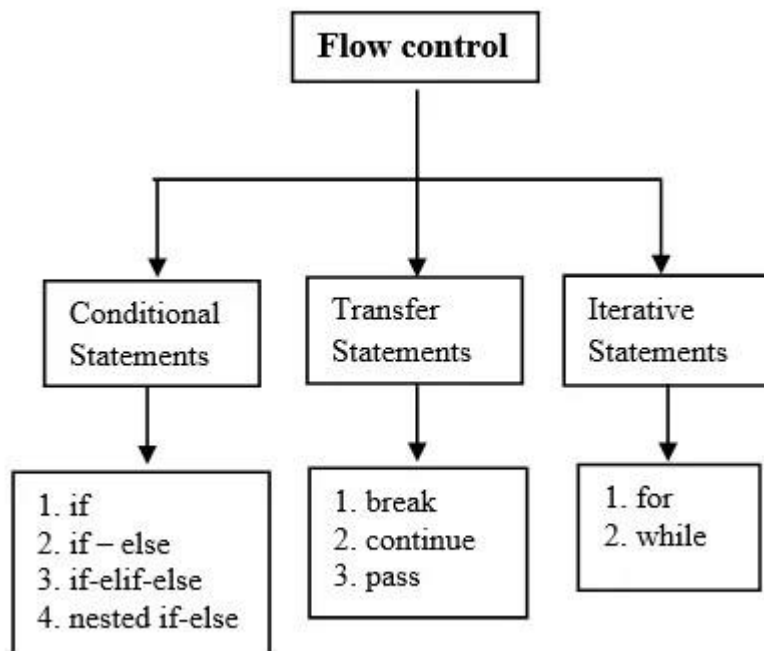
 <b>Marwadi University</b> Marwadi Chandarana Group	NAAC <b>A+</b>	<b>Marwadi University</b> <b>Faculty of Engineering &amp; Technology</b> <b>Department of Information and Communication Technology</b>
<b>Subject: Programming With Python (01CT1309)</b>	<b>Aim:</b> Develop programs to understand the control structures of python.	
<b>Experiment No: 06</b>	<b>Date:</b>	<b>Enrollment No:92400133110</b>

**Aim:** Develop programs to understand the control structures of python.

**IDE:**


**What is a control structure in Python?**

A control structure in Python is a block of code that determines the flow of execution of a program. Control structures enable programmers to create programs that can make decisions based on certain conditions, repeat code blocks multiple times, or execute different code paths based on the values of variables.



There are several types of control structures in Python:

1. **Conditional statements:** These structures allow the program to execute different code blocks based on the value of a condition. The if statement is the most common conditional statement in Python, and it can be accompanied by elif and else statements.
2. **Loops:** These structures allow the program to execute a code block repeatedly based on a condition. Python has two main types of loops: while loops and for loops.

 <b>Marwadi University</b> Marwadi Chandarana Group	<b>Marwadi University</b> <b>Faculty of Engineering &amp; Technology</b> <b>Department of Information and Communication Technology</b>	
<b>Subject: Programming With Python (01CT1309)</b>	<b>Aim:</b> Develop programs to understand the control structures of python.	
<b>Experiment No: 06</b>	<b>Date:</b>	<b>Enrollment No:92400133110</b>

- Exception handling: These structures allow the program to handle errors and exceptions in a controlled manner, preventing the program from crashing. Python has a try-except structure for handling exceptions.
- Function and method definitions: These structures allow the programmer to define reusable blocks of code that can be called from other parts of the program. Functions and methods can take arguments and return values, and they can also contain other control structures.

By using control structures in Python, programmers can create more complex and powerful programs that can make decisions, repeat actions, and handle errors in a controlled way.

### 1. Conditional Statements (if, else, elif)

Conditional statements are fundamental to programming and allow us to make decisions based on specific conditions. In Python, we use the keywords if, else, and elif to implement conditional branching. The syntax is clean and straightforward, making Python code highly readable.

#### 1. if Statement

The if statement is used to execute a block of code if a given condition is True. If the condition is False, the code inside the if block is skipped.



