

# FIAP

## Aula 7

---

COMPUTATIONAL THINKING USING PYTHON

# AGENDA

FIAP

1. Introdução a Sequências
2. Compreender o que são sequências
3. Conhecer os tipos de sequências (strings, listas, duplas e intervalos)
4. Sequências mutáveis e imutáveis
5. Sequências homogêneas e heterogêneas
6. Percorrer sequências

Nesta aula iremos abordar o conceito de **sequências** e sua utilização com ***estruturas de repetição*** e ***estruturas condicionais***

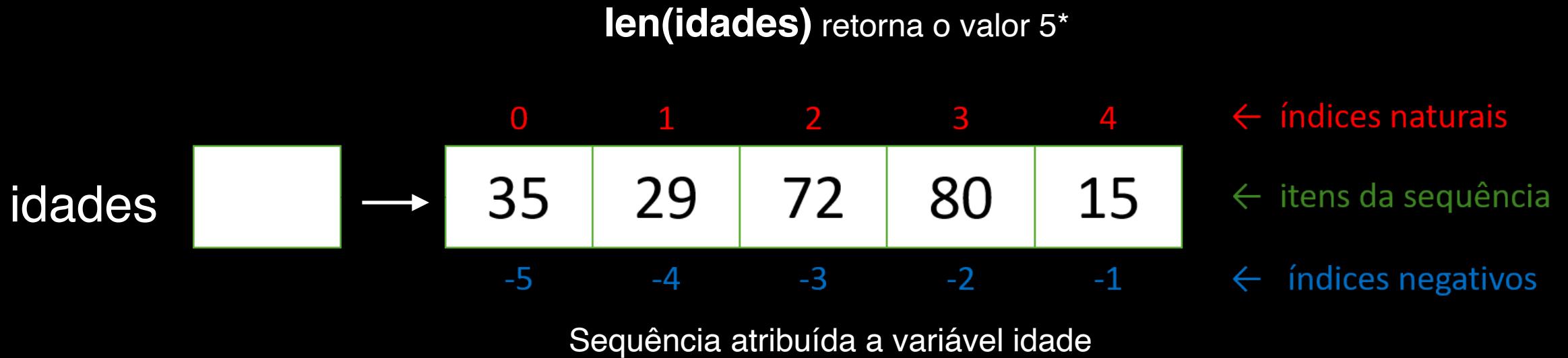
# Sequências

Ao trabalharmos com soluções para problemas mais complexos, é frequente a necessidade de manipulação de grandes quantidades de dados (armazenamento e acesso aos dados)...

# Sequências

- Variável com diversos compartimentos
  - chamados *itens da sequência* (elementos)
- Noção de ordem entre os itens (1o, 2o, 3o...)
- Cada item é identificado dois valores: nome da variável à qual a sequência foi atribuída e um número inteiro referente à sua posição na sequência (índice)
  - índices de uma sequência iniciam em 0 (zero)
- Tamanho da sequência é determinado pelo número de elementos da sequência

# Exemplo



\*A função **len()** recebe como argumento uma sequência e devolve um número inteiro indicando o tamanho desta sequência.

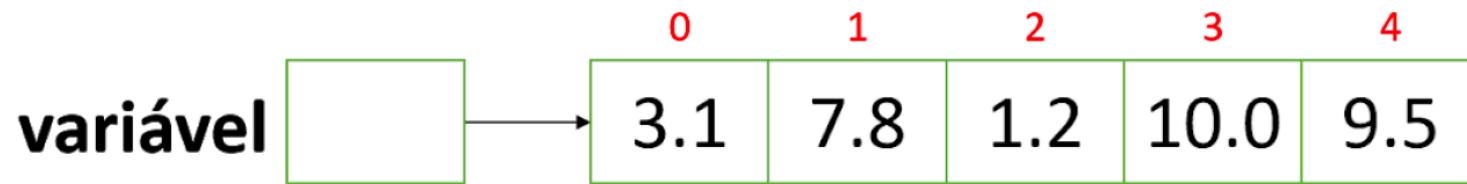
# Acesso aos elementos da sequência

- Para anexar um item específico da sequência, é preciso indexá-la adequadamente a posição em que está o elemento
  - Exemplos:
    - **idades[3]** - acessa o item na posição **3** da sequência **idades**
    - **idades[-2]** - acessa o item na posição **-2** da sequência **idades**

