M30299 – Programming

Worksheet 5: Writing and Using Functions

The functions we have seen so far either receive values by asking the user to provide some input from the shell or interact with the graphic windows (in worksheet 3). Additionally, our functions only print values to the shell or display objects on the window. This worksheet starts you off with defining and using functions with parameters and return values as a first step to writing larger, more useful, programs.

Work through this worksheet and make sure you understand what it has done and why. Remember to have your attempted solutions to the programming exercises available for feedback in the following practical class.

For further support, book a one-to-one session with the Academic Tutors (Simon Jones or Eleni Noussi) using their Moodle page. Additionally, join the Discord channel using these instructions to get help with your questions.

What we have done so far ...

Create a file called `functions.py` in your usual folder and define a function called `greet`:

Editor

```
def greet():
    name = input("What is your name? ")
    print(f"Hello, {name}!")
```

Save `functions.py` and **call** the `greet` function in the shell. Recall that we call or invoke a function by placing its name followed by a pair of round brackets `()`.

Shell

```
greet()
```

We could have a `main` function that calls the `greet` function. Update `functions.py` as highlighted below:

Editor

```
def greet():
    name = input("What is your name? ")
    print(f"Hello, {name}!")

def main():
    greet()
    greet()
```

Run `functions.py` and call the `main` function in the shell.

Shell

```
main()
```

This results in three function calls: The shell is the first caller, and it calls the `main` function. Then the `main` function itself calls the `greet` function twice.

Functions with parameters

Update `functions.py` so that `greet` now takes a parameter. Also, update `main` so that you pass a value for this parameter:

Editor

```
def greet(name):
    print(f"Hello, {name}!")

def main():
    my_name = input("What is your name? ")
    greet(my_name)
```

Run `functions.py` and try the `main` function. Also invoke the `greet` function in the shell a few times, each time supplying a different argument:

Shell

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

Notice that the function can “greet” numbers as well as strings!

Now try to access the value of the parameter `name` defined in the function `greet`:

Shell

```
print(name)
```

Python will give an error message telling us that it doesn't recognise the variable `name`. This is expected. Although the `greet` function has a variable called `name` (its parameter), this variable is local to the function definition: no `name` variable exists outside the function.

If we were to introduce a variable `name`:

Shell

```
name = "Francesca"
```

then this is a different variable from the `name` variable defined in `greet` (even though both variables are called the same thing).

Now, try:

Shell

```
greet()
greet("James", "Brown")
greet
```

Here, we see that we must supply the correct number of arguments to a function we are calling (for example, `greet` takes one parameter). The number of arguments in a function call must equal the number of parameters given 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:

Shell

```
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

At the bottom of `functions.py` add the function `product`:

Editor

```
def product(a, b):
    return a * b
```

Notice how we separate the parameters with commas (`,`) inside the brackets.

Run `functions.py` and invoke the `product` function from the shell as shown below:

Shell

```
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. A call to product is simply another example of an **expression** that can be used just about anywhere.

Now, try the following:

Shell

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

Can you explain what is happening here?

As a small exercise, update the `greet` function so that it returns the greeting as opposed to printing it. Back in the `main` function, receive this greeting and print it:

Editor

```
def main():
    my_name = input("What is your name? ")
    greeting = greet(my_name)
    print(greeting)
```

Functions that return multiple values

Functions don't necessarily need to return one value. Depending on the situation, you may need to return multiple values from a function.

The `divide_and_product` function shown below takes two numbers as parameters and calculates their product and division (by calling the `divide` and `product` functions). It then returns both values back to the caller:

Editor

```
def product(a, b):
    return a * b

def divide(a, b):
    return a / b

def divide_and_product(a, b):
    product_result = product(a, b)
    divide_result = divide(a, b)
    return product_result, divide_result
```

Try this function in the shell by entering the following lines:

Shell

```
a, b = divide_and_product(24, 2)
a
b
x = 10
y = 5
product_result, divide_result = divide_and_product(x, y)
product_result
divide_result
```

A more complete example

Copy the code from the `pract5.py` file available on GitHub. We have two functions `calc_future_value` and `future_value` in this file The user calls the `future_value` which in turn calls the `calc_future_value` function.

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 `calc_future_value` changes the value of its `amount` variable, the value of `future_value`'s `amount` does not change at all.

Recap: Inputs & parameters vs. outputs & 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 (we usually use the `input` function to get inputs). A **parameter** is a special variable used to name an argument to a function. More specifically, parameters appear between parentheses in the first line of the function definition.
- **Outputs and returns are different things:** An **output** is a value that a program displays on the screen (for 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 in `pract5.py` that the `calc_future_value` function has two parameters and returns a value; whilst the `future_value` function handles input and output.

Programming exercises

Make sure you read these programming exercises carefully and do exactly what the questions ask. For example, if a question asks you to write a function that returns something, make sure that you use a `return` statement as opposed to a `print` statement.

Your solutions to the exercises should be written at the bottom of the downloaded file `pract5.py`. Solve as many problems as you can and be present at the next practical so that we can check your work and provide feedback.

1. The `pract5.py` file contains a function `area_of_circle` which has a **parameter** representing a circle's `radius`, and **returns** the area of the circle.

   Write a similar function called `circumference_of_circle` that has a `radius` **parameter** and **returns** the circumference of the circle.

2. Write a function `circle_info` 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). For example, 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 `area_of_circle` and `circumference_of_circle`.

3. The `draw_circle` function draws a circle on a graphics window with a given centre point, radius, and colour. Complete the `draw_brown_eye_in_centre` function so that it calls `draw_circle` three times to draw a brown "eye" in the centre of a graphics window. The radii of the circles should be 120, 60 and 30:

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

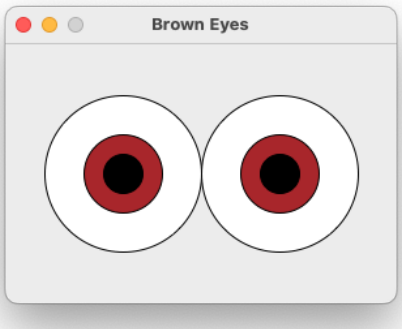
```
*****
*****
*****
```

   Now, write a function `draw_letter_e` that displays a large capital letter E:

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

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

5. Add code to the `draw_brown_eye` function so that, by calling `draw_circle` three times, it draws a single brown "eye". The graphics window, centre point and radius of the eye must all be given as parameters to your `draw_brown_eye` function. Now, using your completed `draw_brown_eye` function, write another function called `draw_pair_of_brown_eyes` (without parameters) that draws a pair of eyes on a graphics window:



6. Write a function `distance_between_points` that has two parameters `p1` and `p2`, each of type `Point` and returns the distance between them. This function should use the Pythagoras' Theorem (see worksheet 2). For example, the function call

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

   should result in the value 5.0 being returned.

7. Write a function `distance_calculator` that shows a graphics window to the user and using a `Text` object asks the user to click on two locations of the window. Your function should then call `distance_between_points` from the previous question and update the text of the `Text` object to display the distance between the points.

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

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

   Note that there are no spaces before the first asterisks due to the 0 argument. Now, write a function `draw_letter_a` that uses `draw_blocks` to display a large capital A in asterisks, such as:

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

9. Write a `draw_four_pairs_of_brown_eyes` function (which doesn't have any 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.

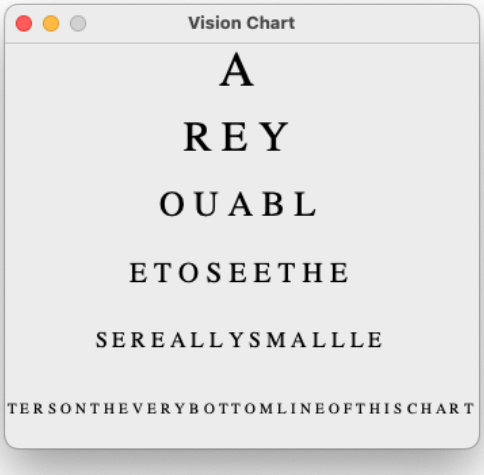
   Hint: This function should call the `distance_between_points` function to obtain the radius of each eye, as well as the `draw_brown_eye` function to draw the eyes.

10. [harder] Write a `display_text_with_spaces` 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 and in uppercase. For example, “hello” should be displayed as “H E L L O”.

Now, using this function, write another function `construct_vision_chart` 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 the one shown below:



11. [harder] Write a `draw_stick_figure_family` 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 (same as exercise 1, worksheet 3), but they should be of different sizes and positions.

    Begin by copying your `draw_stick_figure` function from `pract3.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 `draw_stick_figure_family` function should contain just four or five calls to the modified version of `draw_stick_figure`.