Beyond Blocks: Python Session #2

CS10 Spring 2013
May 7, 2013
Michael Ball

Beyond Blocks: Python #2

Where to find Information

- **Python.org**: www.python.org
- **Python Docs**: www.python.org/doc/
- **Python Modules**: docs.python.org/modindex.html

Beyond Blocks: Python #2

Using Files

```
$ python3 -i file.py
```

--Allows you to use an interpreter

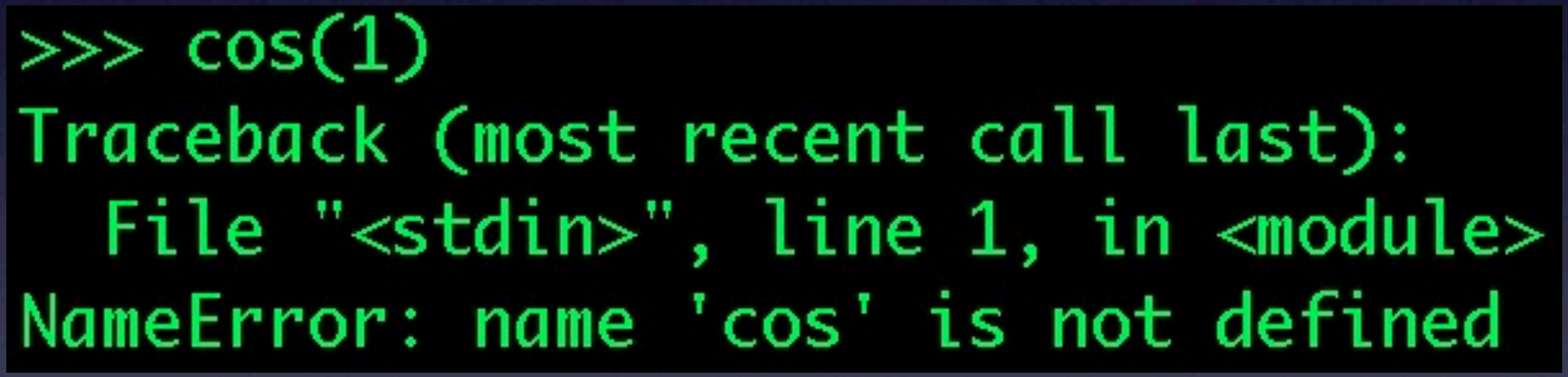
```
$ python3 file.py
```

-- Simply runs the file.

(Files need not actually say `.py`**; but it's cleaner if they do)**

BYOB ⟷ Python

Importing



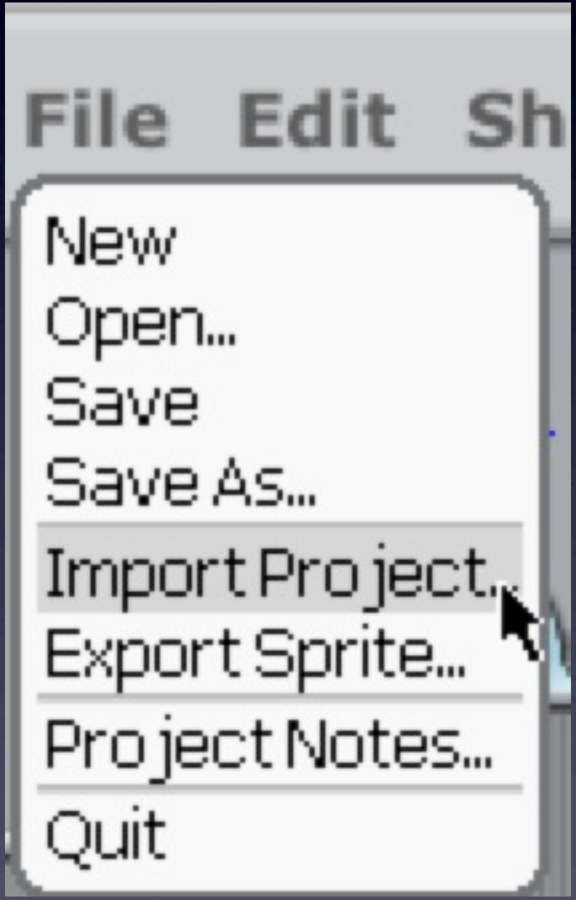
```
>>> cos(1)
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
NameError: name 'cos' is not defined
```

ERROR!

