INTPROG
Introduction to Programming

moodle.port.ac.uk

Practical Worksheet P05: Defining and Using Functions

Introduction

This worksheet is intended to get you started in defining and using functions as a first step to writing larger, more useful, programs. Work though this worksheet at your own pace and, as always, make sure you study each segment of code and try to predict its effects before you enter it, and afterwards make sure you understand what it has done and why. Furthermore, experiment with the shell until you fully understand each concept.

Functions that call functions

Begin by entering the following two function definitions on IEP's shell:

```
def sayHello():
    print("Hello, how are you today?")

def main():
    sayHello()
    sayHello()
```

(You need to press enter twice at the end of each function definition to let the shell know that you've finished.) Now, type the following code to ***call*** or ***invoke*** the main function:

```
main()
```

This results in ***three function calls*** (one from the shell to `main`, and two from `main` to `sayHello`).

Functions with parameters

Now, let's define a function that includes a ***parameter***:

```
def greet(person):
    print("Hello", person)
```

Invoke the function a few times, each time supplying a different ***argument***:

```
greet("Sam")
myName = "Jo"
greet(myName)
greet(myName + " Brown")
greet(42)
```

(The function is able to "greet" numbers as well as strings!).

If we now try to access the value of the parameter:

```
print(person)
```

Python will give an error message telling us that it doesn't recognise the variable `person`. This is correct. Although the `greet` function has a variable called `person` (i.e. its parame-

ter), this variable is local to the function definition: no `person` variable exists outside the function. If we were to introduce one:

```
person = "Fred"
```

then this is a ***different variable*** (even though it has the same name).

Now, try:

```
greet()
greet("Jo", "Brown")
greet
```

Here we see that we must supply the correct number of arguments (here, one) to a function (i.e. the number of arguments in a function call must equal the number of parameters in the function definition). The last line fails to invoke the function at all; instead it just tells us that `greet` is a function. Finally, try:

```
greet = 20
greet("Jo")
greet
```

Here we have made a mistake and "lost" the definition of the function; `greet` is now an integer-valued variable!

Functions that return values

Let's now define a function that ***returns*** a value to the caller:

```
def product(x, y):
    return x * y
```

Let's invoke this function a few times:

```
product(3, 4)
product(3.5, 2)
z = product(4, 2)
z
a = 2
product(a, 3 + 7)
print(a)
product(2, 2) + product(3, 4)
product(product(2, 3), 4)
```

We see that this function returns the product of two numeric values, and that a call to `product` is simply another example of an ***expression*** that can be used just about anywhere.

Now, try the following:

```
product("hello", 5)
product(3, "bye")
product("hello", "bye")
```

Can you explain what is happening here?

A more complete example

Download a copy of the interest.py file from the Moodle and study the functions `calcFutureValue` and `futureValue` which together give a two-function solution to exercise 9 of Practical Worksheet P01.

Experiment by invoking these functions a few times until you fully understand their operation. Notice that both functions use variables called `amount` and `years`. It is important to understand that these are totally separate variables (since variables are local to the functions in which they are defined). So, although `calcFutureValue` changes the value of its amount variable, the value of `futureValue`'s `amount` variable does not change at all.

Inputs & parameters; outputs and returns

Finally, before beginning the programming exercises, make sure you fully understand the difference between the above terms; in particular:

- Inputs and parameters are different things: an ***input*** is a value supplied by the user of the program (we usually use the function `input` to obtain inputs); a ***parameter*** is a special variable used to name an argument to a function (parameters appear between parentheses in the first line of a function definition).
- Outputs and returns are different things: an ***output*** is a value that a program displays on the screen (to produce textual output we use print statements); a ***return*** is used to communicate a value back from a function to another part of a program (we use `return` statements to do this).

For example, we see that the `calcFutureValue` function has two parameters and returns a value; whilst the `futureValue` function handles input and output. Make sure you read this and future weeks' programming exercises very carefully to ensure that your solutions do exactly what the questions ask. For example, if a question asks you to write a function that returns something, make sure that it returns a value instead of outputting it.

