Control Structures PYTHON Language:

Control Structures are just a way to specify flow of control in programs. Any algorithm or program can be clearer and understood if they use self-contained modules called as logic or control structures. It basically analyzes and chooses in which direction a program flows based on certain parameters or conditions. There are three basic types of logic, or flow of control, known as:

1. Sequence logic, or sequential flow
2. Selection logic, or conditional flow
3. Iteration logic, or repetitive flow

1) **Sequential** – this one involves executing all the codes in the order in which they have been written.
2) **Selection** – This involves decision making.
3) **Iteration** – Execution of Statements unit a condition is met.

PYTHON IF Statement.

```
if test expression:
    statement(s)
```

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

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

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

Python if Statement Flowchart

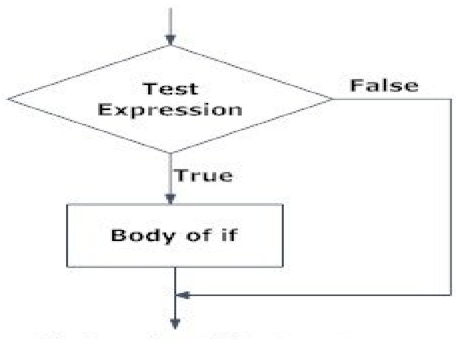


Fig: Operation of if statement

Flowchart of if statement in Python programming

Python Indentation

Indentation refers to the spaces at the beginning of a code line.

Where in other programming languages the indentation in code is for readability only, the indentation in Python is very important.

Python uses indentation to indicate a block of code.

Example

```
if 5 > 2:
  print("Five is greater than two!")
```

Python will give you an error if you skip the indentation:

Example

Syntax Error:

```
if 5 > 2:
print("Five is greater than two!")
```

The number of spaces is up to you as a programmer, the most common use is four, but it has to be at least one.

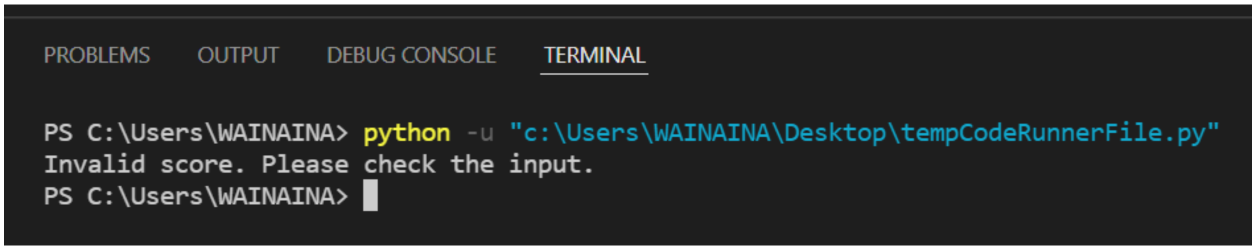
Example

```
if 5 > 2:
 print("Five is greater than two!")
if 5 > 2:
        print("Five is greater than two!")
```

EXAMPLE1: A Program to read theory and practical Marks and Comment Appropriately:

```
score_theory = 50
score_practical = 55
if(score_theory + score_practical >= 100):
    print("Invalid score. Please check the input.")
```

OUTPUT:



```
PS C:\Users\WAINAINA> python -u "c:\Users\WAINAINA\Desktop\tempCodeRunnerFile.py"
Invalid score. Please check the input.
PS C:\Users\WAINAINA>
```

PYTHON IF Else Statement.

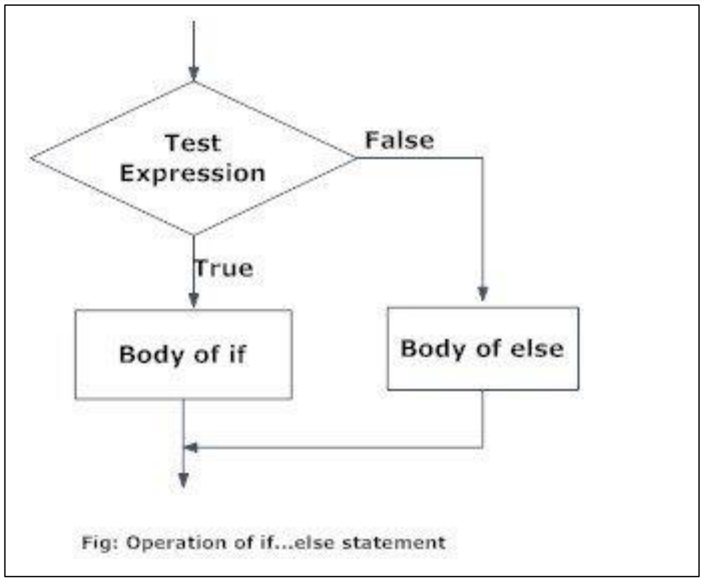
Syntax of if...else

```
if test expression:
    Body of if
else:
    Body of else
```

The if..else statement evaluates test expression and will execute the body of if only when the test condition is True.

If the condition is False, the body of else is executed. Indentation is used to separate the blocks.

Python if..else Flowchart



Flowchart of if...else statement in Python

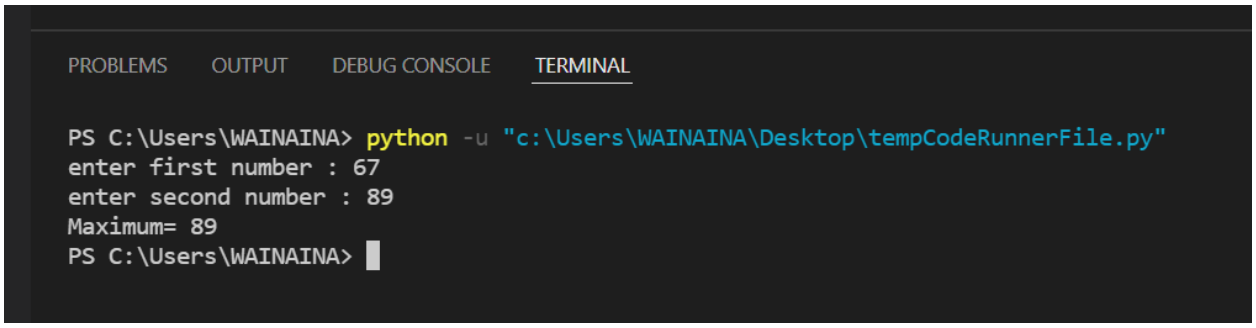
EXAMPLE2: A Program to read two numbers and test the maximum:

```
# This program to test maximum value between two values entered

a=int(input("enter first number : "))

b=int(input("enter second number : "))
# Add two numbers
if (a>b):
 print('Maximum=',a)
else:
 print('Maximum=',b)
```

