Defining Functions

**Creating Functions**

To start off this chapter I am going to give you an example of what you could do but shouldn't (so don't type it in):

a = 23

b = -23

**if** a < 0:

 a = -a

**if** b < 0:

 b = -b

**if** a == b:

 **print**("The absolute values of", a, "and", b, "are equal.")

**else**:

 **print**("The absolute values of", a, "and", b, "are different.")

with the output being:

The absolute values of 23 and 23 are equal.

The program seems a little repetitive. Programmers hate to repeat things -- that's what computers are for, after all! (Note also that finding the absolute value changed the value of the variable, which is why it is printing out 23, and not -23 in the output.) Fortunately Python allows you to create functions to remove duplication. Here is the rewritten example:

a = 23

b = -23

**def** absolute\_value(n):

 **if** n < 0:

 n = -n

 **return** n

**if** absolute\_value(a) == absolute\_value(b):

 **print**("The absolute values of", a, "and", b, "are equal.")

**else**:

 **print**("The absolute values of", a, "and", b, "are different.")

with the output being:

The absolute values of 23 and -23 are equal.

The key feature of this program is the def statement. def (short for define) starts a function definition. def is followed by the name of the function absolute\_value. Next comes a '(' followed by the parameter n (n is passed from the program into the function when the function is called). The statements after the ':' are executed when the function is used. The statements continue until either the indented statements end or a return is encountered. The return statement returns a value back to the place where the function was called. We already have encountered a function in our very first program, the print function. Now we can make new functions.

Notice how the values of a and b are not changed. Functions can be used to repeat tasks that don't return values. Here are some examples:

**def** hello():

 **print**("Hello")

**def** area(width, height):

 **return** width \* height

**def** print\_welcome(name):

 **print**("Welcome", name)

hello()

hello()

print\_welcome("Fred")

w = 4

h = 5

**print**("width =", w, " height =", h, " area =", area(w, h))

with output being:

Hello

Hello

Welcome Fred

width = 4 height = 5 area = 20

That example shows some more stuff that you can do with functions. Notice that you can use no arguments or two or more. Notice also when a function doesn't need to send back a value, a return is optional.

**Variables in functions**

When eliminating repeated code, you often have variables in the repeated code. In Python, these are dealt with in a special way. So far all variables we have seen are global variables. Functions have a special type of variable called local variables. These variables only exist while the function is running. When a local variable has the same name as another variable (such as a global variable), the local variable hides the other. Sound confusing? Well, these next examples (which are a bit contrived) should help clear things up.

a = 4

**def** print\_func():

 a = 17

 **print**("in print\_func a =", a)

print\_func()

**print**("a = ", a)

When run, we will receive an output of:

in print\_func a = 17

a = 4

Variable assignments inside a function do not override global variables, they exist only inside the function. Even though a was assigned a new value inside the function, this newly assigned value was only relevant to print\_func, when the function finishes running, and the a's values is printed again, we see the originally assigned values.

Here is another more complex example.

a\_var = 10

b\_var = 15

e\_var = 25

**def** a\_func(a\_var):

 **print**("in a\_func a\_var =", a\_var)

 b\_var = 100 + a\_var

 d\_var = 2 \* a\_var

 **print**("in a\_func b\_var =", b\_var)

 **print**("in a\_func d\_var =", d\_var)

 **print**("in a\_func e\_var =", e\_var)

 **return** b\_var + 10

c\_var = a\_func(b\_var)

**print**("a\_var =", a\_var)

**print**("b\_var =", b\_var)

**print**("c\_var =", c\_var)

**print**("d\_var =", d\_var)

output:

in a\_func a\_var = 15

in a\_func b\_var = 115

in a\_func d\_var = 30

in a\_func e\_var = 25

a\_var = 10

b\_var = 15

c\_var = 125

d\_var =

Traceback (most recent call last):

 File "C:\def2.py", line 19, in <module>

 print("d\_var = ", d\_var)

NameError: name 'd\_var' is not defined

In this example the variables a\_var, b\_var, and d\_var are all local variables when they are inside the function a\_func. After the statement return b\_var + 10 is run, they all cease to exist. The variable a\_var is automatically a local variable since it is a parameter name. The variables b\_var and d\_var are local variables since they appear on the left of an equals sign in the function in the statements b\_var = 100 + a\_var and d\_var = 2 \* a\_var .

Inside of the function a\_var has no value assigned to it. When the function is called with c\_var = a\_func(b\_var), 15 is assigned to a\_var since at that point in time b\_varis 15, making the call to the function a\_func(15). This ends up setting a\_var to 15 when it is inside of a\_func.

As you can see, once the function finishes running, the local variables a\_var and b\_var that had hidden the global variables of the same name are gone. Then the statementprint("a\_var = ", a\_var) prints the value 10 rather than the value 15 since the local variable that hid the global variable is gone.

Another thing to notice is the NameError that happens at the end. This appears since the variable d\_var no longer exists since a\_func finished. All the local variables are deleted when the function exits. If you want to get something from a function, then you will have to use return something.

One last thing to notice is that the value of e\_var remains unchanged inside a\_func since it is not a parameter and it never appears on the left of an equals sign inside of the function a\_func. When a global variable is accessed inside a function it is the global variable from the outside.

Functions allow local variables that exist only inside the function and can hide other variables that are outside the function.

**Examples**

**temperature2.py**

*#! /usr/bin/python*

*#-\*-coding: utf-8 -\*-*

*# converts temperature to Fahrenheit or Celsius*

**def** print\_options():

 **print**("Options:")

 **print**(" 'p' print options")

 **print**(" 'c' convert from Celsius")

 **print**(" 'f' convert from Fahrenheit")

 **print**(" 'q' quit the program")

**def** celsius\_to\_fahrenheit(c\_temp):

 **return** 9.0 / 5.0 \* c\_temp + 32

**def** fahrenheit\_to\_celsius(f\_temp):

 **return** (f\_temp - 32.0) \* 5.0 / 9.0

choice = "p"

**while** choice != "q":

 **if** choice == "c":

 c\_temp = float(input("Celsius temperature: "))

 **print**("Fahrenheit:", celsius\_to\_fahrenheit(c\_temp))

 choice = input("option: ")

 **elif** choice == "f":

 f\_temp = float(input("Fahrenheit temperature: "))

 **print**("Celsius:", fahrenheit\_to\_celsius(f\_temp))

 choice = input("option: ")

 **elif** choice == "p": *#Alternatively choice != "q": so that print*

 *#when anything unexpected inputed*

 print\_options()

 choice = input("option: ")

Sample Run:

Options:

 'p' print options

 'c' convert from celsius

 'f' convert from fahrenheit

 'q' quit the program

option: **c**

Celsius temperature: **30**

Fahrenheit: 86.0

option: **f**

Fahrenheit temperature: **60**

Celsius: 15.5555555556

option: **q**

**area2.py**

*#! /usr/bin/python*

*#-\*-coding: utf-8 -\*-*

*# calculates a given rectangle area*

**def** hello():

 **print**('Hello!')

**def** area(width, height):

 **return** width \* height

**def** print\_welcome(name):

 **print**('Welcome,', name)

**def** positive\_input(prompt):

 number = float(input(prompt))

 **while** number <= 0:

 **print**('Must be a positive number')

 number = float(input(prompt))

 **return** number

name = input('Your Name: ')

hello()

print\_welcome(name)

**print**()

**print**('To find the area of a rectangle,')

**print**('enter the width and height below.')

**print**()

w = positive\_input('Width: ')

h = positive\_input('Height: ')

**print**('Width =', w, ' Height =', h, ' so Area =', area(w, h))

Sample Run:

Your Name: **Josh**

Hello!

Welcome, Josh

To find the area of a rectangle,

enter the width and height below.

Width: **-4**

Must be a positive number

Width: **4**

Height: **3**

Width = 4 Height = 3 so Area = 12

**Exercises**

Rewrite the area2.py program from the Examples above to have a separate function for the area of a square, the area of a rectangle, and the area of a circle (3.14 \* radius\*\*2). This program should include a menu interface.