



**G. H. Raisoni**  
**College of Engineering & Management**

Wagholi, Pune

# Certificate

*This is to certify that*

*Mr./Ms.* **SWAYAM PRAMOD TERODE**

*studying in Semester/Year* **SEM 2/First Year**

*Branch* **Information of Technology**

*Name of the Dept.* **FIRST YEAR**

*Section* **C**

*has satisfactorily completed the practical work of*

*subject* **Programming for Problem Solving (UITP 102)**

*during the academic year* **2020**

\_\_\_\_\_  
Signature of Subject Teacher

\_\_\_\_\_  
Signature of Head of Department



**RAISONI GROUP**  
A vision beyond



**G.H.Raison College of Engineering and Management  
Wagholi, Pune**

(An Autonomous Institute affiliated to SPPU)

Department of FYBTECH

**Programming for Problem Solving  
(UITP102) (2020 Scheme)**

Lab manual

Prepared By:

Mrs.Mugdha Kirkire (Asst. Professor)

NAME- Swayam Pramod Terode

ROLL NO.- C70

SUBJECT- Programming for Problem Solving

UNDER THE GUIDANCE OF- Mrs. Mugdha Kirkire mam

**Course Objectives:**

1. Introduce the student to Python programming fundamentals
2. Expose students to application development and prototyping using Python
3. Learn to apply fundamental problem solving techniques

**Course Outcomes:**

1. Understand principles of Python
2. Understand how Python can be used for application development as well as quick networking, QA and game programming
3. Understand object oriented programming

## **CONTENTS & INDEX**

<b>Sr no</b>	<b>Experiment List</b>	<b>Page no</b>
1.	Write a Python program to display current date and time.	6
2.	Write a python program to perform mathematical operation in python.	14
3.	Write a python Program for checking whether the given number is an even number or not.	15
4.	Write a python program using WHILE loop to print 100 to 1 numbers.	16
5.	Write a python program to print a table of any number using for loop.	17
6.	Write a python program to find greater number using if statements.	18
7.	Write a python program to swap two numbers.	19
8.	Write a python to print pattern program using FOR loop.	20
9.	Write a python program to demonstrate break and continue statement.	22
10.	Write a python program to calculate percentage and average marks of a student.	24
11.	Write a python program to find factorial of a given number.	25
12.	Write a python program to demonstrate a concept of a function.	26
13.	Write a python program to check whether a given number is prime or not.	28
14.	Write an Anonymous function to perform mathematical operations.	29
15.	Write a program to implement following string operations: a. Count b. concate c. lower and upper	31
16.	Write a python program to check whether a given string is palindrome or not.	33

17.	Write a python program to print a reverse of user entered number.	34
18.	Write a python program to read a file and display content in a file.	35
19.	Write a python program which uses import packages and statements.	37
20.	Write a python program to handle exceptions.	40
21.	Design a project for student management system for GHRCEM.	50

## **Assignment no: 1**

Aim: - Introduction to python programming

Problem Statement: - Write a Python program to display current date and time.

Prerequisites: - Python, IDE, Python data types

## **Theory:**

### **PART 1:- Introduction**

#### **Introduction to python**

Python is a powerful general-purpose programming language. It is used in web development, data science, creating software prototypes, and so on. Fortunately for beginners, Python has simple easy-to-use syntax. This makes Python an excellent language to learn to program for beginners.

#### **Python Characteristics:-**

1. Python is a cross-platform programming language, which means that it can run on multiple platforms like Windows, macOS, Linux.
2. It has even been ported to the Java and .NET virtual machines.
3. It is free and open-source.

## Installation Process:-

The easiest way to run Python is by using **Thonny IDE**.

The Thonny IDE comes with the latest version of Python bundled in it. So you don't have to install Python separately.

Follow the following steps to run Python on your computer.

1. Download [Thonny IDE](#).
2. Run the installer to install **Thonny** on your computer.
3. Go to: **File > New**. Then save the file with `.py` extension. For example, `hello.py`, `example.py`, etc.  
You can give any name to the file. However, the file name should end with `.py`
4. Write Python code in the file and save it.
5. Write Python code in the file and save it.



Running Python using Thonny IDE

6. Then Go to **Run > Run current script** or simply click **F5** to run it.

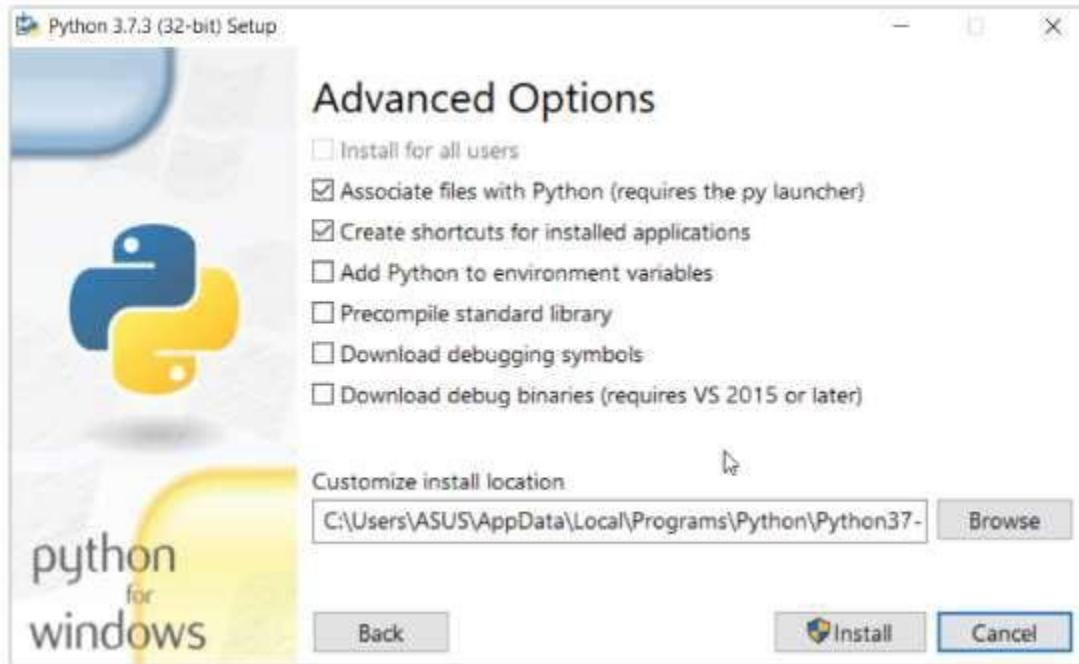


## Install Python Separately

If you don't want to use Thonny, here's how you can install and run Python on your computer.

1. Download the [latest version of Python](#).
2. Run the installer file and follow the steps to install Python  
During the install process, check **Add Python to environment variables**. This will add Python to environment variables, and you can run Python from any part of the computer.

Also, you can choose the path where Python is installed.



Installing Python on the computer

Once you finish the installation process, you can run Python.

Once Python is installed, typing `python` in the command line will invoke the interpreter in immediate mode. We can directly type in Python code, and press Enter to get the output.

Try typing in `1 + 1` and press enter. We get 2 as the output. This prompt can be used as a calculator. To exit this mode, type `quit()` and press enter.



```
Microsoft Windows [Version 10.0.17134.648]
(c) 2018 Microsoft Corporation. All rights reserved.

C:\Users\ASUS>python
Python 3.7.2 (tags/v3.7.2:9a3ffc0492, Dec 23 2018, 22:20:52)
[MSC v.1916 32 bit (Intel)] on win32
Type "help", "copyright", "credits" or "license()" for more
>>> 1 + 1
2
>>> quit()

C:\Users\ASUS>_
```