OUTPUT:



```
PS C:\Users\WAINAINA> python -u "c:\Users\WAINAINA\Desktop\tempCodeRunnerFile.py"
enter first number : 67
enter second number : 89
Maximum= 89
PS C:\Users\WAINAINA>
```

PYTHON IF Else if else Statement.

```
if test expression:
    Body of if
elif test expression:
    Body of elif
else:
    Body of else
```

The elif is short for else if. It allows us to check for multiple expressions. If the condition for if is False, it checks the condition of the next elif block and so on. If all the conditions are False, the body of else is executed.

Only one block among the several if...elif...else blocks is executed according to the condition.

The if block can have only one else block. But it can have multiple elif blocks.

Flowchart of if...elif...else

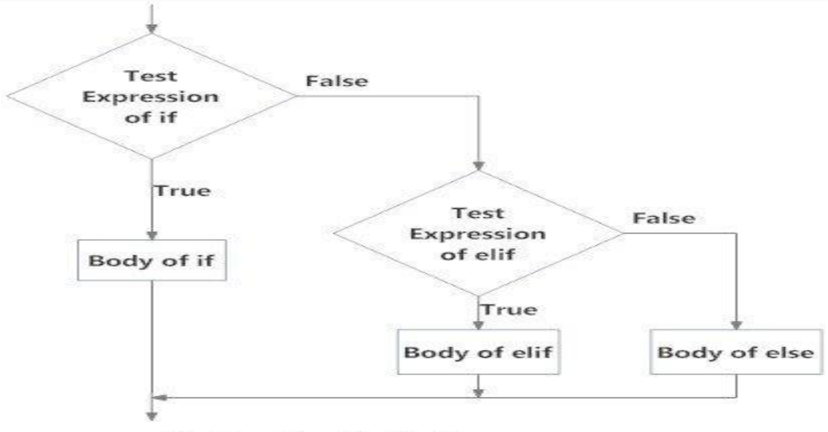


Fig: Operation of if...elif...else statement

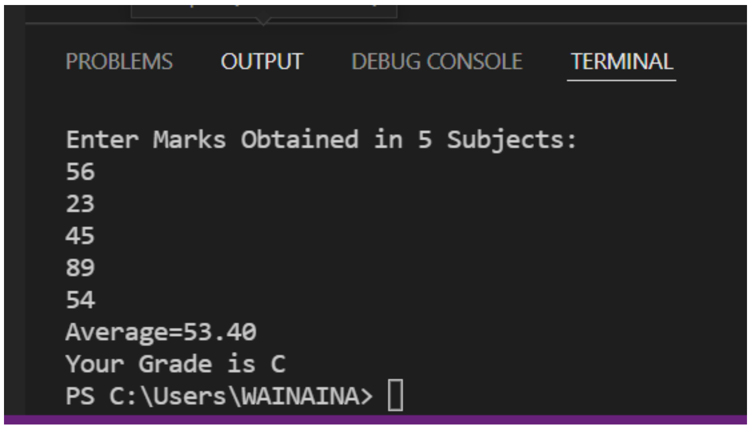
Flowchart of if...else if....else statement in Python

EXAMPLE3: A Program to grade a student appropriately:

```
print("Enter Marks Obtained in 5 Subjects: ")
markOne = int(input())
markTwo = int(input())
markThree = int(input())
markFour = int(input())
markFive = int(input())

tot = markOne+markTwo+markThree+markFour+markFive
avg = tot/5
print("Average=%.2f"%avg)
if avg>=70 and avg<=100:
    print("Your Grade is A")
elif avg>=60 and avg<=69:
    print("Your Grade is B")
elif avg>=50 and avg<=59:
    print("Your Grade is C")
elif avg>=40 and avg<=49:
    print("Your Grade is D")
elif avg>=0 and avg<=39:
    print("Your Grade is F")
else:
    print("Invalid Input!")
```

OUTPUT:



```
Enter Marks Obtained in 5 Subjects:
56
23
45
89
54
Average=53.40
Your Grade is C
PS C:\Users\WAINAINA> 
```

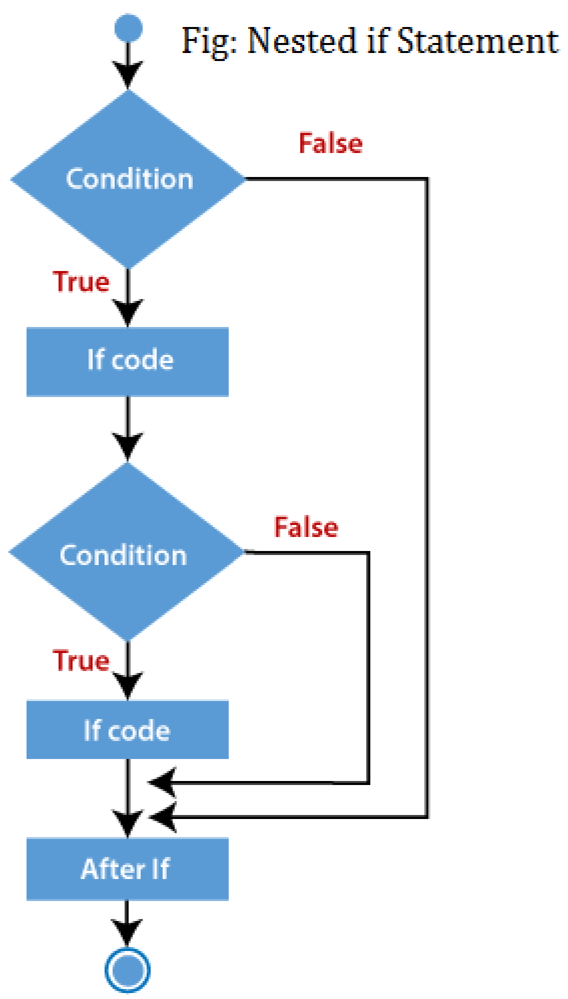
PYTHON Nested If Statement.

