Coding overview

NSC3270 / NSC5270
Computational Neuroscience

Tu/Th 9:35-10:50am
Featheringill 129

Professor Thomas Palmeri
Professor Sean Polyn

Coding in NSC 3270 / 5270

- Regular HW assignments in which you either write your own code to solve some problem, or use and modify sample code provided by us.

- Getting everyone using the Python 3 coding environment.

- Many of the assignments will use Keras, a Python deep learning library. Keras runs on top of Tensorflow, an open source machine learning framework.

- Different students will have different computer configurations (e.g., OS X vs Windows) so the installation process will vary a bit from student to student.

Coding in NSC 3270 / 5270

- When you see “$”, that is shorthand for the Unix-style command prompt that you’ll have in a Linux or OS X environment.
- Don’t actually type “$”, that will give you an error! That just indicates that what follows is something you can type in a shell or at a command prompt.
- OS X: You’ll want to launch Terminal.app, which lives in Applications>Utilities
- Linux: You’ll already know what to do
- Windows: I don’t have access to a Windows machine but later we will talk about 2 apps that will give you access to a command prompt (Anaconda, and Wing IDE). There are others.

Software

- There are several required software packages you'll need to complete the assignments in this class. All are free & open source.

- Minimal requirements: Python 3 (with certain standard packages), Tensorflow, Keras, and a text editor (to write code).

- However, we recommend certain other (also free) software to help make the installation process and programming process easier for you.

Installing Python

- You can install Python 3 directly from the developers: https://www.python.org/downloads/

- However, we recommend you use Anaconda (A Python Data Science platform) to manage Python for you : https://www.anaconda.com

Installing Python

- Anaconda by default installs many of the dependencies you'll need (NumPy, Matplotlib), and has a straightforward way to install supported packages (Tensorflow) and community supported packages (Keras)

- If you know what you are doing re: software installation, you don't have to follow our advice. There are many ways to get your system configured properly!

