Python #1
Python Fundamentals

Location: https://people.sc.fsu.edu/~jburkardt/classes/python_2022/python01/python01.pdf

Freely adapted from the Python lessons at https://software-carpentry.org/

Python Fundamentals

- *Python can be used interactively, as a calculator;*
- *We can save useful data using named variables;*
- *Types of data include integers, real numbers, and strings;*
- *There are some simple rules for choosing variable names;*
- *We can manipulate data once it is stored as variables;*
- *Python functions include* `print()` *and* `type()`*;*
- *You can create new functions to simplify your work.*

1 Interactive calculator

Once you have started a Python session, a simple process is to use it as a calculator. You can write formulas involving numbers, the operators +, -, *, /, **, and parentheses, and Python will print out the value:

```
>>> 3 + 5 * 4
23
>>> 365/12
30.416666666666668
>>> 9**3
729
```

Sometimes, when you are carrying out division, you may want an integer value. In such a case, an easy way to do this is to use the '//' operator, which carries out division but then rounds the result:

```
>>> 7 // 2
3
>>> -7 // 2
-4   # Rounding is ''down'', that is, towards negative infinity
>>> 365//12
30
>>> 365.25 // 12.0
30.0
```

If you have a very large or very small decimal value, you can use exponential format. Thus, 1200 can be written as 1.2E3, and 0.00345 as 3.45E-3.

In some computations, it is necessary to use complex numbers. Python writes a complex number in the form `a+bj` or `(a+bj)`, where `j` represents the imaginary unit, that is, $\sqrt{-1}$. Thus, we can write

```
>>> x = 2 + 3j
>>> y = -4j
>>> z = x * y
(12-8j)
```

Note that if you want to represent $\sqrt{-1}$, you write `1j`, not just `j`:

```
>>> u = 1j
>>> v = u**2
(-1+0j)
```

It is somewhat annoying, but if you try to compute $\sqrt{-1}$, you get an error:

```
>>> u = sqrt ( -1.0 )
ValueError: math domain error
```

This is because your input value `-1.0` is a real number, and Python doesn't realize that you will accept a complex result. To work around this blind spot, get a smarter version of the `sqrt()` function from the `numpy` library, and then replace your original input -1.0 by something that is equal, (-1.0+0j), but which looks like a complex number to Python:

```
>>> import numpy as np
>>> u = np.sqrt ( -1.0 + 0j )
```

We will see more about the `numpy` library shortly!

2 A simple iteration

Is there a value of x for which it is true that $\cos(x) = x$? Strangely enough, it is easy to investigate this question using Python. Let's start by getting the `numpy` library so that the cosine function is defined, and then setting $x = 1$.

```
>>> import numpy as np
>>> x = 1.0
```

Now we are going to replace x by the value $\cos(x)$. Since we want to print this value automatically, we are going to tack on a `print()` statement on the same command line. That is possible as long as we separate the two statements by a semicolon:

```
>>> x = np.cos ( x ); print ( x )
```

Now the funny thing is that we can repeat this command simply by hitting the up-arrow key. And the even funnier thing is that the values we compute seem to be converging. Here are the first 10 values I computed:

```
0.5403023058681398
0.8575532158463934
0.6542897904977791
0.7934803587425656
0.7013687736227565
0.7639596829006542
0.7221024250267077
0.7504177617637605
```

```
0.7314040424225098
0.7442373549005569
```

So the first digit has stopped changing, and the second is about to settle down. If you are patient, you can determine the third and fourth digits, and so on.

If this interests you, investigate two more iteration formulas. One of them also converges, while the other seems to be chaotic.

```
x = x/2 + 1/x
x = 2 * abs ( x - 0.5 )
```

You can actually solve the first equation to determine the two "magic" values of x.

3 Variables

Having an interactive calculator can be useful, but Python can do much more for you. In particular, if we want to process data, we will need to get used to the idea of storing values in *variables*. A variable is simply a name to which we can associate information. We assign a value to a variable by using the equals sign.

For example, a hospital visit often begins by assigning an identifier to a patient. To store a text string, we surround the text with single or double quotes:

```
>>> patient_id = 'Tom Jones'
```

After assigning the identifier, the patient's weight and height would be measured and stored. Instead of writing down these numbers, we could store them as named variables:

```
>>> weight_lb = 145
>>> height_in = 60
```

These are examples of *assignment statements*, in which values are being given names. There are rules for choosing variable names:

- formed from letters, digits, and underscores;
- cannot start with a digit;
- are case sensitive.

4 Using and modifying variables

The advantage of using variable names is that we can apply formulas to the symbolic names in a natural way. For instance, to convert from pounds to kilograms, we divide by 2.204. To convert inches to meters, we divide by 39.370. Thus, if our hospital actually uses the metric system, we can convert our readings using the following formulas:

```
>>> weight_kg = weight_lb / 2.204
65.789
>>> height_m = height_in / 39.370
1.524
```

We can use the addition operator + to combine two strings, or to modify the value of a variable that already contains a string:

```
>>> hospital = 'UPMC' + 'Shadyside'
>>> patient_id = 'Mr ' + patient_id
```

Since we want to separate the 'Mr' from the name, we need to include a space. But if we want to make it more clear what we are doing, we could instead write:

```
>>> patient_id = 'Mr' + ' ' + patient_id
```

Note that, if you issue the last two commands in a row, you will actually get a somewhat mangled version of the patient's name. In that case, you may have to fix this by resetting the variable:

```
>>> patient_id = 'Mr Tom Jones'
```

You should have noticed that, in Python, the = sign does not really indicate equality, but rather *assigment*. A data value, variable number, or formula involving data values and variables appears on the right hand side of the = sign. On the left is the name of the variable into which the resulting value will be stored:

```
>>> a = 1
>>> b = a + 2
>>> c = a * b + 3
>>> a = a + 2
```

The last assignment in the above list would make no sense to a mathematician. But you should understand its meaning in Python: *Get the current value of* `a`*, add 2, and use this as the updated value of* `a`.

