

Flow of Control

- The order of execution of the statements in a program is known as flow of control. The flow of control can be implemented using control structures.
- Python supports two types of control structures—selection and repetition.
- **Selection :**
- A decision involves selecting from one of the two or more possible options. In programming, this concept of decision making or selection is implemented with the help of if..else statement.
- The syntax of if statement is:
if condition:
 statement(s)
- age = int(input("Enter your age "))
if age >= 18:
 print("Eligible to vote")

Flow of Control

- A variant of if statement called if..else statement allows us to write two alternative paths and the control condition determines which path gets executed. The syntax for if..else statement is as follows.

```
if condition:  
    statement(s)  
else:  
    statement(s)
```

Let us now modify the example on voting with the condition that if the age entered by the user is greater than 18, then to display that the user is eligible to vote. Otherwise display that the user is not eligible to vote.

```
age = int(input("Enter your age: "))  
if age >= 18:  
    print("Eligible to vote")  
else:  
    print("Not eligible to vote")
```

Flow of Control

- Many a times there are situations that require multiple conditions to be checked and it may lead to many alternatives.
- In such cases we can chain the conditions using if..elif (elif means else..if).
- The syntax for a selection structure using elif is as shown below.
if condition:
 statement(s)
elif condition:
 statement(s)
elif condition:
 statement(s)
else:
 statement(s)

Flow of Control

- Check whether a number is positive, negative, or zero.

```
number = int(input("Enter a number: "))
```

```
if number > 0:
```

```
    print("Number is positive")
```

```
elif number < 0:
```

```
    print("Number is negative")
```

```
else:
```

```
    print("Number is zero")
```

- Q. Display the appropriate message as per the colour of signal at the road crossing.

Flow of Control

```
➤ signal = input("Enter the colour: ")  
if signal == "red" or signal == "RED":  
    print("STOP")  
elif signal == "orange" or signal == "ORANGE":  
    print("Be Slow")  
elif signal == "green" or signal == "GREEN":  
    print("Go!")
```

Number of elif is dependent on the number of conditions to be checked. If the first condition is false, then the next condition is checked, and so on. If one of the conditions is true, then the corresponding indented block executes, and the if statement terminates.

Flow of Control

- Let us write a program to create a simple calculator to perform basic arithmetic operations on two numbers.
The program should do the following:
- - Accept two numbers from the user.
 - Ask user to input any of the operator (+, -, *, /).An error message is displayed if the user enters anything else.
- Display only positive difference in case of the operator "-".
- Display a message “Please enter a value other than 0” if the user enters the second number as 0 and operator ‘/’ is entered.

Flow of Control

```
➤ val1 = float(input("Enter value 1: "))
val2 = float(input("Enter value 2: "))
op = input("Enter any one of the operator (+,-,*,/): ")
if op == "+":
    result = val1 + val2
elif op == "-":
    if val1 > val2:
        result = val1 - val2
else:
    result = val2 - val1
elif op == "*":
    result = val1 * val2
elif op == "/":
    if val2 == 0:
        print("Error! Division by zero is not allowed. Program terminated")
else:
    result = val1/val2
else:
    print("Wrong input,program terminated")
print("The result is ",result)
```

Indentation

- In most programming languages, the statements within a block are put inside curly brackets. However, Python uses indentation for block as well as for nested block structures.
- Leading whitespace (spaces and tabs) at the beginning of a statement is called indentation.
- Leading whitespace (spaces and tabs) at the beginning of a statement is called indentation.
- The interpreter checks indentation levels very strictly and throws up syntax errors if indentation is not correct. It is a common practice to use a single tab for each level of indentation.
-

repetition

- repetition is also called iteration .
- Repetition of a set of statements in a program is made possible using looping constructs.
- Ex. payment of electricity bill, which is done every month.
- **Write a program to print the first five natural numbers.**
- print(1)
print(2)
print(3)
print(4)
print(5)
What should we do if we are asked to print the first 10,000 natural numbers?
- Writing a program having a loop or repetition is a better solution.
The program logic is given below:
 1. Take a variable, say count, and set its value to 1.
 2. Print the value of count.
 3. Increment the variable (count += 1)

