Project Description: Working with Dates

Practice with Lab Sandbox

In many scripting tasks, you will need to work with dates. You often have collections of data that come from different times and dates. You may want to sort events based on when they occurred, aggregate information from a single day, calculate the time between two events, or do other processing based on dates.

In this project, you will write four functions to do simple processing on dates using Python's datetime module. This will help familiarize you with writing Python functions, using modules in Python, and working with dates in Python.

Preliminaries: Working on the Project

Coding Style

In this class, you will be asked to strictly follow a set of coding style guidelines. Good programmers not only get their code to work, but they also write it in a way that enables others to easily read and understand their code. Please read the style guidelines carefully and get into the habit of following them right from the start. A portion of your grade on the project will be based upon coding style.

Testing

You should always test your code as you write it. Do not try to solve the entire project before running it! If you do this, you will have lots of errors that interact in unexpected ways making your program very hard to debug. Instead, as you write each function, make sure you test it to ensure that it is working properly before moving on to the next function.

Throughout this course, we will be using a machine grader (OwlTest) to help you assess your code. You can submit your code to this Owltest page to receive a preliminary grade and feedback on your project. The OwlTest page has a pale yellow background and does **not** submit your project to Coursera. OwlTest is just meant to allow you to test your project automatically. Note that trying to debug your project using the tests in OwlTest can be very tedious since they are slow and give limited feedback. Instead, we *strongly* suggest that you first test your program using your own tests. Also, note that each OwlTest link is specific to a particular project. You need to come back to this page and click the link above to ensure that you are running the tests for this project.

When you are ready to submit your code to be graded formally, submit your code to the Dates assignment page for this project. You will be prompted to open a tool which will take you to the CourseraTest page. Note that the CourseraTest page looks similar to the OwlTest page, but they are *not* the same! The CourseraTest page has a **white** background and does submit your grade to Coursera.

Note: Due to recent changes in Google App Engine, we have updated the machine grader for this course. If you experience any unexpected issues with OwlTest or CourseraTest (e.g; the grader crashes or doesn't record your grade), feel free to email *interactivepython@rice.edu* with a short description of your issue and a copy of your code.

Project: Working with Dates

The project consists of multiple problems. Each problem will utilize functions you wrote for the previous problems, so we strongly suggest that implement and test these functions in the given order. We have provided the following template that you can use to get you started. It includes the signatures (name, parameters, and docstrings) for all of the functions that you will need to write. The code however, simply returns some arbitrary value no matter what the inputs are, so you will need to modify the body of the function to work correctly. You should not change the signature of any of the functions in the template, but you may add any code that you need to.

Problem 1: Computing the number of days in a month

First, you will write a function called `days_in_month` that takes two integers: a year and a month. The function should return the number of days in that month. You may assume that both inputs are valid (in other words, you do not need to write any code to check whether or not they are reasonable, you can just assume that the month input is a number between 1 and 12 and the year input is a number between `datetime.MINYEAR` and `datetime.MAXYEAR`).

Here is the signature for the `days_in_month` function:

```
    Inputs:
      year  - an integer between datetime.MINYEAR and datetime.MAXYEAR
              representing the year
      month - an integer between 1 and 12 representing the month

    Returns:
      The number of days in the input month.
    """
```

You need to write the code that implements this function.

Hints:

1. You do not need to check that the inputs are within the proper ranges. You can assume that any function that calls this one will already have checked that.
2. For this and all subsequent problems, be sure to familiarize yourself with the datetime module and make use of the capabilities that it provides.
3. One way in which to determine the number of days in a month is to subtract the first of the given month from the first of the next month. The result should be the number of days in the given month.

Problem 2: Checking if a date is valid

Next, you will write a function called `is_valid_date` that takes three integers: a year, a month, and a day. The function should return `True` if that date is valid and `False` otherwise. This function *should not* assume that the inputs are valid. Rather, this function should check that all three inputs combine to form a valid date, with a year between `datetime.MINYEAR` and `datetime.MAXYEAR`, a month between 1 and 12, and a day between 1 and the number of days in the given month. Notice that the function `days_in_month` that you wrote for Part 1 will be useful here!

Here is the signature for the `is_valid_date` function:

```
def is_valid_date(year, month, day):
    """
    Inputs:
      year  - an integer representing the year
      month - an integer representing the month
      day   - an integer representing the day

    Returns:
      True if year-month-day is a valid date and
      False otherwise
    """
```

You need to write the code that implements this function.

Hints:

1. Do not forget to make use of any function(s) that you have already written!
2. This function should just consist of conditional checks that each input is in the proper range.

Problem 3: Computing the number of days between two dates

Now that we have a way to check if a given date is valid, you will write a function called `days_between` that takes six integers (year1, month1, day1, year2, month2, day2) and returns the number of days from an earlier date (year1-month1-day1) to a later date (year2-month2-day2). If either date is invalid, the function should return 0. Notice that you already wrote a function to determine if a date is valid or not! If the second date is earlier than the first date, the function should also return 0.

Here is the signature for the `days_between` function:

```
def days_between(year1, month1, day1, year2, month2, day2):
    """
    Inputs:
      year1  - an integer representing the year of the first date
      month1 - an integer representing the month of the first date
      day1   - an integer representing the day of the first date
      year2  - an integer representing the year of the second date
      month2 - an integer representing the month of the second date
      day2   - an integer representing the day of the second date

    Returns:
      The number of days from the first date to the second date.
      Returns 0 if either date is invalid or the second date is
      before the first date.
    """
```

You need to write the code that implements this function.

Hints:

1. Be sure to first check that all of the inputs are valid. Do not repeat code. Make sure you reuse things that you have already written.
2. Make sure to check the cases where the function is supposed to return 0.

Problem 4: Calculating a person's age in days

Finally, you will write a function called `age_in_days` that takes three integers as input: the year, month, and day of a persons birthday. The function should return that person's age in days as of today. The function should return 0 if the input date is invalid (remember you have a function to check that!). The function should also return 0 if the input date is in the future.

Remember that you already have a function that returns the number of days between two dates that you wrote in Part 3. Which two dates could you pass to that function to solve this problem?

Here is the signature for the `age_in_days` function:

```
def age_in_days(year, month, day):
    """
    Inputs:
      year  - an integer representing the birthday year
      month - an integer representing the birthday month
      day   - an integer representing the birthday day

    Returns:
      The age of a person with the input birthday as of today.
      Returns 0 if the input date is invalid of if the input
      date is in the future.
    """
```

You need to write the code that implements this function.

Go to next item ✓ Completed