We can have a if...elif...else statement inside another if...elif...else statement. This is called nesting in computer programming.

Any number of these statements can be nested inside one another. Indentation is the only way to figure out the level of nesting.

```
if (condition) {
//code to be executed if condition is true
if (condition) {
//code to be executed if condition is true
}
}
```

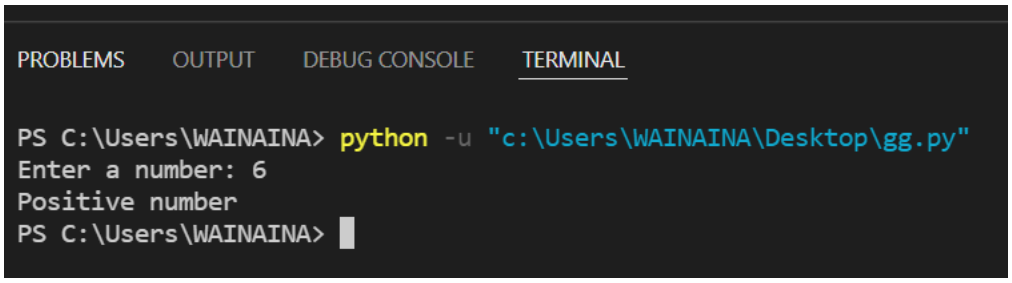
Flowchart



EXAMPLE4: A In this program, we input a number check if the number is positive or negative or zero and display an appropriate message.

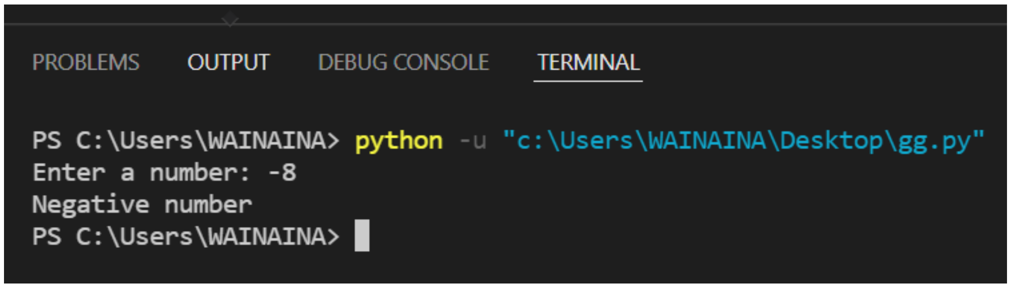
```python
num = float(input("Enter a number: "))
if num >= 0:
    if num == 0:
        print("Zero")
    else:
        print("Positive number")
else:
    print("Negative number")
```

OUTPUT 1:



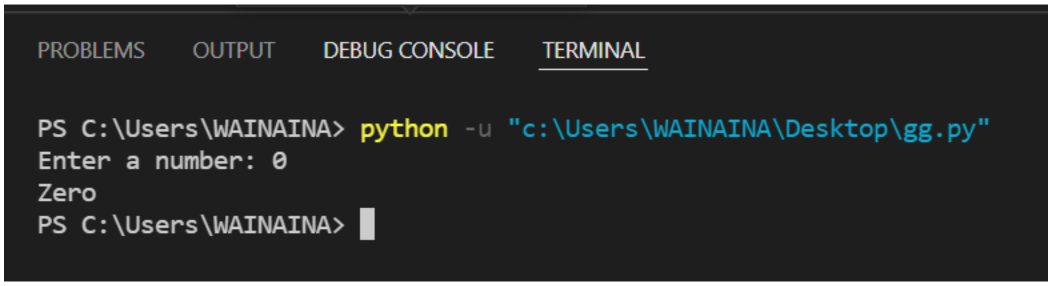
```
PS C:\Users\WAINAINA> python -u "c:\Users\WAINAINA\Desktop\gg.py"
Enter a number: 6
Positive number
PS C:\Users\WAINAINA>
```

OUTPUT 2:



```
PS C:\Users\WAINAINA> python -u "c:\Users\WAINAINA\Desktop\gg.py"
Enter a number: -8
Negative number
PS C:\Users\WAINAINA>
```

OUTPUT 3:



```
PS C:\Users\WAINAINA> python -u "c:\Users\WAINAINA\Desktop\gg.py"
Enter a number: 0
Zero
PS C:\Users\WAINAINA>
```

Loops in PYTHON:

1) For Loop:

A for loop is used for iterating over a sequence (that is either a list, a tuple, a dictionary, a set, or a string).

This is less like the for keyword in other programming languages, and works more like an iterator method as found in other object-orientated programming languages.

Syntax

```
for val in sequence:
    loop body
```

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.

