Python: getting started

Working environment, programming basics, write and run a program

Why python?

- Python is an extremely useful programming language for getting science done. It is fast enough and simple enough for most tasks – although compiled languages like C++, Fortran90, and Java are faster.
- Python is often called a "glue language" because it is adept at working in and across many computing platforms, and integrating programs running in other languages.
- Like MATLAB, it has tight integration of programs, command line interactivity, and publication-quality graphics.
- Unlike MATLAB, it is open source, so there are toolboxes (called "modules" in python) created by many different groups for different tasks. This can be good because you can find a module to do almost anything, and it can be bad because quality control and conventions are not as strict as they are at MathWorks (the company that makes MATLAB).
- Python is free and you can install it easily on any machine.

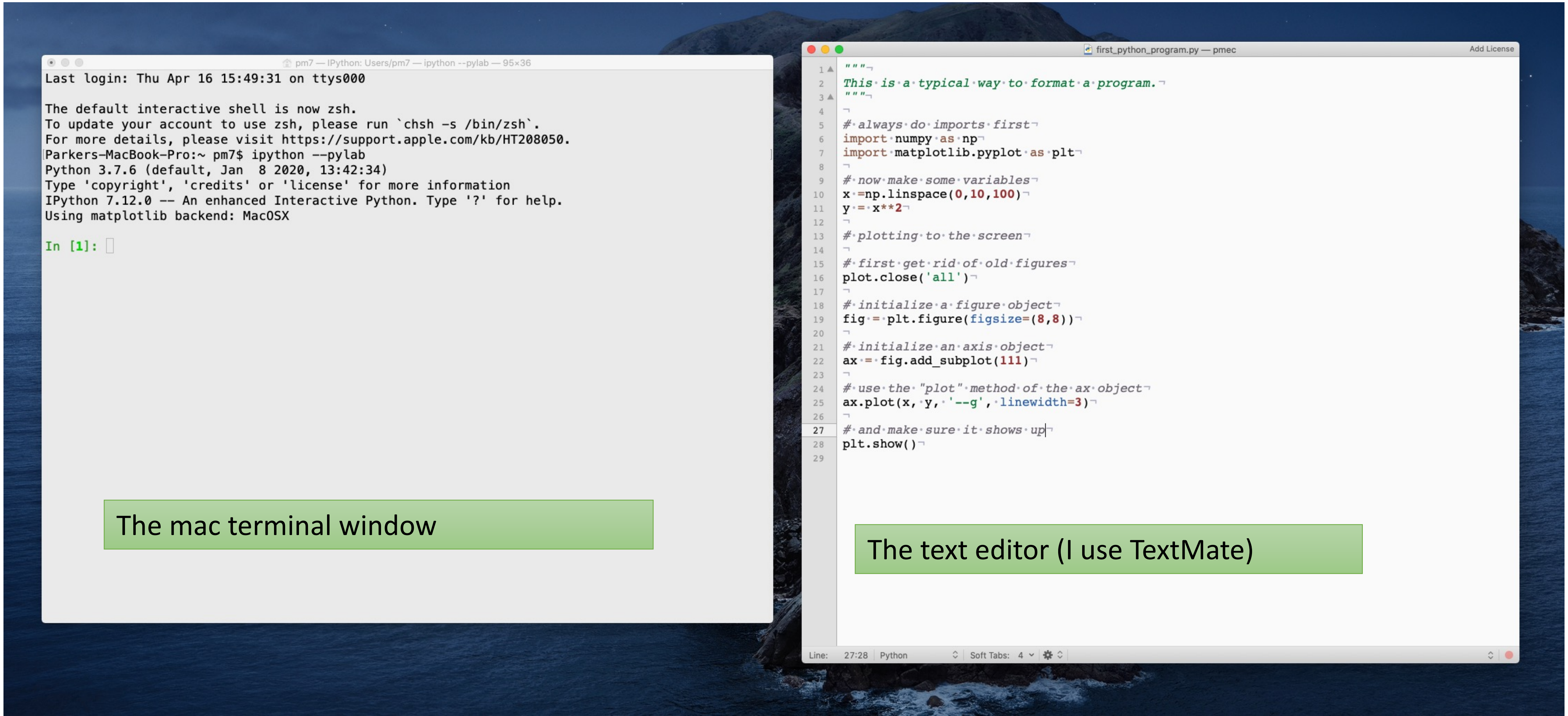
Your work environment

I try to keep my python environment as simple as possible:

1. Launch ipython in a linux terminal window by typing:
   ```
   ipython --pylab
   ```
2. Keep a text editor next to it for editing whatever program(s) you are running
3. Plots come up in their own window.

