|  |  |
| --- | --- |
| LAB: Functions | Name: Lex Baker  Date: 11/1/22 |

A ***function*** is a named sequence of statements that performs a computation or task[[1]](#footnote-1). Functions make it possible to write code that is more efficient, better organized, and easier to reuse.

**1.**Do the following exercises at the command line to review the function terminology we have already encountered. Pay special attention to the terms in **bold**.  
**a.** **Assign** the integer values 7, 23 and 28 to a **variables named** x, y, and z.

**b.** Write a statement to **call** the print function and **pass** it the following **arguments**

* The string "If the side is"
* The variable named x
* The string "then the area is"
* The **expression** x\*\*2

Write down your **statement** and the resulting **output**.

**>>> print("If the side is", x, "then the area is", x\*\*2)**

**If the side is 7 then the area is 49**

**c.** Write a **statement** that **calls** the min function, **passes** it the **arguments** x, y, and z, and **assigns** the **return value** to a variable named smallest. Copy the statement below. What was the end result of this statement?

**>>> smallest = min(x, y, z)**

**>>> print(smallest)**

**7**

**d.** Write an **expression** that calls the type function and **passes** it the expression x/2 as an **argument**. What is the **value** **returned** by the type function, and what is done with this value after the expression is evaluated?

>>> **type(x/2)**

**<class ‘float’>**

**After being evaluated, the value is discarded since it is not assigned or used.**

**e.** Write an **expression** that calls the chr function, **passes** it an **argument** of 72, and **passes** the resulting **return value** directly into the type function.

**>>> type(chr(72))**

**<class ‘str’>**

**2.** Open the file silly\_song.py from Canvas (copied below). The program in the file defines and then calls a function in order to create and print (abridged) lyrics to a silly song. Run the program and then read the information below.

1 ##Prints a silly song

2 ##Illustrates how to define and call a custom function

3

4 def funnyFruit(letter) :

5 """ Create and print a silly message """

6 message = letter + "pples and b" + letter + "n" \

7 + letter + "n" + letter + "s";

8 print(message);

9

10 ##Main program

11 print("I like to eat...");

12 funnyFruit('a');

13 funnyFruit('e');

14 funnyFruit('i');

15 funnyFruit('o');

16 funnyFruit('u');

* Lines 4-8 constitute the **definition** of the function funnyFruit.
* Line 4 is called the **header** of the function (not to be confused with lines 1 and 2 which are the header for the program file). A function header consists of:
  + The keyword def, which tells Python a function is about to be defined.
  + The name of the function, which follows the same rules as names of variables
  + Parentheses which contain a comma-separated list of names. This list is called the **parameter list**. In this simple example there is only one parameter.
  + A colon, which indicates that the header is done (just as with if, for, and while).
* Line 5 is called a doc sting. It is a triple-quoted string that comes right after the header, before the first statement. The doc string documents what the function does.
* Lines 6/7 are a single statement (broken across two lines) that creates a string and assigns it to a variable. Line 8 the prints the string.
* Lines 12 through 17 repeatedly call the function, passing it a different vowel each time as an argument.
* Each time the funnyFruit is called, the value of the argument is assigned to the **parameter** letter and used in lines 6/7 to create the message.

**a.** Add the following two lines to the bottom of the main program in silly\_song.py. They illustrate using a variable as an argument and using a call to another function as an argument.

accented\_a = chr(int('e1',16));

funnyFruit(accented\_a);

funnyFruit(chr(int('fc',16)));

**i.** Write down the results of the two new calls to funnyFruit.

**ápples and bánánás**

**üpples and bününüs**

**ii.** Explain what is happening in the expression chr(int('e1',16))

**int(‘e1’, 16) converts ‘e1’ from base 16 to base 10, which returns 255**

**chr(255) returns accented character á.**

**b.** Modify the main program in silly\_song to produce the following output.   
Hints – Use a for loop. The character 'a' corresponds to the decimal number 97 in the ASCII table.

I like to eat...

apples and bananas

bpples and bbnbnbs

cpples and bcncncs

dpples and bdndnds

epples and benenes

fpples and bfnfnfs

gpples and bgngngs

hpples and bhnhnhs

ipples and bininis

jpples and bjnjnjs

kpples and bknknks

lpples and blnlnls

mpples and bmnmnms

npples and bnnnnns

opples and bononos

ppples and bpnpnps

qpples and bqnqnqs

rpples and brnrnrs

spples and bsnsnss

tpples and btntnts

upples and bununus

vpples and bvnvnvs

wpples and bwnwnws

xpples and bxnxnxs

ypples and bynynys

zpples and bznznzs

**3.** Open the file area\_calculator.py from Canvas (shown below). This program shows how to **return** a value from a function. Run it and explore.

1 ##Gets length and width from the user and calculates area

2 ##Uses robust input technique

3 ##Illustrates a funciton with a return value

4

5 def getNumber():

6 """ Get a number from the user and return it """

7 number = None;

8 while number == None:

9 try :

10 number = int(input("Enter an integer: "));

11 except ValueError :

12 print("Not a valid input.");

13 return number

14

15 ##Main program

16 x = getNumber();

17 y = getNumber();

18 print('If the length is', x, 'and the width is',y,\

19 'then the area is',x\*y);

Three important new things in this function:

* It does not have any parameters. The parentheses in the function header are empty. The parentheses when it is called contain no arguments. Functions without parameters can and do exist, but empty parentheses are still required when defining or calling them.
* Line 13 is critical. The keyword return tells Python that whatever value follows will get returned to main program. It also tells Python that the function is done. The flow of execution immediately moves back to the point where the function was called.
* On lines 16 and 17 the x = and y = are essential to assign the returned value to something. Without assignment the values disappear and cannot be used.