Flowchart of for Loop

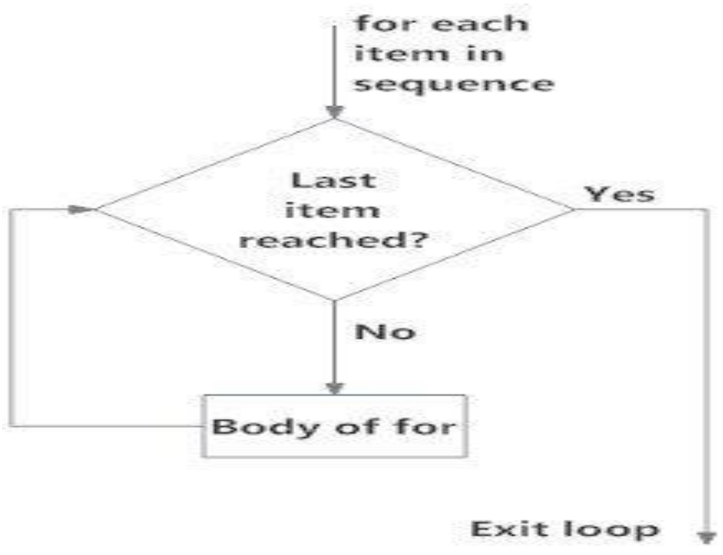


Fig: operation of for loop

Flowchart of for Loop in Python

EXAMPLE5: Program to find the sum of all numbers stored in a list

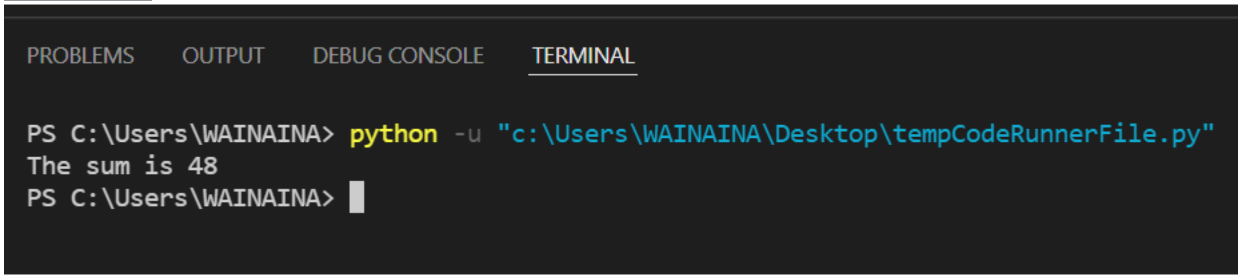
```python
# List of numbers
numbers = [6, 5, 3, 8, 4, 2, 5, 4, 11]

# variable to store the sum
sum = 0

# iterate over the list
for val in numbers:
    sum = sum+val

print("The sum is", sum)
```

OUTPUT:

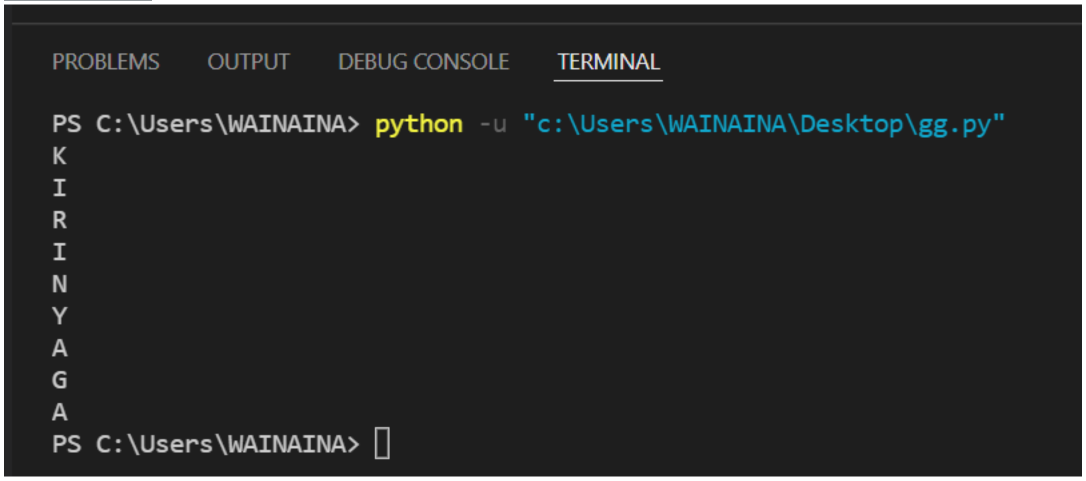


```
PS C:\Users\WAINAINA> python -u "c:\Users\WAINAINA\Desktop\tempCodeRunnerFile.py"
The sum is 48
PS C:\Users\WAINAINA> 
```

EXAMPLE6: Program Iterating string using for loop.

```python
str = "KIRINYAGA"
for i in str:
    print(i)
```

OUTPUT:



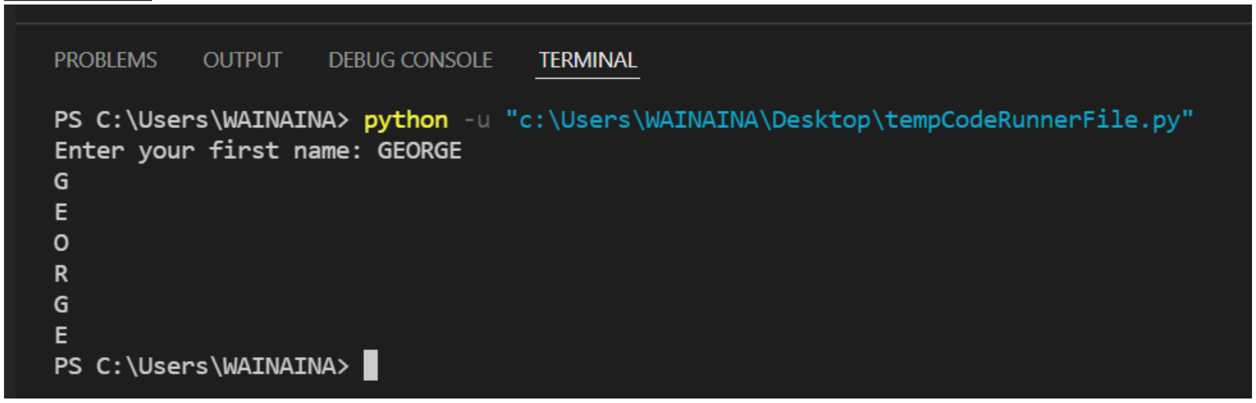
```
PS C:\Users\WAINAINA> python -u "c:\Users\WAINAINA\Desktop\gg.py"
K
I
R
I
N
Y
A
G
A
PS C:\Users\WAINAINA> 
```

EXAMPLE7: Program iterating string entered by the user using for loop.

