

LECTURE 2: INTRO TO PYTHON

LAST LECTURE

- Why scripting?
- Tradeoffs
 - Advantages vs disadvantages
- Use cases

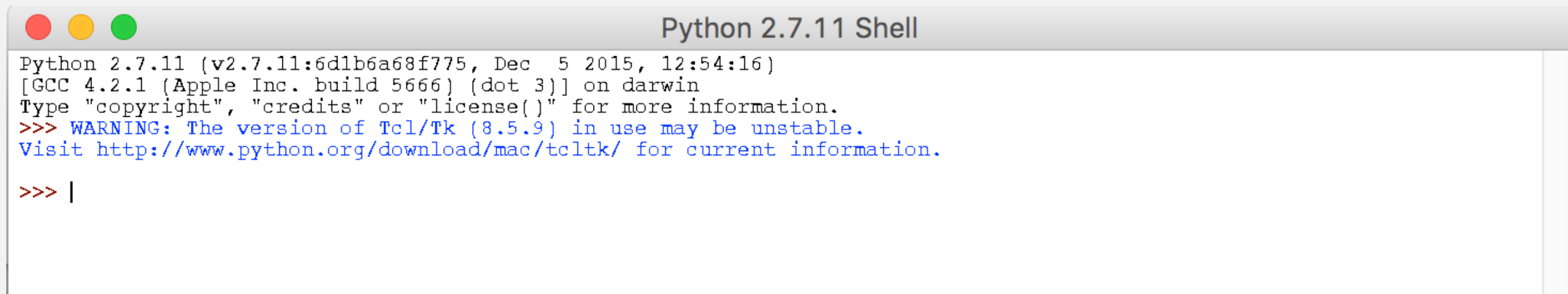
OUTLINE

- Invoking Python interpreter
 - Interactive mode
 - Normal mode
- Language basics
- Creating Python programs

IDLE

- On Unix, Mac: type “idle” in the command line
- On Windows: Find and execute “Idle”
- Idle has modes: Interactive and Normal
- Alternatives? vim, emacs(☹), PyCharm, Sublime Text, Atom...

INTERACTIVE MODE

A screenshot of a Python 2.7.11 Shell window. The window has a title bar with three colored buttons (red, yellow, green) on the left and the text "Python 2.7.11 Shell" on the right. The main content area shows the following text:

```
Python 2.7.11 (v2.7.11:6d1b6a68f775, Dec 5 2015, 12:54:16)
[GCC 4.2.1 (Apple Inc. build 5666) (dot 3)] on darwin
Type "copyright", "credits" or "license()" for more information.
>>> WARNING: The version of Tcl/Tk (8.5.9) in use may be unstable.
Visit http://www.python.org/download/mac/tcltk/ for current information.
>>> |
```

NORMAL MODE



```
print "hello world"
print "welcome to CSE/ISE337"
x = 3 + 5
y = x + y
|
```

WITHOUT USING IDLE

- Without IDLE, you can edit a Python program using any plain-text editor or IDE
- Many text editors also give syntax highlighting
- Then at operating system command line, type
 - `python3 your_script.py` without quotes to run it
- Running your program outside IDLE can easily take command line arguments.
 - e.g., `python3 hi.py argument`

LANGUAGE BASICS

- Data types
- Variables
- Operators
- Print statement
- Functions
- Modules and packages
- Strings, indexing and slicing
- Comments

DATA TYPES

- In Python, types are inferred (**Weakly typed**)
- Integer, float:
 - 1, 2.0, 3.333
- Boolean: True or False
 - `17 > 23` (False)
 - `True and False` (False)
- Strings
 - "hello", "hi there", "25"
- Other? lists, tuples, dictionaries...
- Functions, classes, modules...
- <https://docs.python.org/3.7/library/stdtypes.html>

VARIABLES

- To keep track of things, a name can be associated with a value, using an assignment statement
 - e.g., `id = 9`, the variable name `id` gets the value 9
 - A named value is often termed a variable, represents a quantity that can change over time
- Types are a property of values, not names, a name can store value of a new type by a new assignment
 - e.g., `id + 10`, `id = 'fred'`
- Names can be for variables, they can also be for functions, programs, modules, objects, etc.

OPERATORS

- Some operators:
 - + - * / ** // %<>>&| ^ ~<><=>= == !=
- Some operators work with strings too
 - “abc” + “def”
- Operator precedence
- “in” operator for strings, lists, tuples and dictionaries
 - “x” in “xyz”, 3 in (3,5,6)

PRINT STATEMENT

- Takes a series of values separated by commas, each value is converted into a string and then printed .Quotation marks are removed when printing
 - ```
print('answer is', 2+3)
```

```
answer is 5
```
- A '\n' character is written at the end, unless the print statement ends with a comma, which suppresses \n
- Escape characters
  - \n: newline, moves the output onto a new line
  - \t: tab, moves the output to the next tab stop
  - \\: print a literal backslash
  - ```
print("one\n two\t three")
```
- **print becomes a function in Python 3**

FUNCTIONS AND BUILT-IN FUNCTIONS

- A function is a block of code that often has a name and often is reusable
 - To call a function, give its name first, then a list of arguments in parentheses.
e.g., `abs(-3)`
- The arguments are evaluated first.
 - e.g., `abs(2-3*7)`
- Python has dozens of built-in functions
 - Example: `len('abc')` `len('hi'*4)` `min(5,6,7,2.0)` `ord('a')` `chr(67)` `str(42+2)`
- Other built-in functions:
<http://docs.python.org/3.7/library/functions.html>

KEYBOARD INPUT

- `input()` takes a prompt as argument, prints it, waits for user input, returns the value typed as string

- Example:

```
>>>a = input("Your Name: ")Your Name: Ali
>>>a
'Ali'
```

```
>>>a = input("Enter a number: ")Enter a number: 99
>>>a
'99'
```

MODULES AND DOT NOTATION

- Dot (or class) notation: used to qualify a name by the class or module in which it is defined
- Module: a python module is simply a Python source file. Can expose classes, functions, and global variables
- Use the import statement to use a module

```
>>>import math
>>>math.pi
>>>math.sin(math.pi/2.0)
>>>math.sqrt(1000)
```

Math is a module that defines a number of common mathematical functions

PYTHON PACKAGES

- Many modules are available with Python distribution, many
- more downloadable from the Python Package Index (PyPI)
- To list installed modules, type `help("modules")` at Python shell
- To learn what's in a particular module, say, the math module, type `help("math")`
- A Python package is simply a directory of Python module(s)
- `__init__.py` tells Python to treat this directory as a package
- E.g., given `my package/__init__.py` and `mypackage/mymodule.py`, we can import the package module using
- `import mypackage.mymodule`
- There are currently ~100K packages in PyPI

STRINGS AND INDEXING

- A string is formed by a succession of characters
 - Strings are for manipulating textual information
- Characters can be accessed using position, or index
 - Position denotes *#characters* from start of the string
 - Thus the first character is found at position 0
 - The second character is found at position 1, and so on
- Indexing: the operator used to access a character

```
>>> 'Python' [0]
```

```
>>> 'abc' [1]
```

STRING SLICING

- Slicing: operator to access a substring
A slice is a portion of the string starting at a given position up to but not including the ending position
- ```
>>>'realtor'[2:6]
'alto'
>>>'halter'[:4]
'halt'
>>>'realtor'[5:]
'or'
```
- Strings are **immutable** (You cannot change portions of a string)
- Indexing and slicing also used in list and dictionary

# COMMENTS

- Any text after a hash (#) in a line is ignored by Python interpreter
  - IDLE allows commenting of multiple lines (Alt+3), or uncomments (Alt+4)
    - control+3 on Mac
- When debugging, one may use a pair of triple quotes to comment a block of code

# CREATING PYTHON PROGRAMS

- Assignment, print, import statements so far
- More on Python statements
  - assignments
  - conditionals (if, elif, else)
  - indentation
  - loops (for, while, break, continue)

# ASSIGNMENTS

- An assignment statement associates a name with the result of evaluating an expression

- Same value assignment: several variables assigned the same value at once

```
a = b = c = 3.0
```

- Simultaneous assignment: >| variables separated by commas. Must have the same number of expressions on the right side, also separated by commas

```
sum, diff, prod = x + y, x - y, x * y
sum gets value of x+y, diff x-y, and prod x*y,
respectively
```

# ASSIGNMENTS

- A more subtle example

`x, y = y, x` # exchanges the values of variables x and y

- Note the difference from the two statements below

`x = y`

`y = x`

# CONDITIONALS

- A python program:

```
print('The exam had 40 points')
score = int(input("What was your score?"))
percent = 100 * score / 40.0
print('Your score in percentage was', percent)
if percent >= 90.0: # Note the syntax here, end with :
 print("Congrats, you got an A") #indentation is
significant
```

- If value is larger than or equal to 90, print a congrats message, otherwise, no statement is printed

# INDENTATION

- So far all statements have started in the first column
- In this program, the statement following the if statement is indented
- Indentation is the leading whitespace(space and tabs) at the beginning of a line
- Multiple statements of the same indentation belong to the same group
- There can be multiple levels of indentation



## ELSE STATEMENT

- ```
if percent >= 90.0:  
    print("Congrats, you got an A")  
    print("You are doing well in this class")  
print("see you in class next week")
```
- **If value is larger than 90, print two statements. In either case, print a see you in class statement.**

- ```
if percent >= 90.0:
 print("Congrats, you got an A")
 print("You are doing well in this class")
else:
 print("you did not get an A")
 print("see you in class next week")
```

**If value is higher than 89, print two statements. Otherwise, print one statement. Afterwards, in both cases, print a see you in class statement**

# NESTED CONDITIONALS

- If statements can be nested inside each other. We must
- **pay special attention to the indentation.** E.g.,

```
if percent >= 90.0:
 if percent >= 95.0:
 print('you get an A+!')
 else:
 print('you get an A')

if percent >= 90.0:
 if percent >= 95.0:
 print('you get an A+!')
else:
 print('you get an A')
```

- Any problem here?

# THE ELIF STATEMENT

- ```
print('The exam had 40 points')
score = int(input("What was your score?"))
percent = 100 * score / 40.0
print('Your score in percentage was', percent)

if percent >= 90.0:
    print('Congrats, you got an A')
elif percent >= 80.0:
    print('you got a B')
elif percent >= 70.0:
    print('you got a C')
else:
    print('your grade is lower than C')
```

FIN!

- Questions?