Lab 5

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

- The sequence **\n** in a chain of characters causes a 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 insertion of quotation 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 = 'bye ! ' * 4  # repetition
>>> print(n, m)
abcdef bye ! bye ! bye ! bye !
```

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

```
>>> 'a' in 'abba'
True
```

Triple double quotes

- 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 the chain with *triple double quotes*:

```
>>> s = """aa
    bb
    cc
    """
>>> s # To display it:
'aa\n   bb\n   cc\n   '
```

Exercise 1

Using the Python interpretor, let us assign the value of type chain of characters ‘good‘, ‘bad’ and ‘crazy’ respectively to variables str1, str2 and str3.

Derive the Python expressions with variables str1, str2, and str3 for:

a) ‘azy' is contained in str3

b) a space is not contained in str1

c) The concatenation of str1, str2, and str3

d) The space is contained in the concatenation of str1, str2, and str3

e) The concatenation of 10 copies of str3

f) The total number of characters in the concatenation of str1, str2 and str3

Indexing, extraction, length

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

```
>>> name = 'Cedric'
>>> print(name[1], name[3], name[5])
e r c
>>> print(name[-1], name[-2], name[-4])
C I d
>>> 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]) # the first 3 characters
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), 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

We cannot modify directly character chains.

Use	Explaination
`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:

str = ''' In 1815, M. Charles-François-Bienvenu Myriel was a bishop in Digne. He was a seventy five years old man; he held that position in Digne since 1806. … '''

(From by Victor Hugo's novel «Les miserables».)

Do the following exercises in the interpretor:

(a) Create a copy of str, named nStr, with characters . , ; and \n replaced by spaces.

(b) Remove the spaces that are at the start and end of nStr (and name the new chain nStr).

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

(d) Derive the number of times nStr contains 'in'.

(e) Change all the sub-chains was to is (and name the new chain nStr).

Exercise 4

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

- Develop the main part of the program that gets from the user a character chain named *str*, and call the function twice to get the number of ‘a’. The last part should be:

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

Exercise 5

- Derive a Python function *spaces* that takes a character chain str 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 instance:

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

Exercise 6

- Derive a Python function named code that takes a character chain str 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(‘secret message')
' esrctem seaseg '
>>> code('Message')
‘eMssgae'
```