22 Reunião 22: 12/NOV/2020

Figure 1: Fonte: https://www.alamy.com/

22.1 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 :-\
```

22.2 Strings

22.2.1 Caractere '\n'

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 |
+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+-
```

Tipos nativos: `int`, `float`, `bool`, `str`, `list`, `Nonetype`

`int`, `float`, `bool` e `NoneType` são tipos de dados **primitivos**, pois seus valores *não são* compostos de partes menores. Eles não podem ser "quebrados".

Strings (`str`) e listas (`list`) são diferentes pois são compostos de partes menores.

Um **caractere** é um símbolos gráficos como letras, pontuação, espaços que são exibidos na tela. Exemplos de caracteres diferentes de letras: `' '`, `'\n'`, `'*'`, `'-'`, ...

Os componentes de uma *lista* podem ser **qualquer** *coisa*.
Os componentes de *strings* são *somente* **caracteres**.

A manipulação de listas e strings é muito semelhante.
Há, no entanto, uma diferença fundamental: strings são **imutáveis**!.

Tipos formados por partes menores são chamados de **coleção de tipos de dados**.

Dependendo do que fizermos desejamos tratar uma coleção de tipos como uma única entidade ou desejamos acessar as suas partes.

Esta ambiguidade pode ser útil.

```
  0   1   2   3   4   5   6   7   8   9   10  11  12  13  14  15
+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+
| I | n | s | t | . |   | M | a | t | e | m | á | t | i | c | a |
+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+
 -16 -15 -14 -13 -12 -11 -10 -9  -8  -7  -6  -5  -4  -3  -2  -1
```

Strings são uma sequência de caracteres delimitadas por " ou '.

22.2.2 Algumas operações

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**

Para criar uma **string vazia** fazemos

```
In [9]:  string_vazia = ""

In [10]: string_vazia
Out[10]: ''

In [11]:  string_vazia = ''
```

```
In [12]: string_vazia
Out[12]: ''
```

Para **cria** uma nova string fazemos

```
In [13]:  s = "Como é bom estudar MAC0110!"

In [14]: s
Out[14]: 'Como é bom estudar MAC0110!'

In [15]: print(s)
Como é bom estudar MAC0110!
```

Strings ***são imutáveis***, não podemos alterar seus componentes

```
In [18]: s = "MAC0110"

In [19]: s[2] = 'T'    # ERRO
---------------------------------------------------------------
TypeError                                 Traceback (most rec
<ipython-input-19-32c1c57e2e7a> in <module>
----> 1 s[2] = 'T'

TypeError: 'str' object does not support item assignment
```

Não alteramos strings, **contruímos novas** strings.

```
In [23]: s = 'MAC0110'

In [24]: t = s[:2] + 'T' + s[3:]

In [25]: t
Out[25]: 'MAT0110'
```

22.2.3 Criar strings com *f-strings*

```
In [29]: s = f"n = {123}"

In [30]: s
Out[30]: 'n = 123'

In [31]: print(s)
n = 123

In [32]: t = f"pi = {3.14}"

In [33]: t
Out[33]: 'pi = 3.14'


In [37]: nome = 'Maria'

In [38]: w = f"nome = {nome}"

In [39]: w
Out[39]: 'nome = Maria'

In [40]: k = 123

In [41]: s = f"n = {k}"

In [42]: pi = 3.14

In [43]: t = f"pi = {pi}"

In [44]: s
Out[44]: 'n = 123'
```

```
In [45]: t
Out[45]: 'pi = 3.14'
```

22.2.4 Criar strings com '%?'

```
In [29]: s = "n = %d"%(123)

In [30]: s
Out[30]: 'n = 123'

In [31]: print(s)
n = 123

In [32]: t = "pi = %f"%(3.14)

In [33]: t
Out[33]: 'pi = 3.140000'

In [34]: t = "pi = %s"%(3.14)

In [35]: t
Out[35]: 'pi = 3.14'

In [36]: s = "n = %s"%(123)

In [37]: s
Out[37]: 'n = 123'

In [38]: w = "nome = %s"%("Maria")

In [39]: w
Out[39]: 'nome = Maria'
```

```
In [40]: w = "nome = %s"%"Maria" # com um elemento o () são s

In [41]: s = "n = %s"%123

In [42]: t = "pi = %s"%3.14

In [43]: w
Out[43]: 'nome = Maria'

In [44]: s
Out[44]: 'n = 123'

In [45]: t
Out[45]: 'pi = 3.14'
```

22.2.5 Conversão para `str`

Para **converter** um objeto para uma string utilizamos a função nativa `str()`.

```
In [46]: s = str(123)

In [47]: s
Out[47]: '123'

In [48]: s = str(True)

In [49]: s
Out[49]: 'True'

In [50]: s = str(None)

In [51]: s
Out[51]: 'None'
```

```
In [52]: print(s)
None

In [53]: s = str(3.1415926)

In [54]: s
Out[54]: '3.1415926'

In [55]: print(s)
3.1415926

In [56]: s = str(True)

In [57]: s
Out[57]: 'True'

In [58]: s = str([1,2,3])

In [59]: s
Out[59]: '[1, 2, 3]'
```

Há várias maneiras e **percorrermos** uma string.

```
In [58]: s = "MAC0110!"
In [58]: i = 0
In [59]: while i < len(s):
    ...:     print(s[i])
    ...:     i + = 1
    ...:
M
A
C
0
```

```
1
1
0
!

In [60]: s = "MAC0110!"
In [61]: for i in range(len(s)):
    ...:     print(s[i])
    ...:
M
A
C
0
1
1
0
!

In [62]: s = "MAC0110!"
In [63]: for car in s:
    ...:     print(car)
    ...:
M
A
C
0
1
1
0
!
```

22.3 Listas (`list`)

Listas são uma sequência de objetos delimitados por `[` e `]` e separados por vírgulas.

Para criar uma lista vazia fazemos

```
In [9]:  lst_vazia = []

In [10]: lst_vazia
Out[10]: []
```

22.3.1 Criar listas

Para cria uma nova lista fazemos

```
In [13]: lst = ["M", 'A', True, None, 3.14, 123]

In [14]: lst
Out[14]: ['M', 'A', True, None, 3.14, 123]

In [15]: print(lst)
["M", 'A', True, None, 3.14, 123]
```

22.3.2 Mutabilidade

Listas ***são mutáveis***, *podemos* alterar seus componentes ou acrescentar alguns

```
In [18]: lst = ["M", 'A', True, None, 3.14, 123]

In [19]: lst[2] = 'T' # altera

In [20]: lst += ["x-(wo)men"] # acrescenta

In [21]: lst
Out[21]: ['M', 'A', 'T', None, 3.14, 123, 'x-(wo)men']
```

22.3.3 Algumas operações

A seguir está uma tabela resumindo algumas das operações sobre listas. No que segue `lst` é `lstA` 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 == lstA`	`True` se `lst` e `lstA` são iguais, senão `False`
`lst is lstA`	`True` se `lst` e `lstA` são **apelidos** para uma **mesma** lista
`lst[i]=x`	lista são **mutáveis**

Há várias maneiras de percorrermos uma lista.

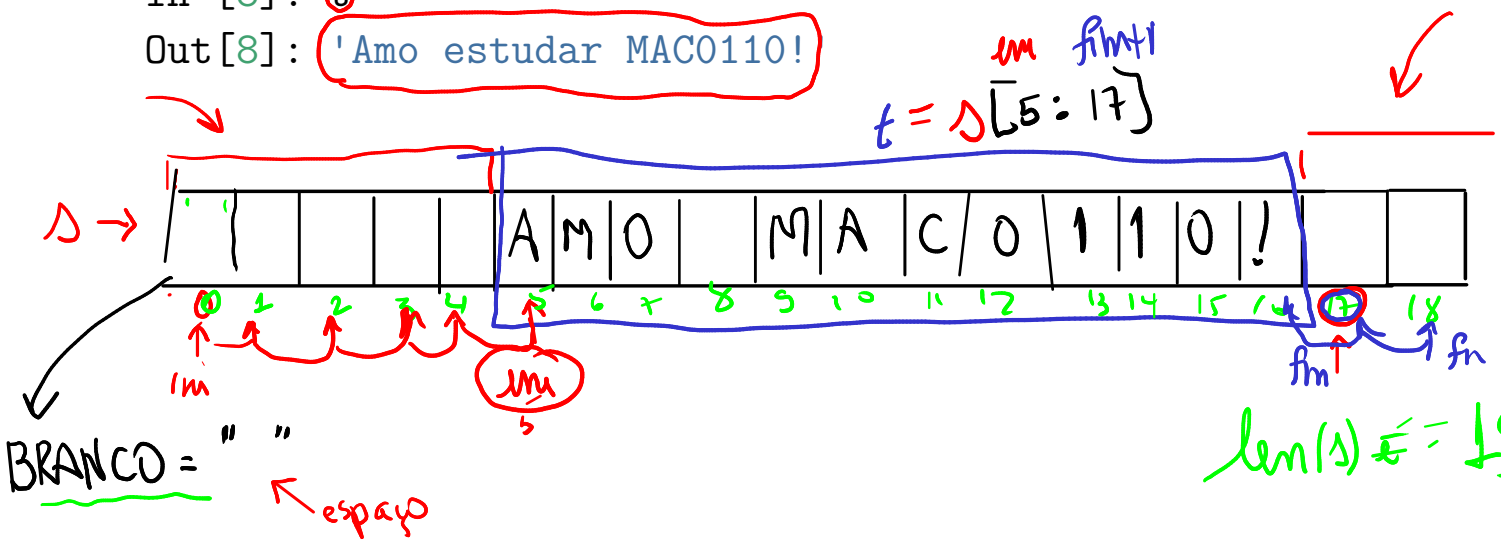
```
In [58]: lst = [1, 2, True]
In [58]: i = 0
In [59]: while i < len(lst):
    ...:     print(lst[i])
    ...:     i + = 1
    ...:
1
2
True

In [60]: lst = [1, 2, True]
In [61]: for i in range(len(lst)):
    ...:     print(lst[i])
    ...:
1
2
True

In [62]: s = "MAC0110!"
In [63]: for item in lst:
    ...:     print(item)
    ...:
1
2
True
```