Running Python on the Command Line

---

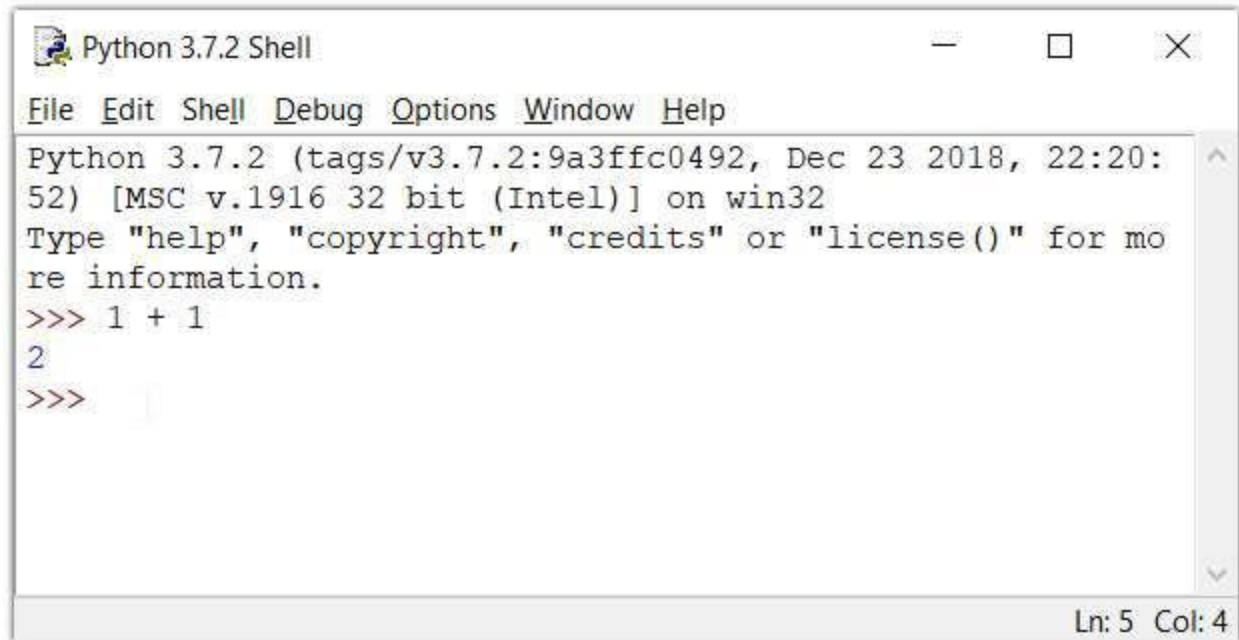
## 2. Run Python in the Integrated Development Environment (IDE)

We can use any text editing software to write a Python script file.

We just need to save it with the `.py` extension. But using an IDE can make our life a lot easier. IDE is a piece of software that provides useful features like code hinting, syntax highlighting and checking, file explorers, etc. to the programmer for application development.

By the way, when you install Python, an IDE named **IDLE** is also installed. You can use it to run Python on your computer. It's a decent IDE for beginners.

When you open IDLE, an interactive Python Shell is opened.



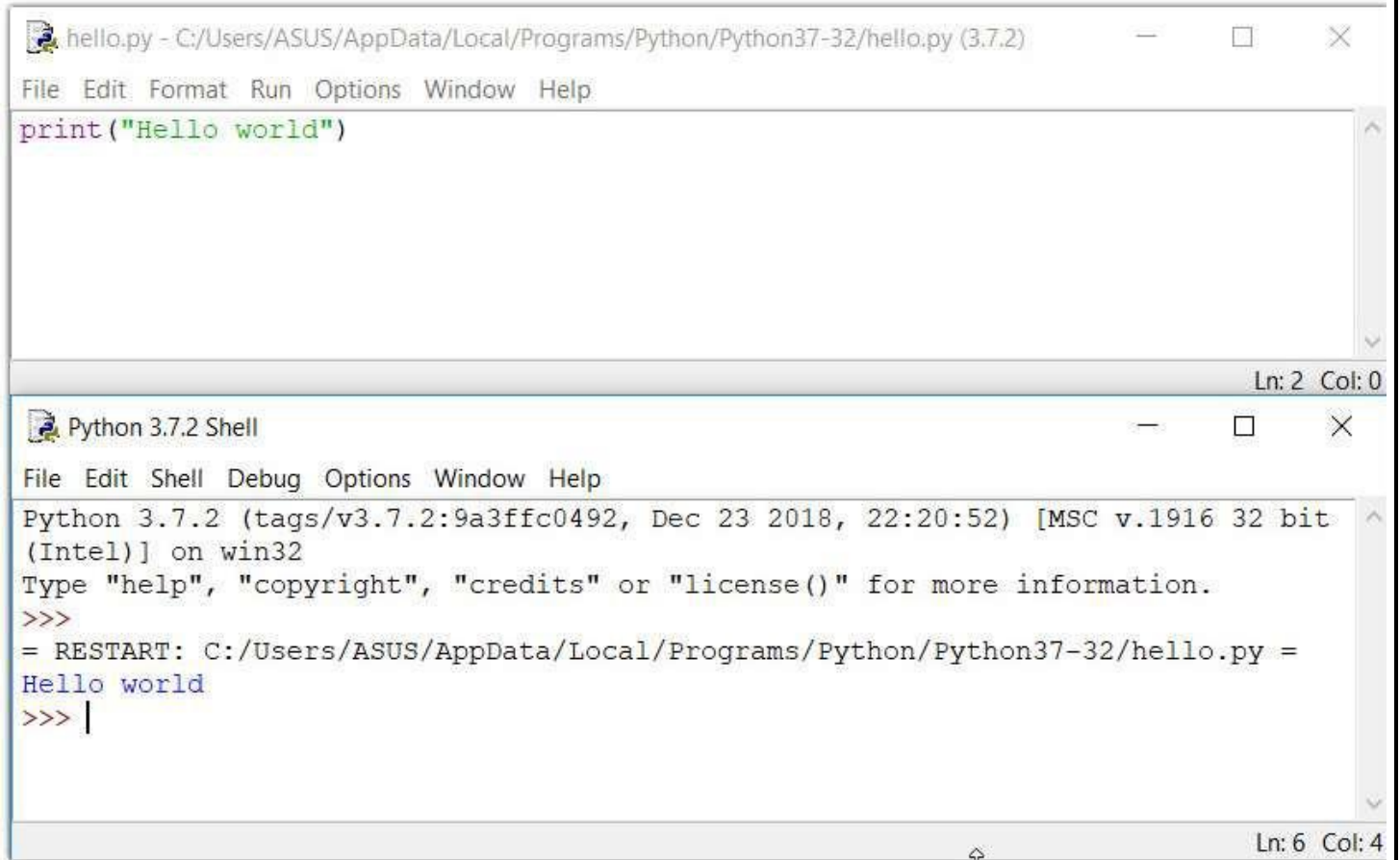
The image shows a screenshot of the Python 3.7.2 Shell window. The window has a title bar that says "Python 3.7.2 Shell" and standard window controls (minimize, maximize, close). Below the title bar is a menu bar with the following items: File, Edit, Shell, Debug, Options, Window, and Help. The main text area contains the following text: "Python 3.7.2 (tags/v3.7.2:9a3ffc0492, Dec 23 2018, 22:20:52) [MSC v.1916 32 bit (Intel)] on win32", "Type 'help', 'copyright', 'credits' or 'license()' for more information.", and the command prompt ">>>" followed by the input "1 + 1" and the output "2". The status bar at the bottom right of the window shows "Ln: 5 Col: 4".

```
Python 3.7.2 (tags/v3.7.2:9a3ffc0492, Dec 23 2018, 22:20:52) [MSC v.1916 32 bit (Intel)] on win32
Type "help", "copyright", "credits" or "license()" for more information.
>>> 1 + 1
2
>>>
```

Python IDLE

Now you can create a new file and save it with **.py** extension. For example, **hello.py**

Write Python code in the file and save it. To run the file, go to **Run > Run Module** or simply click **F5**.



The image shows two windows from the Python IDLE environment. The top window, titled 'hello.py - C:/Users/ASUS/AppData/Local/Programs/Python/Python37-32/hello.py (3.7.2)', contains a single line of Python code: `print("Hello world")`. The bottom window, titled 'Python 3.7.2 Shell', shows the output of running the code. It displays the Python version and build information, followed by the prompt `>>>`. The user has entered `= RESTART: C:/Users/ASUS/AppData/Local/Programs/Python/Python37-32/hello.py =`, and the shell has printed `Hello world` before returning to the `>>>` prompt.

Running a Python program in IDLE

---

---

In Python, **date, time and date time** classes provides a number of function to deal with dates, times and time intervals. Date and date time in Python are the objects, so when you manipulate them, you are actually manipulating objects and not string or timestamps. Whenever you manipulate dates or time, you need to import date time function.

The date time classes in Python are categorized into main 5 classes.

- date – Manipulate just date ( Month, day, year)
- time – Time independent of the day (Hour, minute, second, microsecond)
- date time – Combination of time and date (Month, day, year, hour, second, microsecond)
- time delta— A duration of time used for manipulating dates
- tzinfo— An abstract class for dealing with time zones

Today's Weekday Number

The `date.Today ( )` function also gives you the weekday number. Here is the Weekday Table which start with Monday as 0 and Sunday as 6

Day	Week Day Number
Monday	0
Tuesday	1
Wednesday	2
Thursday	3
Friday	4
Saturday	5
Sunday	6

## Python date time:

The date time module supplies classes for manipulating dates and times in both simple and complex ways. Date time `.now (tz=None)` returns the current local date and time. If optional argument `tz` is `None` or not specified, this is like `today ()`.

`date.strftime (format)` returns a string representing the date, controlled by an explicit format string. Format codes referring to hours, minutes or seconds will see 0 values.

