CME 193: Introduction to Scientific Python
Lecture 1: Introduction to Computing with Python

Austin Benson
Dan Frank
Institute for Computational and Mathematical Engineering (ICME)

April 3, 2013
Administrivia

What is a variable?

Arithmetic and boolean operators

Control Flow

Functions
About the instructors

(a) Austin

(b) Dan

ICME PhD students
Offices: ICME (Huang basement)
Course structure

► Five 2-hour lectures, all in first three weeks

► Homework corresponding to each lecture (5 total)

► Lectures:
  ► 1 hour of instruction
  ► 1 hour of interactive exercises

► Please bring a laptop to class!

All course materials are on coursework and
http://www.stanford.edu/~arbenson/cme193.html
Office hours

- Austin: Monday and Wednesday, 1-2pm, by appointment
- Dan: TBA
- Location: Huang basement near ICME
- Please also use Piazza to ask questions.
Passing the class

The grading scheme is Satisfactory/Not Satisfactory.

To pass, you need to complete all homework assignments and earn at least 70% of the points on each assignment.
Course textbook

- No textbook!

- We will post online materials on the course web site
Online material

- Short (1-3 minute) videos of topics covered in class.
- Experimental: feedback is greatly appreciated!
One reason why we are excited to teach this class:

- Students in: Civil Engr., Mech. Engr., Chemical Engr., EE, ERE, ICME, MSE, Econ, Math, Stats, CS, Business, Biology, Medical School, Political Science, Aero/Astro, Business, Earth Systems, Undeclared

- Undergraduate: year 2, 3, 4; Graduate: year 1, 2, 3, 4+
Homework 0

- Homework 0 is posted (it is not due)

- You should be comfortable with the material before starting Homework 1
Course outline

1. Introduction to Computing with Python
2. Data Structures
3. File I/O, Object-oriented Python, and Introduction to NumPy
4. NumPy and SciPy
5. Data Visualization and Web Scraping
Python 2.x vs Python 3

- Python is currently transitioning from version 2 to 3
- Examples will be in 2.7
- Homeworks have been made compatible with both.
Administrivia

What is a variable?

Arithmetic and boolean operators

Control Flow

Functions
Basic variables

A variable holds information (1.343, 'hi', [1, 1, 2, 3, 5, 8])

In Python this is simple:

```python
x = 1.343  # a number

greeting = 'hi'  # a string

arr = [1, 1, 2, 3, 5, 8]  # a list
```

# signifies the start of a comment in Python. The comment terminates at the end of the line.
Basic variables

Variables can change (they are *variable*)

\[ x = 1.343 \]
\[ x = 2 \]
\[ x = 10 + 4 \]
\[ x = -3 \]

(no more knowledge of 1.343, 2, or 14)
Unlike C/C++ and Java, variables can change types. Python keeps track of the type internally.

If you have PL theory background: Python is strongly typed but not statically typed.
Administrivia

What is a variable?

Arithmetic and boolean operators

Control Flow

Functions
Arithmetic operators

2 + 4
3.2 * 6
(8.7 - 3.3) / 4
'hi' * 10
'hello, ' + 'world!

Python supports this arithmetic on strings. (Note: Matlab does not.)
Arithmetic operators

Shorthand to combine assignment and addition statements:

```python
x = 4
x += 2  # x = x + 2
y = 'hi'
y *= 4  # y = y * 4
```

x is now 6 and y is now ‘hihihihi’
Comparison operators

```python
x == 1  # check for equality

y != 'hello'  # check for inequality

2 > 2  # False

2 >= 2  # True

2.1 < z < 5.4  # chained inequalities
```
Boolean operations

```python
x = 1
y = 'hello'
x == 1 or y == 'hi'
x > 0 and y != 'hi'

y = 0
x or y
x and y
not y
```

Python conveniently uses the keywords **and**, **or**, **not** instead of the symbols `&&`, `||`, `!`
Administrivia

What is a variable?

Arithmetic and boolean operators

Control Flow

Functions
The if statement is the most basic way to control the direction and flow of a program.

```python
if x:
    print 'hello1'

if not x:
    print 'hello2'

if not x and y:
    print 'hello3'
```
if is often accompanied by else to specify a second path for the code if the if statement is false

```python
if x:
    print 'hello'
else:
    print 'hi!'
```
Evaluating numerical values

As boolean values, the numerical value 0 is False and all other numerical values are True (1, 4.33, -12, ...).

```
x = -5
y = 5
z = 0

if x and y and z:
    print 'hello1' # will not print

if (x and y) or z:
    print 'hello2' # will print
```
The **while** loop repeatedly executes a task while a condition is true.

```python
i = 1
s = 0
while i < 10:
    s += i
    i = i + 1

# next lecture
s = sum(range(10))
```
What is this code doing?
The for loop also repeatedly executes a task.

