Command line and Python basics

Command line/Terminal/Console

- command line is powerful - can be dangerous and intimidating, but worth it!
- most people use only a handful of basic commands
- bash command line basics, assume git bash installed on windows
  - bash = "Bourne-Again SHell"
  - bash prompt ends with `$` - this is where you can type a command
  - critical commands:
    - `pwd` - print working (i.e., current) directory
    - `cd` - change working directory
    - `ls` - list directory contents, defaults to current dir
      - `ls /some/path` - list contents of some other dir
  - `COMMAND --help` for help, sometimes also `man COMMAND` for more detailed "manual"
  - up/down arrow keys to access recently used commands
  - specifying paths:
    - `/` - filesystem root, like the base of a tree
    - `.` - current directory
    - `..` - parent directory
    - `~` - home folder
    - `-` - last used directory, i.e. `cd -` changes to last directory
  - others commonly used:
    - `mkdir` - make directory
    - `touch` - create an empty file, or update last access time of existing file
    - `mv` - move files/folders - a rename is just a move from old name to new name
    - `cp` - copy files/folders
    - `rm` - remove files/folders - dangerous! permanently deletes without confirmation
    - many commands accept `-v` (verbose) flag: prints out confirmation of what was done
    - `cat` - concatenate file(s)
      - quickly view file contents using `cat filename`
  - save text output of a command to a 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`

Exercises

1. Launch a terminal, `cd` to your home `~` or `~/Desktop` and list its contents with `ls`
2. Make a new directory called `tmp`. Check that it shows up when you re-list the contents of the current directory.
3. `cd` to your new `tmp`. Use `pwd` to ensure you're in the right folder
4. Use `touch` to make an empty file called `test.txt`. Now re-list the contents of the current directory. Can you see the new file?
5. Rename `test.txt` to `empty.txt`

6. Make another file called `test2.txt` using `cat >`. Punch in a few lines of text, then exit
7. Display the contents of `test2.txt` with `cat`
8. Copy `test2.txt` to `test3.txt`, and remove `test2.txt`
9. Save a **detailed** listing of the current directory to a file called `tmp_list.txt`
10. `cd` back to the parent directory, list the contents of your `tmp`
11. Copy `tmp` to `tmp2`. Need `--help`?
12. Remove both `tmp` and `tmp2`. Make it verbose

Python basics

- 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!')` - print message to screen
  - `input('hello? ')` - queries for user input
- commands can be saved into a `.py` (plain text) file, then run from the command line
  - need to use a plain text editor - http://geany.org is my favourite, but notepad in windows or TextEdit on mac (in plain text mode) are OK
  - make a 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`
    - `int`: counting numbers; `float`: decimal numbers; `str`: text; `bool`: logic

        - literals: `1, 1.0, '1', True`
        - types are also functions that convert input to that type, e.g. `float(1)` gives `1.0`
    - special value: `None`
    - division always gives float, unless `//` (div): `4 / 2` gives `2.0`, `4 // 2` gives `2`
        - find remainder using mod operator `%`
    - use `type()` to determine the type of something
- flow control:
    - comparison operators: `==`, `!=`, `>`, `<`, `>=`, `<=`
        - compare multiple values at once: `1 < 2 < 3`
    - boolean logic with `and`, `or`, `not`
    - `if` statements, each clause on a separate line

```
if a == 1:
    print('a is 1')
elif a == 2:
    print('a is 2')
else:
    print('a is something else')
```

        - indentation (4 spaces) and colons `:` are important
        - compact one-line version:
            - `a =` val1 `if` condition `else` val2
            - e.g. `msg = 'yes' if a == 1 else 'no'`
    - 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`
            - `a = 0 or 5` vs. `a = 1 or 5`
    - `for` loops

```
for i in range(5):
    print('hello!')
    print('goodbye!')
```

        - again indentation (4 spaces) and colons `:` are important
        - `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` - immediately exits for loop, `continue` skips to the next value of the iterator (in this case, `i`)
    - `while` loops

```
a = 0
while a < 5:
    print('hello!')
    print('goodbye!')
    a += 1
```

        - similar to `for` loops, except you manually increment your iterator as you like
    - indentation is used to define blocks inside `if`, `for` and `while` statements, and later as well when defining your own functions using `def`
        - indent with tabs or spaces, but spaces are *strongly* preferred
        - 4 spaces per indentation level, check editor settings
    - paste multiline code from editor directly into python interpreter

Exercises

1. Launch `python`. Do some math. Calulate `2 + 2` and save it to a variable called `genius`. Now print out the result in `genius`.
2. use a `for` loop to print out integers 0 to 9.
3. Exit `python`. Use a text editor to save your code in 2. to a script called `basics.py`. Run it by typing `python basics.py` at the command line. Does it work?
4. Modify the script to print out the square of those integers. Test it!
5. Modify the script to also print out the sum of the integers
6. Modify the script to print out the square root of those integers
7. Restore the script as it was in 2. Modify it to print the word `seven` after printing out the integer `7`
8. Modify it to **also** print out the word `three` after printing out the integer `3`
9. Rewrite the script so that it prints the messages `1 is odd`, `2 is even`, `3 is odd` all the way up to `10 is even`
10. Modify it so that it **doesn't** print the message `7 is odd`
11. Reverse the order of the messages