Agenda

- About the Course
- What is Python? and Why Python?
- Basic Syntax
- Strings
- User Input
- Useful Data Structures
- Introduction to Functions
About the Course
Administrivia

- Lectures and labs will be posted at:

- We will create an Official Mailing List for the course and announce it on Thursday.

- Lab sessions are from 6-9pm. We will be available during that time to check your homework.

- Please do the assigned readings **before** the class. It will help you understand the material covered in class better.
Course Expectations

• Attend class every day
• Collaborate
• Teach others as much as you can
• Do everything you can in the labs
• Ask questions!
Technical Course Outline

• Week 1 – Python
• Week 2 – Rapid Mobile Web Application Development with Django
• Week 3 – Android App Development
• Week 4 – SMS and IVR Application Development
• Weeks 5 & 6 – Final Project
• Week 7 – Project Showcase
What is Python?
Python is…

• …*interpreted*. Languages like C/C++ need to translate high-level code to machine code…
• …which means that a program has to be compiled separately for each type of machine:
Python is...

- Python code is compiled to an intermediate format called **bytecode**, which is understood by a *virtual machine/interpreter*.
Python is...
Why Python?
Python because…

• Portable and architecture-agnostic
• Convenient built-in functions and data structures
• Syntax is readable and fast to write

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

```python
if x:
    if y:
        a()
    b()
```
Python because...

• Great for rapid prototyping
  – No separate compile step
  – No need to explicitly specify method argument types beforehand (due to dynamic typing)
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, led by
  - Django
  - Heroku
  - Google AppEngine
The (Ideal) Development Cycle

• *Clearly* specify the problem:
  – Inputs, input manipulation, outputs
• Design the solution:
  – E.g. what algorithms, data structures
• Implementation
• Test
The (Real) Development Cycle

• As above, but *faster*.
  – Python, as a dynamically typed, programming 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.
Basic Syntax
Syntax

- Blocks are delimited with whitespace: specifically, four spaces (and no tabs)

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

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

- Semicolons are only used to separate multiple statements on the same line, which is discouraged:
Syntax

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

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

"""
This is a longer comment
"""
foo()
```
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
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
Strings

• A string is a piece of text.
• Encase with quotes
  – Single-quotes
    >>> x = ‘abc’
  – Double-quotes
    >>> x = “abc”
  – Triple single-quotes or triple double-quotes
    >>> x = ‘‘‘abc’’’
    >>> 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
String as a sequence

• You can access the characters one at a time using the bracket [] operator

1 fruit = “banana”
2 letter = fruit[1]
3 print letter
String operators

- Applied to strings, produce strings

```python
1    str1 = 'kit '  
2    str2 = 'kat '  
3    str3 = str1 + str2  
4    str4 = str3 * 2  
5    c = str1[0]  
6    c = str1[4]  
```

```
'kit kat '  
'kit kat kit kat '  
'k'
```

IndexError: string index out of range

```
str1
```

```
index 0 1 2 3
```

The slicing operator [m : n]

- Returns the part of the string from the "m-th" character to the "n-th" character, including the first but excluding the last.

```
fruit = "STRAWBERRY"

index = [0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10]

1. str1 = fruit[2:5]  # 'RAW'
2. str1 = fruit[:5]    # 'STRAW'
3. str1 = fruit[5:]    # 'BERRY'
4. str1 = fruit[6:-1]  # 'ERR'
```
User Input
User Input

- `raw_input` prints a prompt to the user and assigns the input to a variable as a string

```python
name = raw_input('What is your name?')
```

- `input` can be used when we expect the input to be a number

```python
age = input('How old are you?')
```
Control Statements
Control statements

- **Conditionals**: control which set of statements is executed.
  - if / else

- **Iteration**: control how many times a set of statements is executed.
  - while loops
  - for loops
The if statement

- If the condition is True, the body gets executed.
- Otherwise, nothing happens.

```python
if x < 0:
    print 'x is negative'
```
The if/else statement

- If the condition is True, body1 gets executed.
- Otherwise, body2 gets executed.

```python
if x < 0:
    print 'x is negative'
else:
    print 'x is positive or zero'
```
Chained conditionals

- If the condition1 is True, body1 gets executed.
- Otherwise, if condition2 is True, body2 gets executed.
- If neither condition is True, body3 gets executed.
An example

```python
a = False
b = True
if a and b:
    print 'I love red.'
elif a or b:
    print 'I love green.'
else:
    print 'I love blue.'
    print 'I also love purple.'
```

What does this output?  

