // 10
  return total

# Main Program
number = int(input("Enter a non-negative integer: "))
result = sum_of_digits(number)
print(f"Sum of digits of {number} is: {result}")
```

OUTPUT

```
Enter a non-negative integer: 12345
Sum of digits of 12345 is: 15
```

5. **Python program to demonstrate any 5 string and List operations.**

```
# ----- STRING OPERATIONS -----
print("----- STRING OPERATIONS -----")
my_string = "Python Programming"

# 1. String Length
print("1. Length of string:", len(my_string)) # Output: 18

# 2. Convert to Uppercase
print("2. Uppercase:", my_string.upper()) # Output: PYTHON PROGRAMMING

# 3. Find substring
print("3. Index of 'gram':", my_string.find("gram")) # Output: 10

# 4. Replace a word
print("4. Replace 'Python' with 'Java':", my_string.replace("Python", "Java")) # Output: Java Programming

# 5. Slicing
print("5. Slicing (0 to 6):", my_string[0:6]) # Output: Python

# ----- LIST OPERATIONS -----
print("\n----- LIST OPERATIONS -----")
my_list = [10, 20, 30, 40, 50]

# 1. Append an element
my_list.append(60)
print("1. After append:", my_list) # Output: [10, 20, 30, 40, 50, 60]

# 2. Insert an element
my_list.insert(2, 25)
print("2. After insert at index 2:", my_list) # Output: [10, 20, 25, 30, 40, 50, 60]
# 3. Remove an element
my_list.remove(40)
print("3. After removing 40:", my_list) # Output: [10, 20, 25, 30, 50, 60]
```

```
# 4. Pop an element
popped = my_list.pop()
print("4. Popped element:", popped)  # Output: 60
print("   List after pop:", my_list)

# 5. Sorting
my_list.sort()
print("5. Sorted list:", my_list)  # Output: [10, 20, 25, 30, 50]
```

OUTPUT

```
----- STRING OPERATIONS -----
1. Length of string: 18
2. Uppercase: PYTHON PROGRAMMING
3. Index of 'gram': 10
4. Replace 'Python' with 'Java': Java Programming
5. Slicing (0 to 6): Python

----- LIST OPERATIONS -----
1. After append: [10, 20, 30, 40, 50, 60]
2. After insert at index 2: [10, 20, 25, 30, 40, 50, 60]
3. After removing 40: [10, 20, 25, 30, 50, 60]
4. Popped element: 60
   List after pop: [10, 20, 25, 30, 50]
5. Sorted list: [10, 20, 25, 30, 50]
```

6. **Create an output tuple that converts the words to uppercase from the input tuple of words.**

```
# Input tuple of words
input_tuple = ("apple", "banana", "cherry", "date")

# Create an empty list to store uppercase words
temp_list = []

# Loop through each word in the tuple
for word in input_tuple:
  temp_list.append(word.upper())  # Convert to uppercase and add to list

# Convert list back to tuple
output_tuple = tuple(temp_list)

# Display result
print("Input Tuple:", input_tuple)
print("Output Tuple (Uppercase):", output_tuple)
```

7. **Python program to demonstrate any 5 operations performed on dictionary.**

```
# ----- DICTIONARY OPERATIONS -----
print("----- DICTIONARY OPERATIONS -----")

# Creating a sample dictionary
student = {
  "name": "Alice",
  "age": 20,
  "course": "BCA",
  "year": 2
}
```

 # 1. Accessing a value using a key

```
print("1. Student Name:", student["name"])
```

 # 2. Adding a new key-value pair

```
student["college"] = "PeopleTree Education Society"
print("2. After adding 'college':", student)
```

```
# 3. Updating the value of an existing key
student["age"] = 21
print("3. After updating 'age':", student)

# 4. Removing a key-value pair using pop()
student.pop("year")
print("4. After removing 'year':", student)

# 5. Getting all keys and values
print("5. Keys:", list(student.keys()))
print("   Values:", list(student.values()))
```

OUTPUT

