

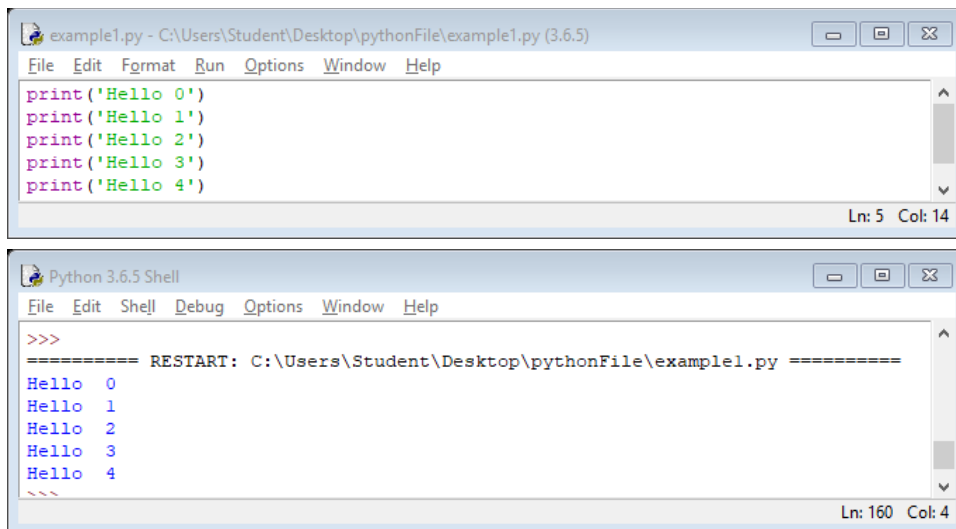
Going Loopy

Nothing is worse than having to do the same thing over and over again. Programmers don't particularly like repeating codes. Therefore, programming languages have what is called *loops*, which allow programmers to set the same code to repeat as many as needed, even to set a loop to run infinite time. The most common loops are *for* and *while* loops.

for loops

A **for** loop is used for iterating over a sequence. With the **for** loop we can execute a set of statements, once for each item in a list, tuple, set etc.

For example, to print hello five times, we could like the following lines of code:



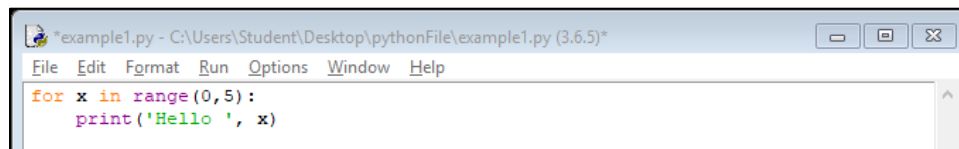
The screenshot shows two windows from a Python IDE. The top window, titled 'example1.py - C:\Users\Student\Desktop\pythonFile\example1.py (3.6.5)', contains the following code:

```
print('Hello 0')
print('Hello 1')
print('Hello 2')
print('Hello 3')
print('Hello 4')
```

The bottom window, titled 'Python 3.6.5 Shell', shows the output of running the code:

```
>>>
===== RESTART: C:\Users\Student\Desktop\pythonFile\example1.py =====
Hello 0
Hello 1
Hello 2
Hello 3
Hello 4
>>>
```

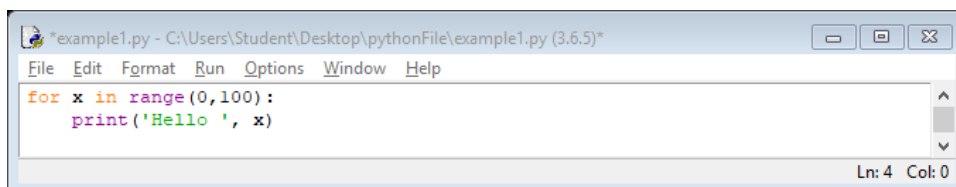
Now, think in what should we do if we need to print the same hello line 100 times? Should we write 100 code of lines? Instead, we can use a *for* loop to reduce the amount of typing and repetition:



The screenshot shows a file editor window titled 'example1.py - C:\Users\Student\Desktop\pythonFile\example1.py (3.6.5)*' with the following code:

```
for x in range(0,5):
    print('Hello ', x)
```

For instant, to repeat the same hello line 100 times, we should only change the range:



The screenshot shows a file editor window titled 'example1.py - C:\Users\Student\Desktop\pythonFile\example1.py (3.6.5)*' with the following code:

```
for x in range(0,100):
    print('Hello ', x)
```

The *for* loop syntax is:

for loop declaration

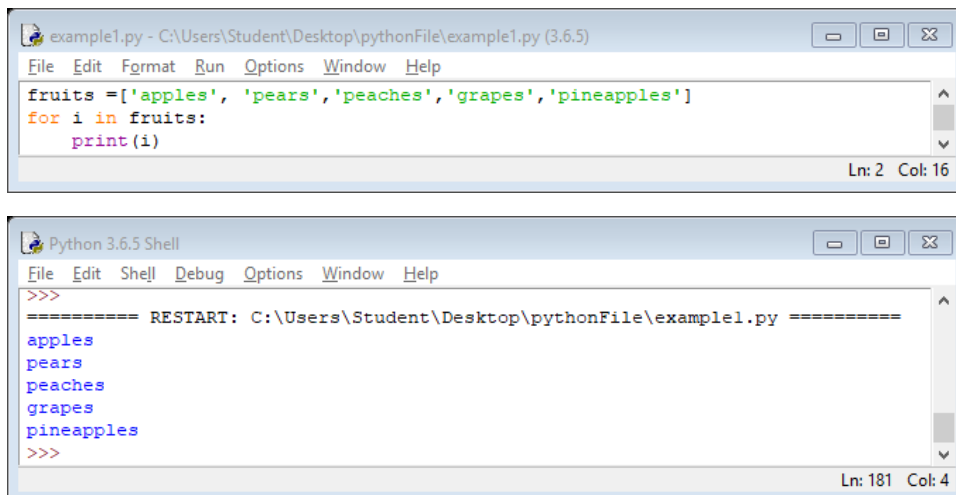
range set the initial value for *x* to be 0 and increasing repeat the loop block code up to 5 (exclusive), including

```
for x in range(0,5):  
    print('Hello ', x)
```

x is the iteration variable or counter variable

loop block

We can also use the *for* loop to print lists. For example, we can create a list of *fruits* and use *for* loop to print each of the value inside the list:



The screenshot shows a Python IDE window titled 'example1.py - C:\Users\Student\Desktop\pythonFile\example1.py (3.6.5)'. The code in the editor is:

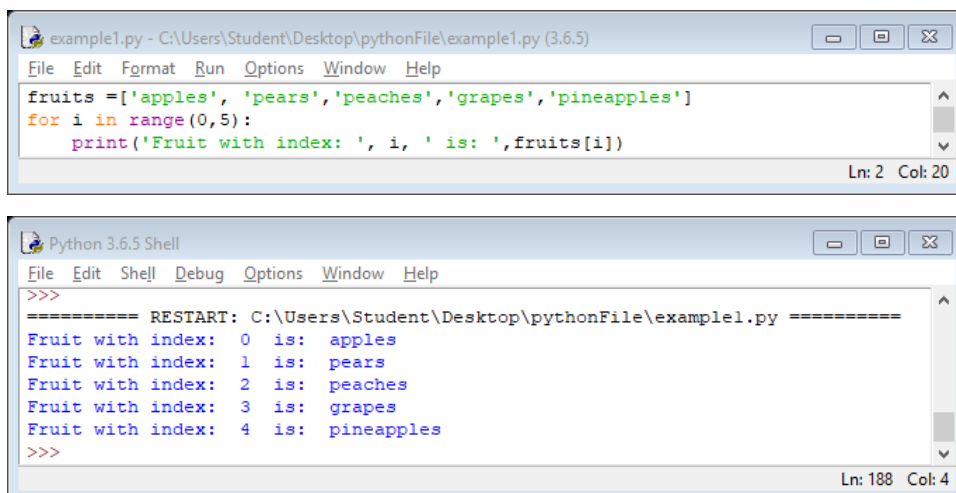
```
fruits = ['apples', 'pears', 'peaches', 'grapes', 'pineapples']  
for i in fruits:  
    print(i)
```

The status bar at the bottom right indicates 'Ln: 2 Col: 16'. Below the IDE window is a 'Python 3.6.5 Shell' window showing the output of the code:

```
>>>  
===== RESTART: C:\Users\Student\Desktop\pythonFile\example1.py =====  
apples  
pears  
peaches  
grapes  
pineapples  
>>>
```

The status bar at the bottom right of the shell indicates 'Ln: 181 Col: 4'.

We can also print the list as the following:



The screenshot shows a Python IDE window titled 'example1.py - C:\Users\Student\Desktop\pythonFile\example1.py (3.6.5)'. The code in the editor is:

```
fruits = ['apples', 'pears', 'peaches', 'grapes', 'pineapples']  
for i in range(0,5):  
    print('Fruit with index: ', i, ' is: ', fruits[i])
```

The status bar at the bottom right indicates 'Ln: 2 Col: 20'. Below the IDE window is a 'Python 3.6.5 Shell' window showing the output of the code:

```
>>>  
===== RESTART: C:\Users\Student\Desktop\pythonFile\example1.py =====  
Fruit with index: 0 is: apples  
Fruit with index: 1 is: pears  
Fruit with index: 2 is: peaches  
Fruit with index: 3 is: grapes  
Fruit with index: 4 is: pineapples  
>>>
```

The status bar at the bottom right of the shell indicates 'Ln: 188 Col: 4'.

The range() function with three argument

The **range()** function defaults to increment the sequence by 1, however it is possible to specify the increment value by adding a third parameter:

Initial value Increment

for x in range(2, 30, 3):

 ↑
 End value (not included)
 exclusive value

Example) Use a for loop to print numbers between 2 and 30 with an increment of 3

```
print("\n-- numbers 2 and 30 with an increment of 3--\n")
for num in range(2, 30, 3):
    print(num)
```

```
-- numbers 2 and 30 with an increment of 3--
2
5
8
11
14
17
20
23
26
29
```

Example) Use a for loop to print numbers between 0 and 10 with a decrement of 2

```
print("\n-- numbers between 0 and 10 with a decrement of 2--\n")
for num in range(10, 0, -2):
    print(num)
```

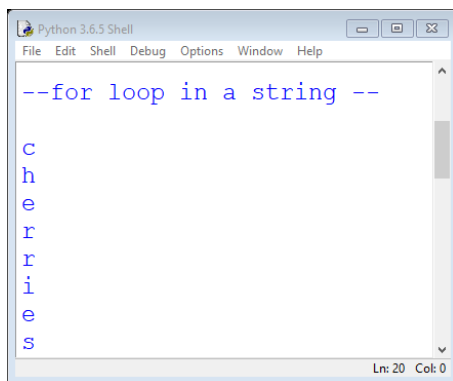
```
-- numbers between 0 and 10 with a decrement of 2--
10
8
6
4
2
```

Looping through a string

Since string is a list of characters, we can use for loop to print of a sequence of characters of a given string.

Example) Use for loop to print each character of **cherries**

```
print("\n--for loop in a string --\n")  
  
for x in "cherries":  
    print(x)
```



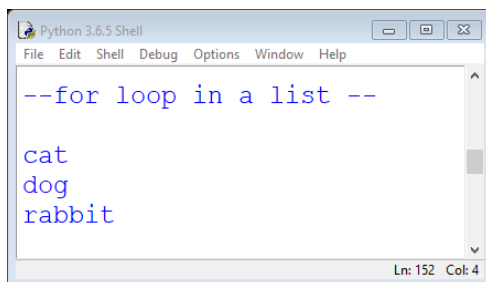
The screenshot shows a Python 3.6.5 Shell window with a menu bar (File, Edit, Shell, Debug, Options, Window, Help). The output of the code is displayed in the shell area: a blank line followed by "--for loop in a string --", then each character of the string "cherries" on a new line. The status bar at the bottom indicates "Ln: 20 Col: 0".

Looping through a list

You can loop through the list items by using a for loop from the beginning to the end of the list without using **range()** function.

Example) Print each item in list **animals**

```
print("\n--for loop in a list --\n")  
  
animals=["cat","dog","rabbit"]  
  
for counter in animals:  
    print(counter)
```



The screenshot shows a Python 3.6.5 Shell window with a menu bar (File, Edit, Shell, Debug, Options, Window, Help). The output of the code is displayed in the shell area: a blank line followed by "--for loop in a list --", then each item of the list "cat", "dog", and "rabbit" on a new line. The status bar at the bottom indicates "Ln: 152 Col: 4".

The break and continue statement

The other way to stop a loop is using a **break**. A *break* will terminate a loop if the condition meets the break even if the while condition is true.

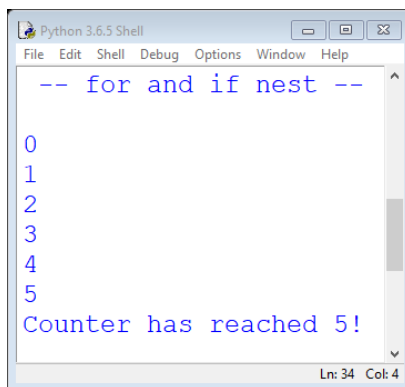
The **continue** statement we can stop the current iteration, and continue with the next iteration

Nesting for loop and if statement

We can also nest an **if** statement inside of a **for** loop.

Example) Use a **for** loop to print numbers from 0 to 10 exclusive and break the **for** loop when the counter reaches to 5

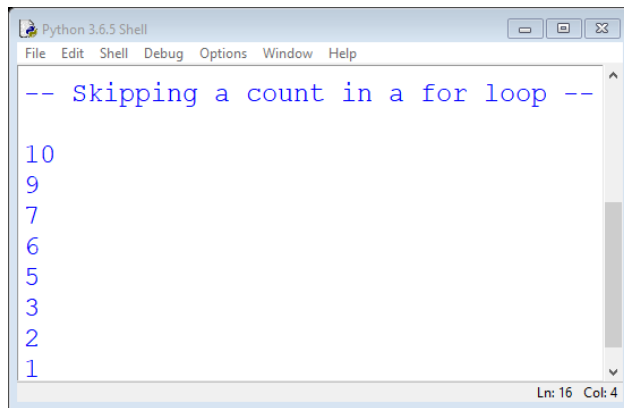
```
print("\n -- for and if nest --\n")
for counter in range(11):
    print(counter)
    if counter == 5:
        print("Counter has reached 5!")
        break
```



break in the **if** statement ends the program when the **if** condition is TRUE.

Example) Use a **for** loop to print numbers from 10 to 0 and skip numbers that are multiple of 4.

```
print("\n-- Skipping a count in a for loop --\n")
for counter in range(10,0,-1):
    if counter%4==0:
        continue
    print(counter)
```



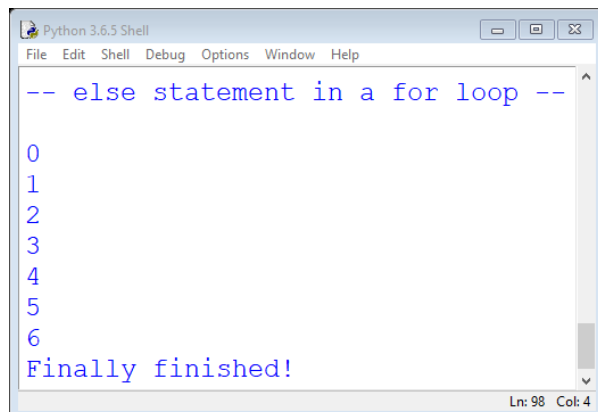
```
Python 3.6.5 Shell
File Edit Shell Debug Options Window Help
-- Skipping a count in a for loop --
10
9
7
6
5
3
2
1
Ln: 16 Col: 4
```

else statement in a for loop

The **else** keyword in a **for** loop specifies a block of code to be executed when the loop is finished.

Example) Print all numbers from 0 to 6, and print a message **Finally finished!** when the loop has ended

```
print("\n-- else statement in a for loop --\n")
for x in range(7):
    print(x)
else:
    print("Finally finished!")
```



```
Python 3.6.5 Shell
File Edit Shell Debug Options Window Help
-- else statement in a for loop --
0
1
2
3
4
5
6
Finally finished!
Ln: 98 Col: 4
```