

CSC 280 Introduction to Computer Science: Programming with Python

Lecture 4 : Functions

September 10, 2015

Prof. Bei Xiao

Fall, 2015

American University

Outline

- Function Calls
- Built-in function
- Modules, Math Module
- `from` keyword
- Function definitions
- Parameters and arguments

Take home reading

- Read Chapter 3.1-6.
- Python Math Functions:
- <https://docs.python.org/2/library/math.html>

Function calls

- Python supports expressions with math-like functions
 - A function in an expression is called function call
- Function expressions have the form `fun(x,y...)`



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

Function calls

- Python supports expressions with math-like functions
 - A function in an expression is called function call
- Function expressions have the form `fun(x,y...)`

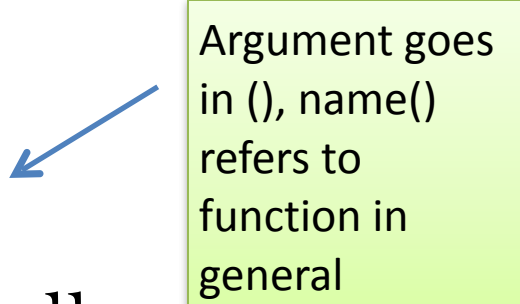


- Examples (math functions that work with Python)
 - `round(2.34)`
 - `max(a+3,24)`

An argument can be an 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 functional call.
 - It is simply a statement (like assignment)
 - But it is in Python 3



Argument goes
in (), name()
refers to
function in
general

Built-In Functions vs. Module

- The number of built-in functions is small
 - <https://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 *prefix*

Example: Module math

```
>>> import math
```

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

```
1.0
```

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

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

```
  File "<pyshell#2>", line 1, in <module>
```

```
    cos(0)
```

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

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

```
3.141592653589793
```

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

```
-1.0
```


Example: Module math

```
>>> import math
```

To access math functions

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

Functions require math prefix!

```
1.0
```

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

Traceback (most recent call last):

```
File "<pyshell#2>", line 1, in <module>
```

```
cos(0)
```

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

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

```
3.141592653589793
```

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

```
-1.0
```

Example: Module math

```
>>> import math
```

To access math functions

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

Functions require math prefix!

```
1.0
```

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

Traceback (most recent call last):

```
File "<pyshell#2>", line 1, in <module>
```

```
cos(0)
```

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

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

Module has variables too.

```
3.141592653589793
```

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

```
-1.0
```

Using the **from** keyword

```
>>>import math
```

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

```
3.141592653589793
```

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

```
-1.0
```

```
>>> 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 variable pi

- Be careful using from
- Using import is *safer*
 - Modules might conflict
 - (functions w/ same name)

Using the **from** keyword

```
>>>import math
```

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

```
3.141592653589793
```

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

```
-1.0
```

```
>>> 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 variable pi

- Be careful using from
- Using import is *safer*
 - Modules might conflict
 - (functions w/ same name)

Composition

- One amazing thing about programming language is that we can **compose** small building blocks to create new ones.
- **Example:** convert degree to radius and compute the sin:

```
>>> x = math.sin((degrees/360.0)*2*math.pi)
```

```
>>> x
```

```
0.2588190451025207
```

Reading the Python documentation

Python » 2.7.10 » Documentation » The Python Standard Library » 9. Numeric and Mathematical Modules » [previous](#) | [next](#) | [modules](#) | [index](#)

Table Of Contents

- 9.2. `math` — Mathematical functions
 - 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
 - 9.2.6. Special functions
 - 9.2.7. Constants

9.2. `math` — Mathematical functions

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

These functions cannot be used with complex numbers; use the functions of the same name from the `cmath` module if you require support for complex numbers. The distinction between functions which support complex numbers and those which don't is made since most users do not want to learn quite as much mathematics as required to understand complex numbers. Receiving an exception instead of a complex result allows earlier detection of the unexpected complex number used as a parameter, so that the programmer can determine how and why it was generated in the first place.

The following functions are provided by this module. Except when explicitly noted otherwise, all return values are floats.

9.2.1. Number-theoretic and representation functions

`math.ceil(x)`
Return the ceiling of `x` as a float, the smallest integer value greater than or equal to `x`.

`math.copysign(x, y)`
Return `x` with the sign of `y`. On a platform that supports signed zeros, `copysign(1.0, -0.0)` returns `-1.0`.

New in version 2.6.

