Python basics

- bash command line basics, assume git bash installed on windows
  - bash = "Bourne-Again SHell"
  - critical commands:
    - `pwd` - print working directory
    - `cd` - change directories
    - `ls` - list directory info
  - others commonly used:
    - `mv` - move files/folders
    - `cp` - copy files/folders
    - `rm` - remove files/folders
    - `mkdir` - make directory
    - `touch` - create an empty file, or update last access time of existing file
    - `cat` - concatenate file(s)
  - `man COMMAND` and `COMMAND --help` for help
  - specifying paths:
    - `/` - filesystem root
    - `.` - current directory
    - `..` - parent directory
    - `~` - home folder
    - `-` - last used directory, i.e. `cd -` changes to last directory
  - up/down arrow keys to access recently used commands
  - quickly view file contents using `cat filename`
  - save text output of a command to file using redirection:
    - `ls -al > file_list.txt` - save detailed directory info to file
    - `cat > shopping_list.txt`
      - start typing, Ctrl+D on a blank line to finish writing to file
    - redirection `>` overwrites any existing file!
    - append to a file with `cat >>`, e.g. `cat >> shopping_list.txt`
- python interpreter
  - interpreted vs compiled languages
  - type `python` at the command line, type `exit()` or hit `Ctrl+D` to exit
  - calculator, math operators
    - `+`, `-`, `*`, `/`, `**`
  - up/down arrow keys to access recently used commands
- functions: take some kind of input, generate some kind of output
  - `print('hello world!')`
  - `s = input('hello? ')`
- make hello world script, run from command line
  - `python hello.py`
  - `#` is the comment character
- variable assignment
  - `a = 1`
  - multiple assignments on a single line (tuple expansion): `a, b = 1, 2`
  - in place math operators:
    - `+=`, `-=`, `*=`, `/=`
    - `a += 2` increments `a` by 2, `a *= 2` multiplies `a` by 2, stores result in `a`
  - variable names

        - case sensitive
        - letters, numbers, `_`
        - can't start with a number
- importing: gives you access to groups of other functions, in a "module"
    - e.g., `import math`
    - use `dir()` to find out what's available in a module
    - `dir(math)`
    - `math.sqrt()`
    - `math.log10()`
- help
    - in Python interpreter: `help(something)`
        - `q` to exit
    - online: search, StackExchange, or official `http://docs.python.org`
- basic Python data types
    - `int, float, str, bool`
        - types also are functions that convert input to that type
        - literals: `1, 1.0, '1', True`
    - special value: `None`
    - division always gives float, unless `//` (div)
        - find remainder using mod operator `%`
    - using `type()`
- flow control:
    - comparison operators: `==`, `!=`, `>`, `<`, `>=`, `<=`
        - compare multiple values at once: `a < b < c...`
    - boolean logic with `and`, `or`, `not`
    - `if` statements, each clause on a separate line
        - `if a == 1:`
        - `elif`, `else`
        - compact one-line version:
            - `a =` val1 `if` condition `else` val2
    - shortcut: assign one of two values based on truth test of first value
        - `a = val1 or val2`
            - assign `val1` if `bool(val1)` evaluates to True, otherwise assign `val2`
    - `for` loops
        - `for i in range(10):`
        - `range(n)` generates values 0 to n-1
            - "give me the first 10 integers"
            - better interpretation: "give me the integer values between fenceposts 0 to n"
            - Python is "0-based" like C, Matlab is "1-based"
            - this convention is useful later for something called "slicing"
        - `range(1, n)` generates values 1 to n-1
        - `range(3, n, 2)` generates values 3 to n-1 in steps of 2
        - `range(10, n, -1)` generates values 10 to n+1 in steps of -1
        - put `range()` in `list()` to quickly see what values it will generate:
            - `list(range(10))`
        - `break`, `continue`
    - `while` loops
        - `while a > 1:`
        - same as `for` loops, except you manually increment your variable as you like
    - indentation is used to define blocks

        - indent with tabs or spaces, but spaces are better
        - 4 spaces per indentation level, check editor settings
    - paste multiline code from editor directly into python interpreter