



python™

Functions

Function Basics

```
def max(x,y) :  
    if x < y :  
        return x  
    else :  
        return y
```

functionbasics.py

```
>>> import functionbasics  
>>> max(3,5)  
5  
>>> max('hello', 'there')  
'there'  
>>> max(3, 'hello')  
'hello'
```



Functions are objects

- Can be assigned to a variable
- Can be passed as a parameter
- Can be returned from a function
- Functions are treated like any other variable in Python, the **def** statement simply assigns a function to a variable



Function names are like any variable

- Functions are objects
- The same reference rules hold for them as for other objects

```
>>> x = 10
>>> x
10
>>> def x () :
...     print 'hello'
>>> x
<function x at 0x619f0>
>>> x()
hello
>>> x = 'blah'
>>> x
'blah'
```



Functions as Parameters

```
def foo(f, a) :  
    return f(a)  
  
def bar(x) :  
    return x * x
```

funcasparam.py

```
>>> from funcasparam import *  
>>> foo(bar, 3)  
9
```

- Note that the function `foo` takes two parameters and applies the first as a function with the second as its parameter



Higher-Order Functions

- **map(func,seq)** – for all i , applies $\text{func}(\text{seq}[i])$ and returns the corresponding sequence of the calculated results.

```
def double(x):  
    return 2*x
```

highorder.py

```
>>> from highorder import *  
>>> lst = range(10)  
>>> lst  
[0,1,2,3,4,5,6,7,8,9]  
>>> map(double,lst)  
[0,2,4,6,8,10,12,14,16,18]
```



Higher-Order Functions

- **filter(boolfunc,seq)** – returns a sequence containing all those items in seq for which boolfunc is True.

```
def even(x):  
    return ((x%2 ==  
0)
```

highorder.py

```
>>> from highorder import *  
>>> lst = range(10)  
>>> lst  
[0,1,2,3,4,5,6,7,8,9]  
>>> filter(even,lst)  
[0,2,4,6,8]
```



Higher-Order Functions

- **reduce(func,seq)** – applies func to the items of seq, from left to right, two-at-time, to reduce the seq to a single value.

```
def plus(x,y):  
    return (x + y)
```

highorder.py

```
■ >>> from highorder import *  
■ >>> lst = ['h','e','l','l','o']  
■ >>> reduce(plus,lst)  
■ 'hello'
```



Functions Inside Functions

- Since they are like any other object, you can have functions inside functions

```
def foo (x,y) :  
    def bar (z) :  
        return z * 2  
    return bar(x) + y
```

```
>>> from funcinfunc import *  
>>> foo(2,3)  
7
```

funcinfunc.py



Functions Returning Functions

```
def foo (x) :  
    def bar(y) :  
        return x + y  
    return bar  
# main  
f = foo(3)  
print f  
print f(2)
```

funcreturnfunc.py

```
% python funcreturnfunc.py  
<function bar at 0x612b0>  
5
```



Parameters: Defaults

- Parameters can be assigned default values
- They are overridden if a parameter is given for them
- The type of the default doesn't limit the type of a parameter

```
■ >>> def foo(x = 3) :  
■ ...     print x  
■ ...  
■ >>> foo()  
■ 3  
■ >>> foo(10)  
■ 10  
■ >>> foo('hello')  
■ hello
```



Parameters: Named

- Call by name
- Any positional arguments must come before named ones in a call

```
>>> def foo (a,b,c) :  
...     print a, b, c  
...  
>>> foo(c = 10, a = 2, b = 14)  
2 14 10  
>>> foo(3, c = 2, b = 19)  
3 19 2
```



Anonymous Functions

- A lambda expression returns a function object
- The body can only be a simple expression, not complex statements

```
>>> f = lambda x,y : x + y
>>> f(2,3)
5
>>> lst = ['one', lambda x : x * x, 3]
>>> lst[1](4)
16
```



Modules

- The highest level structure of Python
- Each file with the py suffix is a module
- Each module has its own namespace



Modules: Imports

<code>import mymodule</code>	Brings all elements of mymodule in, but must refer to as <code>mymodule.<elem></code>
<code>from mymodule import x</code>	Imports x from mymodule right into this namespace
<code>from mymodule import *</code>	Imports all elements of mymodule into this namespace

