PROGRAMMING IN PYTHON - LAB Manual

Ex:1(a) Write a python program to find largest of two numbers using If-Else.

Aim:

To write a Python program to find a largest of two numbers using If-else statement.

Procedure:

Step 1: Get the 2 inputs as a integer from the user.

Step 2: check whether the first value is greater than the second value.

Step 3: If yes, print the first value is greater.

Step 4: Else, print the second value is greater.

Step 5: End.

Program:

```
a = int(input("Enter the first Number: "))
b = int(input("Enter the second Number: "))

if a > b:
   print(a, "is greater")
else:
   print(b, "is greater")
```

OUTPUT:

```
Enter the first Number: 5
Enter the second Number: 7
7 is greater
```

EX: 1(b) Write a Python program to display the Multiplication Table using For loop.

Aim :

To write a Python program to display the multiplication table using for loop.

Procedure :

Prompt the user to enter a number.

Read and store the input number.

Iterate from i = 1 to 10 using a for loop.

Within each iteration:

Multiply the input number with the current value of i.

Print the equation num x i = result, where num is the input number, i is the current iteration value, and result is the product of num and i.

Repeat steps 5-6 for each iteration of the loop.

End of the program

Program:

```
num = int(input("Enter any Number: "))
for i in range(1, 11):
  print(num, 'x', i, '=', num*i)
```

OUTPUT:

Enter any Number: 12

12 x 1 = 12

12 x 2 = 24

12 x 3 = 36

12 x 4 = 48

12 x 5 = 60

12 x 6 = 72

12 x 7 = 84

12 x 8 = 96

12 x 9 = 108

12 x 10 = 120

EX: 1(c) write a python program to find given number is Odd or Even using function

Aim:

To write a python program to find a given number is odd or even by using function.

Procedure:

Step 1: Start the function evenOdd with a parameter x.

Step 2: call the function evenOdd () with the desired input 123 and 30 as a aruguments

Step 2: Check if x is divisible evenly by 2, which means checking if x % 2 == 0.

Step3 : If the condition is true, print "even" to indicate that x is an even number.

Step 4: If the condition is false, meaning x is not divisible evenly by 2, print "odd" to indicate that x is an odd number.

Step 5: End the function.

Program:

```
# Function Definition & Function call
def evenOdd(x):
    if (x % 2 == 0):
        print("even")
    else:
        print("odd")

# Driver code to call the function
evenOdd(123)
evenOdd(30)
```

Output:

odd

even

Ex - 1 (d) : Write a python program to demonstrate String Manipulation

Aim:

To write a python program to demonstrate String manipulation operations.

Procedure:

1. Declare and initialize a string variable `str1` with the value "Python Programming".
2. Declare and initialize another string variable `str2` with the value "hello".
3. Print the character at index 0 of `str1` using the expression `str1[0]`.
4. Print the substring of `str1` from index 1 to 4 (excluding index 5) using the expression `str1[1:5]`.
5. Concatenate the string `str1` with the string " Lab" using the `+` operator and print the result.
6. Capitalize the string `str2` using the `capitalize()` method and print the result.
7. Count the number of occurrences of the letter 'l' in `str2` using the `count()` method and print the result.
8. Declare and initialize a string variable `str3` with the value "this0123".
9. Check if `str3` contains alphanumeric characters only using the `isalnum()` method and print the result.
10. Check if `str3` contains numeric characters only using the `isnumeric()` method and print the result.
11. Check if `str2` contains alphabetic characters only using the `isalpha()` method and print the result.
12. Convert `str1` to uppercase using the `upper()` method and print the result.

Program :

```
## String Manipulation
str1 = 'Python Programming'
str2 = "hello"
print("str1[0]:",str1[0])
print("str1[1:5]:",str1[1:5])
print(str1+" Lab")
print(str2.capitalize())
print(str2.count('l'))
str3 = "this0123"
print(str3.isalnum())
print(str3.isnumeric())
print(str2.isalpha())
print(str1.upper())
```

OUTPUT:

str1[0]: P

str1[1:5]: ytho

Python Programming Lab

Hello

2

True

False

True

PYTHON PROGRAMMING

Ex: 2(a) Tuple Manipulation and Analysis in Python

Aim:

To Perform operations and analysis on a tuple in Python.

