Computational Thinking

Computationeel Denken - T1ACD2E

KU LEUVEN

Exercise Session 4: Functions with return values

Exercise 1: Warm-up: Print vs Return

1.

Write a function `average()` which takes two numbers as parameters and prints their average. Execute your function with two numbers of your choice, to check whether the result is correct.

2.

Adjust your function **definition**[1] so the result is no longer printed, but returned (use the keyword `return`). Don't change your function **execution** (= function **call**)! What happens when you run your code?

3.

Now leave your function definition untouched (with the return), but add a print statement around your function call. It will look as follows: `print(average(6,8))`, but with numbers of your choice of course. Now what happens when you run your code?

For now it looks like returning is just more cumbersome than printing, since you need to add a print to see the output anyway. Of course returning also has its benefits, which we will attempt to demonstrate in the next exercise.

Exercise 2: Sharing info between functions

Open the file ***'lab4_ex2_start.py'***. This file contains two function definitions: one which converts a length from inches to centimeters, and one which calculates the total price of a roll of fabric based on its length (in cm) and the thickness of the fabric. The main-program is currently empty.

1. In the main, calculate the price for a roll of `"thin"` fabric, **20 inches** in length. Do this without changing anything in the function definitions.

   You'll have to call two functions: first to calculate how many centimeters 20 inches corresponds to, and then you use that value in the call to `calc_price()`. This means you'll have to run your program twice, since you will only see the result of the centimeter conversion when running the program. And it's this value which needs to be manually entered as a parameter in the call to `calc_price()`.

[1] Always be mindful of the difference between defining and calling a function!

2. Next, adjust the **definition** of the function `inch_to_cm()` so it no longer prints the result, but returns in. In other words, use the keyword `return`. Note that you will no longer see a result when calling the function.
3. Store the result of the call of `inch_to_cm()` in a variable named `result`.
4. Use this variable `result` as a parameter in the call of the function `calc_price()`, instead of the length in centimeters.

Run your program again. If you did everything correctly, you should get the same price as after step 1. This means that the actions you performed manually in step 1 (calling a function, reading its result and entering this result manually as a parameter for another function call) have now been performed automatically by the computer. This is the main benefit of the return statement: we can now pass information from one function to another.

Exercise 3: Smart kettle

Hint: read the full exercise before starting it, so you don't miss any important information or hints.

You work for a small Belgian startup that designs smart kettles. Business is booming, so you want to sell you product globally. You will have to make sure that your device can process temperatures in °C and °F.

- Define a function `F_to_C()` that converts temperature in °F to °C. The function has one parameter, a number (temperature in °F). The function returns a temperature in °C. The formula to convert between the two is

  °C = (°F - 32) / 1.8

- Define a function `alert_user()` that returns the following messages to the user. The function has a string as return value, meaning **it does not contain print statements**. It has one number (temperature in °C) as parameter.
  - If the water is hotter than 100 °C: `"Water is boiling"`
  - Between 85 °C and 90 °C: `"Ready to make tea"`
  - Between 40 °C and 45 °C: `"Perfect for a foot bath"`
  - In any other case: `"Current water temperature is X °C"`, with X the correct temperature.

- Write a main program that first calls the function `F_to_C()` to convert a certain temperature from Fahrenheit to Celsius, storing the result in a variable `temp_converted_to_C`. Next, use this variable as a parameter for a call to `alert_user()`. Store the return value of this function call in a variable as well, and finally print this variable (failure to do the last part will result in no output being shown).

Hint: in `alert_user()` *you have to return a string with a variable value in it. Take a careful look at the screenshot below. The variable* `age` *is an integer. With the* `str(…)` *function we can convert it to a string, which allows us to simply combine the string parts. We will have to type each space explicitly.* ***Remember this approach, you will have to use it regularly throughout the semester!***

Exercise 4: Good-class indicator

Group T wants to reward classes full of good students with free coupons for the *proffentap*[2]. You – a good student – have been tasked with writing a program that determines whether a class is good or not. In a good class every student takes every exam. If at least one student gets a 0/20, the class is not eligible for the title. If the average grade of a class is above 12/20, they get coupons. Above 16/20 the class gets a free luxury dinner. If the average grade is 20/20, the teachers get suspicious and contact the exam ombuds...

We will assume that every class is made up of exactly 5 students.[3] Grades are whole numbers between 0 and 20. You may assume only numbers between 0 and 20 will be entered by the users of your code.

- Define a function `average()` that has five integers as parameters: the grades of the students. The return value is the average of these numbers.
- Define a function `has_zero()` that also has the five grades as parameters. If at least one of these grade is zero, the return value is the Boolean value `True`, else `False`.
- Define a function `reward_class()` with two parameters: the average grade of a class and whether or not one of the students got a zero (= Boolean value). This function returns a string:
    - If there was a zero: `"Too bad, no prize"`[4]
    - If there was no zero:
        - <12/20: `"Too bad, no prize"`
        - >=12 and <16: `"Free coupons"`
        - >= 16 and <20: `"Free meal"`
        - = 20: `"Exam ombuds notified due to unprecedented score"`

[2] Don't worry, you'll find out…

[3] Later in the semester we will learn how to work with lists of a generic size.

[4] For non-native speakers: priZe = something you win in a contest, priCe = how much something costs.

- Write a main program that calls all functions and stores their return values in variables (!). Make sure your program prints the message for the user to the shell without there being print statements in the function definitions.