Hmmmm....

BYOB ⟷ Python

Importing



```
>>> import math
```

“math” module

Beyond Blocks: Python #2

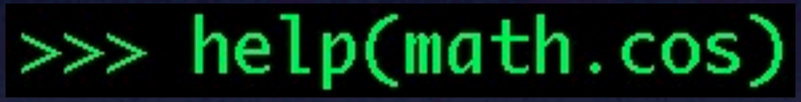
Importing

```
>>> import math
>>> math.cos(1)
0.5403023058681398
>>> from math import cos
>>> cos(1)
0.5403023058681398
>>> math.cos(1)
0.5403023058681398
>>>
```

module.function(args)

Beyond Blocks: Python #2

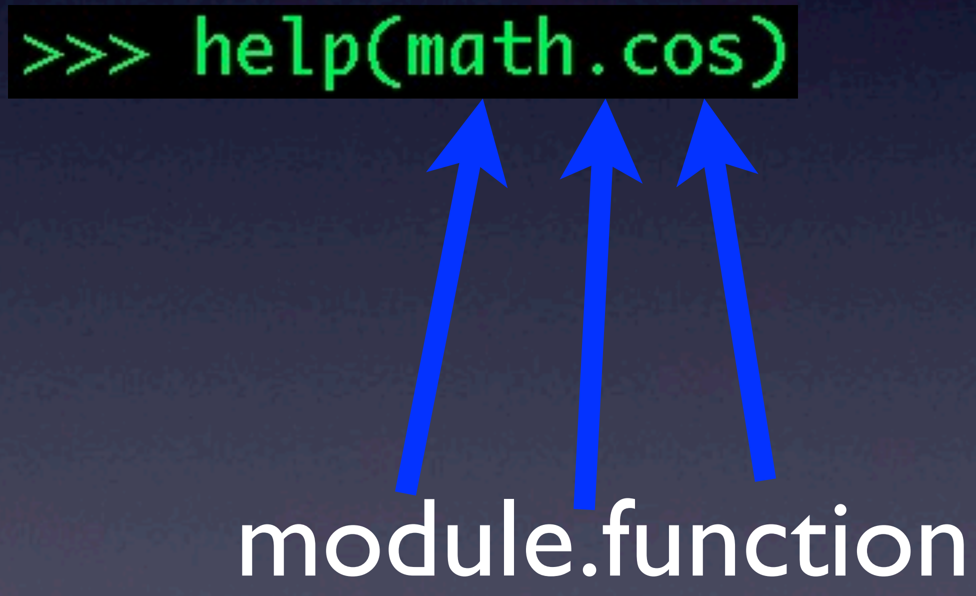
Importing, help!



```
>>> help(math.cos)
```