Example:

```
x = 10
if x > 5:
    print("x is greater than 5")
```

Output

#### 2. elif Statement

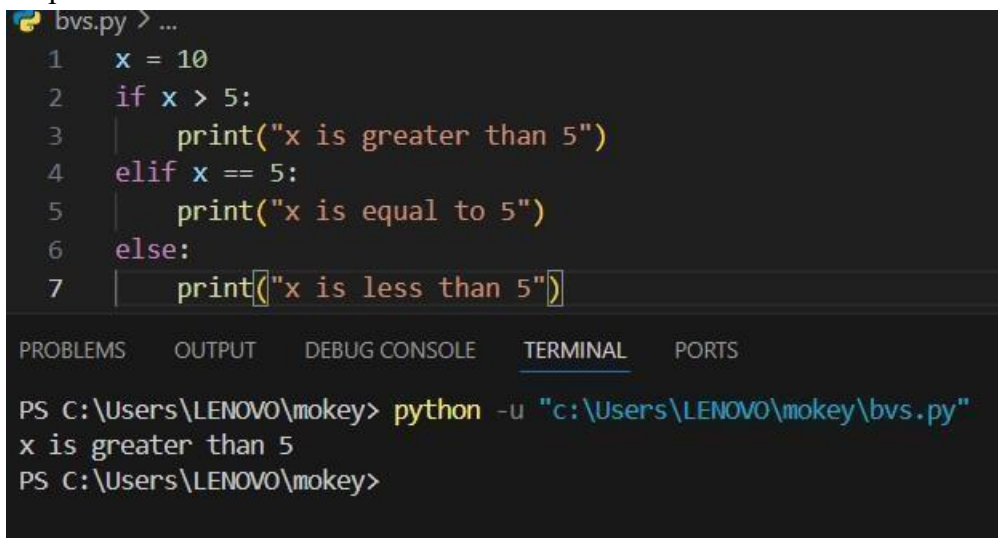
The elif statement is used when you have multiple conditions to check. It comes after an if statement and before an optional else statement. If the initial if condition is False, Python evaluates the elif condition. You can have multiple elif conditions.

 <b>Marwadi University</b> Marwadi Chandarana Group 	<b>Marwadi University</b> <b>Faculty of Engineering &amp; Technology</b> <b>Department of Information and Communication Technology</b>	
<b>Subject: Programming With Python (01CT1309)</b>	<b>Aim:</b> Develop programs to understand the control structures of python.	
<b>Experiment No: 06</b>	<b>Date:</b>	<b>Enrollment No:92400133110</b>

Example:

```
x = 10 if x>5    print("x is
greater than 5") elif x ==5:
    print("x is equal to 5") else:
    print("x is less than 5")
```

Output



```
bvs.py > ...
1  x = 10
2  if x > 5:
3      print("x is greater than 5")
4  elif x == 5:
5      print("x is equal to 5")
6  else:
7      print("x is less than 5")

PROBLEMS  OUTPUT  DEBUG CONSOLE  TERMINAL  PORTS



PS C:\Users\LENOVO\mokey> python -u "c:\Users\LENOVO\mokey\bvs.py"
x is greater than 5
PS C:\Users\LENOVO\mokey>
```

### 3. else Statement

The else statement is used to execute a block of code when the conditions specified in the if and elif statements are not met.

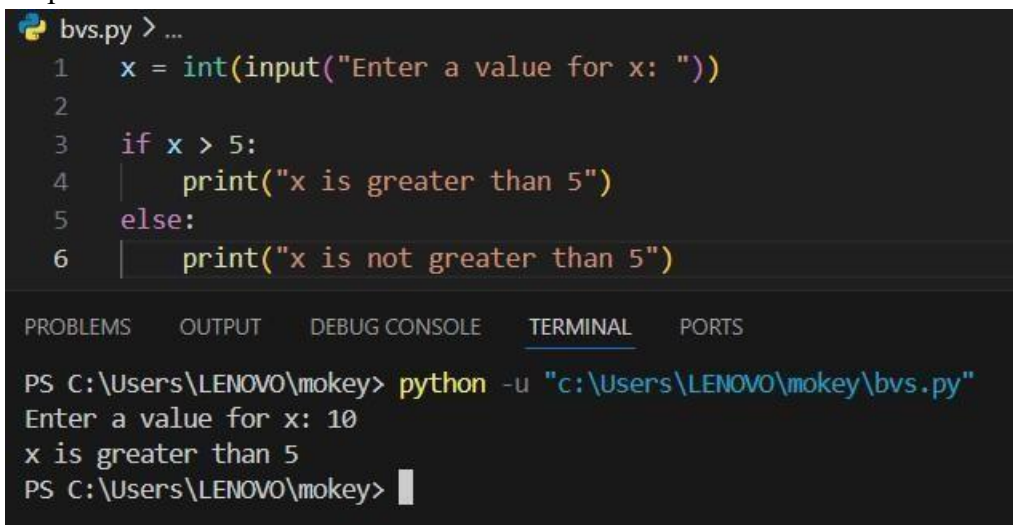
Example

```
if x>5:
```

 <b>Marwadi University</b> Marwadi Chandarana Group 	<b>Marwadi University</b> <b>Faculty of Engineering &amp; Technology</b> <b>Department of Information and Communication Technology</b>	
<b>Subject: Programming With Python (01CT1309)</b>	<b>Aim:</b> Develop programs to understand the control structures of python.	
<b>Experiment No: 06</b>	<b>Date:</b>	<b>Enrollment No:92400133110</b>

```
print("x is greater than 5") else:
print("x is not greater than 5")
```