Procedure:

1. Initialize a tuple `a` with elements: 1, 2.5, 3, 4, 5, 7, 10, True, and "Ram".
2. Print the tuple `a` using the `print()` function.
3. Print the type of `a` using the `type()` function.
4. Access the second element of `a` using `a[1]` and print it.
5. Access the last element of `a` using `a[-1]` and print it.
6. Access a slice of the first two elements of `a` using `a[0:2]` and print it.
7. Print the length of `a` using the `len()` function.
8. Print the minimum value in `a` using the `min()` function.
9. Print the maximum value in `a` using the `max()` function.
10. Iterate over each element of `a` using a for loop and print each element.
11. Check if the string "python" is present in `a` using the `in` keyword and print the corresponding message.

Program:

```
a =  (1, 2.5, 3, 4, 5, 7, 10, True, "Ram")
print(a)
print(type(a))
print(a[1])
print(a[-1])
```

```
print(a[0:2])
print(len(a))
print(min(a))
print(max(a))
for i in a:
    print(i)
if "Raj" in a:
    print("Raja is Found")
else:
    print("Not Found")
```

OUTPUT:

```
(1, 2.5, 3, 4, 5, 7, 10, True, "Ram")
<class 'tuple'>
2.5
Ram
(1, 2.5)
9
1
Ram
1
2.5
3
4
5
7
10
True
Ram
Not Found
```

Ex: 2(b) Manipulating Lists in Python: Append, Extend, Insert, Remove, and Pop Operations"

Aim:

To demonstrate various list manipulation operations in Python.

Procedure:

1. Create a list `a` with elements `[1, 2, 3, 4, 5]`.
2. Print the list `a` and its type.
3. Modify the first element of `a` to `100`.
4. Print the modified list `a`.
5. Create a list `Mylist` with elements `[10, 25, 36, 45, 100, 20]`.
6. Append the element `200` to the end of `Mylist`.
7. Print the updated `Mylist` after appending `200`.
8. Extend the list `Mylist` with elements from `another_list` (which is `[30, 40, 50]`).
9. Print the updated `Mylist` after extending it with elements from `another_list`.
10. Insert the element `15` at index `1` in `Mylist`.
11. Print the updated `Mylist` after inserting `15`.
12. Remove the first occurrence of the element `36` from `Mylist`.
13. Print the updated `Mylist` after removing `36`.
14. Pop and return the element at index `4` (which is `100`) from `Mylist`.
15. Print the popped element and the updated `Mylist` after popping `100`.

PROGRAM:

```
# List in Python
a = [1, 2, 3, 4, 5]
print(a)
print(type(a))
a[0] = 100 #Modify the elements in list
print(a)
print("----------------------------")
Mylist = [10, 25, 36, 45, 100, 20]
# Append an element to the end of the list
Mylist.append(200)
print(Mylist)
```

```
# Extend the list with elements from another iterable (e.g., list, tuple, or string)
another_list = [30, 40, 50]
Mylist.extend(another_list)
print(Mylist)

# Insert an element at a specific index
Mylist.insert(1, 15)
print(Mylist)
# Remove an element from the list (first occurrence of the specified value)
Mylist.remove(36)
print(Mylist)

# Pop an element from the list (remove and return an element based on its index)
popped_element = Mylist.pop(4)
print(popped_element)
print(Mylist)
```

OUTPUT:

```
[1, 2, 3, 4, 5]
<class 'list'>
[100, 2, 3, 4, 5]
------------------------------
[10, 25, 36, 45, 100, 20, 200]
[10, 25, 36, 45, 100, 20, 200, 30, 40, 50]
[10, 15, 25, 36, 45, 100, 20, 200, 30, 40, 50]
[10, 15, 25, 45, 100, 20, 200, 30, 40, 50]
100
[10, 15, 25, 45, 20, 200, 30, 40, 50]
```

Result: The Python program successfully demonstrates the different list manipulation operations.

Ex: 3 Set Operations in Python

AIM: To demonstrate various set operations and methods in Python.

Procedure:

1. Create a set `names` with elements `'Guava'`, `'Mango'`, and `'Banana'`.

2. Print the set `names`.

3. Print the type of `names`, which confirms it's a set.

4. Use a `for` loop to access and print each element of the set `names`.

5. Add the new element `'Jack Fruit'` to the set `names` using the `add()` method and print the updated set ‘ names'.

6. Create another set `a` with elements `'Apple'`, `'Orange'`, and `'Grapes'`.

7. Update the set `names` with the elements from set `a` using the `update()` method and print the updated set ‘ names ‘.

