

PATNA WOMEN'S COLLEGE

AUTONOMOUS, PATNA UNIVERSITY

DEPARTMENT OF COMPUTER APPLICATION

ASSIGNMENT OF PYTHON PROGRAMMING

SESSION :- 2019-2022

PAPER CODE :-BCADSE603

SUBMITTED BY:

NAME – SALONI KUMARI

CLASS – BCA

ROLL NO . – 14

SEMESTER – 6 TH

EXAM ROLL NO. -19BCA10460

REG. NO. – 19PWC01460

SUMMITTED TO:

ANSHU MA'AM

Q. DEFINE/EXPLAIN CONTROL STATEMENT WITH ONE EXAMPLE OF EACH.

a) Selective statements


if statement

if statement is the most simple decision-making statement. It is used to decide whether a certain statement or block of statements will be executed or not i.e if a certain condition is true then a block of statement is executed otherwise not.

Syntax:

```
if condition:
    # Statements to execute if
    # condition is true
```

Example:

The image shows a screenshot of the Python IDLE Shell 3.10.1. The top window is the code editor, and the bottom window is the interactive shell. In the code editor, the following code is written:

```
File Edit Format Run Options Window Help
i = 10

if (i > 15):
    print("10 is less than 15")
print("I am Not in if")
```

The shell window shows the output of the code execution:

```
IDLE Shell 3.10.1
File Edit Shell Debug Options Window Help
Python 3.10.1 (tags/v3.10.1:2cd268a, Dec 6 2021, 19:10:37) [MSC v.1929 64 bit (AMD64)] on win32
Type "help", "copyright", "credits" or "license()" for more information.
>>>
===== RESTART: C:/Users/hp/Documents/assignment.phy.py =====
I am Not in if
>>>
```

if-else

The if statement alone tells us that if a condition is true it will execute a block of statements and if the condition is false it won't. But what if we want to do something else if the condition is false. Here comes the *else* statement. We can use the *else* statement with *if* statement to execute a block of code when the condition is false.

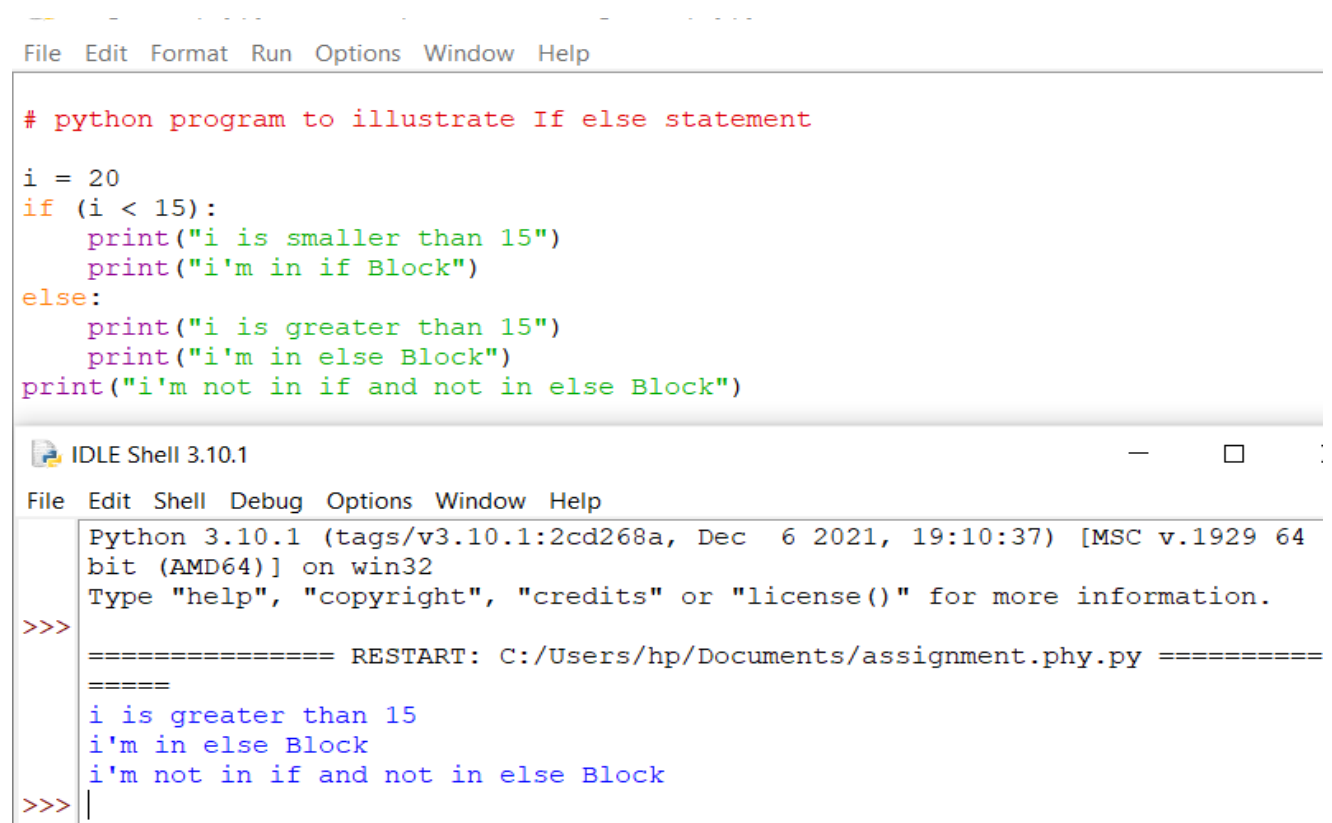
Syntax:

```
if (condition):
    # Executes this block if
    # condition is true
```