NOTE: if you are working on a remote machine you won't have graphics windows that pop up, so just launch ipython by typing `ipython` (no --pylab).

My python working environment


pm7 — IPython: Users/pm7 — ipython --pylab — 95×36
first_python_program.py — pmec
Add License
Line: 27:28
Python
Soft Tabs: 4

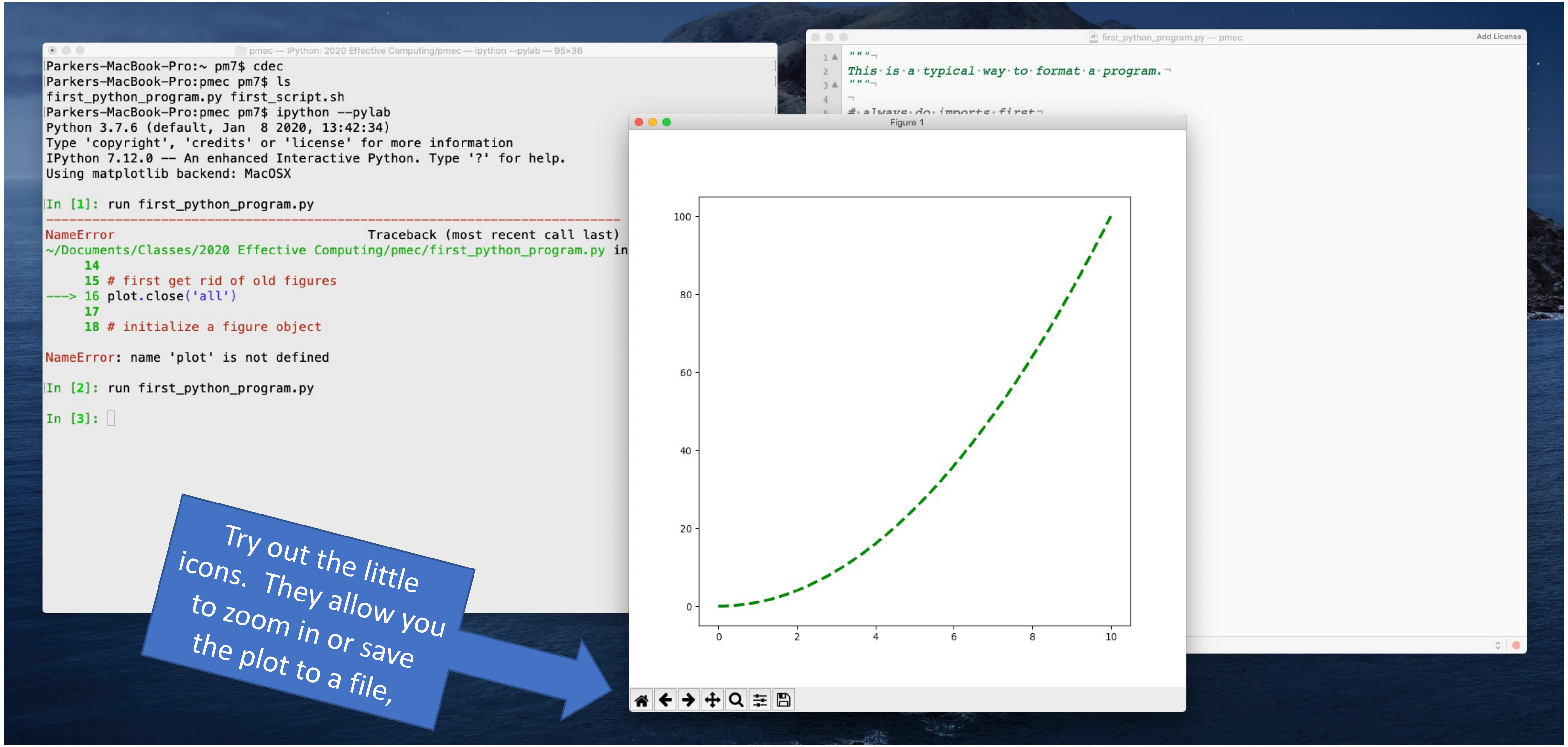
```
Last login: Thu Apr 16 15:49:31 on ttys000

The default interactive shell is now zsh.
To update your account to use zsh, please run `chsh -s /bin/zsh`.
For more details, please visit https://support.apple.com/kb/HT208050.
Parkers-MacBook-Pro:~ pm7$ ipython --pylab
Python 3.7.6 (default, Jan  8 2020, 13:42:34)
Type 'copyright', 'credits' or 'license' for more information
IPython 7.12.0 -- An enhanced Interactive Python. Type '?' for help.
Using matplotlib backend: MacOSX

In [1]:
```

The mac terminal window

```python
"""
This is a typical way to format a program.
"""

# always do imports first
import numpy as np
import matplotlib.pyplot as plt

# now make some variables
x =np.linspace(0,10,100)
y = x**2

# plotting to the screen

# first get rid of old figures
plot.close('all')

# initialize a figure object
fig = plt.figure(figsize=(8,8))

# initialize an axis object
ax = fig.add_subplot(111)

# use the "plot" method of the ax object
ax.plot(x, y, '--g', linewidth=3)

# and make sure it shows up
plt.show()
```

The text editor (I use TextMate)

A plot window comes up when I run the code

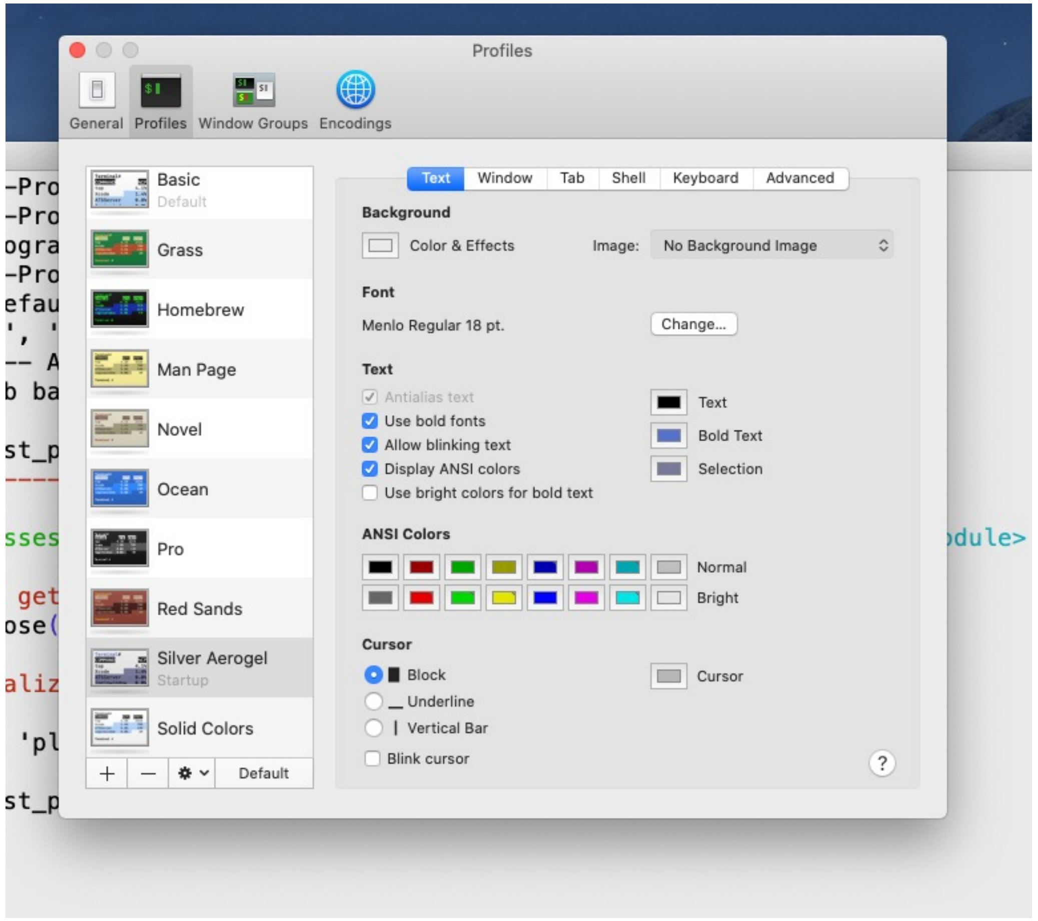

pmec — IPython: 2020 Effective Computing/pmec — ipython --pylab — 95×36
first_python_program.py — pmec
Add License
Figure 1
100
80
60
40
20
0
0
2
4
6
8
10

```
Parkers-MacBook-Pro:~ pm7$ cdec
Parkers-MacBook-Pro:pmec pm7$ ls
first_python_program.py first_script.sh
Parkers-MacBook-Pro:pmec pm7$ ipython --pylab
Python 3.7.6 (default, Jan  8 2020, 13:42:34)
Type 'copyright', 'credits' or 'license' for more information
IPython 7.12.0 -- An enhanced Interactive Python. Type '?' for help.
Using matplotlib backend: MacOSX

In [1]: run first_python_program.py
---------------------------------------------------------------------------
NameError                                 Traceback (most recent call last)
~/Documents/Classes/2020 Effective Computing/pmec/first_python_program.py in
     14
     15 # first get rid of old figures
---> 16 plot.close('all')
     17
     18 # initialize a figure object

NameError: name 'plot' is not defined

In [2]: run first_python_program.py

In [3]:
```

```
"""
This is a typical way to format a program.
"""

# always do imports first
```

Try out the little icons. They allow you to zoom in or save the plot to a file,

You can make adjustments to how your terminal window looks (Terminal => Preferences)



Basic
Default
Grass
Homebrew
Man Page
Novel
Ocean
Pro
Red Sands
Silver Aerogel
Startup
Solid Colors

Background

Color & Effects   Image: No Background Image

Font

Menlo Regular 18 pt.   Change...

Text

- ✓ Antialias text
- ✓ Use bold fonts
- ✓ Allow blinking text
- ✓ Display ANSI colors
- ☐ Use bright colors for bold text

Text
Bold Text
Selection

ANSI Colors

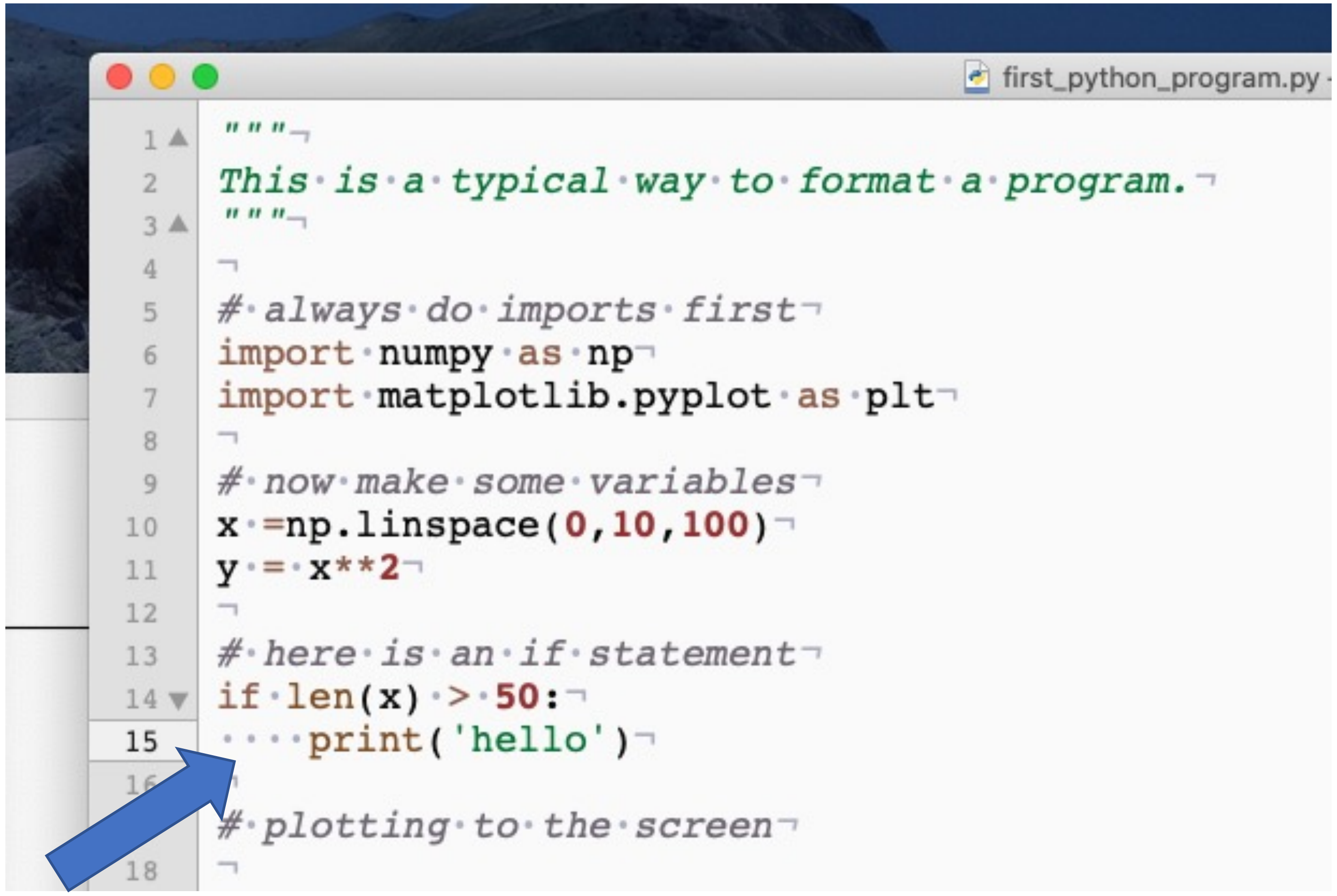
Normal
Bright

Cursor

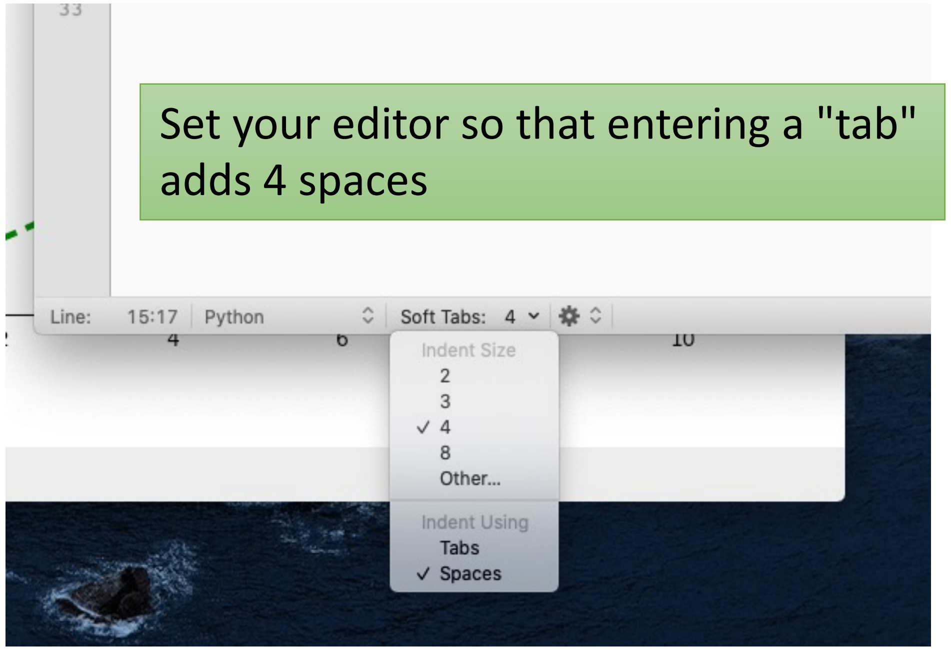
- ● Block
- ○ Underline
- ○ Vertical Bar
- ☐ Blink cursor

