NOTES ON RUNNING PYTHON CODE

ERIC MARTIN

Part 1. Setting things up

1. Installing python if necessary

The School has python 3.2.3 installed.

On personal computers with no version of python 3 installed, get the latest version (python 3.4.3) for the appropriate platform from

https://www.python.org

Mac users: drag the IDLE.app icon in /Applications/Python 3.4 to the dock.

2. Making `python` and `idle` the right commands

In the home directory of your CSE account, create or edit (with an editor such as vi or nedit) the file `.profile` and add the lines

```
alias python='/usr/bin/python3'
alias idle='/usr/bin/idle3'
```

Mac users: in your home directory, create or edit (with an editor such as vi or TextEdit) the file `.profile` and add the lines

```
alias python='/usr/local/bin/python3'
alias idle='/usr/local/bin/idle3'
```

3. Permanently adding directories to `sys.path`

`sys.path` is the list of directories where python looks for modules (files). On a School machine, it is

```
['', '/usr/lib/python3.2', '/usr/lib/python3.2/plat-linux2',
 '/usr/lib/python3.2/lib-dynload', '/usr/local/lib/python3.2/dist-packages',
 '/usr/lib/python3/dist-packages']
```

as can be found out by interpreting from the python prompt

```
from sys import path
path
```

The first directory in this list, `''`, is the working directory.

To add directories to this list, create a sequence of new directories by executing in an xterm window, in the home directory, the Unix command

```
mkdir -p .local/lib/python3.2/site-packages
```

To add the home directory to `sys.path`,

- run in the home directory the command `pwd`,
- create in `~/.local/lib/python3.2/site-packages` the file `my_path.pth`, and
- add to this file the output of that command.

Date: Session 1, 2015.

If you were me, that would be

```
/import/kamen/1/emartin
```

Other directories can be added, one per line. For instance, if you were me and had created in your home directory the sequence of directories `Documents/Python/Code`, then you could also add to `my_path.pth` the line

```
/import/kamen/1/emartin/Documents/Python/Code
```

to make it part of `sys.path`.

Mac Users: Same procedure but replacing `~/.local/lib/python3.2/site-packages` by

`~/Library/Python/3.4/lib/python/site-packages`

Part 2. **Using Idle**

For the following, if you were me, you would have

- `hello_world_v1.py`,
- `hello_world_v2.py`,
- `greet.py`, and
- `greet_and_say_bye.py`

saved in `~/Documents/COMP9021/Lectures/Lecture_1`, and we assume that `~/Documents` is part of `sys.path`. For Mac Users, this could just be because Idle has been started by double clicking on its icon.

If

- neither `~/Documents/COMP9021`
- nor `~/Documents/COMP9021/Lectures`
- nor `~/Documents/COMP9021/Lectures/Lecture_1`

had been added to `sys.path`, then `COMP9021/Lectures/Lecture_1` would be the "missing part" of the path for python to be able to locate those files, unless `~/Documents/COMP9021/Lectures/Lecture_1` is the working directory.

4. At the prompt

4.1. **Executing statements.** Interpret

```
print('Hello world!')
```

4.2. **Defining functions and calling them.** Define a function as

```
def hello_world():
    print('Hello world!')
```

and call it by executing

```
hello_world()
```

5. Opening a file and selecting Run Module from the menu

5.1. **Executing statements.** Use the file `hello_v1.py` whose contents is

```
print('Hello world!')
```

5.2. **Calling functions.** Use the file `hello_v2.py` whose contents is

```
def hello_world():
    print('Hello world!')
```

and call the function from the Idle prompt by executing

```
hello_world()
```

6. Importing or reimporting a module containing the statements to execute

6.1. **Importing the module.** In case Idle has been launched from the directory where `hello_v1.py` is stored (probably by executing the `idle` Unix command in an xterm widow, in that directory), execute

```
import hello_world_v1
```

and in case Idle has been launched from another directory (maybe by clicking on the Idle icon), execute

```
import COMP9021.Lectures.Lecture_1.hello_world_v1
```

6.2. **Reimporting the module.** Repeating the `import` statement will not reevaluate the statements. Executing

```
from importlib import reload
```

allows every call to

```
reload(hello_world_v1)
```

or to

```
reload(COMP9021.Lectures.Lecture_1.hello_world_v1)
```

to reevaluate the statements.

7. Importing a module containing the functions to call or importing the functions themselves

7.1. **Importing the module.** In case Idle has been launched from the directory where `hello_v2.py` is stored, execute

```
import hello_world_v2
```

and in case Idle has been launched from another directory, execute

```
import COMP9021.Lectures.Lecture_1.hello_world_v2
```

and call the function by executing

```
hello_world_v2.hello_world()
```

or

```
COMP9021.Lectures.Lecture_1.hello_world_v2.hello_world()
```

respectively.

7.2. **Importing the functions.** In case Idle has been launched from the directory where `hello_v2.py` is stored, execute

```
from hello_world_v2 import hello_world
```

and in case Idle has been launched from another directory, execute

```
from COMP9021.Lectures.Lecture_1.hello_world_v2 import hello_world
```

and call the function by executing

```
hello_world()
```

8. Calling functions but not when importing

Use the file `greet.py` whose contents is

```
def hello(you):
    print('Hello ' + you + '!')

if __name__ == '__main__':
    hello('world')
    hello('Jane')
    hello('Michael')
```

and select `Run Module` from the menu.

Note that executing

```
import greet
```

does not produce any output.

Note that opening the file `greet_and_say_bye.py` whose contents is

```
import COMP9021.Lectures.Lecture_1.greet

COMP9021.Lectures.Lecture_1.greet.hello('universe')
print('Bye now...')
```

and selecting `Run Module` from the menu or executing

```
import greet_and_say_bye
```

at the prompt does not output

```
Hello world!
Hello Jane!
Hello Michael!
```

either.

In both cases, the test `__name__ == '__main__'` fails because `__name__` is equal to `'greet'`.

This technique is commonly used to easily test the code of one module (such as `hello`) meant to be utilised in other modules (such as `greet_and_say_bye`).

Part 3. Using an xterm window

A new method: execute the Unix command `python hello_world_v1.py`.

For the rest, exactly as when using Idle, except for Section 5 and the parts of Section 8 that are specific to Idle, but executing the Unix `python` command and entering statements from the python prompt rather than from the Idle prompt.

To quit python, press Control D.