repetition

- repetition is also called iteration .
 - Repetition of a set of statements in a program is made possible using looping constructs.
 - Ex. payment of electricity bill, which is done every month.
 - **Write a program to print the first five natural numbers.**
 - - print(1)
 - print(2)
 - print(3)
 - print(4)
 - print(5)
- What should we do if we are asked to print the first 100,000 natural numbers?
- Writing a program having a loop or repetition is a better solution.
The program logic is given below:
 1. Take a variable, say count, and set its value to 1.
 2. Print the value of count.
 3. Increment the variable (count += 1)
 - 4. Repeat steps 2 and 3 as long as count has a value less than or equal to 100,000 (count <= 100,000).

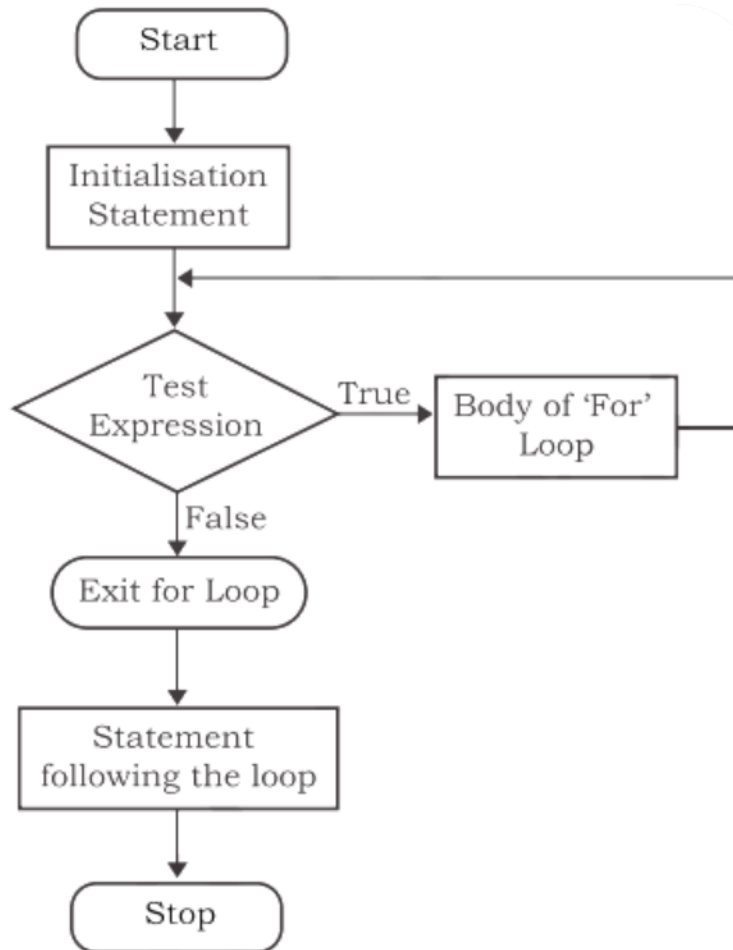
repetition

- Looping constructs provide the facility to execute a set of statements in a program repetitively, based on a condition.
- The statements in a loop are executed again and again as long as particular logical condition remains true.
- This condition is checked based on the value of a variable called the loop's control variable.
- It is the responsibility of the programmer to ensure that this condition eventually does become false so that there is an exit condition and it does not become an infinite loop.
- There are two looping constructs in Python - for and while.

The 'For' Loop

- The for statement is used to iterate over a range of values or a sequence. The for loop is executed for each of the items in the range.
- With every iteration of the loop, the control variable checks whether each of the values in the range have been traversed or not.
- When all the items in the range are exhausted, the statements within loop are not executed; the control is then transferred to the statement immediately following the for loop.
- While using for loop, it is known in advance the number of times the loop will execute.

The 'For' Loop



The 'For' Loop

➤ *Syntax of the For Loop*

➤ for <control-variable> in <sequence/ items in range>:
 <statements inside body of the loop>

➤ #Print the given sequence of numbers using for loop
count = [10,20,30,40,50]
for num in count:
 print(num)

➤ Program to print even numbers in a given sequence using for loop.

```
numbers = [1,2,3,4,5,6,7,8,9,10]
for num in numbers:
    if (num % 2) == 0:
        print(num,'is an even Number')
```

The 'For' Loop

➤ *The Range() Function*

➤ The range() is a built-in function in Python. Syntax of range() function is:

➤
`range([start], stop[, step])`

➤ It is used to create a list containing a sequence of integers from the given start value upto stop value, with a difference of the given step value.

➤ The start and step parameters are optional .

➤ If start value is not specified, by default the list starts from 0. If step is also not specified, by default the value increases by 1 in each iteration.

➤ All parameters of range() function must be integers.

The 'For' Loop

➤ *The Range() Function*

➤ The range() is a built-in function in Python. Syntax of range() function is:

➤ `range([start], stop[, step])`

➤ It is used to create a list containing a sequence of integers from the given start value upto stop value, with a difference of the given step value.

➤ The start and step parameters are optional .

➤ If start value is not specified, by default the list starts from 0. If step is also not specified, by default the value increases by 1 in each iteration.

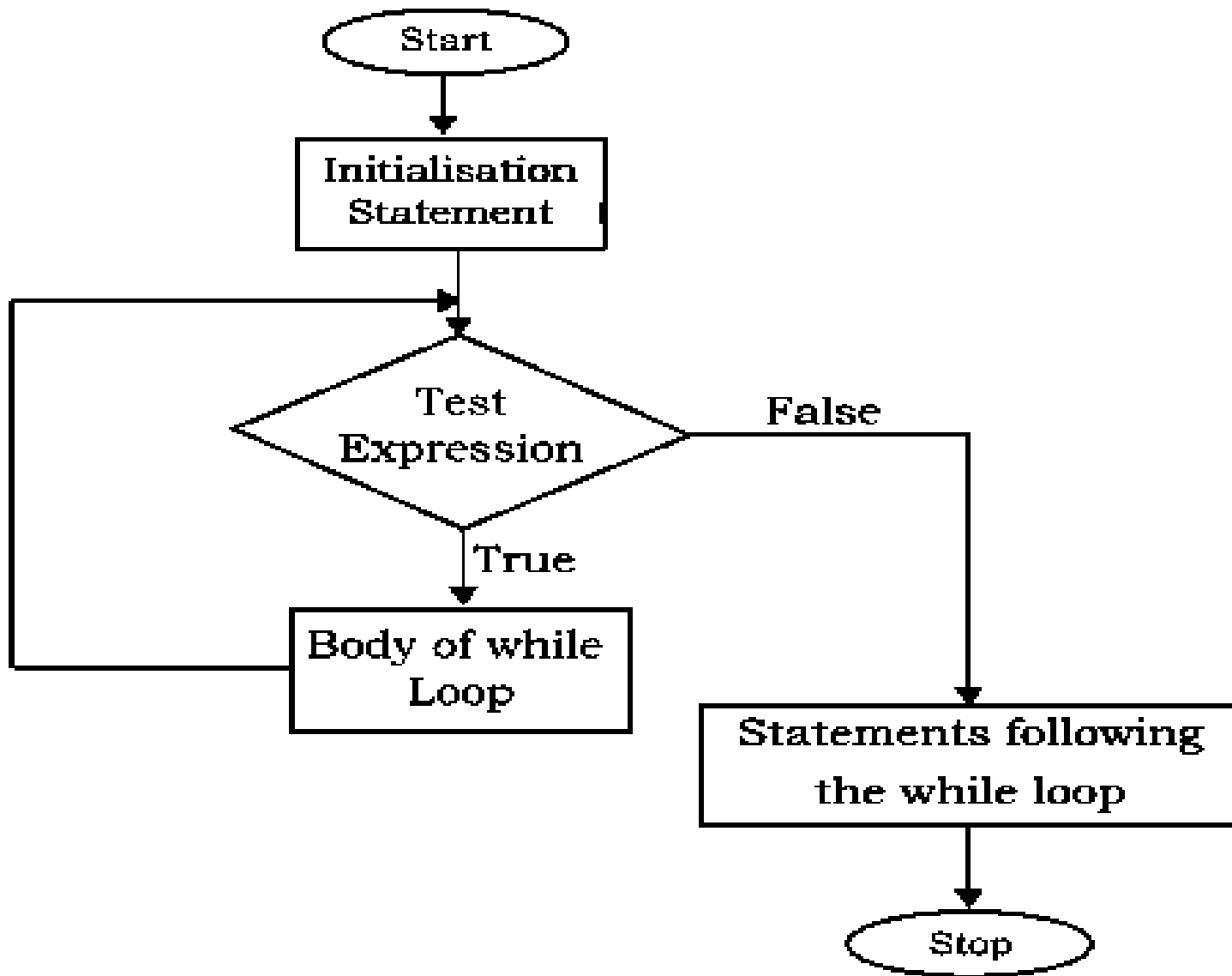
➤ All parameters of range() function must be integers.