### Output



```
bvs.py > ...
1  x = int(input("Enter a value for x: "))
2
3  if x > 5:
4      print("x is greater than 5")
5  else:
6      print("x is not greater than 5")

PROBLEMS  OUTPUT  DEBUG CONSOLE  TERMINAL  PORTS

PS C:\Users\LENOVO\mokey> python -u "c:\Users\LENOVO\mokey\bvs.py"
Enter a value for x: 10
x is greater than 5
PS C:\Users\LENOVO\mokey> |
```

### Nested if-else statements



Nested if-else statements in Python allow you to test multiple conditions and execute different code blocks based on the results of these tests.

### Example

```
age = 35
```

```
if age >= 60:
    print("You are a senior citizen.") else:
    if age >= 18:
        print("You are an adult.")
    else:
        print("You are a teenager.")
```

### Output

 <b>Marwadi University</b> Marwadi Chandarana Group 	<b>Marwadi University</b> <b>Faculty of Engineering &amp; Technology</b> <b>Department of Information and Communication Technology</b>	
<b>Subject: Programming With Python (01CT1309)</b>	<b>Aim:</b> Develop programs to understand the control structures of python.	
<b>Experiment No: 06</b>	<b>Date:</b>	<b>Enrollment No:92400133110</b>

```

bvs.py > ...
1  age = 35
2
3  if age >= 60:
4      print("You are a senior citizen.")
5  else:
6      if age >= 18:
7          print("You are an adult.")
8      else:
9          print("You are a teenager.")
10
PROBLEMS  OUTPUT  DEBUG CONSOLE  TERMINAL  PORTS

PS C:\Users\LENOVO\mokey> python -u "c:\Users\LENOVO\mokey\bvs.py"
You are an adult.
PS C:\Users\LENOVO\mokey>

```



Example num  
= 10

```

if num > 0:    if num % 2 == 0:
print("The number is positive and even.")
else:        print("The number is positive but
odd.") else:    print("The number is not
positive.")

```

Output

 <b>Marwadi University</b> Marwadi Chandarana Group 	<b>Marwadi University</b> <b>Faculty of Engineering &amp; Technology</b> <b>Department of Information and Communication Technology</b>	
<b>Subject: Programming With Python (01CT1309)</b>	<b>Aim:</b> Develop programs to understand the control structures of python.	
<b>Experiment No: 06</b>	<b>Date:</b>	<b>Enrollment No:92400133110</b>

```

bvs.py > ...
1  num = 10
2  if num > 0:
3      if num % 2 == 0:
4          print("The number is positive and even.")
5      else:
6          print("The number is positive but odd.")
7  else:
8      print("The number is not positive.")

```

PROBLEMS   OUTPUT   DEBUG CONSOLE   TERMINAL   PORTS

```

PS C:\Users\LENOVO\mokey> python -u "c:\Users\LENOVO\mokey\bvs.py"
The number is positive and even.
PS C:\Users\LENOVO\mokey>

```

## Looping Statements

### 1. for Loop

The for loop is used to iterate over sequences like lists, tuples, strings, and dictionaries. It allows you to perform an action for each item in the sequence.



### Example

```

Fruits = ["apple", "banana", "cherry"]
for fruit in fruits:
    print(fruit)

```

### Output

 <b>Marwadi University</b> Marwadi Chandarana Group 	<b>Marwadi University</b> <b>Faculty of Engineering &amp; Technology</b> <b>Department of Information and Communication Technology</b>	
<b>Subject: Programming With Python (01CT1309)</b>	<b>Aim:</b> Develop programs to understand the control structures of python.	
<b>Experiment No: 06</b>	<b>Date:</b>	<b>Enrollment No:92400133110</b>

```
bvs.py / ...
1  Fruits = ["apple", "banana", "cherry"]
2  for fruit in Fruits:
3      print(fruit)
4
```

PROBLEMS   OUTPUT   DEBUG CONSOLE   TERMINAL   PORTS

```
PS C:\Users\LENOVO\mokey> python -u "c:\Users\LENOVO\mokey\bvs.py"
apple
banana
cherry
PS C:\Users\LENOVO\mokey>
```

## 2. while Loop

The while loop is used to repeatedly execute a block of code as long as a specified condition is True.

