EMAT10007 – Introduction to Computer Programming

Exercises – Week 10. Functions 2

Getting Started: Pycharm IDE

Open PyCharm on linux lab computers

- Scroll down to bring up log in screen and log in with your UoB user name and password.
- Click activities (top left corner) to bring up the side panel.
- Click the grid of 9 dots to bring up applications.
- Choose JetBrains PyCharm
- When prompted about the user agreement click accept and read

Create a new project and Python file

- Click New project or File >> New project >> Pure python
- Unselect 'Create a main.py welcome script'
- Note the file location:
  /home/**UoB_username**/PycharmProjects/**your_projectname**/venv
  where **UoB_username** is your UoB username and rename **your_projectname** to be a name of your choice e.g. EMAT10007_exercises
- Right click on the folder icon with project name next to it (top left of window).
- Choose new >> python file
- Give your file a name e.g. week_1_exercises.py

Write and run code

Type some code and click the green play arrow at the top to run.

Save your project

File >> Save all to save your wor

Open a project you created previously

Click File >> Open >> /home/**UoB_username**/PycharmProjects/**your_projectname**/venv, Open >> New window

Rules for naming variables

- Variable names may contain letters or numbers
- Variable names must begin with a letter
- Variable names are case sensitive (`time` is not the same as `Time`)
- Some `keywords` are reserved by the Python language and cannot be used as variable names. For a full list of keywords reserved by Python, enter the following run the following comand in the editor you are using:

  ```
  help("keywords")
  ```

- Use a consistent naming convention:
  - `snake_case`: lower case letters, words separated by underscore (_)
  - `camel_Case`: first letter of each word capitalised, excluding first word
  - `Pascal_Case`: first letter of each word capitalised

Exercise 1 - Keyword arguments

1. Write a function called `subtract` that subtracts two numbers, a and b, and returns the result.
2. Now calculate `subtract(1, 3)` and `subtract(3, 1)`. Are the results the same? Why or why not?
3. Now calculate `subtract(1, 3)`, `subtract(a = 1, b = 3)` and `subtract(b = 3, a = 1)`. Are the results the same? Why or why not?
4. Create a function called `annotated_plot` that takes in five arguments: `x_data`, `y_data`, `xlabel` `ylabel`, and `title`. The function should create a plot of `y_data` as a function of `x_data`. In addition, labels for the x-axis, y-axis, and title should be created using the arguments `xlabel`, `ylabel`, and `title`. Use your function along with keyword arguments to create plots of the curves $y = x^3$ and $y = x/(x+1)$. The plots should be in different figures.

Exercise 2 - Default arguments

1. Create a function called `water_supply` that calculates how many litres of water a person should drink throughout their life. The function requires two arguments: `age` and `daily_amount` (daily intake of water). The default values for these arguments are `age = 75` and `daily_amount = 2`. The calculated value should be printed to the screen.
2. Call the function `water_supply` without passing it any arguments. Does the function work? Why or why not?
3. What happens if you call the function and pass it only one positional argument? Explain the value that is produced.

4. Now call the function and pass it one keyword argument. Does the function always work as expected?
5. Call the function and pass it two arguments. Is the value equal to the default value? Why or why not?

Exercise 3 - Variadic functions

Variadic functions can take any number of arguments, just like the print function, which works when used in the following way:
`print("Hello, I am", Age, "years old, born in the month of", Month, ".")`

1. Write a function that takes an unknown amount of numbers (integers or floats) and then:
   - Calculates the sum of the numbers
   - Calculates the product of all of the numbers

   and **returns** these values as opposed to printing them from the function.
2. Write a function that takes an unknown amount of numbers as its arguments and returns the `min`, `max` and `average` of those numbers.

Exercise 4 - Variable scope

Variable scope. It is important to understand a little bit about **scope** – an important concept in programming. The following exercises will demonstrate how variables are treated depending on whether they are declared inside (local scope) or outside (global scope) of a function.

1. Examine the following code and predict the value(s) printed to the screen. Then run the code to check your prediction.

```
def F():
    print(S)

S = "I hate spam"
F()
```

2. Examine the following code and predict the value(s) printed to the screen. Then run the code to check your prediction.

```
def F():
    S = "Me too"
    print(S)

S = "I hate spam"
F()
print(S)
```

3. Examine the following code and predict the value(s) printed to the screen. Then run the code to check your prediction.

```
def F():
    global S
    S = "Me too"
```

```
    print(S)

S = "I hate spam"
F()
print(S)
```

4. Examine the following code and predict the value(s) printed to the screen. Then run the code to check your prediction.

```
def F():
    T = "I am a variable"
    print(T)

F()
print(T)
```

Exercise 5 - Recursive functions

1. Write a recursive function that computes the factorial of a positive integer n. Recall that the factorial is defined as $n! = n \times (n-1) \times \ldots \times 1$. By definition, $0! = 1$. Check your code works by computing $5! = 120$. **Hint:** Notice that $n! = n \times (n-1)!$.
2. Write a recursive function called `Fib(n)` that calculates n-th number in the Fibonacci sequence using a recursive function. The following code

```
for i in range(1,15):
    print(Fib(i), end=" ")
```

   should print:
   `1 1 2 3 5 8 13 21 34 55 89 144 233 377`
   If you are struggling, start by researching how to calculate the Fibonacci sequence. There are lots of tutorials online, so be sure to search for implementations of the Fibonacci function using *recursion*. **Hint:** You will need to identify the "special cases" of your program which prevent the function from calling itself infinitely many times.
3. Read about the relative advantages and disadvantages of using *recursive* functions i.e. what happens if `n >= 100`? Is there another version of the Fibonacci function that does not use recursion? Can this be called on large values of `n`?