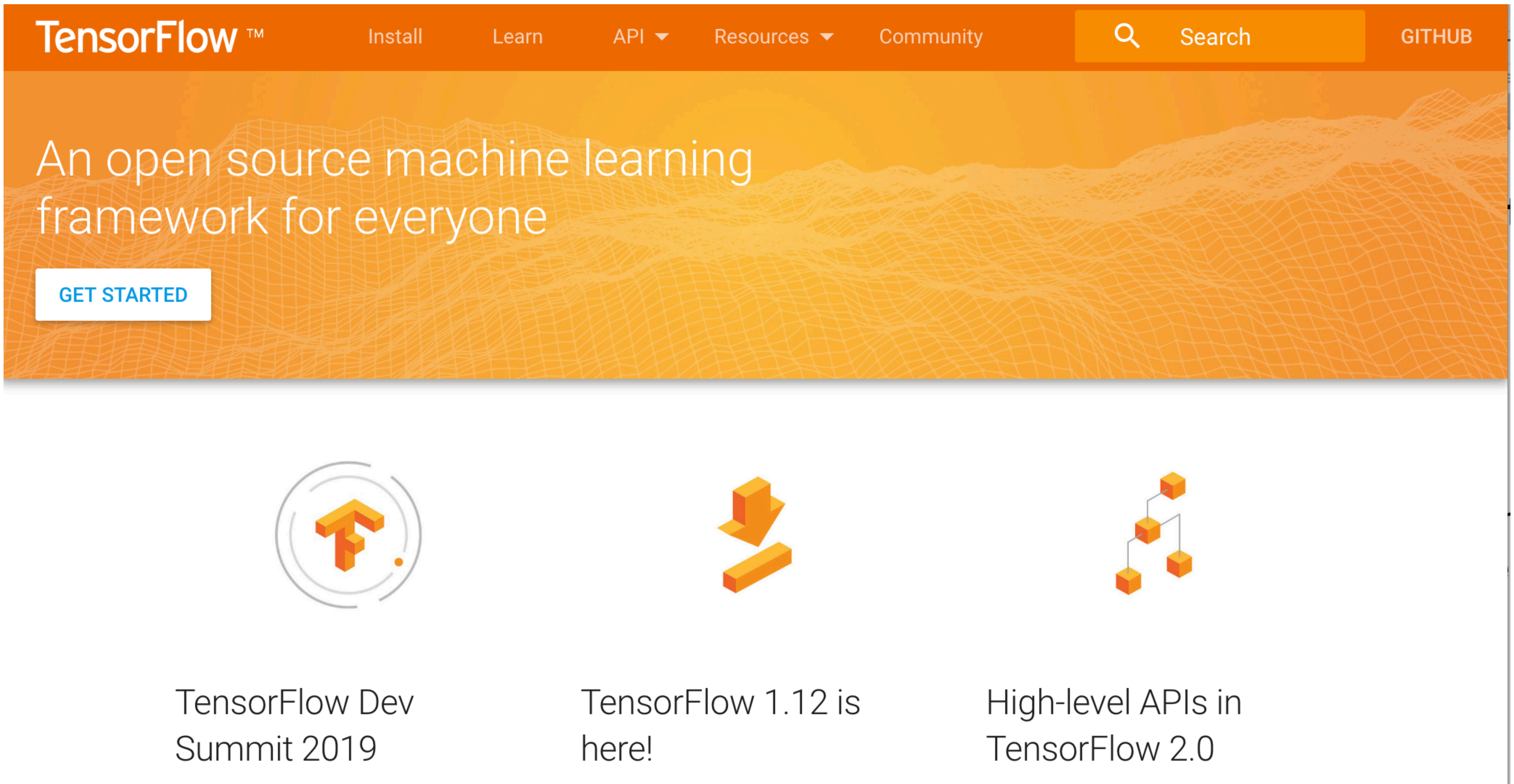
Python environments

- The latest version of Python is 3.7. However, TensorFlow currently only works with Python 3.6. If you want to have both 3.7 and 3.6 installed on your machine, the syllabus details how you can create an Anaconda environment for Python 3.6, which allows you to switch back and forth as needed. The relevant commands:

```
% this will make an environment named py36
$ conda create -n py36 python=3.6 anaconda
% this will let you check that it worked
$ conda info --envs
% this will activate the environment (mac/linux)
$ source activate py36
% activate (windows)
$ activate py36
```

Installing Tensorflow

- https://www.tensorflow.org/



Installing Tensorflow

- Tensorflow provides low-level machine learning code, Keras is the high-level library that uses Tensorflow to get things done.

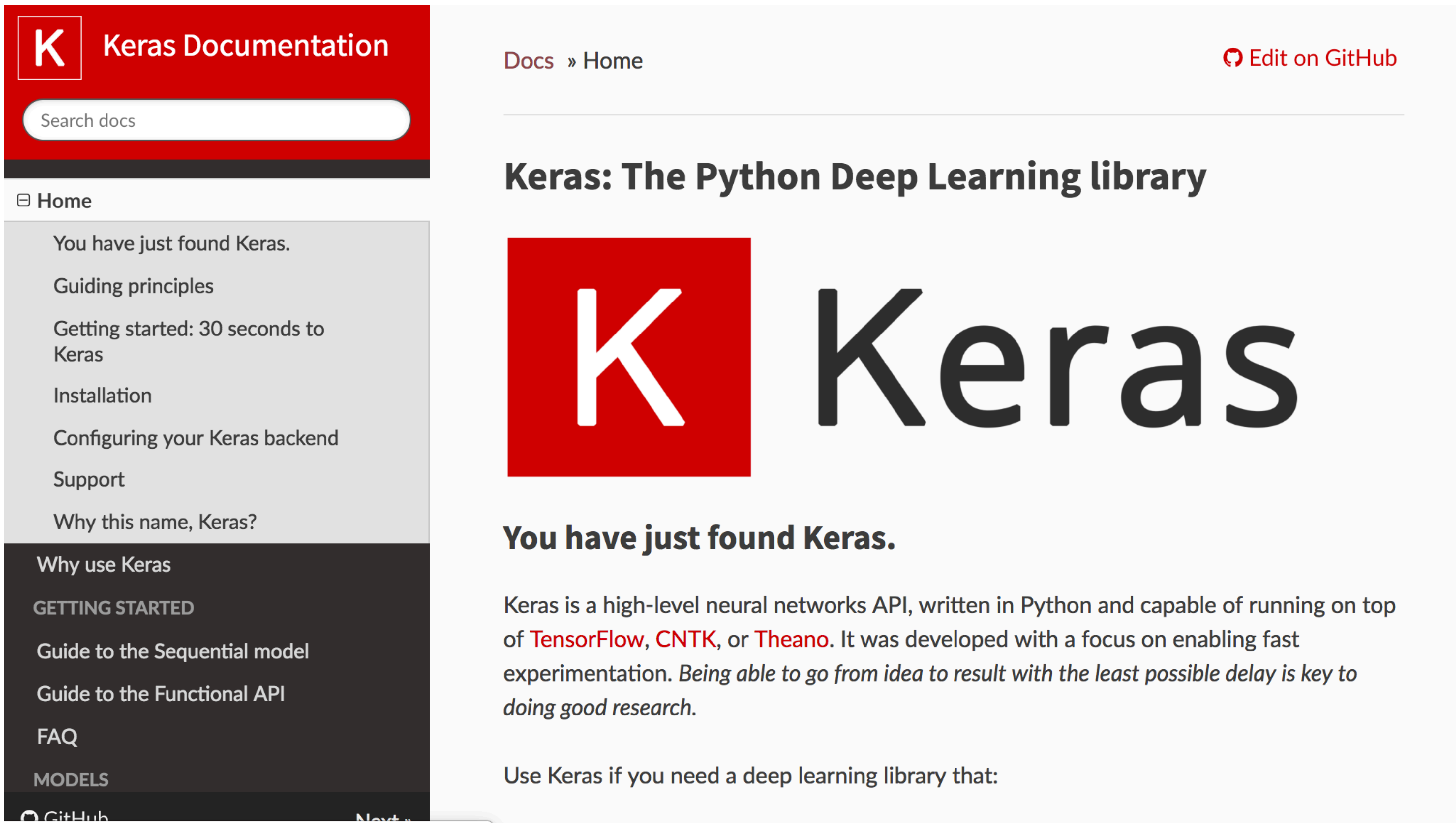
- If you are using Anaconda, you can use the Terminal (OS X) or Anaconda Prompt (Windows)

- `$ conda install tensorflow`

- This will make sure you have all the dependencies Tensorflow is expecting.