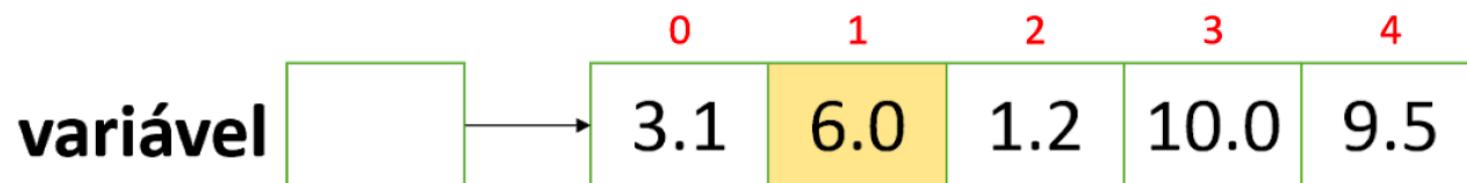
# Sequências mutáveis e imutáveis

- Sequências **Mutáveis**
  - cada item se comporta de modo semelhante a uma variável comum
  - pode receber atribuições, sobrescrevendo seus valores
- Sequências **Imutáveis**
  - uma vez definido, seus itens não podem ser modificados

# Sequência mutável



```
variável[1] = 6.0 # atribui um valor à segunda  
# posição da sequência.
```

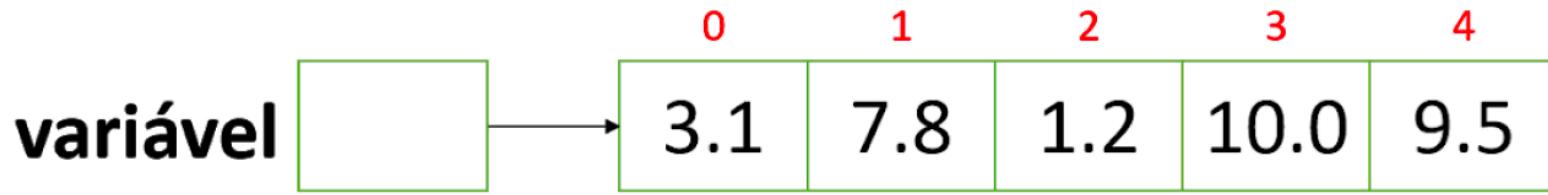


1) Estado inicial de uma sequência **mutável**.

2) Instrução que sobrescreve o conteúdo de um dos itens da sequência mutável.

3) Estado final da sequência mutável após a instrução.

# Sequência imutável



```
variável[1] = 6.0 # tenta atribuir um valor à  
# segunda posição da  
# sequência.
```

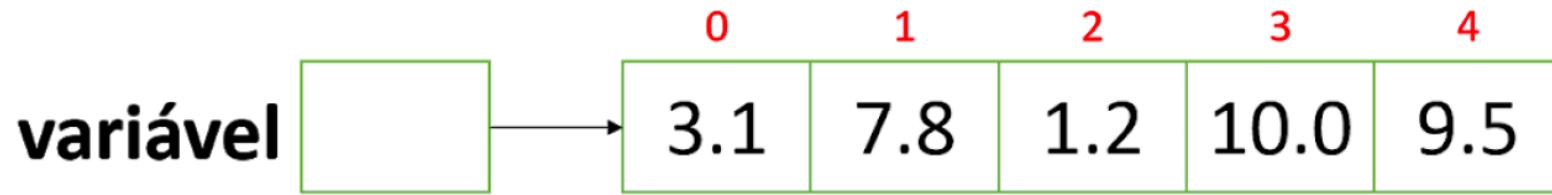
Erro! Não é possível atribuir um novo valor a um item de uma sequência imutável.

1) Estado inicial de uma sequência **imutável**.

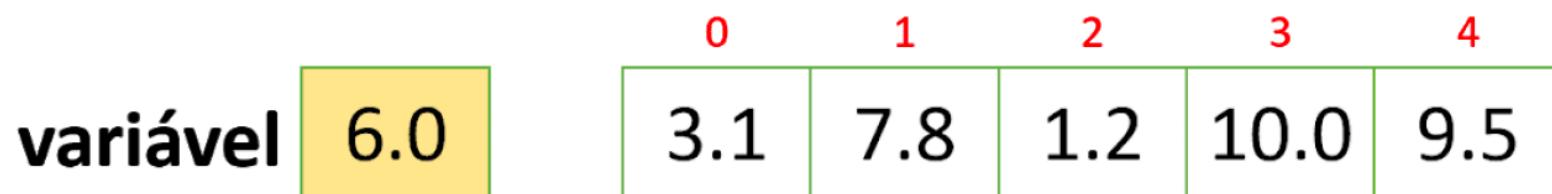
2) Instrução que tenta sobrescrever o conteúdo de um dos itens da sequência imutável.

3) O estado da sequência imutável não foi alterado.

# Modificação da variável (ref. da sequência)



```
variável = 6.0      # atribui um valor à variável  
# que referencia a sequência.
```



1) Estado inicial de uma sequência **qualquer**.

2) Instrução que sobrescreve o conteúdo da variável que referencia a sequência.

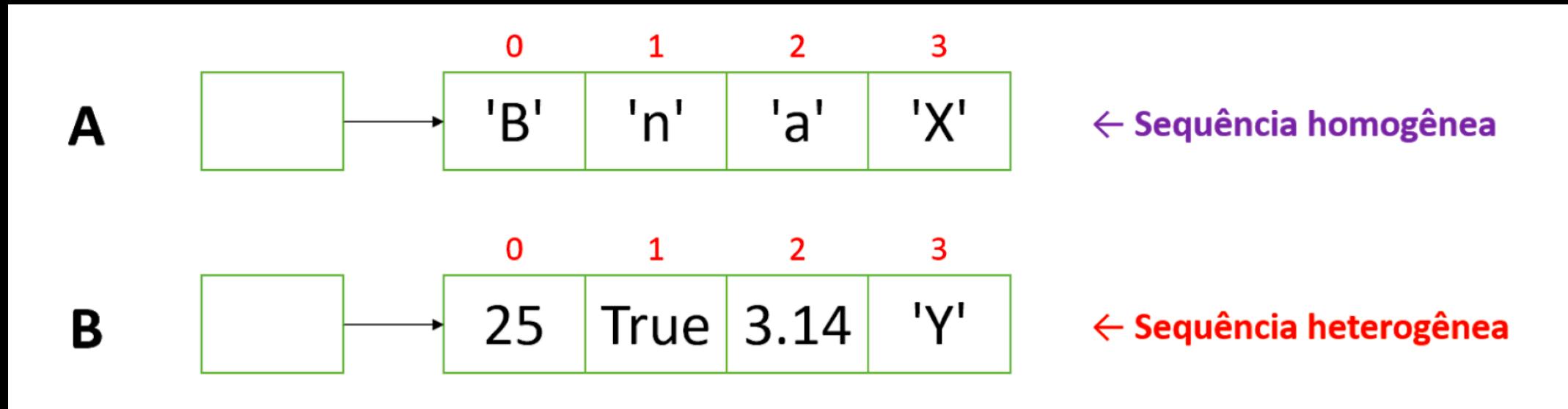
3) O estado final da variável é alterado. A sequência não foi modificada, porém deixou de ser referenciada.

# Sequências Homogêneas e Heterogêneas

Característica que indica a flexibilidade em armazenar itens de tipos distintos simultaneamente

- Sequências **homogêneas**
  - todos os itens devem ser do mesmo tipo
- Sequências **heterogêneas**
  - os itens podem ser de tipos distintos
  - podem conter itens float, int, bool, etc

# Sequências Homogêneas e Heterogêneas



# Tipos de Sequências

**Python** possui três tipos básicos de sequências: listas (*list*), duplas (*tuple*) e intervalos (*range*) e um tipo de sequência para armazenar caracteres (*strings*)



# Strings



# Strings

String em Python são sequências **imutáveis e homogêneas**, em que todos os itens devem ser caracteres **Unicode**

- Construtor de tipo **str()** - permite a conversão para outros tipos
- Assim como em outras sequências, é possível acessar um item específico da string indexando-a.
- Podem ser definidas de várias formas:
  - ‘o carro verde’ (par de apóstrofos)
  - “o carro verde” (pares de aspas)
  - “"o carro verde"" ou “o carro verde” (múltiplas linhas)

# Exemplos

```
>>> texto = '' # atribui uma string vazia à variável.  
>>> texto  
''  
>>> texto = 'Adoro Python!' # atribui uma nova string à variável.  
>>> texto  
'Adoro Python!'  
>>> len(texto) # resulta na quantidade de itens da string.  
13  
>>> texto[0] # acesso ao primeiro item da string.  
'A'  
>>> texto[0] = 'X' # tentativa de alteração da string.  
Traceback (most recent call last):  
  File "<pyshell#18>", line 1, in <module>  
    texto[0] = 'X' # tentativa de alteração da string.  
TypeError: 'str' object does not support item assignment  
>>>
```



# Listas



# Strings

Sequências do tipo *list* são **mutáveis** e **heterogêneas**

- Construtor de tipo `list()` - permite a conversão para outros tipos de sequências em lista
- Podem ser definidas de várias formas:
  - `[ ]` (lista vazia)
  - `[2, True, 1.5]` - itens separados por vírgulas

# Exemplos

The screenshot shows the Python IDLE Shell 3.9.1 window. The title bar reads "IDLE Shell 3.9.1". The menu bar includes "File", "Edit", "Shell", "Debug", "Options", "Window", and "Help". The main area displays the following Python code examples:

```
>>> lista = [] # atribui uma lista vazia à variável.  
>>> lista  
[]  
>>> lista = [3.14, True, 29, 'oi!'] # atribui uma nova lista à variável.  
>>> lista  
[3.14, True, 29, 'oi!']  
>>> len(lista) # resulta na quantidade de itens da lista.  
4  
>>> lista[2] # acesso ao terceiro item da lista.  
29  
>>> lista[3] = 'tchau!' # alteração da lista.  
>>> lista  
[3.14, True, 29, 'tchau!']  
>>>
```

The status bar at the bottom right indicates "Ln: 14 Col: 4".

# Tuplas

# Strings

Sequências do tipo *range* são **imutáveis** e **homogêneas**, gerando um intervalo de números inteiros (muito útil para controle de laços de repetição).

- Construtor de tipo **range()** acompanhando dos argumentos inicio, fim e passo (sempre com números inteiros).
- Podem ser definidas de várias formas:
  - Três argumentos: `range(inicio, fim, passo)` - gera uma sequência com itens no intervalo `[inicio..fim[`, variando de passo em passo
  - Dois argumentos: `range(inicio, fim)` - gera um sequência crescente com itens no intervalo `[inicio..fim[`, variando de 1 em 1
  - Um argumento: `range(fim)` - gera uma sequência crescente com itens no intervalo `[0..fim[`, variando de 1 em 1.

# Exemplos

File Edit Shell Debug Options Window Help

```
>>> intervalo = range(0) # atribui um intervalo vazio à variável.  
>>> intervalo  
range(0, 0)  
>>> intervalo = range(5, 10, 2) # atribui um intervalo [5..10[ e com passo 2.  
>>> intervalo  
range(5, 10, 2)  
>>> lista = list(intervalo) # gera uma lista com os itens do intervalo.  
>>> lista  
[5, 7, 9]  
>>> tupla = tuple(intervalo) # gera uma tupla com os itens do intervalo.  
>>> tupla  
(5, 7, 9)  
>>> intervalo[-1] # acessa o último item do intervalo.  
9  
>>> intervalo[0] = 77 # tentativa de alteração do intervalo.  
Traceback (most recent call last):  
  File "<pyshell#25>", line 1, in <module>  
    intervalo[0] = 77 # tentativa de alteração do intervalo.  
TypeError: 'range' object does not support item assignment  
>>>
```

# Intervalos



# Strings

Sequências do tipo *tuple* são **imutáveis e heterogêneas**. Uso comuns em situações em que é necessário garantir que a sequência não seja modificada (chaves de dicionários, argumento para funções).

- Construtor de tipo **tuple()** - permite a conversão para outros tipos de sequências em tuplas
- Podem ser definidas de várias formas:
  - ( ) (tupla vazia)
  - (2, ) - vírgula a direita indica uma tupla de apenas um item
  - (4, True, 1.5) - itens separados por vírgula

File Edit Shell Debug Options Window Help

```
>>> tupla = () # atribui uma tupla vazia à variável.  
>>> tupla  
()  
>>> tupla = (123,) # atribui uma tupla de apenas um item à variável.  
>>> tupla  
(123,)  
>>> tupla = (3.14, True, 29, 'oi!') # atribui uma nova tupla à variável.  
>>> tupla  
(3.14, True, 29, 'oi!')  
>>> len(tupla) # resulta na quantidade de itens da tupla.  
4  
>>> tupla[1] # acesso ao segundo item da tupla.  
True  
>>> tupla[1] = False # tentativa de alteração da tupla.  
Traceback (most recent call last):  
  File "<pyshell#22>", line 1, in <module>  
    tupla[1] = False # tentativa de alteração da tupla.  
TypeError: 'tuple' object does not support item assignment  
>>>
```