Example

```
x = 1
while x<=5:
    print(x)
    x+=1
```



```
1  x = 1
2  while x<=5:
3      print(x)
4      x+=1
5
```

PROBLEMS   OUTPUT   DEBUG CONSOLE   TERMINAL   PORTS

```
PS C:\Users\LENOVO\mokey> python -u "c:\Users\LENOVO\mokey\bvs.py"
1
2
3
4
5
PS C:\Users\LENOVO\mokey>
```

Loop Control Statements

Python provides several loop control statements to enhance the functionality of loops.

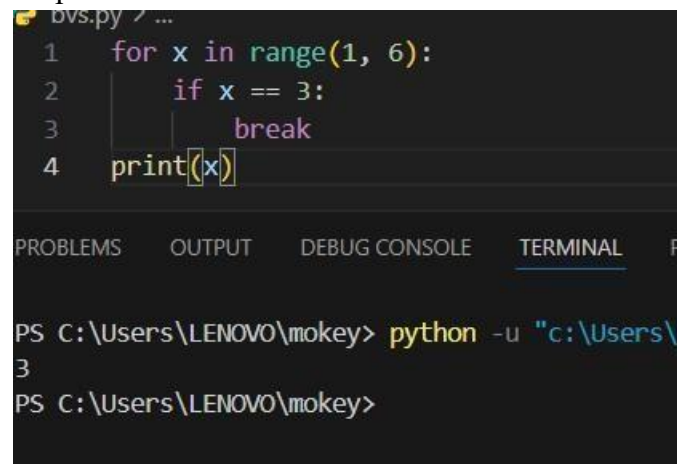
 <b>Marwadi University</b> Marwadi Chandarana Group 	<b>Marwadi University</b> <b>Faculty of Engineering &amp; Technology</b> <b>Department of Information and Communication Technology</b>	
<b>Subject: Programming With Python (01CT1309)</b>	<b>Aim:</b> Develop programs to understand the control structures of python.	
<b>Experiment No: 06</b>	<b>Date:</b>	<b>Enrollment No:92400133110</b>

break: The break statement is used inside a loop (while loop or for loop). When the condition specified in the if statement is true, the break statement is executed, and the control is transferred to the next statement after the loop. This means that the loop is terminated, and the code execution continues from the statement after the loop.

#### Example

```
for x in range(1,6):
    if x==3:
        break
    print(x)
```

#### Output



```
bvs.py 7 ...
1  for x in range(1, 6):
2      if x == 3:
3          break
4  print(x)
```

PROBLEMS   OUTPUT   DEBUG CONSOLE   TERMINAL   P

```
PS C:\Users\LENOVO\mokey> python -u "c:\Users\
3
PS C:\Users\LENOVO\mokey>
```



continue: The continue statement is used to skip a particular iteration of a loop when a specific condition is met. When a continue statement is executed inside a loop, it skips the current iteration of the loop and jumps to the next iteration.

#### Example for x in

```
range(1,6):
    if x==3:
        continue
    print(x)
```

#### Output



 <b>Marwadi University</b> Marwadi Chandarana Group 	<b>Marwadi University</b> <b>Faculty of Engineering &amp; Technology</b> <b>Department of Information and Communication Technology</b>	
<b>Subject: Programming With Python (01CT1309)</b>	<b>Aim:</b> Develop programs to understand the control structures of python.	
<b>Experiment No: 06</b>	<b>Date:</b>	<b>Enrollment No:92400133110</b>

```

bvs.py > ...
1  for x in range(1,6):
2      if x==3:
3          continue
4  print(x)
5
PROBLEMS  OUTPUT  DEBUG CONSOLE  TERMINAL  PORTS

PS C:\Users\LENOVO\mokey> python -u "c:\Users\LENOVO\mokey
5
PS C:\Users\LENOVO\mokey>

```

pass: The pass statement is a placeholder in Python. It doesn't do anything but is used when a statement is syntactically required. It is often used as a placeholder for functions, loops, or conditional blocks that will be implemented later.

#### Example

```

for x in range(1,6):
if x == 3:
pass    print(x)

```

#### Output

```



bvs.py > ...
1  for x in range(1,6):
2      if x == 3:
3          pass
4      print(x)
5
PROBLEMS  OUTPUT  DEBUG CONSOLE  TERMINAL  PORTS

PS C:\Users\LENOVO\mokey> python -u "c:\Users\LENOVO\mok
1
2
3
4
5
PS C:\Users\LENOVO\mokey>

```

### Try and Except Statement – Catching Exceptions

- In Python, you may catch and deal with exceptions by using the try and except commands.
- The try and except clauses are used to contain statements that can raise exceptions and statements that handle such exceptions.