22.4 Exercício- arranca espaços

Escreva uma função `limpe()` que recebe uma string e retorna a string correspondente sem espaços no início e no final.

Exemplo

```
In [2]: verdade = "    MAC0110!  é da hora!    "
In [3]: verdade
Out[3]: '    MAC0110!  é da hora!    '
In [4]: s = limpe(verdade)
In [5]: s
Out[5]: 'MAC0110!  é da hora!'
In [6]: declaracao = '→Amo•estudar•MAC0110!→'
In [7]: t = limpe(declaracao)
In [8]: t
Out[8]: 'Amo estudar MAC0110!
```



22.4.1 Solução

```
BRANCO = " \n\t\r"

def limpe(s):
    '''(str) -> str

    RECEBE uma string e Retorna a correspondente string após
    o espaços em ciobranco serem removidos do início e final
    da string.

    Exemplo:


    In [8]: s = "   Python é da hora!   "
    In [90]: t = limpe(s)
    In [10: t
    Out[10]: 'Python é da hora!'
    '''
    n = len(s)
    ini = 0
    fim = n - 1

    # encontra início
    ini = 0
    while ini < n   and s[ini] in BRANCO:
        ini += 1

    # encotra fim
    fim = n-1
    while fim > ini and s[fim] in BRANCO:
        fim -= 1
```

```
    return s[ini: fim+1]
```

