Python Programming
Asking students to pay the Pistus

Goals:

The goal of this lab assignment is to practice the different notions, previously seen, in order to tackle a real problem.

If you have any question about the syntax or how to write a part of your code, please use the "memo", the Python documentation or any other resource.

For this course and its lab assignments, you will use the PyCharm Edu Integrated Development Environment (IDE). Thus, your first step here is to create a new project named `SIP_LAB3` in PyCharm Edu.

1 Context : automatically sending reminder e-mails

“Pistus” (short for "Piston Ski") is an event organised by WACS (Winter Association CentraleSupélec) during the winter vacations. It is an awesome week-long mountain trip involving many activities (ski, snowboard, raclette, parties, etc.. . . .). After the traditional “Shotgun” event (booking your trip using the “first arrived, first served” rule on the WACS web site), students will be asked to pay for the trip before a given deadline.
The thing is, students in CentraleSupélec tend to forget to pay. You are asked to create a program that will send the users a remainder of the amount they have to pay and the deadline. The mail must be personalized (greeting the person by his/her name).
To do this, WACS gave you a CSV (Comma Separated Values) file with the list of the people who have booked their ticket for the Pistus. It contains the following fields: `name, surname, email, amount, paid, registration_date`.

The goal is to send an email to each person who has not yet paid, using the data stored in the csv file. We will use the `email` module to create the email as well as the `smtplib` module to send it.

Requirements :

- Ensure that the values in the csv file are in the right format.
- Ensure that the file indicates a valid email address for each student.
  - If an email address is missing, generate one as `name.surname@student.centralesupelec.fr`.
- Send an email to people who haven't paid yet:
  - Find people who haven't paid.
  - Create a personalized email (check the file `template.txt`)
  - Send the email.

2 Solving the problem step by step

2.1 Introduction questions

- Today you will learn how to automatically send emails with a Python script. Knowing how to do so, you might feel tempted to use this in order to spam your friends.
- Did you know that sending emails without the explicit authorization of the receiver is illegal in France? Article L.121-20-5 du code de la consommation:
« Est interdite la prospection directe au moyen d'un automate d'appel, d'un télécopieur ou d'un courrier électronique utilisant, sous quelque forme que ce soit, les coordonnées d'une personne physique qui n'a pas exprimé son consentement préalable à recevoir des prospections directes par ce moyen. »
- Find, with your favorite search engine, the risks of massive sending of unwanted commercial emails in France.
- Note that, apart from the judiciary risks, if you use your e-mail address to send spam messages, the mail servers might "blacklist" you, which means your e-mails, spam or not, will be systematically discarded, which is very inconvenient.

L'article 226-18-1 du code pénal, introduit par une loi du 6 août 2004, punit de cinq ans d'emprisonnement et de 300 000 euros d'amende un « traitement de données à caractère personnel concernant une personne physique malgré l'opposition de cette personne, lorsque ce traitement répond à des fins de prospection, notamment commerciale »

To know : Dictionaries

In the remainder of this assignment, you'll use an important Python data structure that is called *dictionary*. A dictionary is an unordered collection of items, each item being a key-value pair. The main advantage of a dictionary is that they are very efficient to find a value given its key. In the following example, a dictionary is used to implement a phone directory, where a key corresponds to the name of a person and a value is a phone number.

```
>>> tel = {'jack': 4098, 'sape': 4139}
>>> tel['guido'] = 4127
>>> tel
{'sape': 4139, 'guido': 4127, 'jack': 4098}
>>> tel['jack']
4098
```

2.2 Reading the data from the CSV file.

The first step of the implementation of our application is to load the data from a CSV file. In this section, you'll learn how to open a CSV file and read its content by using a Python program.

Question 1 CSV files

What is a CSV file? What is it used for?

Question 2 Database

This data has been collected in a spreadsheet and then exported to a CSV file. We could also have setup a database for this purpose. Why didn't we do so?

Question 3 Opening and reading the CSV file

Create a function named `open_csv_file()` that takes in the name of the file, opens the file, reads its content and prints it.
HINT. To find out how to open and read a CSV file, you can look at the documentation.

Question 4 Capturing an exception

What happens if you call the function `open_csv_file()` on a non-existent file? Modify the code of the function by using an exception handler (construct `try`....`except`) in order to print an error message if the file cannot be read.
HINT: The function `open()` raises an `OSError` if the file cannot be read. You can find this information in the official Python documentation.

To know :

It is possible to print the type of an error `ex` that is raised by a function by using the instruction `type(ex).__name__`.

Sometimes, it is useful to let the caller of the function handle the exceptions. For example, in some cases the caller might want to terminate the program when the exception is raised, while in some other cases it might just want to print a warning.

