

Lecture 2

Strings, Functions, & Modules

String: Text as a Value

- String are quoted characters
 - 'abc d' (Python prefers)
 - "abc d" (most languages)
- How to write quotes in quotes?
 - Delineate with “other quote”
 - **Example:** " ' " or ' " '
 - What if need both " and ' ?
- **Solution:** escape characters
 - Format: \ + letter
 - Special or invisible chars

Type: str

Char	Meaning
\'	single quote
\"	double quote
\n	new line
\t	tab
\\	backslash

String are Indexed

- `s = 'abc d'`

0	1	2	3	4
a	b	c		d

- `s = 'Hello all'`

0	1	2	3	4	5	6	7	8
H	e	l	l	o		a	l	l

- Access characters with `[]`

- `s[0]` is 'a'
- `s[4]` is 'd'
- `s[5]` **causes an error**
- `s[0:2]` is 'ab' (excludes c)
- `s[2:]` is 'c d'

- What is `s[3:6]`?

A: 'lo a'
B: 'lo'
C: 'lo '
D: 'o '
E: I do not know

- Called “string slicing”

String are Indexed

- `s = 'abc d'`

0	1	2	3	4
a	b	c		d

- `s = 'Hello all'`

0	1	2	3	4	5	6	7	8
H	e	l	l	o		a	l	l

- Access characters with `[]`

- `s[0]` is 'a'
- `s[4]` is 'd'
- `s[5]` **causes an error**
- `s[0:2]` is 'ab' (excludes c)
- `s[2:]` is 'c d'

- What is `s[3:6]`?

A: 'lo a'
B: 'lo'
C: 'lo ' **CORRECT**
D: 'o '
E: I do not know

- Called “string slicing”

String are Indexed

- `s = 'abc d'`

0	1	2	3	4
a	b	c		d

- `s = 'Hello all'`

0	1	2	3	4	5	6	7	8
H	e	l	l	o		a	l	l

- Access characters with `[]`

- `s[0]` is 'a'
- `s[4]` is 'd'
- `s[5]` **causes an error**
- `s[0:2]` is 'ab' (excludes c)
- `s[2:]` is 'c d'

- What is `s[:4]`?

A: 'o all'
B: 'Hello'
C: 'Hell'
D: **Error!**
E: I do not know

- Called “string slicing”

String are Indexed

- `s = 'abc d'`

0	1	2	3	4
a	b	c		d

- `s = 'Hello all'`

0	1	2	3	4	5	6	7	8
H	e	l	l	o		a	l	l

- Access characters with `[]`

- `s[0]` is 'a'
- `s[4]` is 'd'
- `s[5]` **causes an error**
- `s[0:2]` is 'ab' (excludes c)
- `s[2:]` is 'c d'

- What is `s[:4]`?

A: 'o all'
B: 'Hello'
C: 'Hell' **CORRECT**
D: **Error!**
E: I do not know

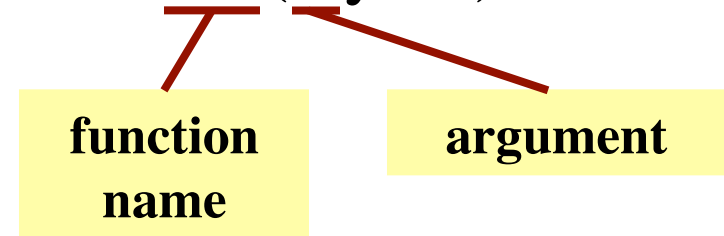
- Called “string slicing”

Other Things We Can Do With Strings

- **Operation** `in`: `s1 in s2`
 - Tests if `s1` “a part of” `s2`
 - Say `s1` a *substring* of `s2`
 - Evaluates to a bool
- **Examples:**
 - `s = 'abracadabra'`
 - `'a' in s == True`
 - `'cad' in s == True`
 - `'foo' in s == False`
- **Function** `len`: `len(s)`
 - Value is # of chars in `s`
 - Evaluates to an int
- **Examples:**
 - `s = 'abracadabra'`
 - `len(s) == 11`
 - `len(s[1:5]) == 4`
 - `s[1:len(s)-1] == 'bracadabr'`

Function Calls

- Python supports expressions with math-like functions
 - A function in an expression is a **function call**
 - Will explain the meaning of this later
- Function expressions have the form **fun**(x,y,...)



