# FirstJupyterNotebook

January 12, 2021

## 1 Welcome to Jupyter

This is one of the Python platforms we'll use in the course. Jupyter is a notebook-based Python platform that allows us to do bits of code at a time in cells (rather than make one huge program), make plots inside the notebook. It has two types of cells: cells containing text (like the one you're reading now), and cells where you can write some code (the one below here). If you're familiar with Mathematica, this will feel similar. We'll learn some of the more advanced features later on, but for now – move your cursor down to the cell below and hit CONTROL-ENTER to run it. You'll see some output appear below the cell. Try changing the text inside the `""` marks and running the cell again.

NOTE: There are two versions of python, 2 and 3, that are commonly used. One of the big differences is in the print command. Python2 wants it without parentheses: `print 2`, and Python3 wants it with: `print(2)`. If the computer complains about `SyntaxError: Missing parentheses in call to 'print'`, you're using Python3.

[ ]:
```
print("Hello, world!")
```

```
Hello, world!
```

## 2 Arithmetic expressions

The few examples below come from here.

Let's start with simple arithmetic expressions. The way that you write arithmetic expressions in Python is very similar to the way that you write arithmetic expressions in, say, grade school arithmetic, or algebra. In the example below, `3 + 5` is the expression, and `print` is a special Python keyword that tells Jupyter to put the results of evaluating the expression below the code.

[ ]:
```
print(3+5)
```

```
8
```

Arithmetic expressions in Python can be much more sophisticated than this, of course. We won't go over all of the details right now, but one thing you should know immediately is that Python arithmetic operations are evaluated using the typical order of operations, which you can override with parentheses:

[ ]:
```
print(4 + 5 * 6)
print((4 + 5) * 6)
```

```
34
54
```

You can write arithmetic expressions with or without spaces between the numbers and the operators (but usually it's considered better style to include spaces):

[ ]:
```
print(10+20+30)
```

```
60
```

Expressions in Python can also be very simple. In fact, a number on its own is its own expression, which Python evaluates to that number itself:

[ ]:
```
print(19)
```

```
19
```

If you write an expression that Python doesn't understand, then you'll get an error. Here's what that looks like:

[ ]:
```
print( + 20 19)
```

```
  File "<ipython-input-6-34fbe60f6cda>", line 1
    print( + 20 19)
                 ^
SyntaxError: invalid syntax
```

Python also supports comments – lines that do nothing, but are there so you can say something about the code. These are prefaced by a #

[ ]:
```
# This line does nothing.
```

## 3 Using python as a calculator

We can use Python as a calculator. Python itself can only do a few basic mathematical operations. Later, when we begin to write more sophisticated programs, we will need to import more functions. For now we can easily perform basic mathematical operations such as: 1. addition: 3 + 2 2. subtraction: 3 − 2 3. multiplication: 3 ∗ 2 4. divsion: 3/2 5. powers: 3 ∗ ∗2 6. Floor division: 3//2 – this somewhat unusual operation gives the largest whole number less than the result of the division 7. Modulus: 3%2 – this gives the remainder when the first number is divided by the second (in this case, the remainder of 3/2)

Calculations involving more than one operator can be performed, but we must pay attention to how we write it. For example, the expression:

```
1+2*3**4
```

will give as a result 163. Python performs calculations using the standard "orders of operations" giving the highest priority to powers, then multiplication and division, and finally addition and

subtraction. Therefore the expression above is equivalent to:

`1+(2*(3**4))`

but not

`(1+2)*3**4`

or

`1+(2*3)**4`

Thus, to avoid confusion and bugs, you should always use parentheses if it is not 100% clear what you mean.

### 3.1 Exercise: try all the expressions in a cell below with a variety of numbers

NOTE: Without a `print` command, the Jupyter notebook will only output the latest thing you entered.

NOTE: There are two versions of python, 2 and 3, that are commonly used. One of the big differences is in the print command. Python2 wants it without parentheses: `print 2`, and Python3 wants it with: `print(2)`. If the computer complains about `SyntaxError: Missing parentheses in call to 'print'`, you're using Python3.

