Parul® University

CHAPTER-3

PROGRAM FLOW CONTROL

1. CONDITIONAL CONSTRUCT – if else STATEMENT

CONDITIONAL CONSTRUCT – if else STATEMENT

Conditional constructs (also known as if statements) provide a way to execute a chosen block of code based on the run-time evaluation of one or more Boolean expressions. In Python, the most general form of a conditional is written as follows:

Contd.. Next Slide

CONDITIONAL CONSTRUCT – if else STATEMENT

: Colon Must

```
if first condition:
    first body
elif second condition:
    second body
elif third condition:
    third body
else:
    fourth body
```

CONDITIONAL CONSTRUCT – if else STATEMENT

✓Each condition is a Boolean expression, and each body contains one or more commands that are to be executed conditionally.

✓If the first condition succeeds, the first body will be executed; no other conditions or bodies are evaluated in that case.

CONDITIONAL CONSTRUCT – if else STATEMENT

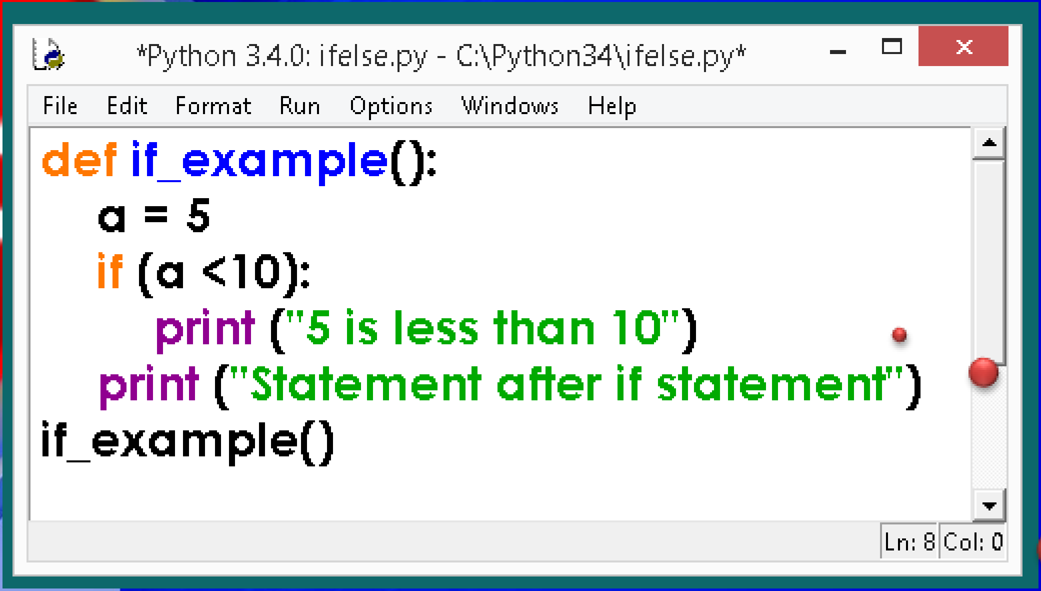
✓If the first condition fails, then the process continues in similar manner with the evaluation of the second condition. The execution of this overall construct will cause precisely one of the bodies to be executed.

✓There may be any number of elif clauses (including zero), and
✓ The final else clause is optional.

CONDITIONAL CONSTRUCT – if else STATEMENT

EXAMPLE - PROGRAM

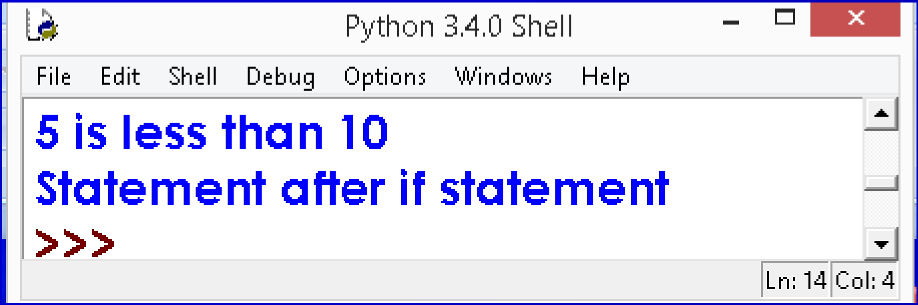
EXAMPLES – if STATEMENT


Python 3.4.0: ifelse.py - C:\Python34\ifelse.py
File Edit Format Run Options Windows Help
Ln: 8 Col: 0

```
def if_example():
    a = 5
    if (a <10):
        print ("5 is less than 10")
    print ("Statement after if statement")
if_example()
```

else is missing, it is an optional statement

OUT PUT



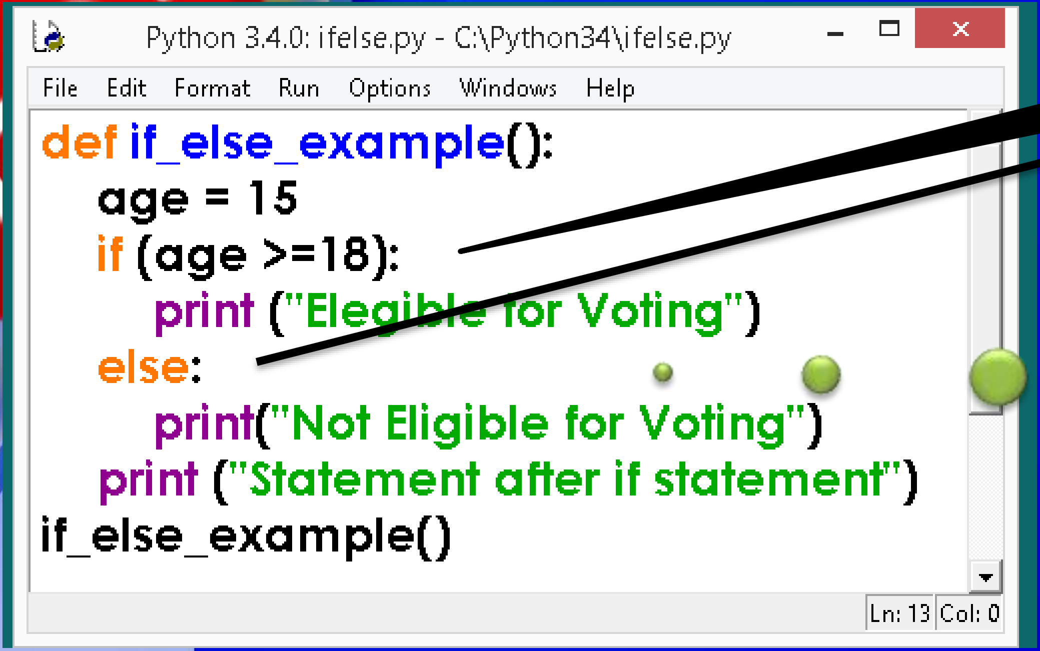
```
5 is less than 10
Statement after if statement
>>>
```

fppt.com

CONDITIONAL CONSTRUCT

EXAMPLE – if else STATEMENT

EXAMPLE – if else STATEMENT

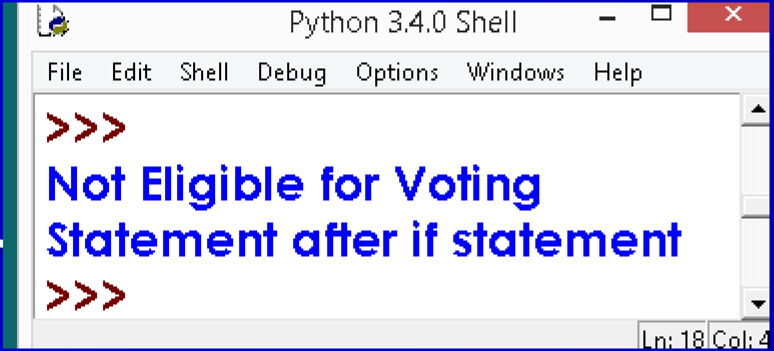

Python 3.4.0: ifelse.py - C:\Python34\ifelse.py
File Edit Format Run Options Windows Help
Ln: 13 Col: 0

```
def if_else_example():
    age = 15
    if (age >=18):
        print ("Elegible for Voting")
    else:
        print("Not Eligible for Voting")
    print ("Statement after if statement")
if_else_example()
```

: Colon Must

else is used

OUT PUT

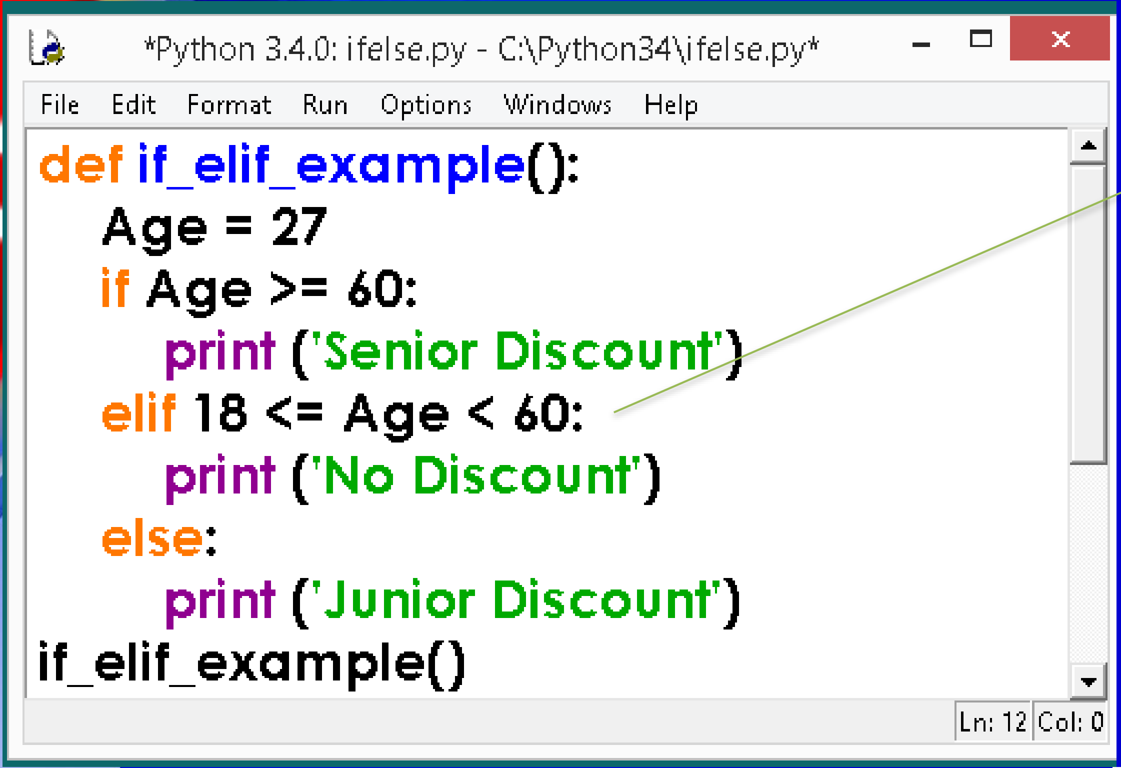


```
>>>
Not Eligible for Voting
Statement after if statement
>>>
```

CONDITIONAL CONSTRUCT

EXAMPLES – if elif STATEMENT

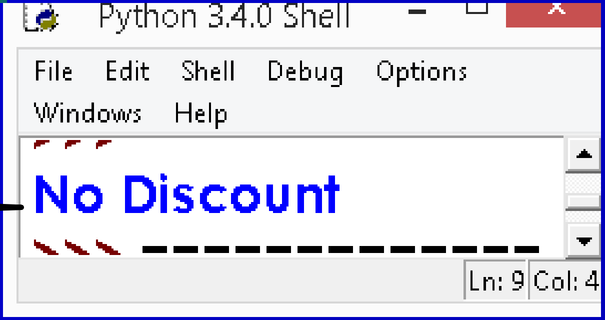
EXAMPLES – if elif STATEMENT


Python 3.4.0: ifelse.py - C:\Python34\ifelse.py
File Edit Format Run Options Windows Help
Ln: 12 Col: 0

```
def if_elif_example():
    Age = 27
    if Age >= 60:
        print ('Senior Discount')
    elif 18 <= Age < 60:
        print ('No Discount')
    else:
        print ('Junior Discount')
if_elif_example()
```

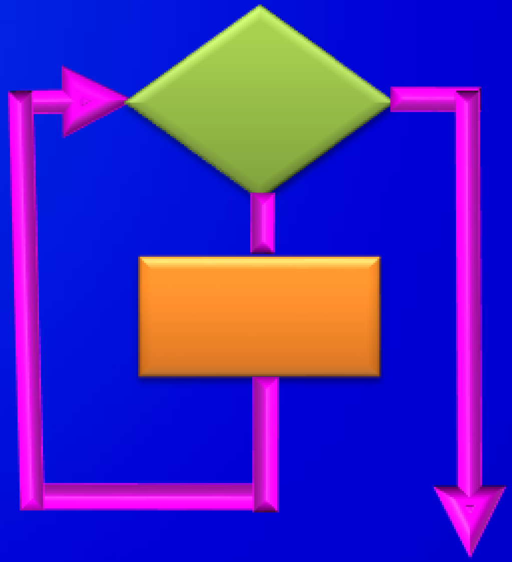
READ AS
18 is less than age and 18 is less than 60

OUTPUT



```
No Discount
```

2. ITERATION OR LOOPING



ITERATION

3. ITERATION OR LOOPING

What is loop or iteration?

Loops can execute a block of code number of times until a certain condition is met.

OR

The iteration statement allows instructions to be executed until a certain condition is to be fulfilled.

The iteration statements are also called as loops or Looping statements.

3. ITERATION OR LOOPING

Python provides two kinds of loops & they are,

while loop

for loop

while loop

while loop

A while loop allows general repetition based upon the repeated testing of a Boolean condition

The syntax for a while loop in Python is as follows:

```
while condition:
    body
```

: Colon Must

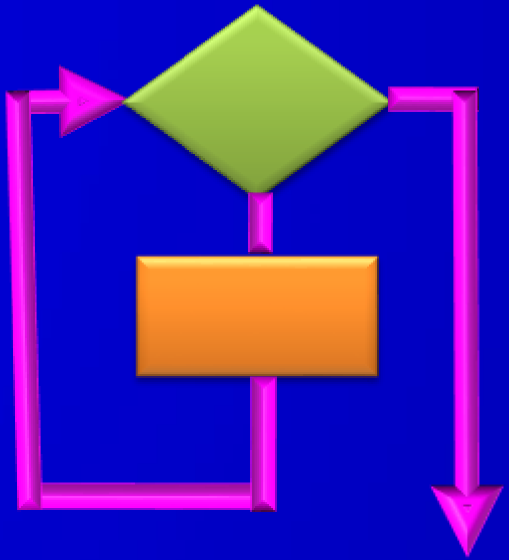
Where, loop body contain the single statement or set of statements (compound statement) or an empty statement.

Contd..

while loop

The loop iterates while the expression evaluates to true, when expression becomes false the loop terminates.

FLOW CHART →

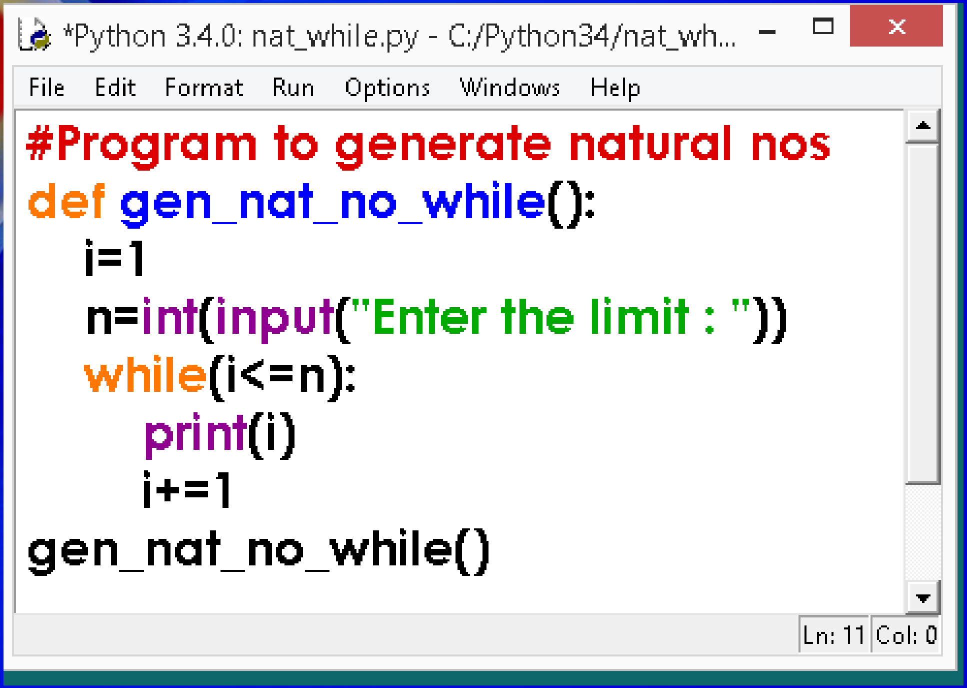


while loop

while loop – Programming example

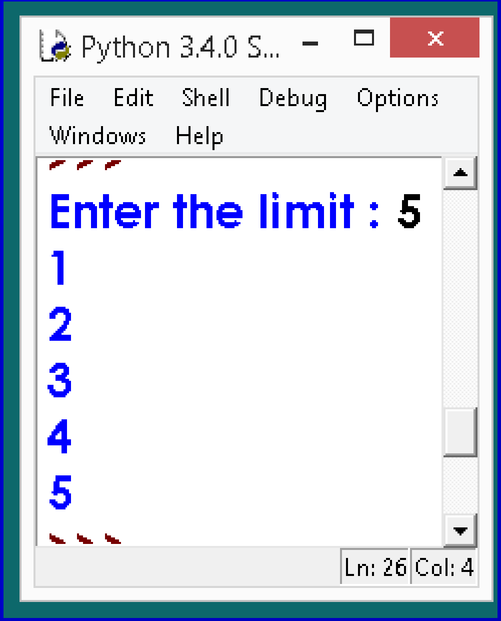
while loop - programs

# Natural Numbers generation


*Python 3.4.0: nat_while.py - C:/Python34/nat_wh...
File Edit Format Run Options Windows Help
Ln: 11 Col: 0

```
#Program to generate natural nos
def gen_nat_no_while():
    i=1
    n=int(input("Enter the limit : "))
    while(i<=n):
        print(i)
        i+=1
gen_nat_no_while()
```

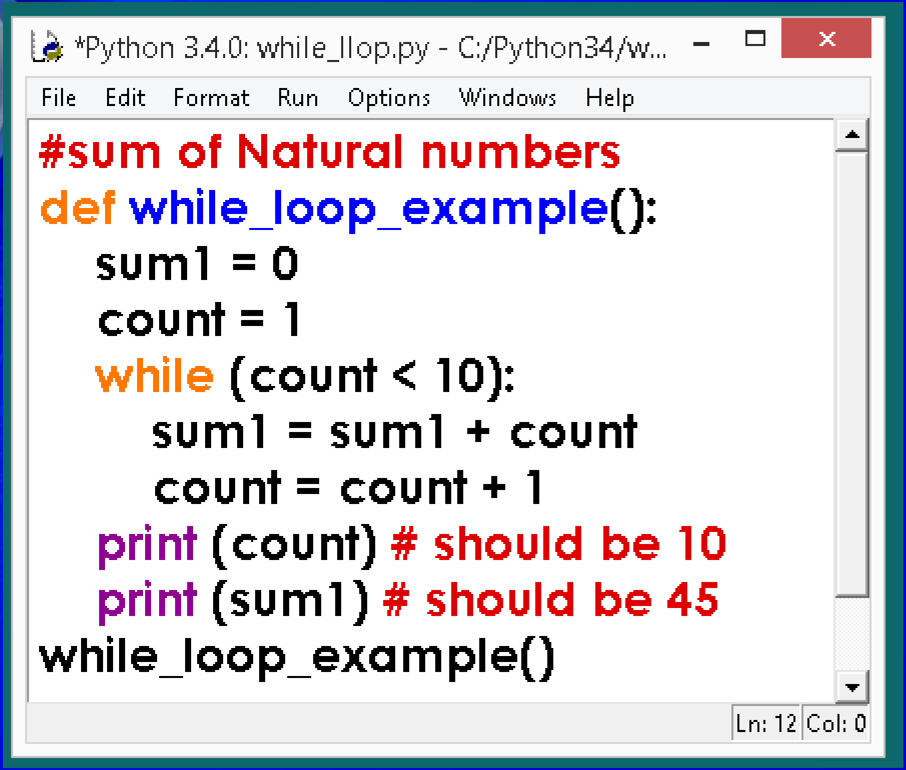
OUTPUT



```
Enter the limit : 5
1
2
3
4
5
```

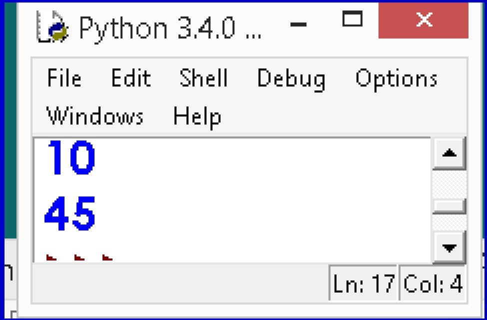
while loop - programs

# Calculating Sum of Natural Numbers


*Python 3.4.0: while_llop.py - C:/Python34/w...
File Edit Format Run Options Windows Help
Ln: 12 Col: 0

```
#sum of Natural numbers
def while_loop_example():
    sum1 = 0
    count = 1
    while (count < 10):
        sum1 = sum1 + count
        count = count + 1
    print (count) # should be 10
    print (sum1) # should be 45
while_loop_example()
```

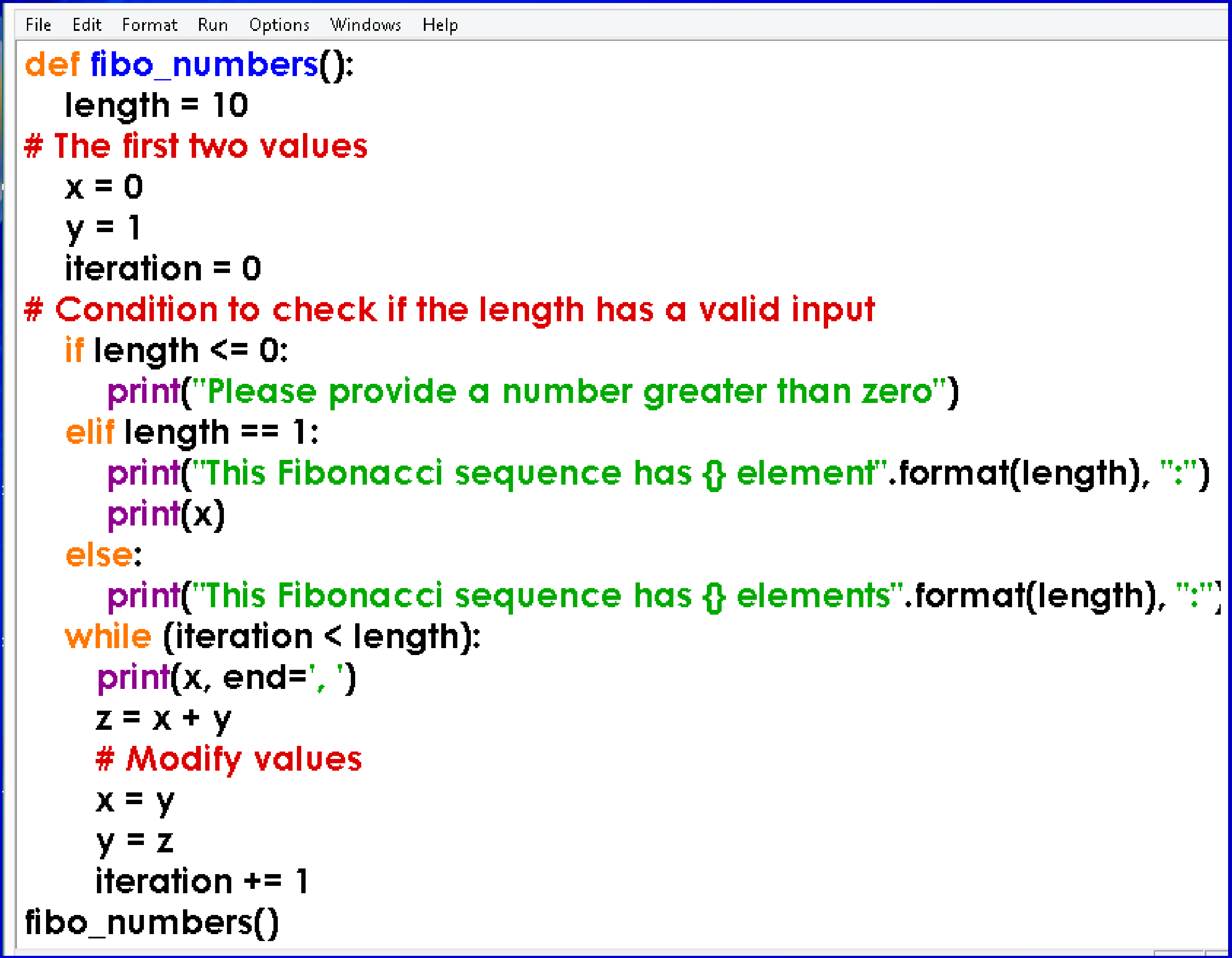
OUTPUT



```
10
45
```

while loop - programs

#Generating Fibonacci numbers

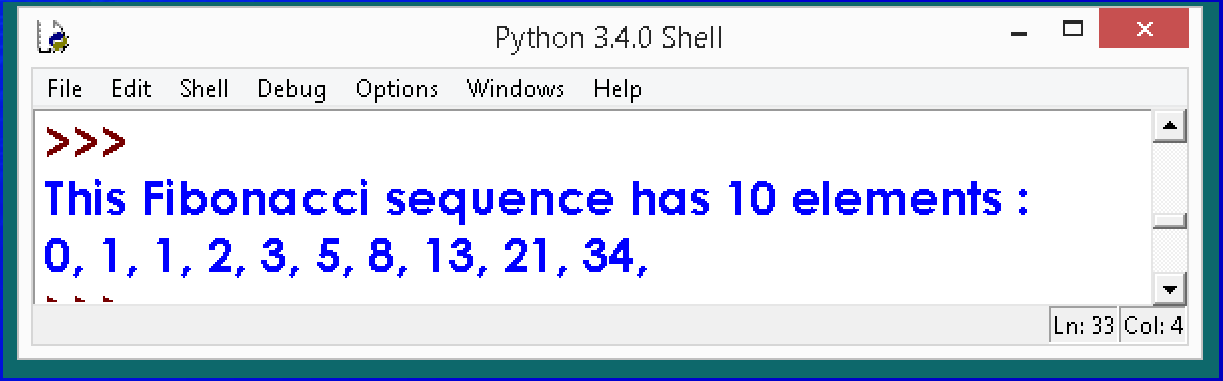


```
def fibo_numbers():
    length = 10
# The first two values
    x = 0
    y = 1
    iteration = 0
# Condition to check if the length has a valid input
    if length <= 0:
        print("Please provide a number greater than zero")
    elif length == 1:
        print("This Fibonacci sequence has {} element".format(length), ":")
        print(x)
    else:
        print("This Fibonacci sequence has {} elements".format(length), ":"
    while (iteration < length):
       print(x, end=', ')
       z = x + y
       # Modify values
       x = y
       y = z
       iteration += 1
fibo_numbers()
```

while loop - programs

#Generating Fibonacci numbers

OUTPUT



```
>>>
This Fibonacci sequence has 10 elements :
0, 1, 1, 2, 3, 5, 8, 13, 21, 34,
```

for LOOP

for LOOP

Python's for-loop syntax is a more convenient alternative to a while loop when iterating through a series of elements. The for-loop syntax can be used on any type of iterable structure, such as a list, tuple str, set, dict, or file

Syntax or general format of for loop is,

```
for element in iterable:
    body
```

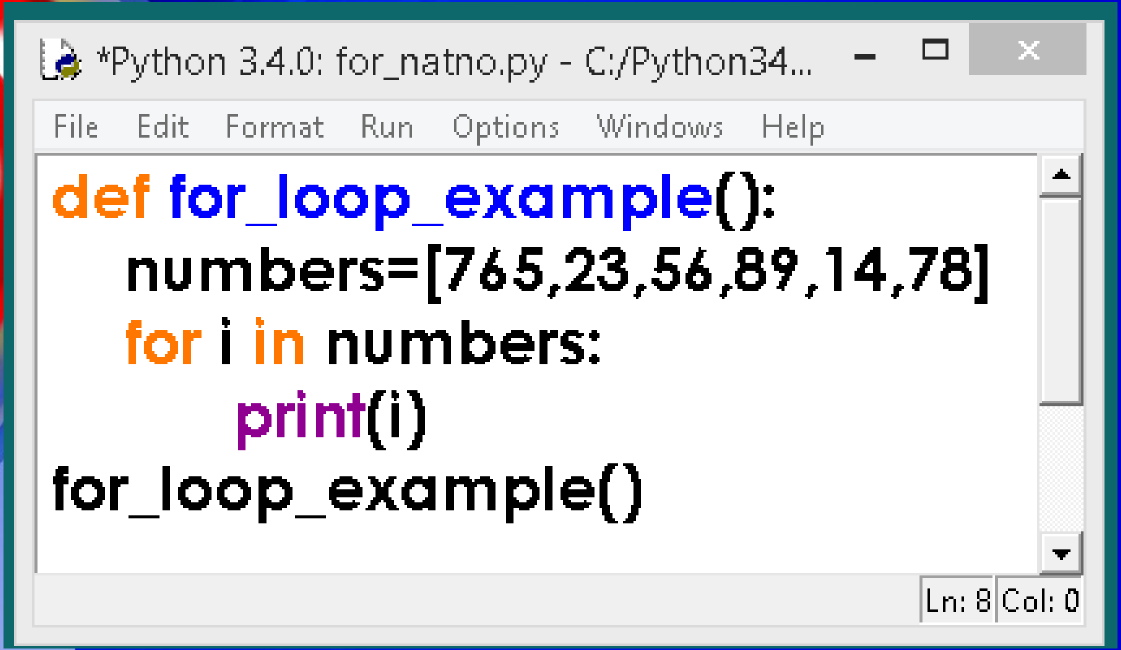
for LOOP

Python's for-loop syntax is a more convenient alternative to a while loop when iterating through a series of elements. The for-loop syntax can be used on any type of iterable structure, such as a list, tuple str, set, dict, or file

Syntax or general format of for loop is,

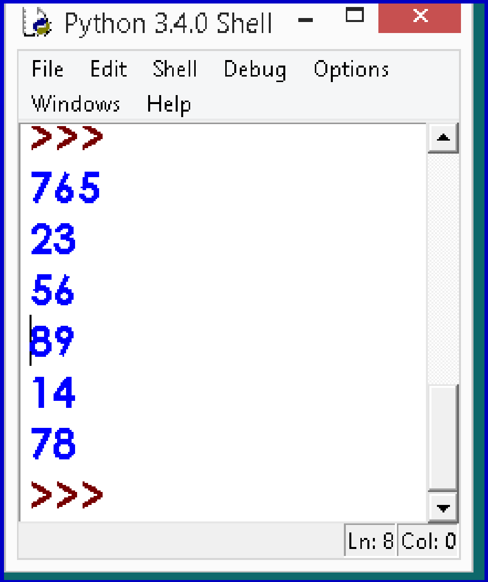
```
for element in iterable:
    body
```

for LOOP


*Python 3.4.0: for_natno.py - C:/Python34...
File Edit Format Run Options Windows Help
Ln: 8 Col: 0

```
def for_loop_example():
    numbers=[765,23,56,89,14,78]
    for i in numbers:
        print(i)
for_loop_example()
```

Till the list exhaust for loop will continue to execute.



```
>>>
765
23
56
89
14
78
>>>
```

for LOOP

range KEYWORD

for LOOP - range KEYWORD

The range() function returns a sequence of numbers, starting from 0 by default, and increments by 1 (by default), and ends at a specified number.

range(*start, stop, step*)

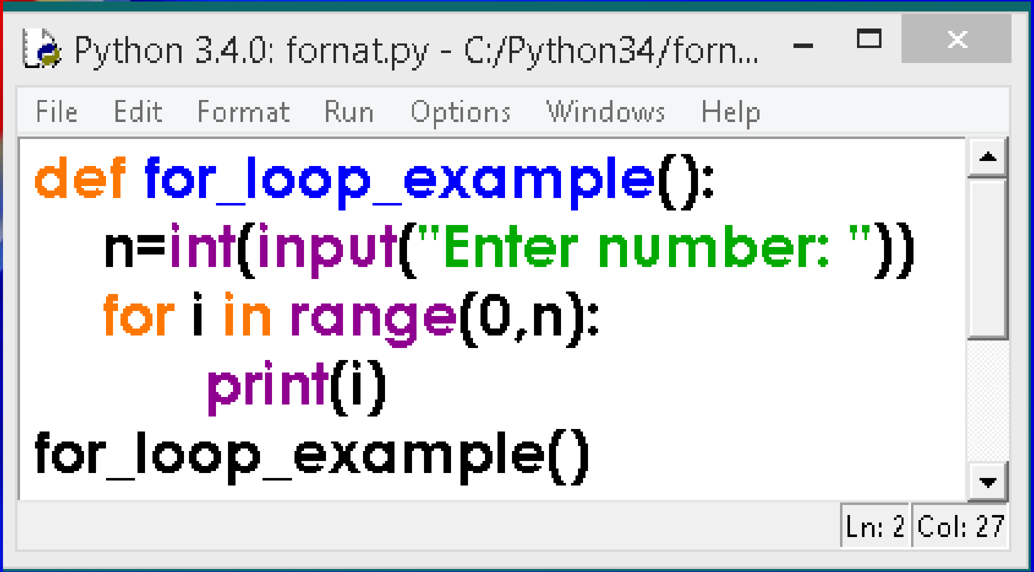
```
for n in range(3,6):
  print(n)
```

OR

```
x = range(3, 6)
for n in x:
  print(n)
```

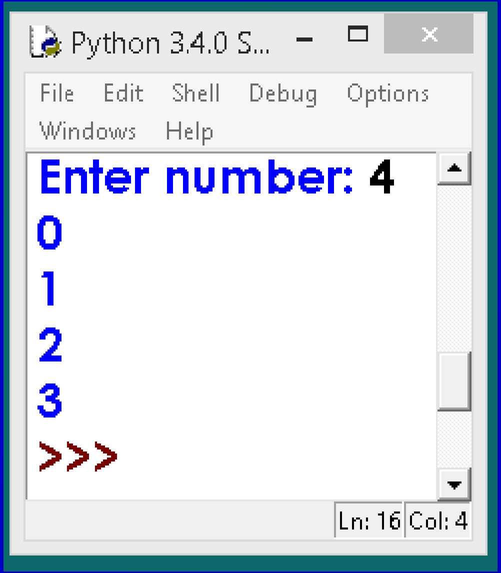
for LOOP - range KEYWORD

#Generating series of numbers


Python 3.4.0: fornat.py - C:/Python34/forn...
File Edit Format Run Options Windows Help
Ln: 2 Col: 27

```
def for_loop_example():
    n=int(input("Enter number: "))
    for i in range(0,n):
        print(i)
for_loop_example()
```

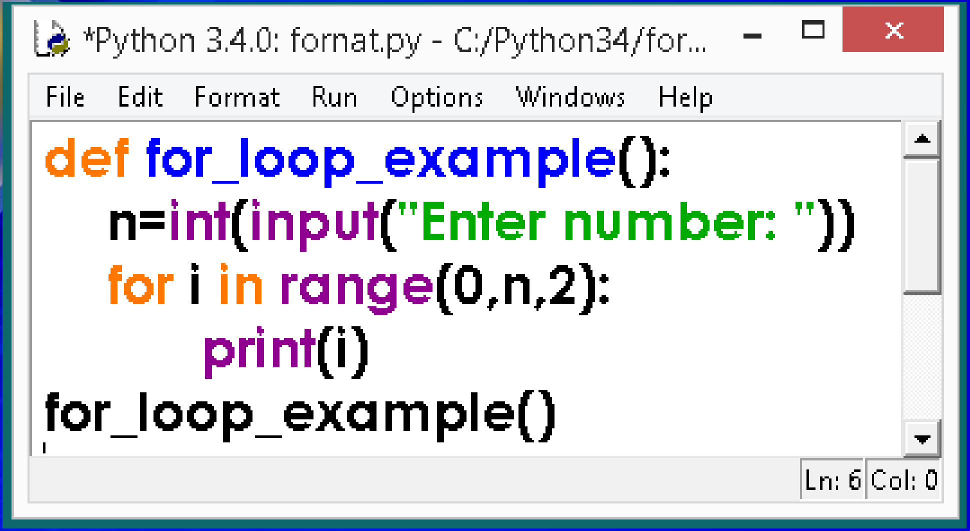
OUTPUT



```
Enter number: 4
0
1
2
3
>>>
```

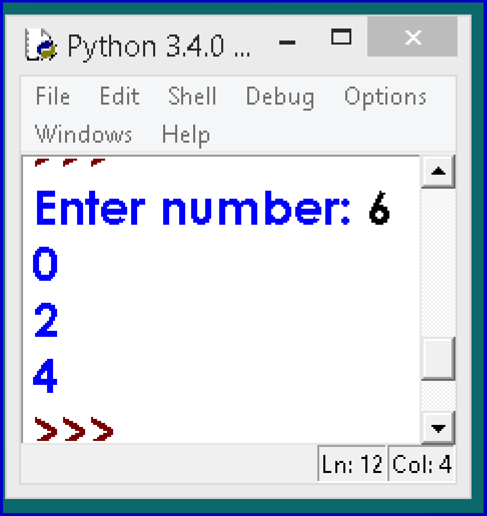
for LOOP - range KEYWORD

#Generating even numbers


*Python 3.4.0: fornat.py - C:/Python34/for...
File Edit Format Run Options Windows Help
Ln: 6 Col: 0

```
def for_loop_example():
    n=int(input("Enter number: "))
    for i in range(0,n,2):
        print(i)
for_loop_example()
```

OUTPUT

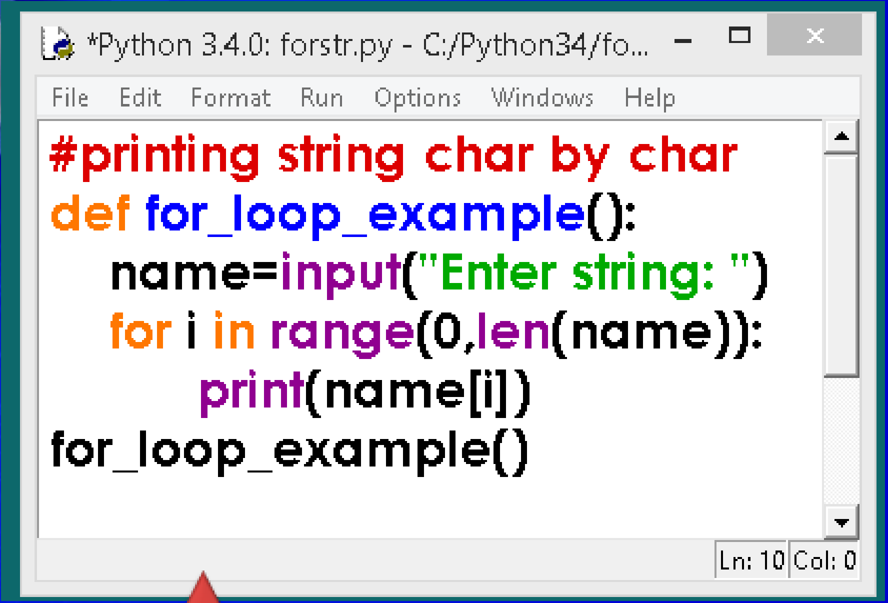


```
Enter number: 6
0
2
4
>>>
```

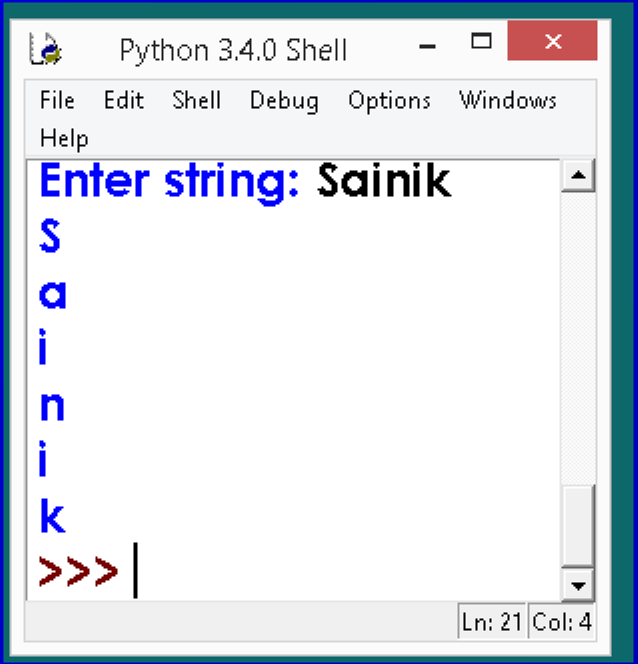
for LOOP – len() FUNCTION

for LOOP - range KEYWORD

# print string character by character


*Python 3.4.0: forstr.py - C:/Python34/fo...
File Edit Format Run Options Windows Help
Ln: 10 Col: 0

```
#printing string char by char
def for_loop_example():
    name=input("Enter string: ")
    for i in range(0,len(name)):
        print(name[i])
for_loop_example()
```

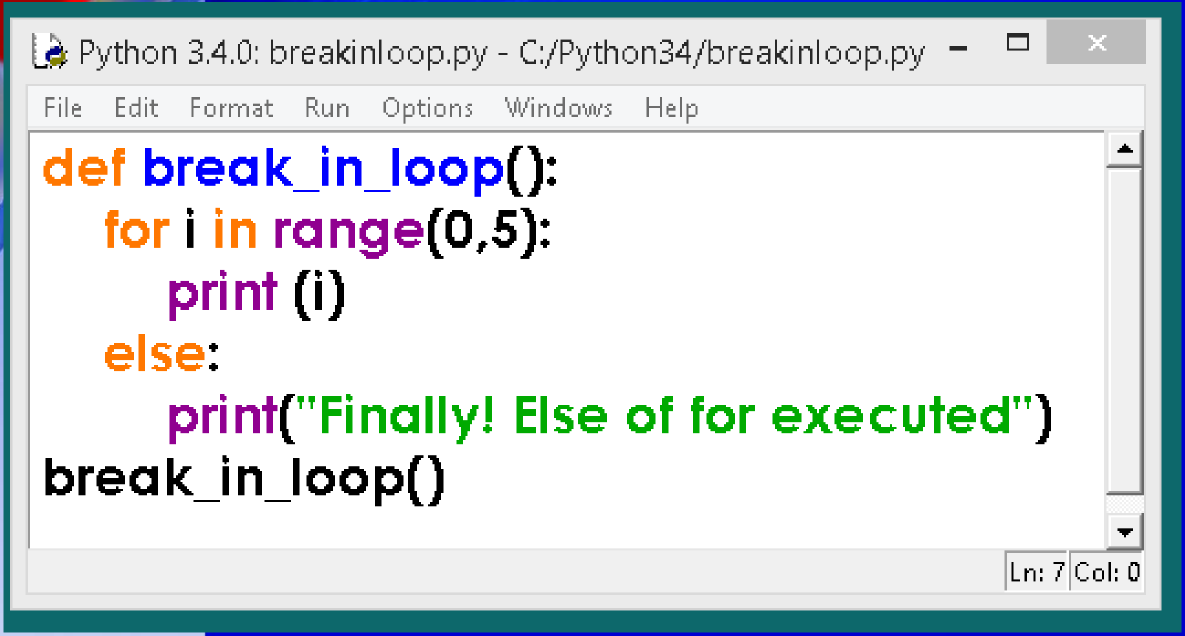


```
Enter string: Sainik
S
a
i
n
i
k
>>> 
```

else statement in loop

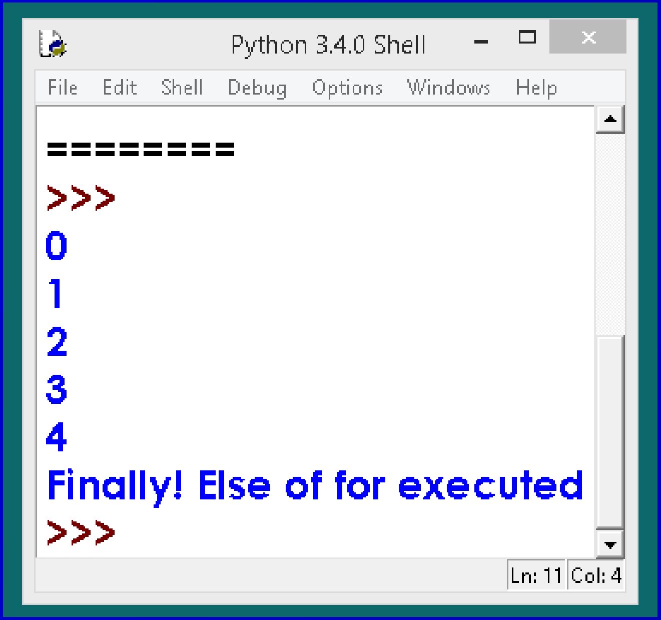
else statement in loop

else can be used in for and while loops
the else body will be executed as and when the loop's conditional expression evaluates to false


Python 3.4.0: breakinloop.py - C:/Python34/breakinloop.py
File Edit Format Run Options Windows Help
Ln: 7 Col: 0

```
def break_in_loop():
    for i in range(0,5):
        print (i)
    else:
        print("Finally! Else of for executed")
break_in_loop()
```

OUTPUT



```
=========
>>>
0
1
2
3
4
Finally! Else of for executed
>>>
```

3. BRANCHING OR JUMPING STATEMENTS

3. BRANCHING OR JUMPING STATEMENTS

Python has an unconditional branching statements and they are,

1. break STATEMENT

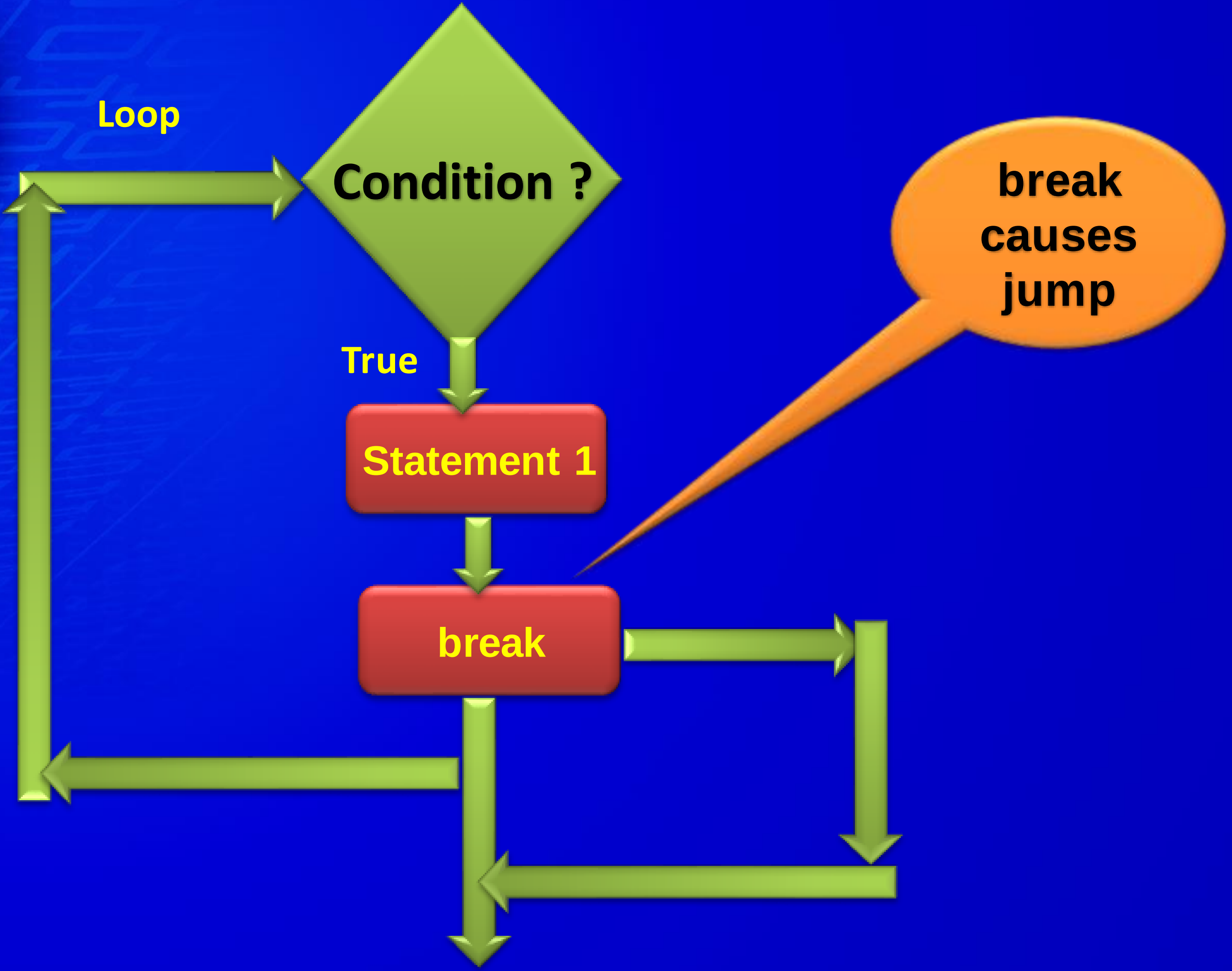
2. continue STATEMENT

3. BRANCHING OR JUMPING STATEMENTS

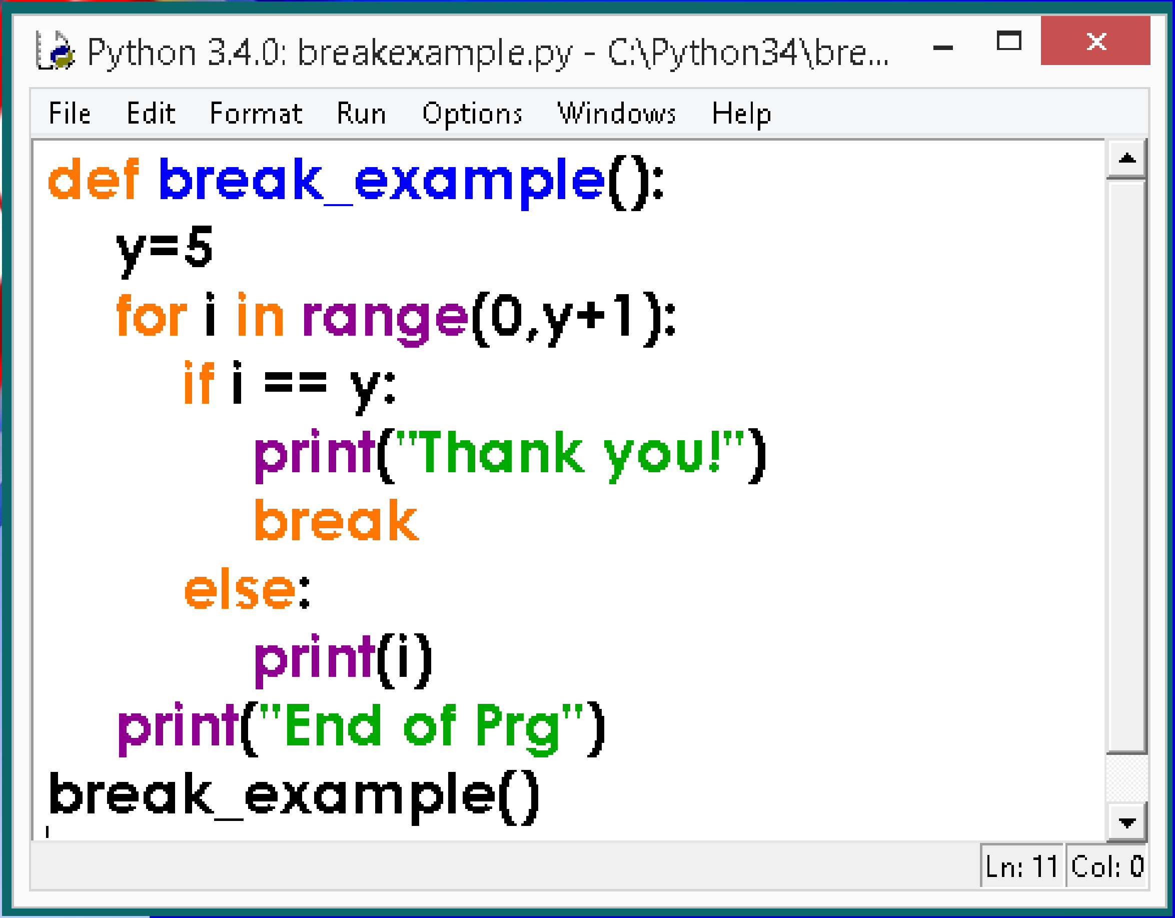
1. break STATEMENT

Break can be used to unconditionally jump out of the loop. It terminates the execution of the loop. Break can be used in while loop and for loop. Break is mostly required, when because of some external condition, we need to exit from a loop.

1. break STATEMENT



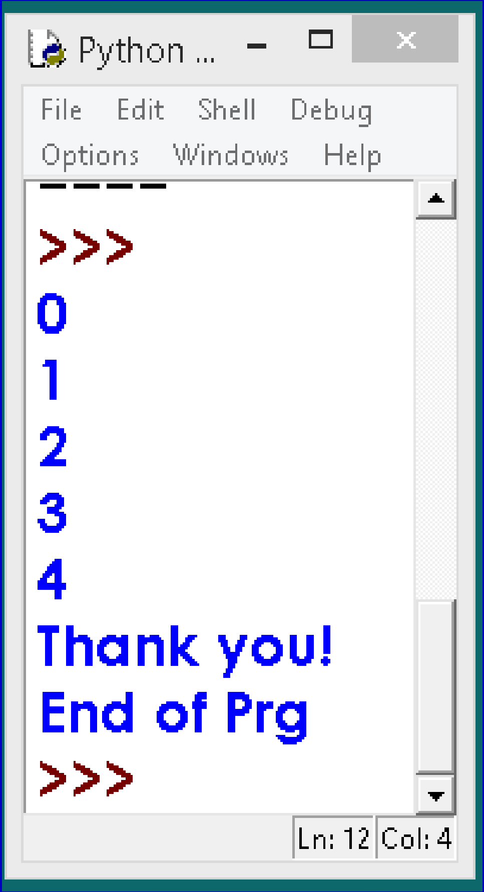
1. break STATEMENT



Python 3.4.0: breakexample.py - C:\Python34\bre...

```
def break_example():
    y=5
    for i in range(0,y+1):
        if i == y:
            print("Thank you!")
            break
        else:
            print(i)
    print("End of Prg")
break_example()
```

OUT PUT



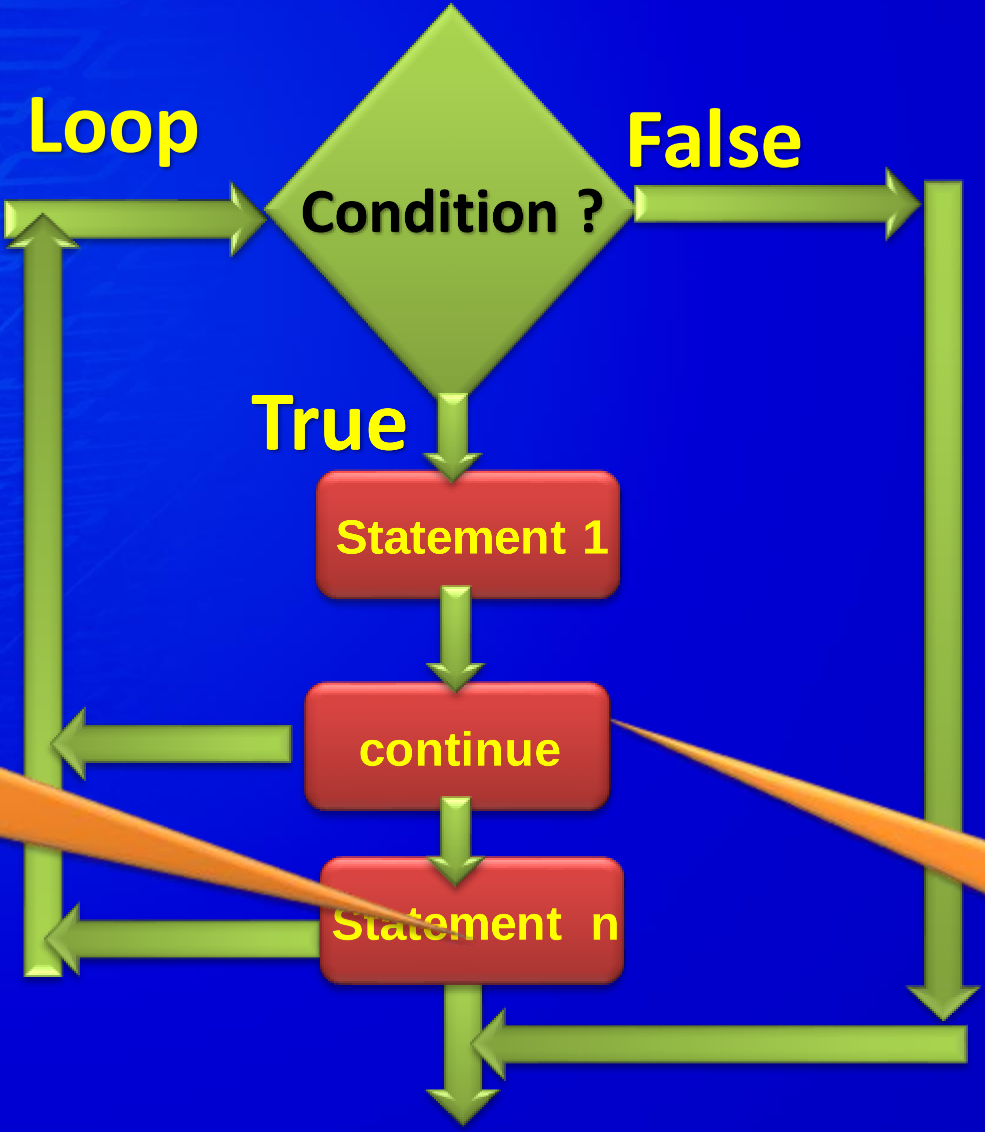
Python ...

```
>>>
0
1
2
3
4
Thank you!
End of Prg
>>>
```

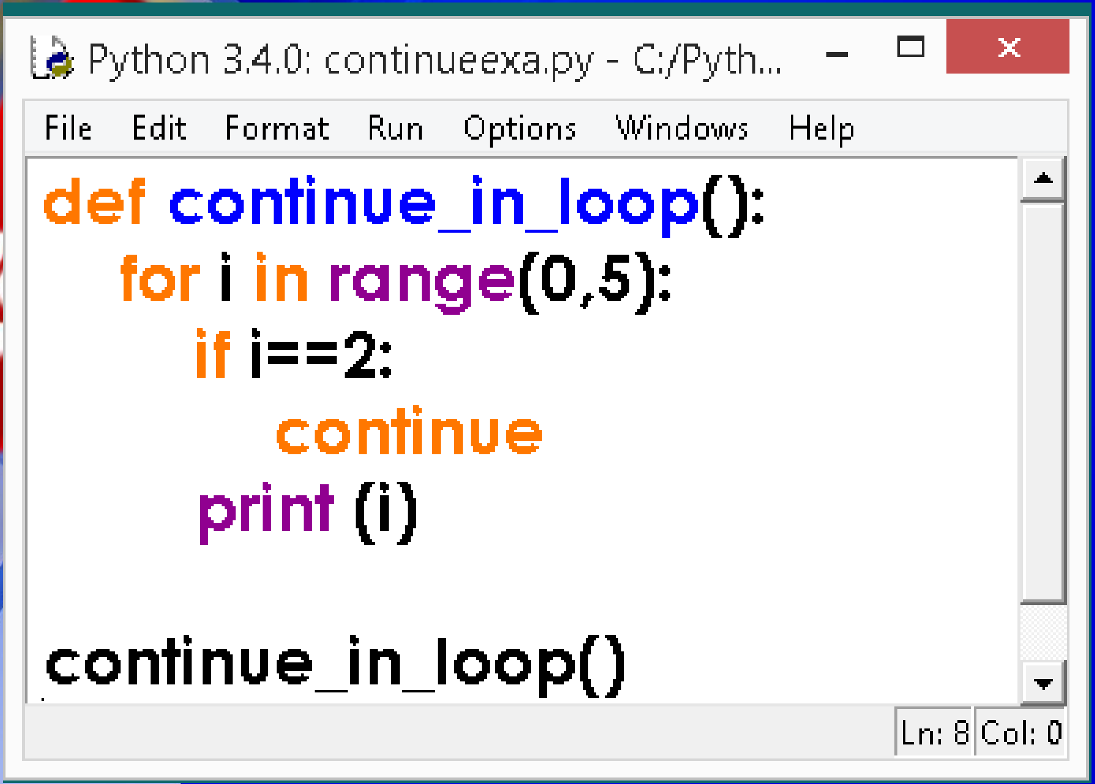
2. continue STATEMENT

The continue statement in Python returns the control to the beginning of the while loop. The continue statement rejects all the remaining statements in the current iteration of the loop and moves the control back to the top of the loop. The continue statement can be used in both while and for loops.

2. continue STATEMENT

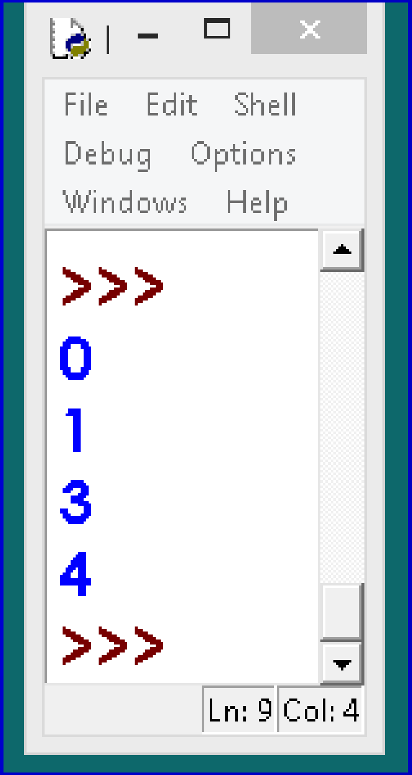


2. continue STATEMENT


Python 3.4.0: continueexa.py - C:/Pyth...
File Edit Format Run Options Windows Help
Ln: 8 Col: 0

```
def continue_in_loop():
    for i in range(0,5):
        if i==2:
            continue
        print (i)

continue_in_loop()
```



```
>>>
0
1
3
4
>>>
```

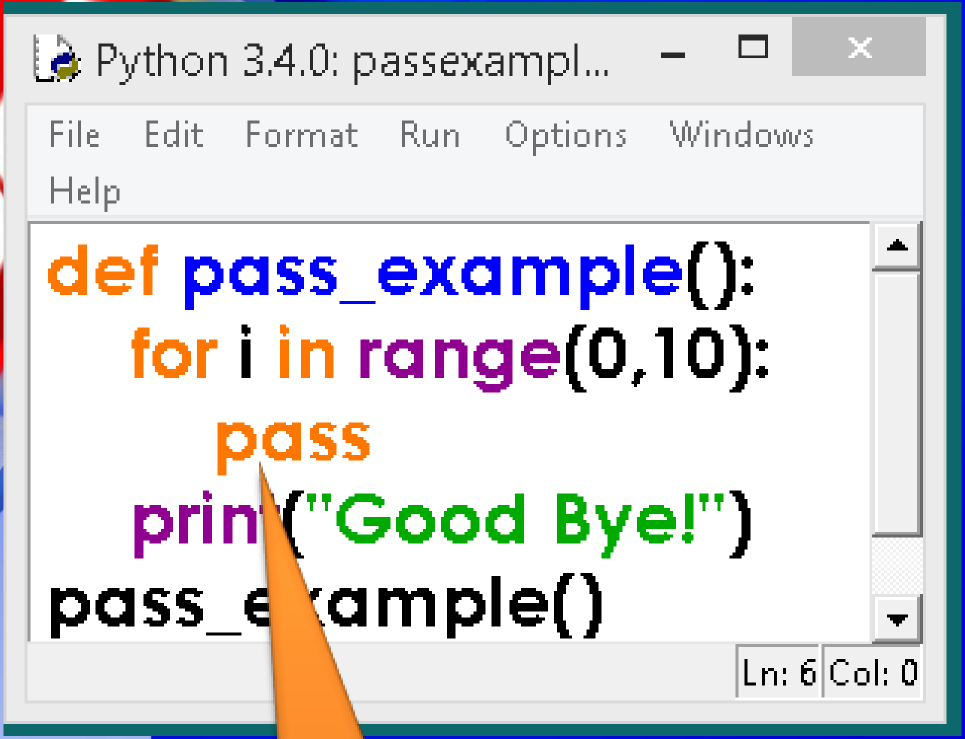
when i value becomes 2 the print statement gets skipped, continue statement goes for next iteration, hence in the out put 2 is not printed

pass STATEMENT

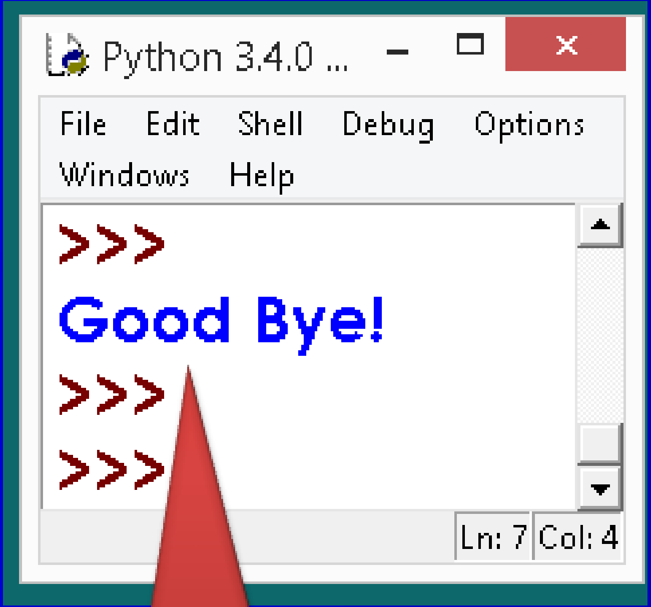
pass STATEMENT

The pass statement in Python is used when a statement is required syntactically but you do not want any command or code to execute.
The pass statement is a *null* operation; nothing happens when it executes.
The pass is also useful in places where your code will eventually go, but has not been written yet (e.g., in stubs for example):

pass STATEMENT


Python 3.4.0: passexampl...
File Edit Format Run Options Windows Help
Ln: 6 Col: 0

```
def pass_example():
    for i in range(0,10):
        pass
    print("Good Bye!")
pass_example()
```

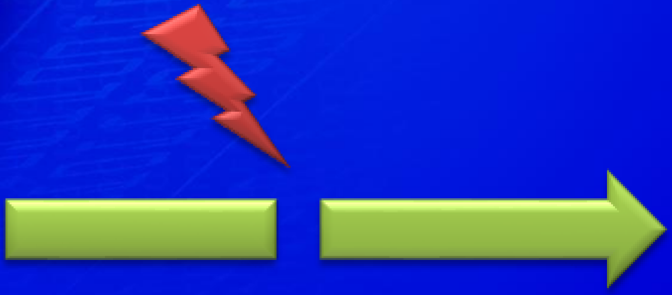
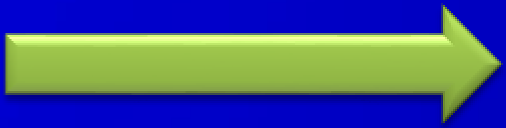


```
>>>
Good Bye!
>>>
>>>
```

pass in loop

pass in loop has no output

Difference Between break and continue

break

continue

Difference Between break and continue

BREAK	CONTINUE
It terminates the execution of remaining iteration of the loop.	It terminates only the current iteration of the loop.
'break' resumes the control of the program to the end of loop enclosing that 'break'.	'continue' resumes the control of the program to the next iteration of that loop enclosing 'continue'.
It causes early termination of loop.	It causes early execution of the next iteration.
'break' stops the continuation of loop.	'continue' do not stops the continuation of loop, it only stops the current iteration.