Question 5 Raising an exception

Modify the function `open_csv_file()` so as it captures the exception `OSError` and, in response, raises a new exception with the message "The file cannot be read". Then call the function and terminates the program in case the file cannot be read.
HINT. The documentation explains how to raise an exception with the keyword `raise`. In order to terminate the program, you can call the function `exit()` in the package `sys`.

3 Loading data into memory

Since we intend to process the data contained in the CSV file, we need to load it from the file and organize it in an in-memory data structure.

Question 6 Choice of the data structure

Given that we need to load a list of students from the file and quickly get the information (e.g., name, email address) about any given student, which data structure would you use to organize the data? Specify the data types that you need to represent the information about a student.

To know :

In order to convert a string to a float number, Python provides the function `float()`. This function raises an error when the string does not contain a valid number.
One might think that the function `bool()` converts a string to a Boolean. However, this is not the case, as the following example shows:

```
>>> bool("1")
    True
>>> bool("yes")
    True
>>> bool("true")
    True
>>> bool("false")
    True
>>> bool("")
    False
```

The function `bool()` doesn't look at the content of the string. It only returns `False` when the string is empty and it returns `True` otherwise.
In order to convert a string to a Boolean, you have to explicitly indicate in your program the values that correspond to `True`.

Question 7 Load data (clean data)

Write a function `load_data()` that takes in the name of the CSV file, reads it and loads its content to the data structure of your choice and returns the data structure. If the file does not exist, the function must raise an exception. Call the function on the file `pistus1.csv`.

HINT. To convert a string to a date, use the function `datetime.strptime()` from the package `datetime`. As for the format, specify `"%d/%m/%Y"` (equivalent to DD/MM/YYYY).

Question 8 Load data (noisy data)

Run the function `load_data()` on the file `pistus2.csv`. What do you get as a result? Why?

3.1 Cleaning the values of `amount`

It's important to remind that all data stored in a CSV file is considered as a string. Therefore, if you want to load a price, you'll need to convert it into a number, more precisely a float. However, if the string doesn't contain an actual number (for instance, the string 50€), then the Python interpreter will raise an error.

Question 9 Removing € and $

One first step to clean some of the values in the field `amount` is to remove the characters € and $. Write a Python function named `remove_special()` that takes in a string and returns the same string without the characters €and $.

The function `remove_special()` is not enough to make sure that a string contains a numeric value. For instance, if the value of the amount is "50 euros", the conversion from string to float will still return an error.

Question 10 Remove all non-numeric characters

Write a function named `remove_non_numeric()` that takes in a string and returns the same string without all the non-numeric characters, with the exception of `.` and `,`

Question 11 Point and Comma

Run the following instruction `print(float(remove_non_numeric('5,50 eur')) + 10.)`. What's wrong with it? Change the implementation of the function `remove_non_numeric()` to fix the issue.

Question 12 Float conversion

Write a function `convert_amount_to_float()` that takes in a string and returns the float representation of that string, after filtering out non-numeric values.

To know : Unit Testing

This part aims to give you the tools in order to do unit testing ; but we'll first see what it is and why it is important. Unit testing is one of the basic aspects of testing.

But what is unit testing anyway ?

Unit tests are **tests** - small pieces of Python code in our case - that will assert that each **unit** - functions of in our example - of our code is doing what it is supposed to be doing, and keep on doing what it is supposed to even as we make new improvements to our code.

Why it is important ?

Imagine you're working on a project that has 3 functions in it at the beginning : they are working fine, and some other developers are using your functions. Everything is beautiful under the sun, but then you decide to write a fourth function, which is using function 2 : but your new function is not working that well, and you decide to change function 2 because it's not exactly what you need now.

Your fourth function works fine now, and everything is perfect again under the sun ... or is it ? Because some of the other developers that were using your second function are now protesting that its new version is the source of all their problems !

Unit tests can help with this kind of situation : for each functionality of your program, there will be a test that will assert that it still works as supposed after any modification.

NB : is unit testing still relevant when working on a project alone ? Yes, because the other developers mentioned earlier could be you in two months for example !

When to do unit testing ?

Is it a good idea to test the code when everything is developed ? No, because the more complex your program is, the more difficult testing will be : it's best to do it as you are writing your code.

How to do unit testing in Python

There are multiple ways of doing unit testing in Python, but one of the easiest way is by using the framework `pytest`

Writing tests To write simple tests, here is a simple protocol:

1. For each file of your project named `filename.py` create a file named `filename_test.py` or `test_filename.py`
2. Start the file by the following lines:

```python
# import pytest to actually be able to use it !!
import pytest
# import the file that you want to test all the functions of the file
import filename as myFile       #(or any name you want to give it)
```

3. Then for each function in the file named *function* write the following lines:

```python
    # In each test function think about what case is useful
    def test_function():
        #Case 1 : description of case
        assert(function(arg1Case1,....argNCase1) == expectedResult)
        #Case 2 : ...
        ...

