

Python basics 2

- review

- running python scripts from command line: `python hello.py`
- `==` VS `=`
- in addition to `else` clause, can also use `elif` clause in `if` blocks:

```
if something:
    x = 1
elif somethingelse:
    x = 2
else:
    x = 3
```

- while loops
- write a script with a loop, either a `for` loop or a `while` loop, that prints "hello" 3 times, then prints "hello?" 3 times, then prints "goodbye!" once

- plain text editors

- key features:
 - plain text format: `.txt`, `.py`, etc.
 - fixed-width font
 - syntax highlighting
 - line numbering
- linux: geany, gedit, mousepad
- windows: geany, notepad++, ultraedit, textpad
- mac: geany, atom, sublime, xcode
- command line editor: nano, even cat
- cross-platform Python IDEs: pycharm, spyder
 - downside:
 - more complicated than simple text editor
 - maybe don't work as well for other types of languages or text files

- coding style: why does it matter? easier to read, understand, debug

- try reading a book without paragraphs...
- a few tips from coding style guide
 - PEP 8: <https://www.python.org/dev/peps/pep-0008>
 - variable assignment: always leave a space on either side of `=`
- comments, docstrings
 - single line: `#`
 - multiline: `"""..."""` or `'''...'''`
 - why comment? what makes a good comment?
 - what happens if you change code without updating comment? confusion!
 - another form of commenting: choose descriptive variable names, use them consistently

- strings

- string formatting, operations and functions
 - combine strings with `+`

- duplicate strings with `*`
- whitespace characters: `\n` and `\t`
- `%` string replacement operator
 - what else does `%` do? how does python know whether to use it as a string replacement operator or as mod operator?
 - format strings act as placeholders: `%s`, `%d`, `%f`, `%g`
- `.split()`, `.replace()`, `.strip()`, `.upper()`, `.lower()`
- `s = 'abcd'`
 - indexing: `s[0]` returns `'a'`; `s[1]` returns `'b'`
 - slicing: `s[0:1]` returns `'a'`, `s[0:2]` returns `'ab'`
- are there other string methods? how to discover them without doing a web search?
 - `dir(s)`
- IPython as replacement for plain Python interpreter
 - `something?` for help, `something??` for help plus source code, if available
 - command completion
 - command history with up/down keys
 - attribute exploration via dot notation, followed by `?`
 - referring to previous outputs and inputs with `_` and `_i`
 - view all local variables with `whos`
 - paste multiline code from editor directly into IPython
- built-ins/keywords
 - listed in `help()`, `keywords`
 - can't be used as variable names
- calling functions
 - what is a function?
 - what are function arguments?
 - remind of `math` module, how do you gain access to it?
 - `help(function)` or `function?` to get call signature
 - functions can have a fixed or variable number of arguments, some of which are optional
 - positional arguments: `function(a, b) != function(b, a)`
 - keyword arguments `function(a=value1, b=value2)` can be specified in any order
 - use `datetime.date()` to demonstrate
 - `datetime.date(2005, 5, 2)`
 - `datetime.date(month=5, year=2005, day=2)`
- defining your own functions:

```
def add(x, y):
    """This is my function. It adds x and y"""
    result = x + y
    return result
```

- body is indented, like a for or while loop
- documentation string

- `return` a value, or multiple values separated by comma
 - can define positional and keyword arguments:
 - `def add(x, y):`
 - `def add(x, y, z=0):`
 - variable scope/namespaces
- optional: errors and debugging
 - `assert` allows you to quickly check assumptions that might not always hold
 - typical errors: `SyntaxError`, `NameError`, `TypeError`, `ValueError`, `IndexError`, `KeyError`, `RuntimeError`, `AttributeError`, `ZeroDivisionError`
 - set a breakpoint and "drop into debugger" with: `import pdb; pdb.set_trace()`
 - debugger commands: `l`, `w`, `s`, `n`
 - `try`, `except` blocks to catch specific types of errors and deal with them
 - `raise` your own errors to stop execution and inform the user of something