

UNIT 2: PYTHON OPERATORS AND CONTROL FLOW STATEMENTS

BASIC OPERATORS:

- **Arithmetic operators**

+(addition) - (subtraction) *(multiplication)
/(divide) %(remainder) //(floor division) exponent (**)

- **Comparison operators**

== != <= >= > <

- **Assignment Operators**

= += -= *= %= **= //=

- **Logical Operators**

and or not

- **Bitwise Operators**

& (binary and) | (binary or) ^ (binary xor)
<< (left shift) >> (right shift) ~ (negation)

- **Membership Operators**

in not in

- **Identity Operators**

is is not

1. Arithmetic operators

Operator	Description
+ (Addition)	It is used to add two operands. For example, if $a = 20$, $b = 10 \Rightarrow a + b = 30$
- (Subtraction)	It is used to subtract the second operand from the first operand. If the first operand is less than the second operand, the value result negative. For example, if $a = 20$, $b = 10 \Rightarrow a - b = 10$
/ (divide)	It returns the quotient after dividing the first operand by the second operand. For example, if $a = 20$, $b = 10 \Rightarrow a / b = 2$
* (Multiplication)	It is used to multiply one operand with the other. For example, if $a = 20$, $b = 10 \Rightarrow a * b = 200$
% (reminder)	It returns the reminder after dividing the first operand by the second operand. For example, if $a = 20$, $b = 10 \Rightarrow a \% b = 0$
** (Exponent)	It is an exponent operator represented as it calculates the first operand power to second operand.
// (Floor division)	It gives the floor value of the quotient produced by dividing the two operands.

2. Comparison/Relational operators

Operator	Description
==	If the value of two operands is equal, then the condition becomes true.
!=	If the value of two operands is not equal then the condition becomes true.
<=	If the first operand is less than or equal to the second operand, then the condition becomes true.
>=	If the first operand is greater than or equal to the second operand, then the condition becomes true.

CO: DEVELOP PYTHON PROGRAM TO DEMONSTRATE USE OF OPERATORS

>	If the first operand is greater than the second operand, then the condition becomes true.
<	If the first operand is less than the second operand, then the condition becomes true.

3. Assignment operator

Operator	Description
=	It assigns the the value of the right expression to the left operand.
+=	It increases the value of the left operand by the value of the right operand and assign the modified value back to left operand. For example, if a = 10, b = 20 => a+ = b will be equal to a = a+ b and therefore, a = 30.
-=	It decreases the value of the left operand by the value of the right operand and assign the modified value back to left operand. For example, if a = 20, b = 10 => a- = b will be equal to a = a- b and therefore, a = 10.
=	It multiplies the value of the left operand by the value of the right operand and assign the modified value back to left operand. For example, if a = 10, b = 20 => a = b will be equal to a = a* b and therefore, a = 200.
%=	It divides the value of the left operand by the value of the right operand and assign the remainder back to left operand. For example, if a = 20, b = 10 => a % = b will be equal to a = a % b and therefore, a = 0.
=	a=b will be equal to a=a**b, for example, if a = 4, b =2, a**=b will assign 4**2 = 16 to a.
//=	A//=b will be equal to a = a// b, for example, if a = 4, b = 3, a//=b will assign 4//3 = 1 to a.

4. Logical operators

Operator	Description
and	If both the expression are true, then the condition will be true. If a and b are the two expressions, $a \rightarrow \text{true}$, $b \rightarrow \text{true} \Rightarrow a \text{ and } b \rightarrow \text{true}$.
or	If one of the expressions is true, then the condition will be true. If a and b are the two expressions, $a \rightarrow \text{true}$, $b \rightarrow \text{false} \Rightarrow a \text{ or } b \rightarrow \text{true}$.
not	If an expression a is true then not (a) will be false and vice versa.

5. Bitwise operators

The bitwise operators perform bit by bit operation on the values of the two operands.

For example,

```
if a = 7;
b = 6;
then, binary (a) = 0111
    binary (b) = 0011

hence, a & b = 0011
    a | b = 0111
    a ^ b = 0100
    ~ a = 1000
```

Operator	Description
& (binary and)	If both the bits at the same place in two operands are 1, then 1 is copied to the result. Otherwise, 0 is copied.
(binary or)	The resulting bit will be 0 if both the bits are zero otherwise the resulting bit will be 1.
^ (binary xor)	The resulting bit will be 1 if both the bits are different otherwise the resulting bit will be 0.

CO: DEVELOP PYTHON PROGRAM TO DEMONSTRATE USE OF OPERATORS

~ (negation)	It calculates the negation of each bit of the operand, i.e., if the bit is 0, the resulting bit will be 1 and vice versa.
<< (left shift)	The left operand value is moved left by the number of bits present in the right operand.
>> (right shift)	The left operand is moved right by the number of bits present in the right operand.

6. Membership operators

Python membership operators are used to check the membership of value inside a Python data structure. If the value is present in the data structure, then the resulting value is true otherwise it returns false.

Operator	Description
in	It is evaluated to be true if the first operand is found in the second operand (list, tuple, or dictionary).
not in	It is evaluated to be true if the first operand is not found in the second operand (list, tuple, or dictionary).

7. Identity Operators

Operator	Description
is	It is evaluated to be true if the reference present at both sides point to the same object.
is not	It is evaluated to be true if the reference present at both side do not point to the same object.

8. Python Operator Precedence

The precedence of the operators is important to find out since it enables us to know which operator should be evaluated first. The precedence table of the operators in python is given below.

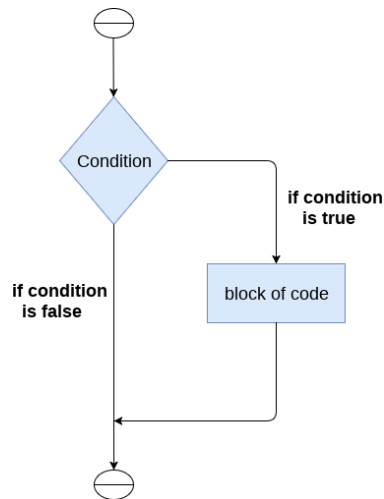
Operator	Description
**	The exponent operator is given priority over all the others used in the expression.
~ + -	The negation, unary plus and minus.
* / % //	The multiplication, divide, modules, reminder, and floor division.
+ -	Binary plus and minus
>> <<	Left shift and right shift
&	Binary and.
^	Binary xor and or
<= < > >=	Comparison operators (less then, less then equal to, greater then, greater then equal to).
<> == !=	Equality operators.
= %= /= //= -= += *= **=	Assignment operators
is is not	Identity operators
in not in	Membership operators
not or and	Logical operators

2.2 control flow:

2.3 conditional statements (if,if.....else,nested if)

The if statement

The if statement is used to test a particular condition and if the condition is true, it executes a block of code known as if-block. The condition of if statement can be any valid logical expression which can be either evaluated to true or false.

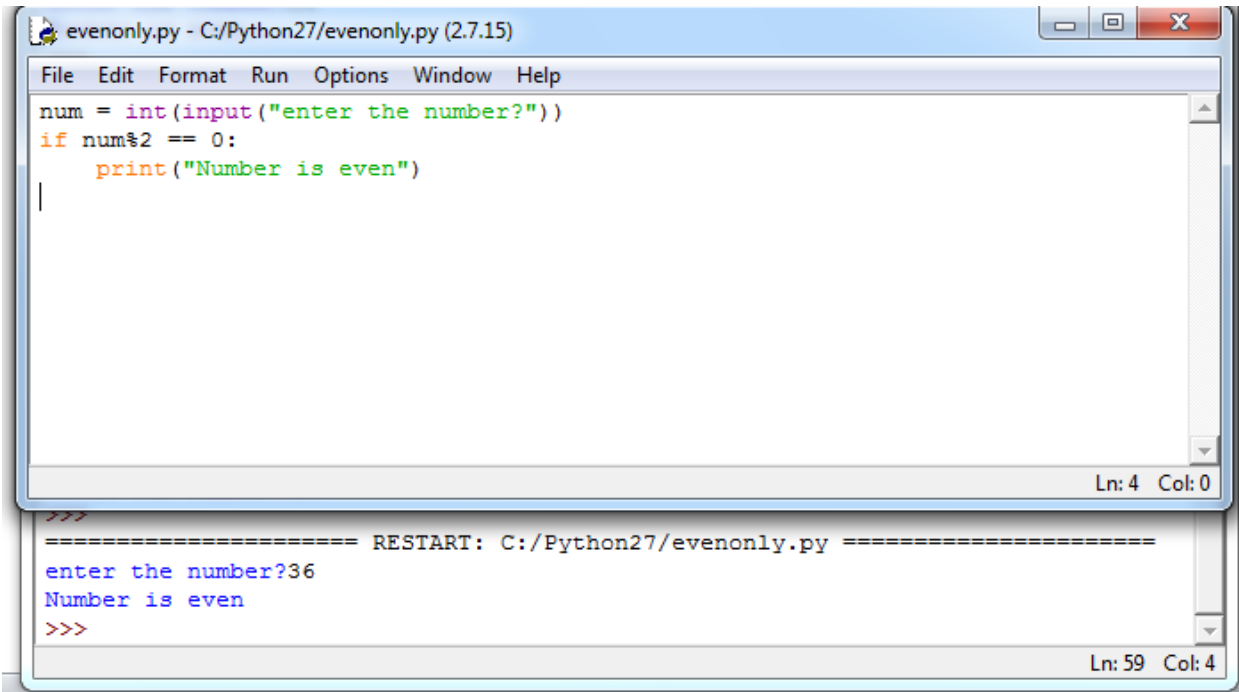


The syntax of the if-statement is given below.

```
if expression:  
    statement
```