else:

Executes this block if

condition is false



The screenshot shows the Python IDLE Shell 3.10.1 interface. The top menu bar includes File, Edit, Format, Run, Options, Window, and Help. The main text area contains the following Python code:

```
# python program to illustrate If else statement

i = 20
if (i < 15):
    print("i is smaller than 15")
    print("i'm in if Block")
else:
    print("i is greater than 15")
    print("i'm in else Block")
print("i'm not in if and not in else Block")
```

The bottom shell window shows the output of the code execution:

```
Python 3.10.1 (tags/v3.10.1:2cd268a, Dec 6 2021, 19:10:37) [MSC v.1929 64
bit (AMD64)] on win32
Type "help", "copyright", "credits" or "license()" for more information.
>>>
===== RESTART: C:/Users/hp/Documents/assignment.phy.py =====
>>>
i is greater than 15
i'm in else Block
i'm not in if and not in else Block
>>> |
```

nested-if

A nested if is an if statement that is the target of another if statement. Nested if statements mean an if statement inside another if statement. Yes, Python allows us to nest if statements within if statements. i.e, we can place an if statement inside another if statement.

Syntax:

```
if (condition1):
```

```
    # Executes when condition1 is true
```

```
    if (condition2):
```

```
        # Executes when condition2 is true
```

```
    # if Block is end here
```

```
# if Block is end here
```

```
File Edit Format Run Options Window Help
# python program to illustrate nested If statement
i = 10
if (i == 10):

    # First if statement
    if (i < 15):
        print("i is smaller than 15")

    # Nested - if statement
    # Will only be executed if statement above
    # it is true
    if (i < 12):
        print("i is smaller than 12 too")
    else:
        print("i is greater than 15")

IDLE Shell 3.10.1
File Edit Shell Debug Options Window Help
Python 3.10.1 (tags/v3.10.1:2cd268a, Dec 6 2021, 19:10:37) [MSC v.1929 64 b
it (AMD64)] on win32
Type "help", "copyright", "credits" or "license()" for more information.
>>>
===== RESTART: C:/Users/hp/Documents/assignment.phy.py =====
>>>
i is smaller than 15
i is smaller than 12 too
>>>
```

if-elif-else ladder

Here, a user can decide among multiple options. The if statements are executed from the top down. As soon as one of the conditions controlling the if is true, the statement associated with that if is executed, and the rest of the ladder is bypassed. If none of the conditions is true, then the final else statement will be executed.

Syntax:

```
if (condition):
```

```
    statement
```

```
elif (condition):
```

```
    statement
```

```
•
```

```
•
```

```
else:
```

```
    statement
```

```
File Edit Format Run Options Window Help
# Python program to illustrate if-elif-else ladder
i = 20
if (i == 10):
    print("i is 10")
elif (i == 15):
    print("i is 15")
elif (i == 20):
    print("i is 20")
else:
    print("i is not present")

IDLE Shell 3.10.1
Python 3.10.1 (tags/v3.10.1:2cd268a, Dec 6 2021, 19:10:37) [MSC v.1929 64 bit (AMD64)] on win32
Type "help", "copyright", "credits" or "license()" for more information.
>>>
===== RESTART: C:/Users/hp/Documents/assignment.phy.py =====
>>> i is 20
>>> |
```

b)Loops

While Loop:

In python, while loop is used to execute a block of statements repeatedly until a given condition is satisfied. And when the condition becomes false, the line immediately after the loop in program is executed.

Syntax :

while expression:

 statement(s)

All the statements indented by the same number of character spaces after a programming construct are considered to be part of a single block of code. Python uses indentation as its method of grouping statements.

```
File Edit Format Run Options Window Help
# Python program to illustrate while loop
count = 0
while (count < 3):
    count = count + 1
    print("Hello saloni")

IDLE Shell 3.10.1
Python 3.10.1 (tags/v3.10.1:2cd268a, Dec 6 2021, 19:10:37) [MSC v.1929 64 bit (AMD64)] on win32
Type "help", "copyright", "credits" or "license()" for more information.
>>>
===== RESTART: C:/Users/hp/Documents/assignment.phy.py =====
>>> Hello saloni
>>> Hello saloni
>>> Hello saloni
>>> |
```

For loop:

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.

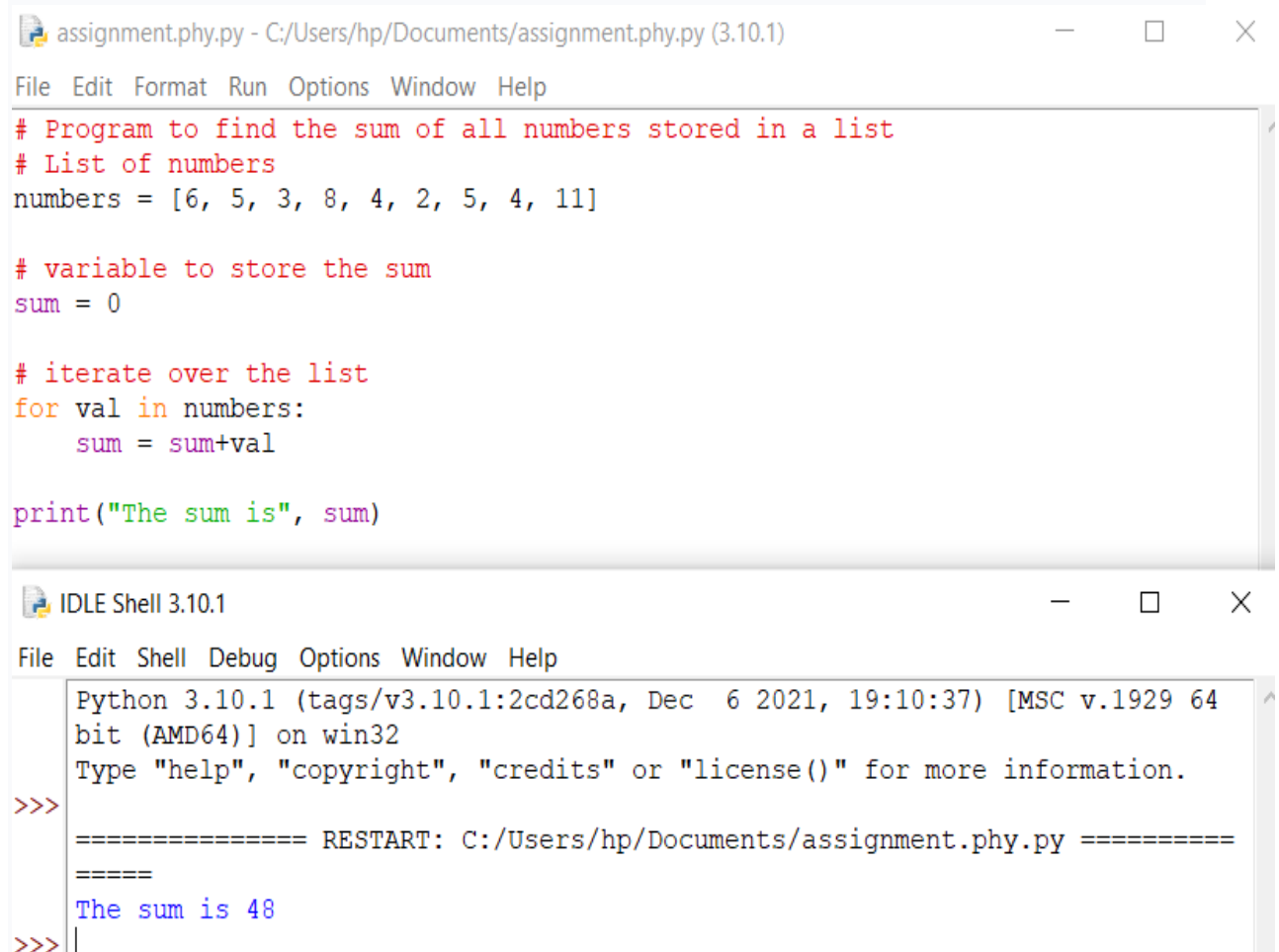
Syntax:

```
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.



The screenshot displays two windows from a Python IDE. The top window, titled 'assignment.phy.py - C:/Users/hp/Documents/assignment.phy.py (3.10.1)', contains the following Python code:

```
File Edit Format Run Options Window Help
# Program to find the sum of all numbers stored in a list
# 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)
```

The bottom window, titled 'IDLE Shell 3.10.1', shows the output of the program after execution:

```
File Edit Shell Debug Options Window Help
Python 3.10.1 (tags/v3.10.1:2cd268a, Dec 6 2021, 19:10:37) [MSC v.1929 64
bit (AMD64)] on win32
Type "help", "copyright", "credits" or "license()" for more information.
>>>
===== RESTART: C:/Users/hp/Documents/assignment.phy.py =====
>>>
The sum is 48
>>> |
```

The range() function:

We can generate a sequence of numbers

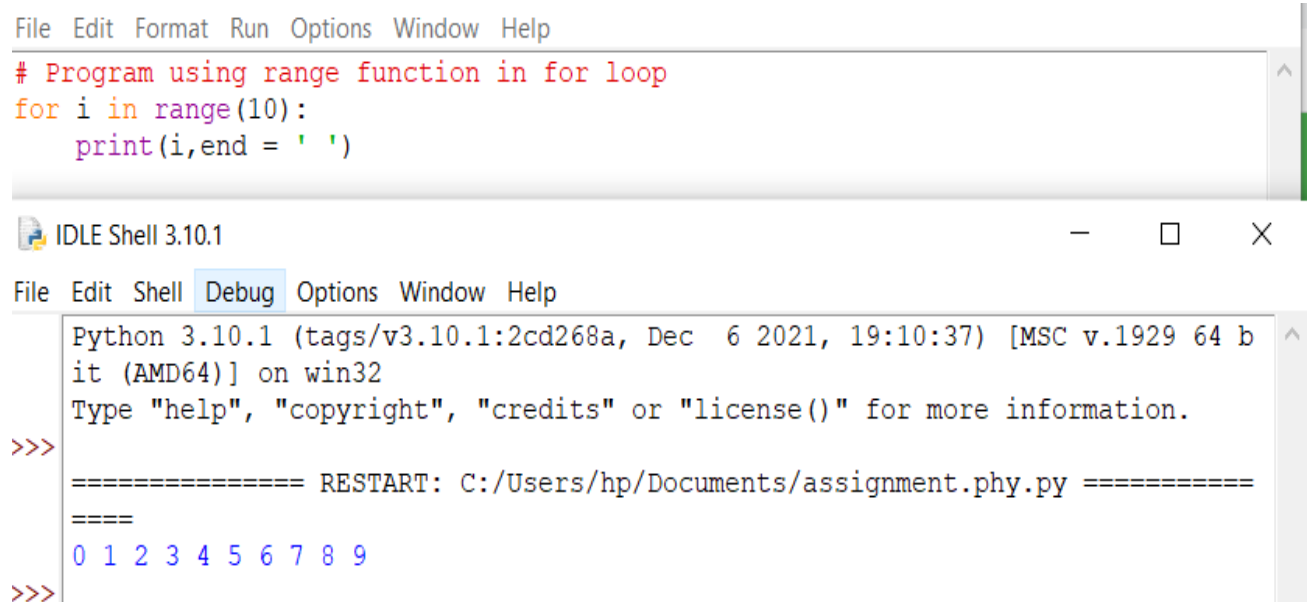
using `range()` function. `range(10)` will generate numbers from 0 to 9 (10 numbers).

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

The `range` object is "lazy" in a sense because it doesn't generate every number that it "contains" when we create it. However, it is not an iterator since it supports `in`, `len` and `__getitem__` operations.

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()`.



The screenshot shows a Python IDE window titled "IDLE Shell 3.10.1". The menu bar includes File, Edit, Format, Run, Options, Window, and Help. The code editor contains the following Python code:

```
# Program using range function in for loop
for i in range(10):
    print(i, end = ' ')
```

Below the code editor, the shell output is displayed. It shows the Python version and architecture: "Python 3.10.1 (tags/v3.10.1:2cd268a, Dec 6 2021, 19:10:37) [MSC v.1929 64 bit (AMD64)] on win32". It also shows the command prompt prompt ">>>". The output of the program is "0 1 2 3 4 5 6 7 8 9".

Q.EXPLAIN JUMPING STATEMENTS.

Jump statements are used to skip, jump, or exit from the running program inside from the loop at the particular condition. They are used mainly to interrupt switch statements and loops. Jump statements are break, continue, return, and exit statement.

Jump Statements in Python

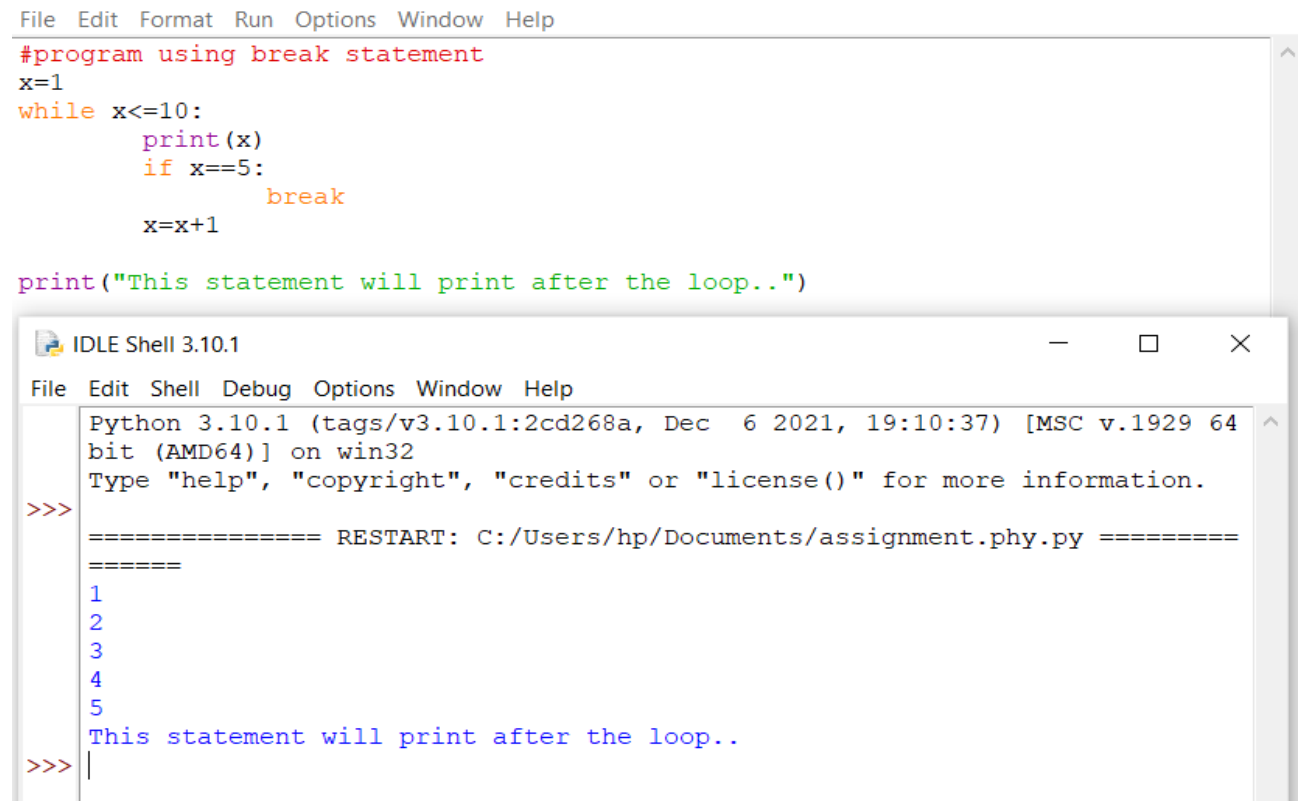
- Break Statement
- Continue Statement
- Pass Statement

Break statement

With the help of the `break` the statement, we can terminate the loop. We can use it will terminate the loop if the condition is true. By the keyword we describe the break statement.

Example of break Statement

We are going to print a series, and we want to stop the loop after executing two time