5 The built-in print() function

The Python language includes many built-in functions, which allow you to carry out common tasks on your data. One of the simplest is the `print()` function, which is given the name of a data item, and displays its value.

```
>>> print ( weight_lb )
```

We can include a descriptive label as part of the printout:

```
>>> print ( 'The weight in pounds is', weight_lb )
```

and we can print several variables in one statement:

```
>>> print ( 'The weight of', patient_id , 'in pounds is', weight_lb )
```

Notice that the `print()` statement automatically separates multiple items of data using a single blank space.

Occasionally, we want to print a string that already contains a quotation mark, and this could confuse Python, which expects such marks to be used to show the beginning and end of the string. Because of that, the following command will **not** be acceptable to Python:

```
>>> print ( 'The patient's weight in pounds is', weight_lb )
```

There are two alternatives. We may be able to get around the problem by using double quotation marks to enclose the string:

```
>>> print ( "The patient's weight in pounds is", weight_lb )
```

or we can simply replace the interfering quotation mark by a pair of such marks, Python gets the point, and prints the information correctly.

```
>>> print ( 'The patient''s weight in pounds is', weight_lb )
```

All Python functions have a name, followed by a list of input, which is surrounded by parentheses. Python will not realize that you are calling a function unless the parentheses are there. Consider what happens when you type the following two commands

```
>>> print weight_kg
>>> print
```

6 The built-in type() function

We have already seen that a variable can hold numeric or string information, and there are a number of other kinds of data that can be involved. Sometimes you can lose track of what you stored in a variable, or a program might be so large that the definition occurred a long time ago. If you wish to know what kind of data is in a variable, There is a built-in Python function called `type()`. Using this function also reveals the labels that Python uses when it classifies your data.

Consider the results of each of the following commands:

```
>>> type ( weight_lb )
>>> type ( weight_kg )
>>> type ( patient_id )
>>> type ( 707 )
>>> type ( print )
```

7 A user function

Hospitals often combine the patient's height and weight in a formula known as the body-mass-index (BMI). In metric units, this is defined by the weight in kilograms, divided by the square of the height in meters. Mathematically, we might write:

$$\text{bmi} = \frac{\text{kg}}{\text{m}^2}$$

Unfortunately, if we are using pounds and inches, there is more work involved to get the proper result. We can convert the weight and height separately, and then combine them, or even try a one shot formula:

$$\text{bmi} = \frac{\text{lb}}{2.204}\left(\frac{39.370}{\text{in}}\right)^2$$

Let's suppose this is the formula we want to implement. If we are going to have to compute the BMI many times, we are going to be find this procedure complicated to use, with numeric constants that are tedious to enter each time.

Fortunately, this is one of the first advantages of working in a computer language. We can recognize that the BMI formula is important, and write it as a new user *function*, similar to the built-in functions `print` and `type`. That means we can get our result by a statement like this:

```
>>> value = bmi ( weight_lb , height_in )
```

How do we make this happen? We define our new function with the following text:

```
>>> def bmi ( weight_lb , height_in ):
>>>     value = ( weight_lb / 2.204 ) * ( 39.370 / height_in )**2
>>>     return value
```

Read the first line carefully. The word `def` is used to indicate that what follows defines a function. The word `bmi` is the name of the function. The names that follow, inside parentheses, are the function *parameters* or simply *input*. They are simply the names we give to values that we expect a user to supply.

The lines that follow explain how the function works. As you will see, the fact that these lines are indented is an important feature of Python. Here, it indicates that these indented lines are to be regarded as part of this function definition. Once you have defined the function, you can stop indenting any new Python commands you want to issue (in fact, you `must` stop indending!)

The first line carries out the calculation as we have already seen. Note that you square a value by using the `**2` expression. We store the result as a variable called `value`, although we are free to choose another name if we like.

The last line uses the word `return` to indicate what information, computed inside the function, is to be given to the "outside world". In this case, we only want to return the quantity we called `value`.

If everything is working properly, once you have defined `bmi`, you can use it by typing commands like:

```
>>> bmi ( 125, 60 )
>>> bmi ( 200, 73 )
>>> bmi ( 150, 63 )
```

We can terminate our Python session by typing `quit()` or by closing the terminal. However, once we do that, our wonderful function `bmi()` disappears. If we need it again for a later session, we have to type it in all over again. But the point of this function was to save that information once and for all, so we did not have to remember it. Is there a way that we can save a useful function once we have created it? We will see later that we can store our function in a file, so that it will be available later, whenever we need it.

8 Exercises

Based on the following lines of code, what will be the final values of `mass` and `age`? Verify your calculations by executing the Python code.

```
>>> mass = 47.5
>>> age = 122
>>> mass = mass * 2.0
>>> age = age - 20
```

Python allows you to assign separate values to multiple variables in one line, by separating the variable names and values with commas. What does the following program print out?

```
>>> first, second = 'Grace', 'Hopper'
>>> third, fourth = second, first
>>> print ( third, fourth )
```

Python allows you to request the type of an object. What are the types of these quantities after the following commands have been executed?

```
>>> alpha = 2
>>> beta = 'beta'
>>> gamma = 7
>>> alpha = 1.5 * alpha
```