# Percorrendo

Crie um  
entrada  
a média  
salários

```
1 soma = 0
2 salario_0 = float(input('Salário: R$ '))
3 soma += salario_0
4 salario_1 = float(input('Salário: R$ '))
5 soma += salario_1
6 salario_2 = float(input('Salário: R$ '))
7 soma += salario_2
8 salario_3 = float(input('Salário: R$ '))
9 soma += salario_3
10 media = soma / 4
11 if salario_0 < media:
12     print(f'Abaixo da média: R$ {salario_0:.2f}')
13 if salario_1 < media:
14     print(f'Abaixo da média: R$ {salario_1:.2f}')
15 if salario_2 < media:
16     print(f'Abaixo da média: R$ {salario_2:.2f}')
17 if salario_3 < media:
```

```
1 soma = 0
2 salario_0 = float(input('Salário: R$ '))
3 soma += salario_0
4 salario_1 = float(input('Salário: R$ '))
5 soma += salario_1
6 salario_2 = float(input('Salário: R$ '))
7 soma += salario_2
8 salario_3 = float(input('Salário: R$ '))
9 soma += salario_3
10 media = soma / 4
11 if salario_0 < media:
12     print(f'Abaixo da média: R$ {salario_0:.2f}')
13 if salario_1 < media:
14     print(f'Abaixo da média: R$ {salario_1:.2f}')
15 if salario_2 < media:
16     print(f'Abaixo da média: R$ {salario_2:.2f}')
17 if salario_3 < media:
18     print(f'Abaixo da média: R$ {salario_3:.2f}')
```

## Percorrendo sequências (sem repetição e sem listas)

# Percorrendo sequências (com listas)

```
1  salarios = [0, 0, 0, 0]
2  soma = 0
3  salarios[0] = float(input('Salário: R$ '))
4  soma += salarios[0]
5  salarios[1] = float(input('Salário: R$ '))
6  soma += salarios[1]
7  salarios[2] = float(input('Salário: R$ '))
8  soma += salarios[2]
9  salarios[3] = float(input('Salário: R$ '))
10 soma += salarios[3]
11 media = soma / 4
12 if salarios[0] < media:
13     print(f'Abaixo da média: R$ {salarios[0]:.2f}')
14 if salarios[1] < media:
15     print(f'Abaixo da média: R$ {salarios[1]:.2f}')
16 if salarios[2] < media:
17     print(f'Abaixo da média: R$ {salarios[2]:.2f}')
18 if salarios[3] < media:
19     print(f'Abaixo da média: R$ {salarios[3]:.2f}')
```

## Percorrendo sequências (com listas e repetição)

```
1  salarios = [0, 0, 0, 0]
2  soma = 0
3  i = 0 # variável que será usada como índice.
4  while i < 4:
5      salarios[i] = float(input('Salário: R$ '))
6      soma += salarios[i]
7      i += 1
8  media = soma / 4
9  i = 0 # variável que será usada como índice.
10 while i < 4:
11     if salarios[i] < media:
12         print(f'Abaixo da média: R$ {salarios[i]:.2f}')
13     i += 1
```

# Estrutura de repetição *for*

**Python** tem uma estrutura de repetição com quantidade de repetições definida, a estrutura *for* (normalmente utilizada quando sabemos antecipadamente a quantidade de vezes que o bloco de código deve ser repetido)