[ ]:
```
# Run this cell.  Do you understand all the output?
print(3+2)
print(3-2)
print(3*2)
print(3/2)
print(3**2)
print(3//2)
```

```
5
1
6
1.5
9
1
```

## 4 Floating point numbers (AKA real or not-whole numbers)

Particular attention must be paid when using the division operator /. Python versions $< 3.0$ (like most other computer languages) use this operator in different ways for integers and real numbers. In particular, division between two integers can be rounded to an integer. If asked to compute 12/5, python $< 3.0$ will return 2 as the result. Python 3.0 will compute 12/5 as 2.4. A strong recommendation is to never use integers in your computer programs, unless explicitly required. Instead of writing 12/5 you should write 12.0/5.0. Then the computer will always return the correct result 2.4.

### 4.1 Exercise: Try the division (/) operator with real and integer numbers

Example: `5/2` vs `5.0/2.0`

[ ]:
```
# Run this cell.  Do you understand all the output?
print(13/2)
print(13./2.)
```

```
6.5
6.5
```

## 5 Mathematical functions

Python comes with a whole host of functions that we can import in a set of "libraries." It comes standard with the `math` library, which handles simple mathematical functions. However, we will be using a different one – the one that comes with `numpy` (the ones from `math` are more limited).

To get access to things provided by `numpy` we need to `import` them:

```
import numpy
```

Now, we have access to a variety of functions and constants that we may access with the `.` suffix:

```
numpy.pi
numpy.sin(30)
numpy.cos(numpy.pi)
numpy.log(10)
numpy.log(numpy.e)
numpy.sqrt(2)
numpy.sqrt(2.)
```

### 5.1 Exercise: Try the operations above and answer the following questions

- Does numpy use radians or degrees?
- What is the base of the `log` function?
- How would we use a log in a different base?
- How do the numpy functions treat integers?

[ ]:
```
# Type and execute the commands above in this cell -- be sure to include␣
↪'import numpy'
import numpy

print(numpy.pi)
print(numpy.sin(30))
print(numpy.cos(numpy.pi))
print(numpy.log(10))
print(numpy.log(numpy.e))
print(numpy.sqrt(2))
print(numpy.sqrt(2.))
```

```
3.141592653589793
-0.9880316240928618
-1.0
2.302585092994046
1.0
1.4142135623730951
1.4142135623730951
```

# 6 Strings

Python can deal with more than numbers and mathematical operators. It can also deal with lists of characters. A character is anything that you can type on a keyboard. If we enclose characters with a set of single quotes (') or double quotes (“), Python will interpret the characters as text, called a string. For example, 'NC State' and"University" are strings. A string can be used with the print command; for example,

```
print "PY251 is a blast."
```

Strings are often used to prompt the user of the routine to enter information, or to give the user context for output from the routine. For example:

```
print "The output of log(10) is "
print numpy.log(10)
```

Later on we will learn to combine these.

## 6.1 Exercise: Try out some string output

[ ]:
```
print("The output of log(10) is ", numpy.log(10))
```

```
The output of log(10) is  2.302585092994046
```

# 7 The = character and variables

In python, as in most other computer languages, the character = acts as an assignment operator. It tells the computer to calculate what is on the right–hand side (which must be a number or a string in quotes), then assign the result to the variable named on the left–hand side. Thus, for example,

```
x = 3 + 5
fruit = 'banana'
```

The first line instructs the computer to compute `3 + 5` and to assign the result, `8`, to the variable `x`. The second line instructs the computer to assign the string banana to the variable fruit.

To see what's in a variable, you can print the variable you want to inspect. For example, to see what's in the variables `x` and `fruit` that we just defined, run

```
print(x)
print(fruit)
```

In ordinary mathematics we refer to the character `=` as an "equals sign', but here it does not behave as in ordinary mathematics. Rather, it is called the"assignment operator," and it acts differently. For example, the following is a perfectly legitimate segment of python code:

```
x = 4
x = x + 7
```

The first statement assigns the number `4` to the variable `x`. The second statement computes `x + 7`, which is `11`, and then assigns the value `11` to the variable `x`. **You may wish to remember that the computer typically evaluates statements going from right to left.**

On the other hand, the following statements are not allowed:

```
3 = x
x + y = 5
3 * x = 15
```

Make sure you understand why these statements are not allowed.

### 7.1 Exercise

Type and execute each of the python commands above in the cell below, and see what happens. What happens when you try to print a variable you haven't defined yet?

*To put some code in a cell but not run it you can make it a "comment" by putting a `#` in front of it. This is useful for the commands above that aren't allowed (after you've tried them) – if you keep them in as actual code they'll stop the execution.*

[ ]: