CME 193: Introduction to Scientific Python

Lecture 1: Introduction

Sven Schmit

`stanford.edu/~schmit/cme193`

Contents

- Administrivia

- Introduction

- Basics

- Variables

- Control statements

- Exercises

Feedback

If you have comments, like things to be done differently, please let me know and let me know asap.

Questionnaires at the end of the quarter are nice, but they won't help you.

Content of course

- Variables

- Functions

- Data types

  - Strings, Lists, Tuples, Dictionaries

- File input and output (I/O)

- Classes

- Exception handling

- Recursion

- Numpy, Scipy and Matplotlib

- Pandas, Statsmodels and IPython

- Unit tests

- More packages

Setup of course

- Lectures: first half lecture, second half exercises

- Portfolio

- Final project

More abstract setup of course

My job is to show you the possibilities and some resources (exercises etc.)

Your job is to teach yourself Python

If you think you can learn Python by just listening to me, you are grossly overestimating my abilities.

Exercises

Exercises in second half of the class. Try to finish them in class, else make sure you understand them all before next class.

At the end of the course, hand in a portfolio with your solutions for exercises.

Feel free (or: You are strongly encouraged) to work in pairs on the exercises. It's acceptable to hand in the same copy of code for your portfolio if you work in pairs, but do mention your partner.

Portfolio

You are required to hand in a portfolio with your solutions to the exercises you attempted.

This is to show your active participation in class

You are expected to do at least 2/3rd of the assigned exercises. Feel free to skip some problems, for example if you lack some required math background knowledge.

Final project

Besides the portfolio, you are required to submit a project, due one week after the final class.

One paragraph proposal due before lecture 4.

Have fun by working on a project that interests you while learning Python.

You are encouraged to use material not taught in class: we can't cover everything in class.

No teams :(

Final project ideas

Some more pointers

- See course website for some possible projects

- Focus on Python, not the application (no research)

Some successful previous projects

- Predicting stock movement, obtaining data using Quandl Api

- Crawling and analyzing data from a large forum

- Finding recent earthquakes close to a given location using Google maps API and USGS data

Workload

The only way to learn Python, is by writing Python... **a lot**. So you are expected to put in effort.

From past experience: If you are new to programming, consider this a hard 3 unit class where you will have to figure out quite a bit on your own. However, if you have a solid background in another language, this class should be pretty easy.

To new programmers

Be warned, this will be difficult.

The problem: 8 weeks, 8 lectures, 1 unit. We will simply go too fast.

Alternative: spend some time learning on your own (Codecademy / Udacity etc). There are so many excellent resources online these days. Then you can always come back in Fall; this course will be offered again.

Misc

Website `stanford.edu/~schmit/cme193`

Piazza Use Piazza for discussing problems. An active user on Piazza has more leeway when it comes to portfolio, as it shows involvement.

Office hours After class or by appointment (shoot me an email).

References

The internet is an excellent source, and Google is a perfect starting point.

The official documentation is also good, always worth a try:
`https://docs.python.org/2/`.

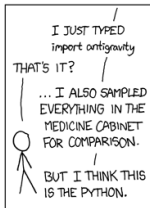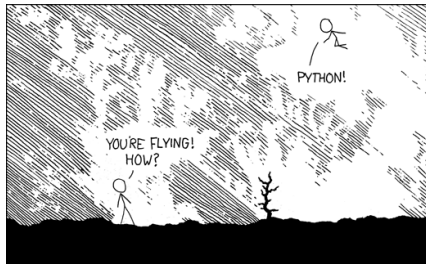Course website has a list of useful references.

Last words before we get to it

- Work

- Make friends

- Fail often

- Fail gently

Contents

- Administrivia

- Introduction

- Basics

- Variables

- Control statements

- Exercises

Python

Python

- Relatively easy to learn

- Fast to write code

- Intuitive

- Very versatile (vs Matlab/R)

- Less control, worse performance (vs. C)

- Less safety handles, responsibility for user

Contents

How to install Python

Many alternatives, but I suggest installing using a prepackaged
distribution, such as Anaconda

`https://store.continuum.io/cshop/anaconda/`

This is very easy to install and also comes with a lot of packages.