```
----- DICTIONARY OPERATIONS -----
1. Student Name: Alice
2. After adding 'college': {'name': 'Alice', 'age': 20, 'course': 'BCA', 'year': 2, 'college': 'PeopleTree Education Society'}
3. After updating 'age': {'name': 'Alice', 'age': 21, 'course': 'BCA', 'year': 2, 'college': 'PeopleTree Education Society'}
4. After removing 'year': {'name': 'Alice', 'age': 21, 'course': 'BCA', 'college': 'PeopleTree Education Society'}
5. Keys: ['name', 'age', 'course', 'college']
   Values: ['Alice', 21, 'BCA', 'PeopleTree Education Society']
```

8. **Python program to create a module Calculation.py that contains functions to perform arithmetic operations. Demonstrate importing the module.**

 # File: Calculation.py

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

def subtract(a, b):
  return a - b

def multiply(a, b):
  return a * b

def divide(a, b):
  if b != 0:
    return a / b
  else:
    return "Division by zero is not allowed"
```

 # File: main.py

```
import Calculation
# Using functions from the Calculation module
a = 10
b = 5

print("Addition:", Calculation.add(a, b))      # Output: 15
print("Subtraction:", Calculation.subtract(a, b)) # Output: 5
print("Multiplication:", Calculation.multiply(a, b)) # Output: 50
print("Division:", Calculation.divide(a, b))     # Output: 2.0
```

OUTPUT

```
Addition: 15
Subtraction: 5
Multiplication: 50
Division: 2.0
```

PART B

1. **Python program to demonstrate modification of an existing table data from MySQL database.**

   Requirements:

   1. ***Install the MySQL connector (if not already installed):***
   2. "**pip install mysql-connector** " type this command on vscode terminal / cmd window and wait for installation.

Python Program: Create Database , Create Table and Update Record in MySQL Table

```
import mysql.connector

# Step 1: Connect to MySQL database

conn=mysql.connector.connect(host="localhost",user="root",password="admin123")

print("\n Connection Successful !")

# Step 2: Create cursor object

cursor=conn.cursor()

cursor.execute("CREATE DATABASE if not exists PeopleTree")

# Step 2: Create DATABASE

print("\n Database PeopleTree created Successfully")

# Step 3: USE Dtabase before creating table

cursor.execute("use PeopleTree")


# Step 4: Create TABLE faculty in the Database "employee"

create_table="""CREATE TABLE if not exists Faculty

(

        id INT PRIMARY KEY UNIQUE,

        name VARCHAR(10),

        salary INT
```

```
)"""
cursor.execute(create_table)
print("\n Table Faculty created Successfully!!")
```

 # Step 5: INSERT data in table FACULTY

```
insert_query="INSERT INTO Faculty (id,name,salary) VALUES (11,'Shubha', 40000)"
cursor.execute(insert_query)
conn.commit()
print("\n Faculty inserted into table Successfully!!")
```

 # Step 6: Write the UPDATE query SET new salary for employee Shubha

```
update_query="UPDATE Faculty SET salary=50000 WHERE name='Shubha' "
```

 # Step 7: Execute the query

```
cursor.execute(update_query)
```

 # Step 8: Commit the changes

```
conn.commit()
```

 # Step 9: Print confirmation

```
print(cursor.rowcount,"Faculty salary with Name: Shubha is Updated Successfully!!!")
```

 # Step 10: Close the connection

```
cursor.close()
conn.close()
```

Sample Table Before Update (students)

id	name	salary
11	Shubha	40000

After Running the Program:

id	name	salary
11	Shubha	50000

2. Python class named Circle constructed by a radius and two methods which will compute the ; and the perimeter of a circle.

```
import math

class Circle:
  def __init__(self, radius):
    self.radius = radius

  def area(self):
    return math.pi * self.radius ** 2

  def perimeter(self):
    return 2 * math.pi * self.radius

# Main Program
r = float(input("Enter the radius of the circle: "))
c = Circle(r)

print(f"Area of the circle: {c.area():.2f}")
print(f"Perimeter of the circle: {c.perimeter():.2f}")
```

OUTPUT

```
Enter the radius of the circle: 5
Area of the circle: 78.54
Perimeter of the circle: 31.42
```

3. Python class named Rectangle constructed by a length and width and a method which, compute the area and perimeter of rectangle. Inherit a class Box that contains additional meth volume. Override the perimeter method to compute perimeter of a Box.