Cursor

Spaces matter in python so you want to have them show up in your text editor


first_python_program.py

```
"""¬
This·is·a·typical·way·to·format·a·program.¬
"""¬
¬
#·always·do·imports·first¬
import·numpy·as·np¬
import·matplotlib.pyplot·as·plt¬
¬
#·now·make·some·variables¬
x·=np.linspace(0,10,100)¬
y·=·x**2¬
¬
#·here·is·an·if·statement¬
if·len(x)·>·50:¬
····print('hello')¬
¬
#·plotting·to·the·screen¬
¬
```


Line: 15:17 Python Soft Tabs: 4

Set your editor so that entering a "tab" adds 4 spaces

ipython

You can launch python in several ways from the linux command line.

To just run a program from the bash shell (or cron) type:

```
        python [program name].py
```

or if it is a long job on a remote machine and you want to logout:

```
        python [program name].py > log.txt &
```

Alternatively you can enter the ipython environment by typing:

```
        ipython --pylab
```

and then…

```
        run [program name].py
```

To escape from ipython back to the bash shell type:

```
        quit
```

ipython, continued...

- ipython is the "interactive python" environment, and it is what you should always use. The "--pylab" option silently adds the numpy and matplotlib modules to your interactive workspace and sets the graphical backend.
- The ipython environment allows you to quickly see information about the variables you are using in your program, and it has some nice help features, and tab-completion when you type.

Useful commands in ipython for getting information (try them!)

- "whos" gives information about all the variables in your workspace
- Python is an "object-oriented" language, which is just a fancy way of saying that <u>everything</u>, whether it be a variable, a function, a line on a plot, or a module, is an "**object**" that has associated **attributes** (facts about it) and **methods** (functions that can act on it).
- "[object]?" gives info about a variable or other object
- "help([object])" is another way of getting info about an object
- "dir([object])" gives a full listing of the attributes and methods available for a given object

