

## CSI31 Lecture 16

### **Topics:** *Chapter 6. Defining Functions*

6.1 The Function of Functions

6.2 Functions, Informally

6.3 Future Value with a Function

6.4 Functions and Parameters: The Details

6.5 Getting Results from a Function

**HW №13** (due date is Wednesday, November 12th):  
programming exercise 4 on page 195

The programs we've written so far had just one function (called `main`).

We also used some pre-written (built-in) functions and methods, including `math.sqrt`, `string.split`, `point.getX` and so forth.

Chapter 6 cover whys and hows of designing our own functions.

### Why do we need functions?

- Mainly to make our programs easier to write, to read, to test, and to update/modify.

We can think of a `function` as a *subprogram* – a small program inside of a program.

- we write a sequence of statements and give that sequence a **name**;  
then these instructions can be executed at any point in the program by referring to the function name.

**Function definition** looks like this:

```
def <name>(<formal parameters>):  
    <body>
```

`name` – is an identifier

`formal parameters` – list (possibly empty) of variable names

`body` – is the body of the function, all formal parameters are accessible only in this part.

Then let's see how can we **invoke a function (function call)**:

```
<name>(<actual parameters>)
```

`actual parameters` – are the the ones whose values are assigned to formal parameters

**Example:** Let's write a program that takes three values from the user (decimal values) and returns their product, their sum, their absolute values, and their average.

Let's split our program into four subprograms (functions):

1 function – finds the product,  
another function – finds the sum,  
another one – finds their absolute values, and  
the last one – finds their average

Questions to think about:

- How do we supply the data values to the functions?

*Through parameters*

- How many of those three values inputted by the user should be given to each function?

*All three, to each function*

Let's write the schema (algorithm) of the program:

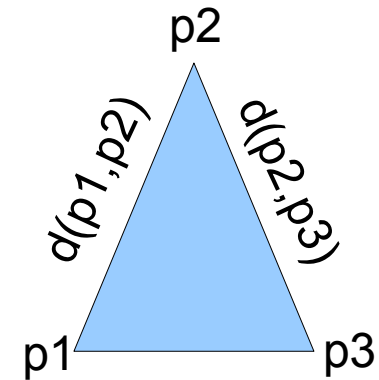
```
def main():  
    get three values from the user (a,b,c)  
  
    call product(a,b,c)  
    call sum(a,b,c)  
    call absolute_v(a,b,c)  
    call average(a,b,c)  
  
    say good bye to the user
```

See [four\\_f.py](#)

Functions may **return a value to the caller** (use **return** statement for that)

**example:** Let's write a function that squares a given value and returns it to the caller.

```
def main():  
    a=input('Please, input a decimal number:')  
  
    b=square(a)  
  
def square(x)  
    return x*x  
  
main()
```



Now, let's write the following program (**next example**):

The program draws a triangle based on three user's mouse clicks. Then it finds the perimeter of the rectangle and displays it in the same graphics window, where rectangle is drawn.

In order to find the perimeter we need to recall the formula for the distance between two points:

For points  $(x_1, y_1)$  and  $(x_2, y_2)$  the distance between them is  $d = \sqrt{(x_2 - x_1)^2 + (y_2 - y_1)^2}$

Thus having three points and knowing the formula for the distance we can find the perimeter:  $P = d(p_1, p_2) + d(p_2, p_3) + d(p_3, p_1)$

We'll write a function that finds the distance between two points, and returns it to the caller. (since the distance formula will be called three times)

*Algorithm:*

```
def main():  
    get three mouse clicks  
  
    call distance function for three pairs of points,  
    add up the results  
  
    output the perimeter  
  
    close the window  
  
def distance(p1,p2)  
    find the distance  
    return the distance
```

See the program [triangle.py](#)