```

In PyCharm, `pytest` is integrated and can be run as seen in the official tutorial `https://www.jetbrains.com/help/pycharm/pytest.html`. Otherwise you need to start pytest from the command line `https://docs.pytest.org/en/latest/`

Question 13 Testing the function `convert_amount_to_float()`

Write a test function for `convert_amount_to_float()`

3.2 Cleaning the values of `paid`

You might have noticed that in file `pistus1.csv` the values of the column `paid` are either `true` or `false`. However, this is not the case in `pistus2.csv`, where the fact that a student has paid for the pistus is specified either as `1`, `yes` or `true` and the fact that a student hasn't paid is indicated as `0`, `false` or `False`.

Question 14 Converting a string to a Boolean

Write a Python function `convert_to_bool()` that takes in a string and returns a Boolean. Suppose that "1", "true" and "yes" (and all their case variations) are the only values that correspond to the value `True`.

Question 15 Test for `convert_to_bool()`

Write a test for `convert_to_bool()`

Question 16 Load data (noisy data)

Modify the function `load_data()` by using the functions defined above in order to read the file `pistus2.csv` without errors.

3.3 Duplicates

In real applications, it is not uncommon that a CSV file, or even a database, contains duplicates. Whenever possible, duplicates should be detected and removed. It is not always easy to define what a duplicate is. Suppose that in our application a student can register for the Pistus by only using his/her CS email address. Therefore, if we have two or more entries in the CSV file associated to the same email address, we have to keep only one.

Question 17 Duplicate removal

Write a function `remove_duplicates()` that takes in the data structure that contains the students registered to the Pistus and returns the data structure without duplicates. If two ore more entries are associated with the same email address, keep the one with the earliest registration date. If two entries have the same registration date, keep the one that occurs first in the file.

Test the function on the CSV file `pistus-duplicates.csv`. The function should remove the students Elvira Fake, Gilbert Fake1 and Gilbert Fake2.

3.4 Limited places

Suppose that the participation to Pistus is capped at a certain number of participants. By mistake, the organizers of Pistus allowed the registration of more participants than allowed.

Question 18 Remove participants

Write a function `remove_participants()` that removes as many participants as needed in order to respect the limit on the number of participants. The function takes in the data structure containing the students registered to Pistus and the maximum number of participants allowed in the event. The function then will:

- create two lists, one containing the students who have paid, and the other containing those who have not.
- sort the two lists by registration date in descending order (i.e., the last student who registered to the event must be the first in the list).
- remove the students starting from those who have not paid and registered later.

3.5 Writing to file

In the previous exercises, we did some data cleaning directly in the in-memory data structure. We now want to write to a CSV file the cleaned data.

Question 19 Writing to file

Write a code to write to a CSV file named `data_cleaned.csv` the data contained in `pistus2.csv` after cleaning the values of `paid` and `amount` (calling the function `load_data()` defined above) and limiting the number of participants to 10.

HINT. Look at the documentation to find out how to write a CSV file.

4 Sending the e-mail

A Python function using smtplib is given to send a mail. The goal of this exercise is to correctly format the message. You may need to change the configuration and/or the permission of your mail server. Furthermore, email sent by this method will sometime be considered as spam and rejected by the recipient server.

```python
import smtplib

# Mail server Identification
# You could use other tools to store them somewhere else
# Change this with your own credentials (GMail address & password)
MY_USERNAME = 'bob.lecanard@gmail.com'
MY_PASSWORD = 'CorrectHorseBatteryStaple'

def send_mail(sender, recipient, message):
    try:
        # You can try another server (office365 for example)
        server = smtplib.SMTP("smtp.gmail.com", 587)
        server.ehlo()
        server.starttls()
        server.login(MY_USERNAME, MY_PASSWORD)
        server.sendmail(sender, recipient, message)
        server.close()
        print 'successfully sent the mail'
    except SMTPException:
        print "Error: unable to send email"
```

Question 20 Deducing the email

1.The email may be missing from the csv. Make a function that create a student email address from the person's name

1bis. Upgrade your function to take into account the accents and compound noun of the students in the third CSV file. You may find the unidecode package useful.

2. Create a testing function that validates a student email address

Question 21 Customizing the message

Using a person's information, create a personalized message.

Question 22 Final loop

Hook all the previous functions in a loop to send an email to each person that didn't pay the Pistus.