Beyond Blocks: Python #2

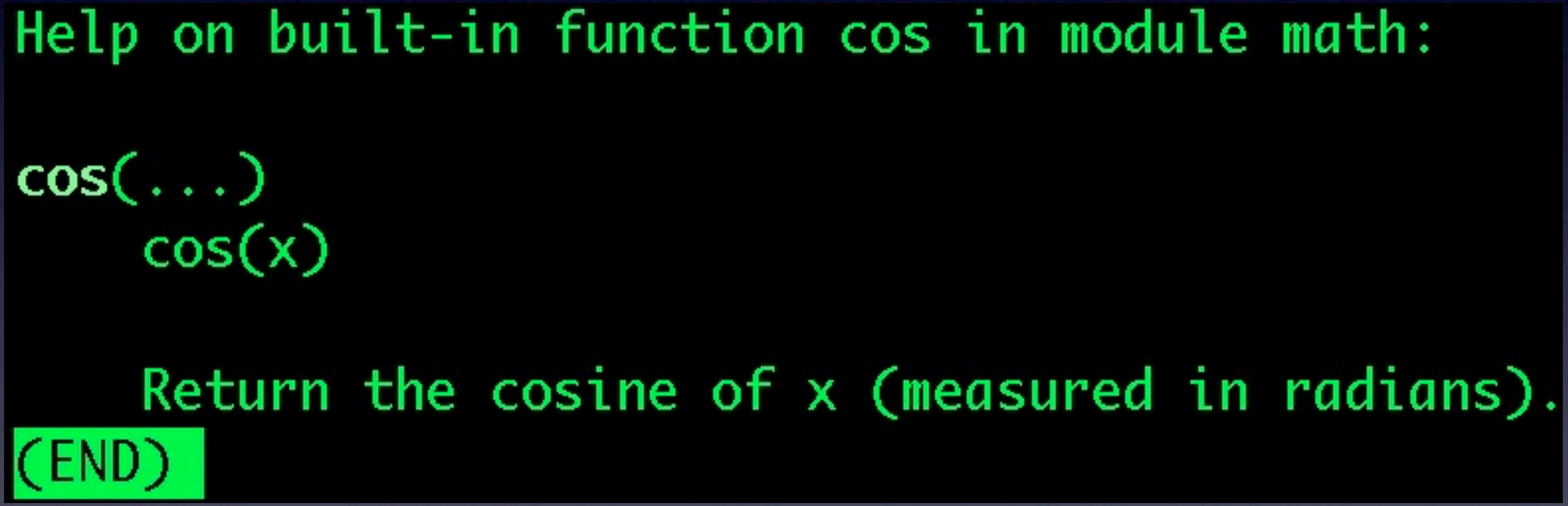
Importing, help!



```
>>> help(math.cos)
```

Beyond Blocks: Python #2

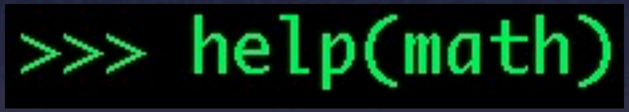
Importing, help!



```
Help on built-in function cos in module math:

cos(...)
    cos(x)

    Return the cosine of x (measured in radians).
(END)
```

Beyond Blocks: Python #2

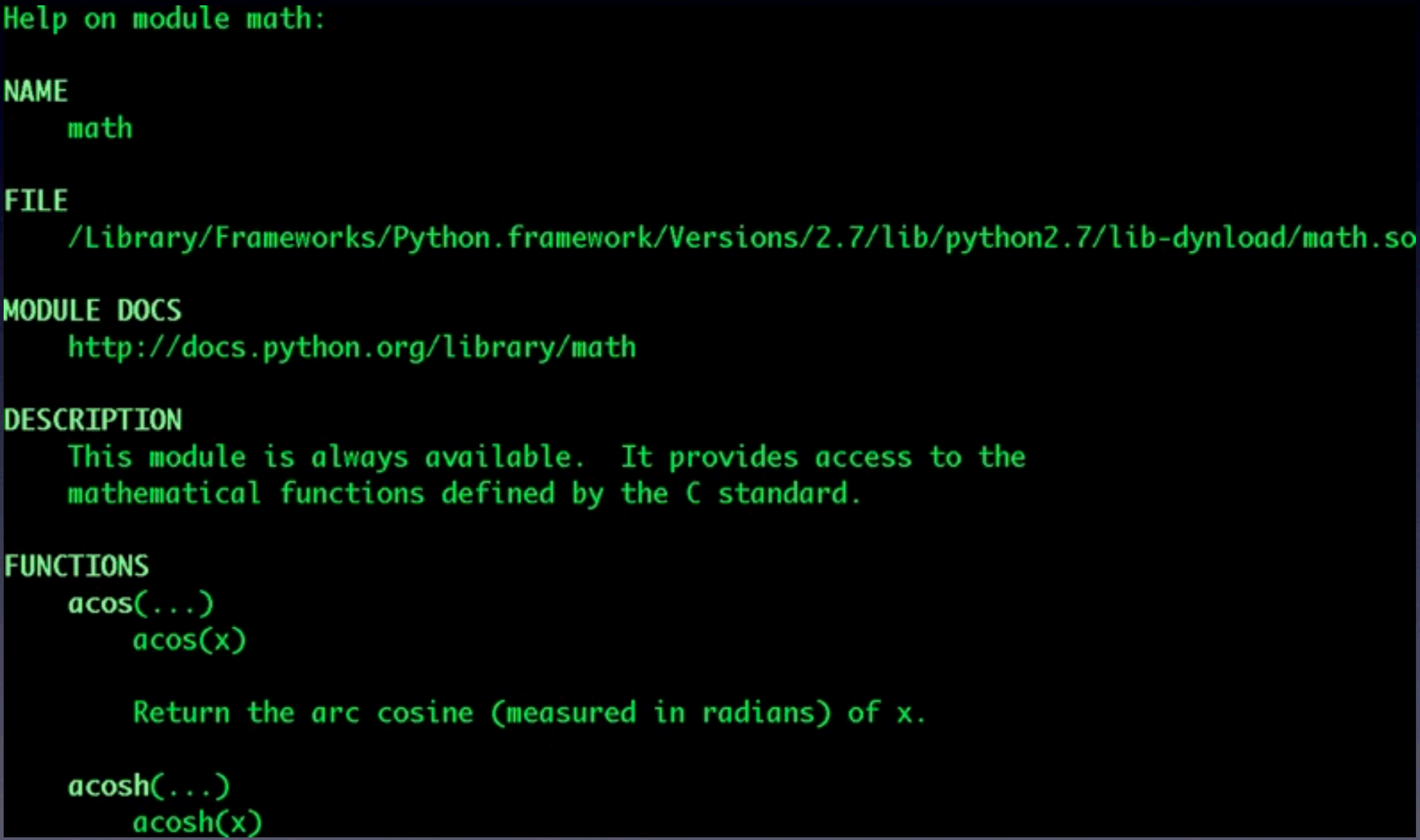
Help!



```
>>> help(math)
```

Beyond Blocks: Python #2

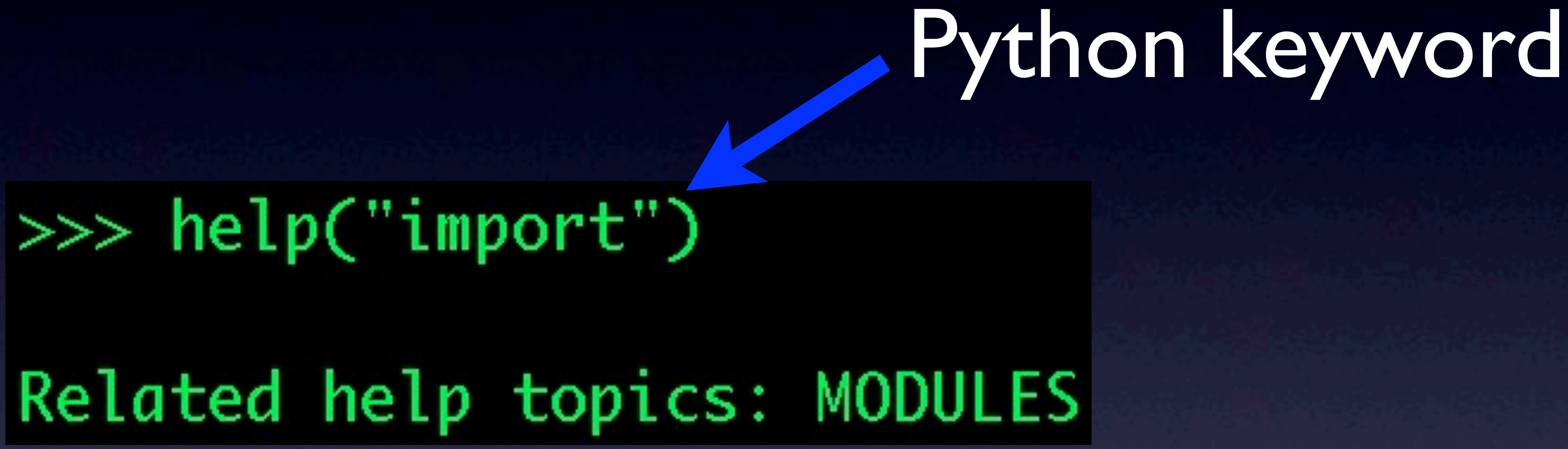
Help!



```
Help on module math:

NAME
    math

FILE
    /Library/Frameworks/Python.framework/Versions/2.7/lib/python2.7/lib-dynload/math.so

MODULE DOCS
    http://docs.python.org/library/math

DESCRIPTION
    This module is always available.  It provides access to the
    mathematical functions defined by the C standard.

FUNCTIONS
    acos(...)
        acos(x)

        Return the arc cosine (measured in radians) of x.

    acosh(...)
        acosh(x)
```