9. Remove the element `'Jack Fruit'` from the set `names` using the `remove()` method and Print the updated set `names`.

10. Attempt to remove the element `'Jack Fruit'` again using the `discard()` method, but since it's already removed, it does nothing and print the updated set ‘names’.

11. Create two sets `a` and `b` with elements `{1, 2, 3, 4}` and `{'a', 'b', 'c', 'd'}`, respectively.

12. Use the `union()` method to create a new set `c` that contains all elements from sets `a` and `b` and Print the set `c`.

13. Reassign `a` and `b` with new sets: `a = {1, 2, 3, 4, 5}` and `b = {5, 6, 7, 8, 9}`.

14. Use the `intersection()` method to create a new set `c` that contains the common elements between sets `a` and `b` and print the set c.

15. Use the `symmetric_difference()` method to create a new set `c` that contains elements that are in either set `a` or set `b`, but not in both.

16. Reassign `a` and `b` with new sets: `a = {5, 6, 7}` and `b = {5, 6, 7}`.

17. Use the `isdisjoint()` method to check if there are any common elements between sets `a` and `b`.

18. Use the `issubset()` method to check if all elements of set `a` are present in set `b`.

19. Use the `issuperset()` method to check if all elements of set `b` are present in set `a`.

Program:

```
names={'Guava','Mango','Banana'}

print(names)

print(type(names))

# Access Values Using For loop

for name in names:

    print(name)
```

```
# Adding New Element
names.add('Jack Fruit')
print(names)
print("--------------------------")
# Update Another Set of Data
a={'Apple','Orange','Grapes'}
names.update(a)
print(names)
print("--------------------------")
names.remove('Jack Fruit')
print(names)

print("--------------------------")
names.discard('Jack Fruit')
print(names)

print("*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*")
a = {1, 2, 3, 4}
b = {'a', 'b', 'c', 'd'}
c=a.union(b)
print(c)
print("---------------------------------")
a.update(b)
print(a)
a = {1, 2, 3, 4, 5}
b = {5, 6, 7, 8, 9}
c=a.intersection(b)
print(c)
print("**********************")
c=a.symmetric_difference(b)
print(c)
print("**********************")
```

```
a = {5,6,7}
b = {5, 6, 7}
c=a.isdisjoint(b)
print(c)
c=a.issubset(b)
print(c)
c=a.issuperset(b)
print(c)
```

Output:

```
{'Guava', 'Mango', 'Banana'}
<class 'set'>
Guava
Mango
Banana
{'Guava', 'Mango', 'Banana', 'Jack Fruit'}
--------------------------
{'Grapes', 'Mango', 'Banana', 'Jack Fruit', 'Guava', 'Orange', 'Apple'}
--------------------------
{'Grapes', 'Mango', 'Banana', 'Guava', 'Orange', 'Apple'}
--------------------------
{'Grapes', 'Mango', 'Banana', 'Guava', 'Orange', 'Apple'}
*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*
{1, 2, 3, 4, 'c', 'b', 'd', 'a'}
---------------------------------
{5}
**********************
{1, 2, 3, 4, 6, 7, 8, 9}
**********************
False
True
True
```

EX: 4 Operations on Dictionary in python

Aim:

To demonstrate various operations on a Python dictionary

Procedure :

1. Create a dictionary `user` with initial key-value pairs representing name, age, city, and occupation.
2. Print the `user` dictionary, its data type, and the value associated with the key "name".
3. Print all the keys and values in the `user` dictionary using `keys()` and `values()` methods.
4. Print all the keys and their corresponding values in the `user` dictionary using a for loop.
5. Print only the values in the `user` dictionary using a for loop.
6. Print only the keys in the `user` dictionary using a for loop.
7. Add a new key-value pair to the `user` dictionary to represent the gender and print the updated dictionary.
8. Update the age in the `user` dictionary to 35 and print the updated dictionary.
9. Remove the "age" key from the `user` dictionary and print the updated dictionary.
10. Clear all the elements from the `user` dictionary and print the empty dictionary.

