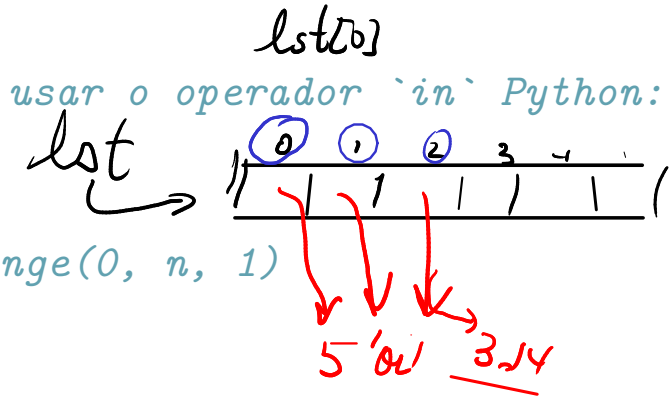
21 Reunião 21: 10/NOV/2020

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

21.1 Reuniões passadas

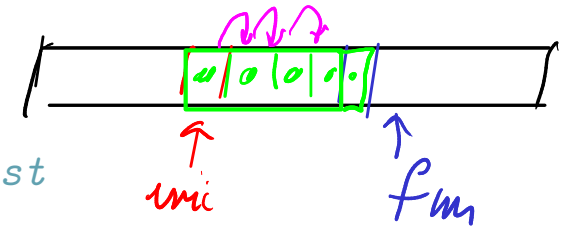
21.1.1 `pertence()` e o operador `in` nativo

```
def pertence(item, lst):
    '''(objeto, list) -> bool

    RECEBE um objeto/coisa `item`  e uma lista `lst`.
    RETORNA True se `item` é um elemento de `lst`, em caso contr

    O efeito é o mesmo que usar o operador `in` Python: `item in
    '''
    n = len(lst)
    for i in range(n): # range(0, n, 1)
        if item == lst[i]:
            return True
    return False

    # equivalente a
    return item in lst
```




```
if item in lst:
    print("Achei")
```

21.1.2 ~~fatia~~ e *slice* nativo




```
def fatia(lst, ini, fim, passo):
    '''(list, int, int, int) -> list

    RECEBE uma lista `lst` e inteiros `ini`, `fim` e `passo`.
    RETORNA um clone da sublista de `lst` que começa na posição
    vai até `fim` (EXCLUSIVE) e agrupa os ítem que de passo em p

    O efeito é o mesmo que o fatiamento do Python: `lst[ini: fim
    '''
    clone = []
    for i in range(ini, fim, passo):
        clone += [lst[i]]
    return clone

    # equivalente a
    return lst[ini: fim: passo]
```

fatia é um clone

clone

zé

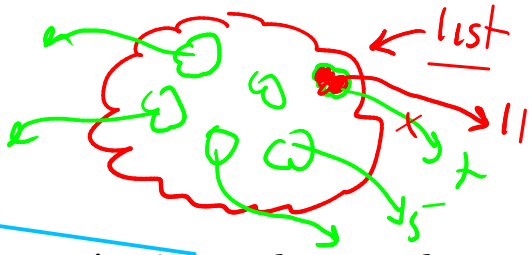
cabo

clone

21.1.3 clones x apelidos: operadores == e is

```
def apelidoXclone():
    lstA = [1, 'oi', True, None, 2.71828]
    # lstB é um apelido para lstA
    lstB = lstA
    print(lstB == lstA) # True
    print(lstB is lstA) # True

    # lstC é igual a lstA, mas é um clone, outra lista
    lstC = lstA[:] # ou lstA[0:len(lstA):1] ou lstA[0:len(lstA)]
    print(lstC == lstA) # True
    print(lstC is lstA) # False
```

