Lab 6

Chains of characters

Loops in chains of characters

Laboratory Objectives

Exercises using:

- Variables of type chains of characters
- Loops in chains of characters

Chains of characters

```
>>> s = 'bonjour'
>>> type(s)
<class 'str'>
```

- The sequence **\n** in a chain of characters causes to jump to the next line.
- the sequence **\'** enable us to insert an apostrophe in a chain of characters bounded by apostrophes.
- Similarly, the sequence **\"** allows the insértion of qutoation marks in a chain itself delimited by quotation marks.
- Note again that the case is significant in variable names (need to be respected scrupulously.

Concatenation, repetition, in

- Chains can be *concatenated with the operator* **+** and repeated by the operator *****

```
>>> n = 'abc' + 'def' # concatenation
>>> m = 'zut ! ' * 4 # repetition
>>> print(n, m)
abcdef zut ! zut ! zut ! zut !
```

- The instruction **in** can be used indépendantly from **for**, to check if a given ele*ment is part or not of a séquence*.

```
>>> 'a' in 'abba‘
True
```

Triple quotation

- To insert more easily special characters in a chain, without making use of the *antislash*, or to accept it in a chain, we can delimit it the chain with *triple guillemets* or *triples apostrophes* :

```
>>> s = """aa
   bb
   cc
   """
>>> s
'aa\n   bb\n   cc\n   '
```

Exercise 1

Using the Python interpretor, affect the value of type chain of characters ‘good' to a variable s1, ‘bad' a variable s2 and ‘crazy’ to a variable s3.

Derive Python expressions with variable s1, s2, and s3 for:

a) ‘azy' ist conteained in s3
b) a espace is not contained in s1
c) the concaténation of s1, s2, and s3
d) The space is contained in the concaténation of s1, s2, and s3
e) the concaténation of 10 copies of s3
f) The total number of characters in the concatenation of s1, s2, and s3

Indexing, extraction, length

- Chains are *sequences* of characters. Each of them occupy a precise place in the sequence. Elements of a sequence are indexed (or numbered) *starting from zero*.
- If the index is negativeit is referenced with respect to the end of the chain. -1 points to the last character, -2 lthe one before, etc.

```
>>> name = 'Cedric'
>>> print(name[1], name[3], name[5])
e r c
>>> print (name[-1], name[-2], name[-4], name[-6])
Cid
>>> print(len(name))
6
```

Extraction of chain fragments

- *Slicing* indicates between hooks indexes corresponding to the start and end of the slice that we want to extract:

```
>>> ch = "Juliette"
>>> print(ch[0:3])
Jul
>>> print(ch[:3]) # the first 3 characters
Jul
>>> print(ch[3:]) # whatever follow the first 3
                   # characters
iette
```

Exercise 2

1. Using the Python interpretor, create a variable named aha and affect to it the value 'abcdefgh' .

2. Derive Python expressions (in the interpretor) by using the variable aha that will be evaluated with:

```
a) 'abcd'
b) 'def'
c) 'h'
d) 'fg'
e) 'defgh'
f) 'fgh'
g) 'adg'
h) 'bd'
```

Character chains (str) methods

On ne peut pas modifier les chaines de caractères directement.

Usage	Explication
`s.capitalize()`	returns a copy of `s` that starts with an upper case
`s.count(target)`	returns the number of times the value of `target` is in `s`
`s.find(target)`	returns the first occurrence of `target` in `s`
`s.lower()`	returns a copy of `s` in upper case
`s.replace(old, new)`	returns a copy of `s` with`old` replaced by `new` `(all occurrences)`
`s.split(sep)`	returns a list of sub-chains (fragments) of `s`, delimited by `sep`
`s.strip()`	returns a copy of `s` without spaces at the start nor at the end
`s.upper()`	returns a copy of `s` `in upper case`

Exersice 3

Copy this expression in the Python interpretor:

s = ''' En 1815, M. Charles-François-Bienvenu Myriel était évêque de
Digne. C’était un vieillard d’environ soixante-quinze ans ; il occupait le
siège de Digne depuis 1806. …   '''

(The begining of the novel Les misérables by Victor Hugo.)

Do the following exercises in the interpretor:

(a) Create a copy of s, named nS, with characters `.` `,` `;` and \n replaced by spaces.

(b) Erase the spaces that are at the start and end of nS (and affect the new chaine in the same variable nS).

(c) Change all the caracters of nS in lower case (and name the new chain nS).

(d) Calculate the number of times nS contains 'de'.

(e) Change all the sub-chains était to est (and name the new chain nS).

Exercise 4

- Derive a Python function named `count` that will calculat the number of occurrences of character c in a chain s. try 2 versions: with the method `count`of the `str` class and without that it (use a loop while or for).

- Develop the main part of the program that get from the user a character chain named s, and call the function twice to calculate the number of ‘a’ and of 'de la'. The last part should be:

```
print(count(s,'a'))
```

Exercise 5

- Derive a Python function `spaces` that takes a character chain s and returns another chain with spaces inserted between the neighboring letters. Do not use `print` in the function. The returned chain should not have any space at the end.
- Test. the function with a main program, or in the interpretor. For instane:

```
>>> spaces(‘important')
'i m p o r t a n t'
```

Exercise 6

- Derive a Python function named code that take a character chain s and returns another coded chain. The code is calculated by taking each pair of consecutive letters and changing the order in the pair (spaces, ponctuation, etc. are traited like letters).
- Test your function with a main program or in the interpretor. For example:

```
>>> code('message secret')
'emssga eesrcte'
>>> code('Message')
‘eMssgae'
```