Accelerating Information Technology Innovation

http://aiti.mit.edu/program/ghana-summer-2013/

Ghana Summer 2013
Lecture 1 – Introduction to Python

MISTI

Reminders

- Materials are online at

aiti.mit.edu/program/ghana-summer-2013/

- Lectures and labs every weekday 11:30am-1pm and 3pm-5:30pm

- Office hours and more lab time: 10-11:30am, 5:30-6pm

Agenda

- Tech curriculum overview
- Python!

Goals

- Develop mobile applications
  - Mobile web
  - Android
  - SMS
- Learn where to find resources to learn on your own

Schedule

- Week 1: Python, Linux, Version control (Git)
- Week 2: Creating a website with Django
- Week 3: Android
- Week 4: SMS apps, mobile web interfaces
- Week 5: Demo day! Computer security

Week 1: Linux, Git, Python

Tuesday: Linux intro, Python intro

Wednesday: Using Linux and Github, Python practice

Thursday: Python functions, objects

Friday: Python Exceptions, regular expressions

Optional weekend challenge: Python ninja

Python Overview

- What is Python?
- Why Python?
- The Development Cycle

What is Python?

Python examples

```
print “hello”

a = 3 + 4

if 3 < 4:
    print “3 is less than 4!”
```

Python is…

- Interpreted
- Runs line by line
- Same on every computer

Python is…

- Dynamically typed; variable types are determined at runtime depending on what you assign to them:

```
# int
a = 1
# string
a = "a"
# list
a = [1,2,3]
# dictionary
a = {1:2,3:4}
```

Why Python?

Python because…

Easy to read and write

- No brackets or semicolons
- Whitespace

```
if (x)
{
  if (y)
  {
    a();
  }
  b();
}
```



```
if x:
    if y:
        a()
    b()
```

Python because…

- Lack of a separate compile step speeds rapid prototyping and debugging
- Dynamic typing speeds up development –no need to explicitly specify method argument types beforehand

Python because…

- Includes many convenient built-in data structures and functions

Why Python, For Us?

Python for us, because…

- We want each of you to reach millions of users, and don't want to waste time building the pipes and plumbing

- Python is supported by a number of good frameworks, including Django

The Development Cycle

The (Ideal) Development Cycle

- *Clearly* specify the problem:
  - Inputs, input manipulation, outputs
- Design the solution:
  - E.g what algorithms, data structures
- Implementation:
  - Coding!
- Test, test, test
  - Strongly suggest unit testing with PyUnit

The (Real) Development Cycle

- As above, but *faster*.
  - Python, as a dynamically typed, dynamic language is perfect for *rapid* prototyping
- Be prepared to throw away one (or more!) prototypes
  - Often you learn crucial things about the problem as you code which cannot be fixed without starting from scratch.

Interaction

- Python has an interactive console which is great for tinkering

```
$ python
Python 2.7.1+ (r271:86832, Apr 11 2011, 18:13:53)
[GCC 4.5.2] on linux2
Type "help", "copyright", "credits" or "license" for
more information
>>> a = 1
>>> a
1
>>> type(a)
<type "int">
>>>
```

- …etc

Try the interactive console

```
$python
>>> print “hello”
```

Python Variables and Operators

Variables and Operators

- Variables and operators
  - Strings
  - Numerics
  - Booleans
- Naming your variables
- Displaying output
- Getting user input

Variables

- Strings

```
>>> x = ‘Hello World’
```

- Numerics

```
>>> x = 3.1415
```

- Booleans

```
>>> x = True
```

- Lists

```
>>> x = [‘Hello’, True, 3.1415]
```

- And many more…

Variables

- Python is a “dynamically typed” language
  - A variable’s data type is not declared.
  - “Statically typed” languages like Java must declare a variable’s data type

```
String x = “Hello World”;
```

- Get a variable’s data type with the `type` function

```
>>> x = ‘Hello World’
>>> type(x)
<type 'str'>
```

Strings

- A string is a piece of text.
- Encase with quotes
  - Single-quotes

    ```
    >>> x = ‘abc’
    ```
  - Double-quotes

    ```
    >>> x = “abc”
    ```

Strings

- Use double-quotes to encase text containing single-quotes

```
>>> “It’s a string with a single-quote!”
```

- What is wrong with this statement?

```
>>> x = abc
```

Common String operations

```
>>> x = ‘Hello’
>>> y = ‘My name is Mike’

# Concatenate strings
>>> x + ‘.’
‘Hello.’
>>> x + ‘. ’ + y
‘Hello. My name is Mike’

# Equality
>>> x == ‘Hello’
True
>>> x == y
False
```

Common String operations

```
>>> x = ‘Hello’
>>> y = ‘My name is Mike’

# length of a string
>>> len(x)
5

# Convert to lowercase
>>> x.lower()
‘hello world’

# Convert to uppercase
>>> x.upper()
‘HELLO WORLD’
```

See all methods

```
>>> x = “Hello”
>>> dir(x)
```

Numerics

- Integers

```
>>> x = 10
>>> type(x)
<type 'int'>

>>> y = 10000000000
>>> type(y)
<type ‘long’>
```

- Decimals

```
>>> x = 3.1415
>>> type(x)
<type ‘float’>
```

Basic Arithmetic Operations

```
>>> x = 5
>>> y = 8
```

- Addition

```
>>> x + y
13
```

- Subtraction

```
>>> x – y
-3
```

- Multiplication

```
>>> x * y
40
```

Basic Arithmetic Operations

```
>>> x = 5
>>> y = 8
```

- Modulo division

```
>>>  y % x
3
>>> -8 % 5
2
```

212 % 100?

