Lab Python n°1

All roads lead to Rome. There is no unique solution to a problem.

Beginner Level :

1. Hello World

The « Hello World! » is the most known program to introduce a new programming language. We won't be breaking the tradition in this lab.

Write a python program which consists of getting python to print "Hello World" to the console.

Solution

```
>> print('Hello World! ')
```

2. Tic Tac Toe

Write a python program which consists of getting python to print a Tic Tac Toe Board, as shown below, to the console.

```
   |   |
---------
   |   |
---------
   |   |
```

Solution (Lets include namespaces and make it complicted)

```
>> line_1 = '  |  |  '
>> line_2 = '------- '
>> namespace = __import__(__name__)
>> print('\n'.join([getattr(namespace,'line_'+str(i%2+1)) for i in range(5)]))
```

Solution (or we can just simply do)

```
>> line_1 = ‘  |  |  \n‘
>> line_2 = ‘------- \n‘
>> print((line_1+line_2)*2+ line_1)  # En python ‘chaine’*2 = ‘chainechaine‘
```

3. Giant Tic Tac Toe

Using loops and conditions, write a python program which consists of getting python to print a giant Tic Tac Toe Board, as shown below, to the console.

```
  |  |  H  |  |  H  |  |
--+--+--H--+--+--H--+--+--
  |  |  H  |  |  H  |  |
--+--+--H--+--+--H--+--+--
  |  |  H  |  |  H  |  |
========+========+========
  |  |  H  |  |  H  |  |
--+--+--H--+--+--H--+--+--
  |  |  H  |  |  H  |  |
--+--+--H--+--+--H--+--+--
  |  |  H  |  |  H  |  |
========+========+========
  |  |  H  |  |  H  |  |
--+--+--H--+--+--H--+--+--
  |  |  H  |  |  H  |  |
--+--+--H--+--+--H--+--+--
  |  |  H  |  |  H  |  |
```

Solution (””” can be used to comment in python but also to define strings)

```
>> schema = “””
     |  |  H  |  |  H  |  |
   --+--+--H--+--+--H--+--+--
     |  |  H  |  |  H  |  |
   --+--+--H--+--+--H--+--+--
     |  |  H  |  |  H  |  |
   ========+========+========
     |  |  H  |  |  H  |  |
   --+--+--H--+--+--H--+--+--
     |  |  H  |  |  H  |  |
   --+--+--H--+--+--H--+--+--
     |  |  H  |  |  H  |  |
   ========+========+========
     |  |  H  |  |  H  |  |
   --+--+--H--+--+--H--+--+--
     |  |  H  |  |  H  |  |
   --+--+--H--+--+--H--+--+--
     |  |  H  |  |  H  |  |
   “””
>> print(schema)
```

Advanced Level :

1. Collatz Sequence

The *Collatz sequence* is an iterative sequence defined on the positive integers by:

```
n -> n / 2     if n is even
n -> 3n + 1    if n is odd
```

For example, using the rule above and starting with 13 yields the sequence:

```
13 -> 40 -> 20 -> 10 -> 5 -> 16 -> 8 -> 4 -> 2 -> 1
```

The sequence stops when 'n' reaches 1.

Choose a random number between 50 and 100 then write a python program that prints its relative *Collatz sequence.*

Solution

```
>> import random  # import a python library
>> n = random.randint(50,100)  # get a random integer between 50 and 100
>> result = str(n) + " -> " # We have to convert n to a string before (using str)
>> while n > 1:
>>    if n % 2:    #same as  if n% 2 != 0 (0 value is considered as False in python)
>>       n = 3*n+1
>>    else:
>>       n = n / 2
>>    result += str(n) + " -> "  :    # x += y  ⇔  x = x + y
>> print(result + str(1))
```

2. Pascal's Triangle

Write a python program that prints to the console the 10 first levels of the pascal's triangle.