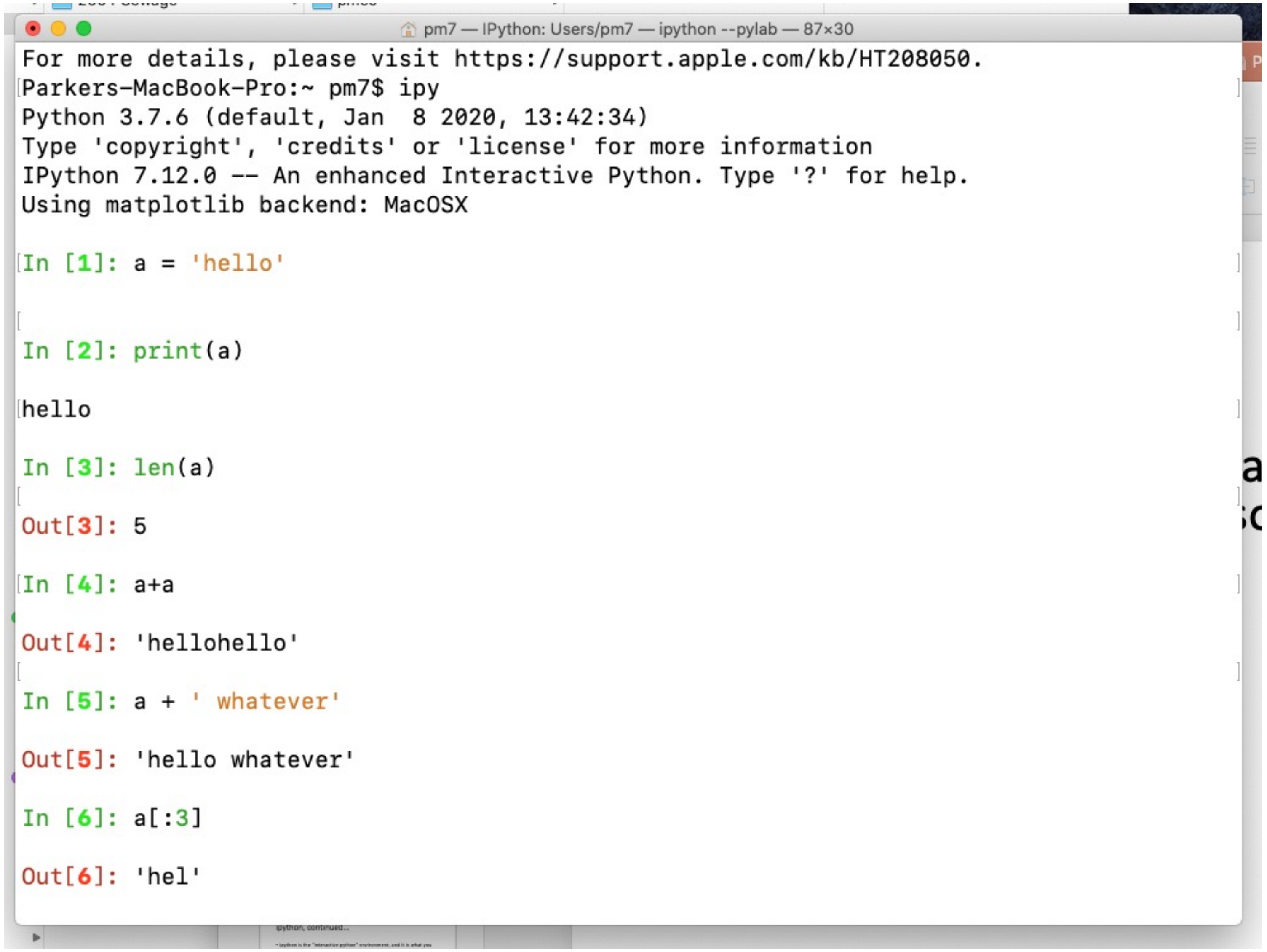
Resources

- Learning python programming is a long process - I learn new things all the time. This class will get you started, but you need to learn most of it on your own.
- Read a book. My favorite is: McKinney (2018) Python for Data Analysis, 2nd Edition. O'Reilly.
- Find a website with instructions. Here is a reasonable one: https://docs.python.org/3/tutorial/
- Ask someone, even me.
- Make a list of commands you want to memorize and keep editing it as you go along.

Now, on to python language basics, starting with some types of variables...

- **Strings**
- Strings are just lists of characters. You often use them to define input and output pathnames, so it is good to learn how to manipulate them. You make them using **quote marks**: 'string'. (Single or double quotes both work.)
- Try typing the commands at the right yourself.
- Then try out the commands from the previous pages to get information on what you can do to strings (like whos and dir and ?)
- E.g., what does "a.title()" do? How about "a.find('h')"?
- "title" is an example of a "method" that string objects have. Methods always have () after them.
- To use a method just type:

```
[object].[method]() # note the dot!
```



pm7 — IPython: Users/pm7 — ipython --pylab — 87×30

```
For more details, please visit https://support.apple.com/kb/HT208050.
Parkers-MacBook-Pro:~ pm7$ ipy