See the getting started instructions on the course website for more
information.

Packages

Packages enhance the capabilities of Python, so you don't have to progam everything by yourself (it's faster too!).

For example: Numpy is a package that adds many linear algebra capabilities, more on that later

How to install packages

To install a package that you do not have, use `pip`, which is the Python package manager.

such as

```
$ pip install seaborn
```

Python 3

Python 3 has been around for a while and is slowly gaining traction.

However, many people still use Python 2, so we will stick with that.

Differences are not too big, so you can easily switch.

Cloud9 demo

Please log into your Cloud9 account on c9.io

Clone the code using: `$ git clone`
`https://github.com/schmit/hangman.git`

If you don't have git and want a local copy, download the code using
`https://github.com/schmit/hangman/archive/master.zip`

How to use Python

There are two ways to use Python:

command-line mode: talk directly to the interpreter

scripting-mode: write code in a file (called script) and run code by typing
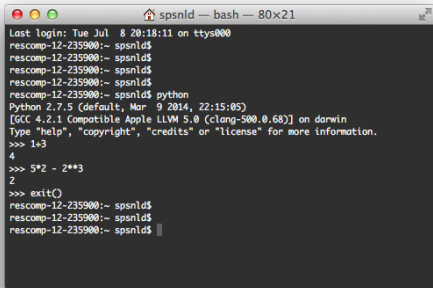
```
$ python scriptname.py
```

in the terminal.

The latter is what we will focus on in this course, though using the command-line can be useful to quickly check functionality.

The interpreter

We can start the intepreter by typing 'python' in the terminal.

Now we can interactively give instructions to the computer, using the

Python language.

Scripting mode

A more convenioned way to interact with Python is to write a script.

A script contains all code you want to execute. Then you call Python on the script to run the script.

First browse, using the terminal, to where the script is saved

Then call `python scriptname.py`

Scripting mode

Suppose the Python script is saved in a folder /Documents/Python
called firstscript.py.

Then browse to the folder by entering the following command into the
terminal

```
$ cd ~/Documents/Python
```

And then run the script by entering

```
$ python firstscript.py
```

Print statement

We can print output to screen using the print command

```
print "Hello, world!"
```

Contents

Values

A value is the fundamental thing that a program manipulates.

Values can be "Hello, world!", 42, 12.34, True

Values have types. . .

Types

Boolean   True/False

String   "Hello, world!"

Integer   92

Float   3.1415

Use type to find out the type of a variable, as in

» type("Hello, world!")

Variables

One of the most basic and powerful concepts is that of a *variable*.

A variable *assigns* a name to a value.

```
message = "Hello, world!"
n = 42
e = 2.71

# note we can print variables:
print n  # yields 42

# note: everything after pound sign is a comment
```

Try it!

Variables

Almost always preferred to use variables over values:

- Easier to update code

- Easier to understand code (useful naming)

What does the following code do:

```
print 4.2 * 3.5
```

```
length = 4.2
height = 3.5
area = length * height
print area
```

Variables

Almost always preferred to use variables over values:

- Easier to update code

- Easier to understand code (useful naming)

What does the following code do:

```
print 4.2 * 3.5
```

```
length = 4.2
height = 3.5
area = length * height
print area
```

Keywords

Not allowed to use keywords, they define structure and rules of a language.

Python has 29 keywords, they include:

- and

- def

- for

- return

- is

- in

- class

Integers

Operators for integers

+ - * / % **

Note: / uses integer division:

5 / 2 yields 2

But, if one of the operands is a float, the return value is a float:

5 / 2.0 yields 2.5

Note: Python automatically uses long integers for very large integers.

Floats

A floating point number approximates a real number.

Note: only finite precision, and finite range (overflow)!

Operators for floats

> \+ addition
>
> \- subtraction
>
> * multiplication
>
> / division
>
> ** power

Booleans

Boolean expressions:

$==$ equals: `5 == 5` yields `True`

$!=$ does not equal: `5 != 5` yields `False`

$>$ greater than: `5 > 4` yields `True`

$>=$ greater than or equal: `5 >= 5` yields `True`

Similarly, we have $<$ and $<=$.

Logical operators:

`True and False` yields `False`

`True or False` yields `True`

`not True` yields `False`

Booleans

Boolean expressions:

$==$ equals: `5 == 5` yields `True`

$!=$ does not equal: `5 != 5` yields `False`

$>$ greater than: `5 > 4` yields `True`

$>=$ greater than or equal: `5 >= 5` yields `True`

Similarly, we have $<$ and $<=$.

Logical operators:

```
True and False yields False

True or False yields True

not True yields False
```

Modules

Not all functionality avaible comes automatically when starting python, and with good reasons.

We can add extra functionality by importing modules:

```
>> import math
>> math.pi
3.141592653589793
```

Useful modules: `math`, `string`, `random`, and as we will see later `numpy`, `scipy` and `matplotlib`.

More on modules later!

Contents

Control statements

Control statements allow you to do more complicated tasks.

- If

- For

- While

If statements

Using if, we can execute part of a program conditional on some
statement being true.

```python
if traffic_light == 'green':
    move()
```

Indentation

In Python, blocks of code are defined using indentation.

This means that everything indented after an `if` statement is only executed if the statement is `True`.

If the statement is `False`, the program skips all indented code and resumes at the first line of unindented code

```python
if statement:
    # if statement is True, then all code here
    # gets executed but not if statement is False
    print "The statement is true"
    print "Else, this would not be printed"
# the next lines get executed either way
print "Hello, world,"
print "Bye, world!"
```

If-Else statement

We can add more conditions to the If statement using `else` and `elif` (short for else if)

```python
if traffic_light == 'green':
    drive()
elif traffic_light == 'orange':
    accelerate()
else:
    stop()
```

For loops

Very often, one wants to repeat some action. This can be achieved by a

for loop

```
for i in range(5):
    print i**2,
# 0 1 4 9 16
```

Here, range(n) gives us a *list* with integers $0, \ldots, n-1$. More on this

later!

While loops

When we not know how many iterations are needed, we can use `while`.

```
i = 1
while i < 100:
    print i**2,
    i += i**2  # a += b is short for a = a + b
# 1 4 36 1764
```

Continue

continue continues with the next iteration of the smallest enclosing loop.

```python
for num in range(2, 10):
    if num % 2 == 0:
        print "Found an even number", num
        continue
    print "Found an odd number", num
```

from: Python documentation

Continue

continue continues with the next iteration of the smallest enclosing
loop.

```python
for num in range(2, 10):
    if num % 2 == 0:
        print "Found an even number", num
        continue
    print "Found an odd number", num
```

from: Python documentation

Break

The break statement allows us to jump out of the smallest enclosing
for or while loop.

Finding prime numbers:

```python
max_n = 10
for n in range(2, max_n):
    for x in range(2, n):
        if n % x == 0:  # n divisible by x
            print n, 'equals', x, '*', n/x
            break
    else:  # executed if no break in for loop
        # loop fell through without finding a factor
        print n, 'is a prime number'
```

from: Python documentation

Break

The break statement allows us to jump out of the smallest enclosing
for or while loop.

Finding prime numbers:

```
max_n = 10
for n in range(2, max_n):
    for x in range(2, n):
        if n % x == 0:  # n divisible by x
            print n, 'equals', x, '*', n/x
            break
    else:  # executed if no break in for loop
        # loop fell through without finding a factor
        print n, 'is a prime number'
```

from: Python documentation

Pass

The pass statement does nothing, which can come in handy when you are working on something and want to implement some part of your code later.

```python
if traffic_light == 'green':
    pass  # to implement
else:
    stop()
```

Contents

- Administrivia

- Introduction

- Basics

- Variables

- Control statements

- Exercises

Exercises

See course website for exercises for this week.

Get to know the person next to you and do them in pairs!

Let me know if you have any question

Class ends at 5:35pm.