➤ `>>> list(range(2, 10))`
`[2, 3, 4, 5, 6, 7, 8, 9]`

➤ `list(range(0, 30, 5))`
`[0, 5, 10, 15, 20, 25]`

'While' Loop

- The while statement executes a block of code repeatedly as long as the control condition of the loop is true.
- The control condition of the while loop is executed before any statement inside the loop is executed.
- After each iteration, the control condition is tested again and the loop continues as long as the condition remains true.
- When this condition becomes false, the statements in the body of loop are not executed and the control is transferred to the statement immediately following the body of while loop.
- If the condition of the while loop is initially false, the body is not executed even once.
- Syntax of while Loop
while test_condition:
 body of while

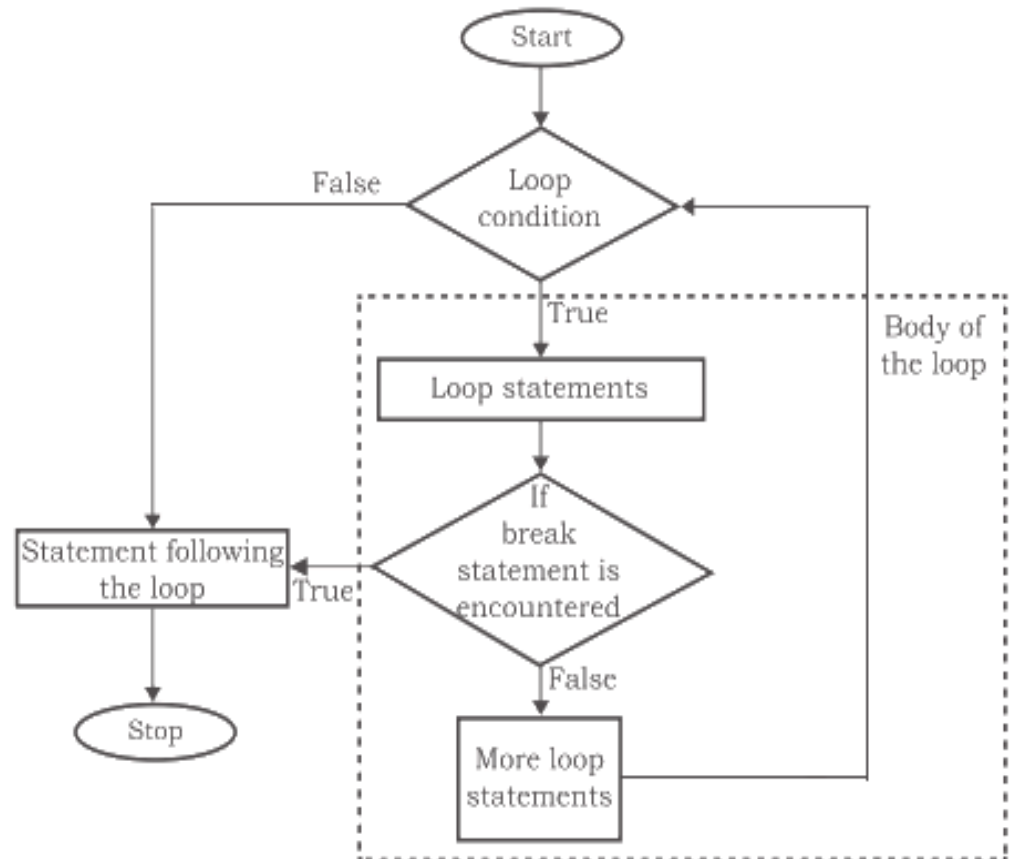


'While' Loop

- Program to print first 5 natural numbers using while loop.
- ```
count = 1
while count <= 5:
 print(count)
 count += 1
```
- Program to find the factors of a whole number using while loop.
- ```
num = int(input("Enter a number to find its factor: "))
print (1, end=' ') #1 is a factor of every number
factor = 2
while factor <= num/2 :
    if num % factor == 0:
        print(factor, end=' ')
        factor += 1
```
- ```
print (num, end=' ')
```

# Break and continue Statement

- Looping constructs allow programmers to repeat tasks efficiently. In certain situations, when some particular condition occurs, we may want to exit from a loop, or skip some statements of the loop before continuing further in the loop.
- These requirements can be achieved by using break and continue statements, respectively.
- **Break Statement :**
- The break statement alters the normal flow of execution as it terminates the current loop and resumes execution of the statement that loop.



# Use of break statement

```
num = 0
for num in range(10):
 num = num + 1
 if num == 8:
 break
 print('Num has value ' + str(num))
print('Encountered break!! Out of loop')
```