Beyond Blocks: Python #2

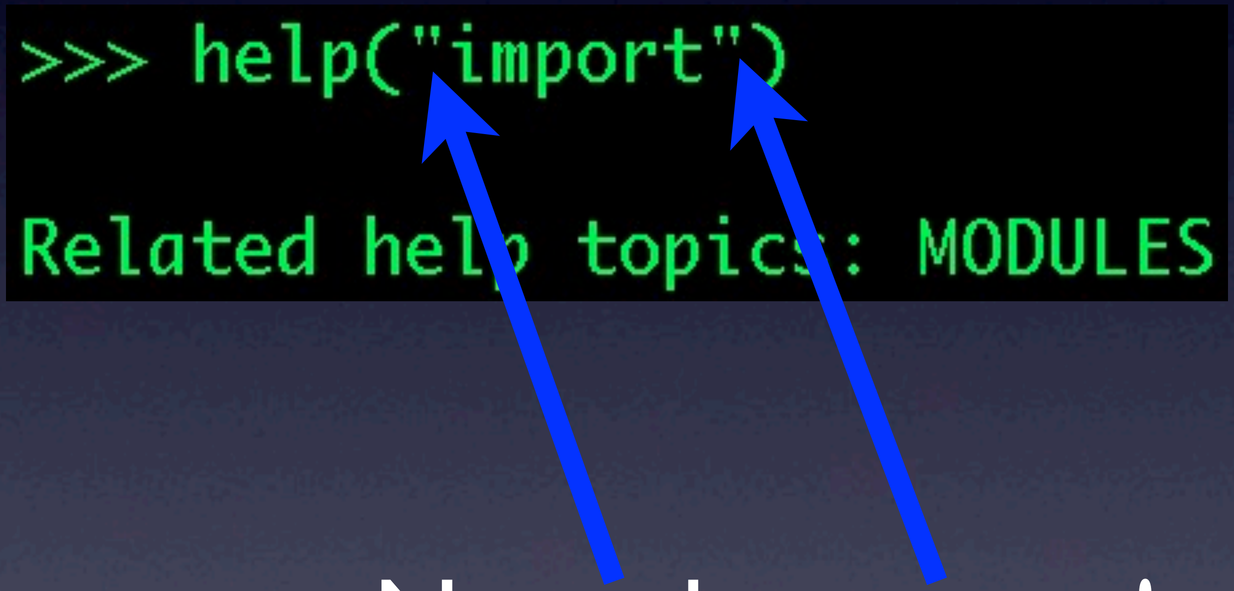
Help!



```
>>> help("import")

Related help topics: MODULES
```

Beyond Blocks: Python #2

Help!