# Estrutura de repetição *for*

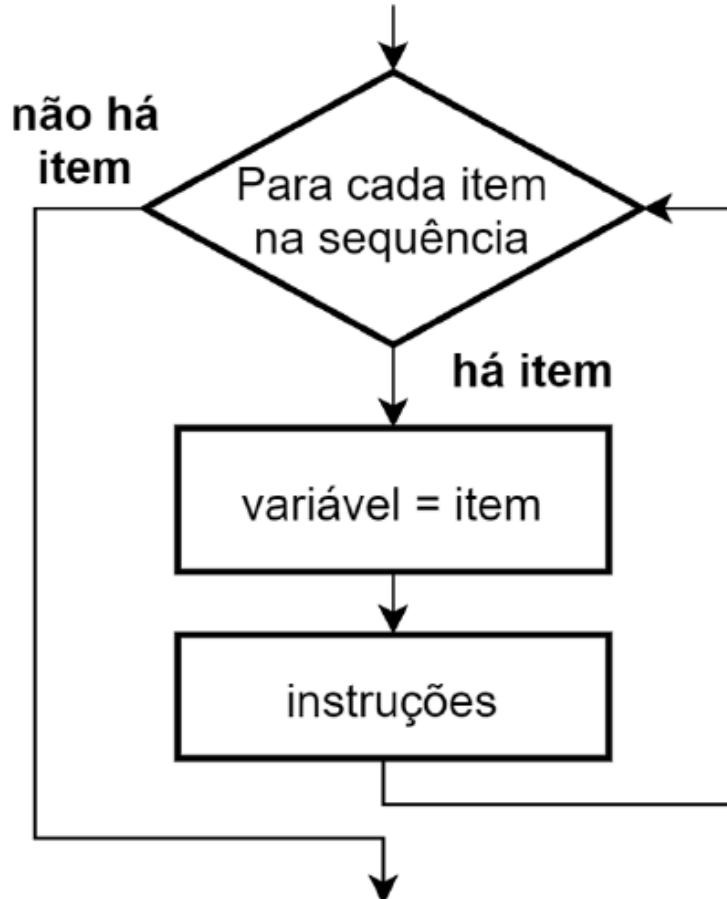
- Conhecido como for each (“para cada”) e podemos interpretar como “para cada item da sequência, faça...”

Sintaxe:

```
for <variável> in sequência:  
    <bloco de código>
```

- Estrutura do for:
  - <sequência> sequência de qualquer tipo de tamanho, até mesmo vazia
  - <variável> é uma variável de controle que, caso não exista precisamente, será criada no inicio da execução do laço.

# Fluxograma - Estrutura de repetição



Para cada item da sequência associada ao laço **for**, o item é atribuído à variável de controle.

Após a atribuição, as instruções do bloco de código são executadas e o fluxo da execução retorna para o início do laço.

Quando não houver mais itens inéditos na sequência para serem atribuídos, o laço é encerrado.

# Exemplo - estrutura for

```
1  salarios = []
2  soma = 0
3  for _ in range(4):
4      salario = float(input('Salário: R$ '))
5      soma += salario
6      salarios.append(salario)
7  media = soma / 4
8  for salario in salarios:
9      if salario < media:
10         print(f'Abaixo da média: R$ {salario:.2f}')
```

## Exemplos de aplicação do for

# Exemplo - estrutura for

```
1 # Crie um programa que exiba o alfabeto minúsculo e maiúsculo.  
2 for letra in 'abcdefghijklmnopqrstuvwxyz':  
3     print(letra)  
4 print()  
5 for letra in 'ABCDEFGHIJKLMNOPQRSTUVWXYZ':  
6     print(letra)
```

```
1 # Crie um programa que exiba, em ordem crescente, os pares de 10  
2 # até 100 e, em ordem decrescente, os ímpares de 100 até 10.  
3 for par in range(10, 101, 2):  
4     print(par, end=' ')  
5 print()  
6 for impar in range(99, 10, -2):  
7     print(impar, end=' ')
```

# Exemplo - estrutura for

```
1 # Crie um programa que exiba os sete dias da semana.  
2 dias = ('domingo', 'segunda', 'terça', 'quarta', 'quinta',  
3 'sexta', 'sábado')  
4 for dia in dias:  
5     print(dia)
```

```
1 # Crie um programa que leia cinco nomes e exiba a quantidade de  
2 # nomes que começam com vogal.  
3 nomes = []  
4 for _ in range(5):  
5     nomes.append(input('Nome: '))  
6 qtd = 0  
7 for nome in nomes:  
8     if (nome[0]=='A' or nome[0]=='E' or nome[0]=='I' or  
9         nome[0]=='O' or nome[0]=='U'):  
10        qtd += 1  
11 print(f'{qtd} dos nomes começam com vogal')
```

# Exercícios

Refaça o código do salário considerando dois novos cenários:

- O programa perguntará quantos salários serão inseridos, em seguida receberá cada salário, calculará a média e exibirá todos os salários que sejam inferiores a media.
- O programa receberá os salários indefinidamente, até que o usuário digite o valor -1 como salário. Então o programa deve seguir o restante do fluxo, calcular a média e exibir todos os salários inferiores, como no primeiro exercício. Neste caso será necessário usar while, pois a quantidade de salários não está definida antes da execução do laço.



Obrigado e até a próxima aula!