### Algorithm:

1. `import datetime`
2. `assign now now = datetime.datetime.now()`
3. `Print current date and time`
4. `Print (now.strftime("%Y-%m-%d %H:%M:%S"))`

### Python code:

```
import datetime  
  
now = datetime.datetime.now()  
  
print ("Current date and time : ")  
  
print (now.strftime("%Y-%m-%d %H:%M:%S"))
```

### OUTPUT:

Shell Clear

Current date and time :  
2021-07-31 17:12:52  
> |

## Assignment no:2

**Problem Statement:** Write a program to perform mathematical operation in python.

### Theory:

It simply takes two integer numbers and performs arithmetic operations like addition, subtraction, multiplication, division

### Algorithm:

- Read two input integers using input() or raw\_input().
- Addition operation using + operator, num1 + num2 adds 2 numbers.
- Subtraction operation using - operator, num1 - num2 right hand operand from left hand operand.
- Multiplication operation using \* operator, num1 \* num2 multiplies 2 numbers.
- Division operation using / operator, num1 / num2 divides left hand operand by right hand operand.
- Print the result of each operation.

### Python code:

```
num1 = int(input('Enter First number: '))
num2 = int(input('Enter Second number '))
add = num1 + num2
dif = num1 - num2
mul = num1 * num2
div = num1 / num2
print('Sum of ',num1 , 'and' , num2 , 'is :',add)
print('Difference of ',num1 , 'and' , num2 , 'is :',dif)
print('Product of' , num1 , 'and' , num2 , 'is :',mul)
print('Division of ',num1 , 'and' , num2 , 'is :',div)
```

### OUTPUT:

Shell Clear

```
Enter First number: 23
Enter Second number 34
Sum of  23 and 34 is : 57
Difference of  23 and 34 is : -11
Product of 23 and 34 is : 782
Division of  23 and 34 is : 0.6764705882352942
> |
```

## Assignment no:03

**Problem Statement:** Write a python Program for checking whether the given number is a even number or not.

### Theory:

A number is even if it is perfectly divisible by 2. When the number is divided by 2, we use the remainder operator % to compute the remainder. If the remainder is not zero, the number is odd.

### Algorithm:

1. Take number as an input
2.  $Mod = num \% 2$
3. If  $mod > 0$

    Print odd number

Else

    Print even number

### Python code:

```
num = int(input("Enter a number: "))  
mod = num % 2  
if mod > 0:  
    print("This is an odd number.")  
else:  
    print("This is an even number.")
```

### OUTPUT:

Shell Clear

Enter a Number: 45  
Odd Number  
> |



## Assignment no: 04

**Problem statement:** Write a python program using WHILE loop to print 100 to 1 numbers .

### Theory:

The while loop in Python is used to iterate over a block of code as long as the test expression (condition) is true.

We generally use this loop when we don't know beforehand, the number of times to iterate.

Syntax of while Loop in Python

```
while test_expression:
```

```
    Body of while
```

In while loop, test expression is checked first. The body of the loop is entered only if the test\_expression evaluates to True. After one iteration, the test expression is checked again. This process continues until the test\_expression evaluates to False.

In Python, the body of the while loop is determined through indentation.

Body starts with indentation and the first unindented line marks the end.

Python interprets any non-zero value as True. None and 0 are interpreted as False.

### Algorithm:

1. initialise value of i
2. put the condition as while(i>=1):
3. print i
4. Decrease value of i

### Python code:

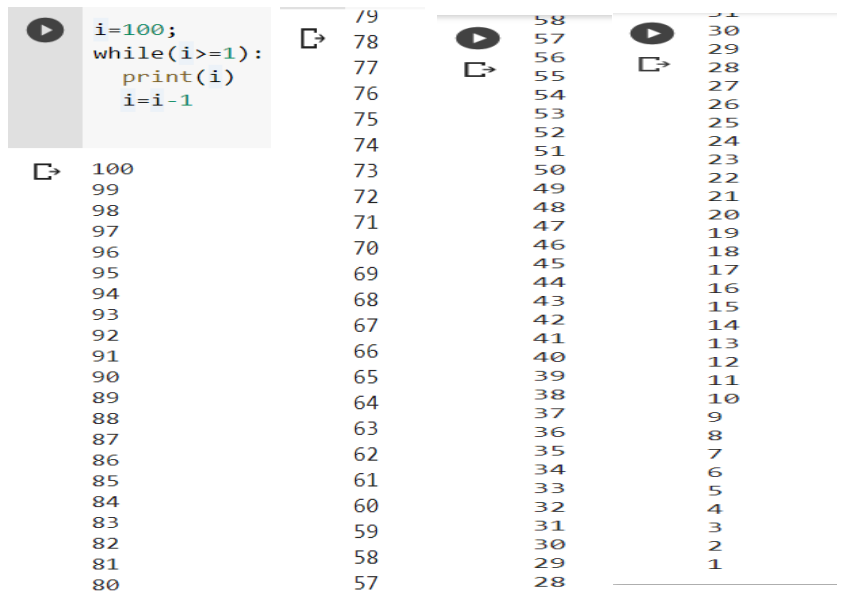
```
i=100;

while(i>=1):
    print(i)

    i=i-1
```

### OUTPUT:

### OUTPUT:



```
i=100;
while(i>=1):
    print(i)
    i=i-1
```

100  
99  
98  
97  
96  
95  
94  
93  
92  
91  
90  
89  
88  
87  
86  
85  
84  
83  
82  
81  
80  
79  
78  
77  
76  
75  
74  
73  
72  
71  
70  
69  
68  
67  
66  
65  
64  
63  
62  
61  
60  
59  
58  
57  
56  
55  
54  
53  
52  
51  
50  
49  
48  
47  
46  
45  
44  
43  
42  
41  
40  
39  
38  
37  
36  
35  
34  
33  
32  
31  
30  
29  
28  
27  
26  
25  
24  
23  
22  
21  
20  
19  
18  
17  
16  
15  
14  
13  
12  
11  
10  
9  
8  
7  
6  
5  
4  
3  
2  
1

## Assignment no:05

**Problem Statement:** Write a python program to print a table of any number using for loop.

### Theory:

What is for loop in Python?

The for loop in Python is used to iterate over a sequence ([list](#), [tuple](#), [string](#)) or other iterable objects. Iterating over a sequence is called traversal.

Syntax of for Loop

```
for val in sequence:
```

```
    Body of for
```

Here, val is the variable that takes the value of the item inside the sequence on each iteration.

Loop continues until we reach the last item in the sequence. The body of for loop is separated from the rest of the code using indentation.

### The range() function

We can generate a sequence of numbers using `range()` function. `range(10)` will generate numbers from 0 to 9 (10 numbers).

We can also define the start, stop and step size as `range(start,stop,step size)`. step size defaults to 1 if not provided.

This function does not store all the values in memory, it would be inefficient. So it remembers the start, stop, step size and generates the next number on the go.

To force this function to output all the items, we can use the function `list()`.

### Algorithm:

- 1.Take input from user
- 2.apply specific range
- 3.print

### Python code:

```
n=int (input("Enter any number: "))
```

```
for i in range(1,11):
```

```
    print n*i
```

### OUTPUT:

#### Shell

```
Enter any number: 4
Multiplication Table of 4
4 X 1 = 4
4 X 2 = 8
4 X 3 = 12
4 X 4 = 16
4 X 5 = 20
4 X 6 = 24
4 X 7 = 28
4 X 8 = 32
4 X 9 = 36
4 X 10 = 40
> |
```

## Assignment no:06

**Problem Statement:** Write a python program to find greater number using if statements.

### Theory:

What are if...else statement in Python?

Decision making is required when we want to execute a code only if a certain condition is satisfied.

The if...elif...else statement is used in Python for decision making.

Python if Statement Syntax

if test expression:

```
statement(s)
```

Here, the program evaluates the test expression and will execute statement(s) only if the text expression is True.

If the text expression is False, the statement(s) is not executed.

In Python, the body of the if statement is indicated by the indentation. Body starts with an indentation and the first unindented line marks the end.

Python interprets non-zero values as True. None and 0 are interpreted as False.

### Algorithm:

1. input values of a,b,c
2. if value of a is greater then b and c  
Print a is greater
3. if value of b is greater then a and c  
Print b is greater
4. if value of c is greater then a and b  
Print c is greater

### Python code:

```
a=int (input("Enter value of a: "))
b=int (input("Enter value of b: "))
c=int (input("Enter value of c: "))
if(a>b)and(a>c)
```

```
    print("a is greater")
```

```
if(b>a)and(b>c)
```

```
    print("b is greater")
```

```
if(c>a)and(c>b)
```

```
    print("c is greater")
```

### OUTPUT:

Shell

```
Enter value of a: 12
Enter value of b: 23
Enter value of c: 34
c is greater
> |
```

## Assignment no:07

**Problem Statement:** Write a python program to swap two numbers.

### Theory:

In this example, we swap two variables by using a temporary variable.

Swapping means exchange the values of a variable.

