Functions

Functions are the most important organizational tool in programming. Although there are other useful tools that we will cover in this course, once you understand functions and how to build programs with them you can call yourself a programmer.

Three Big Ideas

We will spend the next few classes making sense of these ideas.

First Big Idea

A *function* is a block of code that has a name. We can execute the code just by using its name. This is called *calling* the function.

We define a function with the word *def.* Function names always have open and closed parentheses after them; we will see the reason for this momentarily.

For example, here is a function definition:

```
def starBox():
        for i in range(0, 3):
                print( "***" )

        print( )
```

A call to this prints

```
***
***
***
```

A complete program with this function might be

```
def starBox():
        for i in range(0, 3):
                print( "***" )
        print()

starBox()
starBox()
```

This will print

***

***

***

***

***

***

Clicker Question

What will this program print?

```
def line():
    for i in range(2, 10, 2):
        print( i, end = "  ")
    print()

line()
line()
```

Answers:

A) 2 3 4 5 6 7 8 9

B) 2 4 6 8

C) 2 4 6 8 2 4 6 8

D) 2 4 6 8
   2 4 6 8

One great advantage of functions is that they break up code into coherent chunks and give names to those chunks. This makes longer programs much easier to understand. It also helps you to think about the programs, which makes them easier to write.

We are now in a position to understand how the text presents programs. Every program in the textbook is written as

```
def main( ):
        <code for the program>

main( )
```

This creates a function called *main( )* to hold the program; the last line calls this function. Both C and Java require programs to be written this way. Python doesn't require it, but it is still a good idea.

For example,

```
def main():
        n = eval(input( "Enter n: " ))
        prod = 1
        for x in range(2, n+1):
                prod = prod*x
        print( "The factorial of %d is %d"%(n, prod)

main( )
```

This way every program will consist of a bunch of function definitions. We run the program by executing main( ), which usually calls other functions.

Second Big Idea

Functions can have their own variables. The variables of a function can't be seen or referenced outside the function. This means that two functions can have variables with the same name. Variable x in one function has nothing to do with any other variable x in the program.

The Big Idea is that functions can have special variables called *parameters* or *arguments* that allow the caller of a function to give it initial data values. The parameters go inside the parentheses after the function name. When you call a function whose definition has parameters you must supply a value for each parameter. These values are called the *arguments* of the call.

For example

```
def printStars(n):
    for i in range(0, n):
        print( "*", end='')
    print()

printStars(3)
printStars(7)
printStars(2)
```

This prints

```
***
*******
**
```

If a function has multiple parameters, the arguments are given in the same order as the parameters.

```
def printChars(c, n):
    for i in range(0, n):
        print( c, end='')
    print()

printChars('+', 5)
printChars('*', 3)
```

This will print

```
+++++
***
```

Clicker Question

What will this print?

```
def A(n):
    for i in range(0, n):
        print( "*", end='')
    print()

def B(n):
    for i in range(1,n+1):
        A(2*i)

B(3)
```

Answers:

A)
```
**
****
******
```

B)
```
***
***
***
```

C)
```
*
**
***
```

That question would have been easier if we used good names and comments:

```
def  printStars(n):
        # This prints n stars on one line
   for i in range(0, n):
        print( "*", end='')
   print()

def  printTriangle(n):
        # This prints n rows where row i has 2*i stars
for i in range(1,n+1):
        printStars(2*i)

printTriangle(3)
```

Third Big Idea

We use parameters and arguments to put data into a function. Functions can also give data back to their callers by *returning* values.

For example,

```
def square(x):
    return x*x

def main( ):
    n = eval(input( "number: " ))
    nsq = square(n)
    print( "The square of %d is %d."%(n, nsq))

    print( square(9))

main()
```

Functions that return values act like nouns -- they represent the values they return. Functions that don't return values act like verbs -- they do something when they are called.

```
def square(n):
        return n*n

def main():
        x = square(5)
        print(x)
```

```
def printSquare(n):
        print( n*n )

def main():
        printSquare(5)
```

Clicker Question

I am writing a function to return the number of days in a year:

```
def numDays(y):
    if isLeapYear(y):
        return 366
    else:
        return 365
```

What is isLeapYear(y)???

A) A function that prints "that is leap year" if year y is a leap year

B) A function that returns True if year y is a leap year

C) A function that returns True if y is a leap year and False if it is not.

Note that a return statement causes the function to immediately stop and return control to the caller. For example, we could write an isPrime( ) function like this:

```
def isPrime(x):
    for d in range(2, x):
        if x%d == 0:
            return False
    return True
```

One more clicker question. What will this program print?

```
def foo( ):
        return 3
        return 125
def main( ):
        x = foo( )
        y = foo( )
        print( x, y)
main()
```

Answers:

A) 3 125

B) 3 3

C) 125 125

D) an error message

Finally, if you don't tell a function in Python to return something, it will return the value None. For example, the program

```
def size(n):
    if n < 10:
        return "small"
    elif n < 100:
        return "medium"

for x in  [5, 50, 500]:
    print( size(x) )
```

will print

```
small
medium
None
```