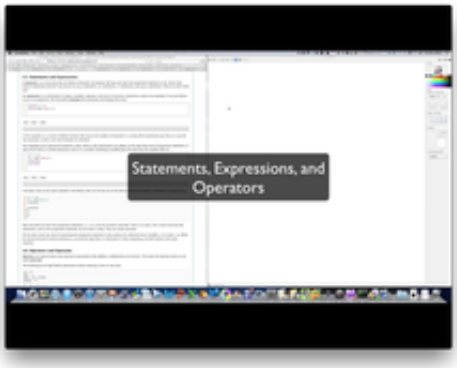
Statements and Expressions



A **statement** is an instruction that the Python interpreter can execute. We have only seen the assignment statement so far. Some other kinds of statements that we'll see shortly are `while` statements, `for` statements, `if` statements, and `import` statements. (There are other kinds too!)

An **expression** is a combination of values, variables, operators, and calls to functions. Expressions need to be evaluated. If you ask Python to `print` an expression, the interpreter **evaluates** the expression and displays the result.

```
1 print(1 + 1)
2 print(len("hello"))
3
```

ActiveCode: 1 (ch02_13)

Run

```
2
5
```

In this example `len` is a built-in Python function that returns the number of characters in a string. We've previously seen the `print` and the `type` functions, so this is our third example of a function!

The *evaluation of an expression* produces a value, which is why expressions can appear on the right hand side of assignment statements. A value all by itself is a simple expression, and so is a variable. Evaluating a variable gives the value that the variable refers to.

```

```

```
y = 3.14
x = len("hello")
print(x)
print(y)

```

ActiveCode: 2 (ch02_14)

Run

```
5
3.14
```

If we take a look at this same example in the Python shell, we will see one of the distinct differences between statements and expressions.

```
>>> y = 3.14
>>> x = len("hello")
>>> print(x)
5
>>> print(y)
3.14
>>> y
3.14
>>>
```

Note that when we enter the assignment statement, `y = 3.14`, only the prompt is returned. There is no value. This is due to the fact that statements, such as the assignment statement, do not return a value. They are simply executed.

On the other hand, the result of executing the assignment statement is the creation of a reference from a variable, `y`, to a value, `3.14`. When we execute the print function working on `y`, we see the value that y is referring to. In fact, evaluating `y` by itself results in the same response.

Operators and Operands

Operators are special tokens that represent computations like addition, multiplication and division. The values the operator works on are called **operands**.

The following are all legal Python expressions whose meaning is more or less clear:

```
20 + 32
hour - 1
hour * 60 + minute
minute / 60
5 ** 2
(5 + 9) * (15 - 7)
```

The tokens `+`, `-`, and `*`, and the use of parenthesis for grouping, mean in Python what they mean in mathematics. The asterisk (`*`) is the token for multiplication, and `**` is the token for exponentiation. Addition, subtraction, multiplication, and exponentiation all do what you expect.

```
1 print(2 + 3)
2 print(2 - 3)
3 print(2 * 3)
4 print(2 ** 3)
5 print(3 ** 2)
6
```

ActiveCode: 3 (ch02_15)

Run

```
5
-1
6
8
9
```

When a variable name appears in the place of an operand, it is replaced with the value that it refers to before the operation is performed. For example, what if we wanted to convert 645 minutes into hours. In Python 3, division is denoted by the operator token `/` which always evaluates to a floating point result.

```
1 minutes = 645
2 hours = minutes / 60
3 print(hours)
4
```

ActiveCode: 4 (ch02_16)

Run

```
10.75
```

What if, on the other hand, we had wanted to know how many *whole* hours there are and how many minutes remain. To help answer this question, Python gives us a second flavor of the division operator. This version, called **integer division**, uses the token `//`. It always *truncates* its result down to the next smallest integer (to the left on the number line).

```
1 print(7 / 4)
2 print(7 // 4)
3 minutes = 645
4 hours = minutes // 60
5 print(hours)
6
```

ActiveCode: 5 (ch02_17)

Run

```
1.75
1
10
```

Pay particular attention to the first two examples above. Notice that the result of floating point division is `1.75` but the result of the integer division is simply `1`. Take care that you choose the correct flavor of the division operator. If you're working with expressions where you need floating point values, use the division operator `/`. If you want an integer result, use `//`.

The **modulus operator**, sometimes also called the **remainder operator** or **integer remainder operator** works on integers (and integer expressions) and yields the remainder when the first operand is divided by the second. In Python, the modulus operator is a percent sign (`%`). The syntax is the same as for other operators.

```
1 quotient = 7 // 3     # This is the integer division operator
2 print(quotient)
3 remainder = 7 % 3
4 print(remainder)
5
```

ActiveCode: 6 (ch02_18)

Run

```
2
1
```

In the above example, 7 divided by 3 is 2 when we use integer division and there is a remainder of 1.

The modulus operator turns out to be surprisingly useful. For example, you can check whether one number is divisible by another—if `x % y` is zero, then `x` is divisible by `y`. Also, you can extract the right-most digit or digits from a number. For example, `x % 10` yields the right-most digit of `x` (in base 10). Similarly `x % 100` yields the last two digits.

Finally, returning to our time example, the remainder operator is extremely useful for doing conversions, say from seconds, to hours, minutes and seconds. If we start with a number of seconds, say 7684, the following program uses integer division and remainder to convert to an easier form. Step through it to be sure you understand how the division and remainder operators are being used to compute the correct values.

```
1  total_secs = 7684
2  hours = total_secs // 3600
3  secs_still_remaining = total_secs % 3600
4  minutes =  secs_still_remaining // 60
5  secs_finally_remaining = secs_still_remaining  % 60
```

<< First | < Back | Program terminated | Forward > | Last >>

line that has just executed

next line to execute

Frames Objects

Global variables

total_secs	7684
hours	2
secs_still_remaining	484
minutes	8

secs_finally_remaining	4

CodeLens: 1 (ch02_19_codelens)

Check your understanding

sdat-6: What value is printed when the following statement executes?

```
print (18 / 4)
```

- (•) a) 4.5
- ( ) b) 5
- ( ) c) 4
- ( ) d) 2

Check Me | Compare Me

Correct!! The / operator does exact division and returns a floating point result.

sdat-7: What value is printed when the following statement executes?

```
print (18 // 4)
```

- ( ) a) 4.25
- ( ) b) 5
- (•) c) 4
- ( ) d) 2

Check Me | Compare Me

Correct!! - The // operator does integer division and returns the truncated integer result.

sdat-8: What value is printed when the following statement executes?

```
print (18 % 4)
```

- a) 4.25
- b) 5
- c) 4
- d) 2

Check Me | Compare Me

Correct!! The % operator returns the remainder after division.