Handling data — Part 1

The objective of this sequence of works is to be able to handle data stored in different formats.

As a first example, we want to handle a file containing contacts (names and phone numbers), see Figure 1 and http://www.barsamian.am/2020-2021/S6ICT/TP10_Contacts.txt[1].

Alice	0606060608	Frédérique	0606060613
0606060606	Djamel	0606060611	Isabelle
Bob	0606060609	Guillaume	0606060614
0606060607	Étienne	0606060612	Jérôme
Charles	0606060610	Hector	0606060615

Figure 1: File containing our contacts' phone numbers.

A skeleton of `python` code to manipulate these contacts, given in Listing 1, can be downloaded from http://www.barsamian.am/2020-2021/S6ICT/TP10_Contacts.py.

```
1  nbContacts = 10
2  names = [""]*nbContacts
3  phones = [0]*nbContacts
4
5  #"utf-8" is the standard encoding on Linux. Depending on the file,
6  # the encoding can also be "iso-8859-1" or "cp1252" (on Windows).
7  f = open("TP10_Contacts.txt", "r", encoding="utf-8")
8  # strip() removes blank characters at the beginning and the end of the string,
9  # here in particular the end of line characters left by readline()
10 for i in range(nbContacts):
11     names[i] = f.readline().strip()
12     phones[i] = f.readline().strip()
13 f.close()
14
15 s = input("What contact do you want to search for ? ")
16 i = 0
17 while (i < nbContacts) and not (s == names[i]):
18     i = i + 1
19 if i < nbContacts:
20     print(phones[i])
21 else:
22     print(s + " is not in your contacts.")
```

Listing 1: Skeleton code to handle our contacts file.

1. When prompted, you type "Alice". What does the program print?
2. When prompted, you type "charles". What does the program print? Modify the program so that it prints what you would expect in this situation[2].
3. When prompted, you type "etienne". What does the program print? This time, it is not easy to modify the program to have it return what you want. Use the function given in Listing 2 to perform the task — and don't forget the remark from the previous question.

```
1 import unicodedata
2 # This removes diacritics: accents, diaereses, umlauts, tildes, cedillas...
3 def normalize(text):
4     nfkd_form = unicodedata.normalize('NFKD', text)
5     return "".join([c for c in nfkd_form if not unicodedata.combining(c)])
```

Listing 2: Skeleton code to normalize strings.

[1] You can also use http://www.barsamian.am/2020-2021/S6ICT/TP10_Contacts_utf8.txt which is encoded in utf-8.

[2] We have seen how to do it in exercise 3 of Work n°5 : http://www.barsamian.am/2020-2021/S6ICT/TP5_Loops.pdf.

4. Add another person in `TP10_Contacts.txt`, then rerun the program. What happens? How can you fix the error? Can you fix the error so that no matter the number of contacts, the program will run without errors?

5. You now know another person also named Bob. Add this person in `TP10_Contacts.txt`, then rerun the program and ask for Bob's phone number. What happens? What would you expect? Can you fix this problem?

This first way of looking at the data is not the best. In fact, in this situation, we would use a data structure called a dictionary. A dictionary in `python` is a list of pairs of the form (key ; value). For example if you want to store the name of a pupil and the IP of its computer, you can build a dictonnary where the names will be the keys and the IP addresses the values like that:

```
names_ip = dict(john="192.168.1.10", jane="192.168.1.11")
```

The output is `{'jane': '192.168.1.11', 'john': '192.168.1.10'}`. Keys are on the left and values on the right. Here are other ways to build the same dictionary :

```
names_ip2 = {"john":"192.168.1.10", "jane":"192.168.1.11"}
names_ip3 = dict([("john","192.168.1.10"), ("jane","192.168.1.11")])
```

Let's see the more common methods that can be used on dictionaries :

```
len(names_ip)          #length, here returns 2
names_ip["john"]       #value corresponding to the key "john", here "192.168.1.10"
                       #(a non-existing key will give an error)
del names_ip["john"]   #removes the corresponding pair
"john" in names_ip     #True iff the key exists
names_ip.clear()       #removes all items
names_ip["john"] = "192.168.1.12" #updates the value associated to the key "john"
names_ip["tim"] = "192.168.1.22"  #add a key "tim" associated with the value "192.168.1.22"
```

How to loop over dictionaries: first if you want to loop over the pairs at once, use `items()`:

```
for name,ip in names_ip.items():
    print(name, " has ip ", ip)
```

Here the loop variables `name` and `ip` will be each key-value of the dictionary. If you want only the keys, use `name_ip.keys()`, for the values use `name_ip.values()`.

```
for n in names_ip.keys():
    print("name = ", n)
for ip in names_ip.values():
    print("IP address = ", ip)
```

If you want to make a list of keys or values you can type:

```
list_of_names = list(names_ip.keys())
list_of_ips = list(names_ip.values())
```

6. Modify the implementation of Listing 1 to use a dictionary `contacts` (keys are names, values are phone numbers), instead of using two arrays.

7. A dictionary cannot hold two different pairs that use the same key (here, the same name). How would you solve the "two Bobs" problem (question 5) with this data structure?