To read a string from console as input to your Python program, you can use input () function. Input () can take an argument to print a message to the console, so that you can prompt the user and let him/her know what you are expecting.

```
#read string from user
str = input('Enter your first name: ')
for i in str:
    print(i)
```

OUTPUT:



```
PS C:\Users\WAINAINA> python -u "c:\Users\WAINAINA\Desktop\tempCodeRunnerFile.py"
Enter your first name: GEORGE
G
E
O
R
G
E
PS C:\Users\WAINAINA> 
```

For loop Using range() function

The range() function

The **range()** function is used to generate the sequence of the numbers. If we pass the range(10), it will generate the numbers from 0 to 9. The syntax of the range() function is given below.

Syntax:

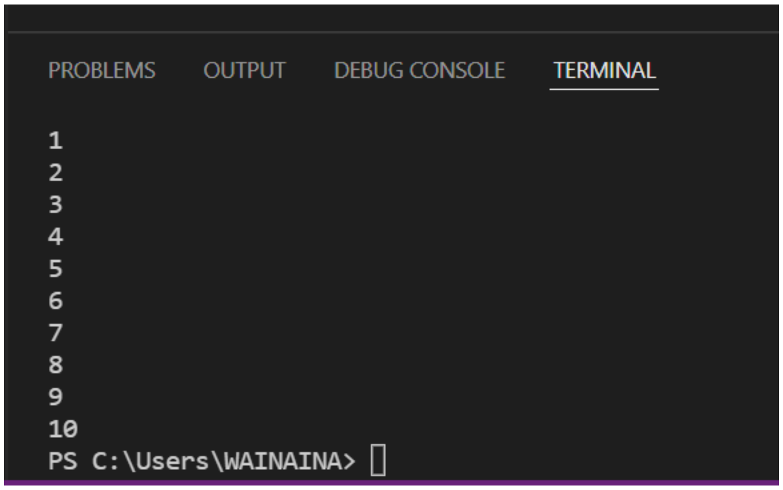
range(start,stop,step size)

- The start represents the beginning of the iteration.
- The stop represents that the loop will iterate till stop-1. The **range(1,5)** will generate numbers 1 to 4 iterations. It is optional.
- The step size is used to skip the specific numbers from the iteration. It is optional to use. By default, the step size is 1. It is optional.

EXAMPLE8: Program to print 1 to 10 using for loop.

```python
for i in range(11):
    print(i)
```

OUTPUT:

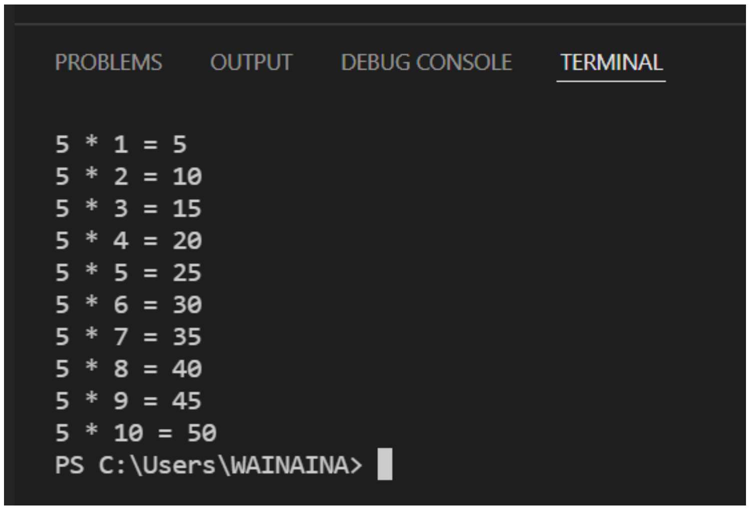


```
1
2
3
4
5
6
7
8
9
10
PS C:\Users\WAINAINA> 
```

EXAMPLE9: Program to print table of given number.

```python
n = int(input("Enter the number "))
for i in range(1,11):
    c = n*i
    print(n,"*",i,"=",c)
```

OUTPUT:

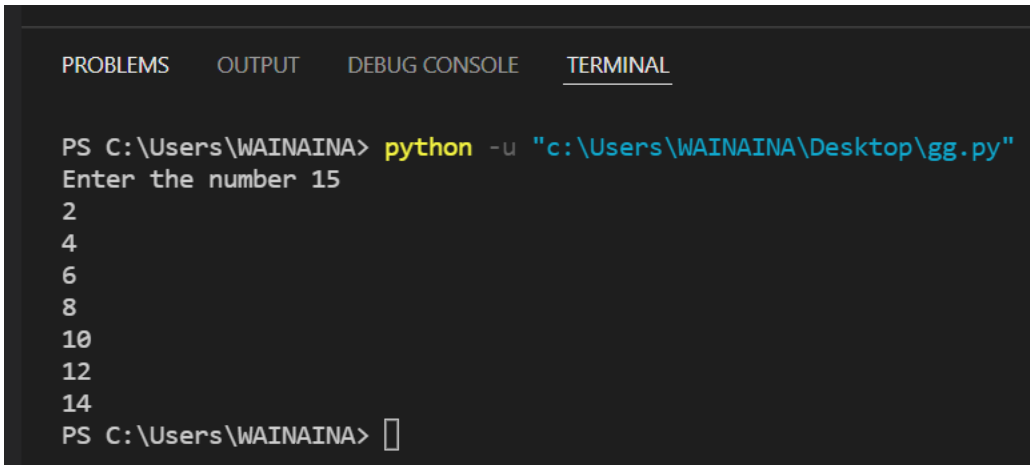


```
5 * 1 = 5
5 * 2 = 10
5 * 3 = 15
5 * 4 = 20
5 * 5 = 25
5 * 6 = 30
5 * 7 = 35
5 * 8 = 40
5 * 9 = 45
5 * 10 = 50
PS C:\Users\WAINAINA> 
```

EXAMPLE10: Program to print even number using step size in range().

```python
n = int(input("Enter the number "))
for i in range(2,n,2):
    print(i)
```

OUTPUT:



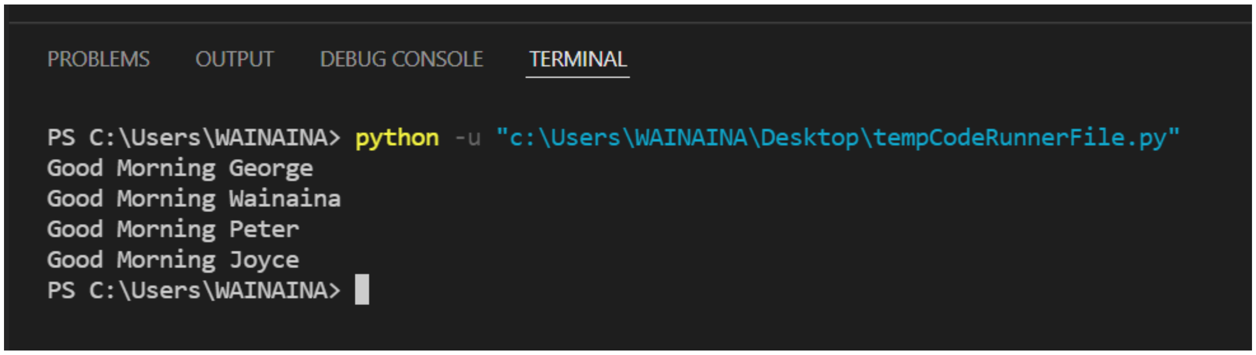
```
PS C:\Users\WAINAINA> python -u "c:\Users\WAINAINA\Desktop\gg.py"
Enter the number 15
2
4
6
8
10
12
14
PS C:\Users\WAINAINA>
```

We can also use the range() function with sequence of numbers. The len() function is combined with range() function which iterate through a sequence using indexing.

EXAMPLE11: Program to print string using len() and range().

```python
list = ['George','Wainaina','Peter','Joyce']
for i in range(len(list)):
    print("Good Morning",list[i])
```

OUTPUT:



```
PS C:\Users\WAINAINA> python -u "c:\Users\WAINAINA\Desktop\tempCodeRunnerFile.py"
Good Morning George
Good Morning Wainaina
Good Morning Peter
Good Morning Joyce
PS C:\Users\WAINAINA>
```

2) Nested for loop in python

Python allows us to nest any number of for loops inside a **for** loop. The inner loop is executed n number of times for every iteration of the outer loop. The syntax is given below.

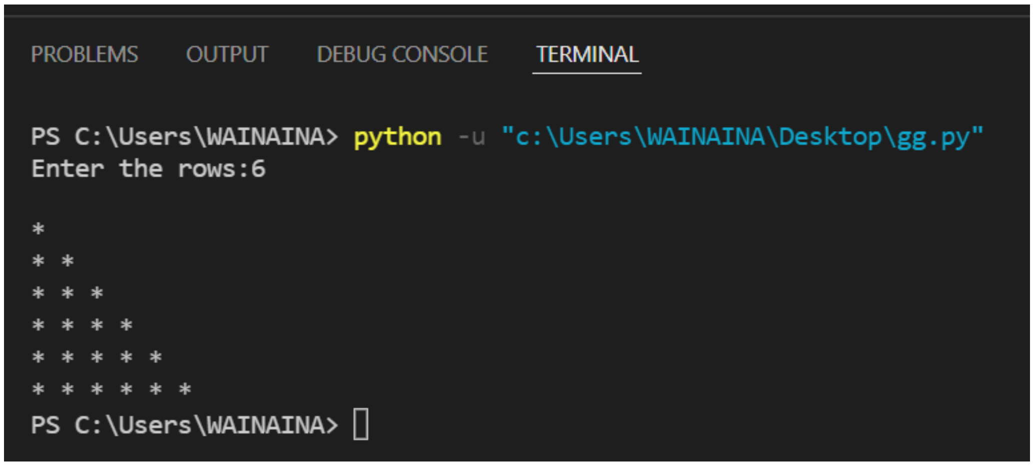
Syntax

```
for iterating_var1 in sequence:  #outer loop
    for iterating_var2 in sequence:  #inner loop
        #block of statements
#Other statements
```

EXAMPLE12: Program to print * in a pattern using nested for loop.

```python
# User input for number of rows
rows = int(input("Enter the rows:"))
# Outer loop will print number of rows
for i in range(0,rows+1):
# Inner loop will print number of Astrisk
    for j in range(i):
        print("*",end = '')
    print()
```

OUTPUT:



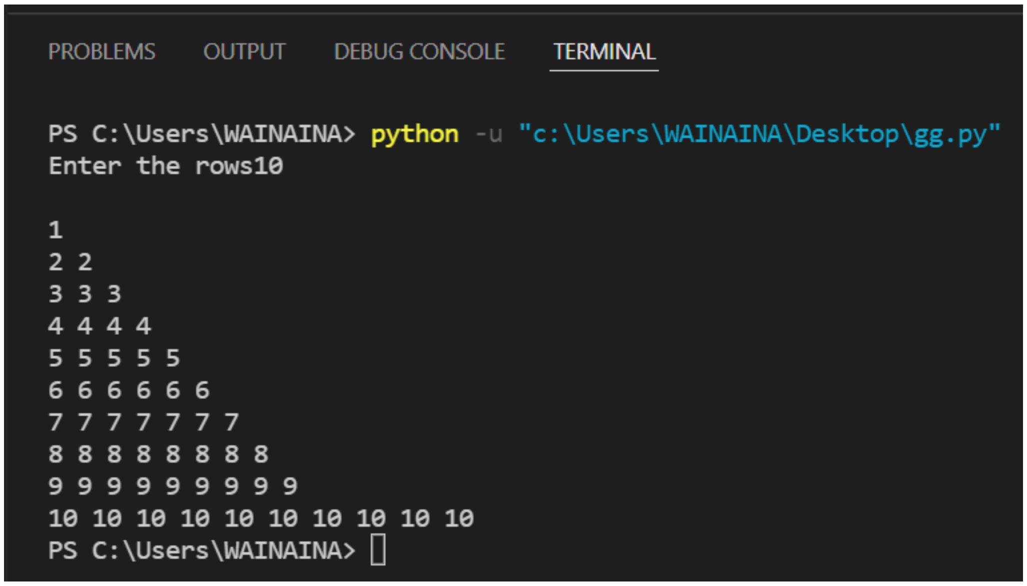
```
PS C:\Users\WAINAINA> python -u "c:\Users\WAINAINA\Desktop\gg.py"
Enter the rows:6

*
* *
* * *
* * * *
* * * * *
* * * * * *
PS C:\Users\WAINAINA> 
```

