UNIVERSITY LIBRARIES
UNIVERSITY AT ALBANY
State University of New York

Python Basics II: Python Statements

A **statement** is an instruction that a Python interpreter can execute. A simple statement is comprised within a single logical line, while a compound statement, containing (groups of) other statements, generally spans multiple lines. In the following tutorial, you will learn about several Python compound statements, such as **if** statements, **for** statements, and **while** statements. You can find a list of more Python statements at https://docs.python.org/3/reference/index.html.

If statements

The if statement is used for conditional execution. When you want to execute a code only if a certain condition is satisfied, decision making is required. The if statement is used in Python for decision making.

The syntax of the if, if…else, and elif statements is as follows:

```
if expression:

    statement (s)

elif:

    statement (s)

else:

    statement (s)
```

The **if** keyword is used to create conditional statements and allows you to execute a block of code only if a condition is True.

The **elif** keyword is short for else if. There can be as many elif conditions as necessary between the if condition and the else conclusion.

The **else** keyword decides what to do if the condition is False.

Python supports the usual logical conditions from mathematics:

Meaning	**Math Symbol**	**Python Symbols**
Less than	<	<
Greater than	>	>
Less than or equal to	≤	<=
Greater than or equal to	≥	>=
Equals	=	==
Not equals	≠	!=

Let’s try the following example:

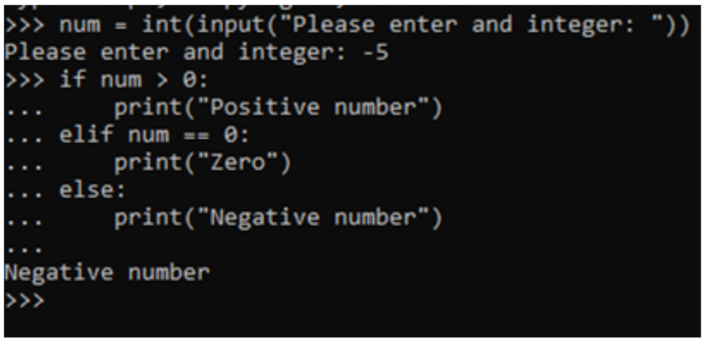
```
# check if the number is positive or negative or zero

num = int(input("Please enter an integer: "))
```

Enter an integer when the text is printed.

```
if num > 0:

    print("Positive number")

elif num == 0:

    print("Zero")

else:

    print("Negative number")
```

Note: Make sure the print statements are indented.

Press enter twice to see the output.



```
>>> num = int(input("Please enter and integer: "))
Please enter and integer: -5
>>> if num > 0:
...     print("Positive number")
... elif num == 0:
...     print("Zero")
... else:
...     print("Negative number")
...
Negative number
>>>
```

For Statements

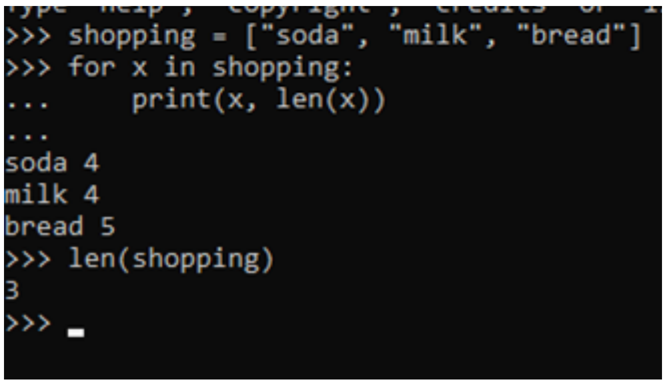
The for statement is used for iterating over a sequence. Let’s say you are going grocery shopping and checking the shopping list. Lists can be created using square brackets [].

```
shopping = [“soda”, “milk”, “bread”]

for x in shopping:

    print(x, len(x))
```

The len() function returns the number of items in an object. When the object is a string, the len() function returns the number of characters in the string.

```
len(shopping)
```

Compare len(x) in shopping and len(shopping) in the output:

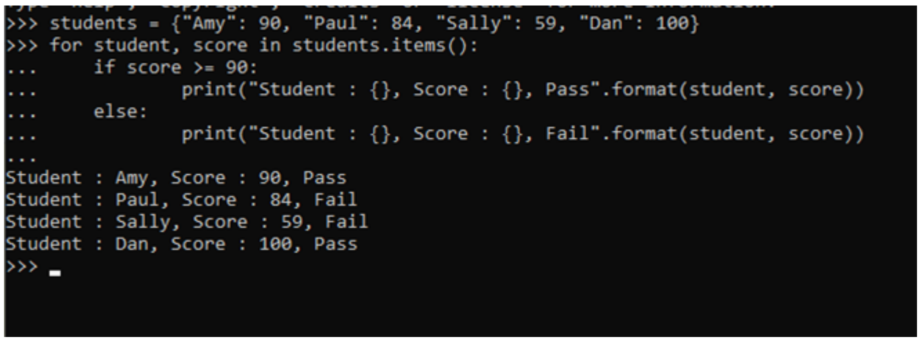


```
>>> shopping = ["soda", "milk", "bread"]
>>> for x in shopping:
...     print(x, len(x))
...
soda 4
milk 4
bread 5
>>> len(shopping)
3
>>>
```

Now you can make a script using both the if and for statements as bellow:

```
students = {“Amy”: 90, “Paul”: 84, “Sally”: 59, “Dan”: 100}

for student, score in students.items():

    if score >= 90:

        print(“Student : {}, Score : {}, Pass”.format(student, score))

    else:

        print(“Student : {}, Score : {}, Fail”.format(student, score))
```

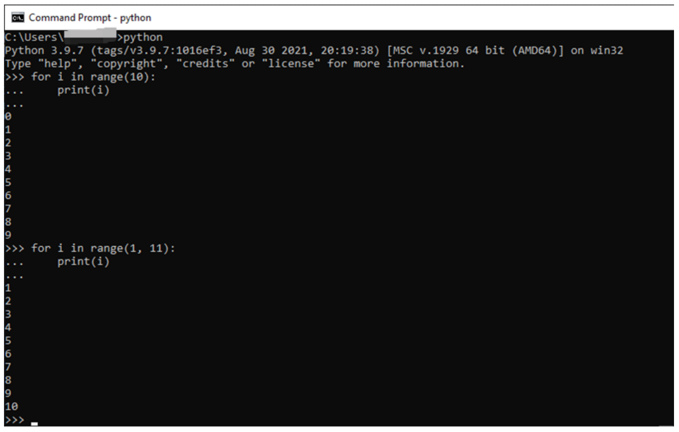
The curly braces {} are used in Python to define a dictionary. Dictionaries are used to store data values in key:value pairs. The items() method is used to return the list with all dictionary keys with values.



```
>>> students = {"Amy": 90, "Paul": 84, "Sally": 59, "Dan": 100}
>>> for student, score in students.items():
...     if score >= 90:
...             print("Student : {}, Score : {}, Pass".format(student, score))
...     else:
...             print("Student : {}, Score : {}, Fail".format(student, score))
...
Student : Amy, Score : 90, Pass
Student : Paul, Score : 84, Fail
Student : Sally, Score : 59, Fail
Student : Dan, Score : 100, Pass
>>>
```

Another useful function is range(). Python range() function returns the sequence of the given number between the given range. Try the following:

```
for i in range(10):

        print(i)

for i in range(1, 11):

        print(i)
```



Command Prompt - python

```
C:\Users\[redacted]>python
Python 3.9.7 (tags/v3.9.7:1016ef3, Aug 30 2021, 20:19:38) [MSC v.1929 64 bit (AMD64)] on win32
Type "help", "copyright", "credits" or "license" for more information.
>>> for i in range(10):
...     print(i)
...
0
1
2
3
4
5
6
7
8
9
>>> for i in range(1, 11):
...     print(i)
...
1
2
3
4
5
6
7
8
9
10
>>>
```

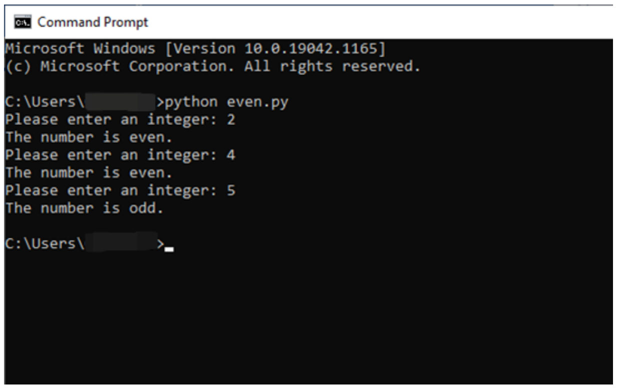
Note that the computer counts from 0 unless you designate the starting number.

While statements

The while statement is used for repeated execution as long as a condition is true. Repeated execution of a set of statements is called **iteration**. If the condition is initially false, the loop body will not be executed.

The following example outputs the value of n until it reaches 5. The code is as follows:

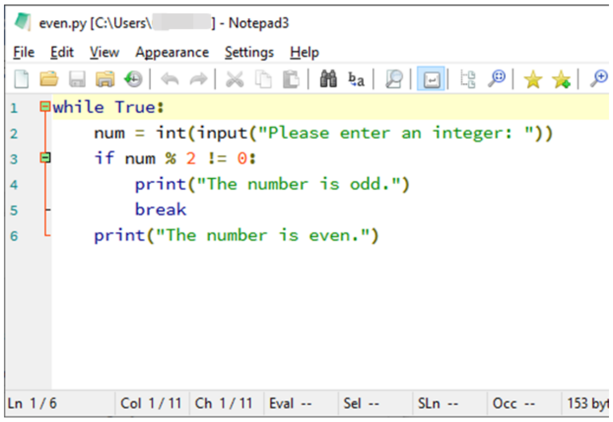
```
n = 1

while n < 6:

    print(n)

    n += 1
```



```
Microsoft Windows [Version 10.0.19042.1165]
(c) Microsoft Corporation. All rights reserved.

C:\Users\[redacted]>python even.py
Please enter an integer: 2
The number is even.
Please enter an integer: 4
The number is even.
Please enter an integer: 5
The number is odd.

C:\Users\[redacted]>
```

You can also make a script using both the if and while statements as bellow:

```
while True:

    num = int(input("Please enter an integer: "))

    if num % 2 != 0:

        print(“The number is odd.”)

        break

    print(“The number is even.”)
```


even.py [C:\Users\[redacted]] - Notepad3
File Edit View Appearance Settings Help
Ln 1 / 6
Col 1 / 11
Ch 1 / 11
Eval --
Sel --
SLn --
Occ --
153 byt

```
while True:
    num = int(input("Please enter an integer: "))
    if num % 2 != 0:
        print("The number is odd.")
        break
    print("The number is even.")
```

The **break** statement is used to terminate the current loop and resume execution at the next statement.

The while loop starts only if the condition evaluates to True. However, if a break statement is found, the loop immediately stops. Otherwise, the loop continues its normal execution, and it stops when the condition evaluates to False.

Written by Ryan Clow, Digital Scholarship Training Specialist, 02/2023

This work is licensed under the Creative Commons Attribution 4.0 International License.
To view a copy of this license, visit https://creativecommons.org/licenses/by/4.0