### Algorithm:

- 1.input the values of variable
- 2.Create one temporary variable.
- 3.Assign temp=first variable
- 4.exchange the variables.
- 5.Print the values.

### Python code:

```
# To take input from the user
x = input('Enter value of x: ')
y = input('Enter value of y: ')

x = 5
y = 10

# create a temporary variable and swap the values
temp = x
x = y
y = temp

print('The value of x after swapping: {}'.format(x))
print('The value of y after swapping: {}'.format(y))
```

### OUTPUT:

```
Shell
Enter value of x: 12
Enter value of y: 23
The value of x after swapping: 10
The value of y after swapping: 5
> |
```

## Assignment no:08

**Problem Statement:** Write a python to print pattern program using FOR loop.

### Theory:

What is for loop in Python?

The for loop in Python is used to iterate over a sequence ([list](#), [tuple](#), [string](#)) or other iterable objects. Iterating over a sequence is called traversal.

Syntax of for Loop

for val in sequence:

Body of for

Here, val is the variable that takes the value of the item inside the sequence on each iteration.

Loop continues until we reach the last item in the sequence. The body of for loop is separated from the rest of the code using indentation.

### The range() function

We can generate a sequence of numbers using range() function. range(10) will generate numbers from 0 to 9 (10 numbers).

We can also define the start, stop and step size as range(start,stop,step size). step size defaults to 1 if not provided.

This function does not store all the values in memory, it would be inefficient. So it remembers the start, stop, step size and generates the next number on the go.

To force this function to output all the items, we can use the function list().

### Algorithm:

- 1.input number of rows from user
- 2.take variable i range of upto n
- 3.take variable j for increment in range of i
- 4.print \*

### Python code:

```
n=int (input("Enter number of rows: "))
```

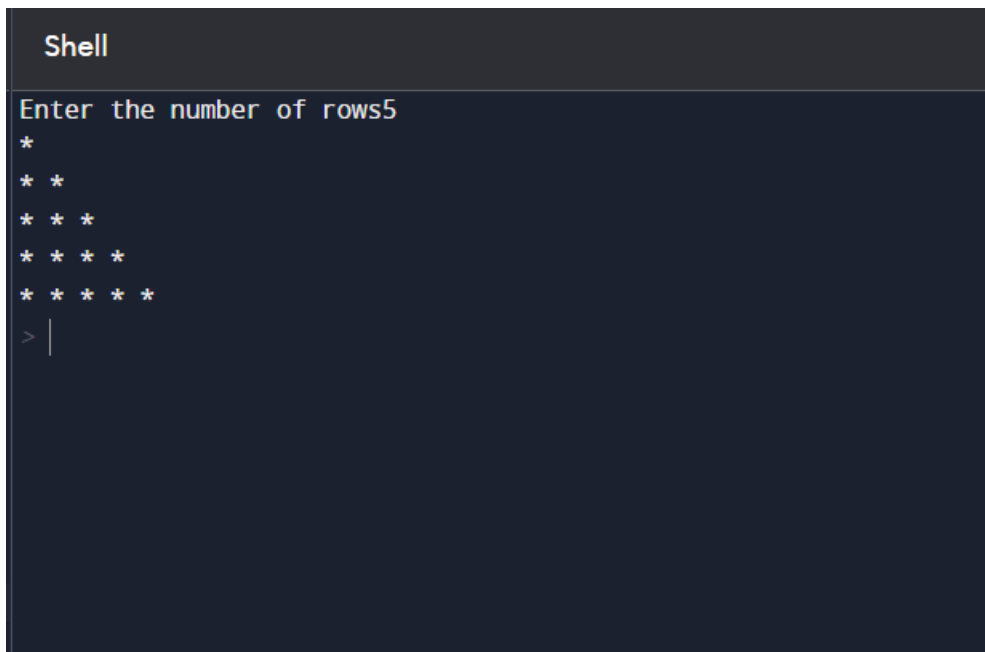
```
for i in range(n);
```

```
for j in range(i+1)
```

```
    print(" ", end=" ")
```

```
    print()
```

**OUTPUT:**



```
Shell
Enter the number of rows5
*
* *
* * *
* * * *
* * * * *
> |
```

## Assignment no:09

**Problem Statement:** Write a python program to demonstrate break and continue statement.

### Theory:

#### Python break statement

The break statement terminates the loop containing it. Control of the program flows to the statement immediately after the body of the loop.

If break statement is inside a nested loop (loop inside another loop), break will terminate the innermost loop.

#### Syntax of break

```
break
```

#### Python continue statement

The continue statement is used to skip the rest of the code inside a loop for the current iteration only. Loop does not terminate but continues on with the next iteration.

#### Syntax of Continue

```
continue
```

### Algorithm:

1. Take value of a from user
2. take range from till you want to display
3. implement for loop to display the numbers.
4. print
5. check value of variable i is greater than range condition.
6. apply condition accordingly.

Python code to implement **break**:

```
a=int (input("Enter value of a: "))
```



```
b=int(input("Enter the range till you want to display: "))
for i in range (1,11):

    print a*i

    if(i>=b):

        break
```

### Python code to implement continue:

```
a=int (input("Enter value of a: "))
b=int(input("Enter the range till you want to display: "))
for i in range (1,11):

    print a*i

    if(i>=b):

        continue
```

### OUTPUT: BREAK

### CONTINUE

```
Shell
Enter value of a: 2
Enter the range till you want to display: 5
10|
>
```

```
Shell
Enter value of a: 7
Enter the range till you want to display: 8
70
> |
```

## Assignment no:10

**Problem statement:** Write a python program to calculate percentage and average marks of a student

### Theory:

#### Logical Operators

Logical operators in Python are used for conditional statements are true or false. Logical operators in Python are AND, OR and NOT. For logical operators following condition are applied.

- For AND operator – It returns TRUE if both the operands (right side and left side) are true
- For OR operator- It returns TRUE if either of the operand (right side or left side) is true
- For NOT operator- returns TRUE if operand is false

The program takes in the marks of 5 subjects and displays the grade.

#### Algorithm:

1. Take in the marks of 5 subjects from the user and store it in different variables.
2. Find the average of the marks.
3. Use an else condition to decide the grade based on the average of the marks.
4. Exit.

#### Python code:

```
sub1=int(input("Enter marks of the first subject: "))
sub2=int(input("Enter marks of the second subject: "))
sub3=int(input("Enter marks of the third subject: "))
sub4=int(input("Enter marks of the fourth subject: "))
sub5=int(input("Enter marks of the fifth subject: "))
avg=(sub1+sub2+sub3+sub4+sub5)/5
if(avg>=90):
    print("Grade: A")
elif(avg>=80&avg<90):
    print("Grade: B")
elif(avg>=70&avg<80):
    print("Grade: C")
elif(avg>=60&avg<70):
    print("Grade: D")
else:
    print("Grade: F")
```

#### OUTPUT:

```
Shell
Enter marks of the first subject: 34
Enter marks of the second subject : 56
Enter marks of the third subject: 76
Enter marks of the fourth subject: 56

Enter marks of the fifth subject: 45
Average mark: 89.0
Grade is B
> |
```

## Assignment no:11

**Problem Statement:** Write a python program to find factorial of a given number.

### Theory:

The factorial of a number is the product of all the integers from 1 to that number.

For example, the factorial of 6 (denoted as 6!) is  $1*2*3*4*5*6 = 720$ . Factorial is not defined for negative numbers and the factorial of zero is one,  $0! = 1$ .

### Algorithm:

1. take input from user
2. check the positive or negative for negative number factorial does not exist
3. use for i in range(1,num + 1) and factorial = factorial\*i
4. print factorial
5. Stop.

### Python code:

```
# Python program to find the factorial of a number provided by the user.

# change the value for a different result
num = 7

# uncomment to take input from the user
#num = int(input("Enter a number: "))

factorial = 1

# check if the number is negative, positive or zero
if num < 0:
    print("Sorry, factorial does not exist for negative numbers")
elif num == 0:
    print("The factorial of 0 is 1")
else:
    for i in range(1,num + 1):
        factorial = factorial*i
    print("The factorial of",num,"is",factorial)
```

### OUTPUT:

```
Shell
The factorial of 7 is 5040
> |
```

## Assignment no:12

Problem Statement: Write a python program to demonstrate a concept of a function

**Theory:** What is a function in Python?

In Python, function is a group of related statements that perform a specific task.

Functions help break our program into smaller and modular chunks. As our program grows larger and larger, functions make it more organized and manageable.

Furthermore, it avoids repetition and makes code reusable.

### Syntax of Function

```
def function_name(parameters):
```

```
    """docstring"""
```

```
    statement(s)
```

Above shown is a function definition which consists of following components.