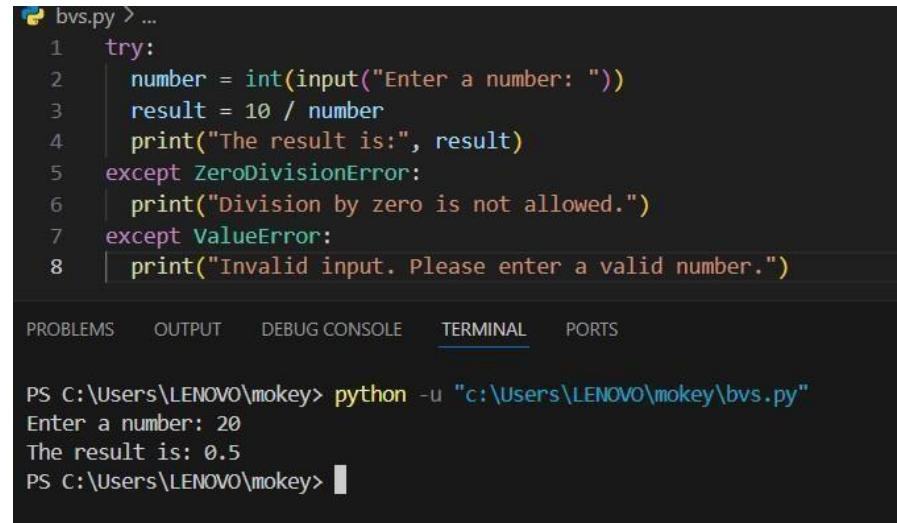
 <b>Marwadi University</b> Marwadi Chandarana Group 	<b>Marwadi University</b> <b>Faculty of Engineering &amp; Technology</b> <b>Department of Information and Communication Technology</b>	
<b>Subject: Programming With Python (01CT1309)</b>	<b>Aim:</b> Develop programs to understand the control structures of python.	
<b>Experiment No: 06</b>	<b>Date:</b>	<b>Enrollment No:92400133110</b>

Example try:

```

number = int(input("Enter a number: "))
result = 10 / number
print("The result is:", result)
except ZeroDivisionError:
    print("Division by zero is not allowed.")
except ValueError:
    print("Invalid input. Please enter a valid number.")

```



The screenshot shows a code editor with a dark theme. The code is as follows:

```

1 try:
2     number = int(input("Enter a number: "))
3     result = 10 / number
4     print("The result is:", result)
5 except ZeroDivisionError:
6     print("Division by zero is not allowed.")
7 except ValueError:
8     print("Invalid input. Please enter a valid number.")

```

Below the code editor, there is a terminal window. It shows the command to run the script:

```

PS C:\Users\LENOVO\mokey> python -u "c:\Users\LENOVO\mokey\bvs.py"
Enter a number: 20
The result is: 0.5
PS C:\Users\LENOVO\mokey>



```

### Python Function:

Python functions are reusable code blocks that carry out particular tasks, helping programmers structure their code and make it easier to read. By preventing duplication, functions make the code more modular and manageable. The 'def' keyword, the function name, and any parameters included in parenthesis define a function. The code to be performed is contained in the function body, and the 'return' statement allows the function to produce a result.

### Types of Functions in Python

Python supports various types of functions, each serving different purposes in programming. Here are the main types of functions in Python, along with examples:

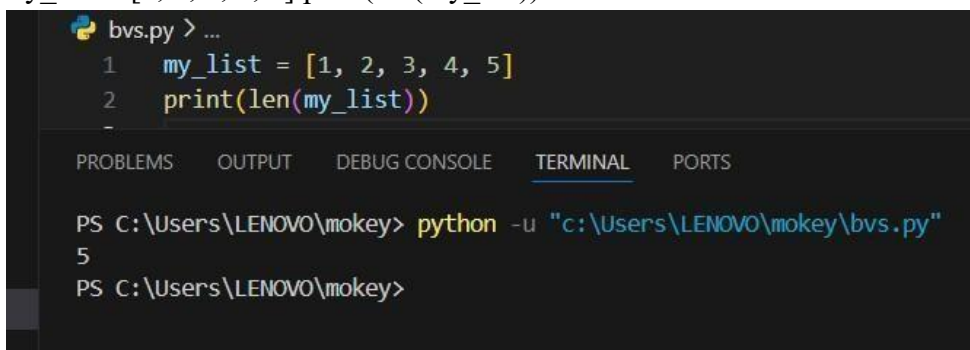
 <b>Marwadi University</b> Marwadi Chandarana Group 	<b>Marwadi University</b> <b>Faculty of Engineering &amp; Technology</b> <b>Department of Information and Communication Technology</b>	
<b>Subject: Programming With Python (01CT1309)</b>	<b>Aim:</b> Develop programs to understand the control structures of python.	
<b>Experiment No: 06</b>	<b>Date:</b>	<b>Enrollment No:92400133110</b>

## 1. Built-in Functions

These functions are pre-defined in Python and can be used directly without any further declaration.