```
File Edit Format Run Options Window Help
#program using break statement
x=1
while x<=10:
    print(x)
    if x==5:
        break
    x=x+1

print("This statement will print after the loop..")

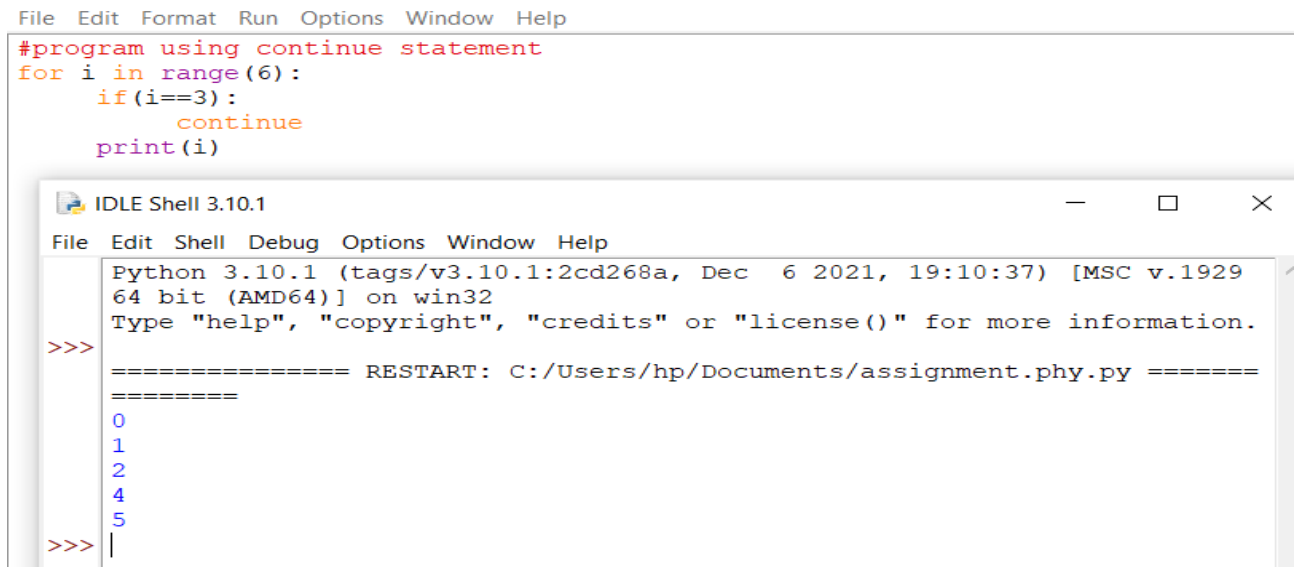
IDLE Shell 3.10.1
File Edit Shell Debug Options Window Help
Python 3.10.1 (tags/v3.10.1:2cd268a, Dec 6 2021, 19:10:37) [MSC v.1929 64
bit (AMD64)] on win32
Type "help", "copyright", "credits" or "license()" for more information.
>>>
===== RESTART: C:/Users/hp/Documents/assignment.phy.py =====
=====
1
2
3
4
5
This statement will print after the loop..
>>> |
```

Continue statement

With the help of the continue statement, we can terminate any iteration and it returns the control to the beginning again. Suppose we want to skip/terminate further execution for a certain condition of the loop. By using the continue keyword we define the continued statement.

Example of continue Statement


```
File Edit Format Run Options Window Help
#program using continue statement
for i in range(6):
    if(i==3):
        continue
    print(i)
```



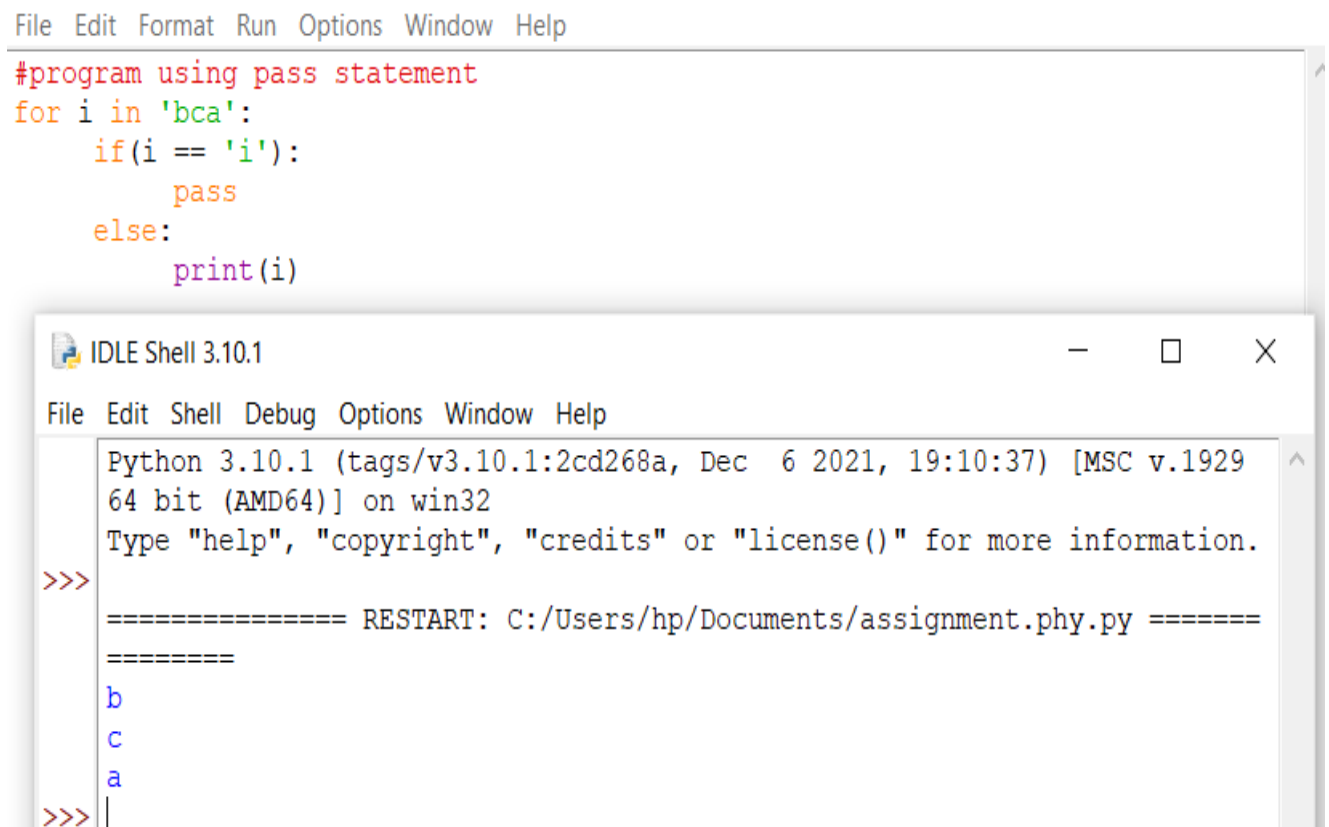
```
Python 3.10.1 (tags/v3.10.1:2cd268a, Dec 6 2021, 19:10:37) [MSC v.1929
64 bit (AMD64)] on win32
Type "help", "copyright", "credits" or "license()" for more information.
>>>
===== RESTART: C:/Users/hp/Documents/assignment.phy.py =====
0
1
2
4
5
>>> |
```

Pass Statement

Pass statement in python does nothing. You can use pass statement when you create a method that you don't want to implement, yet.

Example of Pass statement

```
File Edit Format Run Options Window Help
#program using pass statement
for i in 'bca':
    if(i == 'i'):
        pass
    else:
        print(i)
```



```
Python 3.10.1 (tags/v3.10.1:2cd268a, Dec 6 2021, 19:10:37) [MSC v.1929
64 bit (AMD64)] on win32
Type "help", "copyright", "credits" or "license()" for more information.
>>>
===== RESTART: C:/Users/hp/Documents/assignment.phy.py =====
b
c
a
>>> |
```