BASIC PROGRAMMING EXERCISES

by Olatz C. P. (a.k.a. Darkatom)

DISCLAIMER & NOTES

- These exercises are for practicing basic programming skills, regardless of the programming language being used.
- Sometimes, some programming languages don't allow certain things, which can go against what is being asked in the exercises. Any of those exceptions will be noted after the statements and, in most cases, a second option will be provided.
- If you notice something incorrect, an exception missing, have a possible solution, have a question, etc., you can send me an email to darkatomish@gmail.com. I will try to answer as soon as possible.

BEFORE YOU BEGIN

a) Make yourself some tea, is good for your nerves.
b) Make sure you have everything you need (tea, the correct compiler, a text / code editor, free time, tea, intent... you know).
c) Don't panic. If you have a general idea of what you have to do, but don't know how to get there, just write down the steps as comments. Follow them in order and rearrange when needed. It's better than to be blocked forever.
d) Try to do as many as you can, including "Stepping up a notch!" ones. Don't hesitate to ask, but try thinking / Googling a bit first. Writing things down or drawing them always helps.
e) Enjoy it. These exercises are for practicing, not for torturing you. If you get stressed, just relax and go to step a).

You will find the exercises in the next page.

VARIABLES

1. Print "Hello, world!".
2. Get user input (any input) and print that input.
3. Get user input (integer) and print if the number is even or odd.

4. Ask two integers from the user, add them, and print them to the user.
   i. Same, but substracting.
   ii. Same, but multiplying.
   iii. Same, but modulus.
   iv. Same, but dividing. If the user inputs a 0 as the denominator, print a message instead of dividing.

5. Ask two integers from the user, *a* and *N*, for example. Then, iterate N times, adding 1 to *a* each time. Do this with both the for-loop and the while-loop, separately. (**NOTE**: This will give you the same result as *a* + *N*.)
   i. Same, but adding 2 instead of 1.
   ii. Same, but asking the user for a third integer, which will be added to *a* each iteration.
   iii. Optionally, you can do also substracting, multiplying and modulus.

6. Do the 5th again, but dividing. Ask the user for the numerator (*a*), the denominator (*b*), and the loop range (*N*). Two variants:
   i. Divide *a* until it is 0, then print the result. Also, print how many times did it have to loop. For-loops cannot be broken, but can you make the while-loop have less iterations than the for-loop? (Tip: modify the condition).
   ii. Iterate *N* times, substracting 1 to *b* and printing *a*/*b* each iteration. Bear in mind that you can't divide if *b* is 0.

7. **STEPPING UP A NOTCH!** Can you make the 5th a function that works for adding and substracting? What arguments would it have? (Tip: substracting 1 and adding -1 produces the same result.)

ARRAYS

8. Make a function for printing arrays.
9. Ask ten numbers from the user. Print them (use the function in the 8th exercise).
10. Ask ten numbers from the user. Make the even numbers odd. Print the array.
    i. Same, but make the odd numbers even.

11. Ask ten numbers from the user. Add 1 to every even position of the array. Print the array.
    i. Same, but for odd positions.
    ii. Same, but adding an integer asked to the user (both even / odd positions).
    iii. Same, but instead of even / odd positions, every three positions. Watch out for the end of the array.
    iv. Ask the user another integer to the user, for example, *p*. Then, do the same as above, every *p* positions.
    v. Ask the user how many positions the array should have, then do iv. again.
    vi. Optionally, try substracting, multiplying, dividing and modulus.

12. Ask ten numbers from the user. Print the array reversed (last numbers are printed first). Three variants:
    i. Print directly in reverse (looping in reverse from 9 to 0).
    ii. Copy the array reversed into another array. Then print this second array.
    iii. **STEP IT UP A NOTCH!** Reverse the array by swapping the numbers inside of it. That is, exchange the one at middle-1 with middle+1, middle-2 with middle+2, and so on until 0 exchanges with 9. Make it a method that uses the printing function from the 8th exercise.
    iv. **STEP IT UP A NOTCH! (2)** Same as iii., but asking the user for the length of the array. Take into account that the middle is different for even / odd lengths.

13. Study the selection sort (https://en.wikipedia.org/wiki/Selection_sort), then program it. Make it a method that prints the results, but doesn't return the sorted integer array. Use the function programmed in the 8th exercise to print the array.
    i. Do it first with an array of ten integers asked from the user.
    ii. Do it again, but asking the length of the array from the user.
    iii. Reverse the array as in the 12th, and then do the selection sort again.
    iv. Have fun using any of the exercises from the 10th to the 11th before and after this one.

14. Same as 13th, but make it a function and return the array. Then, print the resulting sorted array from outside the selection sort function.
    - For C users, use a pointer to fake the returning, as C doesn't allow to return an array. This is intended for practicing pointers.

15. Make the exercises from the 10th to the 12th functions so they return the modified arrays.
    i. Then, combine them with the 14th.
    - For C users, use a pointer to fake the returning, as C doesn't allow to return an array. This is intended for practicing pointers.

POINTERS

16. (WIP)

FILES

NOTES:

- We *always* open the file before reading/writing it and close it after we are done.
- *Always* close the file when you are done.
- Unless specified (f.e., we ask the user for the address), the address to the file is hardcoded. That is, it is directly written in the code.
- It is always useful to make small functions to read/write files, if you are going to do that a lot in the same way. Just saying.

17. Create manually a .txt file with a sole character inside. Read the char and print it.
18. Create manually a .txt file and write "Hello, world!" inside. Read the line and print it.

19. Create manually a .txt file and write a text inside (it doesn't have to be too long, three lines is more than enough). Make sure there are new lines.
    i. Print every char in the file as it comes.
    ii. Count how many lines has the file. Print that number. (Tip: \n is also a char).
    iii. Print each line of the file as single strings.
    iv. **STEP IT UP A NOTCH!** Print each sentence in the file. A sentence ends with a dot (.). (Tip: You can do this manually or save some lines using a Split [not available in some languages, like C]).

20. Create a .txt file with an integer in each line. Read the integers and store them in an array. Then, print the array.
    i. Same, but sort the array first. You know how to do that from the 13th exercise.

21. Create (from code) a .txt with the name *hello*. Then, write "Hello, world!" in it.

22. Create (from code) a .txt with the name *array*. Then, write the numbers from 1 to 100 in it, using a line for each.
    i. Write the numbers from 1 to 100 in it, using a space to separate them this time, instead of a new line.

23. Ask the user ten numbers. Ask the user the name for a file. Create a .txt with that name, and print the numbers given by the user in it.
    i. Do both by lines and separated by spaces. You can name the second file the same name with a 2 at the end.

24. Create manually a .txt with the name *unsorted*. Write integers in it, one for each line. Read the integers from code and sort them. Write them to a file called *sorted*.
    i. Have two files with numbers instead of one. Read both of them into one single array, sort it and write it to the *sorted.txt* file.
    ii. Same, but ask the user for the names of the files.
    iii. Same, but ask the user for how many files does s/he want to read, and then ask for as many names.