Typically, for provides more structure:

```
for (i = 0; i < n; i = i + 1)
```
for (i = 0; i < n; i = i + 1)

Initialize index variable
Increment at the end of each loop
Condition to continue running the code
Python for

- Python uses `for` differently than Matlab, C++, ...
- `for` is used to iterate over elements in an “object”
- This is one reason why Python is easy and powerful
- More next lecture on how you can iterate over a “object”
Python for

```python
arr = [1, 2, 3, 4, 5, 6, 7, 8, 9]
for number in arr:
    print number * 2
```

more next lecture...
Administrivia

What is a variable?

Arithmetic and boolean operators

Control Flow

Functions
Functions are used to organize programs into coherent pieces.

In other words, functions are used to generalize or “abstract” components of a program.
What is a limitation of this code?

```python
x = 7.0
lower = 0.0
upper = x
guess = (upper + lower) / 2
while (abs(x - guess * guess) > 0.1):
    if guess * guess > x:
        upper = guess
    else:
        lower = guess
guess = (upper + lower) / 2
```
We do not want a $\sqrt{7}$ method, we want a $\sqrt{x}$ method

We still have the same capabilities (just let $x = 7$), but now our “abstracted” root finder can be used for more cases
Root finding

With this function, we can call root(7)

def root(x):
    lower = 0.0
    upper = x
    guess = (upper + lower) / 2
    while (abs(x - guess * guess) > 0.1):
        if guess * guess > x:
            upper = guess
        else:
            lower = guess
        guess = (upper + lower) / 2
    return guess
More general, we can call root(7, 0.1):

```python
def root(x, tol):
    lower = 0.0
    upper = x
    guess = (upper + lower) / 2
    while (abs(x - guess * guess) > tol):
        if guess * guess > x:
            upper = guess
        else:
            lower = guess
        guess = (upper + lower) / 2
    return guess
```
Python makes it easy to provide default argument values:

```python
def root(x, tol=0.1):
    lower = 0.0
    upper = x
    guess = (upper + lower) / 2
    while (abs(x - guess * guess) > tol):
        if guess * guess > x:
            upper = guess
        else:
            lower = guess
        guess = (upper + lower) / 2
    return guess
```

Can call `root(3)`, `root(113, 0.01)`
Root finding

A few things to think about:

- Are there cases where `root()` will have an error?
- Are there cases where `root()` will run forever? (fail to converge)
- How else can we generalize the function?
Basic functions

What will happen when this code runs?

```python
x = 5
def printer(y):
    print x + y

printer(5)
```
Basic functions

What about this code?

```python
x = 5

def printer(x, y):
    print x + y

printer(4, 5)
```
def polyval(p, x):
    val = 0
    i = 0
    for coeff in p:
        val += coeff * (x ** i)  # ** is ^
        i = i + 1
    return val

print polyval([1, 2, 0, 1], 4)  # prints '73'
def polyval(p, x):
    val = 0
    for i, coeff in enumerate(p):
        val += coeff * (x ** i)
    return val

print polyval([1, 2, 0, 1], 4)  # prints '73'
Python uses indentation ("white space") to group statements.

Each code block is indented the same amount (for loop, while loop, function definition, etc.).

Python will complain if your indentation is incorrect.
Indentation

```
a = 10
b = 2
while a > 1:
    print a + b
    a = a - 1  # wrong indentation --> error!
```

2-space or 4-space indentation is standard.
def func_incorrect1(x, y, z):
    if x < y:  # wrong indentation --> error!
        return z
    return 0

def func_incorrect2(x, y, z):
    if x < y:
        return z  # wrong indentation --> error!
    return 0

def func_correct(x, y, z):
    if x < y:
        return z
    return 0
Some people like this indentation structure and some people do not.

... Python isn’t going to stop using it
Assignments 1 is posted on the course website (due Tuesday 4/9)
In-class exercises, python codes from slides, and readings also posted

Next time:
1. More on Python functions
2. Strings
3. Lists
4. Dictionaries
5. Tuples