

# EECS 12: Lecture 2

## Functions, Conditionals, Input

**Mark E. Phair**  
**mphair@gmail.com**  
**UC Irvine EECS**

**July 3rd, 2006**

# Agenda

- Functions
- Tracebacks
- Modules and the Standard Library
- Comparisons
- `if...elif...else`
- `raw_input`
- Random python factoid of the day

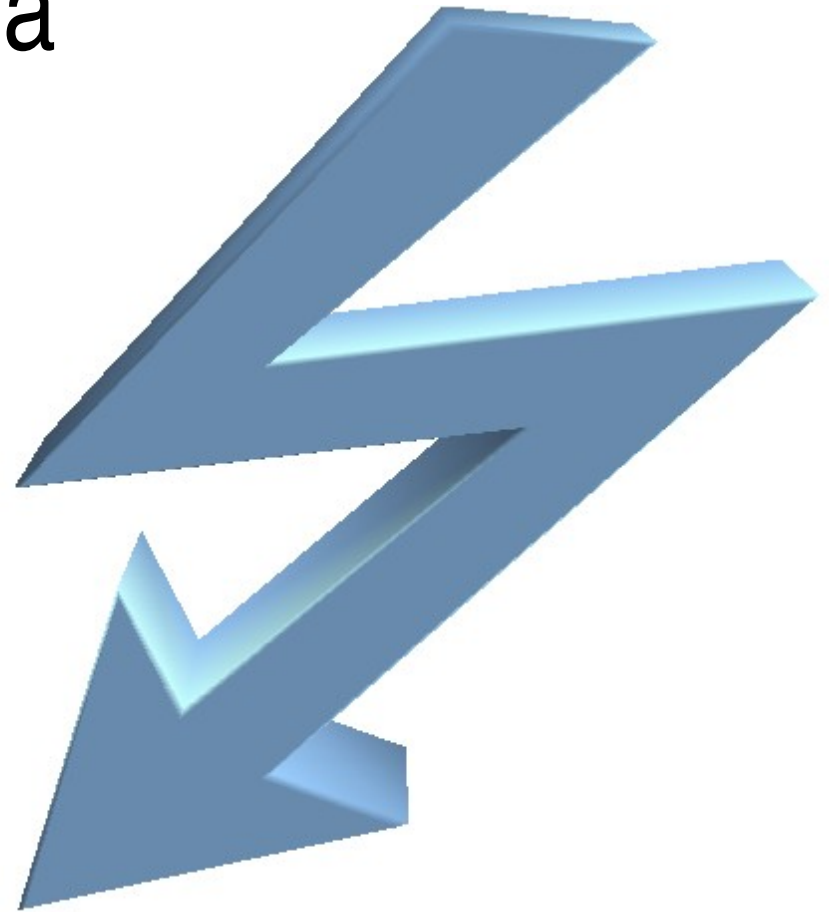
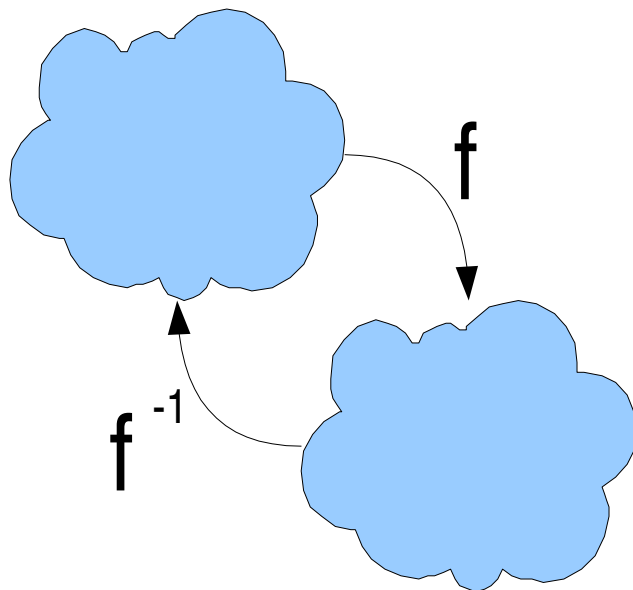