Python 3.7.6 (default, Jan  8 2020, 13:42:34)
Type 'copyright', 'credits' or 'license' for more information
IPython 7.12.0 -- An enhanced Interactive Python. Type '?' for help.
Using matplotlib backend: MacOSX

In [1]: a = 'hello'


In [2]: print(a)

hello

In [3]: len(a)

Out[3]: 5

In [4]: a+a

Out[4]: 'hellohello'

In [5]: a + ' whatever'

Out[5]: 'hello whatever'

In [6]: a[:3]

Out[6]: 'hel'
```

Indexing, with a = 'hello'

- a[:3] => 'hel' is an example of indexing, or "slicing". The same indexing rules apply to strings, lists, numpy arrays and pandas DataFrames, so you need to know how it works.
- The first item is at index 0, so the index numbers of 'hello' would be 0,1,2,3,4
- Specify a range with a ":" and some integer indices for the range:
- **[start index]:[index *after* the last one you want to get]**
- a[1:4] => 'ell' (when I type "=>" that means "returns", it is not part of the command!)
- Note that you can always tell the ***length*** of what you get by the ***difference*** of the two indices (4 - 1 = 3)
- Omitting the start index means starting at 0.
- Omitting the end index means get through the last item, a[2:] => 'llo'
- You can also add a **step** with another colon (start:stop:step)
- a[::2] => 'hlo'
- Use a step of -1 to reverse the order
- a[::-1] => 'olleh'
- Using an ending index of -2 cuts off the last two letters:
- a[:-2] => 'hel'

Lists

- Lists are just ordered collections of objects. You form them by using **square brackets and commas**:
- my_list = ['hi', 99, 'hello']
- the same indexing rules apply:
- my_list[1] => 99
- You concatenate lists using the plus sign
- my_list + ['NNN'] => ['hi', 99, 'hello', 'NNN']
- Note that the thing I added, ['NNN'], had to also be a list, so it needed square brackets of its own.
- "Tuples" are another kind of list that you form using **parentheses and commas**. They act just like lists but are meant for information that you don't want to be able to change.
- my_tup = ('hi', 88, 6)

Dictionaries ("dict")

- A dictionary is a way of associating two lists of information. You make them using curly brackets, colons, and commas:

```
my_dict = {'a':'Thing One', 'b': 'Thing Two', 'c':'Thing Three'}
```

- these are called "keys" and "values"
- Try typing my_dict.keys() or my_dict.values()
- Accessing one value using its key: my_dict['b'] => 'Thing Two'
- or more generally:
- my_dict[key] => value associated with key
- You can make a dict quickly out of two equal-length lists:
- my_dict = dict(zip(list1, list2))
- the function dict() is another way of creating a dictionary.

Iteration

- You can "iterate" over any list using a colon and then a tab (4 spaces!) in front of the subsequent lines.

```
for item in my_list:
    print(item)
```

- The "range" function is a quick way to make a range of integers.

```
for item in range(10):
    print(item)
```

- You can iterate over the keys in a dict using "for item in my_dict.keys():".
- Python lingo: anything you can iterate over is called "iterable". You can even iterate over the characters in a string, or the numbers in a numpy vector, or the indices in a pandas DataFrame.
- The word "item" in the examples above is not special, but the words "for" and "in" are part of the programming language (try using "in" as a variable name and see what happens).

if-elif-else and "Booleans" (True, False)

- Another variable type in python is a Boolean, which is either "True" or "False" (you have to type them capitalized).
- The usual comparison operations will return Booleans:

```
for x in range(5):
    if x == 0:
        print(x)
    elif x >=3:
        print('Big x = ' + str(x))
    else:
        print('I do not care')
```

- Each if, elif, else must be followed by one or more code lines. You can use the command "pass" as a way of doing nothing.
- Note that the indenting is essential. Python does not end things explicitly with and "end" or "endif". Instead you just go back to the original indenting. ipython will tell you when something is amiss.

while loop

- "while" is another common control structure

```
counter = 1
while counter < 10:
    print(counter)
    counter += 1 # same as counter = counter + 1
```

- Be careful to make sure your while loop has a reason to stop!

common imports

- In order to use the full range of python functions, you have to import "modules". Most of these came with your anaconda installation, and you imported a few more using conda or pip.
- Typical import statements:

```
import numpy as np # like matlab operators
import matplotlib.pyplot as plt # plotting
```

- The "as" means that when you use a function in a module you don't have to type as much: `x = np.linspace(0,10,100)`

functions

- If you are performing the same operation multiple times in a piece of code it should be done by defining a function (must be defined before it is first used)

```
def a_plus_b(a,b):
    c = a + b
    return c
```

- The variable names used in the function are limited in their "scope" meaning they only have meaning inside the function. You could use the same variable names in the code that calls the function and they would not mess each other up.

Structure of a complete program

- You would save this as a text file called "test0.py" and run in ipython as "run test0" or "run test0.py".

```
"""
This code is to test calling a function.
"""
import numpy as np # do imports first

def my_fun(x): # function to compute the sine of the square
    y = np.sin(x**2)
    return y

for xx in np.linspace(0, 2*np.pi, 12):
    yy = my_fun(xx)
    print('x = %0.2f, y = %0.2f' % (xx, yy))

# that's it!
```