CO: DEVELOP PYTHON PROGRAM TO DEMONSTRATE USE OF OPERATORS

Example



The screenshot shows a Python IDE window titled "evenonly.py - C:/Python27/evenonly.py (2.7.15)". The code in the editor is:

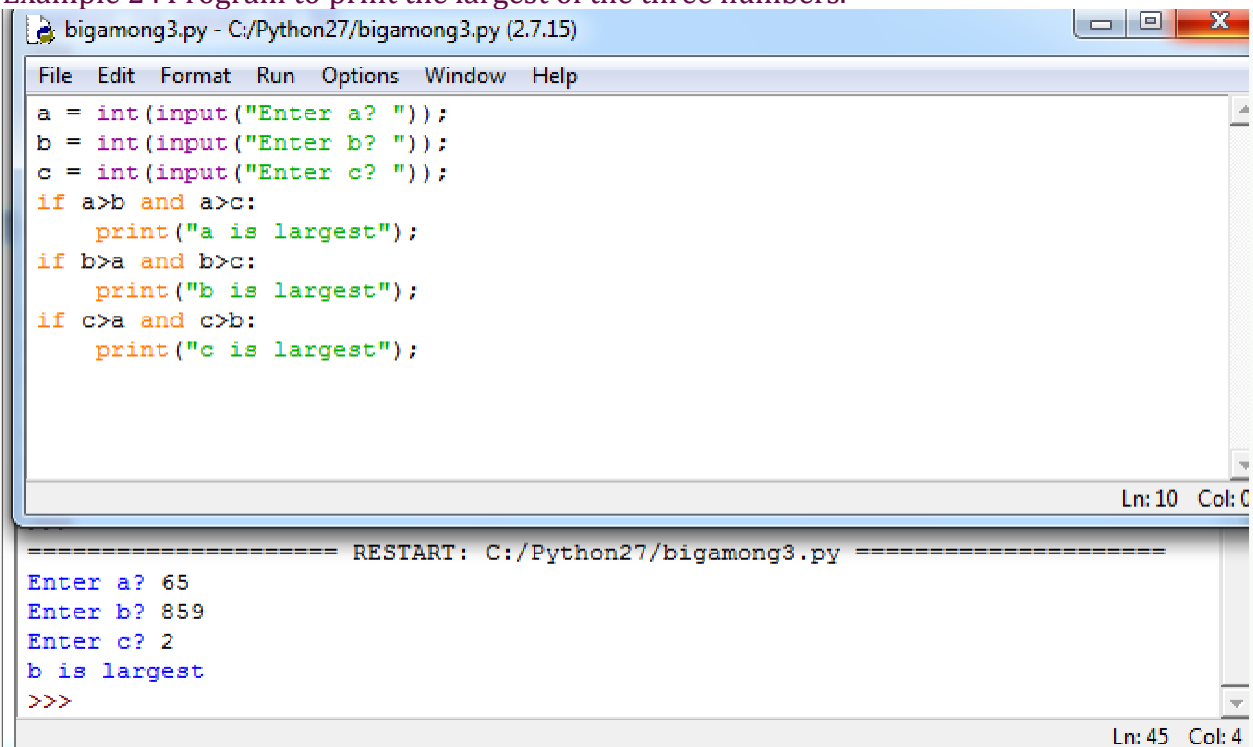
```
File Edit Format Run Options Window Help
num = int(input("enter the number?"))
if num%2 == 0:
    print("Number is even")
|
```

The status bar at the bottom right indicates "Ln: 4 Col: 0". Below the editor, the output console shows the execution results:

```
===== RESTART: C:/Python27/evenonly.py =====
enter the number?36
Number is even
>>>
```

The status bar at the bottom right of the output console indicates "Ln: 59 Col: 4".

Example 2 : Program to print the largest of the three numbers.



The screenshot shows a Python IDE window titled "bigamong3.py - C:/Python27/bigamong3.py (2.7.15)". The code in the editor is:

```
File Edit Format Run Options Window Help
a = int(input("Enter a? "));
b = int(input("Enter b? "));
c = int(input("Enter c? "));
if a>b and a>c:
    print("a is largest");
if b>a and b>c:
    print("b is largest");
if c>a and c>b:
    print("c is largest");
```

The status bar at the bottom right indicates "Ln: 10 Col: 0". Below the editor, the output console shows the execution results:

```
===== RESTART: C:/Python27/bigamong3.py =====
Enter a? 65
Enter b? 859
Enter c? 2
b is largest
>>>
```

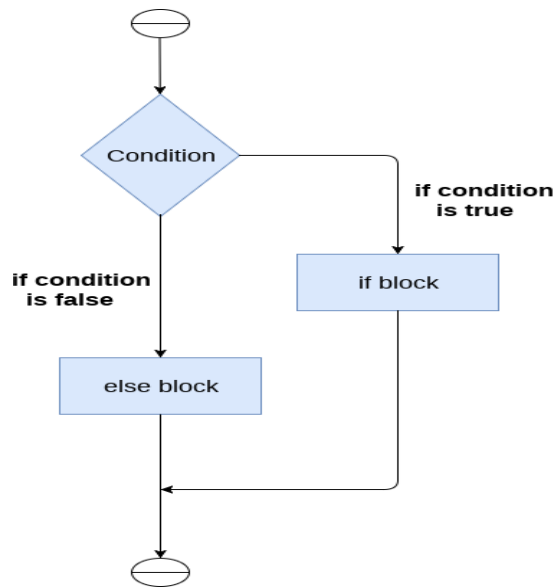
The status bar at the bottom right of the output console indicates "Ln: 45 Col: 4".

The if-else statement

The if-else statement provides an else block combined with the if statement which is executed in the false case of the condition.

CO: DEVELOP PYTHON PROGRAM TO DEMONSTRATE USE OF OPERATORS

If the condition is true, then the if-block is executed. Otherwise, the else-block is executed.



The syntax of the if-else statement is given below.

if condition:

#block of statements

else:

#another block of statements (else-block)

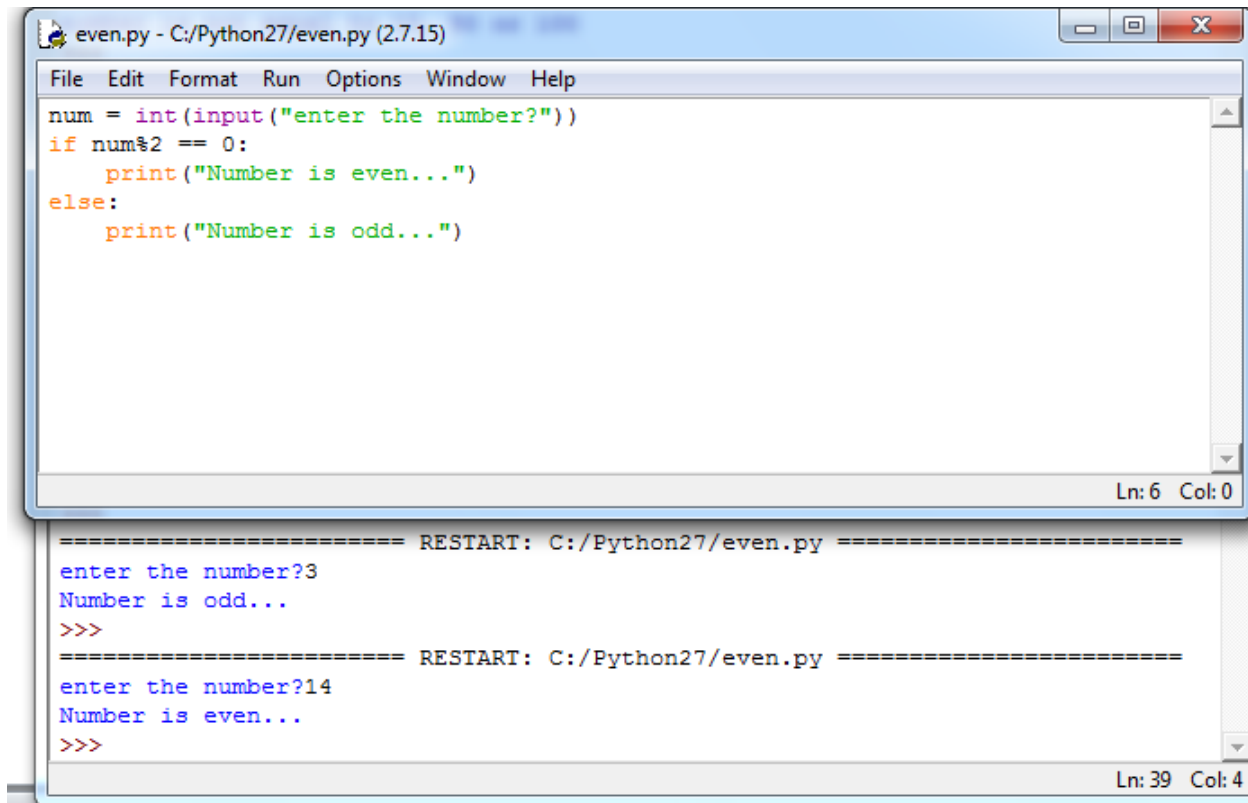
Example 1 : Program to check whether a person is eligible to vote or not.

```
File Edit Format Run Options Window Help
age = int(input("Enter your age? "))
if age >= 18:
    print("You are eligible to vote !!");
else:
    print("Sorry! you have to wait !!");
|
```

Ln: 6 Col: 0

```
===== RESTART: C:/Python27/vote.py =====
Enter your age? 32
You are eligible to vote !!
>>>
===== RESTART: C:/Python27/vote.py =====
Enter your age? 15
Sorry! you have to wait !!
>>>
```

Example 2: Program to check whether a number is even or not.



```
even.py - C:/Python27/even.py (2.7.15)
File Edit Format Run Options Window Help
num = int(input("enter the number?"))
if num%2 == 0:
    print("Number is even...")
else:
    print("Number is odd...")

===== RESTART: C:/Python27/even.py =====
enter the number?3
Number is odd...
>>>

===== RESTART: C:/Python27/even.py =====
enter the number?14
Number is even...
>>>
```

The elif statement

The elif statement enables us to check multiple conditions and execute the specific block of statements depending upon the true condition among them.

However, using elif is optional.

The syntax of the elif statement is given below.

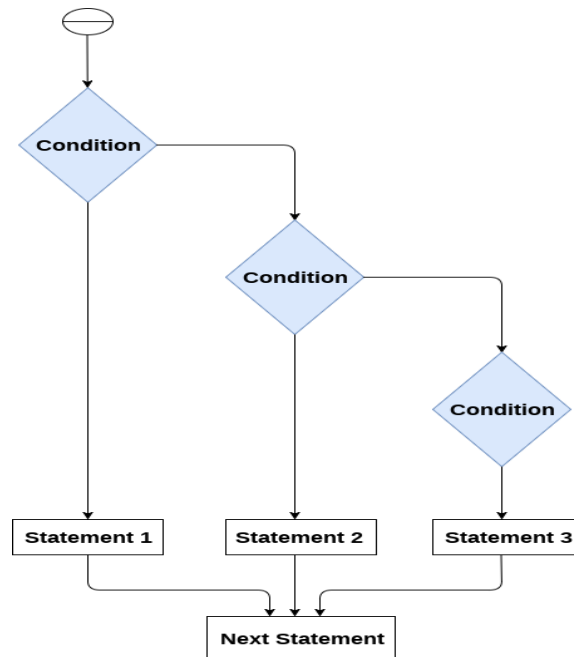
if expression 1:
 # block of statements

elif expression 2:
 # block of statements

elif expression 3:
 # block of statements

else:
 # block of statements

CO: DEVELOP PYTHON PROGRAM TO DEMONSTRATE USE OF OPERATORS



Example 1

```
File Edit Format Run Options Window Help
number = int(input("Enter the number?"))
if number==10:
    print("number is equals to 10")
elif number==50:
    print("number is equal to 50");
elif number==100:
    print("number is equal to 100");
else:
    print("number is not equal to 10, 50 or 100");

Ln: 10 Col: 0

===== RESTART: C:/Python27/elif1.py =====
Enter the number?36
number is not equal to 10, 50 or 100
>>>

===== RESTART: C:/Python27/elif1.py =====
Enter the number?20
number is not equal to 10, 50 or 100
>>>

===== RESTART: C:/Python27/elif1.py =====
Enter the number?50
number is equal to 50
>>>
```

Example 2

```

elif.py - C:/Python27/elif.py (2.7.15)
File Edit Format Run Options Window Help
marks = int(input("Enter the marks? "))
if marks > 85 and marks <= 100:
    print("Congrats ! you scored grade A ...")
elif marks > 60 and marks <= 85:
    print("You scored grade B + ...")
elif marks > 40 and marks <= 60:
    print("You scored grade B ...")
elif (marks > 30 and marks <= 40):
    print("You scored grade C ...")
else:
    print("Sorry you are fail ?")

Ln: 8 Col: 36

>>>
===== RESTART: C:/Python27/elif.py =====
Enter the marks? 33
You scored grade C ...
>>>
===== RESTART: C:/Python27/elif.py =====
Enter the marks? 99
Congrats ! you scored grade A ...
>>>

```

2.4 looping in python (while loop,for loop,nested loops)

There are the following advantages of loops in Python.

1. It provides code re-usability.
2. Using loops, we do not need to write the same code again and again.
3. Using loops, we can traverse over the elements of data structures (array or linked lists).

There are the following loop statements in Python.

Loop Statement	Description
for loop	The for loop is used in the case where we need to execute some part of the code until the given condition is satisfied. The for loop is also called as a per-tested loop. It is better to use for loop if the number of iteration is known in advance.

CO: DEVELOP PYTHON PROGRAM TO DEMONSTRATE USE OF OPERATORS

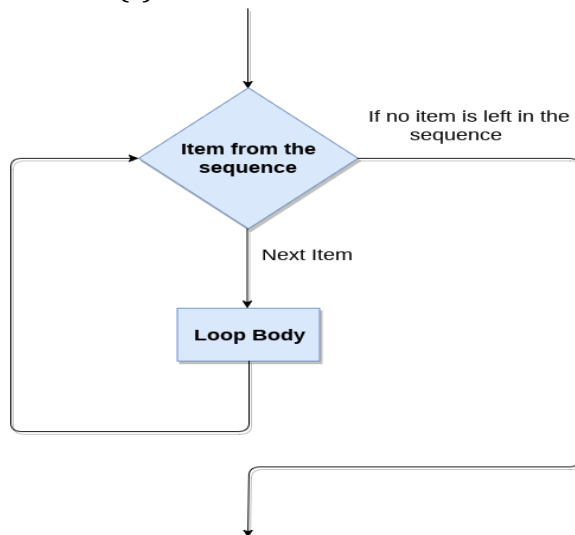
while loop	The while loop is to be used in the scenario where we don't know the number of iterations in advance. The block of statements is executed in the while loop until the condition specified in the while loop is satisfied. It is also called a pre-tested loop.
do-while loop	The do-while loop continues until a given condition satisfies. It is also called post tested loop. It is used when it is necessary to execute the loop at least once (mostly menu driven programs).

Python for loop

The for **loop in Python** is used to iterate the statements or a part of the program several times. It is frequently used to traverse the data structures like list, tuple, or dictionary.

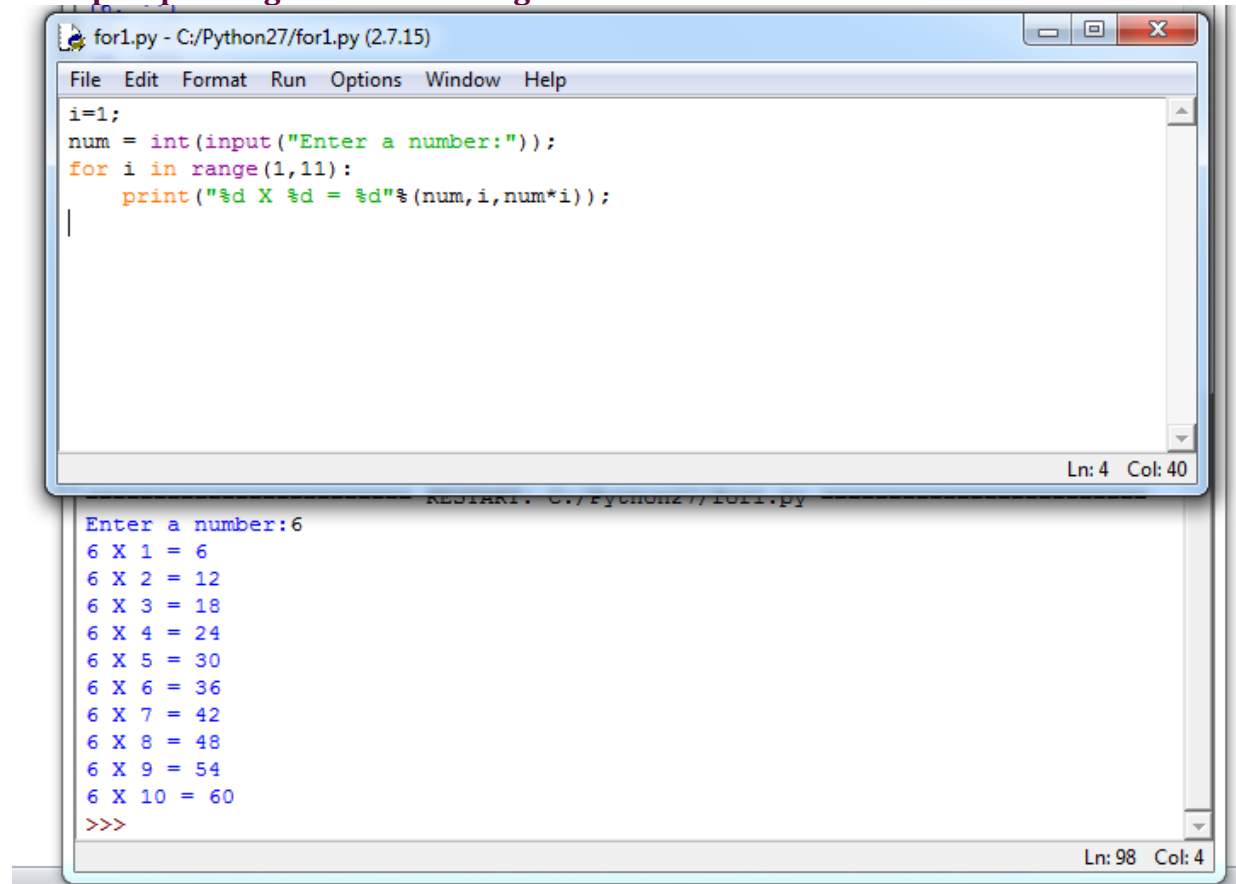
The syntax of for loop in python is given below.

for iterating_var **in** sequence:
statement(s)



CO: DEVELOP PYTHON PROGRAM TO DEMONSTRATE USE OF OPERATORS

example : printing the table of the given number



```
for1.py - C:/Python27/for1.py (2.7.15)
File Edit Format Run Options Window Help
i=1;
num = int(input("Enter a number:"));
for i in range(1,11):
    print("%d X %d = %d"%(num,i,num*i));
|

Ln: 4 Col: 40

Restart: C:/Python27/for1.py
Enter a number:6
6 X 1 = 6
6 X 2 = 12
6 X 3 = 18
6 X 4 = 24
6 X 5 = 30
6 X 6 = 36
6 X 7 = 42
6 X 8 = 48
6 X 9 = 54
6 X 10 = 60
>>>

Ln: 98 Col: 4
```

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 of the nested for loop in python is given below.

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

Example 1

```
n = int(input("Enter the number of rows you want to print?"))
i,j=0,0
for i in range(0,n):
    print()
    for j in range(0,i+1):
        print("*",end="")
```

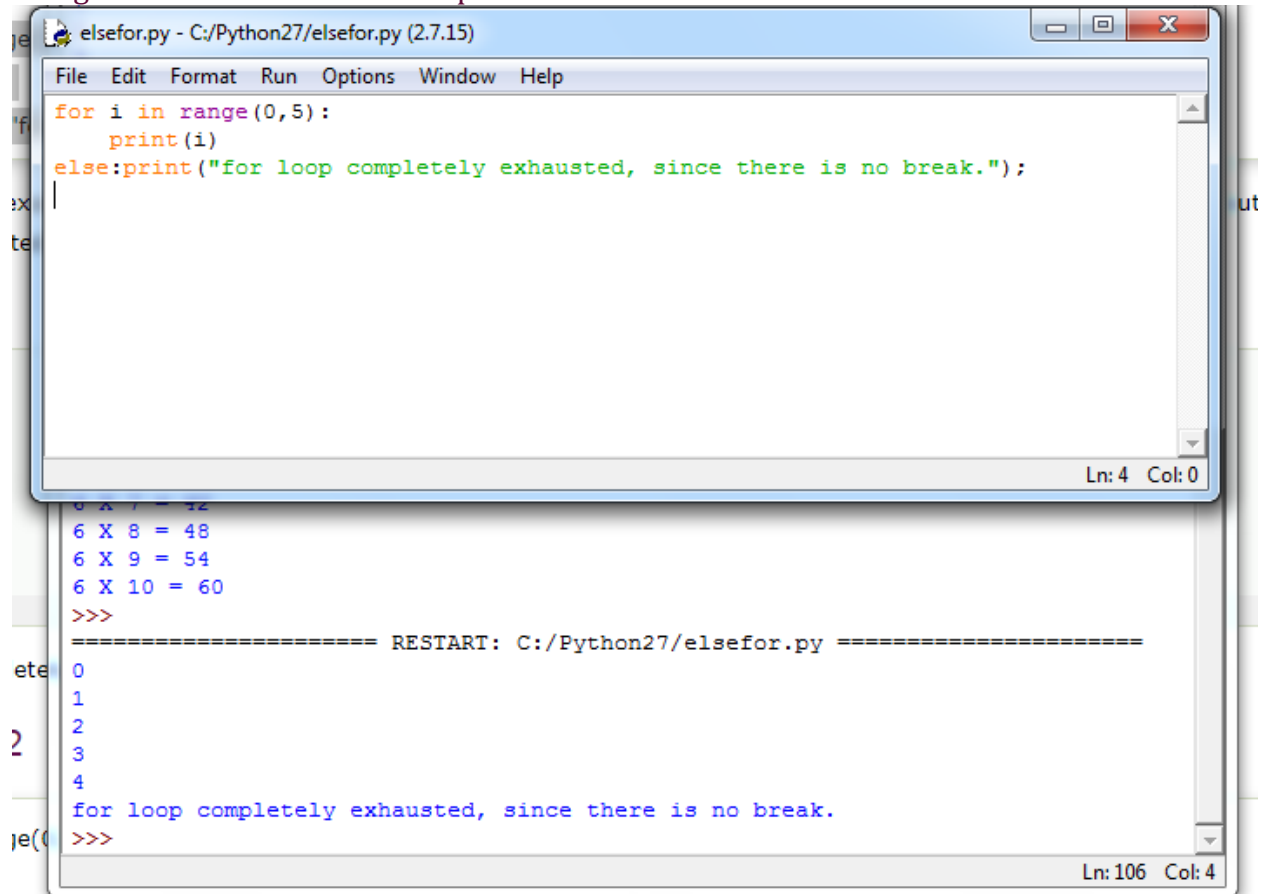
CO: DEVELOP PYTHON PROGRAM TO DEMONSTRATE USE OF OPERATORS

Output:

Enter the number of rows you want to print?5

```
*  
**  
***  
****
```

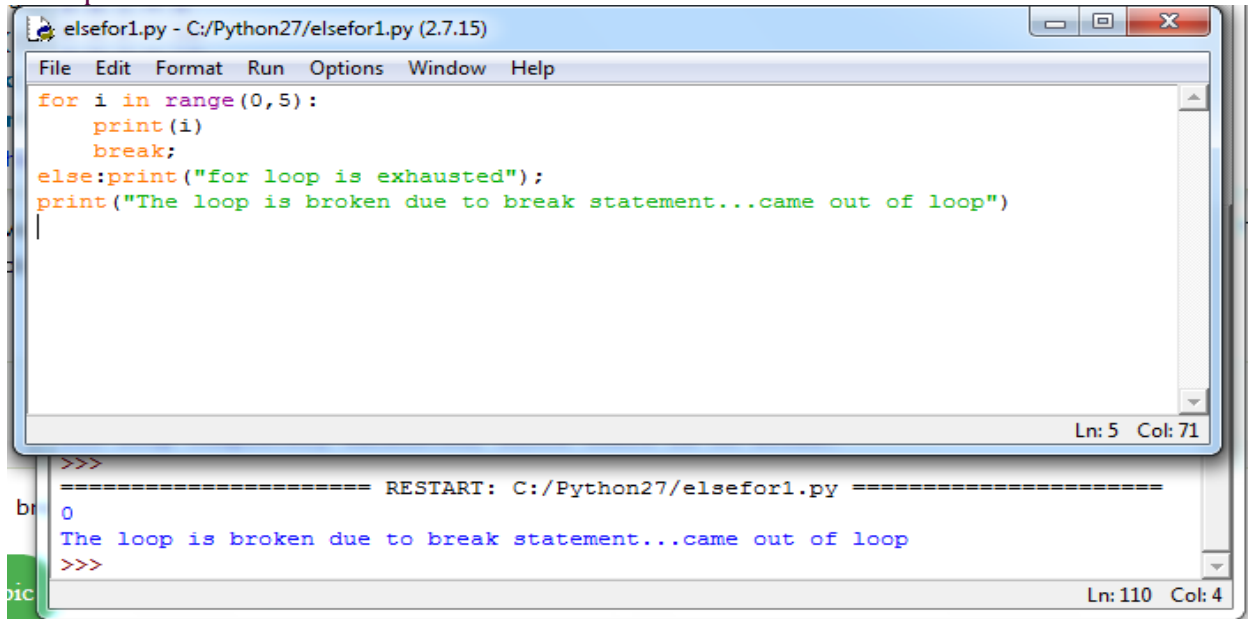
Using else statement with for loop



```
File Edit Format Run Options Window Help  
for i in range(0,5):  
    print(i)  
else:print("for loop completely exhausted, since there is no break.");  
|  
Ln: 4 Col: 0  
  
6 X 7 = 42  
6 X 8 = 48  
6 X 9 = 54  
6 X 10 = 60  
>>>  
===== RESTART: C:/Python27/elsefor.py =====  
0  
1  
2  
3  
4  
for loop completely exhausted, since there is no break.  
>>>  
Ln: 106 Col: 4
```

CO: DEVELOP PYTHON PROGRAM TO DEMONSTRATE USE OF OPERATORS

Example 2

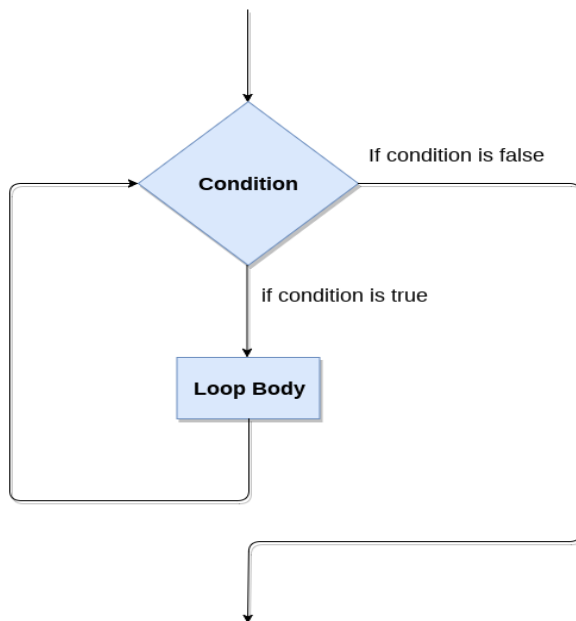


```
elsefor1.py - C:/Python27/elsefor1.py (2.7.15)
File Edit Format Run Options Window Help
for i in range(0,5):
    print(i)
    break;
else:print("for loop is exhausted");
print("The loop is broken due to break statement...came out of loop")

>>>
===== RESTART: C:/Python27/elsefor1.py =====
0
The loop is broken due to break statement...came out of loop
>>>
```

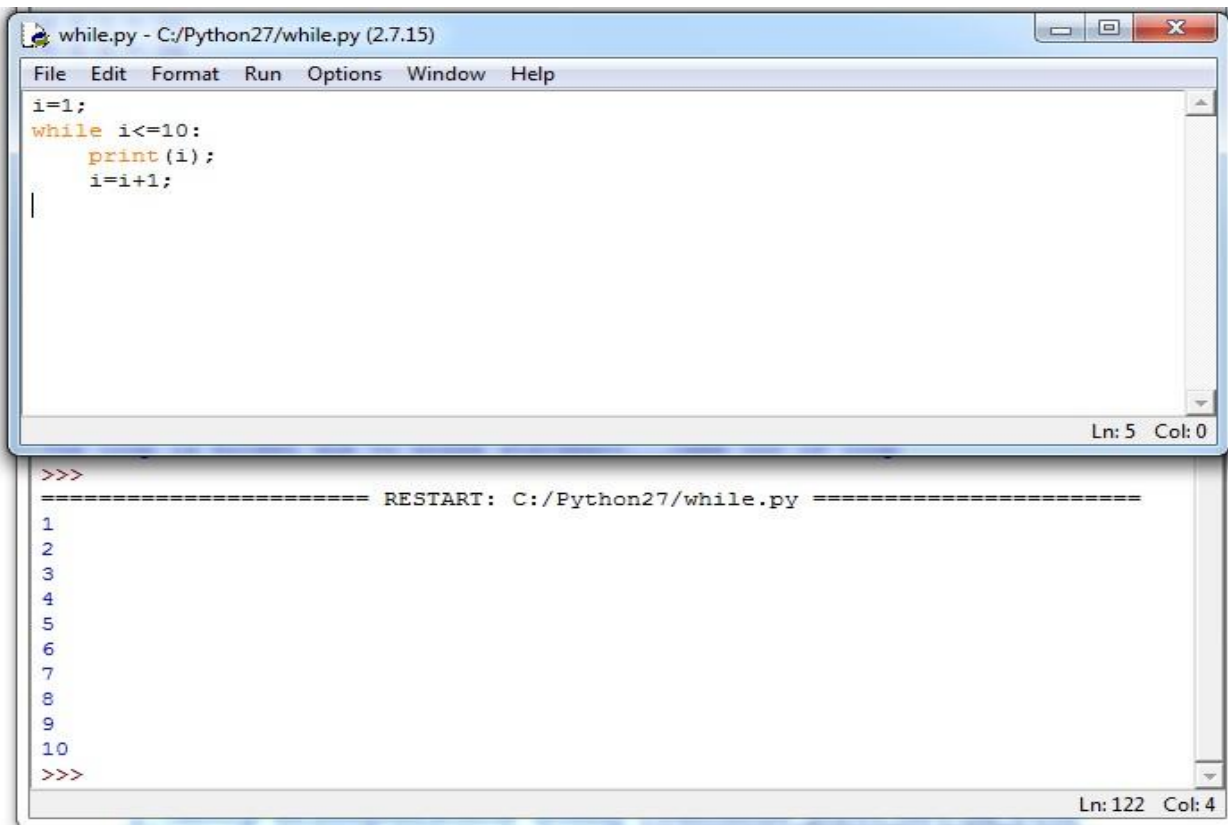
Python while loop

while expression:
statements



Example 1

CO: DEVELOP PYTHON PROGRAM TO DEMONSTRATE USE OF OPERATORS



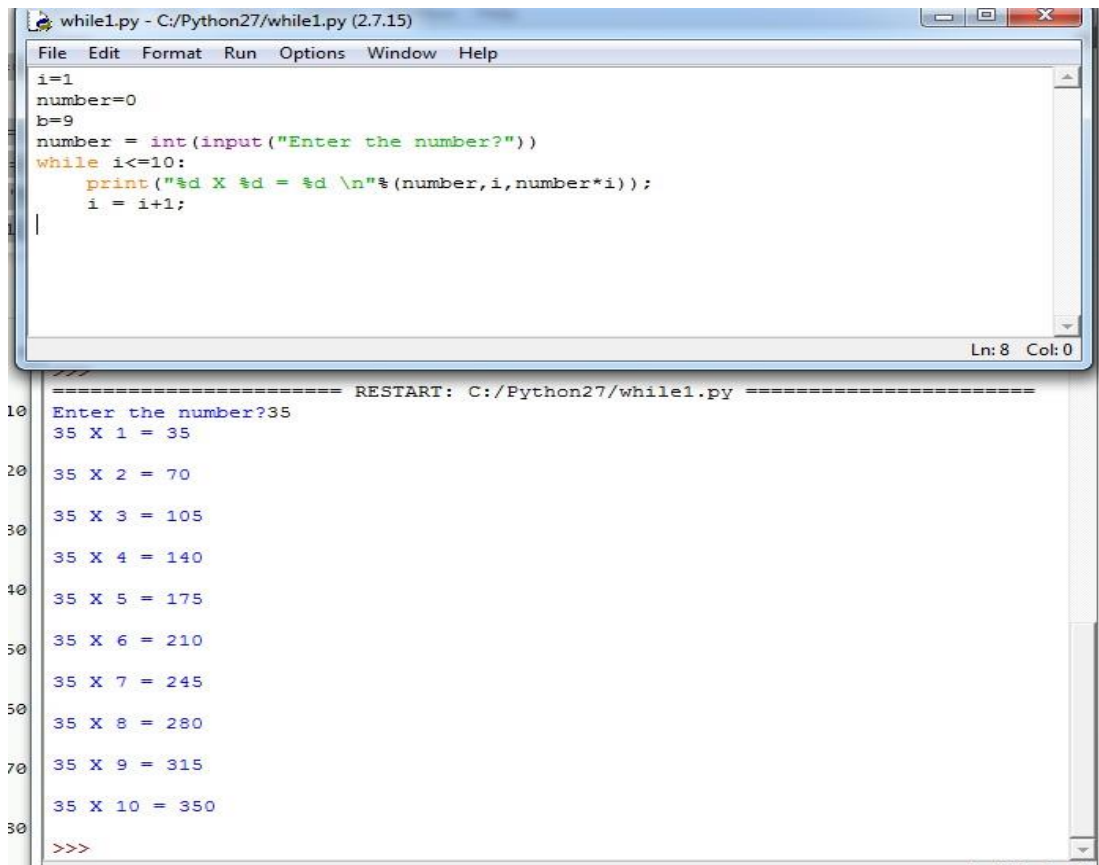
The screenshot shows a Python IDE window titled 'while.py - C:/Python27/while.py (2.7.15)'. The menu bar includes File, Edit, Format, Run, Options, Window, and Help. The code editor contains the following Python code:

```
i=1;
while i<=10:
    print(i);
    i=i+1;
```

The status bar at the bottom right indicates 'Ln: 5 Col: 0'. Below the code editor, the output console shows the execution results:

```
>>>
===== RESTART: C:/Python27/while.py =====
1
2
3
4
5
6
7
8
9
10
>>>
```

The output console status bar at the bottom right indicates 'Ln: 122 Col: 4'.



The screenshot shows a Python IDE window titled 'while1.py - C:/Python27/while1.py (2.7.15)'. The menu bar includes File, Edit, Format, Run, Options, Window, and Help. The code editor contains the following Python code:

```
i=1
number=0
b=9
number = int(input("Enter the number?"))
while i<=10:
    print("%d X %d = %d \n"%(number,i,number*i));
    i = i+1;
```

The status bar at the bottom right indicates 'Ln: 8 Col: 0'. Below the code editor, the output console shows the execution results:

```
>>>
===== RESTART: C:/Python27/while1.py =====
10 Enter the number?35
    35 X 1 = 35
20 35 X 2 = 70
30 35 X 3 = 105
40 35 X 4 = 140
50 35 X 5 = 175
60 35 X 6 = 210
70 35 X 7 = 245
80 35 X 8 = 280
90 35 X 9 = 315
30 35 X 10 = 350
>>>
```

The output console status bar at the bottom right indicates 'Ln: 115 Col: 4'.

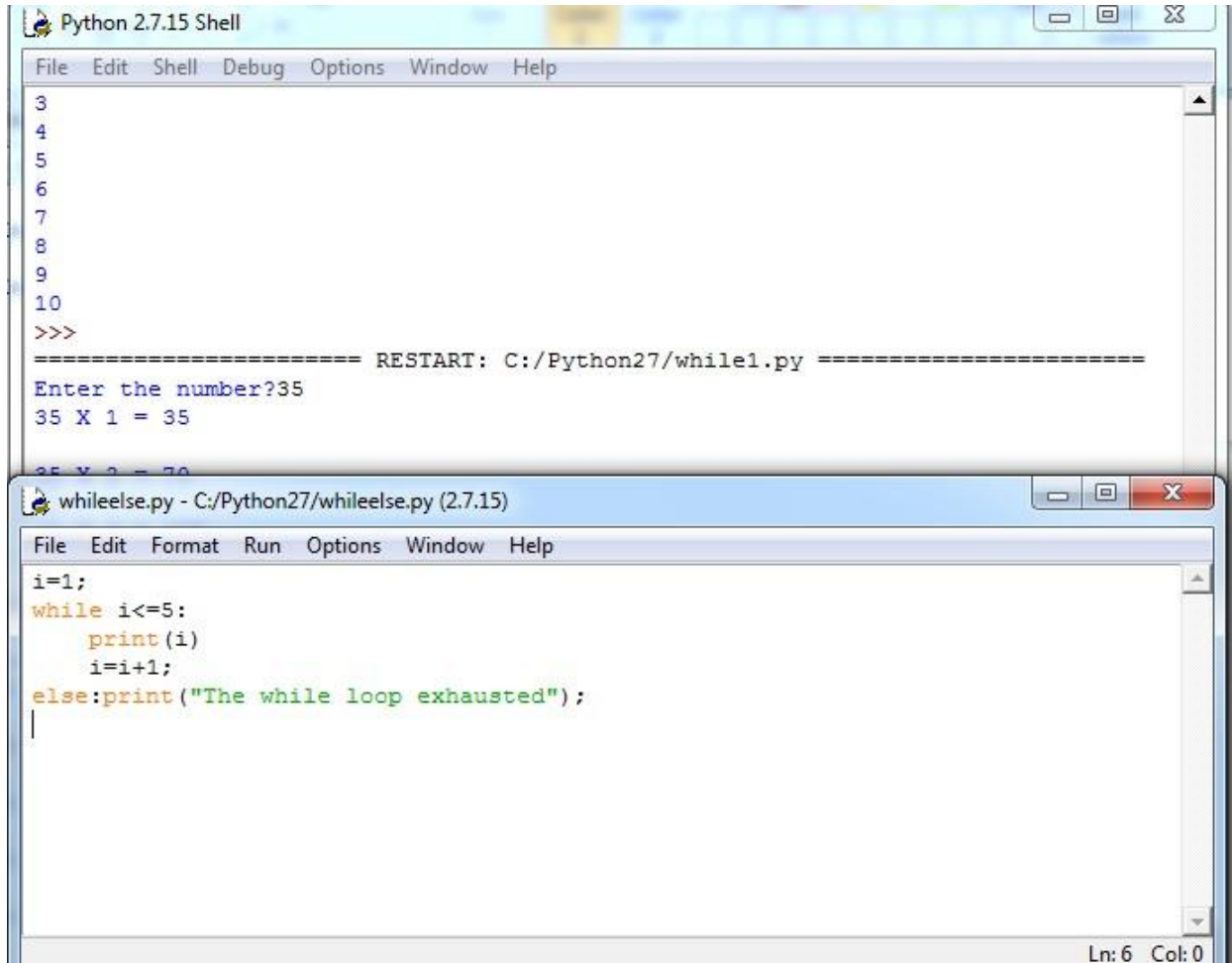
Example 2:

Using else with Python while loop

The else block is executed when the condition given in the while statement becomes false.

if the while loop is broken using break statement, then the else block will not be executed and the statement present after else block will be executed.

Consider the following example.



The image shows two windows from a Python 2.7.15 environment. The top window is a 'Python 2.7.15 Shell' with a menu bar (File, Edit, Shell, Debug, Options, Window, Help) and a list of numbers 3 through 10. Below the list, it shows a prompt '>>>' followed by a restart message: '===== RESTART: C:/Python27/while1.py ====='. Then, it prompts 'Enter the number?35' and shows the output '35 X 1 = 35'. The bottom window is an IDE titled 'whileelse.py - C:/Python27/whileelse.py (2.7.15)' with a menu bar (File, Edit, Format, Run, Options, Window, Help). The code in the IDE is as follows:

```
i=1;
while i<=5:
    print(i)
    i=i+1;
else:print("The while loop exhausted");
|
```

The status bar at the bottom right of the IDE window shows 'Ln: 6 Col: 0'.

2.5 loop manipulation using continue, pass, break, else

Python continue Statement

The continue statement in python is used to bring the program control to the beginning of the loop.

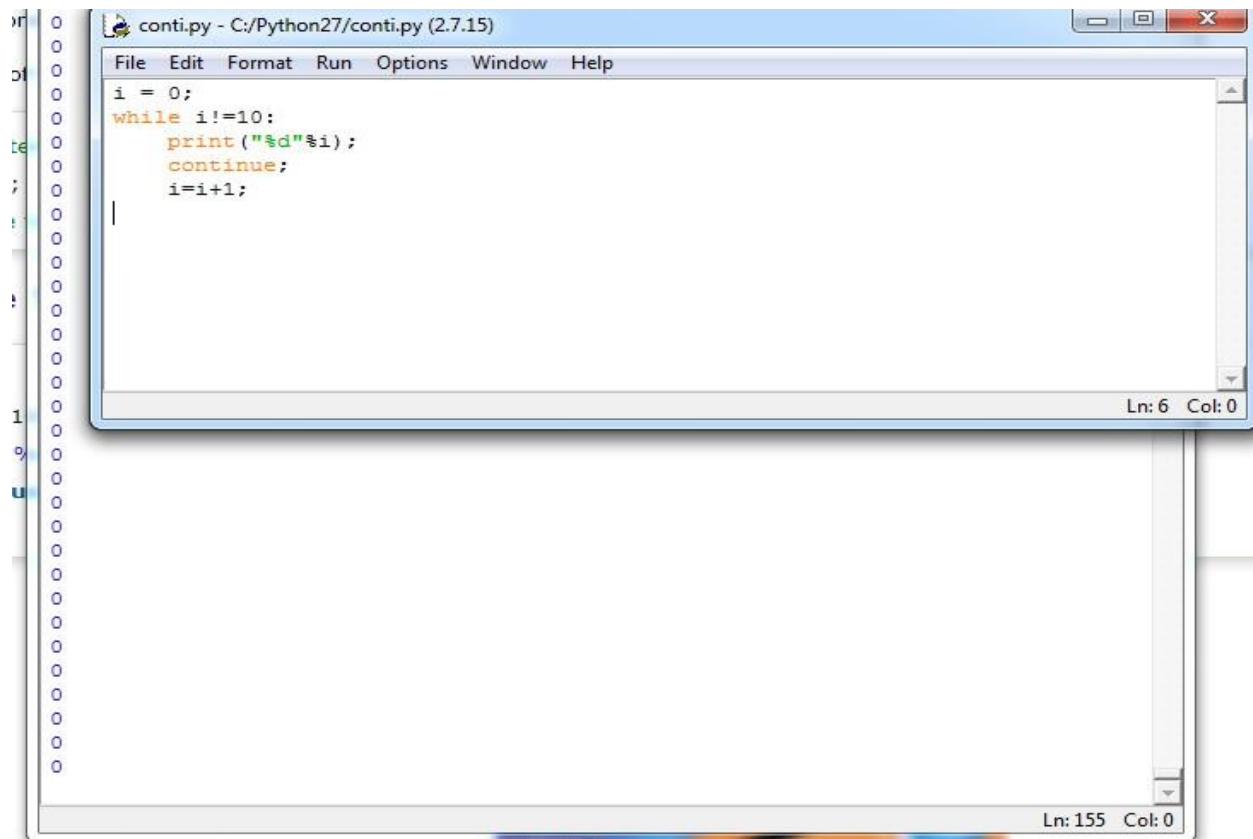
CO: DEVELOP PYTHON PROGRAM TO DEMONSTRATE USE OF OPERATORS

The continue statement skips the remaining lines of code inside the loop and start with the next iteration.

It is mainly used for a particular condition inside the loop so that we can skip some specific code for a particular condition.

The syntax of Python continue statement is given below.

1. **#loop statements**
2. **continue;**
3. **#the code to be skipped**



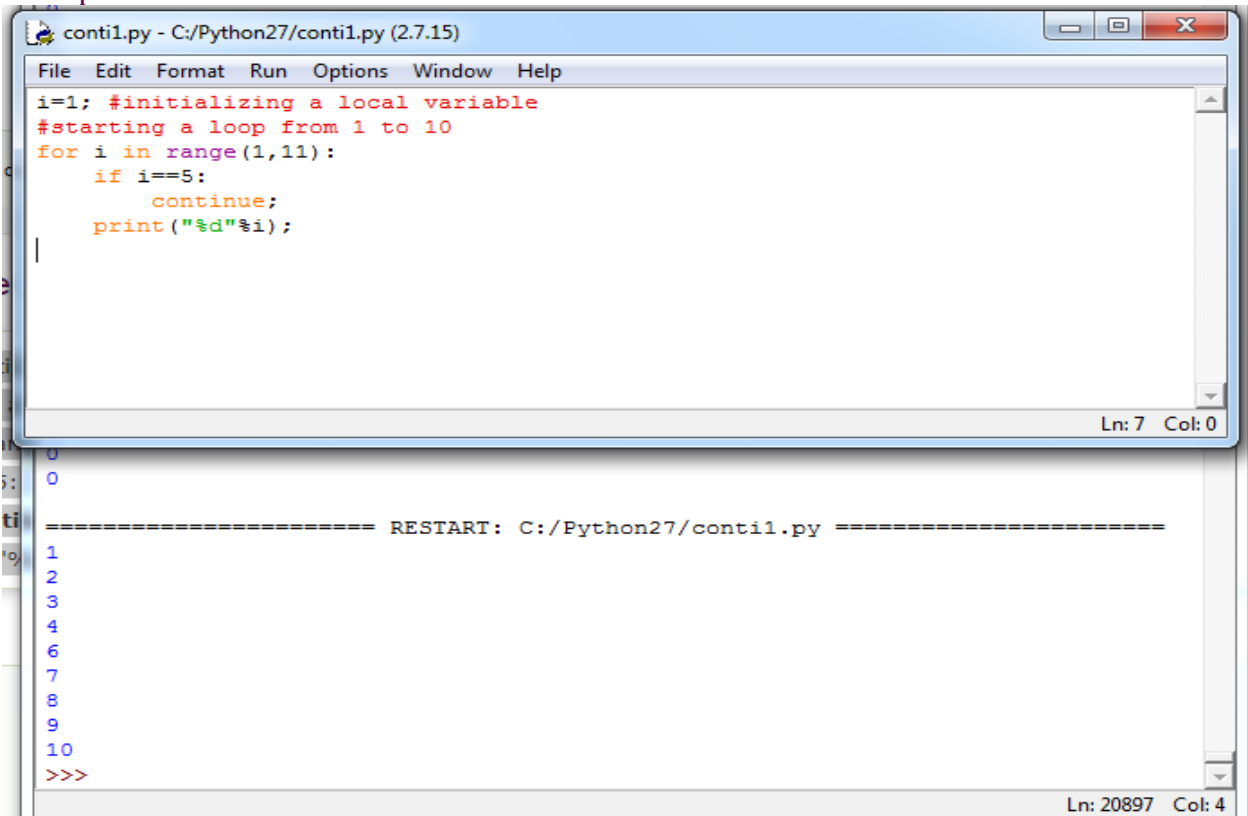
The screenshot shows a Python IDE window titled 'conti.py - C:/Python27/conti.py (2.7.15)'. The code in the editor is as follows:

```
i = 0;
while i!=10:
    print ("%d"%i);
    continue;
    i=i+1;
```

The status bar at the bottom right of the window indicates 'Ln: 6 Col: 0'.

CO: DEVELOP PYTHON PROGRAM TO DEMONSTRATE USE OF OPERATORS

Example 2



```
contil.py - C:/Python27/contil.py (2.7.15)
File Edit Format Run Options Window Help
i=1; #initializing a local variable
#starting a loop from 1 to 10
for i in range(1,11):
    if i==5:
        continue;
    print ("%d"%i);

Ln: 7 Col: 0

===== RESTART: C:/Python27/contil.py =====
1
2
3
4
6
7
8
9
10
>>>
```

Python Pass

In Python, pass keyword is used to execute nothing;

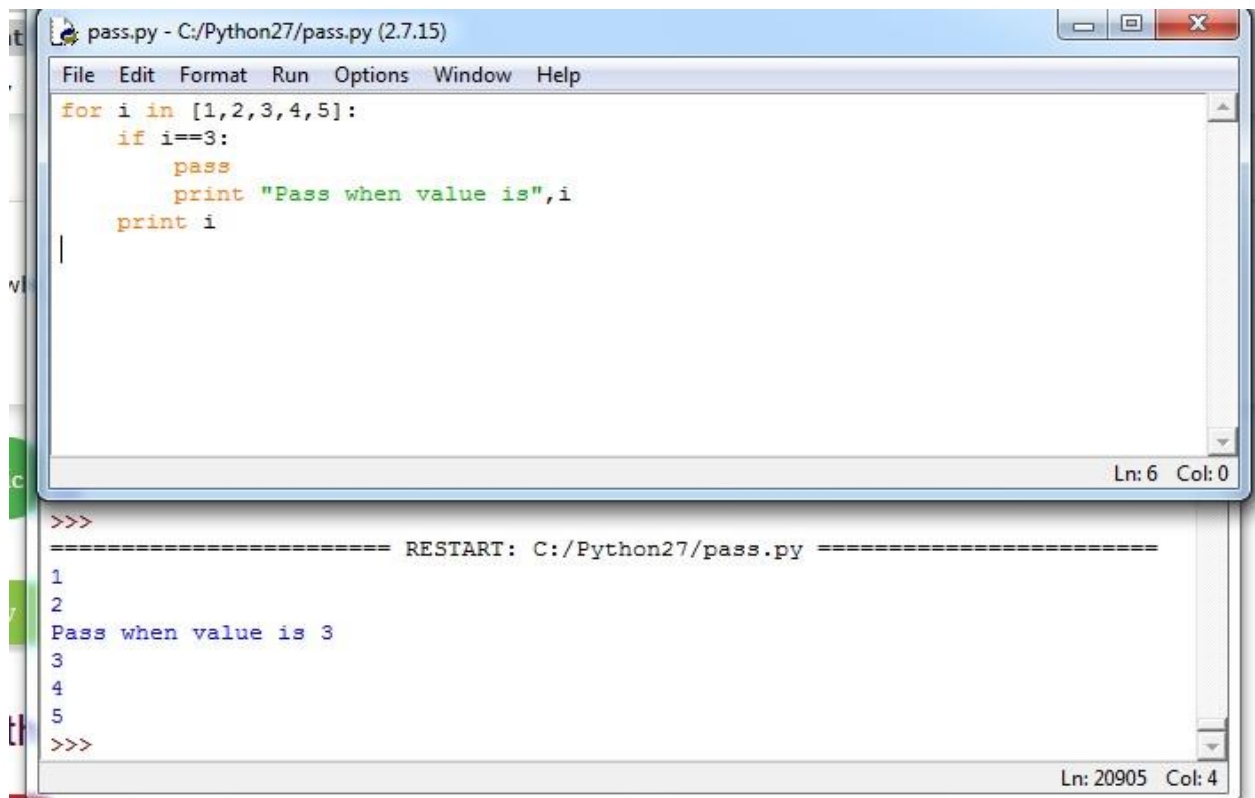
It means, when we don't want to execute code, the pass can be used to execute empty.

It is same as the name refers to.

Python Pass Syntax

pass

CO: DEVELOP PYTHON PROGRAM TO DEMONSTRATE USE OF OPERATORS



```
pass.py - C:/Python27/pass.py (2.7.15)
File Edit Format Run Options Window Help
for i in [1,2,3,4,5]:
    if i==3:
        pass
        print "Pass when value is",i
    print i

>>>
===== RESTART: C:/Python27/pass.py =====
1
2
Pass when value is 3
3
4
5
>>>
```

Python break statement

The break statement breaks the loops one by one, i.e., in the case of nested loops, it breaks the inner loop first and then proceeds to outer loops.

break is used to abort the current execution of the program and the control goes to the next line after the loop.

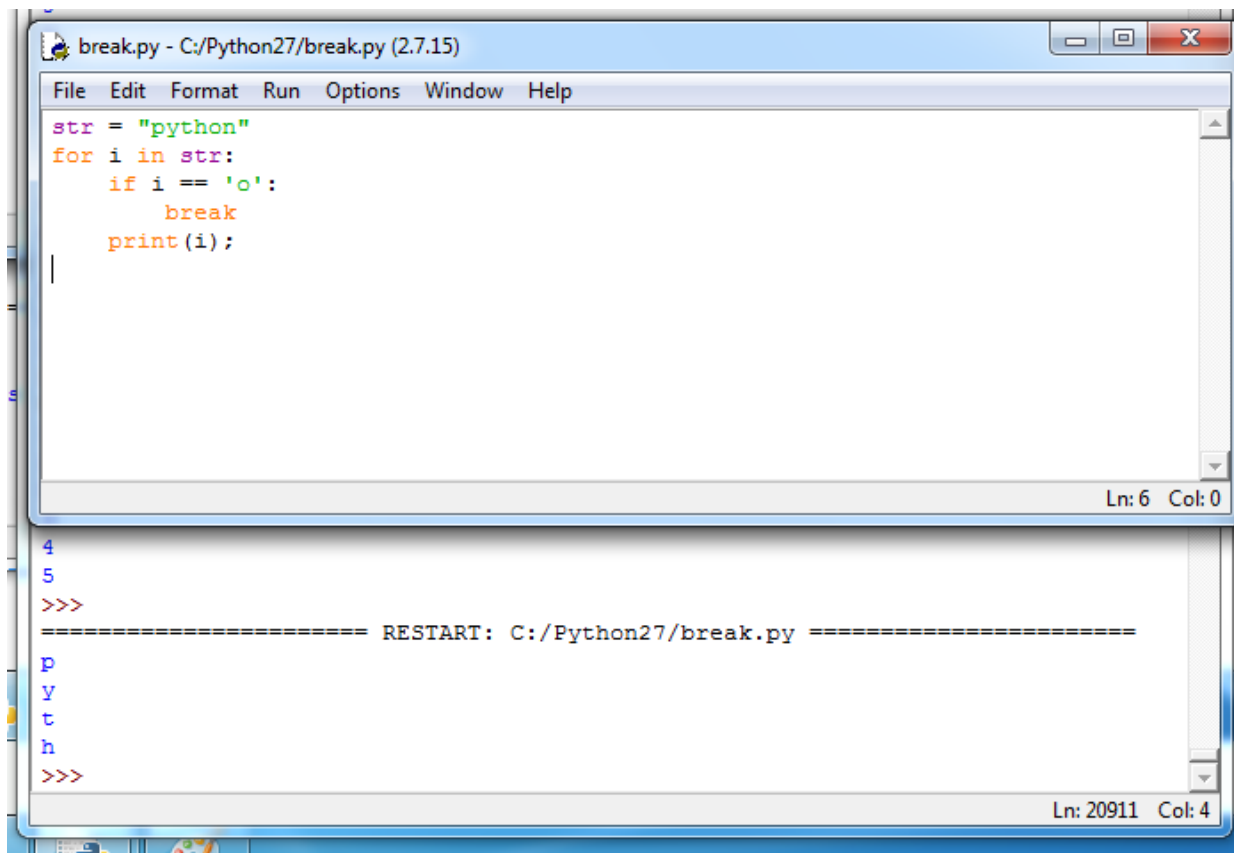
The break is commonly used in the cases where we need to break the loop for a given condition.

The syntax of the break is given below.

#loop statements

break;

CO: DEVELOP PYTHON PROGRAM TO DEMONSTRATE USE OF OPERATORS



The image shows a screenshot of a Python IDE window titled "break.py - C:/Python27/break.py (2.7.15)". The window has a menu bar with "File", "Edit", "Format", "Run", "Options", "Window", and "Help". The main text area contains the following Python code:

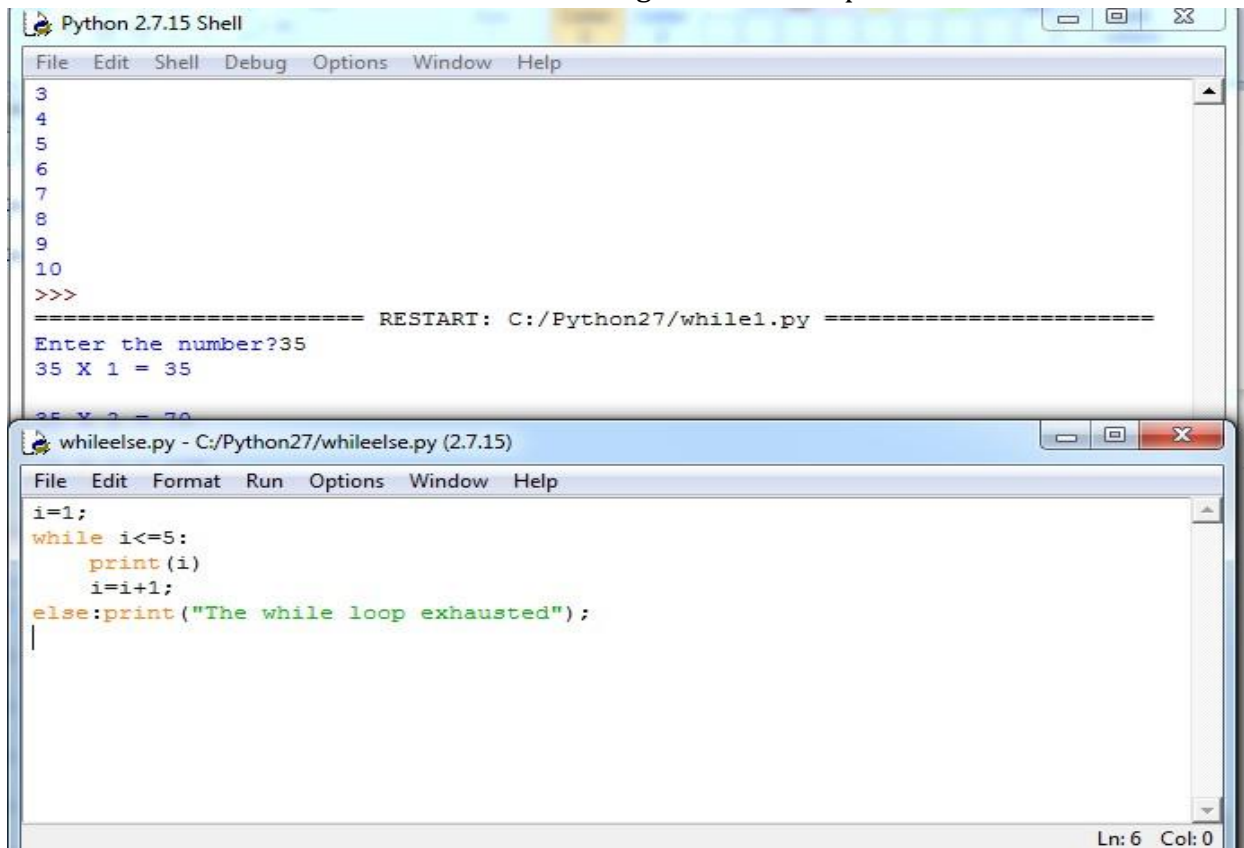
```
str = "python"
for i in str:
    if i == 'o':
        break
    print(i);
```

The status bar at the bottom right of the editor shows "Ln: 6 Col: 0". Below the editor is a console window showing the output of the program. It starts with a prompt "4" and "5", followed by ">>>". Then, a separator line appears: "===== RESTART: C:/Python27/break.py =====". The output shows the characters "p", "y", "t", and "h" on separate lines, followed by another ">>>" prompt. The console status bar at the bottom right shows "Ln: 20911 Col: 4".

CO: DEVELOP PYTHON PROGRAM TO DEMONSTRATE USE OF OPERATORS

Python else statement

The else block is executed when the condition given in the loop statement becomes false.



The image shows two overlapping windows from a Python 2.7.15 environment. The top window, titled 'Python 2.7.15 Shell', displays a list of numbers 3 through 10, followed by a prompt '>>>'. Below the prompt, a separator line indicates a restart of 'C:/Python27/while1.py'. The prompt then asks 'Enter the number?35', and the output shows '35 X 1 = 35'. The bottom window, titled 'whileelse.py - C:/Python27/whileelse.py (2.7.15)', shows the source code for a while loop with an else clause. The code is as follows:

```
i=1;
while i<=5:
    print(i)
    i=i+1;
else:print("The while loop exhausted");
```

The status bar at the bottom right of the code editor indicates 'Ln: 6 Col: 0'.

CO: DEVELOP PYTHON PROGRAM TO DEMONSTRATE USE OF OPERATORS