LECTURE 1: INTRO

Introduction to Scientific Python, CME 193

Jan. 9, 2014

web.stanford.edu/~ermartin/Teaching/CME193-Winter15

Eileen Martin

Some slides are from Sven Schmit's Fall '14 slides

Course Details

- When and where: Nvidia Auditorium, 12:50-2:05 Fridays for the first 8 weeks of the quarter
- Grading: 1 unit, pass/fail
- Weekly homeworks due each Friday at 12:50 posted on course website:

`web.stanford.edu/~ermartin/Teaching/CME193-Winter15/assignments.html`

- Must get at least 70% of homework points to pass
- Follow the honor code. Discussion with classmates is ok but coding must be individual. Moss will be used.
- Forums on CourseWork can be used for discussion
- Office hrs. Huang basement Mon. 9:30-10:30, Wed. 3:15-4:15

Homework submission

- Submit assignments in zipped folder to Coursework Drop Box
- 3 late days throughout the quarter
- All assignments will be graded on corn (Stanford FarmShare machines), so make sure your code runs there
- Submit writeups in pdf format

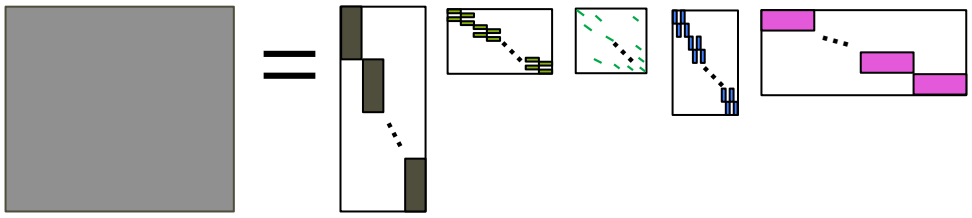
Feedback

- Course evaluations only help future classes
- If you want something changed, say so!
  - Talk to me
  - Email
  - Anonymous online survey:

    https://www.surveymonkey.com/s/NSVJDDJ

Instructor

- Eileen Martin, PhD student in ICME
- From Texas (went to UT)
- Undergrad in math and computational physics
- Research in seismic imaging and associated computational & mathematical issues



Course Overview

- Getting started with Python
- Variables and flow control
- Functions
- Unit testing
- Basic data structures (lists, dictionaries, tuples, sets)
- Working with strings
- Using data from and writing to files (input/output)
- Numpy, Scipy, Matplotlib
- Object-oriented programming
- Ipython notebooks and visualization tools, Guest Lecture
- Extras: Cython, exception handling

Overview of Python

- Easy to use *object-oriented* language
  - Data structures have attributes and methods
- Typically used as an *interpreted* language
  - Code is executed step-by-step, lines translated to subroutines
  - Cython (another lecture) gives an option for interpreted or compiled
- Can be run in *interactive mode* or with *scripts*
- We'll use Python 2 since it's on the corn machines, but it is relatively easy to switch to Python 3

Overview Today

- Python's interactive mode and using scripts
- Values, types, and variables
- Keywords, pre-defined operators
- Modules
- Flow control

Starting Python in interactive mode

- ssh into corn:

```
ssh -X sunetID@corn.stanford.edu
```

- Open python in interactive mode:

```
python
```

- Type your first command:

```
“Hello, world!”
```

- Get some help with the print function:

```
help()
print
q
```

- Try using the print function to display `Hello, world!`
- Quit interactive mode:

```
quit()
```

Create and run a Python script

- Create and browse to the folder where you want to work, for example:

```
mkdir ~/cme193
mkdir ~/cme193/lecture1
cd ~/cme193/lecture1
```

- Open a new document in your favorite text editor, for example:

```
emacs helloWorld.py
```

- Put the following line in that file and save it:

```
print("Hello, world!")
```

- Run the script:

```
python helloWorld.py
```

Overview Today

- Python's interactive mode and using scripts
- Values, types, and variables
- Keywords, pre-defined operators
- Modules
- Flow control

Values and types

- Examples:

Value	Type
True, False	bool (boolean)
3.14159, -1.4, 2e-4	float
4, -5, 10000, 0	int (integer)
“Hello, world!”	str (string)
2+3j, -2e-4+2.5e1j	complex
None	NoneType

indicates absence of a value

- To check the type of a variable, `x`, in interactive mode use `type(x)`

First Use of Variables

- Open up a new script, `helloWorld2.py`
- Put the following commands in the script and save:

```
a = “Hello, world!”
print(a)
print type(a)     # print can use keyword or function () syntax
a = 12
print(a)
print(type(a))    # see how a can easily change types
```

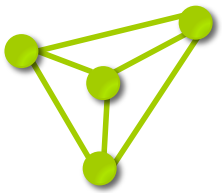
- Run the script: `python helloWorld2.py`
- Note that Python doesn’t complain when you suddenly ask a to change types

Tips for Using Variables

- Frequently use variables instead of numbers (easy to modify/understand code)
- Variables should start with a letter and not be a keyword
- Choose meaningful variable names, concise & descriptive

- These codes check if a connected graph can be drawn with no edge crossings in a plane:

```
variable = 4 – 6 + 4
if variable == 2:
    print(“Planar graph”)
```



```
face = 4
vert = 4
edge = 6
EulerChar = vert – edge + face
if EulerChar == 2:
    print(“Planar graph”)
```

```
f = 4
v = 4
e = 6
ec = v – e + f
if ec == 2:
    print(“Planar graph”)
```

