CSc 110 Assignment 10:
Classes and instance data

Reminder: Your code is to be designed and written by only you and not to be shared with anyone else. See the Course Outline for details explaining the policies on Academic Integrity. Submissions that violate the Academic Integrity policy will be forwarded directly to the Computer Science Academic Integrity Committee. All materials provided to you for this work are copyrighted, these and all solutions you create for this work cannot be shared in any form (digital, printed or otherwise). Any violations of this will be investigated and reported to Academic Integrity.

Learning Outcomes:

- How to write and call instance methods
- How to write and call functions that operate on lists of objects

Getting started

1. Download the starter files provided: `assignment10.py`, `date.py` and `pet.py` and open them in Wing
2. Download the provided input csv files to the same directory as the starter files
3. We cannot upload an empty file to BrightSpace so you will need to create an empty file in the same directory as your starter files and name it e`mpty.csv`
4. Design the methods/functions according to the specifications in the documentation provided in the starter file and using the further explanation given in the ***Method/Function Specifications*** section below.

Submission

1. Double check your file before submission to avoid a **zero grade** for your submission for issues described in the Grading section of this document:
    a. Open and run `assignment10.py`, `date.py` and `pet.py` in your Wing.
    You should see no errors and no output to screen or to a file.
    If there are errors, you must fix them before submitting.
    If there is output printed to the shell or to a file, find and remove any top-level print statements and/or top level function calls.
    b. At the shell prompt in Wing (>>>) type the following `import assignment10`
    You should see no errors and no output to screen or to a file.
    If there are errors, you must fix them before submitting.
    If there is output printed to the shell or to a file, or read from a file, find and remove any top-level print statements and/or top level function calls.
    c. At the shell prompt in Wing (>>>), make calls to the required functions to ensure you have named them correctly. Ensure the function is exhibiting the expected behaviour when called and does not contain any unexpected output. You should be able to make calls to the following functions with expected input. Note: When making these calls, you must precede it with the module name ie. `assignment10.read_file('pet_data.csv')`
        i. is_after (method in Date class)
        ii. == to compare 2 pet instances (`__eq__` in Pet class)
        iii. read_file
        iv. find_pet
        v. get_all_of_species
        vi. get_latest_birthdate
        vii. get_youngest_pets
    d. Ensure you have closed any files that you have opened!

2. Upload your `assignment10.py`, `date.py` and `pet.py` to BrightSpace

Grading:

- Late submissions will be given a **zero grade**.
- The files you submit must be named **`assignment10.py`**, **`date.py`** and **`pet.py`**
  The filenames must be EXACT for them to work with our grading scripts. Errors in the filenames will result in a **zero grade**.
  Example mistakes often made by students include but are not limited to: spelling errors, different case letters, space characters and incorrect extension (not using .py)
- Your function names and the order of your function arguments must match EXACTLY as specified in this document or you will be given a **zero grade**. Use the example tests we give you to ensure your function header is correct.
- Your submission must not contain any print statements that are not required in the specification or any top-level calls to functions. This unexpected code can cause the automated tester to crash and will result in a **zero grade**.
- We will do **spot-check grading** in this course. That is, all submissions are graded BUT only a subset of your code might be graded. You will not know which portions of the code will be graded, so all of your code must be complete and adhere to specifications to receive marks.
- Your code must run without errors with Python 3. If you tested your configuration with setup.py file this would have verified you are using Python 3. Code that generates errors cannot be tested and will be given a **zero grade**.

Marks will be awarded for correctness, considering:

- the function signature matches the description given (has the name and arguments EXACTLY as specified)
- the function has the expected behaviour

and for code quality according to software engineering properties such as:

- documentation in docstring format: type hints, purpose, examples
- Test coverage – examples within the function docstring cover all boundary cases of all conditions within your function
- readability
  - use of whitespace
  - splitting complex computation or long statements across multiple lines
- meaningful variable names
  - lower case, starting with an alpha-character
- proper use of constants (avoid magic numbers)
  - defined above all function definitions
  - in UPPERCASE
- use of code constructs (functions, variables) to:
  - eliminate redundant code and redundant computation
  - make complex code easier to read

Method/Function Specifications:

Familiarize yourself with the code provided in `date.py` and `pet.py` You will design a method in each of `date.py` and `pet.py` according to the specification given below.

In the given specifications we have omitted examples as coming up with those examples is part of the problem solving process. Feel free to use the forum to clarify any ambiguities that you encounter in the specifications when trying to establish your examples. Don't forget your FULL documentation.

1. Within `date.py` implement a method called `is_after` that takes as arguments the hidden reference to the `Date` instance itself (*self*) and an additional instance of a date (*other*).
   The function should return True if the date represented by *self* is after that represented by *other* in the calendar and False otherwise.

2. Within `pet.py` implement the `__eq__` method so that it overrides the existing implementation automatically inherited. When your implementation is complete, when an instance of a `Pet` is compared to another `Pet` using == it should return True if the name, species and birthdate of the two `Pet` instances are the same and return False otherwise.
   Note: the header of this function must have the correct function name and number of arguments for this overriding to work. See `date.py` for an example of this.

3. Within `assignment10.py`, complete the implementation for the functions marked TODO according to the documentation given. You must understand how to call the instance methods within the Date and Pet classes in order to complete these function designs.