- **Examples** (math functions that work in Python):
 - `round(2.34)`
 - `max(a+3,24)`

Arguments can be
any **expression**

Built-In Functions

- You have seen many functions already
 - Type casting functions: `int()`, `float()`, `bool()`
 - Dynamically type an expression: `type()`
 - Help function: `help()`
- Getting user input: `raw_input()`
- `print <string>` is **not** a function call
 - It is simply a statement (like assignment)
 - But it is in Python 3.x: `print(<string>)`

Arguments go in (),
but `name()` refers to
function in general

Method: A Special Type of Function

- Methods are unique (right now) to strings
- Like a function call with a “string in front”
 - Usage: *string.method*(x,y...)
 - The string is an *implicit argument*
- Example: upper()
 - `s = 'Hello World'`
 - `s.upper() == 'HELLO WORLD'`
 - `s[1:5].upper() == 'ELLO'`
 - `'abc'.upper() == 'ABC'`

Will see why we
do it this way
later in course

Examples of String Methods

- `s1.index(s2)`
 - Position of the first instance of `s2` in `s1`
 - `s1.count(s2)`
 - Number of times `s2` appears inside of `s1`
 - `s.strip()`
 - A copy of `s` with white-space removed at ends
- `s = 'abracadabra'`
 - `s.index('a') == 0`
 - `s.index('rac') == 2`
 - `s.count('a') == 5`
 - `' a b '.strip() == 'a b'`

See Python
Docs for more

Built-in Functions vs Modules

- The number of built-in functions is small
 - <http://docs.python.org/2/library/functions.html>
- Missing a lot of functions you would expect
 - **Example:** `cos()`, `sqrt()`
- **Module:** file that contains Python code
 - A way for Python to provide optional functions
 - To access a module, the `import` command
 - Access the functions using module as a *prefix*

Example: Module math

```
>>> import math
```

To access math functions

```
>>> math.cos(0)
```

```
1.0
```

```
>>> cos(0)
```

Functions require math prefix!

```
Traceback (most recent call last):
```

```
File "<stdin>", line 1, in <module>
```

```
NameError: name 'cos' is not defined
```

```
>>> math.pi
```

Module has variables too!

```
3.141592653589793
```

```
>>> math.cos(math.pi)
```

```
-1.0
```

Example: Module math

```
>>> import math
```

To access math functions

```
>>> math.cos(0)
```

```
1.0
```

```
>>> cos(0)
```

Functions require math prefix!

```
Traceback (most recent call last):
```

```
File "<stdin>", line 1, in <module>
```

```
NameError: name 'cos' is not defined
```

```
>>> math.pi
```

Module has variables too!

```
3.141592653589793
```

```
>>> math.cos(math.pi)
```

```
-1.0
```

Other Modules

- `io`
 - Read/write from files
- `random`
 - Generate random numbers
 - Can pick any distribution
- `string`
 - Useful string functions
- `sys`
 - Information about your OS

Reading the Python Documentation

The screenshot shows the Python 2.7.3 documentation page for the `math` module. The page title is "9.2. math — Mathematical functions". The left sidebar contains a "Table Of Contents" for the `math` module, listing sections like "9.2.1. Number-theoretic and representation functions", "9.2.2. Power and logarithmic functions", "9.2.3. Trigonometric functions", "9.2.4. Angular conversion", "9.2.5. Hyperbolic functions", and "9.2.6. Special functions". The main content area describes the `math` module and lists functions provided by the module. Callouts point to specific parts of the page:

- Function name**: Points to the function name `math.ceil(x)`.
- Possible arguments**: Points to the argument `x` in the function signature `math.ceil(x)`.
- Module**: Points to the module name `math` in the function signature `math.ceil(x)`.
- What the function evaluates to**: Points to the description of the function's return value: "Return the ceiling of `x` as a float, the smallest integer value greater than or equal to `x`."

The screenshot also shows the URL `http://docs.python.org/library` in the address bar and a search bar at the bottom left.