`math.fabs(x)`

Previous topic: 9.1. `numbers` — Numeric abstract base classes

Next topic: 9.3. `cmath` — Mathematical functions for complex numbers

This Page: [Report a Bug](#), [Show Source](#)

Quick search:

<https://docs.python.org/2/library/>

Define a new function

- Function is an named sequence of statements that performs a desired operation.
- **A function definition** specifies the name of a new function and the sequence of statements that execute when the function is called:

```
def hello():  
    print 'Hello!'
```

→ Function **Header**, Has to end with a :

→

- Function body: any number of statement
- Must be indented

Define a new function

- Function is an named sequence of statements that performs a desired operation.
- **A function definition** specifies the name of a new function and the sequence of statements that execute when the function is called:

```
def hello():  
    print 'Hello!'
```

→ Function **Header**, Has to end with a :

→

- Function body: any number of statement
- Must be indented

→ Calling the function, must have ()

```
hello()
```


Calling a function

name

parameter



```
Def Name(arg1, arg2):  
    Statements
```

```
Name(arg1, arg2)
```

Without calling a function, a function output **NOTHING.**

Calling a function

Function can be called using a statement.

Without calling a function, a function output **NOTHING**.

Example:

```
def printNumber(number):  
    print str(number)
```

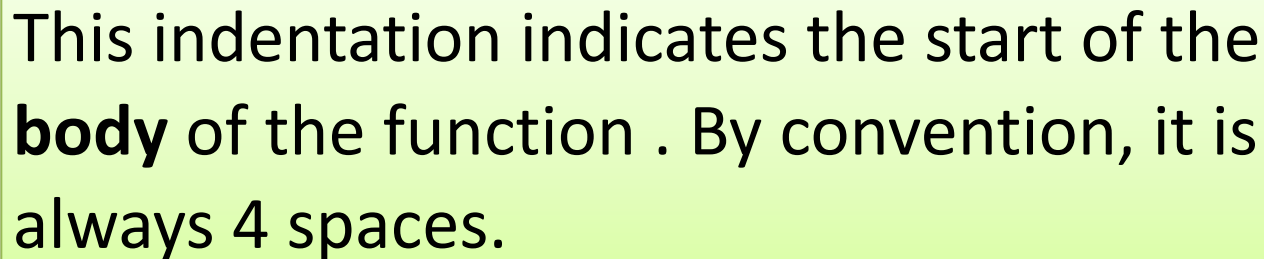
```
printNumber(5)
```

```
print type(printNumber)
```

A word about indentation

- Python is strict about **indentation**.

```
>>> def print_lyrics():  
    print "I am a lumberjack, and I am okay."  
    print "I sleep all night and I work all day."
```




This indentation indicates the start of the **body** of the function . By convention, it is always 4 spaces.

A word about indentation (use tab key)


- Python is strict about **indentation**.

```
>>> def print_lyrics():  
    print "I am a lumberjack, and I am okay."  
    print "I sleep all night and I work all day."
```



This indentation indicates the start of the **body** of the function

```
>>> print_lyrics()
```



No indentation means outside the function definition

```
I am a lumberjack, and I am okay.
```

```
I sleep all night and I work all day.
```

```
>>>
```

Function can contains another function

```
def two_pipes():  
    print '| |'
```

```
def more_pipes():  
    two_pipes()  
    print ''  
    two_pipes()
```


Parameters and arguments

Function Call

Command to do the function

```
greet('Walker')
```

Function
Header



Function Definition

Defines what function does

```
def greet(n):  
    print "hello " +n + "!"
```

- **Parameter**: variable that is listed within the parentheses of a method header.
- **Arguments**: a value to assign to the method parameter when it is called

Parameters and arguments

Function Call

Command to do the function

```
greet('Walker')
```

Function
Header

Function Definition

Defines what function does

```
def greet(n):  
    print "hello " +n + "!"
```

Function
Body

- **Parameter**: variable that is listed within the parentheses of a method header.
- **Arguments**: a value to assign to the method parameter when it is called

Parameters and arguments

Function Call

Command to do the function

```
greet('Walker')
```

Function
Header

Function Definition

Defines what function does

```
def greet(n):
```

```
    print "hello " + n + "!"
```

Declaration of
parameter n

Function Body

- **Parameter**: variable that is listed within the parentheses of a method header.
- **Arguments**: a value to assign to the method parameter when it is called