Program:

```
user = {
    "name": "Ram",
    "age": 25,
    "city": “TamilNadu",
    “occupation": “Software developer”
}
print(user)
print(type(user))
print(user["name"])
print(user.keys())
print(user.values())
for x in user:
    print(x," ",user[x])
for x in user.values():
    print(x)
for x in user.keys():
    print(x)
```

```
# Changing Values
user.update({"gender":"male"})
print(user)
user["age"]=35
print(user)
user.pop("age")
print(user)
user.clear()
print(user)
```

Output:

```
{'name': 'Ram', 'age': 25, 'city': 'TamilNadu', 'occupation': 'Software developer'}
<class 'dict'>
Ram
dict_keys(['name', 'age', 'city', 'occupation'])
dict_values(['Ram', 25, 'TamilNadu', 'Software developer'])
name   Ram
age   25
city   TamilNadu
occupation   Software developer
Ram
25
TamilNadu
Software developer
name
age
city
occupation
{'name': 'Ram', 'age': 25, 'city': 'TamilNadu', 'occupation': 'Software developer', 'gender': 'male'}
{'name': 'Ram', 'age': 35, 'city': 'TamilNadu', 'occupation': 'Software developer', 'gender': 'male'}
{'name': 'Ram', 'city': 'TamilNadu', 'occupation': 'Software developer', 'gender': 'male'}
{}
```

EX: 5 Simple OOP– Constructors – create a class for representing a car

Aim:

Procedure:

Step 1: Define a class called `Car` with the constructor (`__init__`) method that takes two parameters, `make` and `model`. This method is called when a new object of the class is created and initializes the car's attributes `make` and `model`.

Step 2: Inside the constructor, set the class attributes `self.make` and `self.model` to the values passed as arguments during object creation.

Step 3: Define a method named `display_info` inside the `Car` class. This method takes no parameters besides the implicit `self`, which represents the car object itself.

Step 4: Inside the `display_info` method, use the `print` function to display the car's make and model attributes with appropriate labels.

Step 5: Outside the class definition, create a car object `my_car` by calling the `Car` class constructor with the arguments "Toyota" and "Camry". This will create a new car object with the make "Toyota" and model "Camry".

Step 6: Call the `display_info` method on the `my_car` object. This will display the car's make and model attributes using the `print` statements inside the `display_info` method.

PROGRAM:

```
# Define the Car class

class Car:

    def __init__(self, make, model):

        self.make = make

        self.model = model


    def display_info(self):

        print("Make:", self.make)

        print("Model:", self.model)


# Create a car object

my_car = Car("Toyota", "Camry")


# Call the display_info method to print car information

my_car.display_info()
```

Output:

Make: Toyota

Model: Camry

EX: 6 Method Overloading – create classes for vehicle and Bus and demonstrate

method overloading

AIM:

Procedure:

Step 1: Define the `Vehicle` class with a constructor (`__init__`) method that takes two parameters, `make` and `model`. This method initializes the vehicle's attributes `make` and `model`.

Step 2: Inside the `Vehicle` class, define a method `display_info` that prints the vehicle's make and model.

Step 3: Define the `Bus` class, which is a subclass of the `Vehicle` class. The `Bus` class has a constructor (`__init__`) method that takes three parameters: `make`, `model`, and `passenger_capacity`. This method initializes the bus's attributes `make`, `model`, and `passenger_capacity`.

Step 4: Inside the `Bus` class, override the `display_info` method. This new implementation extends the `display_info` method from the `Vehicle` class by calling the superclass's `display_info` method using `super()`. It then prints the passenger capacity if `show_capacity` is True.

Step 5: Create a `Vehicle` object named `vehicle1` with the make "Toyota" and model "Camry".

Step 6: Call the `display_info` method on the `vehicle1` object, which will print the vehicle's make and model.

Step 7: Create a `Bus` object named `bus1` with the make "Volvo", model "XC60", and a passenger capacity of 50.

Step 8: Call the `display_info` method on the `bus1` object, which will print the bus's make, model, and passenger capacity.

Step 9: Call the `display_info` method on the `bus1` object again, but this time with `show_capacity=False` as an argument. This will print the bus's make and model, but not the passenger capacity.