```
I love green.
```

---

38
An example

```python
a = False
b = True
if a and b:
    print 'I love red.'
elif a or b:
    print 'I love green.'
else:
    print 'I love blue.'
print 'I also love purple.'
```

What does this output? I love green.
I also love purple.
Nested conditionals

```python
if is_adult:
    if is_senior_citizen:
        print 'Admission $2 off.'
    else:
        print 'Full price.'
else:
    print 'Admission $5 off.'
```

• Can get confusing. Indentation helps to keep the code readable and the python interpreter happy!
The while loop

- As long as the condition is true, the body gets executed repeatedly.
- The first time the condition is false, execution ends.
The while loop

```
i = 0
while i < 3:
    print i
    i = i + 1
```

• What does this output?

0
1
2
The break statement

- Immediately exits the innermost loop.

```python
while True:
    line = raw_input('>>> ')
    if line == 'done':
        break
    print line
print 'Done!'```

```console
>>> not done
>>> not done
>>> done
Done!
```
Useful Data Structures
Lists

• A list is a sequence of values.
• Each element (value) is identified by an index.
• The elements of the list can be of any type.

```python
tens = [10, 20, 30, 40]
cities = ['Manila', 'Cebu', 'Boracay']
empty = []
mixed = ['hello', 2.0, 5, [10, 20]]
```

• Lists can have mixed types in them, even other lists (nested).
Creating a list

- Use the [] brackets

```python
list_of ints = [10, 20, 30, 50]
```

**list_of_ints**

- Only one name
- Four int values
Individual elements are accessed using the [] operator.

```python
list_of_ints[0] = 17
```

Lists are mutable! Assigns the first element to 17.

List indexing starts at 0, not 1!

```
now has value 17
```

```
new_var = list_of_ints[0]
```

accesses the value of the first element.

```
now also has value 17
```
Printing a list

- We can use the print function to output the contents of the list:

```python
cities = ['Manila', 'Cebu', 'Boracay']
numbers = [17, 123]
empty = []
print cities, numbers, empty
```

```
['Manila', 'Cebu', 'Boracay'] [17, 123] []
```
Lists vs. Strings

- Lists are mutable - their contents can be modified
- Strings are immutable

```
name = 'Lenny'
name[0] = 'J'
```

TypeError: object doesn't support item assignment
Control Structures
The for loop

- Example:

```python
for i in [0, 1, 2, 3]:
    print i
```

- Sequence of values – list, string, etc.
- Any set of statements
- Indentation is important
Using range

index variable

generates sequence of \( n \) values starting at 0 and incrementing by 1

```python
for INDEX in range(n):
    BODY
```

any set of statements

• What does this output?

```python
for i in range(4):
    sq = i * i
    print i, sq
```

<table>
<thead>
<tr>
<th>i</th>
<th>sq</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>2</td>
<td>4</td>
</tr>
<tr>
<td>3</td>
<td>9</td>
</tr>
</tbody>
</table>
Using range

index variable

generates sequence of values
start and step are optional

for INDEX in range([start], stop, [step]):
BODY

any set of statements

• What does this output?

for i in range(1, 7, 2):
    print i

1
3
5
For loop and strings

• Iterating through the characters of a string

```python
str1 = 'stressed'
for c in str1:
    print c,
```

\texttt{stressed}
For vs While

• For loop is primarily used
  • for iterating over a sequence of values
  • when we know the number of iterations in advance

• While loop is primarily used
  • when we don't know the number of iterations in advance (they could be controlled by user input)
Introduction to Functions
Functions

- A **function** is a sequence of statements that has been given a name.

```python
def NAME (PARAMETERS):
    STATEMENTS
```

- function name
- list of function arguments
- any set of statements
- function definition
- function signature
Now you are all set to work on Lab 1! 😊
Lab 1

1. Calculate Fibonacci number
   \( \text{fib}(n) \)

2. Display the day of the week given a date
   \text{zellers}()

3. Implement the Rock Paper Scissors game
   \text{rock\_paper\_scissors}()

4. Encode a given string using the Caesar cipher
   \text{cipher}()
Next Class

• More on Functions
• Object Oriented Programming
• Exceptions
• Regular Expressions
• How to be a Python Ninja!