Parameters and arguments

Function Call

Command to do the function
`greet('Walker')`

argument to assign to n

Function
Header

Function Definition

Defines what function does

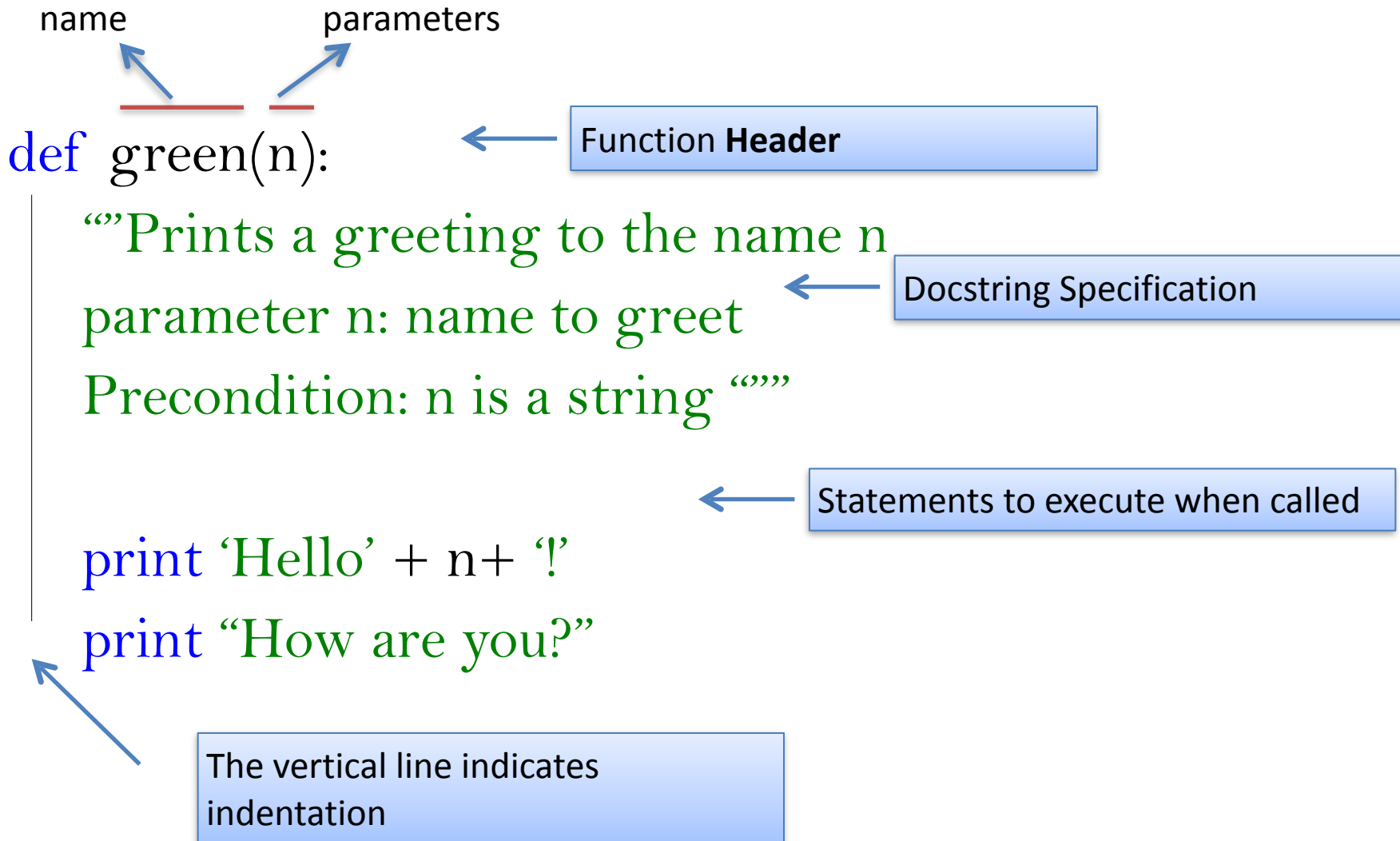
```
def greet(n):  
    print "hello " + n + "!"
```

Declaration of
parameter n

Function **Body**

- **Parameter**: variable that is listed within the parentheses of a method header.
- **Arguments**: a value to assign to the method parameter when it is called

Anatomy of a function definition



Exercise 1: say hello

- Define a function to print “hello” to a specific name.
- Call the function by `sayHello(“Bob”)`
- Which should print: Hello, Bob
- Call the function again to say Hello to Emily
- Using `raw_input` to ask the user about his or her name and call the function `sayHello(name)`

Exercise 2: write a few arithmetic functions

Write a the following functions that output arithmetic operation on two numbers. Note: please print out the functions.

- `def add(num1, num2):`
- `def sub(num1, num2):`
- `def mul(num1, num2):`
- `def div(num1,num2):`

Calling the above function by:

```
add(2,3)
```

```
num1 = 2
```

```
num2 = 3
```

```
num3 = 6.0
```

```
sub(num2,num1)
```

```
div(num3, num1)
```

Example: function without argument

Write a function that print the following shape:

```
  ||  
  ---  
  ||  
  ---  
  ||
```

Write another function to call the above function to print:

```
  ||  
  ---  
  ||  
  ---  
  ||  
  ---  
  ||  
  ---  
  ||  
  ---  
  ||  
  ---
```

Next week

- Function with returns
- Modules
- Lab 1 is due on Wed before class. No need to hand in the code. Just hand in the sheets.