23 Reunião 23: 17/NOV/2020

Figure 1: Fonte: https://hellomariworld.com/aprendendo-a-programar-sozinha-dicas/

23.1 Reuniões passadas

strings versus listas

Figure 2: Xavier

Mutabilidade: listas são **mutáveis**; *podemos* alterar um componente:

```
lst[i] = x # :-)
```

Imutabilidade: strings são **imutáveis**; *não podemos* alterar um componente:

```
s[i] = 'x' # ERRO :-\
```

Atribuições: só criam ou atualizam *apelidos*, **não** fazem uma *cópia* ou *clone* de algum valor/coisa/objeto.

Caracteres ao sempre exibidos por `print()` exibem símbolos gráfico e **efeitos não gráficos**.

```
 0   1   2
+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+-
| l | i | n | h | a | 0 |\n | l | i | n | h | a | 1 |\n | l | i | n | h | a | 2 |\n | l | i |
+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+-
```

A seguir está uma tabela resumindo algumas das operações sobre strings. No que segue `s` e `t` são strings e `ini`,`fim`, `passo` e `n` são números inteiros.

Operação	Resultado
`x in s`	`True` se `x` é item de `s`, senão `False`
`x not in s`	`False` se `x` é item de `s`, senão `True`
`s + t`	a *concatenação* se `s` e `t`
`s * n` ou `n * s`	equivalente a `s` ser concatenada `n` vezes
`s[i]`	i-ésimo caractere de `s`
`s[ini:fim]`	fatia de `s` de `ini` a `fim`
`s[ini:fim:passo]`	fatia de `s` de `ini` a `fim` com passo `passo`
`len(s)`	comprimento de `s`
`s == t`	`True` se `s` e `t` são iguais, senão `False`
`s is t`	strings são **imutabilidade**, equivalente a `s == t`
`s[i]='x'`	ERRO, strings são **imutáveis**

A seguir está uma tabela resumindo algumas das operações sobre listas. No que segue `lst` é `lstB` são listas e `ini`,`fim`, `passo` e `n` números inteiros.

Operação	Resultado
`x in lst`	`True` se `x` é item de `lst`, senão `False`
`x not in lst`	`False` se `x` é item de `lst`, senão `True`
`lstA + lstB`	a *concatenação* se `lstA` e `lstB`
`lst * n` ou `n * lst`	equivalente a `lst` ser concatenada `n` vezes
`lst[i]`	i-ésimo caractere de `lst`
`lst[ini:fim]`	fatia de `lst` de `ini` a `fim`
`lst[ini:fim:passo]`	fatia de `lst` de `ini` a `fim` com passo `passo`
`len(lst)`	comprimento de `lst`
`lst == lstB`	`True` se `lst` e `lstB` são iguais, senão `False`
`lst is lstB`	`True` se `lst` e `lstB` são **apelidos** para uma mesma lista
`lst[i]=x`	lista são **mutáveis**

23.2 Exercício: leitora csv

Escreva um programa que lê um texto com campos/valores separados por vírgulas e exiba cada valor em uma linha.

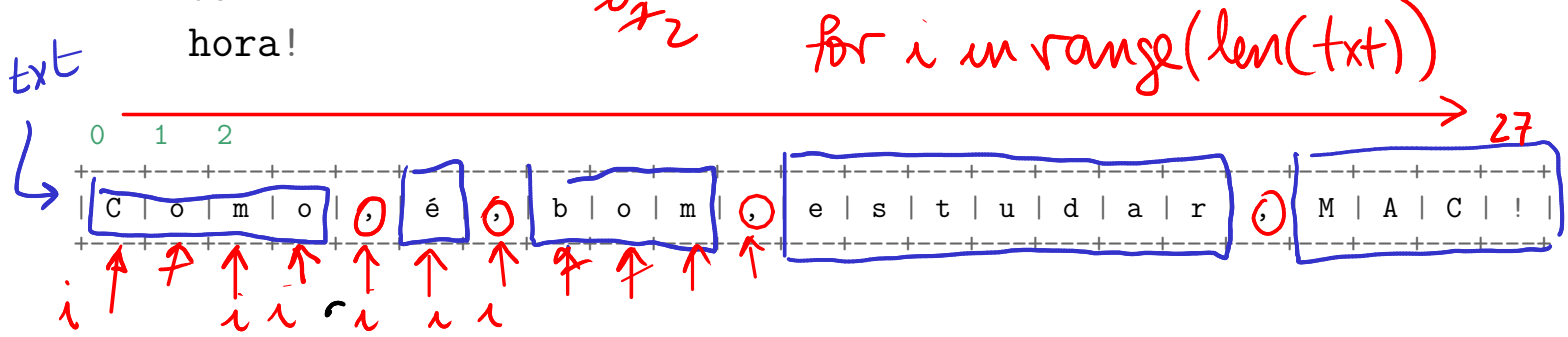
Versão *de luxe*: Escreva um programa que leia um arquivo no formato csv e exiba cada campo do arquivo em uma linha.