Program:

```
# Define the Vehicle class

class Vehicle:

    def __init__(self, make, model):

        self.make = make

        self.model = model

    def display_info(self):

        print("Make:", self.make)

        print("Model:", self.model)
```

```
# Define the Bus class as a subclass of Vehicle
class Bus(Vehicle):
    def __init__(self, make, model, passenger_capacity):
        super().__init__(make, model)
        self.passenger_capacity = passenger_capacity

    def display_info(self, show_capacity=True):
        super().display_info()
        if show_capacity:
            print("Passenger Capacity:", self.passenger_capacity)

# Create a Vehicle object and demonstrate method overloading
vehicle1 = Vehicle("Toyota", "Camry")
vehicle1.display_info()
print()  # Just to add some space in the output
# Create a Bus object and demonstrate method overloading
bus1 = Bus("Volvo", "XC60", 50)
bus1.display_info()
print()  # Just to add some space in the output
# Call the display_info method of Bus, but don't show the capacity
bus1.display_info(show_capacity=False)
```

Output:

Make: Toyota

Model: Camry

Make: Volvo

Model: XC60

Passenger Capacity: 50

Make: Volvo

Model: XC60

Ex: 7 Files – Reading and Writing

Aim: To demonstrate file reading and writing operations in Python

Procedure:

1. The code begins by opening a file named "demofile.txt" in read mode ("r").

2. It enters a loop that reads each line of the file, and the lines are printed one by one. Once the loop is done, it means the end of the file has been reached.

3. The file "demofile.txt" is closed using the `f.close()` method.

4. The code now opens another file named "demofile2.txt" in append mode ("a").

5. It writes the string "Now the file has more content!" to "demofile2.txt" using the `f.write()` method. Since the file is opened in append mode, the new content is added at the end of the file without overwriting the existing content.

6. The file "demofile2.txt" is closed using `f.close()` after the content has been appended.

7. It then reopens "demofile2.txt" in read mode ("r") and reads the entire content using `f.read()`.

8. The content of "demofile2.txt" is printed to the console.

9. The file "demofile3.txt" is opened in write mode ("w").

10. The string "Woops! I have deleted the content!" is written to "demofile3.txt" using the `f.write()` method. Since the file is opened in write mode, it clears any existing content and writes the new content from the beginning.

11. The file "demofile3.txt" is closed using `f.close()` after writing the new content.

12. It then reopens "demofile3.txt" in read mode ("r") and reads the entire content using `f.read()`.

13. The content of "demofile3.txt" is printed to the console.

Program:

```
## File Reading & Writing

f = open("demofile.txt", "r")

for x in f:

 print(x)

f.close()

f = open("demofile2.txt", "a")

f.write("Now the file has more content!")

f.close()

#open and read the file after the appending:

f = open("demofile2.txt", "r")

print(f.read())
```

```
f.close()

f = open("demofile3.txt", "w")

f.write("Woops! I have deleted the content!")

f.close()

#open and read the file after the appending:

f = open("demofile3.txt", "r")

print(f.read())
```

Ex: 8 Regular Expressions

Aim: To demonstrate various functionalities of regular expressions in Python

procedure:

1. `import re`: Imports the Python regular expression module.

2. `txt = "Welcome to python Learning"`: Defines a text string to be used for regular expression operations.

3. `x = re.search("^The.*python$", txt)`: Searches for a pattern that starts with "The", followed by any characters (denoted by ".*"), and ends with "python" in the given text string "txt".

4. `if x:`: Checks if a match is found using the `re.search()` method.

5. `print("YES! We have a match!")`: Prints a message indicating that a match is found if the regular expression pattern is satisfied.

6. `else:`: If no match is found, this block of code executes.

7. `print("No match")`: Prints a message indicating that there is no match.

8. `x = re.findall("ai", txt)`: Uses `re.findall()` to find all occurrences of the pattern "ai" in the text string "txt".

9. `print(x)`: Prints the list of all occurrences of "ai" found in the text.

10. `x = re.search("\s", txt)`: Searches for the first white-space character in the text string "txt".

11. `print("The first white-space character is located in position:", x.start())`: Prints the position (index) of the first white-space character found.

12. `x = re.split("\s", txt)`: Splits the text string "txt" using white-space characters as delimiters.

13. `print(x)`: Prints the list of substrings obtained after splitting.

14. `x = re.split("\s", txt, 1)`: Splits the text string "txt" using white-space characters as delimiters, but only performs the split once.

15. `print(x)`: Prints the list of substrings obtained after the first split.

16. `x = re.sub("\s", "9", txt)`: Replaces all occurrences of white-space characters with "9" in the text string "txt".

17. `print(x)`: Prints the updated text string after the substitution.