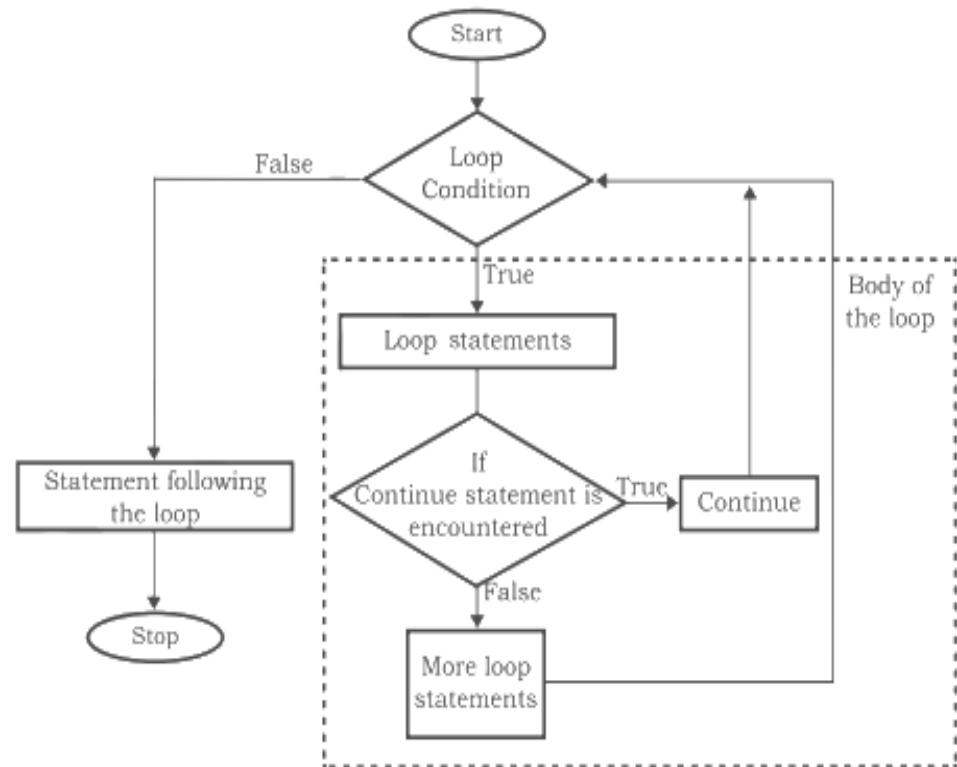


# Use of break statement

- **Q. Program to check if the input number is prime or not.**
- `num = int(input("Enter the number to be checked: "))`
- `flag = 0               #presume num is a prime number`
- `if num > 1 :`
- `for i in range(2, int(num / 2)):`
- `if (num % i == 0):`
- `flag = 1               #num is a not prime number`
- `break                #no need to check any further`
- `if flag == 1:`
- `print(num , "is not a prime number")`
- `else:`
- `print(num , "is a prime number")`
- `else :`
- `print("Entered number is <= 1, execute again!")`

# Continue Statement

- When a continue statement is encountered, the control skips the execution of remaining statements inside the body of the loop for the current iteration and jumps to the beginning of the loop for the next iteration.
- If the loop's condition is still true, the loop is entered again, else the control is transferred to the statement immediately following the loop.