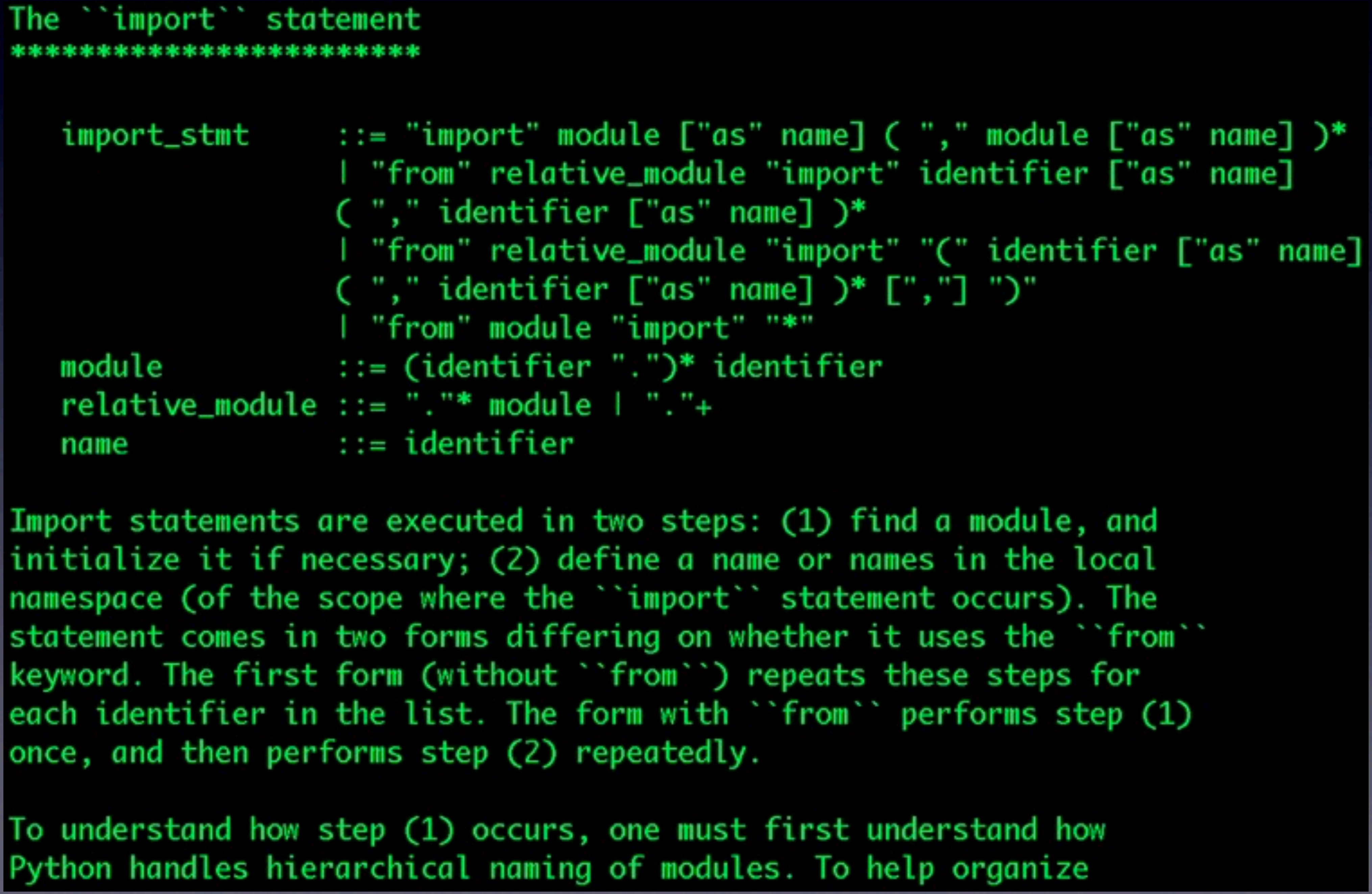


```
>>> help("import")

Related help topics: MODULES
```

Note the quotes!

Beyond Blocks: Python #2

Help!



```
The ``import`` statement
************************


   import_stmt     ::= "import" module ["as" name] ( "," module ["as" name] )*
                   | "from" relative_module "import" identifier ["as" name]
                   ( "," identifier ["as" name] )*
                   | "from" relative_module "import" "(" identifier ["as" name]
                   ( "," identifier ["as" name] )* [","] ")"
                   | "from" module "import" "*"
   module          ::= (identifier ".")* identifier
   relative_module ::= "."* module | "."+
   name            ::= identifier

Import statements are executed in two steps: (1) find a module, and
initialize it if necessary; (2) define a name or names in the local
namespace (of the scope where the ``import`` statement occurs). The
statement comes in two forms differing on whether it uses the ``from``
keyword. The first form (without ``from``) repeats these steps for
each identifier in the list. The form with ``from`` performs step (1)
once, and then performs step (2) repeatedly.

To understand how step (1) occurs, one must first understand how
Python handles hierarchical naming of modules. To help organize
```

Data Structures (overview)

- Sequences
  - Iterators
  - Operators
- Dictionaries
  - Hash Tables

Review

- Typing, built-in types
- Variables
- Looping and conditionals
- Functions
- Recursion

Review++

- Typing, built-in types
- Variables
- Looping and conditionals
- Functions
- Recursion
- This week's content
  - Strings and string operators
  - Lists, Dictionaries, etc.