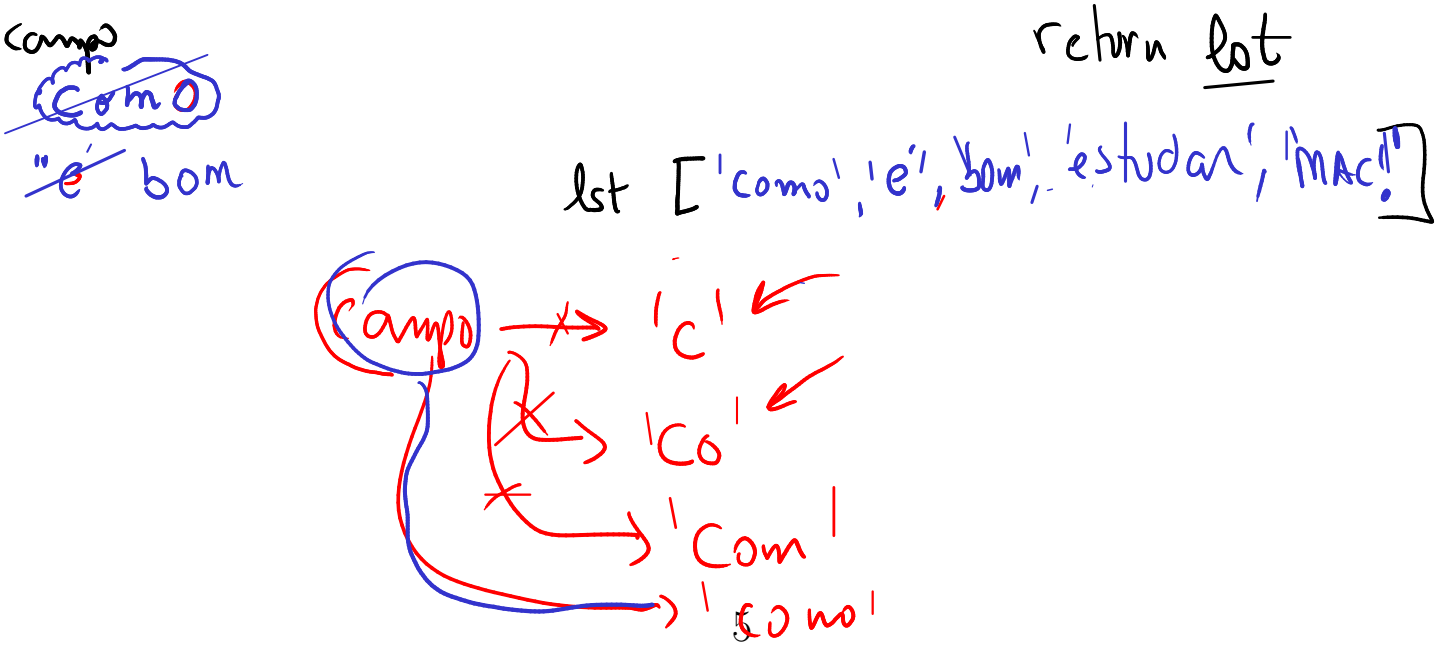
Em um arquivo no **formato csv** cada linha é um registro de dados. Cada registro consiste em um ou mais campos ou valores separados por uma vírgulas O uso da vírgula como separador de campos é razão do nome do formato *Comma-Separated Values*.

Um arquivo no formato csv normalmente armazena dados tabulados e sem formatação; nesse caso, cada linha terá o mesmo número de campos. Frequentemente arquivos no formato csv são usados por cientistas de dados e são bem grandes. Veja por exemplo o arquivo geographic-distribution-covid-19-cases-worldwide.csv que acabamos de copiar do *European Centre for Disease Prevention and Control*.

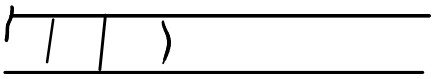
23.2.1 Exemplos

```
Digite um texto csv: Como,é,bom,estudar,MAC0110!
Como
 é
 bom
 estudar
 MAC0110!

Digite um texto csv: MAC0110,é,massa!
MAC0110
 é
 massa!

Digite um texto csv: MAC0110 , é , da , hora!
MAC0110
  é
 da
 hora!
```

