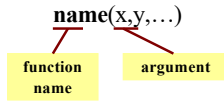


Function Calls

- Python supports expressions with math-like functions
 - A function in an expression is a *function call*

- Function calls** have the form



- Arguments** are
 - Expressions**, not values
 - Separated by commas
- Examples:**
 - `round(2.34)`
 - `max(a+3, 24)`

1

Built-in Functions vs Modules

- The number of built-in functions is small
 - <http://docs.python.org/3/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*

2

Example: Module math

```

>>> import math
>>> math.cos(0)
1.0
>>> cos(0)
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
NameError: name 'cos' is not defined
>>> math.pi
3.141592653589793
>>> math.cos(math.pi)
-1.0
    
```

To access math functions

Functions require math prefix!

Module has variables too!

Other Modules

- os**
 - Information about your OS
 - Cross-platform features
- random**
 - Generate random numbers
 - Can pick any distribution
- intros**
 - Custom module for the course
 - Will be used a lot at start

3

Reading the Python Documentation

Function name: `math.ceil(x)`

Possible arguments: `x`

Module: `math`

What the function evaluates to: Return the ceiling of `x`, the smallest integer greater than or equal to `x`.

<http://docs.python.org/3/library>

4

Interactive Shell vs. Modules

```

[wmw@r1yeh]~$ python
Python 3.6.5 (Anaconda, Inc.) (default, Apr 26 2018,
08:42:37)
[GCC 4.2.1 Compatible Clang 4.0.1 (tags/RELEASE_401/
final)] on darwin
Type "help", "copyright", "credits" or "license()" for
more information.
>>> x = 1+2
>>> x = 3*x
>>> x
6
>>>
    
```

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

```

1 # A simple module.
2
3 This file shows how modules work.
4
5 Author: Walker M. White (wmw2)
6 Date: August 25, 2017 (Python 3 Version)
7
8
9
10 x = 1+2 # I am a comment
11 x = 3*x
12
    
```

- Write in a code editor**
 - We use Pulsar
 - But anything will work
- Load module with `import`

5

Using a Module

Module Contents

```

""" A simple module.

This file shows how modules work

"""

# This is a comment
x = 1+2
x = 3*x
x
    
```

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

Single line comment
(not executed)

Commands
Executed on import

Not a command.
`import` ignores this

6

Using a Module

Module Contents

```
""" A simple module.

This file shows how modules work
"""

# This is a comment
x = 1+2
x = 3*x
x
```

"Module data" must be prefixed by module name

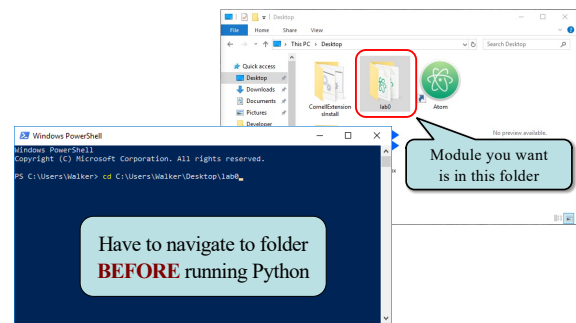
Prints **docstring** and module contents

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)
```

7

Modules Must be in Working Directory!



8

Modules vs. Scripts

Module

- Provides functions, variables
 - Example:** temp.py
- import it into Python shell


```
>>> import temp
>>> temp.to_fahrenheit(100)
212.0
>>>
```

Script

- Behaves like an application
 - Example:** helloApp.py
- Run it from command line:


```
python helloApp.y
```



Files look the same. Difference is how you use them.

9

Scripts and Print Statements

module.py

```
""" A simple module.

This file shows how modules work
"""

# This is a comment
x = 1+2
x = 3*x
x
```

script.py

```
""" A simple script.

This file shows why we use print
"""

# This is a comment
x = 1+2
x = 3*x
print(x)
```



10

User Input

```
>>> input("Type something")
Type somethingabc
'abc'
>>> input('Type something: ')
Type something: abc
'abc'
>>> x = input("Type something: ")
Type something: abc
>>> x
'abc'
```

No space after the prompt.

Proper space after prompt.

Assign result to variable.

11

Numeric Input

- input returns a string
 - Even if looks like int
 - It cannot know better
- You must convert values
 - int(), float(), bool(), etc.
 - Error if cannot convert
- One way to program
 - But it is a *bad* way
 - Cannot be automated

```
>>> x = input('Number: ')
Number: 3
>>> x
'3'
>>> x + 1
TypeError: must be str, not int
>>> x = int(x)
>>> x+1
4
```

Value is a string.

Must convert to int.

12