Exercise 5: Budget tracker

Make sure to have processed the document "Intro to test code" before attempting this exercise!

Life as a student in Leuven is expensive. Let us write a program to keep an eye on our expenses.

You can start this exercise from a Python file: ***'lab4_ex5_start.py'*.** We have written some test code (that starts at line 21) that will test all code you add once you run the program. You must finish the function definitions. They already contain the names of the functions, the correct parameter list and a return statement. **The return statement of the given functions is still wrong, you will have to change it.** You may not change the test code, but you can look at it to see how it works and what is expected of your functions.

Hint: your functions don't have to return the euro sign (€), just the numbers.

- `supermarket()` returns how much money you spend in the supermarket. The parameter `weekday` is a string that can only contain the values `"Monday"`, `"Tuesday"`, `"Wednesday"`, `"Thursday"` and `"Friday"`. You may assume our test code only uses these values. If you go to the supermarket on a Monday, you buy groceries for the entire week and pay € 40, on every other day you pay € 10. If you are throwing a party, you will pay more: `bought_alcohol` is a Boolean value that indicates whether you bought alcohol. If so, you pay twice as much.
- `buy_textbooks()` returns how much money you spend on books, based on your `willingness_to_pay`. This is an integer that indicates how much money you are willing to spend on text books:
    - = 0: You don't buy books and try to pass all courses using only material available on Toledo. The function returns € 0.
    - = 1: You buy books second-hand and are able to get them at half price.
    - = 2: You buy the books new. This costs € 400.
    - = 3: You buy all the books and out of interest you also buy extra literature on the subjects. You pay € 300 more than the normal price.
    - If you buy your books new (so only in cases 2 and 3) you can buy an `acco_card`. If you buy it, this Boolean value gives a 10 % discount, but the card itself does cost € 30.

*Hint: make sure the Acco card is only taken into account in case 2 and 3, don't forget to add the € 30 price of the card and make sure that the discount is given **only** on the books and thus not on the € 30 of the Acco card itself.*

- `calc_fine()` returns the fine for riding your bike drunk. *Rewrite the function from session 3 exercise 7* so that the fine is returned rather than printed to the shell. Careful: the function should return a number and not a string! The parameter `alcohol_pct` indicates how much alcohol is in the cyclist's blood **per mille**:
    - alcohol level < 0.5‰ : € 0;
    - 0.5‰ to 0.8‰: € 50;
    - more than 0.8‰ to 1.5‰: the fine rises linearly from € 50 at 0.8‰ to € 150 at 1.5‰;
    - more than 1.5‰: € 250.

Hint: This time you don't have to take negative alcohol levels into account. Think about the difference between <, <=, > and >=.

Exercise 6: Simple Pokémon Rip-Off

Since 1996 the Pokémon franchise grossed more than 92 billion dollars, 17 billion of which from videogame sales. Let us try to capitalize on this by making an extremely simplified version.

For this exercise there is a Python file to get you started: ***'lab4_ex6_start.py'.*** This file contains test code, analogous to the previous exercise.

Hint: we test each function independently, so you can continue with other functions if you get stuck.

- `attack_is_valid()` tests whether the attack power is a valid value. If the attack power is larger than zero, it is valid. Returns a Boolean value.
- `defense_is_valid()` tests whether the defense power is a valid value. The defense power must be positive and cannot be larger than the attack power of the same monster.
- `calc_damage()` calculates how much damage an attack inflicts. The formula is as follows: attack power of the attacker minus defense power of the defender. An attack will always inflict at least 1 point of damage. An attack can be super effective (critical), in which case it deals double the damage.
- `calc_hitpoints()` calculates the remaining hitpoints and returns a string. The amount of remaining hitpoints is equal to the `current_hitpoints` minus the `damage` (both of which are integer parameters to the function).
    - If zero or less hitpoints remain: `"Target has fainted"`
    - Else: `"Target has X hitpoints left"`, with X the remaining hitpoints.

Hint: The test code is case- and space-sensitive, so make sure everything is exactly as asked.

- `heal()` checks how many hitpoints a monster has lost and heals it for a random amount. If a monster has all its hitpoints left, it will be healed for 0 points. If a monster has lost X hitpoints already, it will be healed for a random number between 1 and X (so NOT zero). The function returns the number of hitpoints healed.

Hint: in `heal()` *you have to generate a random number between certain bounds (minimum 1, maximum the amount of hitpoints you have already lost). There exists a predefined function in python for that:* `randint`***.***

`random.` **`randint`**`(a, b)`

Return a random integer *N* such that `a <= N <= b`.

The 'creators' of Python wrote this function – together with loads of other ones – allowing us to use them without writing them ourselves. The function is defined in a module named `random`**.** *A module is also a Python file from which we can call functions.* The module `random` contains a number of functions to generate random numbers.

- First we have to import the module (this has been done for you ***in 'lab4_ex6_start.py'*** in line 1 using the `import` keyword). It is as if we copied all the code from the `random` file and pasted it at the top of our program, which allows us to call all functions in the module.

- Whenever we call the function `randint` we have to explicitly mention from which module we are calling it. **The example below shows the syntax and the operation of the function:** every time we run this piece of code, we read 5 new numbers between 1 and 10 (boundaries included). Note that the exact same function call gives different results. These are our randomly generated numbers!