01 Variables, assignment and operator precedence

September 13, 2020

1 Introduction

We begin with assignment to variables and familiar mathematical operations.

1.1 Objectives

- Introduce expressions and basic operators
- Introduce operator precedence
- Understand variables and assignment

2 Evaluating expressions: simple operators

We can use Python like a calculator. Consider the simple expression $3 + 8$. We can evaluate and print this by:

[1]:
```
3 + 8
```

[1]:
```
11
```

Another simple calculation is the gravitational potential V of a body of mass m (point mass) at a distance r from a body of mass M, which is given by

$$V = \frac{GMm}{r}$$

where G is the *gravitational constant*. A good approximation is $G = 6.674 \times 10^{-11}$ N m^2 kg^{-2}.

For the case $M = 1.65 \times 10^{12}$ kg, $m = 6.1 \times 10^2$ kg and $r = 7.0 \times 10^3$ m, we can compute the gravitational potential V:

[2]:
```
6.674e-11*1.65e12*6.1e2/7.0e3
```

[2]:
```
9.59625857142857
```

We have used 'scientific notation' to input the values. For example, the number 8×10^{-2} can be input as `0.08` or `8e-2`. We can easily verify that the two are the same via subtraction:

[3]:
```
0.08 - 8e-2
```

[3]:
```
0.0
```

A common operation is raising a number to a power. To compute 3^4:

[4]:
```
3**4
```

[4]:
```
81
```

The remainder is computed using the modulus operator '`%`':

[5]:
```
11 % 3
```

[5]:
```
2
```

To get the quotient we use 'floor division', which uses the symbol '`//`':

[1]:
```
11 // 3
```

3 Operator precedence

Operator precedence refers to the order in which operations are performed, e.g. multiplication before addition. In the preceding examples, there was no ambiguity as to the order of the operations. However, there are common cases where order does matter, and there are two points to consider:

- The expression should be evaluated correctly; and
- The expression should be simple enough for someone else reading the code to understand what operation is being performed.

It is possible to write code that is correct, but which might be very difficult for someone else (or you) to check.

Most programming languages, including Python, follow the usual mathematical rules for precedence. We explore this through some examples.

Consider the expression $4 \cdot (7 - 2) = 20$. If we are careless,

[7]:
```
4*7 - 2
```

[7]:
```
26
```

In the above, `4*7` is evaluated first, then `2` is subtracted because multiplication (`*`) comes before subtraction (`-`) in terms of precedence. We can control the order of the operation using brackets, just as we would on paper:

[8]:
```
4*(7 - 2)
```

[8]:
```
20
```

A common example where readability is a concern is

$$\frac{10}{2 \times 50} = 0.1$$

The code

[9]:
```
10/2*50
```

[9]:
```
250.0
```

is not consistent with what we wish to compute. Multiplication and division have the same precedence, so the expression is evaluated 'left-to-right'. The correct result is computed from

[10]:
```
10/2/50
```

[10]:
```
0.1
```

but this is hard to read and could easily lead to errors in a program. Better is to use brackets to make the order clear:

[11]:
```
10/(2*50)
```

[11]:
```
0.1
```

Here is an example that computes $2^3 \cdot 4 = 32$ which is technically correct but not ideal in terms of readability:

[12]:
```
2**3*4
```

[12]:
```
32
```

Better would be:

[13]:
```
(2**3)*4
```

[13]:
```
32
```

4 Variables and assignment

The above code snippets were helpful for doing some arithmetic, but we could easily do the same with a pocket calculator. Also, the snippets are not very helpful if we want to change the value of one of the numbers in an expression, and not very helpful if we wanted to use the value of an expression in a subsequent computation. To improve things, we need *assignment*.

When we compute something, we usually want to store the result so that we can use it in subsequent computations. *Variables* are what we use to store something, e.g.:

[14]:
```
c = 10
print(c)
```

```
10
```

Above, the variable `c` is used to 'hold' the value `10`. The *function* `print` is used to print the value of a variable to the output (more on functions later).