Example

```
my_list = [1, 2, 3, 4, 5] print(len(my_list))
```



```
bvs.py > ...
1 my_list = [1, 2, 3, 4, 5]
2 print(len(my_list))
-
PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS

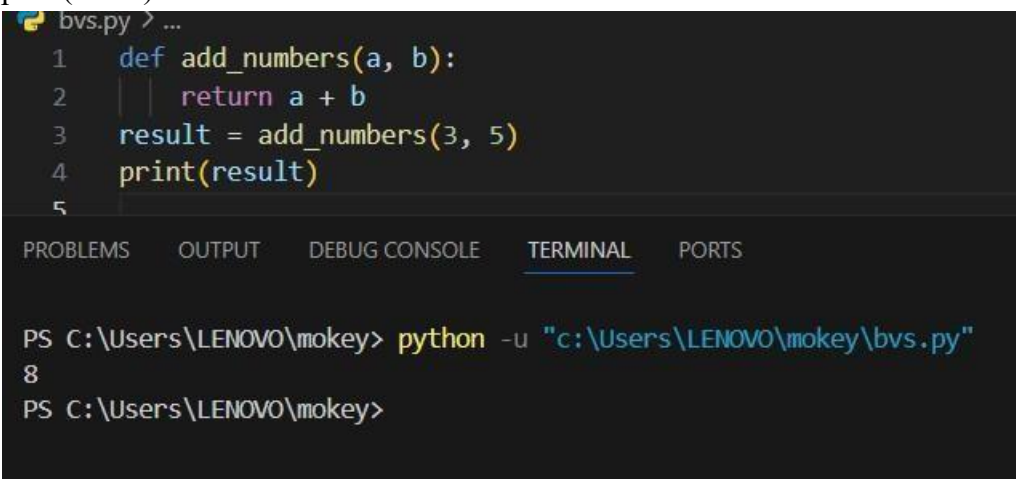
PS C:\Users\LENOVO\mokey> python -u "c:\Users\LENOVO\mokey\bvs.py"
5
PS C:\Users\LENOVO\mokey>
```

## 2. User-defined Functions

These are functions that users create to perform specific tasks.

Example



```
def
add_numbers(a, b):
    return a + b result =
add_numbers(3, 5)
print(result)
```



```
bvs.py > ...
1 def add_numbers(a, b):
2     return a + b
3 result = add_numbers(3, 5)
4 print(result)
5
PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS

PS C:\Users\LENOVO\mokey> python -u "c:\Users\LENOVO\mokey\bvs.py"
8
PS C:\Users\LENOVO\mokey>
```

## 3. Anonymous Functions (Lambda Functions)

 <b>Marwadi University</b> Marwadi Chandarana Group 	<b>Marwadi University</b> <b>Faculty of Engineering &amp; Technology</b> <b>Department of Information and Communication Technology</b>	
<b>Subject: Programming With Python (01CT1309)</b>	<b>Aim:</b> Develop programs to understand the control structures of python.	
<b>Experiment No: 06</b>	<b>Date:</b>	<b>Enrollment No:92400133110</b>

These are small, unnamed functions defined using the lambda keyword. They are typically used for short, simple operations.

Example

```
add = lambda x, y: x + y
print(add(3, 5))
```

Output



```
bvs.py > ...
1  add = lambda x, y: x + y
2  print(add(3, 5))

PROBLEMS  OUTPUT  DEBUG CONSOLE  TERMINAL  PORTS

PS C:\Users\LENOVO\mokey> python -u "c:\Users\LENOVO\mokey\bvs.py"
8
PS C:\Users\LENOVO\mokey>
```

#### 4. Recursive Functions



These are functions that call themselves within their definition. They help solve problems that can be broken down into smaller, similar problems.

