Foundations of Computer Science

Functions and Conditionals

Golden Rules for Problem Solving

- Analyze the Problem
- Work out Concrete Examples; Make note of boundary cases
- Brainstorm about the Problem

What is a function?

$$f(x) = x^3 + 3x^2 - 87$$

$$f(4)$$

```
>>> x=4
>>> def Polynomial (n):
        """Returns the value of the polynomial x^3+3x^2-87 """
        return x**3+3*x**2-87

>>> Polynomial(x)
25
>>> |
```

Functioning in Python

```
# my own function!
def dbl( x ):
    """ returns double its input, x """
    return 2*x
```

Some of Python's *baggage*…

Docstrings

They become part of python's built-in help system! With each function be sure to include one that

(1) describes overall what the function does, and
(2) explains what the inputs mean/are

keywords

`def` starts the function
`return` stops it immediately and sends back the return value

Comments

They begin with #

Essential Definitions and Rules

(do memorize)

parameter (also called argument)

```
# my own function!
def dbl( x ):
    """ returns double its input, x """
    print ("Doubling input ", x)
    return 2*x
```

comment

function header

docstring

Function body

Indentation: All the lines in the function body are indented from the function header, and all to the same degree

Flow of Execution

```
# my own function!
def dbl( x ):
    """ returns double its input, x """
    print ("Doubling input ", x)
    return 2*x
```

Function definitions (including calls to other functions)

```
>>> dbl( 21 )
```

Function calls

When you call a function, Python executes the function starting at the first line in its body, and carries out each line in order (though some instructions cause the order to change… more soon)

Parameters are special variables

```
# my own function!
def dbl( x ):
    """ returns double its input, x """
    print "Doubling input ", x
    return 2*x
```

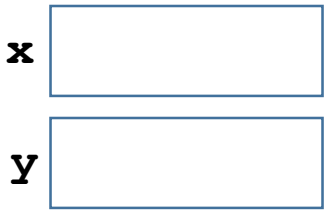
x

```
>>> dbl( 21 )
```

When you call a function, the value you put in parenthesis gets put into the "box" labeled with the name of the parameter and is available for use within the function.

Multiple parameters are allowed

```
# my own function!
def times( x, y ):
    """ returns x times y """
    print "Multiplying ", x, "and", y
    return x*y
```




```
>>> times( 21, 2 )
```

When you call a function, the values you put in parenthesis gets put into the “boxes” labeled with the names of the parameters (in the order in which they are listed)

No parameters is also allowed

```
# my own function!
def fortyTwo( ):
    """ returns 42 """
    return 42
```

```
>>> fortyTwo
```

As much as I like 42, I don't quite like this…

(But you still need parentheses)

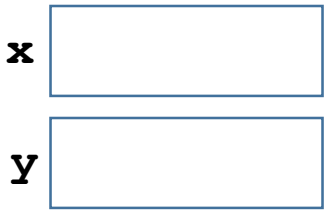
```
# my own function!
def fortyTwo( ):
    """ returns 42 """
    return 42
```

```
>>> fortyTwo()
```

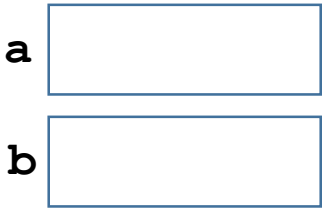
Ahh(), much better

You can also pass values via variables

```
# my own function!
def times( x, y ):
    """ returns x times y """
    print ("Multiplying ", x, "and", y)
    return x*y
```



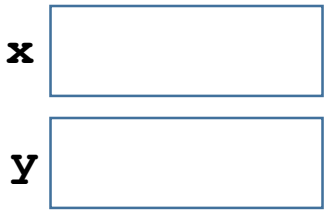

```
>>> a = 21
>>> b = 2
>>> times( a, b )
```



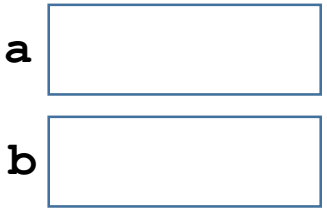
When you call a function, the values you put in parenthesis gets put into the “boxes” labeled with the names of the parameters (in the order in which they are listed)

