OPS445 Online Assignment 1

Overview

When making back up of data files or log files, it is a very common practice to name the backup directories and/or files based on the date the backup was done. In order to restore or locate the directory/file, we often need to find out the backup date from today's date.

The computational task for this assignment is to design an algorithm and write a python script according to your algorithm with appropriate functions. The script should take a date in the "DD-MM-YYYY" format as well as a number of days, and return the date when that number of days is applied to the date. That is, if the user enters "18-06-2020" and "7", the script will return "25-06-2020". Similarly, if the number is a negative number, the script will return a date in the past.

Required Files

- assign1.py contains starter code for your assignment. You should commit all of your code to this file.
    - Requires VPN to access or you can access via Blackboard
- checkA1.py is a check script that will help you evaluate your assignment. For full marks, your assignment should pass all checks.
    - Requires VPN to access or you can access via Blackboard

The First Milestone (Due June 19th 11:59 PM)

- Your first task will be to complete and submit an algorithm document. This document should be named **algorithm_[student_id].txt**. This file should be plaintext. The document will contain two sections:

```
* A description of how the "after()" function works. The
"after()" function is provided to you in the assign1.py
template. Open the file, and use clear English to describe what
line of code does in such a way that a competent programmer
could reproduce the code without seeing it firsthand.
 * You will then apply the same principles to create an
algorithm for "before()", and "dbda()".
```

- The object of this milestone is not to have a 100% perfect algorithm, but to plan ahead and anticipate challenges and issues with the assignment. The milestone will also give your professor an opportunity for feedback.
- Here is an basic introduction to Algorithm

- While you are working on the step-by-step instructions, note that there are different number of days in each month and some years have 365 days and some years have 366 days.
- You should also do some research to find out when we started using the Calendar in the current form. (This will pose a limit on the validity of your algorithm.)

The Assignment (Due July 11th 11:59 PM)

As stated before, your code will be inside the file "assign1.py". The first step will be to clone (or create) the Assignment 1 repository.

Assignment Requirements

Required Modules and Functions

Your python script is allowed to import only the sys module from the standard library, and only to handle command line arguments.

Based on the algorithm you have designed for this assignment, you should at least have the following four functions defined in your python script (see later section on the purpose of each function) in order to get a passing grade for this assignment:

- dbda()
- after()
- before()

You should also create additional functions to improved the re-usability of your python code by adding the following functions to earn the maximum possible mark for this assignment:

- days_in_mon()
- leap_year()
- valid_date()
- usage()

Coding Standard

Your python script must follow the following coding guide:

- PEP-8 -- Style Guide for writing Python Code

Command Line Argument to be supported

- You will provided with a file called assign1.py.
- Be sure that this python script has your name and student_id set inside the docstring.
- Your python script must support one or two command line arguments only: the first should be a valid date in DD-MM-YYYY format and the second an integer (negative or positive).

- If there are no arguments, more than two arguments, or an invalid date, your script should display the correct usage message and exit.

Documentation

- Please use python's docstring to document your python script (script level documentation) and each of the functions (function level documentation) you created for this assignment. The docstring should describe 'what' the function does, not 'how' it does.
- Refer to the docstring for after() to get an idea of the function docstrings required.

Authorship Declaration

All your Python code for this assignment must be placed in a **single source python file**. Please complete the declaration **as part of the docstring** in your Python source code file (replace "Student Name" with your own name).

Tests and Test results

You must name your python 3 script as `assign1.py`. The script should accept two command line arguments, the first one is the date in "DD-MM-YYYY" format, and the second one is the number of day from the given date, a positive value indicates the number of days after the given date, and a negative value indicates the number of days before the given date. If the "DD-MM-YYYY" format is broken, your script should give an appropriate error message. Invalid months (>12) or invalid days of month(different for each month), should be detected and give appropriate error messages. For example:

- **`python3 assign1.py 01-01-2019 1`**, and the output should be

```
02-01-2019
```

- **`python3 assign1.py 01-01-2019 -1`**, and the output should be

```
31-12-2018
```

