Lecture Notes on Scientific Computing in Python

We use programming languages – Python, C/C++, Java, etc. – because, until very recently, computers weren't smart enough to convert natural human language into a form that would work for them. Our languages involve interpretation, and more vagueness than computers could deal with until now. (And even though they now can be "programmed" with human language, that process is prone to error.) Programming languages, in contrast, avoid the ambiguity in human languages; the cost is that they force us to think the way computers do. Even though computers now can be programmed somewhat effectively with natural language, the kind of ***algorithmic thinking*** that programming languages force us into is good because it provides a systematic way of tackling even difficult problems in mathematics, the sciences, and engineering. That is, it underlies analytical ("paper & pencil") problem-solving methods in those fields.

Question for students: what is an algorithm?
Simply put, an algorithm is a step-by-step procedure for achieving some goal. An everyday example of algorithmic thinking is a recipe: e.g., "combine 1 tsp salt and 3 cups flour in a bowl with 2 cups water; mix vigorously with spoon 10 times; if dough is lumpy, repeat another 5 times" etc. This simple made-up example involves all the key elements used in computer code: ***variables*** that take values (# tsps salt); actions that may be repeated in ***loops*** (mix vigorously); and ***conditional statements*** (if… then…) that involve checking whether/when specific conditions are met. These components often are "packaged" together into ***functions*** or ***routines***. These simple elements are the foundations of all computer code.

(Instructor tip: annotate the code cells created below with comments, which students will copy.)

Variables

- Types of variables: integers (1, 17, -6), floating point numbers ("floats"; 3.65, –7.2), strings ('greetings', "BMC is the best"), Booleans (TRUE, FALSE)

- Lists, arrays, and their differences: `[7, "howdy", 42.00]`, `[6, 12, 18, 24]`

- Named variables: `num = 3.5, times = 4`

Operations & Functions

- Basic mathematical operations: multiplication, division, addition, subtraction, power (`x**y`), modulo (`x%y` gives the remainder after dividing `x` by `y` an integer number of times; e.g., `16%7 = 2`)
  Exercise: create a computation involving the first five operations; check it by hand.

- The simple `print` statement: `print("text")`
  Exercise: print out a phrase of your choice.

- An OLD formatted `print` function:

  ```
  print("You typed the number %f %d times." %(num, times))
  ```

Here, `%f` is a placeholder indicating a floating-point number will be printed, `%d` indicates an integer to be printed. `% (num,times)` shows the values to be substituted for the `%f` and `%d`, in order.

- The NEW formatted print function, using "`f-strings`":

  ```
  print(f"You typed the number {num} {times} times.")
  ```
  Exercise: try this with your own values.

- The input statement: `reply = input("Enter the value of x: ")`
  Exercise: code this `reply` statement, provide an input, and run it.

  NOTE: This is a good way to get a single input, but not multiple ones.

- The output of this statement is formatted as a string in Python 3. To convert it to an `int` or `float`, "typecast" it; e.g., `reply = int(input("Enter the value of x: ")).`

Conditional Statements

- `if` and `if-else` statements:

  ```
  if x > 100:
         print("x is pretty big")

  if x > 100:
         print("x is pretty big")
  else:
         print("x is not so big")
  ```

  In one cell, set `x = 120`, then run first example. Change `x` to `90`, then retry.
  It is possible to have multiple `else` statements. This will be discussed in Module 1.

Loops

- The `for` loop:

  ```
  for i in [1, 2, 3]:
        print(i+1)
  ```

  Exercise: predict what this will output; then code it up and run it.

- The `while` loop:

  ```
  n = 0
  while n < 4:
        print(n+1)
        n = n + 1
  ```

  Exercise: predict what this will output; then code it up and run it.
  Note that the `while` requires "initializing" and explicitly incrementing the loop counter, `n` in this case (`i` in the previous one).

Creating and Using Functions

- Importing functions from modules (`from module import [function]` and importing entire modules or (`from module import *` or just `import module`). Module nicknames, e.g., `import numpy as np`. Use `help('[module]')` to get an extended "manual" printout of a modules functions.

- Constructing simple functions:
  Functions take inputs, called "arguments" (which are "passed to" the function). Functions can "return" values (or strings) using a `return` statement. The basic form of a function definition is

  ```
  def func([arguments]):
       <function statements here...
        [return]>
  ```

  E.g.,

  ```
  def my_info(name, age):
     print("My name is", name, "and my age is", age.)
     return name, age
  ```

  [Note that functions (and variables) with compound names typically are named in the form `wordWord` or `word_word`. It's good practice to name functions in an informative way, without making the name too long.] Run the function using the format

  ```
  my_info("Mark", 29)
  ```

  This will print the values, but it won't store them in a variable. To save the "returned" value(s) use the format

  ```
  a, b = my_info("Mark", 29)
  ```

  Of course, a function also can perform the kinds of operations introduced earlier. For a simple example, we can write a function to find the larger of two numbers.

  ```
  def larger(a, b):
     if a > b:
        print(a, "is greater than ", b)
        return a
     else:
        print(b, "is greater than ", a)
        return b
  ```

Transferring an ipynb File to jupyterhub

If the file is on Moodle, click its link. This will open a page that looks like text & gibberish.

In Windows: right-click on that page, choose Save File As and select All Files type. That should result in a .ipynb file that can be uploaded directly to jupyterhub. Delete any other extension.