1. Keyword `def` marks the start of function header.
2. A function name to uniquely identify it. Function naming follows the same [rules of writing identifiers in Python](#).
3. Parameters (arguments) through which we pass values to a function. They are optional.
4. A colon (`:`) to mark the end of function header.
5. Optional documentation string (docstring) to describe what the function does.
6. One or more valid python statements that make up the function body. Statements must have same indentation level (usually 4 spaces).
7. An optional `return` statement to return a value from the function.

Algorithm:

- 1.define the function
- 2.perform all mathematical operations
- 3.print result
- 4.stop.

Python code without function for mathematical operations:

```
a=10
b=20
print("the sum is: ",sum)
print("the difference is: ",difference)
print("the product is: ",product)
print("the division is: ",division)
a=200
```

```

b=300
print("the sum is: ",sum)
print("the difference is: ",difference)
print("the product is: ",product)
print("the division is: ",division)
a=400
b=500
print("the sum is: ",sum)
print("the difference is: ",difference)
print("the product is: ",product)
print("th print("the sum is: ",sum)
print("the difference is: ",difference)
print("the product is: ",product)
print("the division is: ",division)

a=700
b=900
print("the sum is: ",sum)
print("the difference is: ",difference)
print("the product is: ",product)
print("the division is: ",division)

```

Python code with function for mathematical operations:

```

def calculate(a,b):
    print("the sum is: ",sum)
    print("the difference is: ",difference)
    print("the product is: ",product)
    print("the division is: ",division)
calculate (10,20)
calculate (200,300)
calculate (400,500)
calculate (700,900)

```

## OUTPUT:

```

Shell
the sum is 30
the difference is -10
the product is 200
the division is 0.5
the sum is 500
the difference is -100
the product is 60000
the division is 0.6666666666666666
the sum is 900
the difference is -100
the product is 200000
the division is 0.8
the sum is 1600
the difference is -200
the product is 630000
the division is 0.7777777777777778
>

```

## Assignment no:13

**Problem Statement:** Write a python program to check whether a given number is prime or not.

### Theory:

Prime number is number which is divisible by 1 and number itself.

A positive integer greater than 1 which has no other factors except 1 and the number itself is called a prime number. 2, 3, 5, 7 etc. are prime numbers as they do not have any other factors. But 6 is not prime (it is composite) since,  $2 \times 3 = 6$ .

### Algorithm:

1. `num = int(input("Enter a number: "))`
2. `if num > 1:`
3. `for i in range(2,num):`
4. `if (num % i) == 0:`
5. `print(num,"is not a prime number")`
6. `print(i,"times",num//i,"is",num)`
7. `break.`
8. `else:`

### Python code:

```
#take input from the user
# num = int(input("Enter a number: "))

# prime numbers are greater than 1
if num > 1:
    # check for factors
    for i in range(2,num):
        if (num % i) == 0:
            print(num,"is not a prime number")
            print(i,"times",num//i,"is",num)
            break
    else:
        print(num,"is a prime number")

# if input number is less than
# or equal to 1, it is not prime
else:
    print(num,"is not a prime number")
```

### OUTPUT:

```
Shell
Please enter the number: 34
34 is not a prime number
2 times 17 is 34
> |
```

## Assignment no:14

**Problem Statement:** Write a Anonymous function to perform mathematical operations.

**Theory:**

### What are lambda functions in Python?

In Python, anonymous function is a [function](#) that is defined without a name.

While normal functions are defined using the `def` keyword, in Python anonymous functions are defined using the `lambda` keyword.

Hence, anonymous functions are also called lambda functions.

### How to use lambda Functions in Python?

A lambda function in python has the following syntax.

Syntax of Lambda Function in python

```
lambda arguments: expression
```

Lambda functions can have any number of arguments but only one expression. The expression is evaluated and returned. Lambda functions can be used wherever function objects are required.

A lambda function is a small anonymous function.

A lambda function can take any number of arguments, but can only have one expression.

### Example

A lambda function that adds 10 to the number passed in as an argument, and print the result:

```
x = lambda a : a + 10  
print(x(5))
```

[Run example »](#)

Lambda functions can take any number of arguments:

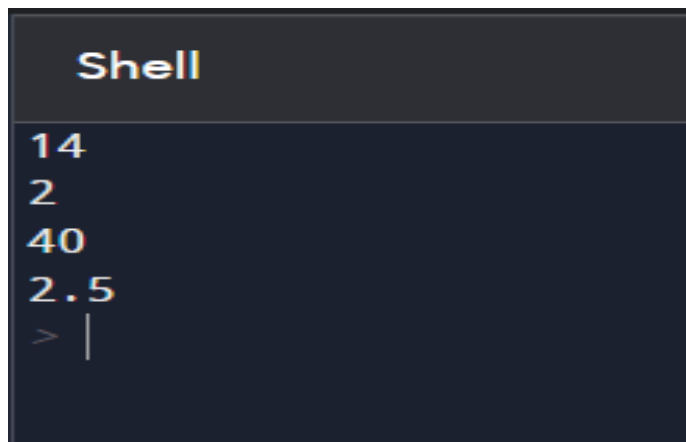


**Algorithm:**

1. Declare a function
2. perform mathematical operations
3. stop.

**Python code:**

```
sum= lambda a,b: (a+b)
print (sum (10,4))
subtraction= lambda a,b: (a-b)
print (subtraction(4,2))
product= lambda a,b: (a*b)
print (product (10,4))
division= lambda a,b: (a/b)
print (product (10,4))
```

**OUTPUT:**

```
Shell
14
2
40
2.5
> |
```

## Assignment no:15

**Problem Statement:** Write a program to implement following string operations:

a.Count b.Concat c.uppercase d.lowercase

**Theory:**

### What is String in Python?

A string is a sequence of characters.

A character is simply a symbol. For example, the English language has 26 characters.

Computers do not deal with characters, they deal with numbers (binary). Even though you may see characters on your screen, internally it is stored and manipulated as a combination of 0's and 1's.

This conversion of character to a number is called encoding, and the reverse process is decoding. ASCII and Unicode are some of the popular encoding used.

In Python, string is a sequence of Unicode character. Unicode was introduced to include every character in all languages and bring uniformity in encoding. You can [learn more about Unicode](#) from here.

### How to create a string in Python?

Strings can be created by enclosing characters inside a single quote or double quotes. Even triple quotes can be used in Python but generally used to represent multiline strings and docstrings.

### Algorithm for concatenate:

1. Accept first string
2. accept second string
3. concatenate = first string + second string
4. print concatenate
5. stop.

Python code:

```
# Python Program to Count Total Characters in a String

str1 = input("Please Enter your Own String : ")
total = 0

for i in range(len(str1)):
    total = total + 1

print("Total Number of Characters in this String = ", total)
```

```
# Python Program to Concatenate Strings

str1 = input("Please Enter the First String : ")
str2 = input("Please Enter the Second String : ")

concat1 = str1 + str2
print("The Final String After Python String Concatenation = ", concat1)

concat2 = str1 + ' ' + str2
print("The Final After String Concatenation with Space = ", concat2)
```

```
# Python Program to Convert String to Uppercase

string = input("Please Enter your Own String : ")

string1 = string.upper()

print("\nOriginal String in Lowercase = ", string)
print("The Given String in Uppercase = ", string1)
```

```
# Python Program to Convert String to Lowercase

string = input("Please Enter your Own String : ")

string1 = string.lower()

print("\nOriginal String in Uppercase = ", string)
print("The Given String in Lowercase = ", string1)
```

## OUTPUT: COUNT

```
Shell

Please Enter your Own String : 6
Total Number of Characters in this String = 1
> |
```

## CONCATE

```
Shell

Please Enter the First String : 4
Please Enter the Second String : 5
The Final String After Python String Concatenation = 45
The Final After String Concatenation with Space = 4 5
> |
```

## UPPERCASE

```
Shell

Please Enter your Own String : 6
Original String in Lowercase = 6
The Given String in Uppercase = 6
> |
```

## LOWERCASE

```
Shell

Please Enter your Own String : 7
Original String in Uppercase = 7
The Given String in Lowercase = 7
> |
```

## Assignment no:16

**Problem Statement:** Write a python program to check wheather a given string is palindrome or not.

### Theory:

This python program allows the user to enter a string. Next, we used If statement to check whether given string is equal to reverse of that string or not. If True, Palindrome string otherwise, not a palindrome string in python.

A palindrome is a string which is same read forward or backwards.

For example: "dad" is the same in forward or reverse direction. Another example is "aibohphobia" which literally means, an irritable fear of palindromes.

### Algorithm:

1. Accept a input from user.
2. for i in string  
    str1 = i + str1
3. Print string in reverse order
4. if string==str1  
    Print sting is palindrome  
    Else  
    Print sting is not palindrome
5. stop.

### Python code:

```
# Python Program to Check a Given String is Palindrome or Not

string = input("Please enter your own String : ")
str1 = ""

for i in string:
    str1 = i + str1
print("String in reverse Order : ", str1)

if(string == str1):
    print("This is a Palindrome String")
else:
    print("This is Not a Palindrome String")
```

### OUTPUT:

Shell

```
Please enter your own string: nitin
This is a Palindrome string
> |
```

## Assignment no:17

**Problem Statement:** Write a python program to print a reverse of user entered number.

### Theory:

This program reverses the given number.

### Program Explanation

1. User must first enter the value and store it in a variable n.
2. The while loop is used and the last digit of the number is obtained by using the modulus operator.
3. The last digit is then stored at the one's place, second last at the ten's place and so on.
4. The last digit is then removed by truly dividing the number with 10.
5. This loop terminates when the value of the number is 0.
6. The reverse of the number is then printed.

### Algorithm:

1. Take the value of the integer and store in a variable.
2. Using a while loop, get each digit of the number and store the reversed number in another variable.
3. Print the reverse of the number .
4. Exit.

### Python Code:

```
n=int(input("Enter number: "))
rev=0
while(n>0):
    dig=n%10
    rev=rev*10+dig
    n=n//10
print("Reverse of the number:",rev)
```

### OUTPUT:

```
Shell
Enter Number: 12345
Reverse of the number= 54321
> |
```

## Assignment no: 18

**Problem Statement:** Write a python program to read a file and display content in a file.

### Theory:

Python File Open

[< Previous](#)[Next >](#)

File handling is an important part of any web application.

Python has several functions for creating, reading, updating, and deleting files.

### File Handling

The key function for working with files in Python is the `open()` function.

The `open()` function takes two parameters; *filename*, and *mode*.

There are four different methods (modes) for opening a file:

`"r"` - Read - Default value. Opens a file for reading, error if the file does not exist

`"a"` - Append - Opens a file for appending, creates the file if it does not exist

`"w"` - Write - Opens a file for writing, creates the file if it does not exist

`"x"` - Create - Creates the specified file, returns an error if the file exists

In addition you can specify if the file should be handled as binary or text mode

`"t"` - Text - Default value. Text mode

`"b"` - Binary - Binary mode (e.g. images)

### Syntax

To open a file for reading it is enough to specify the name of the file:

```
f = open("demofile.txt")
```

The code above is the same as:

```
f = open("demofile.txt", "rt")
```

### Program Explanation

1. User must enter a file name.
2. The file is opened using the open() function in the read mode.
3. The readline() outside the while loop is used to read the first line of the file.
4. Inside the loop, the first line is first printed and then the remaining lines are read and subsequently printed.
5. This continues this the end of file.
6. The file is then closed.

### Algorithm:

1. Take the file name from the user.
2. Use readline() function for the first line first.
3. Use a while loop to print the first line and then read the remaining lines and print it till the end of file.
4. Exit.

### Python code:

```
a=str(input("Enter the name of the file with .txt extension:"))
file2=open(a,'r')
line=file2.readline()
while(line!=""):
    print(line)
    line=file2.readline()
file2.close()
```

### OUTPUT:

```
Output:
Enter the name of the file with .txt extension: data2.txt
This programming language is
Python.
```



## Assignment no:19

**Problem Statement:** Write a python program which uses import packages and statements.

### Theory:

#### Packages in Python

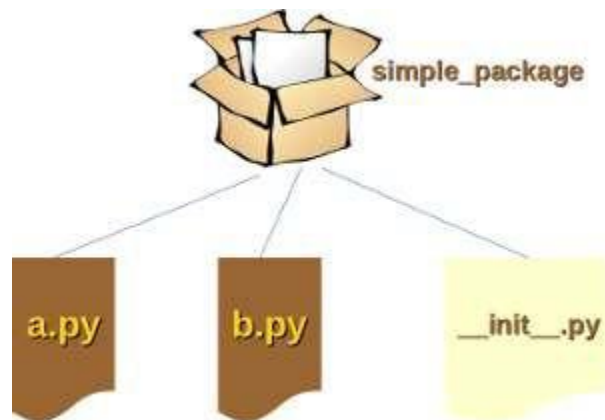
We learned that modules are files containing Python statements and definitions, like function and class definitions. We will learn in this chapter how to bundle multiple modules together to form a package.

A package is basically a directory with Python files and a file with the name `_init_.py`. This means that every directory inside of the Python path, which contains a file named `_init_.py`, will be treated as a package by Python. It's possible to put several modules into a Package.

Packages are a way of structuring Python's module namespace by using "dotted module names". A.B stands for a submodule named B in a package named A. Two different packages like P1 and P2 can both have modules with the same name, let's say A, for example. The submodule A of the package P1 and the submodule A of the package P2 can be totally different.

A package is imported like a "normal" module.

We will start this chapter with a simple example



#### A simple Package:

We will demonstrate with a very simple example how to create a package with some Python modules.

First of all, we need a directory. The name of this directory will be the name of the package, which we want to create. We will call our package "simple\_package". This directory needs to contain a file with the name "`_init_.py`". This file can be empty, or it can contain valid Python code. This code will be executed when a package will be imported, so it can be used to initialize a package, e.g. to make sure that some other modules are imported or some values set. Now we can put into this directory all the Python files which will be the submodules of our module.

We create two simple files `a.py` and `b.py` just for the sake of filling the package with

modules.

The content of a.py:

```
def bar():  
    print("Hello, function 'bar' from module 'a' calling")
```

The content of b.py:

```
def foo():  
    print("Hello, function 'foo' from module 'b' calling")
```

We will also add an empty file with the name `__init__.py` inside of `simple_package` directory.

Let's see what's happening, when we import `simple_package` from the interactive Python shell, assuming that the directory `simple_package` is either in the directory from which you call the shell or that it is contained in the search path or environment variable "PYTHONPATH" (from your operating system):

```
>>> import simple_package  
>>>  
>>> simple_package  
<module 'simple_package' from '/home/bernd/Dropbox  
(Bodenseo)/websites/python-course.eu/examples/simple_package/__init__.py'>  
>>>  
>>> simple_package/a  
Traceback (most recent call last):  
  File "<stdin>", line 1, in <module>  
NameError: name 'a' is not defined  
>>>  
>>> simple_package/b  
Traceback (most recent call last):  
  File "<stdin>", line 1, in <module>  
NameError: name 'b' is not defined
```

We can see that the package `simple_package` has been loaded but neither the module "a" nor the module "b"! We can import the modules a and b in the following way:

```
>>> from simple_package import a, b  
>>> a.bar()  
Hello, function 'bar' from module 'a' calling  
>>> b.foo()  
Hello, function 'foo' from module 'b' calling  
>>>
```

As we have seen at the beginning of the chapter, we can't access neither "a" nor "b" by solely importing simple\_package.

Yet, there is a way to automatically load these modules. We can use the file `__init__.py` for this purpose. All we have to do is add the following lines to the so far empty file `__init__.py`:

```
import simple_package.a
import simple_package.b
```

It will work now:

```
>>> import simple_package
>>>
>>> simple_package.a.bar()
Hello, function 'bar' from module 'a' calling
>>>
>>> simple_package.b.foo()
Hello, function 'foo' from module 'b' calling
```

## OUTPUT:

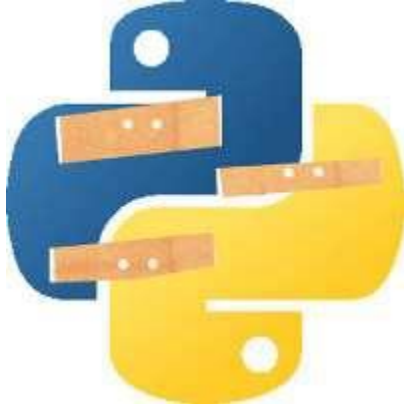
```
=====
Hello, function 'bar' from module 'a' calling
Hello, function 'foo' from module 'b' calling
>>>
=====
>>> |
```

Assignment no:20

**Problem Statement:** write a python program to handle exceptions.

**Theory:**

## Exception Handling



An exception is an error that happens during the execution of a program. Exceptions are known to non-programmers as instances that do not conform to a general rule. The name "exception" in computer science has this meaning as well: It implies that the problem (the exception) doesn't occur frequently, i.e. the exception is the "exception to the rule". Exception handling is a construct in some programming languages to handle or deal with errors automatically. Many programming languages like C++, Objective-C, PHP, Java, Ruby, Python, and many others have built-in support for exception handling.

Error handling is generally resolved by saving the state of execution at the moment the error occurred and interrupting the normal flow of the program to execute a special function or piece of code, which is known as the exception handler. Depending on the kind of error ("division by zero", "file open error" and so on) which had occurred, the error handler can "fix" the problem and the program can be continued afterwards with the previously saved data.

## Exception Handling in Python

Exceptions handling in Python is very similar to Java. The code, which harbours the risk of an exception, is embedded in a try block. But whereas in Java exceptions are caught by catch clauses, we have statements introduced by an "except" keyword in Python. It's possible to create "custom-made" exceptions: With the raise statement it's possible to force a specified exception to occur.