EXAMPLE13: Program to number pyramid

```
rows = int(input("Enter the rows"))
for i in range(0,rows+1):
    for j in range(i):
        print(i,end = ' ')
    print()
```

OUTPUT:



```
PS C:\Users\WAINAINA> python -u "c:\Users\WAINAINA\Desktop\gg.py"
Enter the rows10

1
2 2
3 3 3
4 4 4 4
5 5 5 5 5
6 6 6 6 6 6
7 7 7 7 7 7 7
8 8 8 8 8 8 8 8
9 9 9 9 9 9 9 9 9
10 10 10 10 10 10 10 10 10 10
PS C:\Users\WAINAINA>
```

3) While Loop:

The Python while loop allows a part of the code to be executed until the given condition returns false. It is also known as a pre-tested loop.

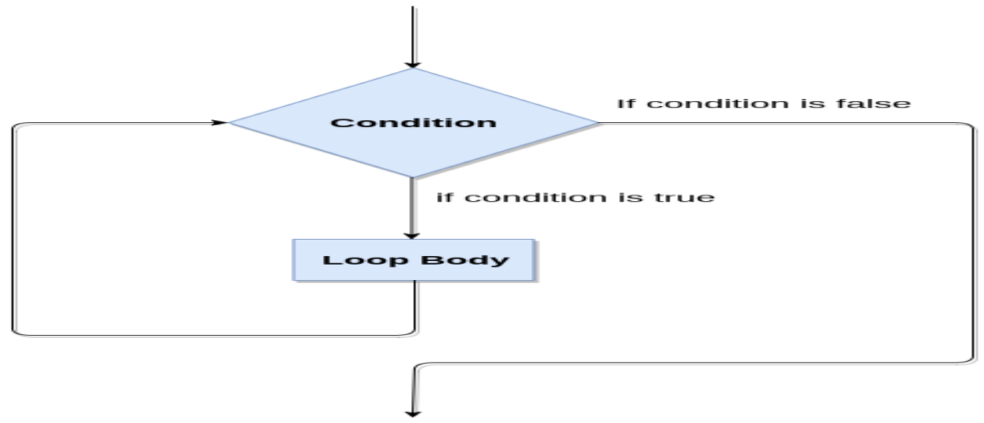
It can be viewed as a repeating if statement. When we don't know the number of iterations then the while loop is most effective to use.

The syntax is given below.

```
while expression:
  statements
```

Here, the statements can be a single statement or a group of statements. The expression should be any valid Python expression resulting in true or false. The true is any non-zero value and false is 0.

While loop Flowchart



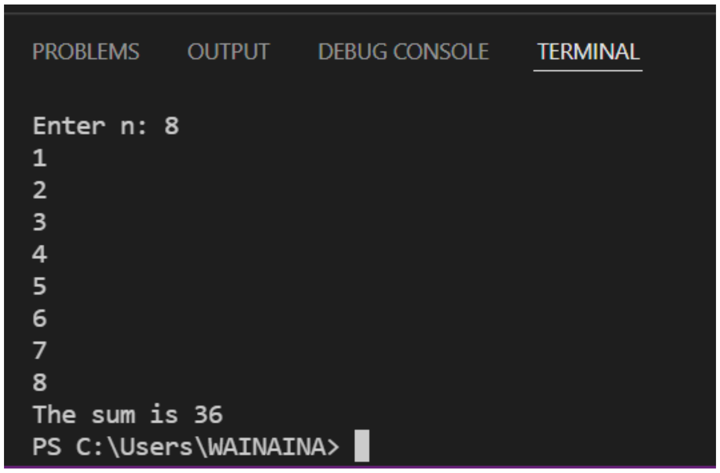
EXAMPLE14: Program to add natural numbers up to the number you will specify at run time.

```python
# To take input from the user,
n = int(input("Enter n: "))
# initialize sum and counter
sum = 0
i = 1

while i <= n:
    sum = sum + i
    print(i)
    i = i+1    # update counter

# print the sum
print("The sum is", sum)
```

OUTPUT:

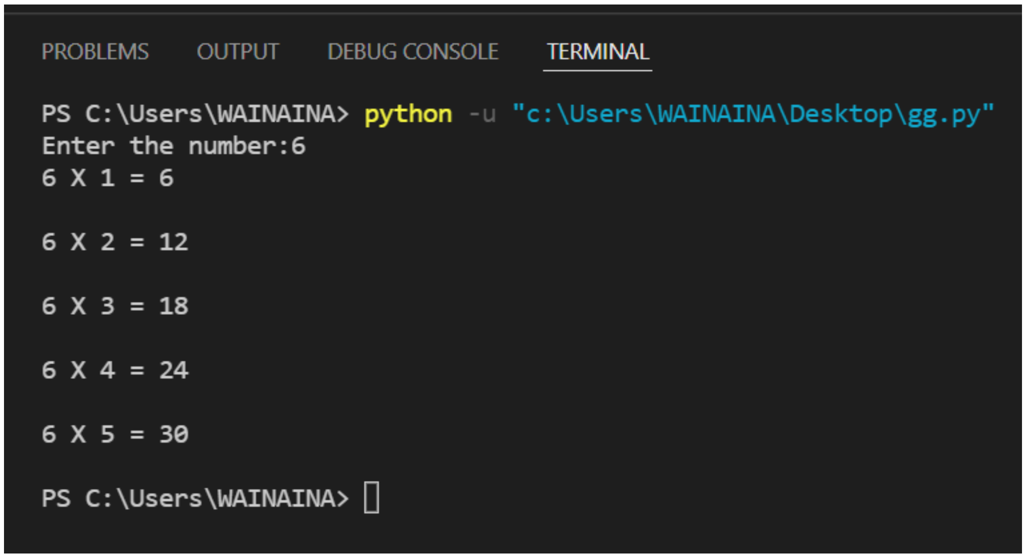


```
Enter n: 8
1
2
3
4
5
6
7
8
The sum is 36
PS C:\Users\WAINAINA>
```