- **`python3 assign1.py 01-06-2020 365`**, and the output should be

```
01-06-2021
```

- **`python3 assign1.py 01-01-2019 365`**, and the output should be

```
01-01-2020
```

- **`python3 assign1.py 01-01-2021 -366`** , and the output should be

```
01-01-2020
```

- **`python3 assign1.py 01-13-2018 1`**, and the output should be

```
Error: wrong month entered
```

- **`python3 assign1.py 99-01-2020 1`**, and the output should be

```
Error: wrong day entered
```

- **`python3 assign1.py 2018 2`**, and the output should be

```
Error: wrong date entered
```

If there is too few or too many command line arguments given, display the proper usage:

- `Usage: assign1.py DD-MM-YYYY N`

Script structure and sample template

The following is a brief description of each function:

- The dbda() function should be the main function of your script. The dbda() function will take a date in "DD-MM-YYYY" format, a positive or negative integer, and return a date either before or after the given date according to the value of the given integer in the same format. Your dbda() function should delegate the actual calculation of the target date to either the after() function or the before() function.
- The before() function will take a date in "DD-MM-YYYY" format and return the date of the previous day in the same format.
- The after() function will take a date in "DD-MM-YYYY" format and return the date of the next day in the same format. Next paragraph is a sample python code for the after() function. To earn the maximum possible mark for the assignment, you should modify the sample after() function to make use of the days_in_mon() function.
- The leap_year() function will take a year in "YYYY" format, and return True if the given year is a leap year, otherwise return False.
- The valid_date() function will take a date in "DD-MM-YYYY" format, and return True if the given date is a valid date, otherwise return False plus an appropriate status message. The valid_date() function should make use of the days_in_mon() function.
- The days_in_mon() function will take a year in "YYYY" format, and return a dictionary object which contains the total number of days in each month for the given year. The days_in_mon() function should make use of the leap_year() function.
- The usage() function will take no argument and return a string describing the usage of the script.

Sample code for the after() function

```
# Return the date in DD-MM-YYYY after the given day
#
def after(today):
    if len(today) != 10:
       return '00-00-0000'
    else:
       str_day, str_month, str_year = today.split('-')
       year = int(str_year)
       month = int(str_month)
       day = int(str_day)

       lyear = year % 4
       if lyear == 0:
          feb_max = 29 # this is a leap year
       else:
          feb_max = 28 # this is not a leap year

       lyear = year % 100
       if lyear == 0:
          feb_max = 28 # this is not a leap year

       lyear = year % 400
       if lyear == 0:
          feb_max = 29 # this is a leap year

       tmp_day = day + 1 # next day

       mon_max = { 1:31, 2:feb_max, 3:31, 4:30, 5:31, 6:30, 7:31, 8:31,
9:30, 10:31, 11:30, 12:31}
       if tmp_day > mon_max[month]:
          to_day = tmp_day % mon_max[month] # if tmp_day > this month's
max, reset to 1
          tmp_month = month + 1
       else:
          to_day = tmp_day
          tmp_month = month + 0

       if tmp_month > 12:
           to_month = 1
```

```
            year = year + 1
        else:
            to_month = tmp_month + 0

        next_date = str(to_day).zfill(2)+"-"+str(to_month).zfill(2)+"-
"+str(year).zfill(2)

        return next_date
```

Rubric

Task	Maximum mark	Actual mark
Program Authorship Declaration	5	
Program usage	5	
after() function	5	
before() function	10	
dbda() function	10	
script level docstring	5	
leap_year() function	5	
valid_date() function	5	
check script results	15	

First Milestone	10	
Second Milestone	10	
Total	85	

Due Date and Final Submission requirement

Check with your professor for the due date for your section.

Please submit the following files by the due date:

- [ ] your algorithm document, named as 'algorithm_username.txt', to Blackboard. This is your first milestone.
- [ ] your python script, named as 'assign1.py' **submitted to Blackboard.**
- [ ] the output of the checking script checkA1.py, named as 'a1_output.txt', should be submitted to Blackboard as well.