17 % 8?

Basic Arithmetic Operations

```
>>> x = 5
>>> y = 8
```

- Equality

```
>>> x == y
False
>>> x == 5
True
```

- Inequalities

```
>>> x < y
True
>>> x <= y
True
>>> x > y
False
```

Division

- Float division

```
>>> x = 10.0
>>> y = 8.0
>>> x / y
1.25
```

- Integer division. The result is rounded down to the nearest integer.

```
>>> x = 10
>>> y = 8
>>> x / y
1               # 1.25 rounded down

>>> x = -10
>>> x / y
-2              # -1.25 rounded down
```

Division

- If one variable is a float, then do float division.
- This is known as “type coercion”, i.e. coercion of integers to float.

```
>>> x = 10
>>> y = 8.0
>>> x / y
1.25
```

Order of numeric operations

- Same as standard arithmetic writing

1. Parenthesis
2. `**` (Exponent)
3. `*, /` (Multiplication, division)
4. `+,–` (Addition, subtraction)
5. `–` (Negative)

- If operations have equal precedence, then evaluate from left to right.
- Evaluate

```
>>> 3 + 6 / 3 * (1 + 1)
7
```

Booleans

- Variables with two values
  - `True`
  - `False`

```
# It’s a sunny day!
>>> is_sunny = True
>>> type(is_sunny)
<type ‘bool’>

# It’s not raining!
>>> is_raining = False
>>> type(is_raining)
<type ‘bool’>
```

Boolean logic
the not statement

```
>>> a = True
>>> b = True
>>> c = False
>>> d = False

# not x := the opposite of x
>>> not a
False
>>> not c
True
```

Boolean logic
the and statement

```
>>> a = True
>>> b = True
>>> c = False
>>> d = False


# x and y  := Evaluate x.  If x is False, return x. If not, return y
#          := True only when both x and y are True
>>> a and b
True
>>> a and c
False
>>> c and d
False
```

Boolean logic
the or statement

```
>>> a = True
>>> b = True
>>> c = False
>>> d = False


# x or y  := Evaluate x.  If x is True, return x.  If not, return y
#         := False only when both x and y are False.
>>> a or b
True
>>> a or c
True
>>> c or d
False
```

Boolean logic practice

```
>>> ((True or False) and False)
```

Naming your variables

- Name your variables to indicate what they're storing
  - Not helpful

```
>>> x = 'Ghana'
```

  - Informative

```
>>> country = 'Ghana'
```

- Use lowercase_with_underscores for multi-word functions and variable names
  - Encouraged

```
>>> soccer_team = 'Black Stars'
```

Naming your variables

- First character must be a letter
  - Invalid

```
>>> 1country = ‘Ghana’
>>> ₵five = 5
```

  - Valid

```
>>> one_country = ‘Ghana’
```

- Keep the name short for readibility
  - Too long:

```
>>> the_capital_city_of_ghana = ‘Accra’
```

  - Shorter

```
>>> ghana_capital = ‘Accra’
```

Output

- Just `print` it out!

```
# print a string
>>> print ‘Goooooal!’
Goooooal!

# without a print, the quotes remain
>>> ‘Goooooal!’
‘Goooooal!’

# print other data types
>>> print 3.1415
3.1415
```

Output

- Print newlines with the \n character

```
>>> print ‘First line\nSecond line’
First line
Second line
```

- Separate multiple phrases with commas

```
>>> players = 11
>>> print ‘There are’, players, ‘players’
There are 11 players on each team
```

Input

raw_input(“what is your name? ”)

>>> what is your name? *Leah*

Comments

- Single line comments are denoted with hash (#), multiline with three quotes ”””

```
# This is a comment
foo()
```

```
”””
This is a
longer comment
”””
foo()
```

- Comment your code!

Lab 1: Python basics

You can collaborate!

Python Control Structures

Agenda

- Control structures
  - `if…elif…else`
  - `for` loops
  - `while` loops
- Built-in functions
  - `range`
- Keywords
  - `break`
  - `continue`

if…elif…else

`if` statement

```
temp = 5

if temp < 10:
    print "It is cold!"
```

```
It is cold!
```

`if` statement

```
temp = 20

if temp < 10:
    print "It is cold!"
```

`if…else` statement

```
temp = 20

if temp < 10:
    print "It is cold!"
else:
    print "It feels good!"
```

```
It feels good!
```

`if…else` statement

```
temp = 40

if temp < 10:
    print "It is cold!"
else:
    print "It feels good!"
```

```
It feels good!
```

`if…elif…else` statement

```
temp = 40

if temp < 10:
    print "It is cold!"
elif temp > 30:
    print "It is hot!"
else:
    print "It feels good!"
```

```
It is hot!
```

Blocks

- Blocks are delimited with whitespace
- Use 2 spaces per level

```
if x:
  if y:
    a()
  b()
```

```
total = 0
for i in range(0:5)
  total += i
```

Loops

- Do the same thing multiple times

`while` loops

```
i = 0
while (i < 5):
    i = i + 1
    print i
```

```
1
2
3
4
5
```

`for` loops

```
for i in [1,2,3,4,5]:
    print i
```

```
1
2
3
4
5
```

the `range` function

```
# range(x) := [0,1,2,…,x-1]
for i in range(5):
    print i
```

```
0
1
2
3
4
```

the `break` statement

```
# break := break out of the loop
for i in range(5):
    if i > 2:
        break
    print i
```

```
0
1
2
```

the `continue` statement

```
# continue := continue with the next iteration
for i in range(5):
    if i == 3:
            continue
    print i
```

```
0
1
2
4
5
```

Lab 2: Control Structures