Let's look at a simple example. Assuming we want to ask the user to enter an integer number. If we use a input(), the input will be a string, which we have to cast into an integer. If the input has not been a valid integer, we will generate (raise) a ValueError. We show this in the following interactive session:

```
>>> n = int(input("Please enter a number: "))
Please enter a number: 23.5
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
ValueError: invalid literal for int() with base 10: '23.5'
```

With the aid of exception handling, we can write robust code for reading an integer from input:

```
while True:
    try:
        n = input("Please enter an integer: ")
        n = int(n)
        break
    except ValueError:
        print("No valid integer! Please try again ...")
print("Great, you successfully entered an integer!")
```

It's a loop, which breaks only, if a valid integer has been given.

The example script works like this:

The while loop is entered. The code within the try clause will be executed statement by statement. If no exception occurs during the execution, the execution will reach the break statement and the while loop will be left. If an exception occurs, i.e. in the casting of n, the rest of the try block will be skipped and the except clause will be executed. The raised error, in our case a ValueError, has to match one of the names after except. In our example only one, i.e. "ValueError:". After having printed the text of the print statement, the execution does another loop. It starts with a new input().

An example usage could look like this:

```
$ python integer_read.py
Please enter an integer: abc
No valid integer! Please try again ...
Please enter an integer: 42.0
No valid integer! Please try again ...
Please enter an integer: 42
Great, you successfully entered an integer!
$
```

## Multiple Except Clauses

A try statement may have more than one except clause for different exceptions. But at most one except clause will be executed.

Our next example shows a try clause, in which we open a file for reading, read a line from this file and convert this line into an integer. There are at least two possible exceptions:

- an IOError
- ValueError

Just in case we have an additional unnamed except clause for an unexpected error:

```
import sys

try:
    f = open('integers.txt')
    s = f.readline()
    i = int(s.strip())
except IOError as e:
    errno, strerror = e.args
    print("I/O error({0}): {1}".format(errno, strerror))
    # e can be printed directly without using .args:
    # print(e)
except ValueError:
    print("No valid integer in line.")
except:
    print("Unexpected error:", sys.exc_info()[0])
    raise
```

The handling of the IOError in the previous example is of special interest. The except clause for the IOError specifies a variable "e" after the exception name (IOError). The variable "e" is bound to an exception instance with the arguments stored in instance.args.

If we call the above script with a non-existing file, we get the message:

```
I/O error(2): No such file or directory
```

And if the file integers.txt is not readable, e.g. if we don't have the permission to read it, we get the following message:

```
I/O error(13): Permission denied
```

An except clause may name more than one exception in a tuple of error names, as we see in the following example:

```

try:
    f = open('integers.txt')
    s = f.readline()
    i = int(s.strip())
except (IOError, ValueError):
    print("An I/O error or a ValueError occurred")
except:
    print("An unexpected error occurred")
    raise

```

We want to demonstrate now, what happens, if we call a function within a try block and if an exception occurs inside the function call:

```

def f():
    x = int("four")

try:
    f()
except ValueError as e:
    print("got it :-) ", e)

print("Let's get on")

```

We learn from the above result that the function catches the exception:

```

got it :-)  invalid literal for int() with base 10: 'four'
Let's get on

```

We will extend our example now so that the function will catch the exception directly:

```

def f():
    try:
        x = int("four")
    except ValueError as e:
        print("got it in the function :-) ", e)

try:
    f()

```

```
except ValueError as e:
    print("got it :-) ", e)

print("Let's get on")
```

As we have expected, the exception will be caught inside of the function and not in the callers exception:

```
got it in the function :-)  invalid literal for int() with base 10: 'four'
Let's get on
```

We add now a "raise", which generates the ValueError again, so that the exception will be propagated to the caller:

```
def f():
    try:
        x = int("four")
    except ValueError as e:
        print("got it in the function :-) ", e)
        raise

try:
    f()
except ValueError as e:
    print("got it :-) ", e)

print("Let's get on")
```

We will get the following result:

```
got it in the function :-)  invalid literal for int() with base 10: 'four'
got it :-)  invalid literal for int() with base 10: 'four'
Let's get on
```

## Custom-made Exceptions

It's possible to create Exceptions yourself:

```
>>> raise SyntaxError("Sorry, my fault!")
```



```
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
SyntaxError: Sorry, my fault!
```

The best or the Pythonic way to do this, consists in defining an exception class which inherits from the Exception class. You will have to go through the chapter on "Object Oriented Programming" to fully understand the following example:

```
class MyException(Exception):
    pass

raise MyException("An exception doesn't always prove the rule!")
```

If you start this program, you will get the following result:

```
$ python3 exception_eigene_klasse.py
Traceback (most recent call last):
  File "exception_eigene_klasse.py", line 4, in <module>
    raise MyException("Was falsch ist, ist falsch!")
main .MyException: An exception doesn't always prove the rule!
```

### Clean-up Actions (try ... finally)

So far the try statement had always been paired with except clauses. But there is another way to use it as well. The try statement can be followed by a finally clause. Finally clauses are called clean-up or termination clauses, because they must be executed under all circumstances, i.e. a "finally" clause is always executed regardless if an exception occurred in a try block or not.

A simple example to demonstrate the finally clause:

```
try:
    x = float(input("Your number: "))
    inverse = 1.0 / x
finally:
    print("There may or may not have been an exception.")
print("The inverse: ", inverse)
```

Let's look at the output of the previous script, if we first input a correct number and after this a string, which is raising an error:

```
bernd@venus:~/tmp$ python finally.py
Your number: 34
```

```
There may or may not have been an exception.  
The inverse: 0.0294117647059  
bernd@venus:~/tmp$ python finally.py  
Your number: Python  
There may or may not have been an exception.  
Traceback (most recent call last):  
  File "finally.py", line 3, in <module>  
    x = float(input("Your number: "))  
ValueError: invalid literal for float(): Python  
bernd@venus:~/tmp$
```

### Combining try, except and finally

"finally" and "except" can be used together for the same try block, as can be seen the following Python example:

```
try:  
    x = float(input("Your number: "))  
    inverse = 1.0 / x  
except ValueError:  
    print("You should have given either an int or a float")  
except ZeroDivisionError:  
    print("Infinity")  
finally:  
    print("There may or may not have been an exception.")
```

The output of the previous script, if saved as "finally2.py", for various values looks like this:

```
bernd@venus:~/tmp$ python finally2.py  
Your number: 37  
There may or may not have been an exception.  
bernd@venus:~/tmp$ python finally2.py  
Your number: seven  
You should have given either an int or a float  
There may or may not have been an exception.  
bernd@venus:~/tmp$ python finally2.py
```

```
Your number: 0
Infinity
There may or may not have been an exception.
bernd@venus:~/tmp$
```

### else Clause

The try ... except statement has an optional else clause. An else block has to be positioned after all the except clauses. An else clause will be executed if the try clause doesn't raise an exception.

The following example opens a file and reads in all the lines into a list called "text":

```
import sys
file_name = sys.argv[1]
text = []
try:
    fh = open(file_name, 'r')
    text = fh.readlines()
    fh.close()
except IOError:
    print('cannot open', file_name)

if text:
    print(text[100])
```

This example receives the file name via a command line argument. So make sure that you call it properly: Let's assume that you saved this program as "exception\_test.py". In this case, you have to call it with

```
python exception_test.py integers.txt
```

If you don't want this behaviour, just change the line "file\_name = sys.argv[1]" to "file\_name = 'integers.txt'".

The previous example is nearly the same as:

```
import sys
file_name = sys.argv[1]
text = []
```

```

try:
    fh = open(file_name, 'r')
except IOError:
    print('cannot open', file_name)
else:
    text = fh.readlines()
    fh.close()

if text:
    print(text[100])

```

The main difference is that in the first case, all statements of the try block can lead to the same error message "cannot open ...", which is wrong, if `fh.close()` or `fh.readlines()` raise an error.

## OUTPUT:

```

Please enter an integer: 34
Great, you successfully entered an integer!
> |

```

## MULTIPLE EXCEPT CLAUSES:

```

got it in the function :-) invalid literal for int() with base 10: 'four'
Let's get on
> |

```

```

got it in the function :-) invalid literal for int() with base 10: 'four'
got it :-) invalid literal for int() with base 10: 'four'
Let's get on
> |

```

```

C:\Users\91797\PycharmProjects\pythonProject
I/O error(2): No such file or directory

Process finished with exit code 0

```

```

An I/O error or a ValueError occurred
> |

```

```

got it :-) invalid literal for int() with base 10: 'four'
Let's get on
> |

```

### CUSTOM MADE EXCEPTIONS:

```
Traceback (most recent call last):
  File "<string>", line 9, in <module>
__main__.MyException: An exception doesn't always prove the rule!
> |
```

**SyntaxError:** Sorry, my fault!

### CLEAN -UP ACTIONS (try....finally):

```
Your number: 34
There may or may not have been an exception.
The inverse: 0.029411764705882353
> |
```

### COMBINING TRY, EXCEPT AND FINALLY:

```
Your number: 37
There may or may not have been an exception.
> |
```

### ELSE CLAUSE:

Shell

cannot open -f

> |

Shell

cannot open -f

> |

## Experiment No. 21

Aim- Design a project for student management system for GHRCEM.