Using the from Keyword

```
>>> import math
```

```
>>> math.pi
```

Must prefix with
module name

```
3.141592653589793
```

```
>>> from math import pi
```

```
>>> pi
```

No prefix needed
for variable pi

```
3.141592653589793
```

```
>>> from math import *
```

```
>>> cos(pi)
```

```
-1.0
```

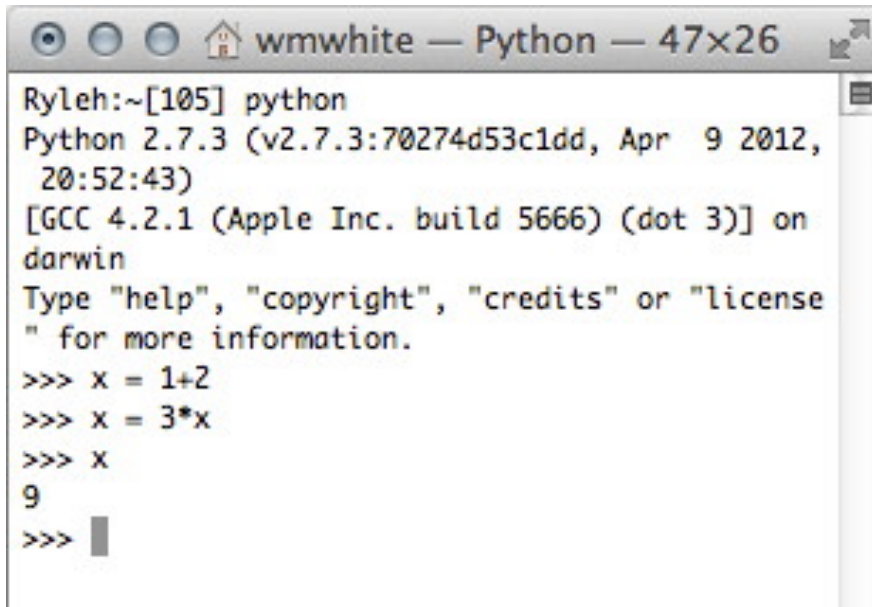
No prefix needed
for anything in math

- Be careful using from!
- Namespaces are *safer*
 - Modules might conflict (functions w/ same name)
 - What if import both?
- **Example:** Turtles
 - Use in Assignment 4
 - 2 modules: turtle, tkturtle
 - Both have func. Turtle()

How Do We Make Our Own Modules?

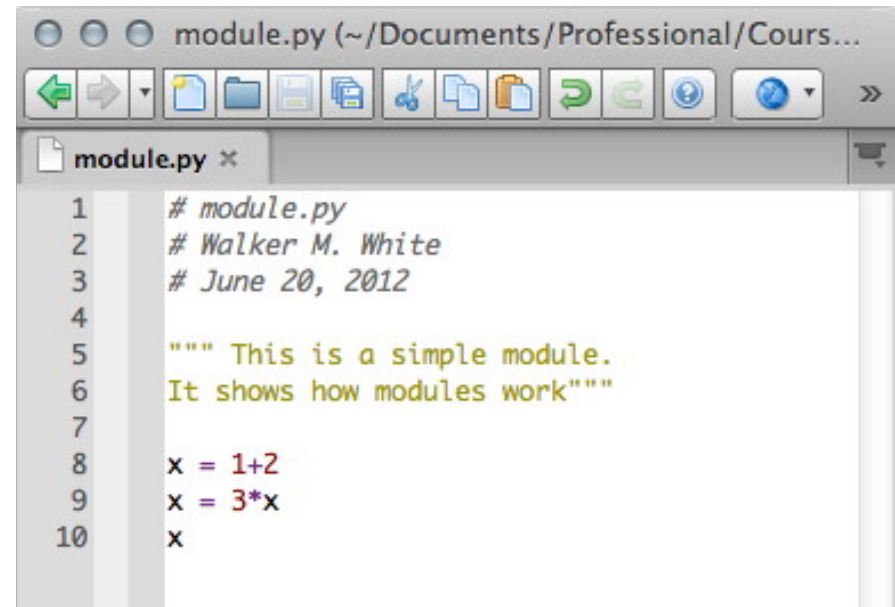
- Modules provide extra functions, variables
 - **Example:** math provides `math.cos()`, `math.pi`
- We might want to make our own
 - Custom scientific functions
 - Specialized scientific constants
- This requires **two different** programs
 - **Komodo Edit** to *make* a module
 - **Python** to *use* the module