Modify the getNumber function so that a string passed as an argument will be used as the prompt for the input function inside the function getNumber. Name the parameter in the function body prompt. A run of the program should look something like this:

(input window) How would you like me to prompt you to add an integer?

(typed response) Integer. Now.

(input window) Integer. Now.

(typed response)ten

(interpreter) Not a valid input.

(input window)Integer. Now.

(typed response) 10

(input window) Integer. Now.

(typed response) 14

(interpreter) If the length is 10 and the width is 14 then the area is 140

**4. Default arguments**

So far, we've only looked at functions that take one input. However functions may take any number of arguments and may also have multiple return values if those are stored as a tuple or list or something similar. Since tuples can contain variables of different types, this doesn't limit the return values. Type the following into a new module in the editor:

def areaPerimeter(width,height):

return (width\*height, 2(width+height));

a,p = areaPerimeter(2,3)

print("The area is", a, "and the perimeter is", p);

Now suppose you were using these functions because you wanted to calculate the area and perimeter of prints you sold. However, most of the prints you sell have a height of 3 inches. It would get very repetitive to type in 3 every time you called the areaPerimeter function if the height argument is almost always going to be the same. An easy fix is to add a second function definition. Add the following to the above code:

def areaPerimeter(width,height=3)

return (width\*height, 2(width+height);

a2, p2 = areaPerimeter(2);

print("The area is still", a2, "and the perimeter is still", p2);

Here, height is a **default parameter value**. Play around with this example until you can explain below what default parameter values do.

**5.** Write a function to draw a regular polygon. The parameters for the function will be the number of sides and the length of each side. Below is the template you should use. It includes the program header, import statement, the header of the function, the doc string for the function, and the entire main program. What is hidden behind the grey box is the block of statements in the function body that actually direct the turtle to draw the regular polygon. It is your job to fill in this bit of the program.

##turtlePolygon.py - Draws a polygon with a turtle

##Practice with functions, while loops and the turtle module

from turtle import \*

def drawRegularPolygon(my\_turtle, number\_sides, side\_length):

""" Directs a turtle to draw a regular polygon """

internal\_angle = (number\_sides - 2)\*180/number\_sides;

turn\_angle = 180 - internal\_angle;

side\_number = 0;

while side\_number < number\_sides :

my\_turtle.fd(side\_length);

my\_turtle.left(turn\_angle);

side\_number += 1;

##Main program

cage = Screen();

cage.setup(width=400, height=400, startx=0, starty=0);

bernie = Turtle(shape = "turtle");

drawRegularPolygon(bernie,5,50);

**6. Using functions to solve math problems*.***

The Forever Green Nursery currently owns 7000 white pine trees. Each year the nursery plans to harvest and sell 12% of its trees and plant 600 new trees.

1. Create a python program trees.py that uses a function harvestPlant to calculate how many trees are left after one year. The function should have three arguments, the initial number of trees, the rate harvested per year, and the amount planted per year. Add default values of rate\_harvested =.12 and trees\_planted=600. Ask the user to type in the number of years over which the harvesting and planting will take place and use the harvestPlant function inside of a for loop to calculate the number of trees left after that time. Print out this number. Use this to calculate how many trees will be left after 10 years given the initial conditions above. How many trees will be left after 20 years?

**After 10 years: 5560 trees**

**After 20 years: 5159 trees**

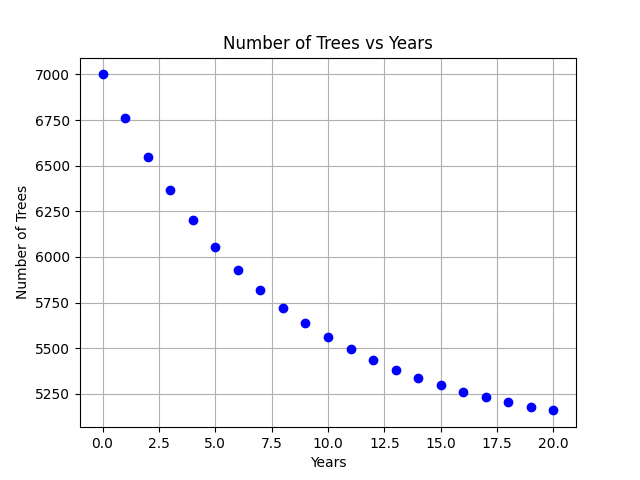
1. Modify this program to ask the user to input the initial number of trees, the harvest percentage, and the number of new trees planted each year as well. Then use this program to investigate the long term behavior of the nursery’s tree population. What is the long-term behavior? How does the long-term behavior depend on the three parameters given above (initial number of trees, harvest percentage and number of new trees planted per year)?

**If the initial number of trees times the harvest percentage is greater than the number of new trees planted per year, then the number of trees will drop to approach the number of trees at which the number of trees times the harvest percentage is the same as the number of new trees planted per year.**

**If the initial number of trees times the harvest percentage is less than the number of new trees planted per year, then the number of trees will rise to approach the number of trees at which the number of trees times the harvest percentage is the same as the number of new trees planted per year.**

**d.**  Plot this function for the given set of initial conditions.

**At the original 7000 initial trees, .12 harvesting rate, and 600 new trees per year, over 20 years:**

****

1. http://www.greenteapress.com/thinkpython/html/thinkpython004.html [↑](#footnote-ref-1)