A Rectangle class with:

- Constructor (__init__) accepting **length** and **width**
- Methods to compute **area** and **perimeter**

A Box class that **inherits** from Rectangle, adds a **height** property, a method to compute **volume**, and **overrides** the perimeter method (to compute total edge length of a 3D box)

```
class Rectangle:
  def __init__(self, length, width):
    self.length = length
    self.width = width

  def area(self):
    return self.length * self.width

  def perimeter(self):
    return 2 * (self.length + self.width)


class Box(Rectangle):
  def __init__(self, length, width, height):
    super().__init__(length, width)
    self.height = height

  def volume(self):
    return self.length * self.width * self.height

  # Override perimeter to mean total edge length of the box
  def perimeter(self):
    return 4 * (self.length + self.width + self.height)
# Main Program
# Rectangle demo
print("Rectangle:")
r = Rectangle(10, 5)
```

```
print("Area:", r.area())
print("Perimeter:", r.perimeter())

# Box demo
print("\nBox:")
b = Box(10, 5, 4)
print("Area of base:", b.area())          # inherited method
print("Volume:", b.volume())              # box method
print("Perimeter (sum of all edges):", b.perimeter())  # overridden method
```

OUTPUT

```
Rectangle:
Area: 50
Perimeter: 30

Box:
Area of base: 50
Volume: 200
Perimeter (sum of all edges): 76
```

4. Python program to show use of Regular expressions with match(), search. Findall(), sub() : split).

Here is a **complete Python program** that demonstrates the use of **Regular Expressions (regex)** in Python using the re module, including the following functions:

- match()
- search()
- findall()
- sub()
- split()

```
import re

# Sample text
text = "Hello123, welcome to Python 3 programming. Call me at 9876543210."

# 1. match() - checks if the pattern matches at the beginning of the string
match_result = re.match(r"Hello\d+", text)
print("1. match():", match_result.group() if match_result else "No match at beginning")

# 2. search() - searches the string for a match anywhere
search_result = re.search(r"Python \d", text)
print("2. search():", search_result.group() if search_result else "Pattern not found")

# 3. findall() - returns all matches in a list
all_numbers = re.findall(r"\d+", text)
print("3. findall():", all_numbers)

# 4. sub() - replaces all occurrences of a pattern
replaced_text = re.sub(r"\d", "*", text)  # replaces digits with '*'
print("4. sub():", replaced_text)

# 5. split() - splits the string by the pattern
split_text = re.split(r"\s+", text)  # splits by any whitespace
print("5. split():", split_text)
```

OUTPUT

1. match(): Hello123
2. search(): Python 3
3. findall(): ['123', '3', '9876543210']
4. sub(): Hello***, welcome to Python * programming. Call me at **********.
5. split(): ['Hello123,', 'welcome', 'to', 'Python', '3', 'programming.', 'Call', 'me', 'at', '9876543210.']

Summary:

Function	Purpose
match()	Match at start only
search()	Match anywhere in the string
findall()	Find all matches and return list
sub()	Substitute (replace) pattern
split()	Split string using pattern

5. python program to demonstrate Exception handling using 'try', 'except'. 'finally' and 'else block.

Here is a **complete Python program** that demonstrates **Exception Handling** using:

- try block
- except block
- else block
- finally block

```
def divide_numbers(a, b):
  try:
    result = a / b
  except ZeroDivisionError:
    print("Error: Division by zero is not allowed.")
  else:
    print("Result:", result)
  finally:
    print("Execution of divide_numbers is complete.")

# Main Program
num1 = int(input("Enter numerator: "))
num2 = int(input("Enter denominator: "))

divide_numbers(num1, num2)
```

OUTPUT

Case 1: Valid Input

```
Enter numerator: 10
Enter denominator: 2
Result: 5.0
Execution of divide_numbers is complete.
```

Case 2: Division by Zero

```
Enter numerator: 10
Enter denominator: 0
Error: Division by zero is not allowed.
Execution of divide_numbers is complete.
```

try: Code that may raise an exception.

except: Handles specific exceptions (e.g., ZeroDivisionError).

else: Runs if no exception is raised.

finally: Always executes (cleanup or confirmation).

6. Python program to read a file line by line store it into an array.

File: ***read_lines.py***

```
# Specify the file name (make sure the file exists in the same directory)
filename = "sample.txt"

# Initialize an empty list to store lines
lines_array = []

# Open and read file line by line
with open(filename, 'r') as file:
    for line in file:
        # Remove trailing newline characters and append to list
        lines_array.append(line.strip())

# Print the list of lines
print("Lines in the file:")
for line in lines_array:
    print(line)
```

Sample Content of sample.txt:

```
Hello, this is line 1.
Welcome to Python.
This file has three lines.
```

Output:

```
Lines in the file:
Hello, this is line 1.
Welcome to Python.
This file has three lines.
```

7. Python GUI program to design Student Registration Form using any 5 widgets.

Below is a **self-contained Tkinter program** that builds a simple *Student Registration Form* using **five different widget types**:

1. **Label** – field captions
2. **Entry** – text boxes for data entry
3. **Radiobutton** – gender selection
4. **Checkbutton** – terms & conditions confirmation
5. **Button** – Submit / Reset actions

```
import tkinter as tk

from tkinter import messagebox


def submit_form():

    name = entry_name.get()

    email = entry_email.get()

    gender = gender_var.get()

    course = course_var.get()


    sem_selected = []

    if var1.get():

        sem_selected.append("Ist Semester")

    if var2.get():

        sem_selected.append("IIIrd Semester")

    if var3.get():

        sem_selected.append("Vth Semester")
```

```
    # Convert list to comma separated string
    sem_str = ", ".join(sem_selected) if sem_selected else "None"

    if name == "" or email == "" or gender == "" or course == "":
        messagebox.showwarning("Input Error", "Please fill all fields!")
    else:
        messagebox.showinfo("Registration Successful",
                            f"Name: {name}\nEmail: {email}\nGender:
{gender}\nCourse: {course}\nSemester:{sem_str}")
def reset_form():
    """Reset all input fields"""
    entry_name.delete(0, tk.END)
    entry_email.delete(0, tk.END)
    gender_var.set(" ")     # reset radio button/BLANK
    course_var.set(" ")     # reset dropdown
    var1.set(False)        # uncheck checkbox
    var2.set(False)
    var3.set(False)
# ---------------- Main Window ----------------
root = tk.Tk()
root.title("Student Registration Form")
root.geometry("650x600")

# ---------- Labels and Entries ----------
tk.Label(root, text="Name:").pack(anchor="w", padx=10, pady=5)
entry_name = tk.Entry(root, width=30)
```

```
entry_name.pack(padx=10)


tk.Label(root, text="Email:").pack(anchor="w", padx=10, pady=5)
entry_email = tk.Entry(root, width=30)
entry_email.pack(padx=10)


# ---------- Gender (Radio Buttons) ----------
tk.Label(root, text="Gender:").pack(anchor="w", padx=10, pady=5)
gender_var = tk.StringVar(value=" ")
tk.Radiobutton(root, text="Male", variable=gender_var,
value="Male").pack(anchor="w", padx=20)
tk.Radiobutton(root, text="Female", variable=gender_var,
value="Female").pack(anchor="w", padx=20)


# ---------- Course (Dropdown Menu) ----------
tk.Label(root, text="Course:").pack(anchor="w", padx=10, pady=5)
course_var = tk.StringVar()
course_choices = ["BCA", "BSc", "BCom", "BA"]
course_menu = tk.OptionMenu(root, course_var, *course_choices)
course_menu.pack(anchor="w",padx=10, pady=5)


#--------------CheckBox---------------------------
var1=tk.BooleanVar()
var2=tk.BooleanVar()
var3=tk.BooleanVar()
```

```
cbx1=tk.Checkbutton(root,text="Ist Semester",variable=var1).pack(anchor="w",
padx=10, pady=5)
cbx2=tk.Checkbutton(root,text="IIIrd Semester",variable=var2).pack(anchor="w",
padx=10, pady=5)
cbx3=tk.Checkbutton(root,text="Vth Semester",variable=var3).pack(anchor="w",
padx=10, pady=5)



# ---------- Submit Button ----------
submit_btn = tk.Button(root, text="Submit", command=submit_form, bg="blue",
fg="white",width=20)
submit_btn.pack(pady=20)
reset_btn = tk.Button(root, text="Reset", command=reset_form, bg="red",
fg="white", width=20)
reset_btn.pack(pady=5)


# Run the window
root.mainloop()
```