Python Shell vs. Modules



```
Ryleh:~[105] python
Python 2.7.3 (v2.7.3:70274d53c1dd, Apr  9 2012,
20:52:43)
[GCC 4.2.1 (Apple Inc. build 5666) (dot 3)] on
darwin
Type "help", "copyright", "credits" or "license
" for more information.
>>> x = 1+2
>>> x = 3*x
>>> x
9
>>>
```

- Launch in command line
- Type each line separately
- Python executes as you type



```
1  # module.py
2  # Walker M. White
3  # June 20, 2012
4
5  """ This is a simple module.
6  It shows how modules work"""
7
8  x = 1+2
9  x = 3*x
10 x
```

- **Write in a text editor**
 - We use Komodo Edit
 - But anything will work
- Run module with import

Creating a Module

Module Contents

module.py

Single line comment
(not executed)

""" This is a simple module.
It shows how modules work"""

Docstring (note the Triple Quotes)
Acts as a multiple-line comment
Useful for *code documentation*

x = 1+2

x = 3*x

Commands
Executed on import

x

Not a command.
import **ignores this**

Creating a Module

Module Contents

```
# module.py
```

```
""" This is a simple module.  
It shows how modules work"""
```

```
x = 1+2
```

```
x = 3*x
```

```
x
```

“**Module data**” must be
prefixed by module name

Prints **docstring** and
module contents

Python Shell

```
>>> import module
```

```
>>> x
```

Traceback (most recent call last):

File "<stdin>", line 1, in <module>

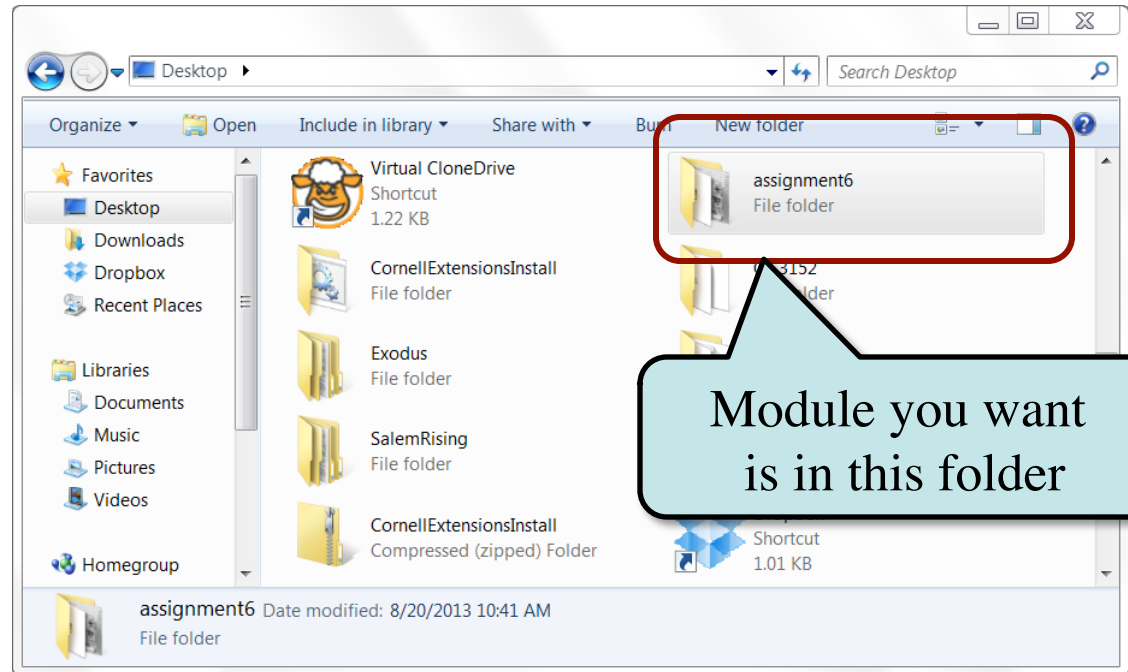
NameError: name 'x' is not defined

```
>>> module.x
```

```
9
```

```
>>> help(module)
```

Modules Must be in Working Directory!



Modules Must be in Working Directory!

Have to navigate to folder **BEFORE** running Python

Module you want is in this folder