Functions

Overview

In this lab, you'll refactor the date processing code from the previous lab, so that it makes appropriate use of functions. This will make the code more readable, and will also aid reuse. You'll also explore the use of default arguments along the way.

If time permits, you'll have a go at defining variadic functions.

Source folders

Student folder : …`\Student\04-Functions`

Solution folder: …`\Solutions\04-Functions`

Roadmap

There are 3 exercises in this lab, of which the last exercise is "if time permits". Here is a brief summary of the tasks you will perform in each exercise; more detailed instructions follow later:

1. Refactoring Python code to use functions
2. Defining and using default arguments
3. Additional suggestions (if time permits)

Exercise 1: Refactoring Python code to use functions

In the *Student* folder, open `dateprocessing.py` in the editor. This is the solution code for the previous lab, so take a moment to familiarize yourself with the code.

Refactor the code so that it makes use of functions to improve modularity and readability. We suggest you define the following functions (and call them at the appropriate point in your code, obviously ☺). This is the main exercise in this lab, so don't feel rushed:

- `isLeapYear(year)`
  Takes a year as a parameter, and returns `True` or `False` to indicate it it's a leap year.
- `getDaysInMonth(month, year)`
  Takes numeric parameters for the month and year, and returns the number of days in that month.
- `isValidDate(day, month, year)`
  Takes numeric parameters for the day, month, and year, and returns `True` or `False` to indicate it it's a valid date.
- `getMonthName(month)`
  Takes a month number as a parameter, and returns the name of the month.
- `getDaySuffix(day)`
  Takes a day number as a parameter, and returns the suffix for that day number. For example, if the day number is `1`, the method returns "`st`". If the day number is `2`, the method returns "`nd`", and so on.
- `displayAllDatesInMonth(month, year)`
  Takes numeric parameters for the month and year, and displays all the dates in that month and year. Display the dates in verbose format for now (e.g. 1st February 2014).

Run the program and verify that it all works correctly.

Exercise 2: Defining and using default arguments

Enhance the `displayAllDatesInMonth()` function so that it takes an optional argument named `verbose`, and set the default value to `False`. Refactor the function so that it tests this flag, and outputs dates in verbose format or non-verbose format accordingly:

- Verbose format: 1st February 2016
- Non-verbose format: 01/02/2016

Invoke `displayAllDatesInMonth()` function from the client code, with or without a value for the `verbose` argument. Verify that the program outputs dates in the appropriate format.

Exercise 3: Additional suggestions (if time permits)

Write a function named `displaySpecialDatesInMonth()`. This function should be similar to `displayAllDatesInMonth()`, except that it takes a variadic list of day numbers. The idea is that these days are special days in the month (birthdays, anniversaries, important football games, etc.). The function should also allow the user to display the dates in verbose or non-verbose mode.

For example, client code should be able to call the function as follows, to display important days in December for the current year (in verbose mode):

```
displaySpecialDatesInMonth(12, year,
                           3, 25, 26, 31,
                           verbose=True)
```

In this case, the function should display the following output:

```
3rd December 1964
25th December 1964
26th December 1964
31st December 1964
```

Likewise, if the client code should be able to omit the `verbose` argument, as follows:

```
displaySpecialDatesInMonth(12, year,
                           3, 25, 26, 31)
```

In this case, the function should display the following output:

```
03/12/1964
25/12/1964
26/12/1964
31/12/1964
```