Background

Procedures

A *procedure* (or a *function* or a *method*) is a block of code set aside from the rest of the program. They allow you to write a chunk of code that can be run over and over without duplicating the code. Using *arguments* (or *parameters*), a function can also be made to work differently based on user input.

Python Function Syntax

The `def` keyword

Functions are created, or *defined* using the `def` keyword.

```
def FUNCTIONNAME(ARGLIST):
  FUNCTIONBODY
```

Functions have four distinct parts:

`def`	The keyword signifying that we're making a new function
Function name	The name of the function; e.g.: `raw_input`
Argument list	A list of arguments [*optional*]
Function body	The body of the function, the list of commands to run

Function Name

The function name follows the same rules as variable naming in Python. For many languages, the following rules apply:

1. Numbers, letters, and the underscore may be used
2. Names cannot start with numbers
3. Names are case sensitive. `book` $\neq$ `Book` $\neq$ `BoOk`

Argument List

An argument is used to pass information into the function for it to work with. A function can take $0+$ arguments.

The simplest way to do an argument list in Python is simply a comma-separated list of variable names:

```
def myFunction(arg1, arg2, arg3):
  # function body goes here
```

If desired, you can specify default values for any or all parameters. When a default argument is present, this argument becomes optional. If a value is given, it overrides the default. If no value is given, the default is used. Keep all arguments with default values after all arguments without default values.

```
def myFunction(reqArg1, optArg1=42, optArg2='Hello, World!'):
  # function body goes here
```

To use a function, simply call it like we did the `raw_input` function:

```
def myfunc(age=5, name='Timmy'):
  print name, 'is', age

myfunc()
myfunc(25, 'Lisa')
myfunc(name='Someone')
```

Output:

```
Timmy is 5
Lisa is 25
Someone is 5
```

Function Body

The function body can contain any executable statement. Examples from last week are `print` and `raw_input`.

Functions can end with a `return` statement. This is used to *return* a value to the caller. You have used functions with return statements when you called `int()`, `float()`, and `raw_input()`. Here is a simple example:

```
def getSix():
  return 6
var = getSix() # var = 6
```

One thing to be aware of is that spacing is important in Python. All statements in a function body must be indented from the `def`. How you choose to space them in is up to you. One space, two spaces, 18 spaces, one tab, two tabs, whatever. The spacing must be consistent. Consider the following piece of code:

```
def myfunc():
  print 'Hello'
print 'World'
```

The third line, `print ''World''`, is not part of `myfunc()` because it is not indented with `print ''Hello''`.

While this may seem like it would be obvious to catch now, when decision statements and looping structures are added (particularly when *nesting* them, having one inside another), keeping track of what body a particular statements belongs to can become difficult.

Function Placement

Languages based off of the C programming language have a function called `main()`, or at least one equivalent to it. Many other types of languages, including Python, have an implicit main.

The main function, the body of code that runs when execution starts, is the end of a Python program. All functions must be defined above it. Example:

```
def myfunc():
  # body
def myotherfunc():
  # body
def yetanother():
  # body
# this is my ''main''
```

Week 4 Lab: Python Functions

Directions

This week's lab involves making modifications to your calculator from last week. Open `calc.py` and save it as `funccalc.py`. As with last week, be sure to name it exactly as shown; all letters are lower-case. Save it to *Z:*.

Last week's program read in two values from the user and performed the four basic math operations on them. For this week, make the following changes:

1. Do not ask the user for the values, we will hard-code them
2. Create a function named `calc` that takes two arguments, `a` and `b`
3. Give the `a` a default value of `8` and `b` a default value of `16`
4. The function prints out the four values as it did before; there is no return value

When this is done, use the following as your "main":

```
calc()
calc(8, 16)
calc(b=32)
calc(b=-8)
calc(0, 1)
```

Do not add any other `print` statements to the code. I will have a script compare your responses to mine.

Handing in

Submit your two Python scripts on the `cse101l` handin. They are due by 23:59 on Friday, February 6, 2009.

Grading

Area	Points
Works:	10
Correct output:	5
Correct filename:	5
Total:	20