Example

```
def factorial(n):
```

```
    if n == 1:
    return 1    else:
        return n * factorial(n - 1)
print(factorial(5))
```

Output

 <b>Marwadi University</b> Marwadi Chandarana Group 	<b>Marwadi University</b> <b>Faculty of Engineering &amp; Technology</b> <b>Department of Information and Communication Technology</b>	
<b>Subject: Programming With Python (01CT1309)</b>	<b>Aim:</b> Develop programs to understand the control structures of python.	
<b>Experiment No: 06</b>	<b>Date:</b>	<b>Enrollment No:92400133110</b>

```

bvs.py 7 ...
1  def factorial(n):
2      if n == 1:
3          return 1
4      else:
5          return n * factorial(n - 1)
6  print(factorial(5))
7
PROBLEMS  OUTPUT  DEBUG CONSOLE  TERMINAL  PORTS

PS C:\Users\LENOVO\mokey> python -u "c:\Users\LENOVO\mokey\bvs.py"
120
PS C:\Users\LENOVO\mokey>

```

## 5. Higher-Order Functions

These functions can take other functions as arguments or return them as results. Examples include map(), filter(), and reduce().

### Example

```

def square(x): return x
* x numbers = [1, 2, 3,
4, 5]
squared_numbers = list(map(square, numbers)) print(squared_numbers)

```

```

1  def square(x):
2      return x * x
3  numbers = [1, 2, 3, 4, 5]
4  squared_numbers = list(map(square, numbers))
5  print(squared_numbers)
PROBLEMS  OUTPUT  DEBUG CONSOLE  TERMINAL  PORTS

PS C:\Users\LENOVO\mokey> python -u "c:\Users\LENOVO\mokey\bvs.py"
[1, 4, 9, 16, 25]
PS C:\Users\LENOVO\mokey>

```



## 6. Generator Functions

These functions yield values one at a time and can produce a sequence of values over time, using the yield keyword. Example def generate\_numbers(): for i in range(1, 6):

```

yield i for number in
generate_numbers():
print(number)

```

 <b>Marwadi University</b> Marwadi Chandarana Group 	<b>Marwadi University</b> <b>Faculty of Engineering &amp; Technology</b> <b>Department of Information and Communication Technology</b>	
<b>Subject: Programming With Python (01CT1309)</b>	<b>Aim:</b> Develop programs to understand the control structures of python.	
<b>Experiment No: 06</b>	<b>Date:</b>	<b>Enrollment No:92400133110</b>

```

bvs.py > ...
1 def generate_numbers():
2     for i in range(1, 6):
3         yield i
4 for number in generate_numbers():
5     print(number)

```

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS



```

PS C:\Users\LENOVO\mokey> python -u "c:\Users\LENOVO\mokey\bvs.py"
1
2
3
4
5
PS C:\Users\LENOVO\mokey>

```

**Post Lab Exercise:**

- Write a Python program to print all odd numbers between 1 to 100 using a while loop.

 <b>Marwadi University</b> Marwadi Chandarana Group 	<b>Marwadi University</b> <b>Faculty of Engineering &amp; Technology</b> <b>Department of Information and Communication Technology</b>	
<b>Subject: Programming With Python (01CT1309)</b>	<b>Aim:</b> Develop programs to understand the control structures of python.	
<b>Experiment No: 06</b>	<b>Date:</b>	<b>Enrollment No:92400133110</b>

```

1  num = 1
2  while num <= 100:
3      if num % 2 != 0:
4          print(num)
5          num += 1

```

PROBLEMS   OUTPUT   DEBUG CONSOLE   TERMINAL   PORTS



```

39
41
43
45
47
49
51
53
55
57
59
61
63
65
67
69
71
73
75
77
79
81
83
85
87
89
91
93
95
97
99
PS C:\Users\LENOVO\mokey>

```

b. Write a Python program to find the sum of all natural numbers between 1 to n.



 <b>Marwadi University</b> Marwadi Chandarana Group 	<b>Marwadi University</b> <b>Faculty of Engineering &amp; Technology</b> <b>Department of Information and Communication Technology</b>	
<b>Subject: Programming With Python (01CT1309)</b>	<b>Aim:</b> Develop programs to understand the control structures of python.	
<b>Experiment No: 06</b>	<b>Date:</b>	<b>Enrollment No:92400133110</b>

```

bvs.py > ...
1  n = int(input("Enter a number: "))
2  sum = 0
3  for i in range(1, n + 1):
4      sum += i
5  print("Sum of natural numbers from 1 to", n, "is:", sum)

```

PROBLEMS   OUTPUT   DEBUG CONSOLE   TERMINAL   PORTS

```

PS C:\Users\LENOVO\mokey> python -u "c:\Users\LENOVO\mokey\bvs.py"
Enter a number: 20
Sum of natural numbers from 1 to 20 is: 210
PS C:\Users\LENOVO\mokey>

```

c. Write a Python function program to count a number of digits in a number.

```

bvs.py > count_digits
1  def count_digits(num):
2      count = 0
3      num = abs(num)
4      if num == 0:
5          return 1
6      while num > 0:
7          count += 1
8          num //= 10
9      return count
10 number = int(input("Enter a number: "))
11 print("Number of digits:", count_digits(number))

```

PROBLEMS   OUTPUT   DEBUG CONSOLE   TERMINAL   PORTS



```

PS C:\Users\LENOVO\mokey> python -u "c:\Users\LENOVO\mokey\bvs.py"
Enter a number: 30
Number of digits: 2
PS C:\Users\LENOVO\mokey>

```

d. Write a Python program to find the first and last digits of a number.



 <b>Marwadi University</b> Marwadi Chandarana Group 	<b>Marwadi University</b> <b>Faculty of Engineering &amp; Technology</b> <b>Department of Information and Communication Technology</b>	
<b>Subject: Programming With Python (01CT1309)</b>	<b>Aim:</b> Develop programs to understand the control structures of python.	
<b>Experiment No: 06</b>	<b>Date:</b>	<b>Enrollment No:92400133110</b>

```
bvs.py > find_first_last_digits
1 def find_first_last_digits(num):
2     num = abs(num)
3     last_digit = num % 10
4     first_digit = num
5     while first_digit >= 10:
6         first_digit //= 10
7     return first_digit, last_digit
8 number = int(input("Enter an integer: "))
9 first, last = find_first_last_digits(number)
10 print(f"First digit: {first}")
11 print(f"Last digit: {last}")
```

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS

```
PS C:\Users\LENOVO\mokey> python -u "c:\Users\LENOVO\mokey\bv
Enter an integer: 20
First digit: 2
Last digit: 0
PS C:\Users\LENOVO\mokey> |
```



e. Write a Python program to swap the first and last digits of a number.

```
1 def swap_first_last_digits(num):
2     num_str = str(num)
3     if len(num_str) == 1:
4         return num
5     swapped = num_str[-1] + num_str[1:-1] + num_str[0]
6     return int(swapped)
7 number = int(input("Enter a number: "))
8 swapped_number = swap_first_last_digits(number)
9 print(f"Number after swapping first and last digits: {swapped_number}")
```

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS

```
PS C:\Users\LENOVO\mokey> python -u "c:\Users\LENOVO\mokey\bvs.py"
Enter a number: 20
Number after swapping first and last digits: 2
PS C:\Users\LENOVO\mokey> |
```

f. Write a Python program to calculate the product of digits of a number.

 <b>Marwadi University</b> Marwadi Chandarana Group 	<b>Marwadi University</b> <b>Faculty of Engineering &amp; Technology</b> <b>Department of Information and Communication Technology</b>	
<b>Subject: Programming With Python (01CT1309)</b>	<b>Aim:</b> Develop programs to understand the control structures of python.	
<b>Experiment No: 06</b>	<b>Date:</b>	<b>Enrollment No:92400133110</b>

```

bvs.py > product_of_digits
1  def product_of_digits(number):
2      number = abs(number)
3      product = 1
4      if number == 0:
5          return 0
6      while number > 0:
7          digit = number % 10
8          product *= digit
9          number //= 10
10     return product
11 try:
12     num = int(input("Enter an integer: "))
13     result = product_of_digits(num)
14     print(f"The product of digits of {num} is {result}")
15 except ValueError:
16     print("Invalid input! Please enter a valid integer.")

```

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS

```

PS C:\Users\LENOVO\mokey> python -u "c:\Users\LENOVO\mokey\bvs.py"
Enter an integer: 20
The product of digits of 20 is 0
PS C:\Users\LENOVO\mokey>

```

g. Write a Python program to enter a number and print its reverse

```

1  def reverse_number(num):
2      if num < 0:
3          reversed_num = -int(str(-num)[::-1])
4      else:
5          reversed_num = int(str(num)[::-1])
6      return reversed_num
7  try:
8      number = int(input("Enter a number: "))
9      reversed_result = reverse_number(number)
10     print("Reversed number:", reversed_result)
11 except ValueError:
12     print("Invalid input! Please enter a valid integer.")

```

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS

```

PS C:\Users\LENOVO\mokey> python -u "c:\Users\LENOVO\mokey\bvs.py"
Enter a number: 20
Reversed number: 2
PS C:\Users\LENOVO\mokey>

```

**Subject: Programming With  
Python (01CT1309)**

**Aim:** Develop programs to understand the control structures of python.

**Experiment No: 06**

**Date:**

**Enrollment No:92400133110**