Python Conditional statement and loops

Thursday, 15 December 2022

“An investment in knowledge pays the best interest“

- *BENJAMIN FRANKLIN*

Today's Agenda

- Decision Control Statements
  - The if Statement
  - Concept of Indentation
  - The if-else Statement
  - The if-elif-else Statement

Using Format Specifiers With print()

Just like C language Python also allows us to use format specifiers with variables.

The format specifiers supported by Python are:

- %d: Used for int values
- %i: Used for int values
- %f: Used for float values
- %s: Used for string value

Using Format Specifiers With print()

Syntax:

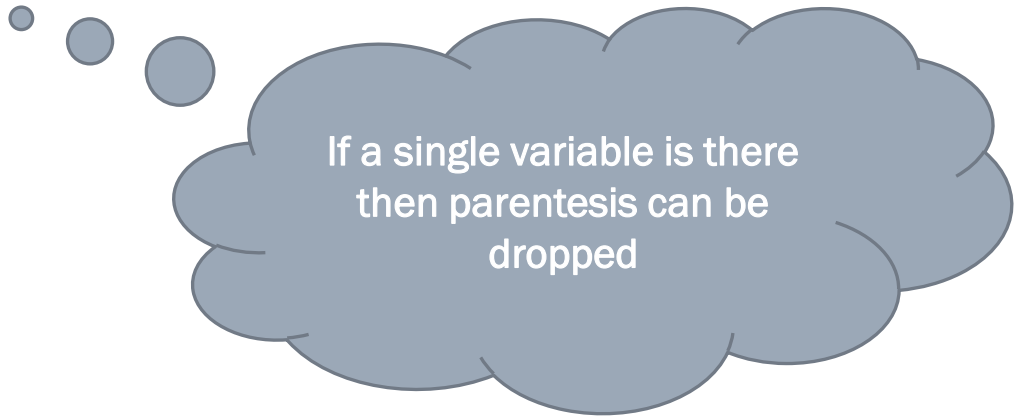
- print("format specifier" %(variable list))

Example:

```
a=10

print("value of a is %d " %(a))
```

Output:

```
value of a is 10
```



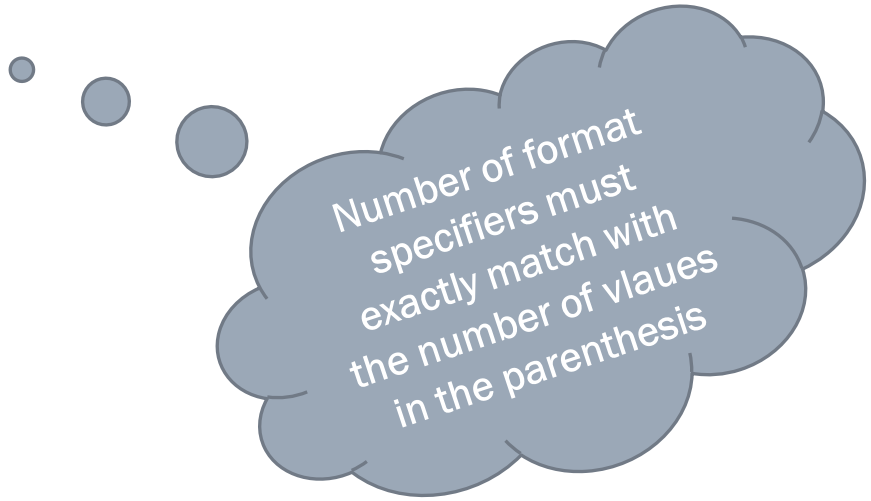
Using Format Specifiers With print()

Example:

```
a=10
msg="Welcome"
c=1.5
print("values are %d,%s,%f" %(a,msg,c))
```

Output:

Values are 10, Welcome, 1,500000



Key Points About Format Specifiers

The number of format specifiers and number of variables must always match

We should use the specified format specifier to display a particular value.

For example we cannot use %d for strings

However we can use %s with non string values also , like boolean

Examples

```
a=10

print("%s" %a)
```

Output:

10

```
a=10

print("%f" %a)
```

Output:

10.000000

```
a=10.6
print("%f" %a)
```

Output:

10.600000

```
a=10.6
print("%.2f" %a)
```

Output:

10.60

```
a=10.6
print("%d" %a)
```

Output:

10

```
a=10.6
print("%s" %a)
```

Output:

10.6

```
a=True
print("%s" %a)
```

Output:

True

```
a=True
print("%d" %a)
```

Output:

1

Examples

```
a=True

print("%f" %a)
```

Output:

1.000000

```
a="Bhopal"

print("%f" %a)
```

Output:

Type Error

```
a="Bhopal"
print("%s" %a)
```

Output:

Bhopal

```
a="Bhopal"
print("%d" %a)
```

Output:

TypeError: number required , not str

Using The Function format()

Python 3 introduced a new way to do string formatting by providing a method called format( ) in string object

This "new style" string formatting gets rid of the % operator and makes the syntax for string formatting more regular.

Using The Function format()

Syntax:

- print("string with { }".format(values))

Example

- name="Sunny"
- age=36
- print("My name is {0} and my age is {1}".format(name,age))

Output:

- My name is Sunny and my age is 36

Examples

```
name="Sunny"

age=25

print("My name is {1} and my age is{0}".format(age,name))
```

Output:

My name is Sunny and my age is 25

Using The Function format()

Example

- name="Sunny"
- age=25
- print("My name is {n} and my age is {a}".format(n=name,a=age))

Output:

- My name is Sunny and my age is 25

Decision Control Statements

Decision Control Statements are those statements which decide the execution flow of our program.

In other words , they allow us to decide whether a particular part of our program should run or not based upon certain condition.

The 4 decision control statements in Python are:

- if
- if....else
- if...elif...else
- nested if

The if Statement

The if the statement in Python is similar to other languages like in Java, C, C++, etc.

It is used to decide whether a certain statement or block of statements will be executed or not .

If a certain condition is true then a block of statement is executed otherwise not.

The if Statement

Syntax:

```
if (expression):
        statement1
        statement2
        .
        .
        statement..n
```

Some Important Points:

Python does not use { } to define the body of a code block , rather it uses indentation.

A code block starts with indentation and ends with the first unindented line.

The amount of indentation is up to the programmer, but he/she must be consistent throughout that block.

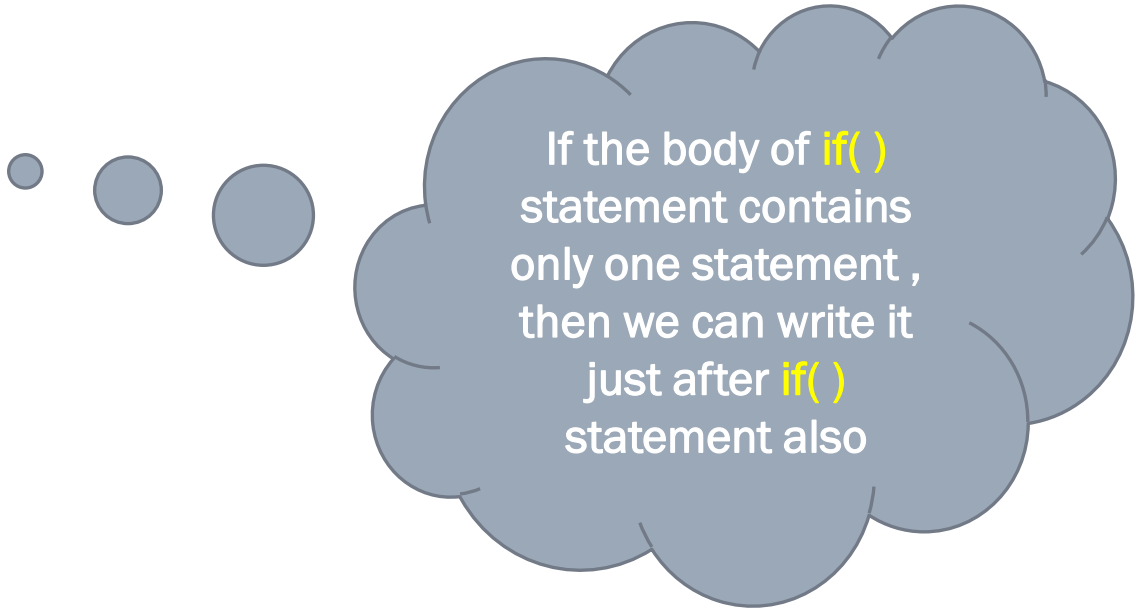
The colon after if( ) condition is important and is a part of the syntax. However parenthesis with condition is optional

Exercise

WAP to accept an integer from the user and check whether it is an even or odd

Solution 1:

```
a=int(input("Enter a number:"))
if(a%2==0):
  print("No is even")
if(a%2!=0):
  print("No is odd")
```



Solution

Solution 2:

```
if(a%2==0):print("No is even")

if(a%2!=0):print("No is odd")
```

What About Multiple Lines ?

If there are multiple lines in the body of **if( )** , then :

- Either we can write them inside if( ) by properly indenting them

OR

- If we write them just after if ( ) , then we must use semicolon as a separator

What About Multiple Lines ?

Solution 1:

```
if(a%2==0):
  print("No is even")
  print(“Hello”)
if(a%2!=0):
  print("No is odd")
  print(“Hi”)
```

Solution 2:

```
if(a%2==0): print("No is even");print(“Hello”)
if(a%2!=0):  print("No is odd");print(“Hi”)
```

The if –else Statement

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

If the condition is **False**, body of **else** is executed.

Indentation is used to separate the blocks.

The if-else Statement

Syntax:

```
if (expression):
        statement 1
        statement 2
else:
        statement 3
        statement 4
```

Indentation and colon are important for else also

Exercise

WAP to accept a character from the user and check whether it is a capital letter or small letter. Assume user will input only alphabets

Solution 1:

```
s=input("Enter a character:")
ch=s[0]
if  "A"<=ch<="Z":
  print("You entered a capital letter")
else:
  print("You entered a small letter")
```

We also can use the logical and operator and make the conditions separate

Solution

Solution 2:

```
s=input("Enter a character:")
ch=s[0]
if ch>="A" and ch<="Z":
  print("You entered a capital letter")
else:
  print("You entered a small letter")
```

The if –elif-else Statement

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**, body of **else** is executed.

The if –elif-else Statement

Syntax:

```
if (expression):
        statement 1
        statement 2
elif (expression):
        statement 3
        statement 4
else:
        statement 5
        statement 6
```

Although it is not visible in the syntax , but we can have multiple elif blocks with a single if block

Exercise

WAP to accept a character from the user and check whether it is a capital letter or small letter or a digit or some special symbol

```
s=input("Enter a character:")
ch=s[0]
if  "A" <=ch <="Z":
        print("You entered a capital letter")
elif  "a" <=ch <="z":
        print("You entered a small letter")
elif  "0" <=ch <="9":
        print("You entered a digit")
else:
        print("You entered some symbol")
```

The 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

The nested if Statement

Syntax:

```
if (expression):
        if (expression):
            statement 1
            statement 2
else:
        statement 3
        statement 4
statement 5
statement 6
```

Exercise

WAP to accept 3 integers from the user and without using any logical operator and cascading of relational operators , find out the greatest number amongst them

Exercise

WAP to accept an year from the user and check whether it is a leap year or not.

Hint:

An year is a leap year if:

It is exactly divisible by 4 and at the same time not

divisible by 100

OR

It is divisible by 400

For example:
2017 is not a leap year
2012 is a leap year
1900 is a not leap year
2000 is a leap year

Iterative Statements

There may be a situation when we need to execute a block of code several number of times.

For such situations , Python provides the concept of loop

A loop statement allows us to execute a statement or group of statements multiple times

Iterative Statements

The 2 popular loops provided by **Python** are:

- **The while Loop**
- **The for Loop**

Recall that **Python** doesn't provide any **do..while** loop like other languages

The while Loop

Syntax:

```
while condition:
        <indented statement 1>
        <indented statement 2>
        ...
        <indented statement n>
<non-indented statement 1>
<non-indented statement 2>
```

Some Important Points:

First the condition is evaluated. If the condition is **true** then statements in the **while** block is **executed.**

After executing statements in the **while** block the condition is checked again and if it is still **true**, then the statements inside the while block is **executed again**.

The statements inside the **while** block will keep executing until the condition is **true**.

Each execution of the loop body is known as **iteration**.

When the condition becomes **false** loop terminates and program control comes **out of the while loop** to begin the **execution** of statement following it.

Examples

- Example 1:

```
i=1
while i<=10:
    print(i)
    i=i+1
print("done!")
```

- Example 2:

```
i=1
total=0
while i<=10:
    print(i)
    total+=i
    i=i+1
print("sum is {0}".format(total))
```

Guess The Output

```
i=1
while i<=10:
        print(i)
i=i+1
print("done!")
```

```
i=1
while i<=10:
        print(i)
        total+=i
        i=i+1
print("sum is {0}".format(total))
```

Another Form Of "while" Loop

In Python , just like we have an else with if , similarly we also can have an else part with the while loop.

The statements in the else part are executed, when the condition is not fulfilled anymore.

Another Form Of "while" Loop

Syntax:

```
while condition:
        <indented statement 1>
        <indented statement 2>
        ...
        <indented statement n>
else:
        <indented statement 1>
        <indented statement 2>
```

Some Important Points:

Many programmer's have a doubt that If the statements of the additional else part were placed right after the while loop without an else, they would have been executed anyway, wouldn't they.

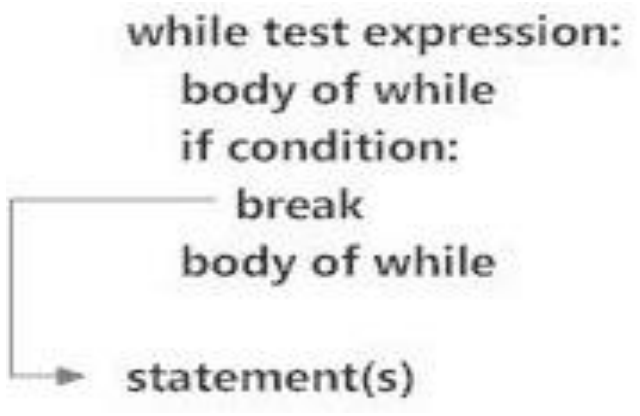
Then what is the use of else

To understand this , we need to understand the break statement,

The "break" Statement

Normally a **while** loop ends only when the **test condition** in the loop becomes **false**.

However , with the help of a **break** statement a **while** loop can be left prematurely,



Now comes the crucial point:

If a loop is left by break, the else part is not executed.

Example

- Example 1:

```
i=1
while i<=10:
        if(i==5):
            break
        print(i)
        i=i+1
else:
        print("bye")
```

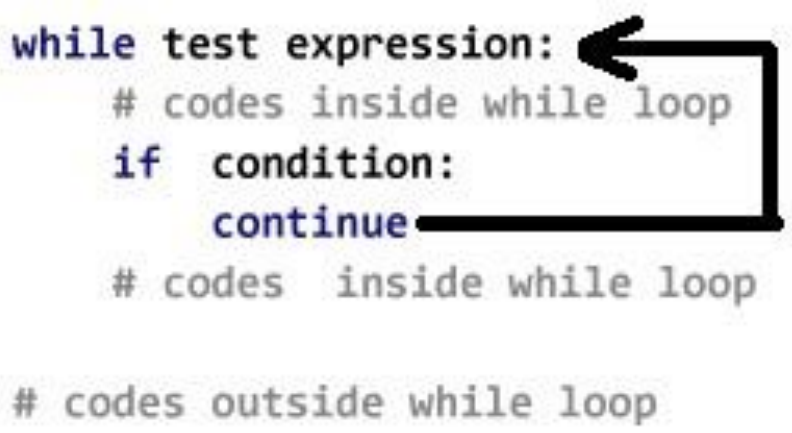
- Example 2:

```
i=1
while i<=10:
        print(i)
        i=i+1
else:
        print("bye")
```

The "continue" Statement

The continue statement in Python returns the control to the beginning of the while loop.

It rejects all the remaining statements in the current iteration of the loop and moves the control back to the top of the loop.



Example

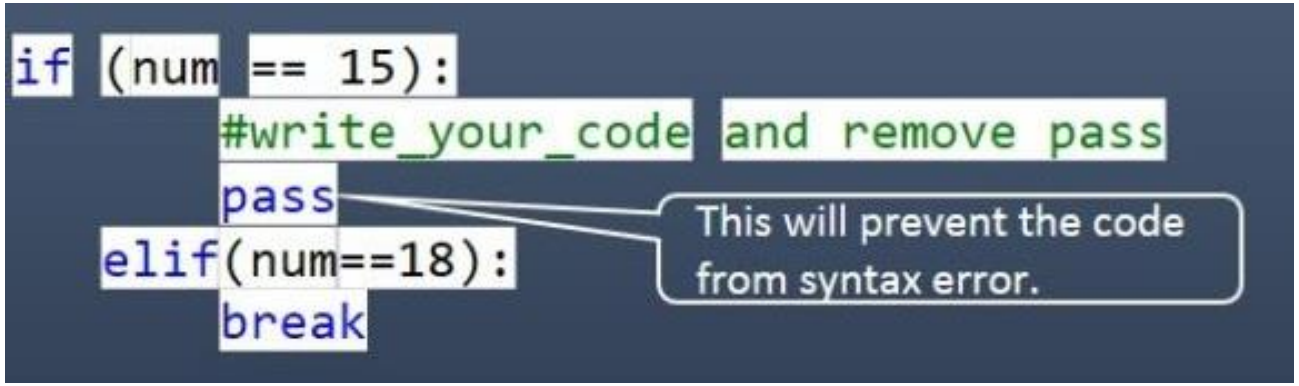
```
i=0
while i<10:
    i=i+1
    if(i%2!=0):
     continue
    print(i)
```

The “pass” Statement

In Python, the pass statement is a no operation statement.

That is , nothing happens when pass statement is executed.

Example:



Example

```
i=0
while i<10:
 i=i+1
 if(i%2!=0):
     pass
 else:
     print(i)
```

The for Loop

Like the while loop the for loop also is a programming language statement, i.e. an iteration statement, which allows a code block to be executed multiple number of times.

There are hardly programming languages without for loops, but the for loop exists in many different flavours, i.e. both the syntax and the behaviour differs from language to language

The for Loop

Different Flavors Of “for” Loop:

Count-controlled for loop (Three-expression for loop):

- This is by far the most common type. This statement is the one used by C , C++ and Java . Generally it has the form:
  for (i=1; i <= 10; i++)
  This kind of for loop is not implemented in Python!

Numeric Ranges

- This kind of for loop is a simplification of the previous kind. Starting with a start value and counting up to an end value, like
- for i = 1 to 100
  Python doesn't use this either.

The for Loop

Iterator-based for loop

- Finally, we come to the one used by Python. This kind of a for loop iterates over a collection of items. In each iteration step a loop variable is set to a value in a sequence or other data collection.

- This kind of for loop is known in most Unix and Linux shells and it is the one which is implemented in Python.

Syntax Of for Loop In Python

iNeuron

Syntax:

```
for some_var in some_collection:
        # loop body
        <indented statement 1>
        <indented statement 2>
        ...
        <indented statement n>
<non-indented statement 1>
        <non-indented statement 2>
```

Examples

Example 1:

```
word="Sunny“
for ch in word:
    print(ch)
```

Example 2:

```
fruits=["Apple","Banana","Guava","Orange"]
for fruit in fruits:
    print(fruit)
```

Exercise

Write a program using for loop to accept a string from the user and display it vertically but don't display the vowels in it.

Ans.

```
word="sunny savita"
if(ch in ["a","e","i","o","u"]):
        continue
print(ch,end=" ")
```

The range Function

The range() function is an in-built function in Python, and it returns a range object.

This function is very useful to generate a sequence of numbers in the form of a List.

The range( ) function takes 1 to 3 arguments

The range Function With One Parameter

Syntax:

- range(n)

For an argument n, the function returns a range object containing integer values from 0 to n-1.

Example:

```
a=range(10)
print(a)
```

As we can see that when we display the variable a , we get to see the description of the range object and not the values.

To see the values , we must convert range object to list

The range Function With One Parameter

Example:

```
a=range(10)

b=list(a)

print(b)
```

The function list( ) accepts a range object and converts it into a list of values .
These values are the numbers from 0 to n-1 where n is the argument passed to the function range()

What If We Pass Negative Number ?

Guess:

```
a=range(-10)

b=list(a)

print(b)
```

The output is an empty list denoted by [ ] and it tells us that the function range( ) is coded in such a way that it always moves towards right side of the start value which here is 0.

But since -10 doesn't come towards right of 0, so the output is an empty list

The range Function With Two Parameter

Syntax:

- range(m,n)

For an argument m,n , the function returns a range object containing integer values from m to n-1.

Example:

```
a=range(1,10)

print(a)
```

Here again when we display the variable a , we get to see the description of the range object and not the values. So we must use the function list( ) to get the values

The range Function With Two Parameter

Example:

```
a=range(1,10)

b=list(a)

print(b)
```

The output is list of numbers from 1 to 9 because 10 falls towards right of 1

What If We Pass First Number Greater?

Guess:

```
a=range(10,1)

b=list(a)

print(b)
```

The output is an empty list because
as mentioned earlier it traverses towards right
of start value and 1 doesn't come
to the right of 10

Passing Negative Values

We can pass **negative start** or/and **negative stop value** to **range( )** when we call it with **2 arguments** .

Example:

```
a=range(-10,3)

b=list(a)

print(b)
```

Since 3 falls on right of -10 ,
so we are getting range of numbers from
-10 to 3

Guess The Output

```
a=range(-10,-3)

b=list(a)

print(b)
```

```
a=range(-3,-10)

b=list(a)

print(b)
```

```
a=range(-3,-3)

b=list(a)

print(b)
```

The range Function With Three Parameter

Syntax:

- range(m,n,s)

Finally, the range() function can also take the third parameter .This is for the step value.

Example:

```
a=range(1,10,2)

b=list(a)

print(b)
```

Since step value is 2 , so we got nos from 1 to 9 with a difference of 2

Guess The Output

```
a=range(7,1,-2)

b=list(a)

print(b)
```

Pay close attention , that we are having start value greater than end value , but since step value is negative , so it is allowed

```
a=range(5,10,20)

b=list(a)

print(b)
```

Here, note that the first integer, 5, is always returned, even though the interval 20 sends it beyond 10

Guess The Output

```
a=range(2,14,1.5)

b=list(a)

print(b)
```

Note that all three arguments must be integers only.

```
a=range(5,10,0)

b=list(a)

print(b)
```

It raised a Value Error because the interval cannot be zero if we need to go from one number to another.

Guess The Output

```
a=range(2,12)

b=list(a)

print(b)
```

The default value of step is 1 , so the output is from 2 to 11

```
a=range(12,2)

b=list(a)

print(b)
```

As usual , since the start value is greater than end value so we get an empty list

Using range( ) With for Loop

We can use range() and for together for iterating through a list of numeric values

Syntax:

- for <var_name> in range(end)
  indented statement 1
  indented statement 2
  .
  .
  indented statement n

Example

Code:

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

Using 2 Parameter range( ) With for Loop

We can use 2 argument range() with for also for iterating through a list of numeric values between a given range

Syntax:

- for <var_name> in range(start,end)
  indented statement 1
  indented statement 2
  .
  .
  indented statement n

Example

Code:

```
for i in range(1,11):
  print(i)
```

Exercise

Write a program to accept an integer from the user and display the sum of all the numbers from 1 to that number.

```
num=int(input("Enter an int:"))

total=0

for i in range(1,num+1):

 total=total+i

print("sum of no's from 1 to {} is {}".format(num,total))
```

Exercise

Write a program to accept an integer from the user and calculate it's factorial

Using 3 Parameter range( ) With for Loop

Syntax:

- for <var_name> in range(start,end,step)

  indented statement 1

  indented statement 2

  .

  .

  indented statement n

Example

Code:

```
for i in range(1,11,2):
    print(i)
```

Example

Code:

```
for i in range(100,0,-10):
    print(i)
```

Using for With else

Just like **while** , the **for** loop can also have an **else** part , which executes if no **break** statements executes in the f**or loop**

Syntax:

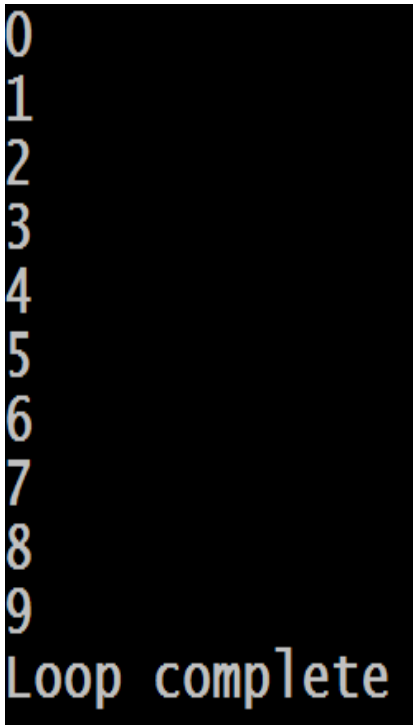
```
for <var_name> in some_seq:
        indented statement 1
        if test_cond:
            break
else:
        indented statement 3
        indented statement 4
```

Example

Code:

```
for i in range(10):
    print(i)
else:
    print("Loop complete")
```

Output:



Example

Code:

```
for i in range(1,10):
    print(i)
    if i%5==0:
            break
else:
    print("Loop complete")
```

Output:

Using Nested Loop

Loops can be nested in **Python**, as they can with other programming languages.

A **nested loop** is a loop that occurs within another loop, and are constructed like so:

Syntax:

```
for <var_name> in some_seq:
    for <var_name> in some_seq:
                        indented statement 1
                        indented statement 2
                        .
                        .
      indented statement n
  unindented statement
  unindented statement
```

Example

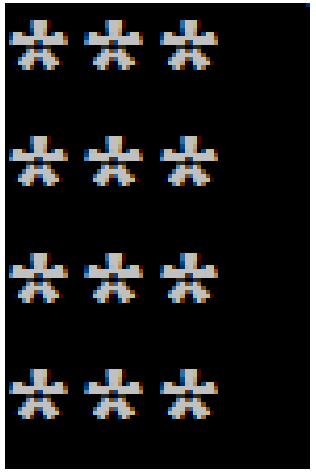
Code:

```
numbers = [1, 2, 3]

alpha = ['a', 'b', 'c']

for n in numbers:

 print(n)

 for ch in alpha:

            print(ch)
```

Exercise

Write a program to print the following pattern

Sample Output:



Solution

Code:

```
for i in range(1,5):
 for j in range(1,4):
            print("*",end="")
 print()
```

Solution

Can you write the same code using only

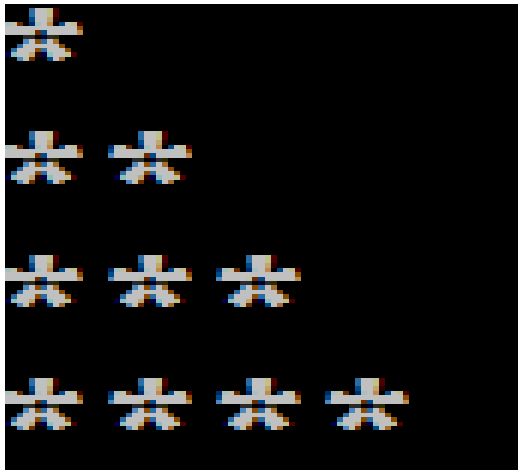
single loop ?

```
for i in range(1,5):
    print("*"*3)
```

Exercise

Write a program to print the following pattern

Sample Output:



Solution

Code:

```
for i in range(1,5):
 for j in range(1,i+1):
            print("*",end="")
 print()
```

Exercise

Write a program to print the following pattern

Sample Output:

Solution

Code:

```
for i in range(4,0,-1):
 for j in range(1,i+1):
            print("*",end="")
 print()
```

Exercise

Write a program to accept an integer from the user and display all the numbers from 1 to that number. Repeat the process until the user enters 0.

Sample Output:

```
Enter a number: 3
1
2
3
Enter a number: 9
1
2
3
4
5
6
7
8
9
Enter a number: 0
```

Solution

Code:

```
x = int(input('Enter a number: '))
while x != 0:
    for y in range (1, x+1):
        print (y)
        y+=1
    x = int(input('Enter a number: '))
```

iNeuron

Thank you