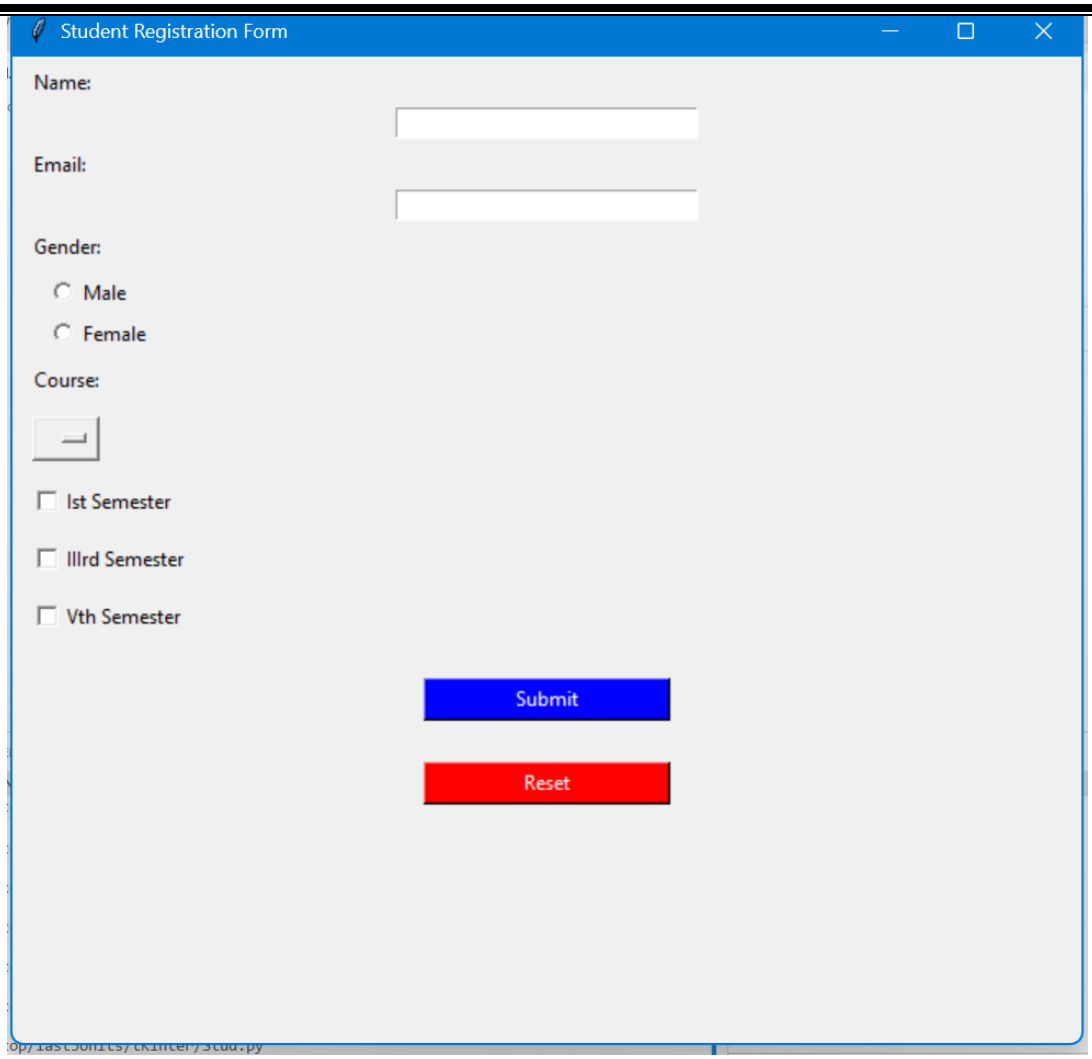
How to run

1. Save the script as student_registration_form.py.
2. Ensure Python's standard library tkinter is available (it is included with most Python installs).
3. Execute:

python student_registration_form.py

A window will open where you can fill in the details, check the terms box, and click **Submit** to see a confirmation dialog,

OUTPUT


Student Registration Form
Name:
Email:
Gender:
Male
Female
Course:
Ist Semester
IIIrd Semester
Vth Semester
Submit
Reset