Variables and Types

- Python is a **dynamically typed language**
  - Variables can hold values of any type
  - Type of value in variable can change over time
- The following is acceptable in Python:

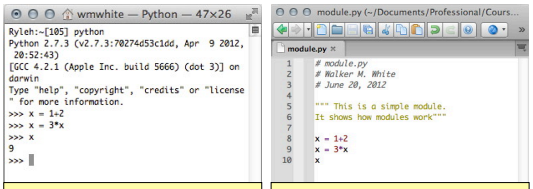
  >>> x = 1 ← x contains an **int** value

  >>> x = x / 2.0 ← x contains a **float** value (why?)
- Alternative is a **statically typed language**
  - Each variable restricted to values of just one type
  - Java is an example of such a language

Dynamic Typing

- Often want to track the type in a variable
  - What is the result of evaluating x / y?
  - Depends on whether x, y are **int** or **float** values
- Use expression type(<expression>) to get type
  - type(2) evaluates to <type 'int'>
  - type(x) evaluates to type of contents of x
- Can use in a boolean expression to test type
  - type("abc") == str evaluates to **True**

Python Shell vs. Modules



- The interactive shell
  - Simple to use
  - Experience with Lab 1
- But very inefficient
  - One command at a time
  - What if need 1000+ lines?

- Alternative: **modules**
  - Files with commands
  - Write in a special editor
- Run module with import
  - Loads the file into Python
  - Executes each line

Using a Module

Module Contents

```
# module.py

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

x = 1+2
x = 3*x
x
```

Single line comment (not executed)

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

Commands executed on import

Not a command Does nothing

Using 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)
```

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,…)

function name **argument**

- Simplest example of functions are in module math

```
>>> import math
>>> math.sin(math.pi)
```

module variable

module function

Using the from Keyword

```
>>> import math
>>> math.pi
3.141592653589793
>>> from math import pi
>>> pi
3.141592653589793
>>> from math import *
>>> cos(pi)
-1.0
```

Must prefix with module name

No prefix needed for variable pi

No prefix needed for anything in math

- Be careful using from!
- Modules are **namespaces**
  - There is only one variable or function for each name
  - Other modules may reuse names for variable/function
  - Prefix keeps them distinct
- **Example**: badpi.py

How Well Are You Following?

Module Contents	Python Shell

# data.py 	
 	
""" Module with two variables """
 	
x = 4
y = 3
	>>> x = 1
	>>> y = 2
	>>> from data import x
	>>> x+y
	???

A: 3
B: 7
C: 6
D: 4
E: I do not know

How Well Are You Following?

Module Contents	Python Shell

# data.py 	
 	
""" Module with two variables """
 	
x = 4
y = 3
	>>> from data import *
	>>> x = 3
	>>> from data import x
	>>> x+y
	???

A: 6
B: 7
C: Error!
D: I do not know

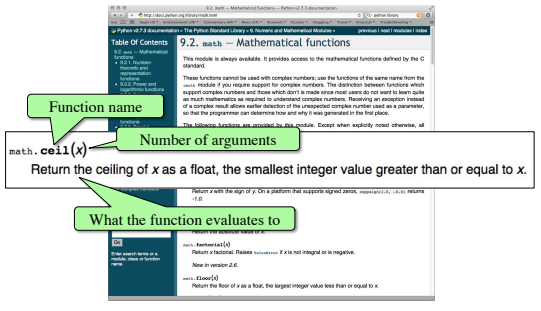
Importing a variable "clobbers" any existing variable of same name

Python Comes with Many Modules

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

- Complete list:
- http://docs.python.org/library
- **Library**: built-in modules
  - May change each release
  - Why version #s are an issue
- Documentation is the **API**
  - **A**pplication
  - **P**rogramming
  - **I**nterface
- **Interface**: *specification* of the functions and data in a module

Reading the Python API



The Komodo Editor