Say we want to compute $c = a + b$, where $a = 2$ and $b = 11$:

[15]:
```
a = 2
b = 11
c = a + b
print(c)
```

```
13
```

What is happening above is that the expression on the right-hand side of the assignment operator '`=`' is evaluated and then stored as the variable on the left-hand side. You can think of the variable as a 'handle' for a value. If we want to change the value of a to 4 and recompute the sum, we would just replace `a = 2` with `a = 4` and execute the code (try this yourself by running this notebook interactively).

The above looks much like standard algebra. There are however some subtle differences. Take for example:

[16]:
```
a = 2
b = 11
a = a + b
print(a)
```

```
13
```

This is not a valid algebraic statement since '`a`' appears on both sides of '`=`', but it is a very common statement in a computer program. What happens is that the expression on the right-hand side is evaluated (the values assigned to `a` and `b` are summed), and the result is assigned to the left-hand side (to the variable `a`). There is a mathematical notation for this type of assignment:

$$a \leftarrow a + b$$

which says 'sum a and b, and copy the result to a'. You will see this notation in some books, especially when looking at *algorithms*.

4.1 Shortcuts

Adding or subtracting variables is such a common operation that most languages provide shortcuts. For addition:

[3]:
```
# Long-hand addition
a = 1
a = a + 4
print(a)

# Short-hand addition
a = 1
```

```
a += 4
print(a)
```

```
5
5
```

> In Python, any text following the hash (#) symbol is a 'comment'. Comments are not executed by the program; they help us document and explain what our programs do. Use comments in your programs.

For subtraction:

[18]:
```
# Long-hand subtraction
a = 1
b = 4
a = a - b
print(a)

# Short-hand subtraction
a = 1
b = 4
a -= b
print(a)
```

```
-3
-3
```

Analogous assignment operators exist for multiplication and division:

[19]:
```
# Long-hand multiplication
a = 10
c = 2
a = c*a
print(a)

# Short-hand multiplication
a = 10
c = 2
a *= c
print(a)

# Long-hand division
a = 1
a = a/4
print(a)

# Short-hand division
a = 1
a /= 4
```

```
print(a)
```

```
20
20
0.25
0.25
```

4.2 Naming variables

It is good practice to use meaningful variable names in a computer program. Say you used '`x`' for time, and '`t`' for position, you or someone else will almost certainly make errors at some point. If you do not use well-considered variable names:

1. You're much more likely to make errors.
2. When you come back to your program after some time you will have trouble recalling and understanding what the program does.
3. It will be difficult for others to understand your program - serious program development is almost always a team effort.

Languages have rules for what charcters can be used in variable names. As a rough guide, in Python variable names can use letters and digits, but cannot start with a digit.

Sometimes for readability it is useful to have variable names that are made up of two words. A convention is to separate the words in the variable name using an underscore '`_`'. For example, a good variable name for storing the number of days would be

```
num_days = 10
```

Python is a case-sensitive language, e.g. the variables '`A`' and '`a`' are different. Some languages, such as Fortran, are case-insensitive.

Languages have reserved keywords that cannot be used as variable names as they are used for other purposes. The reserved keywords in Python are:

[20]:
```
import keyword
print(keyword.kwlist)
```

```
['False', 'None', 'True', 'and', 'as', 'assert', 'break', 'class', 'continue',
'def', 'del', 'elif', 'else', 'except', 'finally', 'for', 'from', 'global',
'if', 'import', 'in', 'is', 'lambda', 'nonlocal', 'not', 'or', 'pass', 'raise',
'return', 'try', 'while', 'with', 'yield']
```

If you try to assign something to a reserved keyword, you will get an error.

Python 3 supports Unicode, which allows you to use a very wide range of symbols, including Greek characters:

[21]:
```
  = 10
  = 12
  =  +
print( )
```

22

Greek symbols and other symbols can be input in a Jupyter notebook by typing the LaTeX command for the symbol and then pressing the `tab` key, e.g. '`\theta`' followed by pressing the `tab` key.

5 Exercises

Complete now the 01 Exercises notebook.