EXAMPLE15: Program to print table of given numbers.

```python
i=1
number=0
b=9
number = int(input("Enter the number:"))
while i<=5:
    print("%d X %d = %d \n"%(number,i,number*i))
    i = i+1
```

OUTPUT:



```
PS C:\Users\WAINAINA> python -u "c:\Users\WAINAINA\Desktop\gg.py"
Enter the number:6
6 X 1 = 6

6 X 2 = 12

6 X 3 = 18

6 X 4 = 24

6 X 5 = 30

PS C:\Users\WAINAINA> 
```

Loop Control Statements

We can change the normal sequence of **while** loop's execution using the loop control statement. When the while loop's execution is completed, all automatic objects defined in that scope are demolished. Python offers the following control statement to use within the while loop.

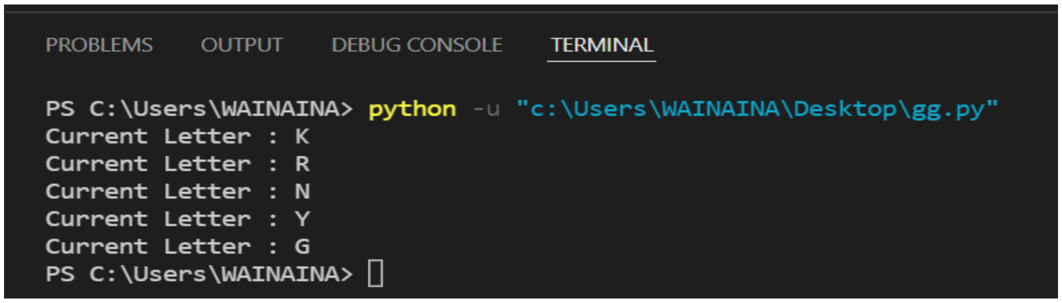
1. Continue Statement - When the continue statement is encountered, the control transfer to the beginning of the loop. Let's understand the following example.

EXAMPLE16: Program to implement Continue statement.

```
# prints all letters except 'I' and 'A'
i = 0
str1 = 'KIRINYAGA'

while i < len(str1):
    if str1[i] == 'I' or str1[i] == 'A':
        i += 1
        continue
    print('Current Letter :', str1[i])
    i += 1
```

OUTPUT:



```
PS C:\Users\WAINAINA> python -u "c:\Users\WAINAINA\Desktop\gg.py"
Current Letter : K
Current Letter : R
Current Letter : N
Current Letter : Y
Current Letter : G
PS C:\Users\WAINAINA> 
```

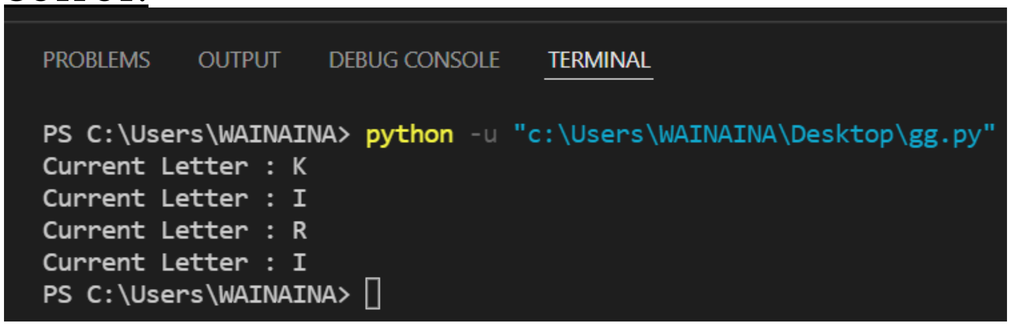
2. Break Statement - When the break statement is encountered, it brings control out of the loop.

EXAMPLE17: Program to implement Break statement.

```
# prints all letters but terminates after encountering 'N'
i = 0
str1 = 'KIRINYAGA'

while i < len(str1):
    if str1[i] == 'N':
        i += 1
        break
    print('Current Letter :', str1[i])
    i += 1
```

OUTPUT:



```
PS C:\Users\WAINAINA> python -u "c:\Users\WAINAINA\Desktop\gg.py"
Current Letter : K
Current Letter : I
Current Letter : R
Current Letter : I
PS C:\Users\WAINAINA> 
```

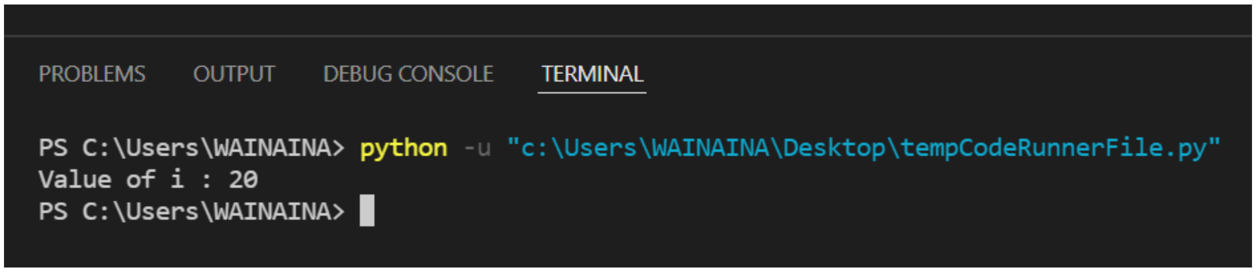
3. Pass Statement - The pass statement is used to declare the empty loop. It is also used to define empty class, function, and control statement.

EXAMPLE17: Program to implement Pass statement.

```
# An empty loop
str1 = 'SOFTWARE_ENGINEERING'
i = 0

while i < len(str1):
    i += 1
    pass
print('Value of i :', i)
```

OUTPUT:



```
PS C:\Users\WAINAINA> python -u "c:\Users\WAINAINA\Desktop\tempCodeRunnerFile.py"
Value of i : 20
PS C:\Users\WAINAINA> 
```