Overview Today

- Python's interactive mode and using scripts
- Values, types, and variables
- Keywords, pre-defined operators
- Modules
- Flow control

Keywords in Python

Some examples that we'll use in the next few weeks:

print	not	or	def
if, elif, else	while	for	break
import	from	class	return

More description of keywords:
http://www.pythonforbeginners.com/basics/keywords-in-python

Testing whether something is a keyword:
https://docs.python.org/2/library/keyword.html

Statements and Expressions

- *Statements* are instructions Python executes

```
var = “Hello, world!”
```

- *Expressions* are made up of values, operators, and variables (that have assigned values)

```
var + “ Hello, again!”
```

- In interactive mode, statements aren’t printed out unless they call print, but expressions are echoed
- When using a script, an expression is not echoed

Booleans

- Expressions:

syntax	==	!=	>	>=	<	<=
name	equals	doesn't equal	greater than	greater than or equal	less than	less than or equal
example	5 == 5	5 != 5	5 > 4	5 >= 5	5 < 4	5 <= 5
ex. result	True	False	True	True	False	True

- Basic operators:

`True and False` yields `False`

`True or False` yields `True`

`not True` yields `False`

Strings

- Basic operators that output strings:

```
str + str,   str * int,   int * str
```

- Try accessing subsets of strings:

```
a = “Hello, world!”
print(a[0])
print(a[:3])
print(a[1:3])
print(a[3:])
```

Integers vs. floats

- Basic operators: `+ - * /`
- `Modulus %`
- `Exponential **`
- If operands of / are integers, Python returns the floor of their quotient by default*
- If either operand of / is a float, Python returns a float
- Note the limits of finite precision and max & min values

*In Python 3, / always returns the true quotient as a float

Overview Today

- Python's interactive mode and using scripts
- Values, types, and variables
- Keywords, pre-defined operators
- Modules
- Flow control

Modules

- Python has a lot of extra functionality that must be loaded as needed by *importing modules*
- Examples: Numpy, Matplotlib, Scipy
- For example, if you want to overwrite the default integer division in Python 2 to return the true result, try:

```
>>> from __future__ import division
>>> 1/2
0.5
```

- Or you could get more mathematical functionality:

```
>>> import math
>>> math.pi
3.14159265358979
```

Overview Today

- Python's interactive mode and using scripts
- Values, types, and variables
- Keywords, pre-defined operators
- Modules
- Flow control

Flow control

- These control statements structure a code's logical flow:
  `if, elif, else, for, while, break, continue, pass`
- Each brown control statement determines whether a block of code is executed
- That block is set apart with an **indentation** (not spaces).

```
a = 2
statement = (a >= 1)
if statement:
    # If statement is True, execute line below
    # If statement is False, don't execute it
    print "True statement"
    a -= 1
# No more indention so next line is executed
# whether statement is True or False
print a
```

If-Elif-Else statements

- An if-elif-else statement decides whether to execute a block of code one time.
- These two snippets are logically equivalent:

```
if traffic_light == ‘green’:
    drive()
elif traffic_light == ‘yellow’:
    accelerate()
else:
    stop()
```

```
if traffic_light == ‘green’:
    drive()
else:
    if traffic_light == ‘yellow’:
        accelerate()
    else:
        stop()
```

The elif (short for else if) keeps code more compact.

For loops

- A for loop lets us repeat a code block
- The for loop iterates over some object with data that is organized like a list
- A common function producing a list of integers is `range(n)` which yields 0, 1, 2, … , n-1
- Try this in interactive mode (… means it's waiting for end of code block):

```
# Triangular numbers
for i in range(5):
    print i*(i+1)/2
# outputs 0 1 3 6 10
```

While loops

- A while loop lets us repeat a code block even when we don't know how many times it needs to be iterated
- It continues evaluating the code block as long as the statement is true
- Try this in interactive mode:

```
# Triangular numbers
t = 0
i = 1
while t < 12:
    print t
    t += i
    i += 1
# outputs 0 1 3 6 10
print t
# outputs 15
```

- Beware of infinite loops!

```
# Triangular numbers
t = 0
i = 1
while t < 12:
    print t
    i += 1
# outputs lots of 0's
```

Pass

- `pass` is nice to use as a placeholder while you're working on code because it does nothing

```
# Checking for prime numbers
if n % 2 == 1:
    pass # implement test for odd numbers later
elif n == 2:
    print n, " is prime"
else:
    print n, " is not prime"
```

Break

- `break` lets us exit the smallest for or while loop enclosing that line
- Try this example from Python documentation:

```
# Finding prime numbers
for n in range(2, 10):
    for x in range(2, n):
        if n % x == 0:
            print n, 'equals', x, '*', n/x
            break
    else:
        # execute if for loop completed
        # without finding a factor
        print n, 'is prime'
```

Continue

- `continue` skips the rest of the lines in the smallest enclosing loop and continues with the next iteration

```
# Finding a given letter in a sentence
letter_chosen = “s”
sentence = “this is a sentence”
other_chars = 0
for letter in sentence:
        if letter == letter_chosen:
                print letter_chosen, “found”
                continue
        other_chars += 1
print “There were”, other_chars, “other characters”
```

First assignment

- Posted on the course website: http://stanford.edu/~ermartin/Teaching/CME193-Winter15/assignments.html
- Tips:
  - Online documentation is your friend. Don't hesitate to use it!
  - Stuck? test smaller, simpler statements in interactive mode
  - Build test cases to verify correctness of your code
  - Talk to each other. Use the CourseWork Forums.
  - Come to office hrs. Mon. 9:30-10:30, Wed. 3:15-4:15