# Continue Statement

- Q. Prints values from 0 to 6 except 3
- ```
num = 0
for num in range(6):
    num = num + 1
    if num == 3:
        continue
    print('Num has value ' + str(num))
print('End of loop')
```


Nested loops

- A loop may contain another loop inside it. A loop inside another loop is called a nested loop.
- ```
for var1 in range(3):
 print("Iteration " + str(var1 + 1) + " of outer loop")
 for var2 in range(2): #nested loop
 print(var2 + 1)
 print("Out of inner loop")
print("Out of outer loop")
```

# Nested loops

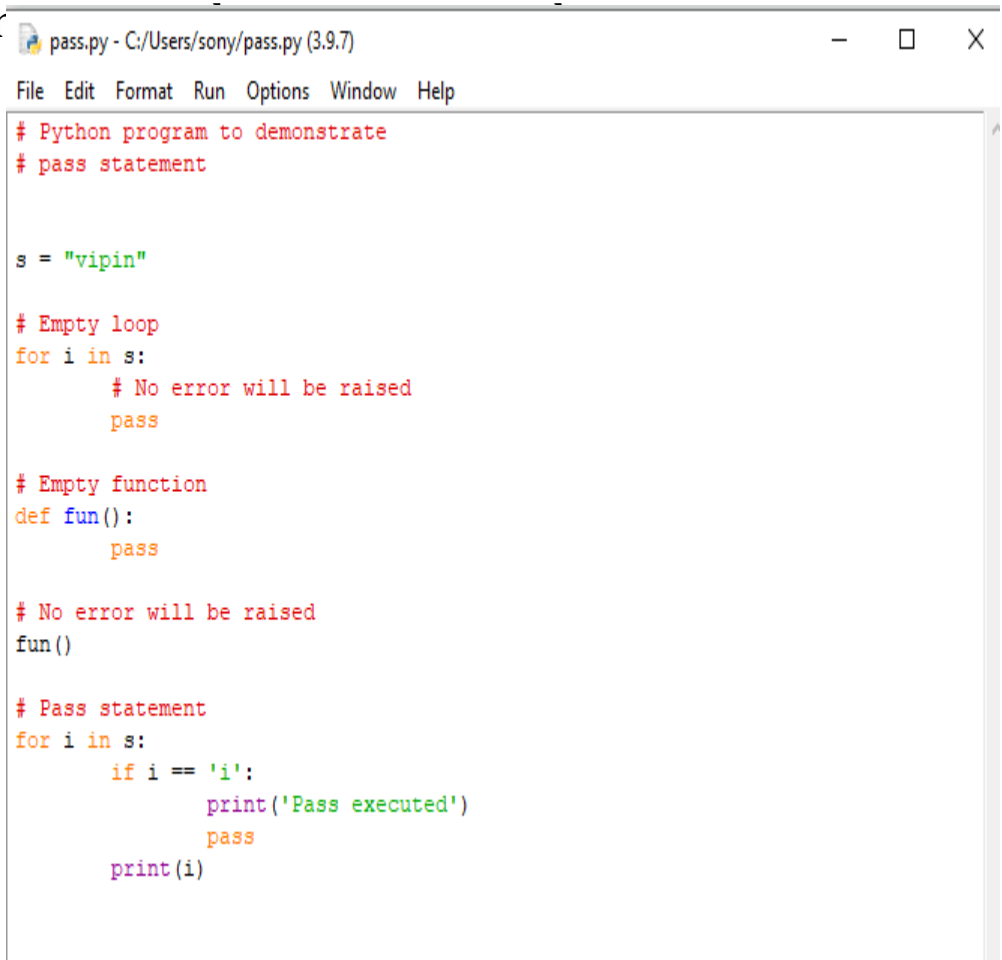
- Program to find prime numbers between 2 to 50 using nested for loops.
- `num = 2`
- `for i in range(2, 51):`
  - `j= 2`
  - `while ( j <= (i/2)):`
    - `if (i % j == 0):` `#factor found`
      - `break` `#break out of while loop`
      - `j += 1`
    - `if ( j > i/j) :` `#no factor found`
    - `print ( i, "is a prime number")`
  - `print ("Bye Bye!!")`

