Examples of good documentation further explanation

Please refer to "PEP 257 -- Docstring Conventions" for detailed instructions on Python conventions for docstrings (and "PEP 8 -- Style Guide for Python Code" for style conventions). A good guide for commenting is available in https://realpython.com/python-comments-guide/.

Note that we will be using clear variable names and Python type hints to reduce the amount of comments needed to understand the code.

Please make sure you use good documentation and coding practices. You will see more examples in our solutions to Tutes and Prac0.

Modules

Note we start with three double quotes, in a single phrase that succinctly expresses the file's aim, starting with uppercase and ending on a full stop. Then there is a blank line, and then more information if needed without indentation. We finish the comments for the module with a line containing only three double quotes (no empty line before it!).

Author

You will be asked to always provide the author name in your file, to acknowledge this is your own work.

```
__author__ = "Maria Garcia de la Banda"
```

Import

I have put an in-line comment to illustrate its syntax but it was not needed. Please use in-line comments sparingly, mainly to illustrate high-level information that is important to understand at a particular step. We will see examples later in the course

```
from typing import List, Union, TypeVar  # this is how we import
```

Functions

> **max_age**
>
> The name of the variables and the types make the function so clear we are not going to require you to provide extra documentation. Note the use of 4 spaces for indentation (spaces are preferred to tabs), underscores rather than uppercase to separate the names parts of identifiers, and a single space between arguments and between operands.

has_a_negative

Again, the function is so clear we will not ask you to write a comment. I added one here with the complexity and pre/post conditions to illustrate how to do this when we ask you to add them. Note that the first line gives the aim of the function. If you have not seen complexity before, that's OK; you will find a guide later in Week 4, before you need it.

string_to_number

This time I wanted to illustrate how to comment on the exceptions thrown by a function. If you have not seen exceptions before, that's OK too; you will find a guide later also in Week 4, before you need it.

disjoint

this time I wanted to illustrate three things. First, how to define and use a "polymorphic" type-hint, where type variable T is used to indicate repeated occurrences of the same type, whatever that type it is. In the example T is used to represent the type of the elements in the two lists (which has to be the same). Second, good names and type hints are not always enough to properly document a function, so this shows you how to use :param: to document the arguments more thoroughly. And third, an example of a more difficult kind of complexity, with which you will become familiar later in the unit.

main

A single line is enough here to explain what the purpose of this main function is. Note that in this case the three double quotes appear in the same line. I also used it to show an in-line comment: as you can see, it provides high level info that is not so easy to "get" from reading the code.

if __name__ == '__main__'

it ensures that whenever this file is read by the python interpreter/compiler (when this happens, your module is the main program and, thus, variable __name__ is assigned to string ""__main__") the code in the "if-then" condition is executed (in this case, a call to the main function).