Installing Keras / Tensorflow

- https://keras.io/#installation



Installing Keras / Tensorflow

- Keras isn't directly supported by Anaconda, but conda-forge is a community supported package manager that works with conda to install Keras.

- https://conda-forge.org/

- The following worked for me:

- `$ conda install -c conda-forge keras`

WING IDE (Integrated Development Environment)

- All you technically need to work in Python is the Python software itself and a text editor.
- However, there are many IDEs out there that greatly facilitate code development and debugging.
- Wing IDE is an advanced development environment which provides free licenses for educational use.
- https://wingware.com/
- The license code is posted in the syllabus, you'll need this to get it working

WING IDE (Integrated Development Environment)

- The license code will allow you to use Wing Pro.

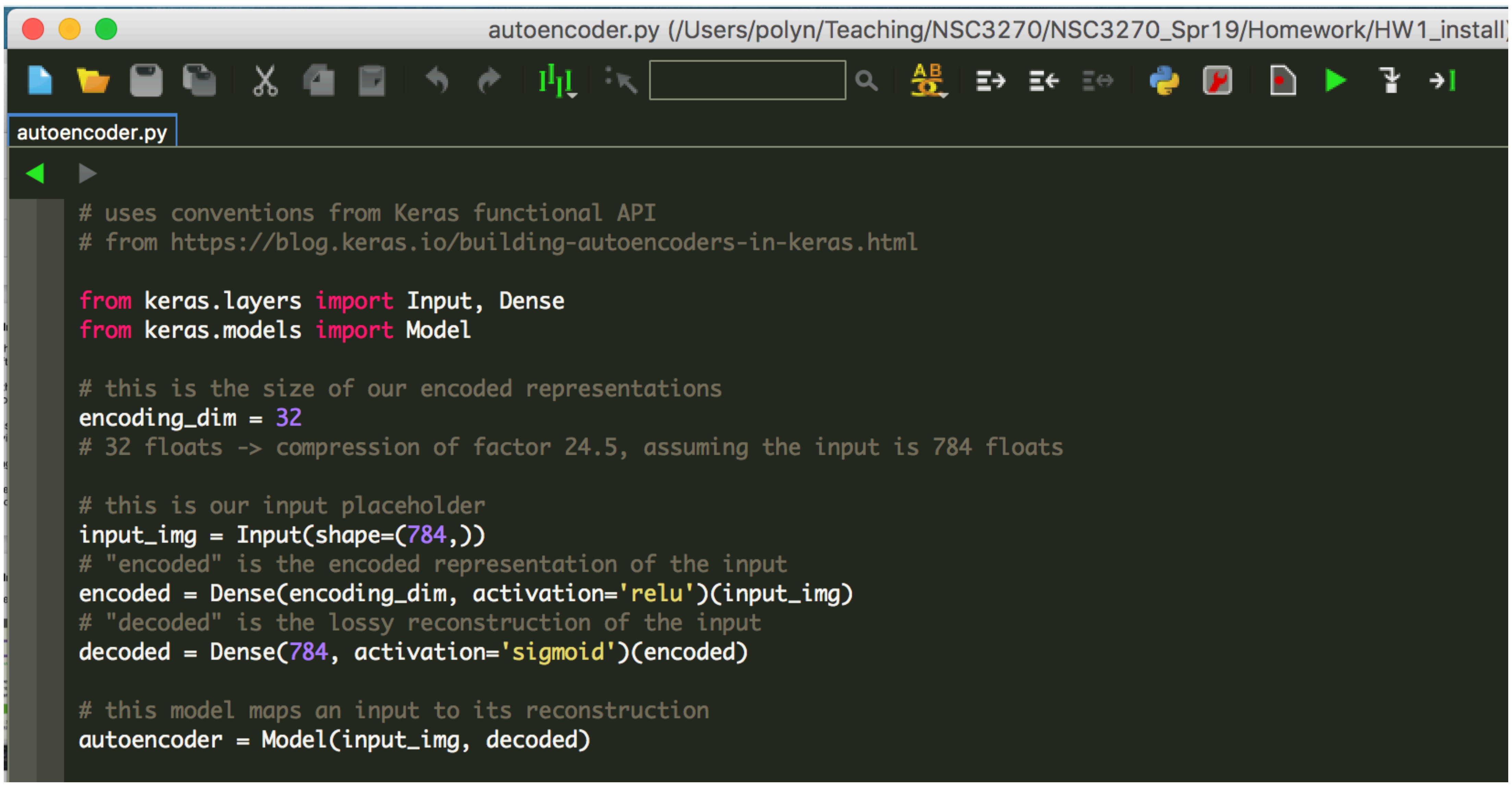
HOMEWORK #1

- Get all of this installed and run the “autoencoder.py” program that is on Brightspace. We won’t go too far into what this code is doing today, but if it works it means you have the key packages installed and configured correctly!

- To complete the homework, take a screen shot showing that it ran successfully, and submit the screenshot through Brightspace.

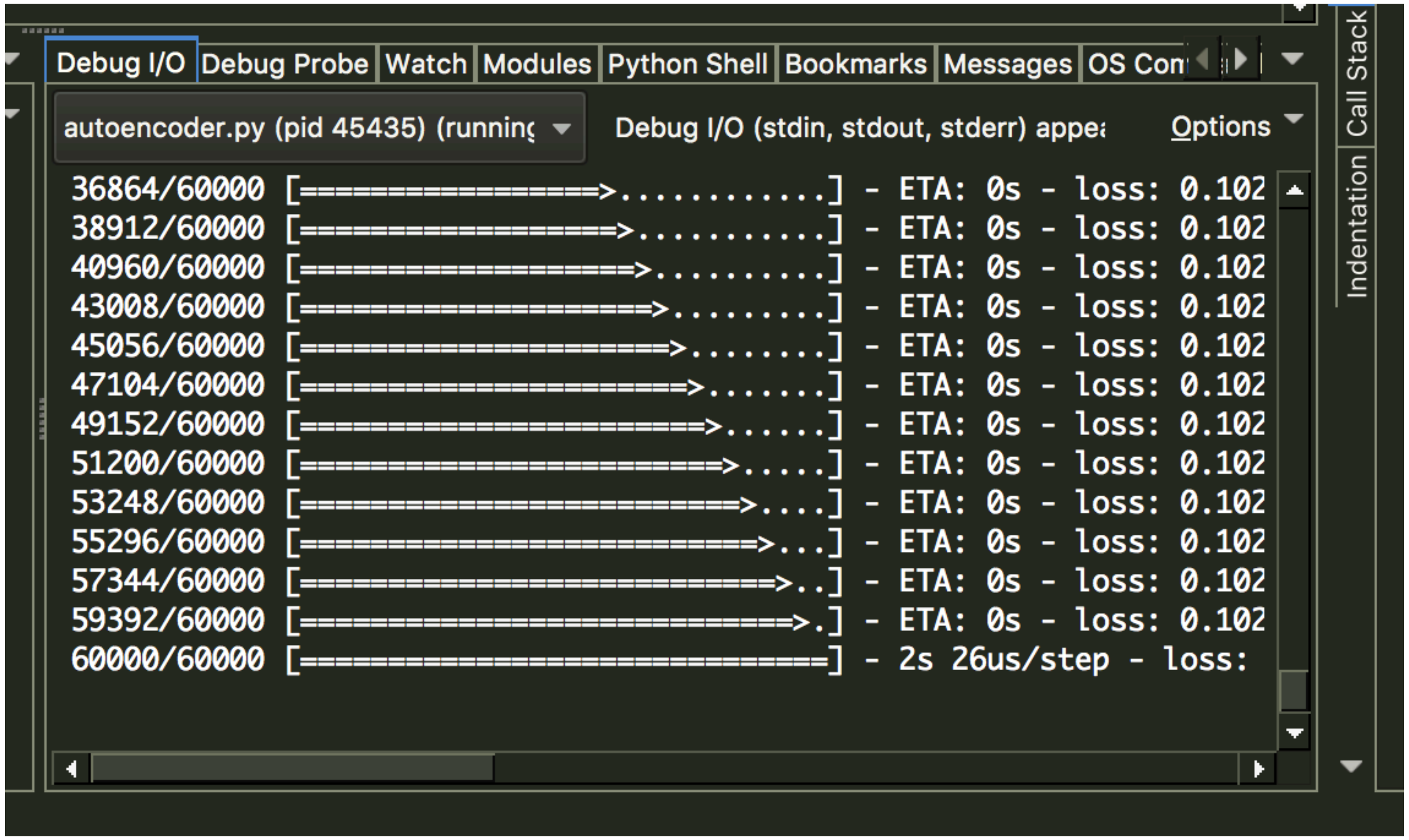
HOMEWORK #1

- The python code, once opened.
- Press the green “play” button to run the code:



HOMEWORK #1

- Here's what the output should look like:



Online Python documentation

- Within Python
  ```
  >>> help()
  >>> help('if')
  ```

- More useful: Online documentation
  https://docs.python.org/3/
  https://docs.python.org/3.6/tutorial/

- NumPy documentation
  https://docs.scipy.org/doc/
  (look for the NumPy user's guide, but there's a lot of other useful stuff here)

- Matplotlib documentation (plotting and graphics)
  https://matplotlib.org

Topics we will cover

- The Python environment
- Using modules
- Variables, assignment, data types, basic syntax
- Scripts and functions, paths
- Comparison, logical operators
- Control flow: if, for, while
- Making fancy figures
- Vectors and matrices
- Python hotdogging

HW#2: The logistic function

$$f(x) = \frac{L}{1 + e^{-k(x - x_0)}}$$

Due next Thursday before class starts.

Create two functions that implement a logistic transformation (next slides have specifics).

Submit the files containing the functions through Brightspace.

The lecture slides contain all the Python functions and commands you need to do this assignment.

HW#2: The logistic function

$$f(x) = \frac{L}{1 + e^{-k(x - x_0)}}$$

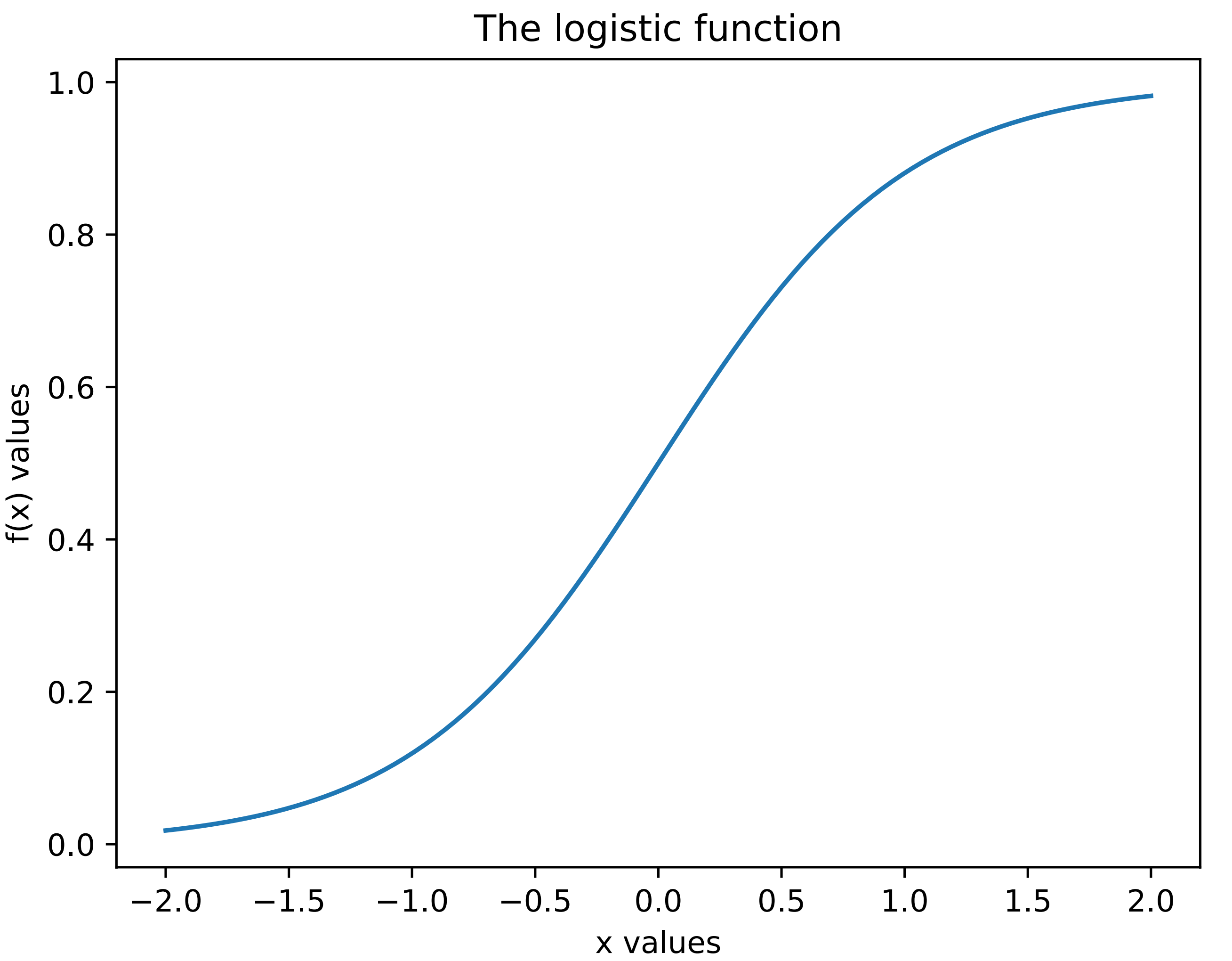
That L term can be fixed at 1. The x0 term can be fixed at 0. e is Euler's number (check out the exp function in numpy).

The function should take a vector of x values as an input argument. Another input argument should allow the user to specify k. The output argument should be a vector of f(x) values corresponding to the x values.

The function should also create a figure plotting the input x values against the output f(x) values. The figure's axes should be labeled "x values" and "f(x) values".

HW#2: The logistic function

So if we gave it x values ranging from -2 to 2, and k=2, the plot should look something like this:



This was ~14 lines of python code. Helpful functions from numpy: linspace, exp. From matplotlib.pyplot: plot, xlabel, ylabel, title, savefig

HW#2: The logistic function

The two functions

The first function should use a for loop to iterate through the vector of x values.

The second function should use vector arithmetic to transform the x values into f(x) values without a for loop.

(both functions should produce the same output values for a given set of input values!)

The Python environment - Vocabulary corner

- interpreted language
- execute / evaluate
- path
- workspace
- script
- function
- scope

The Python interpreter

Launch the interpreter by typing "$ python3" at a command prompt

Launch the interpreter by launching WingIDE and finding the "Python Shell" tab

Type code at the command line & Enter will **execute** it

Any variables you create here live in the global workspace

When you execute a function it creates its own workspace. It can't see the variables in the global workspace unless they are passed to the function as arguments.

Modules

Modules allow you to access helpful functions written by other people. You can make these other functions available in your workspace by typing:

```
import numpy as np
# now numpy functions are accessible as, e.g.
x1 = np.zeros((1,5))

import matplotlib.pyplot as plt

# without the ‘as’ syntax you don’t need to have
# the prefix, the functions are just directly
# available
import os
```

The python environment - arithmetic

```
>> 1+1
>> 10-1
>> 10*10
>> 10/5
>> 5**2          # raise to a power
>> np.sqrt(25)  # can you guess?
>> np.exp(1)    # Euler's number: e¹
```

1. That was meant to be an exponent, not a footnote[2]
2. BTW you need this function for your homework

The python environment - order of operations

PEMDAS applies

(parenthesis, exponent, multiplication/division, addition/subtraction)

operations with same precedence are evaluated left to right

If you can't remember, just add some parentheses?

```
>> 5*4-3

>> (5*4)-3  # same thing

>> 10/2*2

>> 10/(2*2) # not the same thing
```

Variables, assignment, data types

```
Basic types: float, int, str, bool
>> x1 = 5       # ends up being an int
>> x2 = 5.0     # is a float
>> x3 = ‘hi!’; # a string
>> x4 = True;  # logical (True, 1)
>> x5 = x1==6; # logical (False, 0)
>> type(x1)
<class ‘int’>
```

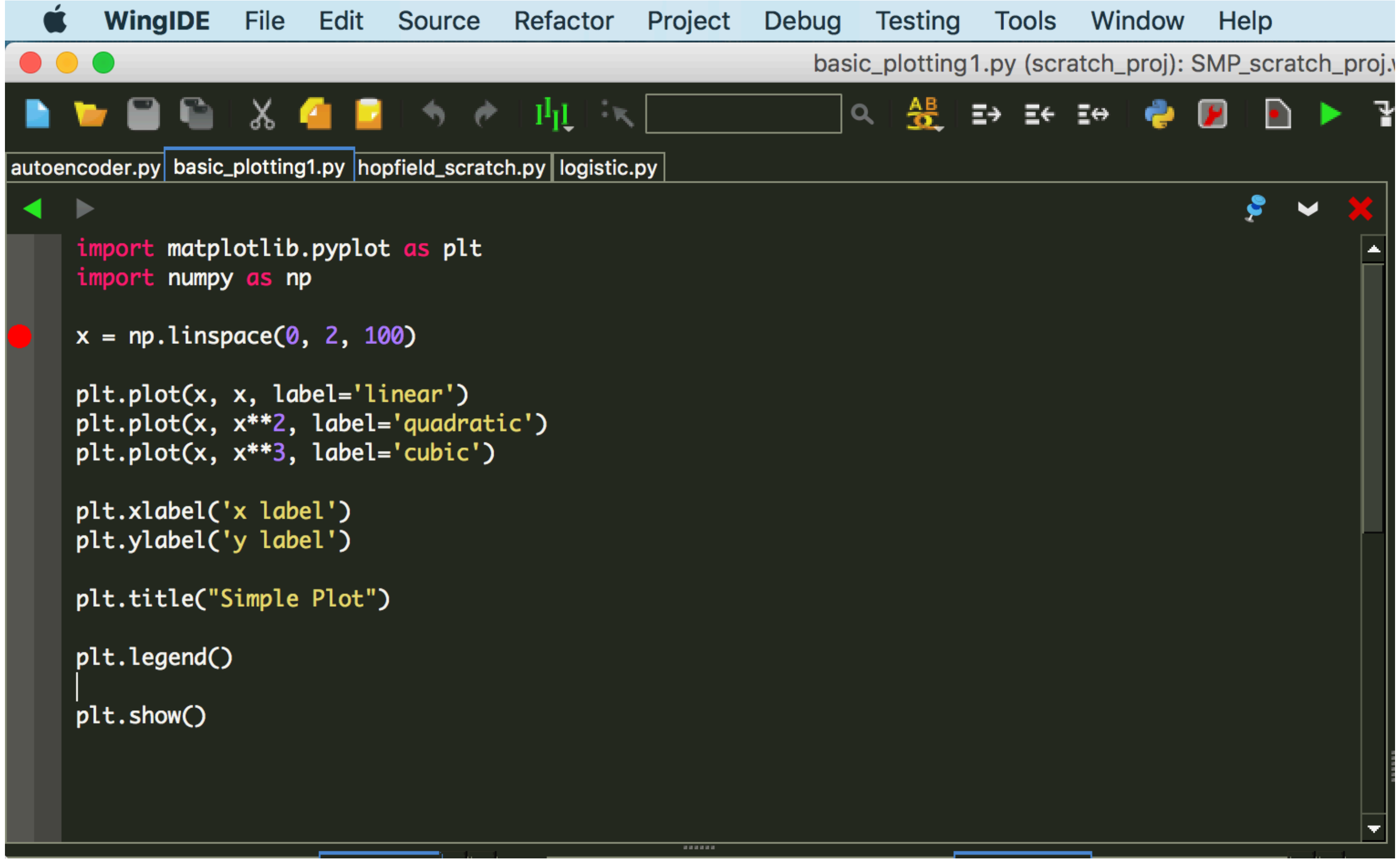
Writing scripts

The Wing Editor allows you to write scripts and functions.

The editor is just a text editor, you could use another text editor to make scripts & functions (like Emacs, but not MS Word).

You write some code in this window, and save it, and a text file containing your code is saved to disk.

Keep track of where (what directory) that file is saved!



Writing scripts

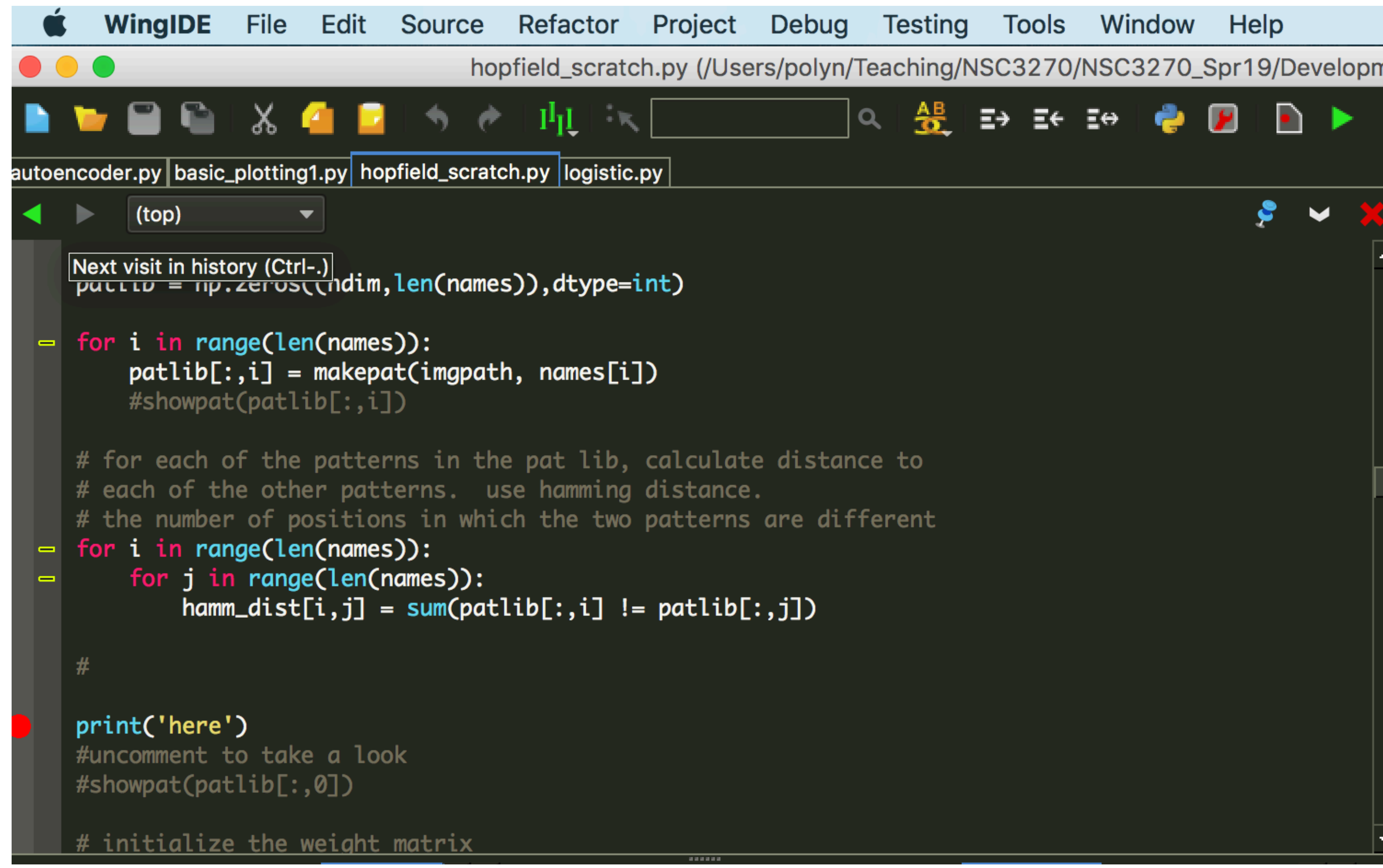
Comments:

If you have a # sign in your code, that indicates the following text is a comment, that code won’t be evaluated.

Useful for making notes to yourself or to us.

Execution:

Press the green play button to run the script that’s currently open



Writing scripts

The path:

The path is the set of directories python will look in to check for scripts or functions. Some directories are automatically on the path, like whatever directory you are currently in.

If you get an error like:

```
>> my_logistic(5)

Traceback (most recent call last):

  Python Shell, prompt 38, line 1

builtins.NameError: name ‘my_logistic' is not defined
```

One possibility is that you are in the wrong directory, and wherever my_logistic.py got saved is not on your path.

```
>> os.getcwd()

‘/Users/polyn/Teaching/NSC3270/NSC3270_Spr19/Wing'
```

This will tell you your current working directory.

Writing scripts

Changing directory / Where am I?

The “os” module has functions to change your working directory / determine where you are currently

Adding directories to your path:

If you have some scripts / functions in one directory, but need to navigate to another directory, you can add a directory to your path.

When specifying a path, the tilde ~ symbol is usually shorthand for your home directory.

In WingIDE, go to Project > Project Properties for a dialogue box to add folders to your path

Functions

In mathematics, a function is used to map one set of values to another set of values.

Here are two linear functions. They are equivalent to one another.

By convention, m and b are constants, but x and y are variables. Here, you can pick a value for x, plug in the values of m and b, and calculate the value of y. So x is like an input to the function, and y is an output.

In programming, a function is very similar!

$$y = mx + b \qquad\qquad f(x) = mx + b$$

Functions and scripts

If you just start writing code in the editor, then you are writing a **script**.

You can define a function within a script, using a particular syntax. The function definition has to have a particular structure.

```
def function_name(input_args):

    # the function code

    return output_arg
```

Functions and scripts

While a function is being executed, it behaves a bit differently than when a script is being executed.

The function has its own workspace: The only variables it has access to are the ones specified in the set of input arguments, and any that you define within the function.

```
def function_name(input_arg):
    # the function code would go here
    output_arg = input_arg / 10
    return output_arg

# calling the function
>> x2 = function_name(1000)
100.0
```

Relational operations (on numbers)

Single = means assignment, Double == means comparison

produces a logical: True (1) or False (0)

```
>> x=5
>> y=6
>> x==y  # these two things are equal: False
>> x!=y  # these two things are not equal: True
>> val = x==y      # val has type 'bool'
>> val = np.equal(x,y) # the functional form equivalent
>> val = x < y     # x is less than y: TRUE
>> val = x > y     # x is greater than y: FALSE
>> x >= y
>> x <= y
```

Logical operators

```
>> x=True

>> y=False

>> val = x and y     # logical AND

>> val = x or y     # logical OR

>> val = not y     # logical NOT
```

Logical operators

```
>> x=True

>> y=False

>> val = x and y    # logical AND: true if both x and y are True, so evaluates as False

>> val = x or y    # logical OR: True

>> val = not y    # logical NOT: True
```

Logical operators

```
>> x = [True, True, False] # this is a list
>> y= [True, False, False]
>> val = x and y     # logical AND, element-wise
```

Data types - Arrays (vectors) and lists

Creating an array / vector

```
>> x = np.zeros((1,10)) # one row, 10 columns
>> y = np.zeros((10,1)) # 10 columns, 1 row
>> x = [8, 1, 4, 2, 3, 1] # type is 'list'
>> x[0] > x[1] # 8 > 1, True
>> x = np.array([8, 1, 4, 2, 3, 1]) # an array
>> x[0] > x[1] # 8 > 1, still True
```

Data types - Arrays (vectors) and lists

```
Referencing elements of an array or list

>> x = np.array(range(10))

>> y = np.array([8, 1, 4, 2])

>> x[0]       # the first element, ‘zero-indexed’

>> y[0]

>> x[:3] # everything up to the 3rd index, exclusive

array([0, 1, 2])

>> y[[1,3,2,0]] # what will be the result?
```

Data types - Arrays (vectors)

```
>> x = np.array([])  # the empty array
>> x.shape # is (0,)
>> x = np.array([1,2,3])
>> x.shape
>> x[0] = 6; # now x is [6 2 3]
>> x = np.array((range(10)))  # makes a row vector, 0 to 9
>> x[7:]   # will give you [7 8 9]
>> x[:] # is the same as just typing x, gives all elements
>> x[12]
builtins.IndexError: index 12 is out of bounds for axis 0
with size 10
```

Array assignment

```
>> x=np.array((range(10)))

>> y=np.array([58,59,60])

>> x[:3] = y

array([58, 59, 60,  3,  4,  5,  6,  7,  8,  9])

>> x[3:5] = y

builtins.ValueError: could not broadcast input array
from shape (3) into shape (2)

>> x[3:6] = y

array([ 0,  1,  2, 58, 59, 60,  6,  7,  8,  9])
```

Working With Arrays (vectors)

```
>> x = np.array([1,2,3])
>> x * 5      # [5 10 15]
# multiply each element of an array by a certain number
>> x / 2      # [0.5 1. 1.5]
# divide each element of an array by a certain number
```

Control flow - if statements

Pseudocode version:

If this condition is true

run this code

Otherwise

run this code

```
if x==y:
    print(‘they are equal!’)
else:
    print(‘they are not equal!)
end
```

Control flow - if statements

```
if x==y:
   print(‘x and y are equal!’)
elif x==z:
   print(‘well, at least x and z are equal!)
else:
   print(‘x is not equal to anything :(‘)
```

Control flow - for loops

The workhorse of programming. Run the enclosed code for each element in a list. On the left side of the equals sign you specify the index variable you'll use, on the right side, you provide the list.

```
count = 0;

for i in range(10):

   count = count + i


# alternative syntax: count += i
```

Control flow - nested for loops

```
for i in range(10):
    for j in range(2,5):
        print(‘%d’ % j)
    print(‘\n’)

# what will this code do?
```

Control flow - while loops

```
% a while loop will just keep looping as long as the condition is
true
count = 0
flag = True
while flag:
    count = count + 1
    if count > 100:
        flag = False

% same thing
count = 0
while True
    count = count + 1
    if count > 100
        break
```

Making fancy figures

```
# the bare basics
>> import matplotlib.pyplot as plt
>> xvals = np.linspace(0,2*np.pi,100)
>> yvals = np.sin(np.linspace(0,2*np.pi,100))
>> plt.plot(xvals,yvals)
>> plt.xlabel("time")
```

Data types - Arrays (matrices)

```
>> x1=np.array([[1,2,3],[4,5,6],[7,8,9]])
array([[1, 2, 3],
       [4, 5, 6],
       [7, 8, 9]])
# what will get displayed for each of these?
>> x1[2,2]
>> x1[1,:]
>> x1[:,1]
>> x1[0:2,0:2]
```

Data types - Arrays (matrices)

Logical operators work on vectors and matrices, and can be used to create masks.

```
>> x1=np.array([[1,2,3],[4,5,6],[7,8,9]])

>> x1 >= 5

array([[False, False, False],
       [False,  True,  True],
       [ True,  True,  True]])

>> x1[x1>=5]

# what values will it give you?

# hotdog: in what order will it give them?
```

Good luck! Have fun storming the castle!