You can also pass values via variables

```
# my own function!
def times( x, y ):
    """ returns x times y """
    print ("Multiplying ", x, "and", y)
    return x*y
```



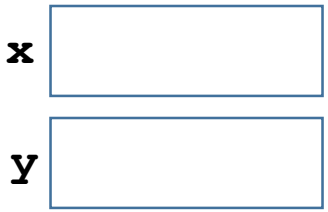

```
>>> a = 21
>>> b = 2
>>> times( b, a )
```



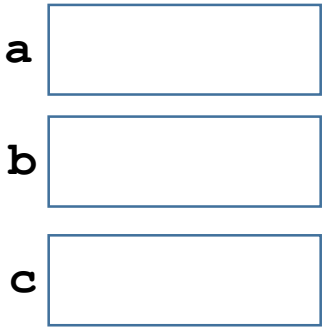
When you call a function, the values you put in parenthesis gets put into the “boxes” labeled with the names of the parameters (in the order in which they are listed)

Return gives back a value, which you store

```
# my own function!
def times( x, y ):
    """ returns x times y """
    print ("Multiplying ", x, "and", y)
    return x*y
```



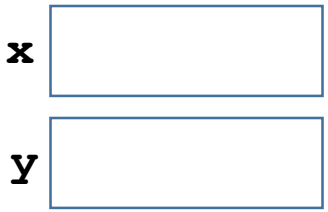

```
>>> a = 21
>>> b = 2
>>> c = times( b, a )
```



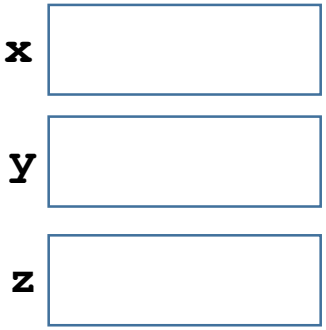
When you call a function, the values you put in parenthesis gets put into the “boxes” labeled with the names of the parameters (in the order in which they are listed)

Warning!

```
# my own function!
def times( x, y ):
    """ returns x times y """
    print ("Multiplying ", x, "and", y)
    return x*y
```




```
>>> x = 21
>>> y = 2
>>> z = times( y, x )
```

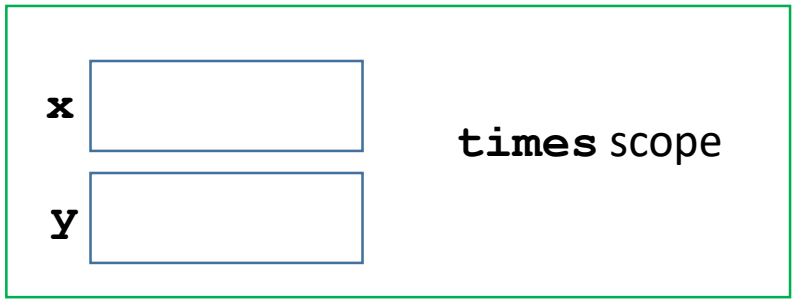


When you call a function, the values you put in parenthesis gets put into the “boxes” labeled with the names of the parameters (in the order in which they are listed)

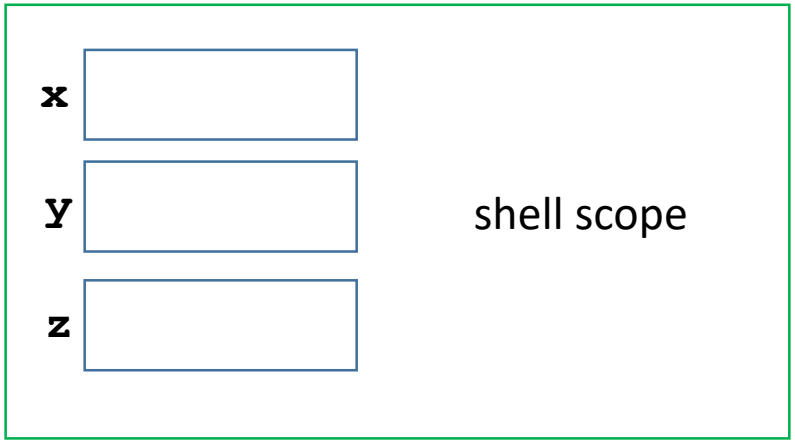
Variable *scope*

```
# my own function!
def times( x, y ):
    """ returns x times y """
    print ("Multiplying ", x, "and", y)
    return x*y
```

