Functions
A function is a piece of code you can use over and over again.
- Treat it like a black box.
- You pass it (optional) values, it does some work, and it (optionally) returns values.
- You “call it”, “invoke it”, or “use it” by using its name and parentheses.
  - The things you pass it go inside the parentheses.
  - output = function(input)
Using Simple Functions

- Built-in functions
  - `max(num1, num2)`
  - `abs(number)`
- Pass in arguments
- Execute in sequential order
  - flow of execution

`max(4, -61)`
`abs(-24.5)`
Writing Simple Functions

- **Defining functions**
  - Creates function
  - Does not execute/run them

- Indenting indicates a “block” of code
- Call functions from top-level or other functions
- Program execution: Choose-Your-Own-Adventure books

```python
def doMath():
    print "max of 4 and -61 is",
    print max(4,-61)
    print "finished computation"

doMath()
```

Indenting indicates a "block" of code.

Call functions from top-level or other functions.

Program execution: Choose-Your-Own-Adventure books.
Format of a function definition

def function-name():
    statement
    statement
    ...
    statement
Writing Functions with Parameters

def doMath(number1, number2):
    print "Figuring out the max of", number1, number2
    print max(number1, number2)
    print "finished with function doMath"

doMath(-1, -2)
doMath(100, 150)
Parameters are Variables

- When you pass values into functions as parameters, they get assigned to the variable names declared in the definition line of the function.
- For example, when you call `doMath(-1, 4)`
  The number1 variable is assigned (points to) the value -1 and the number2 variable is assigned the value 4
- When the code in the function refers to the number1 variable, it evaluates to the number -1
- So, when you call `doMath(-1, 4)` and the `doMath` function calls `max(number1, number2)`, it's the same as if it called `max(-1, 4)`
Format of a Function Definition with Parameters

```python
def function-name(list-of-params):
    statement
    statement
    ...
    statement

function-name(list-of-params)
```
Functions with return values

Functions can produce results (e.g. max() and abs()) but they don't have to

Example:

```python
def printMyName(first, last):
    print "My first name is", first
    print "My last name is", last

printMyName("Joe", "Smith")
printMyName("Valerie", "Summet")
```

Learn more in Chapter 5: Fruitful Functions
Composing Functions

• You can use the output (return value) of one function as the input (parameter) to another function.

abs( max(-4, -100))

• In this example, the max() function executes first (things inside parenthesis execute before things outside parenthesis)
• The abs() function computes the absolute value of what is returned from the max function
Variables in a Function are Local

- Variables in a function are private
  - Including the parameters
  - This means they are not “visible” to (can't be used by) the rest of the program.

- Each function has its own variables
  - Even when the names are the same

- Allows you to write functions independently without worrying about using the same name
Functions with Local Variables

def area(radius):
    a = 3.14 * radius**2
    print "Area", a

def circumference(diameter):
    c = 3.14 * diameter
    print "Circumference", c

area(3)
circumference(6)
print a  #this will produce an error
print c  #as will this
Different Variables - Same Name

def area(radius):
    a = 3.14 * radius**2
    print "area", a

def circumference(radius):
    a = 3.14 * 2 * radius
    print "circumference", a

a = 20
print "Area of a 3 ft circle", area(3)
print "Circumference", circumference(3)
print a
Passing variables to a function

- If you pass a variable to a function, the function gets the value that the variable is pointing at

```plaintext
x = 40
area(x)

is the same as

area(40)
```
Write a function named `hoursToMinutes` which takes a parameter that references a number of hours. The function should print out the equivalent number of minutes.