21.1.4 Objetos ou coisas mutáveis



Mutabilidade: listas são objetos, coisas **mutáveis**. *podemos alterar um componente*

```
def main():
    lst = [1, True, None, 2.71, 'x-women']
    print(lst)
    lst[0] = True
    print(lst)
    f(lst)

def mutadora(lst):
    '''(list) -> None

    RECEBE uma lista e ALTERA a lista que é argumento.
    Está função é MUTADORA.
    '''
    lst[len(lst)-1] = 'xavier'


def nao_mutadora(lst):
    '''(list) -> None

    RECEBE uma lista e NÃO ALTERA a lista que é argumento.
    Está função não é MUTADORA.

    Atribuição dá um apelido.nome a um objeto/coisa.
    '''
    lst = [1, 2, 3]
```

21.2 Exercício 1 - contador de vogais

Escreva um programa que lê um texto e imprime a frequência relatica de vogais.

21.2.1 Exemplo

```
Digite um texto: Como é bom estudar MAC0110
Frequência das vogais = 8/27 = 0.296296

Digite um texto: Fracassei em tudo o que tentei na vida.
Frequência das vogais = 16/39 = 0.410256
```

21.2.2 Exemplos *de luxe*

```
Digite o nome de um arquivo: jeff.txt

We hold these truths to be self-evident:
that all men are created equal;
that they are endowed by their Creator
with certain unalienable rights;
that among these are life, liberty,
and the pursuit of happiness.

Frequência das vogais = 65/211 = 0.308057


Digite o nome de um arquivo: darci.txt

Fracassei em tudo o que tentei na vida.
Tentei alfabetizar as crianças brasileiras, não consegui.
Tentei salvar os índios, não consegui.
Tentei fazer uma universidade séria e fracassei.
Tentei fazer o Brasil desenvolver-se autonomamente
e fracassei.
Mas os fracassos são minhas vitórias.
Eu detestaria estar no lugar de quem me venceu
Darci Ribeiro

Frequência das vogais = 134/349 = 0.383954
```

21.2.3 Solução

```
VOGAIS = ['a', 'e', 'i', 'o', 'u', 'á', 'é', 'ó', 'à', 'A', 'E',
VOGAIS = "aeiouáéóàAEIOUÁÉÓ"

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(txt)
    n = len(txt)
    no_vogais = conte_vogais(txt)
    print(f"Frequência das vogais = {no_vogais}/n = {no_vogais/n}

# ----------------------------------------------------------
# versão 1 de conte_vogais
def conte_vogais(txt):
    ''' (str) -> int

    Recebe uma string txt e retorna o número de vogais em txt.
    '''
    cont = 0
    n = len(txt)
```

```
    k = len(VOGAIS)
    for i in range(n):
        c = txt[i]
        for j in range(k):
            v = VOGAIS[j]
            if c == v:
                cont += 1
    return cont


# ---------------------------------------------------------
# versão 2 de conte_vogais
def conte_vogais(txt):
    ''' (str) -> int

    Recebe uma string txt e retorna o número de vogais em txt.
    '''
    cont = 0
    n = len(txt)
    for i in range(n):
        c = txt[i]
        if pertence(c, VOGAIS): # usa nossa função pertence
            cont += 1

    return cont

# ---------------------------------------------------------
# versão 3 de conte_vogais
def conte_vogais(txt):
    ''' (str) -> int

    Recebe uma string txt e retorna o número de vogais em txt.
    '''
```

```
    cont = 0
    n = len(txt)
    for i in range(n):
        c = txt[i]
        if c in VOGAIS: # usa pertence nativo: operador `in`
            cont += 1

    return cont


# ---------------------------------------------------------
# versão 4
def conte_vogais(txt):
    ''' (str) -> int

    Recebe uma string txt e retorna o número de vogais em txt.
    '''
    cont = 0
    n = len(txt)
    for c in txt: # usa for ... in str: ...
        if c in VOGAIS:
            cont += 1

    return cont

# ---------------------------------------------------------
main()
```

21.3 Strings

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 **quaquer** *coisa*. Já 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
+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+
| 'I' | 'n' | 's' | 't' | . |   | 'M' | a | t | e | m | á | t | i | c |
+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+
 -16 -15 -14 -13 -12 -11 -10 -9  -8  -7  -6  -5  -4  -3  -2
```