1 Defining functions

(This is part 1 of defining functions; more details will follow in later handouts.) A function is like a mini program. It takes some inputs, does some computation, and produces some output.

Function definition: We can define new functions like this:

```
def greet(n):
    print "Hi, " + n + "!"
```

The first line of a function definition is called the **header** and it always starts with the keyword `def`, which indicates to python that this is a function definition. What follows `def` is the name of the function: this function is named `greet`. The **parameters** to the function are listed inside parentheses: this function has a single **parameter** named `n`. The computation that the function performs is described in the function **body**. The body can contain any number of statements: this function has only one statement in its body.

Function call: Once the function has been defined, you can **call** it. This is just like calling a builtin function. For example, suppose that after we write the definition above, we call the `greet` function twice, passing in different names each time.

```
greet("Michael")        # calling the function greet
friends_name = raw_input("What is your friend's name? ")
greet(friends_name)     # calling greet a second time
```

If I run this program and type `"Owen"` at the prompt, this is the output I see in IDLE:

```
Hi, Michael!
What is your friend's name? Owen
Hi, Owen!
```

A function must be defined before it can be called. Consider this program where a new function `goodbye` is called and defined:

```
goodbye("Michael")  # function call appearing before definition

def goodbye(n):     # function definition
    print "See you later, " + n + "!"
```

We would get this error: `NameError: "name 'goodbye' is not defined"`. Python says that `goodbye` is not defined because it reads programs from top to bottom and when it encountered the function call `goodbye("Michael")`, the function `goodbye` had not yet been defined.

2 Parameters vs. Arguments

A **parameter** is a variable name that is listed in the parentheses of a function header. An **argument** is a value to assign to a function parameter when the function is called. Remember: parameters appear in the function *definition*; arguments appear in the function *call*.

In the preceding example, the parameter is `n`. During the first function call, the argument is `"Michael"`. During the second call, the argument is `friends_name`. As this example suggests, an argument can be a variable name. In fact, an argument can be any python expression, even this:

```
greet("Graham " + "Spam " * 3 + "Chapman")
```

3 Flow of execution

It is very important to understand what happens when python encounters a function call. Later handouts will explore this in even more detail.

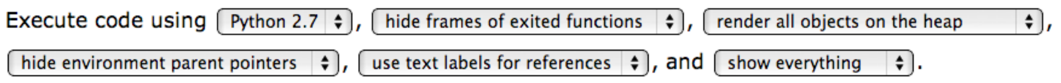
The rules for executing a function call:

1. Evaluate the *arguments* to produce memory addresses.
2. Store those memory addresses in the corresponding *parameters*.
3. Execute the body of the function.
4. When complete, *return* to the location in the program where the function was called.

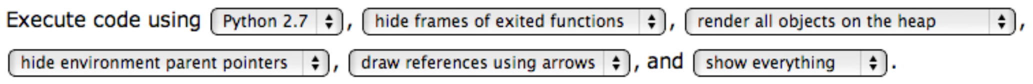
Consider this example of a function call to `print_name` inside the body of `silly_greeting`.

```
def print_name(name):
    print name,

def silly_greeting(first, last):
    print "The one, the only,",
    print_name(first)
    print "The Rock",
    print_name(last)
    print "is in the house!"

silly_greeting("Dwayne", "Johnson")
```

Use the python visualizer (`http://www.pythontutor.com/visualize.html`). Be sure to adjust the settings so they look like this:


Execute code using Python 2.7, hide frames of exited functions, render all objects on the heap, hide environment parent pointers, use text labels for references, and show everything.

or this


Execute code using Python 2.7, hide frames of exited functions, render all objects on the heap, hide environment parent pointers, draw references using arrows, and show everything.