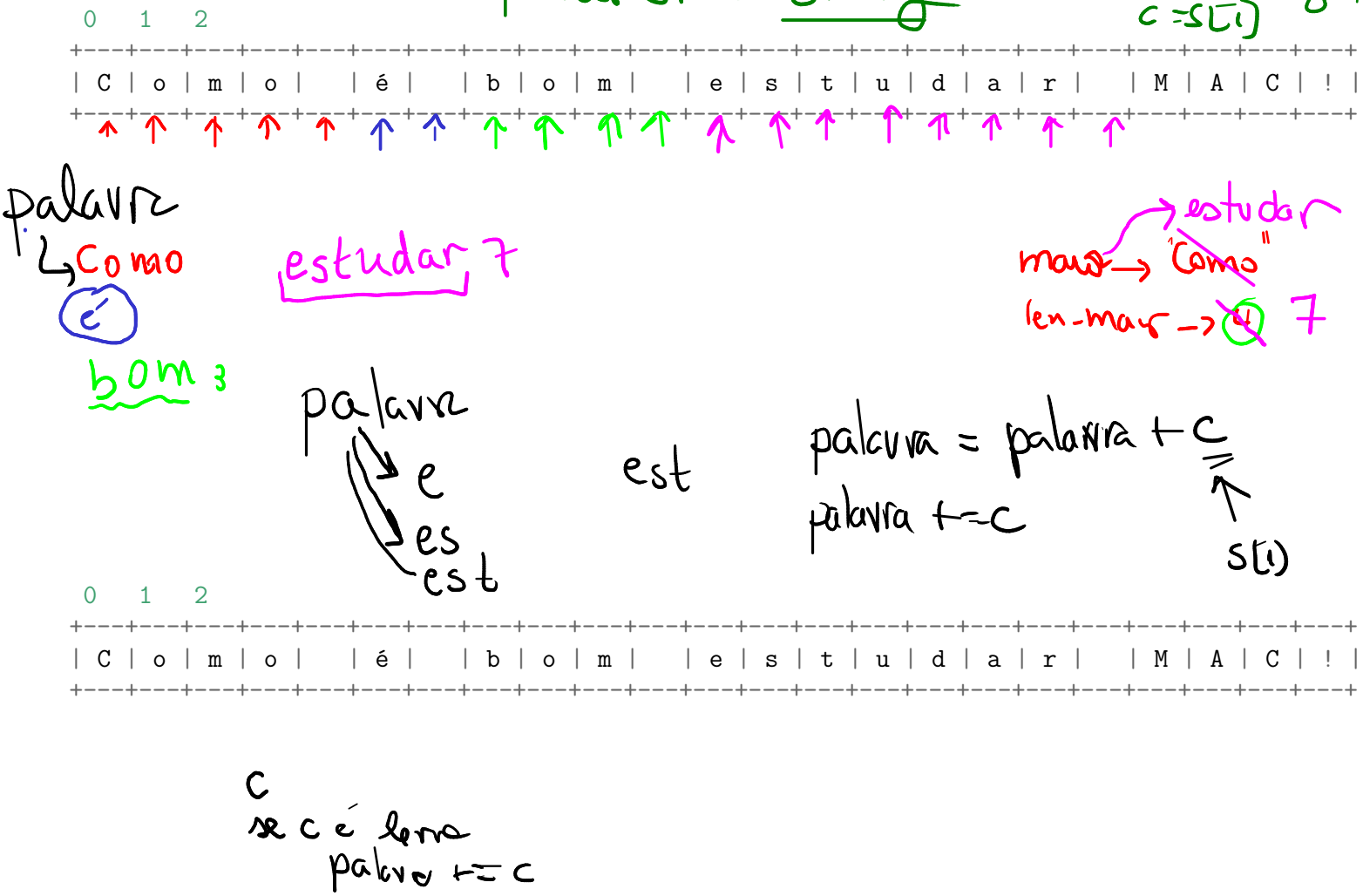
22.5 Exercício: palavra mais longa

Dado um texto determinar a palavra de maior comprimento.

Aqui, por **palavra** entenda-se uma *fatia* de uma string que só tenha letras. Por exemplo, `'Oi'` é uma palavra, `'Oi,'` não é uma palavra.

22.5.1 Exemplos

```
Digite um texto: Como é bom estudar MAC0110!
Maior comprimento de uma palavra = 7 (estudar)

Digite um texto: Na terra de cego quem tem olho é caolho.
Maior comprimento de uma palavra = 6 (caolho)
```




```
c
se c é letra
    palavra += c
```

22.5.2 Solução

```
MAIUSCULAS = "ABCDEFGHIJKLMNOPQRSTUVWXYZ"
MINUSCULAS = "abcdefghijklmnopqrstuvwxyz" # maiusculas.lower() #
LETRAS     = MAIUSCULAS + MINUSCULAS

'''
    Programa que lê um texto e determina uma
    maior palavra do texto.

'''
def main():
    # txt = input("Digite um texto: ")

    # leitura do texto de um arquivo
    # 1. pegue o nome do arquivo
    nome = input("Digite o nome do arquivo: ")
    # 2. "abra" o arquivo para leitura ("r"ead)
    arq = open(nome, "r", encoding="utf-8")
    # 3. leia todo o conteúdo
    txt = arq.read()
    # 4. feche o arquivo
    arq.close()

    print("Texto:\n", txt)
    palavra = maior_palavra(txt)
    print(f"Maior comprimento de uma palavra = {len(palavra)} ({p
```

```
def maior_palavra(s):
    '''(str) -> str

    RECEBE uma string `s`.
    RETORNA (um)a string com (um)a maior palavra de `s`.
    '''
    # maior palavra
    maior = ""
    # comprimento da maior palavra
    len_maior     = 0

    # palavra que estamos percorrendo
    palavra = ''
    # comprimento da palavra que estamos percorrendo
    len_p = 0

    # percorra s
    n = len(s)
    for i in range(n):
        c = s[i]
        if letra(c): # if c.isalpha():
            len_p += 1
            palavra += c
        else:
            # print(f"\n2 nao contou '{frase[i]}',  tam {tam}")
            len_p = 0
            palavra = ''
            # print("%d\n" %tam)

        if len_p > len_maior:
            # print("3 len_maior {len_maior}")
            len_maior = len_p
            maior = palavra
```

```
            # print(f"{len_maior}\n")

    return maior

#-------------------------------------------------------------
# Versão 1
def letra(c):
    ''' (str) -> bool

    Recebe um string s.
    RETORNA True se `s` é uma letra, caso contrário retorna Fals
    '''
    resp = False
    n = len(LETRAS)
    i = 0
    while i < n:
        if c == letras[i]:
            resp = True
        i += 1

    return resp

#-------------------------------------------------------------
# Versão 2
def letra(c):
    ''' (str) -> bool

    Recebe um string s.
    RETORNA True se `s` é uma letra, caso contrário retorna Fals
    '''
    n = len(LETRAS)
    for i in range(n):
        if c == LETRAS[i]:
```

```
            return True
    return resp

#-------------------------------------------------------------
# Versão 3
def letra(c):
    ''' (str) -> bool

    Recebe um string s.
    RETORNA True se `s` é uma letra, caso contrário retorna Fals
    '''
    return c in LETRAS

# início da execução do programa
main()
```