- The if statement is used for selection or decision making.
- The looping constructs while and for allow sections of code to be executed repeatedly under some condition.
  - for statement iterates over a range of values or a sequence.
  - The statements within the body of for loop are executed till the range of values is exhausted.
  - The statements within the body of a while are executed over and over until the condition of the while is false.
  - If the condition of the while loop is initially false, the body is not executed even once.
  - The statements within the body of the while loop must ensure that the condition eventually becomes false; otherwise, the loop will become an infinite loop, leading to a logical error in the program.
  - The break statement immediately exits a loop, skipping the rest of the loop's body. Execution continues with the statement immediately following the body of the loop. When a continue statement is encountered, the control jumps to the beginning of the loop for the next iteration.
  - A loop contained within another loop is called a nested loop

# Pass

- ❖ It is used when a statement is required syntactically but you don't want any code to execute.
- ❖ It is a null statement, nothing happens.

Syntax: pass



```
pass.py - C:/Users/sony/pass.py (3.9.7)
File Edit Format Run Options Window Help

Python program to demonstrate
pass statement

s = "vipin"

Empty loop
for i in s:
 # No error will be raised
 pass

Empty function
def fun():
 pass

No error will be raised
fun()

Pass statement
for i in s:
 if i == 'i':
 print('Pass executed')
 pass
 print(i)
```

- Write a function to print the table of a given number. The number has to be entered by the user.



Program:

```
#Program to print the table of a number
```

```
num = int(input("Enter the number to display its table: "))
```

```
print("Table: ");
```

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

```
 print(num, " x ", a, " = ", (num*a))
```

- Write a program to check if the year entered by the user is a leap year or not.

The year is a leap year if it is divisible by 4.

Program:

```
year = int(input("Enter the year: "))
if (year % 4 == 0):
 print("The year",year," is a leap year.")
else:
 print("The year",year," is not a leap year.")
```

- Write a program to find the sum of digits of an integer number, input by the user.

```
#Initializing the sum to zero
sum = 0
#Getting user input
n = int(input("Enter the number: "))
looping through each digit of the number
Modulo by 10 will give the first digit and
floor operator decreases the digit 1 by 1
while n > 0:
 digit = n % 10
 sum = sum + digit
 n = n//10
Printing the sum
print("The sum of digits of the number is",sum)
```

➤ Write a function that checks whether an input number is a palindrome or not. .

```
rev = 0
n = int(input("Enter the number: "))
#Storing the number input by user in a temp variable to run the loop
temp = n
#Using while function to reverse the number
while temp > 0:
 digit = (temp % 10)
 rev = (rev * 10) + digit
 temp = temp // 10
#Checking if original number and reversed number are equal
if(n == rev):
 print("The entered number is a palindrome.")
else:
 print("The entered number is not a palindrome.")
```