CUBE
Project 0: How to say hello

Before we begin, let us lay out the structure of this (and future) project document. We will start with main goal, which will be specified in terms of *input/output behavior*. This means that you will told what the input to your program will be, and what the output should be. In general, that is how one starts a program. First, you figure out in what form the input is provided, and figure out what the output looks like.

For future projects, these tasks may be quite complex, so you will need to break it down into simpler, smaller blocks. The next sections will do that, by suggesting simpler input/output behavior. Think of these problems as stepping stones, that will help you build the final solution. Until you are very comfortable with programming (and even then), always start small. Write small, correct programs before tacking a bigger problem.

1 The goal: saying hello with command line arguments and files

The goal of this project is quite simple. Your program is provided an input file with a *single line* that a word w and a positive integer k. It will generate a file that has the text "hello w" printed k times. For example, suppose the input file has:
sesh 3

The output file should have:

Hello sesh
Hello sesh
Hello sesh

The input and output files are provided as *command line arguments*. Let me explain using python as the example language. Remember that all coding and executing must happen on the command line. Suppose your program is written in the file `hello.py`. Then, we will run your program using the following command.

```
python3 hello.py <INPUT FILE> <OUTPUT FILE>
```

So the names of the input and output files are provided in the command to execute the program. In the case of C/C++, you will first compile the code using the command: `gcc hello.c`. This generates an executable, typically `a.out`. You will run the command `./a.out <INPUT FILE> <OUTPUT FILE>`.

2 The path: breaking into smaller tasks

If you're familiar with basic file I/O and command line arguments, you can go ahead and code this up. If you are unsure on how to do this, the following steps might be helpful.

1. Start by just writing a hello world program. On running it, it simply prints "Hello world" in the console.

2. Let us add one more element. The program should now print "Hello world" in a file `output.txt`.

3. Now we add one more element. The program takes in a single command line argument, the output file, and prints "Hello world" in that output file.

4. You have already dealt with two major components: file I/O and command line arguments. You are ready to solve the full problem. You are left with reading from a file and parsing text.

2.1 Help!

Maybe you have no clue how to read/write into files and get command line arguments. No problem! To be able to search effectively on the internet, you need to right *keywords* to search for. In this case, if you search for "python reading and writing files" or "python command line arguments", the first few hits on any search engine will give you what you need. Read those results, tinker with the code given, and develop it further into the solution you need. *This is a very important skill!*

3 Error handling: the lurking enemy

This seems like a fairly easy and short program to write. If you don't handle any errors, this is. Once you've written the basic program, think about all the ways that the input does *not* conform to the right structure. And this gives you a number of ways in which your code can crash. Ideally, you should do some error handling. Meaning, your program detects errors, and terminates with an error message telling the user what the problem is.

Here are some errors your program should detect.

- Insufficient number of arguments. Your program should terminate with a *usage statement*, explaining what the right format for the command is. For example, the message could say:

```
Insufficient number of arguments.  Run as:
python3 hello.py <INPUT FILE> <OUTPUT FILE>
```

- Input file does not exist.
- First line of input file does not have the correct format.

4 Going above and beyond

Now for a bigger challenge. Suppose your program had to translate the greeting "Hello" in a desired language. So the program is run as:

```
python3 hello.py <INPUT FILE> <OUTPUT FILE> -l <LANGUAGE>
```

The "-l" argument is optional. If not provided, the program works as before. But the user could specify the language as (say) Javanese, in which case one should use the greeting "selamat". In the repo, I have provided a file `hello-languages.txt` which contains "hello" in about a hundred languages. Think about how you would write such a multilingual hello world program.