As a reminder, each element in a row of Pascal's triangle is formed by summing the two elements in the previous row directly above (to the left and right) that elements. If there is only one element directly above, we only add that one. For example, the first 5 rows of Pascal's triangle look like:

```
        1
      1   1
    1   2   1
  1   3   3   1
1   4   6   4   1
```

Solution 1 (solutions using simple tricks are generally the most elegant ones, but be careful, tricks can be tricky)

```
   0  1  4   6   4  1           0  1  5  10  10  5  1
+  1  4  6   4   1  0        +  1  5  10  10  5  1  0
_______________             ________________
   1  5  10  10  5  1           1  6  15  20  15  6  1
```

```
>> number_of_rows = 10
>> pascal_rows = [[1]]  # a list of lists (each row is a list)
>> for i in range(1,number_of_rows):
>>       list_0_left = [0] + pascal_rows[i-1] # In python: [a] +[b] = [a,b]
>>       list_0_right = pascal_rows[i-1] + [0]
>>       new_pascal_row = []
>>       for j in range(len(list_0_left)):
>>             new_pascal_row.append(list_0_left[j] + list_0_right[i])
>>       pascal_rows.append(new_pascal_row)
>> # now you can print it as you want
```

Solution 2 (same trick, but using python shortcuts)

```
>> number_of_rows = 10
>> pascal_rows = [[1]]
>>
>> for i in range(1,number_of_rows):
>>       pascal_rows.append([
                ([0] + pascal_rows[i-1])[j]+ (pascal_rows[i-1] + [0])[j]
                for j in range(len(pascal_rows[i-1])+1)
         ])
 >> # now you can print it as you want
```

Solution 3 (same trick, but using even more python shortcuts and the zip built-in function)

```
>> number_of_rows = 10
>> pascal_rows = [[1]]
>> [pascal_rows.append([
                left + right
                for left,right in zip([0] + pascal_rows[i-1], pascal_rows[i-1]+[0])
    ]) for i in range(1,number_of_rows)]
>> # now you can print it as you want
```

3. Surpassing Words

Surpassing words are English words for which the gap between each adjacent pair of letters strictly increases. These gaps are computed without "wrapping around" from Z to A.

For example:

SUPERB

SU / UP / PE / ER / RB

2 / 5 / 11 / 13 / 16

Write a python program that decides whether “Juridic” is a surpassing word or not.

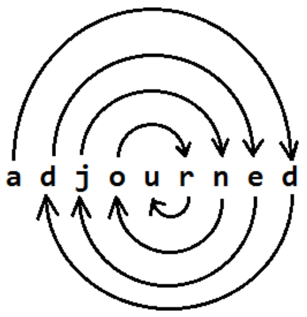
Solution

```
>> def is_surpassing_word(word) :
       word = word.upper()
       Distances = [abs(ord(left)-ord(right)) for left,right in zip(word,word[1:])]
       return all(left < right for left, right in zip(distances, distances[1:]))
>> # can you see what happened in those two lines?
```

Bonus Level :

Cyclone Words

Cyclone words are English words that have a sequence of characters in alphabetical order when following a cyclic pattern. For example:



Write a python script that determines whether a given word is a Cyclone word or not.

Solution (el classico – step by step)

```
>> def is_cyclonic_word(word):
        word = word.upper()
        previous_char_index = 0
        current_char_index = len(word)-1
        direction = 0
        while previous_char_index != (len(word)+1)//2:
              if ord(word[previous_char_index]) > word[current_char_index]:
                   return False  # No need to execute the rest of the function
              current_char_index = -previous_char_index % len(word) + direction
              if direction == 0
                   direction = -1
              else:
                   direction = 0
              # The previous if, else block can be replaced in a single line by:
              # direction = -1 if direction == 0 else 0
              iterations_left -= 1
        return True
```

Solution (In fewer lines)

```
>> def is_cyclonic_word(word):
        word = word.upper()
        letters = [None] * len(word)  # In python [object]*2 = [object,object]
        half = (len(word) + 1) // 2
        letters[::2] = word[:half]
        letters[1::2] = word[:half - 1:-1]
        return all([left <= right for left, right in zip(letters, letters[1:])])
>> # Just insert prints between each line to understand what happened there!
```

Useful Links

- Python 2.x built-ins functions (print, zip, all, getattr,…) : https://docs.python.org/2/library/functions.html
- Python 3.x built-ins functions (print, zip, all, getattr,…) : https://docs.python.org/3/library/functions.html
- Python 2.x string operations: https://docs.python.org/2/library/string.html
- Python 3.x string operations: https://docs.python.org/3/library/string.html
- Python 2.x Data Stuctures (list, dict,..) operation: https://docs.python.org/2/tutorial/datastructures.html
- Python 3.x Data Stuctures (list, dict,..) operation: https://docs.python.org/3/tutorial/datastructures.html