The scope of a variable is where it is defined to have a particular value. Each time a function is called in Python, it gets a fresh copy of its variables (including parameters). Their scope is the body of the function in that call only.




```
>>> x = 21
>>> y = 2
>>> z = times( y, x )
```



When you call a function, the values you put in parenthesis gets put into the "boxes" labeled with the names of the parameters (in the order in which they are listed)

Making choices: bool(ean) values


```
>>> 42 == 41
0 (i.e. False)
>>> 42 < 43
1 (i.e. True)
>>> not 42 > 42
1
>>> not 42 >= 42
0
>>> x = 42 != 42
>>> x
???
```

What is the value of x?

A. True (i.e. 1)
B. False (i.e. 0)
C. 42
D. Error

Boolean expressions can be complex

```
>>> x = 42
>>> y = 42
>>> x > 42 and y == 42


>>> x > 42 or y <= 42



>>> not x < 42 and y == 42
```

Making choices: conditional statements

```
def sameLastDigit ( num1, num2 ):
    """ Return True if integers num1 and num2
        end in the same digit, else False """
    if (num1%10) == (num2%10):
        return True
    else:
        return False
```

Let’s have some fun!

```
>>> def SameLastTwoDigits(x,y):
        """Returns true if x and y share the same last two digits"""
        if (x // 10)%10 == (y//10)%10 and x%10 == y%10:
                        return True
        elif x%10 == y%10:
                return "Same last digit, only. Close, but no cigar"
        else:
                return False
```

```
>>> SameLastTwoDigits(21,31)
'Same last digit, only. Close, but no cigar'
>>> SameLastTwoDigits(321, 1894821)
True
>>> SameLastTwoDigits(-983,183)
False
```

Ooops!

We can fix it!

```
def SameLastTwoDigits(x,y):
    """Returns true if x and y share the same last two digits"""
    if (abs(x)// 10)%10 == (abs(y)//10)%10 and abs(x)%10 == abs(y)%10:
                    return True
    elif abs(x)%10 == abs(y)%10:
            return "Same last digit, only. Close, but no cigar"
    else:
            return False
```

Functions can call Functions!!

When in doubt, draw it out!

```
def halve( x ):
    """ returns half its input, x """
    return div(x, 2)
def div( y, x ):
    """ returns y / x """
    return y / x
```

```
>>> halve( 84 )
```

return != print

```
def dbl(x):
    """ dbls x? """
    return 2*x

>>> dbl(21)

>>> dbl(21) * 2
```

```
def dblPR(x):
    """ dbls x? """
    print 2*x

>>> dblPR(21)

>>> dblPR(21) * 2
```

What is the difference between these?

A. No difference—they will behave the same way
B. The one of the left causes an error, while the one of the right does not
C. The one on the right causes and error, while the one on the left does not
D. The one of the right will print values, while the one on the left will not, but neither will cause an error

return != print

```
def dbl(x):
    """ dbls x? """
    return 2*x

>>> ans = dbl(21)
```

```
def dblPR(x):
    """ dbls x? """
    print 2*x

>>> ans = dblPR(21)
```

`print` just prints stuff to the screen...

`return` yields the function call's ***value*** …

… which the shell will print!

Variables

This program uses variables ***constantly***!

```
def convertFromSeconds(s):  # total seconds
    """ convertFromSectons(s): Converts an
          integer # of seconds into a list of
          [days, hours, minutes, seconds]
        input s: an int
    """
    seconds = s % 60        # leftover seconds
    m = (s / 60)            # total minutes
    minutes = m % 60        # leftover minutes
    h = m / 60              # total hours
    hours = h % 24          # leftover hours
    days = h / 24           # total days
    return [days, hours, minutes, seconds]
```