Programming exercises

Begin by downloading a copy of the pract05.py file from Moodle, and changing the details at the top of the file. Your solutions to this week's exercises should be added to your copy of this file, which should be made available for you to show to us in next week's practical session.

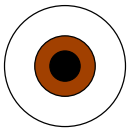
1. The pract05.py file contains a function `areaOfCircle` which has a ***parameter*** representing a circle's radius, and ***returns*** the area of the circle. Write a similar function called `circumferenceOfCircle` that has a `radius` ***parameter*** and ***returns*** the circumference of the circle.

2. Write a function `circleInfo` which ***asks the user*** to input the radius of a circle, and then ***outputs*** a message that includes both the area and the circumference of the circle (displayed to three decimal places); e.g. if the user enters a radius of 5, then the output message might be:

   ```
   The area is 78.540 and the circumference is 31.416
   ```

   Your function should ***call*** both the `areaOfCircle` and the `circumferenceOfCircle` functions to do the calculations.

3. The `drawCircle` function in pract05.py draws a circle on a graphics window with a given centre point, radius and colour. Complete the supplied `drawBrownEyeInCentre` function so that it ***calls*** `drawCircle` three times in order to draw a brown "eye" in the centre of a graphics window:



The radii of the white, brown and black circles should be 60, 30 and 15 respectively.

4. Write a function `drawBlockOfStars` which has two parameters `width` and `height`, and outputs a rectangle of asterisks of the appropriate dimensions. For example, the function call `drawBlockOfStars(5, 3)` should result in the following output:

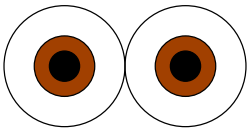
```
*****
*****
*****
```

   Now, write a function `drawLetterE` that displays a large capital letter E; for example:

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

   Your function should work by calling the `drawBlockOfStars` function an appropriate number of times.

5. Add code to the supplied `drawBrownEye` function so that, by calling `drawCircle` three times, it draws a single brown eye, where the graphics window, centre point and radius of the eye are ***all given as parameters*** to your function. Now, by using your completed `drawBrownEye` function, write a `drawPairOfBrownEyes` function (without parameters) that draws a pair of eyes:



   on a graphics window.

6. Write a function `distanceBetweenPoints` that has two parameters `p1` and `p2`, each of type `Point`, and returns the distance between them. This function should use the formula for Pythagoras' Theorem, as in practical worksheet P02. For example, the function call:

```
distanceBetweenPoints(Point(1, 2), Point(4, 6))
```

   should result in the value 5.0 being returned. (Hint: you'll need to use the `getX` and `getY` methods to get the x and y coordinates of points `p1` and `p2`.)

7. It should be clear that it is impossible to output letters such as A or O using only the `drawBlockOfStars` function. To allow for more complex letters such as these, write a new function `drawBlocks` that outputs up to four rectangles next to each other (consisting of spaces, then asterisks, then spaces and finally asterisks, all of the same height). The widths of the four rectangles and their common height should be parameters. E.g., a call: `drawBlocks(0, 5, 4, 3, 2)` will result in the output:

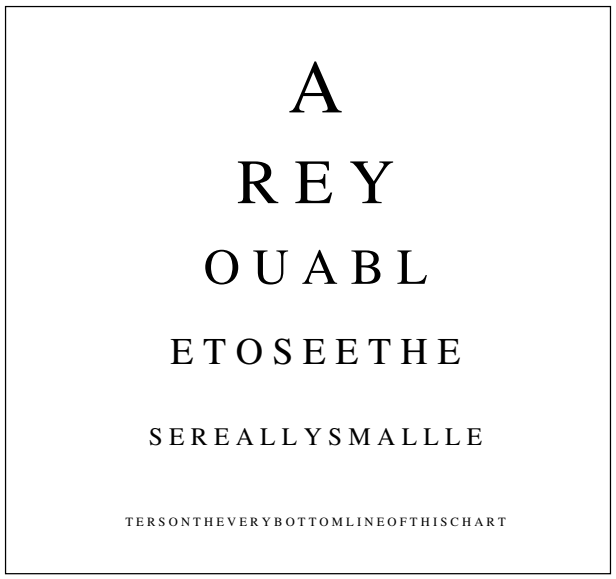
```
*****    ***
*****    ***
```

   (with no space before the first asterisks due to the 0 argument). Now, write a function `drawLetterA` that uses `drawBlocks` in order to display a large capital A in asterisks, such as:

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

8. Write a `drawFourPairsOfBrownEyes` function (which doesn't have parameters) that opens a graphics window and allows the user to draw four pairs of eyes. Each pair is drawn by clicking the mouse twice: the first click gives the centre of the left-most eye, and the second gives any point on the outer circumference of this eye. (Hint: This function should call the `distanceBetweenPoints` function from exercise 6 to obtain the radius of each eye, as well as the `drawBrownEye` function from exercise 5 to draw the eyes.)

9. [harder] Write a `displayTextWithSpaces` function which will display a given string at a given point-size at a given position on a given graphics window (i.e. it should have four parameters). The string should be displayed with spaces between each character (for example, `hello` would be displayed as `h e l l o`). Now, using this function, write another function `constructVisionChart` that constructs an optician's vision chart. Your function should first open a graphics window. It should then ask the user for six strings, displaying them on the graphics window as they are entered. The strings should be displayed in upper case, and from the top of the window to the bottom with descending point sizes of 30, 25, 20, 15, 10 and 5. (Make sure that the lines are well-spaced out — you might need to experiment a little with spacing.) For example, if the user enters the strings "a", "rey" "ouabl", "etoseethe", "sereallysmallle" and "tersontheverybottomlineofthischart", the window should look something like:



10. [harder] Write a `drawStickFigureFamily` function. This function should display a group of four or five stick figures (representing a family) in a graphics window. All the stick figures should be the same shape (e.g. that of exercise 1, worksheet P03), but they should be of different sizes and positions. Begin by copying your `drawStickFigure` function from pract03.py, and changing it so that it has three parameters, representing a graphics window, the position of the figure (a `Point`) and its size (an `int`). (What the position and size mean exactly is up to you.) Your `drawStickFigureFamily` function should contain just four or five calls to the modified version of `drawStickFigure`.

Written exercises

For each of the following sections of code, write down the ***output*** produced (i.e. the values printed to the screen) if the `main` function is called. For example, calling the `main0` function in the following code:

```
def mystery0(a):
    return a + 1

def main0():
    x = 5
    print(mystery0(x))
```

results in the output 6.

1.

```
def mystery1(a):
    print(a * 10)

def main1():
    x = 5
    mystery1(x)
```

2.

```
def mystery2(a):
    b = a + 1
    return 10 * b

def main2():
    y = 6
    print(mystery2(y))
```

3.

```
def mystery3(g):
    h = g + 1
    print(h)
    return 10 * g

def main3():
    y = 5
    z = mystery3(y)
    print(z)
```

4.

```
def mystery4(a):
    return a + 1

def main4():
    mystery4(41)
```

5.

```
def mystery5(a, b, c):
    return a + b + c

def main5():
    x = 3
    y = 7
    answer = mystery5(x, y, x)
    print(answer)
```

6.

```
def mystery6(a):
    return a - 1, a + 1

def main6():
    w = 4
    x, y = mystery6(w)
    print(x, y)
```

7.

```
def mystery7(a, b):
    print(a, b)

def main7():
    a = 7
    b = 3
    mystery7(b, a)
```

8.

```
def mystery8(x):
    value = 2 * x
    return value

def main8():
    value = 3
    answer = mystery8(value)
    print(answer, value)
```

9.

```
def mystery9(x):
    return x * 10
    print(x * 2)

def main9():
    y = 3
    answer = mystery9(y)
    print(answer)
```

10.

```
def mystery10(x):
    print(x * 3)

def main10():
    y = 4
    z = mystery10(y)
    print(z)
```