Figure 1: Mildly important and sensible graphic

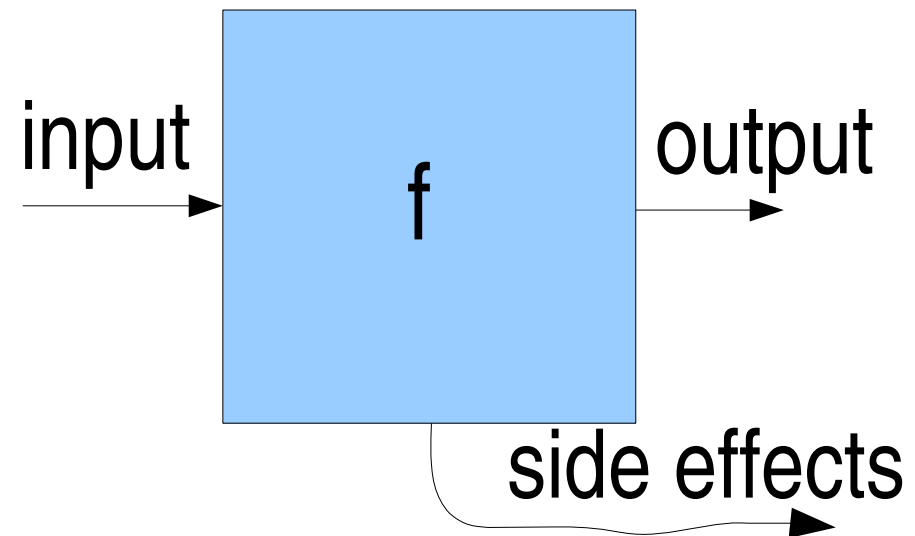
# Functions

- In mathematics, functions are a mapping
- In programming, they tend to act more like black boxes

## MATH



## PROGRAMMING



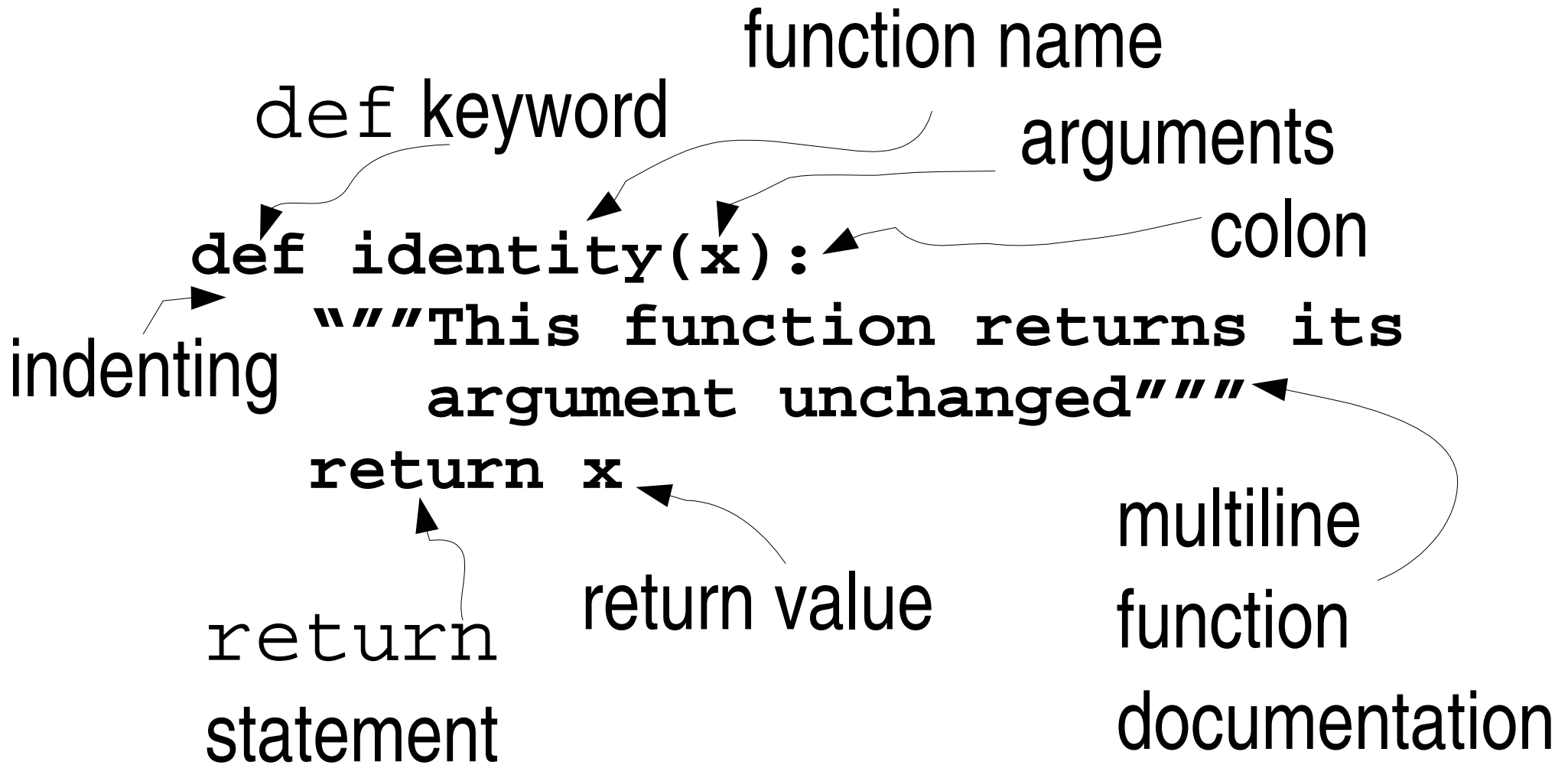
# A Generalization

- In programming, functions are a generalization of functions in math because of side effects
- Isolation of side effects is another dividing issue between programming languages
- Functions that have no output, only side effects, are often called “procedures,” but python does not make a distinction

# Our first function: the identity

```
def identity(x):  
    """This function returns its  
       argument unchanged"""  
    return x
```

# Identity: in pieces



# Identity: Use

```
>>> def identity(x):  
...     """This function returns its  
...     argument unchanged"""  
...     return x  
...  
>>> identity(5)  
5  
>>> identity('hi')  
'hi'
```

# Identity: Help

```
>>> help(identity)  
Help on function identity in module  
__main__:
```

```
identity(x)
```

```
This function returns its  
argument unchanged
```

# A (slightly) more useful function

```
def addThree(x):  
    """This function returns its  
       argument plus three"""  
    return x + 3
```

# Scope

```
def hasInternals(x):  
    "This function has internals"  
    i = 3  
    return x + i  
  
>>> hasInternals(5)  
8  
>>> i  
Traceback (most recent call last):  
  File "<stdin>", line 1, in ?  
NameError: name 'i' is not defined
```

# Scope

sidenote



```
def hasInternals(x):  
    "This function has internals"  
    i = 3  
    return x + i
```

```
>>> hasInternals(5)
```

```
8
```

```
>>> i
```

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

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

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

# Multiple arguments / return values

```
def multiple(x,y,z):  
    "This function is multi!"  
    return x+1, y+z, z+5
```

```
>>> multiple(1,2,3)  
(2, 5, 8)  
>>> a,b,c = multiple(1,2,3)  
>>> print a, ':', c, ':', b  
2 : 8 : 5
```

# Tracebacks

Tell you where an error happened...

aFun calls bFun which tries to call nonexistent cFun:

Traceback (most recent call last):

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

```
File "<stdin>", line 2, in aFun
```

```
File "<stdin>", line 2, in bFun
```

```
NameError: global name 'cFun' is not  
defined
```

# Modules and the Standard Library

- Modules are groupings of code which you can `import` to use in your own programs
- The standard library is everything that comes with python (motto: batteries included!)
- <http://docs.python.org/lib/lib.html> <-- library reference

# Play around with a standard module: math

```
>>> import math
```

```
>>> dir(math)
```

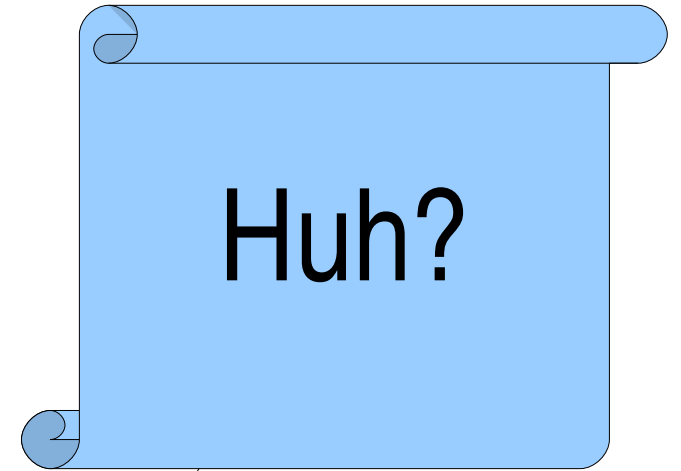
```
[contents of math listed]
```

```
>>> help(cos)
```

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

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

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



# Two solutions: Dot Notation and `from`

- `math.cos`      `<-- cos is in module math`
  
- `from`
  - `import math`    `<-- cos in module math`
  - `from math import cos`    `<-- cos is in toplevel`
  - `from math import *`    `<-- everything in math is in toplevel`

# Actually play around with `math`

Write your own function that:

- 1) Takes an argument expressed in degrees
- 2) Converts to radians
- 3) Uses a trig function
- 4) Uses an inverse trig function
- 5) Converts to degrees
- 6) Returns the result

# Comparisons

- Comparisons yield Boolean values: `True` and `False`
- Comparison operators: `==` `!=` `<` `>` `<=` `>=`
- Boolean operators: `not` `and` `or`

```
>>> 1 == 1
```

```
True
```

```
>>> 1 == 1 or 1 != 1
```

```
True
```

# Try out everything

- Comparison operators: `==` `!=` `<` `>` `<=` `>=`
- Boolean operators: `not` `and` `or`
- On different data types: `str`, `int`, `float`

# Conditional Execution: `if`

Allows the program to “make a decision”

```
if x == 5:  
    print 'x is equal to five'
```

# Conditional Execution: `if...else`

Allows the program to pick one or the other

```
if x == 5:  
    print 'x is equal to five'  
else # think of this as "otherwise"  
    print 'x is not equal to five'
```

# Conditional Execution:

## `if...elif...else`

Allows the program to pick from multiple possibilities

```
if x < 5: # checks this first
    print 'x is less than five'
elif x > 5: # if not(x<5) and (x>5)
    print 'x is greater than five'
else: # if not(x<5) and not(x>5)
    print 'x is equal to five'
```

# raw\_input

- Gets a string from the user of the program
- Can convert the string into other desired types
- Takes a “prompt” argument

```
>>> response = raw_input( 'why?  ' )
```

```
why? because i said so
```

```
>>> print 'user said:', response
```

```
user said: because i said so
```

# User input example

```
def getInputAddTwo():  
    return int(raw_input('#? '))+2
```

```
>>> getInputAddTwo()
```

```
#? 5
```

```
7
```

# Random Python Factoid of the Day

The underscore character by itself has a special meaning when used in the interactive interpreter...

```
>>> 5 + 6 + 5
```

```
16
```

```
>>> _ + 3
```

```
19
```