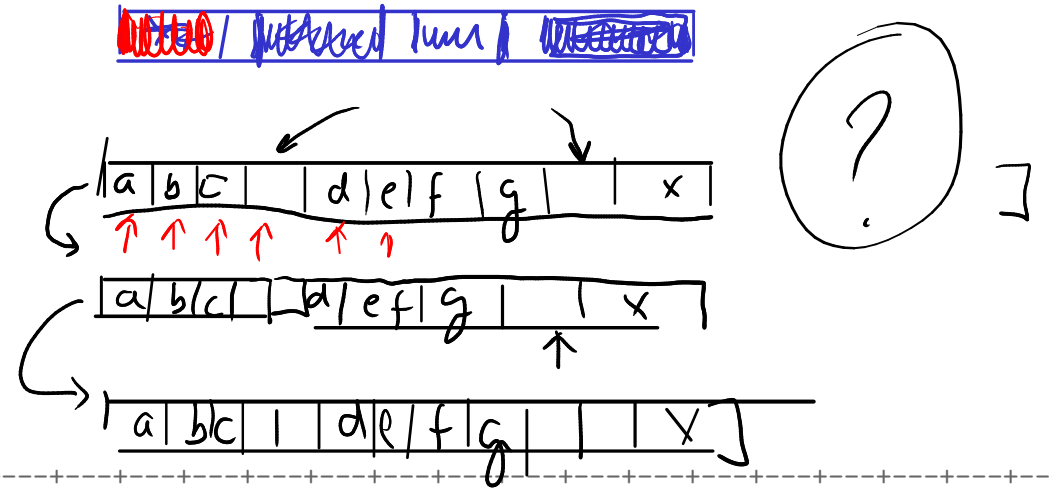




23.2.2 Rascunho



```
  0   1   2
+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+
| C | o | m | o | , | é | , | b | o | m | , | e | s | t | u | d | a | r | , | M | A | C | ! |
+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+
```




```
  0   1   2
+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+
| C | o | m | o | , | é | , | b | o | m | , | e | s | t | u | d | a | r | , | M | A | C | ! |
+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+
```

23.2.3 Solução

```
# vamos usar a funçao limpe() da aula passada que está em util.p
from util import limpe

BRANCO    = " \n\t\v\f"
SEPARADOR = ","

#-------------------------------------------------------------
def main():
    '''
    Programa que lê um texto no formato csv e exibe cada campo/u
    do arquivo em uma linha.
    '''
    # 1 leia a string
    txt = input("Digite um texto csv: ")
    print(f"texto= '{txt}'")

    # 2 pegue a lista de strings com os campos
    lista_str = separe(txt) # txt.split(",")

    # 3 mostre cada campo em uma linha
    for item in lista_str:
        print(limpe(item)) # item.strip()


#-------------------------------------------------------------
def separe(txt, sep = SEPARADOR):
    ''' (str, str) -> list

    RECEBE uma string `txt` e uma string `sep`.
    RETORNA uma lista contendo os ´campos´ de txt considerando
    os caracteres `sep` como delimitadores.
```

```
    A função "corta" o txt nos separadores.
    '''
    lst_campos = []
    n = len(txt)
    campo = ''
    for i in range(n):
        car = txt[i]
        if car == sep:
            lst_campos += [campo]
            campo = ''
        else:
            campo += car

    # último campo
    if campo != '':
       lst_campos += [campo]

    return lst_campos

#-----------------------------------------------
def pause():
    input("Tecle ENTER para continuar.")

#-----------------------------------------------
if __name__ == "__main__":
    main()
```

23.2.4 Solução *de luxe*

```
#!/usr/bin/env python3
# -*- coding: utf-8 -*-

# vamos usar a funçao limpe() da aula passada que está em util.p
from util import limpe

SEPARADOR = ",\n"

#-----------------------------------------------------------------
def main():
    '''
    Programa que lê um texto no formato csv e exibe cada campo/u
    do arquivo em uma linha.
    '''
    # 1 leia a string
    # txt = input("Digite um texto csv: ")
    # 1. leia o nome do arquivo
    nome_arq  = input("Digite o nome do arquivo csv: ")
    # 2. abra o arquivo
    arq = open(nome_arq, "r", encoding="utf-8")
    # 3. leia o arquivo
    txt = arq.read()
    # 4. feche o arquivo
    arq.close()

    # imprima o texto
    print("Texto:")
    print(txt)

    pause()
    # 2 pegue a lista de strings com os campos
```

```
lista_str = separe(txt) # texto.split(",")
print("lista_str=", lista_str)

pause()
# 3 mostre cada campo em uma linha
for item in lista_str:
    print(limpe(item)) # item.strip()
```

```
#--------------------------------------------------
def separe(txt, sep = SEPARADOR):
    ''' (str, str) -> list

    RECEBE uma string `txt` e uma string `sep`.
    RETORNA uma lista contendo os ´campos´ de txt considerando
    os caracteres `sep` como delimitadores.

    A função "corta" o txt nos separadores.
    '''
    lst_campos = []
    n = len(txt)
    campo = ''
    for i in range(n):
        car = txt[i]
        if car in sep:
            lst_campos += [campo]
            campo = ''
        else:
            campo += car

    # último campo
    if campo != '':
       lst_campos += [campo]

    return lst_campos

#--------------------------------------------------
def pause():
    input("Tecle ENTER para continuar.")

#--------------------------------------------------
```

```
if __name__ == "__main__":
    main()
```

23.2.5 Unicode

```
NAIPES  = ['\u2661', '\u2665', '\u2662', '\u2663']
>>> NAIPES  = ['\u2661', '\u2665', '\u2662', '\u2663']
>>> print(NAIPES[0])

>>> print(NAIPES)

>>> alef = "\u2135"
>>> print(alef)

>>>
```