Typing, (Some) Built-In Types

- Numeric types
  - <int>, <float>, <long>
- Sequence types
  - <str>, <unicode>, <list>, <tuple>, <range>
- New: Collection types
  - <set>, <frozenset>, <dict>

Variables

- Simple assignments
- Multiple assignments
- “Mutable” vs. “Immutable”
  - We’ll see more of these as examples

Looping and Conditionals

- While loops
- If statements with boolean comparisons
  - Parenthetical evaluation
  - `or, and, not, <, <=, >, >=, ==,` `is`, `is not`

Looping and Conditionals

- While loops
- If statements with boolean comparisons
  - Parenthetical evaluation
  - or, and, not, <, <=, >, >=, ==, =, is, is not
- For loops (e.g. “for x in range(0,10):”)
  - We’ll talk more about ranges later...

Recursion

- Recursion in Python is like recursion in BYOB
- Factorial(n)?
- IsPalindrome(word)?
  - IsPalindrome is left as an exercise for you!

Sequences (overview)

- Str “”
- List []
- Tuple ()
- Range

Sequences (overview)

- Str “” - immutable
- List [] - mutable
- Tuple () - immutable
- Range - mutable-ish

Strings & String Operators

- Sequence (or “list” or “array”) of chars
- Quoting
  - Single vs. double vs. triple and mixing
    - Triple is 3 double quotes. “”””
- Printing
  - Formatted and unformatted
- Concatenation, finding length, etc.
  - help(“string”)
- Slicing and slicing notation [::]
- http://docs.python.org/library/stdtypes.html#string-methods

Lists

- Collection of *any* type
    - Including itself!
- Indexing (BYOB: Item () of [])
- Modifying (Replace item () of [] with ())
- Slicing and slicing notation (i.e., [::])
    - Exactly the same as string notation!
- Operators
    - append(x), insert(i,x), count(x), sort(), etc.
- http://docs.python.org/library/stdtypes.html#mutable-sequence-types

Tuples (|ˈtjuːp(ə)l| :)

- Immutable
  - Same as strings
- Also contains *any* type of element(s).
- Syntax: ( )
- What are the advantages of using them?
  - Faster and “Safer,”
  - Can be used as Dictionary keys
    - More on dictionaries later...

Ranges

- Range syntax (start, stop, step)
  - Start: Inclusive; stop: exclusive
  - Results in an iterable object
  - `list(range(x))` is a list.
    - `range(start, stop)` or `range(stop)` also work.
    - Default start is 0, Default step is 1.
- http://docs.python.org/library/stdtypes.html#xrange-type

Iterators

- Syntax
  - i = iter(object)
- Usage
  - next(i) #In Python3!
  - Python 2.x: i.next()
- Why does Python have them?
  - You’ll see...
- http://docs.python.org/library/stdtypes.html#iterator-types

Sequence (general) Operators

- `X in` & `not in Y`
- `+` & `*`
- slice [::]
- len()
- min() & max()
- even map() filter() & reduce() !
- Many, many more:
    - http://docs.python.org/library/stdtypes.html#typesseq

Sets

- NO duplicate members (unique)
- Unordered
- Syntax: `set([1,2,3,4])` or `set(“blah”)`
- NO array-like indexing (e.g., s[0])
  - Iterators are used instead...
- Faster (for large number of entries)

Set Operators

- `len(s)`
- `s.add(elem)`
- `X in` & `not in S`
- remove & pop & -
- Iteration
- Union, intersection, isdisjoint, etc.
- Much, much more:
  - `help("set")`
  - http://docs.python.org/library/stdtypes.html#set

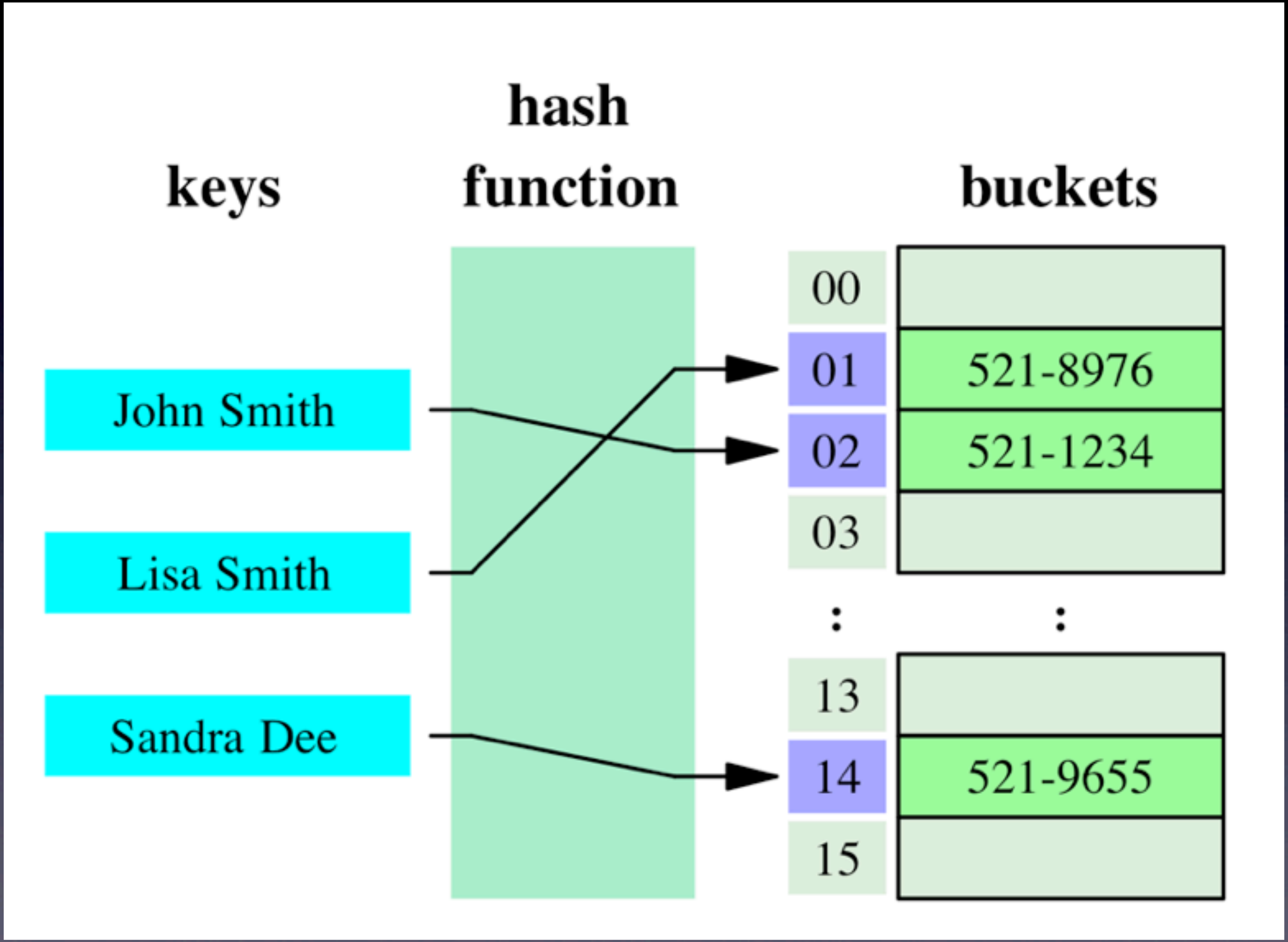
Dictionaries

- Syntax
  - {key:value}
- Adding elements
  - dict[key]=value
- Accessing elements
  - dict[key]
- Keys
  - Looking for specific keys (has_key() & "in")
  - Iterating over (iterkeys())
- http://docs.python.org/library/stdtypes.html#dict

How Do Dictionaries Work, and Why Use Them?

- Hash table based
  - Hash codes & array indexes
- Very fast look-up time (i.e., O(1))
- Classic trade-off:
  - Speed and space

Dictionaries = Hash



http://en.wikipedia.org/wiki/File:Hash_table_3_1_1_0_1_0_0_SP.svg