PROGRAM:

```
import re

txt = "Welcome to python Learning"

# Searching for a pattern
x = re.search("^The.*python$", txt)
if x:
    print("YES! We have a match!")
else:
    print("No match")

# Finding all occurrences of "ai"
x = re.findall("ai", txt)
print(x)

# Finding the position of the first white-space character
x = re.search("\s", txt)
print("The first white-space character is located in position:", x.start())

# Splitting the text using white-space characters as delimiters
x = re.split("\s", txt)
print(x)

# Splitting the text once using white-space characters as delimiters
x = re.split("\s", txt, 1)
print(x)

# Replacing all white-space characters with "9"
x = re.sub("\s", "9", txt)
print(x)
```

Output :

No match

['Learning']

The first white-space character is located in position: 7

['Welcome', 'to', 'python', 'Learning']

['Welcome', 'to python Learning']

Welcome9to9python9Learning

EX:9(a) Predefined Modules

AIM:

PROCEDURE:

1. Import the math module using the statement "import math".
2. Print the value of the mathematical constant "π" (pi) using the expression "math.pi".
3. End of the program.

PROGRAM:

```
Import math
print("The value of PI is :", math.pi)
```

output:

The value of pi is 3.141592653589793

Ex: 9(b) create a module

Aim: To demonstrate the use of modules in Python

Procedure :

For ".py":

1. Define the function "add(a, b)" to calculate the sum of two numbers a and b.
2. Return the sum of a and b.
3. Define arithmetic the function "multi(a, b)" to calculate the product of two numbers a and b.
4. Return the product of a and b.
5. Save and close the "arithmetic.py" file.

For "main.py":

1. Import the "arithmetic.py" module using "import arithmetic".
2. Call the "arithmetic.add(5, 6)" function.
3. Pass the arguments 5 and 6 to the "add()" function.
4. Print the result of the addition.
5. End of the program.

Program :

```
# arithmetic.py
def add(a, b):
    return a + b
def multi(a, b):
    return a * b
# import module
# main.py
import arithmetic
print("My Module:", arithmetic.add(5, 6))
```

Ex: 10 packages in python

Aim: To demonstrate the use of package in python

Procedure:

1. Create a new directory named "mypackage."
2. Inside "mypackage," create the following files:
   - `__init__.py`: An empty file.
   - `module1.py`: Define a function `greet()` that returns "Hello from Module 1!" as the output.
   - `module2.py`: Define a function `greet()` that returns "Hello from Module 2!" as the output.
3. Create a file named `main.py` outside the "mypackage" directory.
4. In `main.py`, import the `greet()` function from both `module1` and `module2` and print the results.

Program:

mypackage/module1.py:

```
def greet():
    return "Hello from Module 1!"
```

mypackage/module2.py:

```
def greet():
    return "Hello from Module 2!"
```

main.py:

```
from mypackage import module1, module2

print(module1.greet())
print(module2.greet())
```

OUTPUT:

Hello from Module 1!

Hello from Module 2!

RESULT:

The program successfully demonstrates the use of a package in Python.

Ex: 11 Exception handling in Python

Aim :

Procedure:

1. Ask the user to input two numbers.
2. Try to convert the user input to integers and store them in variables 'a' and 'b'.
3. If the conversion is successful, proceed to step 6.
4. If a 'ValueError' occurs during the conversion, jump to step 8.
5. If 'b' is 0 and division is attempted, jump to step 10.
6. Perform the division 'c = a / b'.
7. Print the result of the division and jump to step 12.
8. Print "Only integers allowed" and jump to step 12.
9. Print "Cannot divide by Zero" and jump to step 12.
10. Print "Error" due to any other unexpected error and jump to step 12.
11. Print "The Program is completed."
12. End of the program.

Program:

```
try:
  a= int(input("Enter 1st number:"))
  b=int(input("Enter 2nd number:"))
  c=a/b
  print ("Division result:",c)
except zeroDivisionError:
  print ("cannot divide by Zero")
except valueError:
  print ("Only integers allowed")

except BaseException:
  print ("Error")

finally:
  print(" The Program is completed ")
```

OUTPUT 1:

Enter 1st number: 10

Enter 2nd number: 2

Division result: 5.0

The Program is completed

Output 2:

Enter 1st number: 10

Enter 2nd number: 0

cannot divide by Zero

The Program is completed