**INPUT:**

Run

Debug

Stop

Share

Save

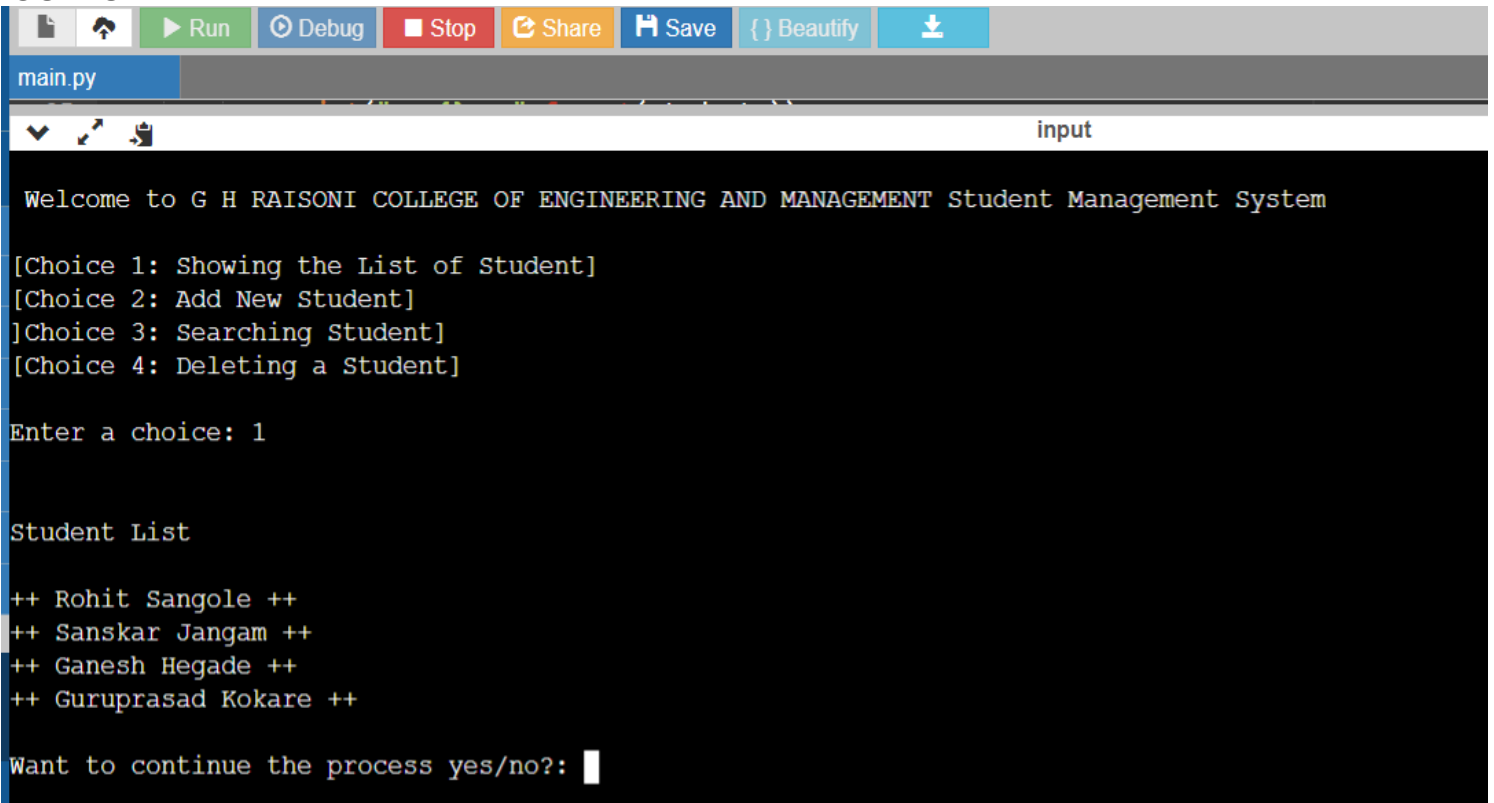
Beautify

Language Python 3

```
main.py
1 import os
2 import platform
3
4 global studentlist
5 studentlist = ["Rohit Sangole", "Sanskar Jangam", "Ganesh Hegade", "Guruprasad Kokare"]
6
7 def studentmanagement():
8
9     print("\n Welcome to G H RAISONI COLLEGE OF ENGINEERING AND MANAGEMENT Student Management System \n")
10    print("[Choice 1: Showing the List of Student]")
11    print("[Choice 2: Add New Student]")
12    print("[Choice 3: Searching Student]")
13    print("[Choice 4: Deleting a Student]\n")
14
15    try:
16        x = int(input("Enter a choice: "))
17    except ValueError:
18        exit("\nHy! This is not a Number")
19    else:
20        print("\n")
21
22    if(x==1):
23        print("Student List\n")
24        for students in studentlist:
25            print("++ {} ++".format(students))
26
27    elif(x==2):
28        studentnew = input("Enter New Student: ")
29        if(studentnew in studentlist):
30            print("\nThis Student {} Already In The Table".format(studentnew))
31        else:
32            studentlist.append(studentnew)
33            print("\n++ New Student {} Added Successfully ++\n".format(studentnew))
34            for students in studentlist:
35                print("++ {} ++".format(students))
36
37    elif(x==3):
38        studentsearching = input("Choose Student Name To Search: ")
39        if(studentsearching in studentlist):
40            print("\n++ There is a Record Found of this Student {} ++".format(studentsearching))
41        else:
42            print("\n++ There is No Record Found Of this Student {} ++".format(studentsearching))
43
44    elif(x==4):
45        studentdelete = input("Choose a Student Name To Delete: ")
46        if(studentdelete in studentlist):
```

```
main.py
20
27 elif(x==2):
28     studentnew = input("Enter New Student: ")
29     if(studentnew in studentlist):
30         print("\nThis Student {} Already In The Table".format(studentnew))
31     else:
32         studentlist.append(studentnew)
33         print("\n++ New Student {} Added Successfully ++\n".format(studentnew))
34         for students in studentlist:
35             print("++ {} ++".format(students))
36
37 elif(x==3):
38     studentsearching = input("Choose Student Name To Search: ")
39     if(studentsearching in studentlist):
40         print("\n++ There is a Record Found of this Student {} ++".format(studentsearching))
41     else:
42         print("\n++ There is No Record Found Of this Student {} ++".format(studentsearching))
43
44 elif(x==4):
45     studentdelete = input("Choose a Student Name To Delete: ")
46     if(studentdelete in studentlist):
47         studentlist.remove(studentdelete)
48         for students in studentlist:
49             print("++ {} ++".format(students))
50     else:
51         print("\n++ There is No Record Found of This Student {} ++".format(studentdelete))
52
53 elif(x < 1 or x > 4):
54     print("Please Enter Valid Choice")
55
56 studentmanagement()
57
58 def continueAgain():
59     runningagain = input("\nWant to continue the process yes/no?: ")
60     if(runningagain.lower() == 'yes'):
61         if(platform.system() == "Windows"):
62             print(os.system('cls'))
63         else:
64             print(os.system('clear'))
65         studentmanagement()
66         continueAgain()
67     else:
68         quit()
69
70 continueAgain()
71
```

## OUTPUT:



```
main.py
input

Welcome to G H RAISONI COLLEGE OF ENGINEERING AND MANAGEMENT Student Management System

[Choice 1: Showing the List of Student]
[Choice 2: Add New Student]
[Choice 3: Searching Student]
[Choice 4: Deleting a Student]

Enter a choice: 1

Student List

++ Rohit Sangole ++
++ Sanskar Jangam ++
++ Ganesh Hegade ++
++ Guruprasad Kokare ++

Want to continue the process yes/no?:
```





Welcome to G H RAISONI COLLEGE OF ENGINEERING AND MANAGEMENT Student Management System

[Choice 1: Showing the List of Student]

[Choice 2: Add New Student]

]Choice 3: Searching Student]

[Choice 4: Deleting a Student]

Enter a choice: 2

Enter New Student: Nanu bhai

++ New Student Nanu bhai Added Successfully ++

++ Rohit Sangole ++

++ Sanskar Jangam ++

++ Ganesh Hegade ++

++ Guruprasad Kokare ++

++ Nanu bhai ++

Want to continue the process yes/no?: