Object Oriented Programming II

Dr. Burak Kaleci

February 7, 2019

Content

Performing Calculations

More About Data Output

Introduction

- Most real-world algorithms require calculations to be performed. A programmer's tools for performing calculations are math operators. Table lists the math operators that are provided by the Python language.

Symbol	Operation	Description
+	Addition	Adds two numbers
-	Subtraction	Subtracts one number from another
*	Multiplication	Multiplies one number by another
/	Division	Divides one number by another and gives the result as a floating-point number
//	Integer division	Divides one number by another and gives the result as an integer
%	Remainder	Divides one number by another and gives the remainder
**	Exponent	Raises a number to a power

An Example

Program 2-14 (simple_math.py)

```
1  # Assign a value to the salary variable.
2  salary = 2500.0
3
4  # Assign a value to the bonus variable.
5  bonus = 1200.0
6
7  # Calculate the total pay by adding salary
8  # and bonus. Assign the result to pay.
9  pay = salary + bonus
10
11 # Display the pay.
12 print('Your pay is', pay)
```

Program Output

```
Your pay is 3700.0
```

- When we use a math expression to calculate a value, normally we want to save that value in memory so we can use it again in the program. We do this with an assignment statement.
- Line 2 assigns 2500.0 to the salary variable, and line 5 assigns 1200.0 to the bonus variable.
- Line 9 assigns the result of the expression salary + bonus to the pay variable.
- As you can see from the program output, the pay variable holds the value 3700.0.

Another Example

```
 8 # This program gets an item's original price and
 9 # calculates its sale price, with a 20% discount
10
11 # Get the item's original price.
12 original_price = float(input("Enter the item's original price: "))
13
14 # Calculate the amount of the discount.
15 discount = original_price * 0.2
16
17 # Calculate the sale price.
18 sale_price = original_price - discount
19
20 # Display the sale price.
21 print('The sale price is', sale_price)
```

Floating-Point and Integer Division

- Notice in Table that Python has two different division operators.
- The / operator performs floating-point division, and the //operator performs integer division.
- Both operators divide one number by another.
- The difference between them is that the /operator gives the result as a floating-point value, and the //operator gives the result as an integer.

```
In [2]: 5/2
Out[2]: 2.5

In [3]: 5//2
Out[3]: 2

In [4]: -5//2
Out[4]: -3
```

The //operator works like this:

- When the result is positive, it is truncated, which means that its fractional part is thrown away.
- When the result is negative, it is rounded away from zero to the nearest integer.

Operator Precedence

- First, operations that are enclosed in parentheses are performed first.
- Then, when two operators share an operand, the operator with the higher precedence is applied first.
- The precedence of the math operators, from highest to lowest, are:
  - Exponentiation (**)
  - Multiplication (*), division (/ and //), and remainder (%)
  - Addition (+) and subtraction (-)
- Notice that the multiplication (*), floating-point division (/), integer division (//), and remainder (%) operators have the same precedence.
- The addition (+) and subtraction (-) operators also have the same precedence.
- When two operands with the same precedence share an operand, the operators execute **from left to right**.

Average Example

```
 8 # Get three test scores and assign them to the
 9 # test1, test2, and test3 variables.
10
11 test1 = float(input('Enter the first test score: '))
12 test2 = float(input('Enter the second test score: '))
13 test3 = float(input('Enter the third test score: '))
14
15 # Calculate the average of the three scores
16 # and assign the result to the average variable.
17 average = (test1 + test2 + test3) / 3.0
18
19 # Display the average.
20 print('The average score is', average)
```

The Exponent Operator

- In addition to the basic math operators for addition, subtraction, multiplication, and division, Python also provides an exponent operator.
- Two asterisks written together (**) is the exponent operator, and its purpose it to raise a number to a power.
- For example, the following statement raises the length variable to the power of 2 and assigns the result to the area variable:

  area = length**2

```
In [6]: 4**2
Out[6]: 16

In [7]: 5**3
Out[7]: 125

In [8]: 2**10
Out[8]: 1024
```

The Remainder Operator

- In Python, the % symbol is the remainder operator. (This is also known as the modulus operator.)
- The remainder operator performs division, but instead of returning the quotient, it returns the remainder. The following statement assigns 2 to leftover:

  leftover = 17 % 3

```
 8 # Get a number of seconds from the user.
 9 total_seconds = float(input('Enter a number of seconds: '))
10
11 # Get the number of hours.
12 hours = total_seconds // 3600
13
14 # Get the number of remaining minutes.
15 minutes = (total_seconds // 60) % 60
16
17 # Get the number of remaining seconds.
18 seconds = total_seconds % 60
19
20 # Display the results.
21 print('Here is the time in hours, minutes, and seconds:')
22 print('Hours:', hours)
23 print('Minutes:', minutes);
24 print('Seconds:', seconds)
```

Mixed-Type Expressions and Data Type Conversion

When you perform a math operation on two operands, the data type of the result will depend on the data type of the operands. Python follows these rules when evaluating mathematical expressions:

- When an operation is performed on **two int** values, the result will be an **int**.
- When an operation is performed on **two float** values, the result will be a **float**.
- When an operation is performed on an **int** and a **float**, the int value will be temporarily converted to a float and the **result of the operation will be a float**.
- Consider the following statement:

  my_number = 5 * 2.0
- When this statement executes, the value 5 will be converted to a float(5.0) and then multiplied by 2.0. The result, 10.0, will be assigned to my_number.

Mixed-Type Expressions and Data Type Conversion

- The int to float conversion that takes place in the previous statement happens implicitly.
- If you need to explicitly perform a conversion, you can use either the int() or float() functions.
- For example, you can use the int() function to convert a floating-point value to an integer, as shown in the following code:

  fvalue = 2.6
  ivalue = int(fvalue)
- As demonstrated in the previous example, the int() function converts a floating-point argument to an integer by truncating it.
- As previously mentioned, that means it throws away the number's fractional part.
- Here is an example that uses a negative number:

  fvalue = -2.9
  ivalue = int(fvalue)
- In the second statement, the value -2 is returned from the int() function.

Breaking Long Statements into Multiple Lines

- Python allows you to break a statement into multiple lines by using the line continuation character,which is a backslash (\).
- You simply type the backslash character at the point you want to break the statement, and then press the Enter key.
- Here is a print function call that is broken into two lines with the line continuation character:

```
print('We sold', units_sold, \
    'for a total of', sales_amount)
```

- The line continuation character that appears at the end of the first line tells the interpreter that the statement is continued on the next line.
- Here is a statement that performs a mathematical calculation and has been broken up to fit on two lines:

```
result = var1 * 2 + var2 * 3 + \
    var3 * 4 + var4 * 5
```

Specifying an Item Separator

- When multiple arguments are passed to the print function, they are automatically separated by a space when they are displayed on the screen.
- If you do not want a space printed between the items, you can pass the argument **sep=”** to the print function.
- You can also use this special argument to specify a character other than the space to separate multiple items.

```
In [10]: print('One', 'Two', 'Three')
One Two Three

In [11]: print('One', 'Two', 'Three', sep='')
OneTwoThree

In [12]: print('One', 'Two', 'Three', sep='*')
One*Two*Three

In [13]: print('One', 'Two', 'Three', sep='---')
One---Two---Three
```

Specifying an Item Separator

- An escape character is a special character that is preceded with a backslash (\), appearing inside a string literal.
- When a string literal that contains escape characters is printed, the escape characters are treated as special commands that are embedded in the string.
- For example, \n is the newline escape character. When the \n escape character is printed, it isn't displayed on the screen. Instead, it causes output to advance to the next line.

Escape Character	Effect
\n	Causes output to be advanced to the next line.
\t	Causes output to skip over to the next horizontal tab position.
\'	Causes a single quote mark to be printed.
\"	Causes a double quote mark to be printed.
\\	Causes a backslash character to be printed.

Formatting Numbers

- You might not always be happy with the way that numbers, especially floating-point numbers, are displayed on the screen.
- When a floating-point number is displayed by the print statement, it can appear with up to 12 significant digits.
- This is shown in the output of the following program.
- Because this program displays a dollar amount, it would be nice to see that amount rounded to two decimal places.
- Fortunately, Python gives us a way to do just that, and more, with the built-in **format** function.

Program 2-19 (no_formatting.py)

```
1  # This program demonstrates how a floating-point
2  # number is displayed with no formatting.
3  amount_due = 5000.0
4  monthly_payment = amount_due / 12.0
5  print('The monthly payment is', monthly_payment)
```

Program Output

```
The monthly payment is 416.666666667
```

Formatting Numbers

- When you call the built-in **format** function, you pass two arguments to the function:
  - a numeric value
  - a format specifier.
- The format specifier is a string that contains special characters specifying how the numeric value should be formatted.
- Let's look at an example:
  format(12345.6789, '.2f')
- The first argument, which is the floating-point number 12345.6789, is the number that we want to format.
- The second argument, which is the string '.2f', is the format specifier.

Formatting Numbers

- Here is the meaning of its contents:
  - The .2 specifies the precision. It indicates that we want to round the number to two decimal places.
  - The f specifies that the data type of the number we are formatting is a floating-point number.
- The **format function returns a string** containing the formatted number.
- The format specifier can also include a minimum field width, which is the minimum number of spaces that should be used to display the value.
- Here is an example that specifies field width and precision:

```
print('The number is', format(12345.6789, '12.2f'))
```

- You can also use the format function to format integers.
- In the following statement, the number 123456 is printed in a field that is 10 spaces wide:

```
print(format(123456, '10d'))
```