

# Aula 1 (2019-04-04)

## Estrutura de dados

Em Ciência da computação, uma **estrutura de dado** é um formato de organização, gerenciamento e armazenamento de dados que habilita acesso e modificação eficiente.

## Revisão

Variáveis simples \*

Python

```
a = 1 # Inteiro
b = 2
c = True # Booleano
d = 'uma string' # String
e = 1.1 # Ponto flutuante (Float)

print(a)
1
print(b)
2
print(c)
True
print(d)
uma string
print(e)
1.1
```

## Operadores \*

Python

```
# Aritmeticos principais: + - * / %
# Extra: ** (potencia)
a = 2
b = 4

a + b
6
a**2
4

# Logicos principais: not and or
# Comparacao: < > <= >= == !=
a = 2
b = 4

a > b
False
b > a
True
```

## Estruturas de repetição \*\*

FOR, WHILE, DO WHILE, REPEAT UNTIL.

FOR \*\*\*

Python    C

```
for ELEMENTO in ITERADOR:
    CORPO DO FOR
```

Um iterador é um "conjunto de elementos", podendo ser vazio, ter apenas um elemento, ou mais do que um elemento. Se for vazio, o corpo (escopo) do `for` não é executado. Caso contrário o corpo do `for` é executado em todos elementos do iterador.

Python

```
for i in [1, 2, 3, 4, 5, 6, 7, 8, 9, 10]:  
    print('i: %s', i)
```

```
1  
2  
3  
4  
5  
6  
7  
8  
9  
10
```

Outros exemplos com FOR:

## Python

```
range(10) # Funcao pronta (interna) do Python.
range(0, 10)

# Transformar o "range" em uma "list" (vetor).
# Para fins de aprendizado, vetor e lista (list) sao sinonimos.
list(range(10))
[0, 2, 3, 4, 5, 6, 7, 8, 9]

list(range(5, 10))
[5, 6, 7, 8, 9]

list(range(5, 10, 2))
[5, 7, 9]

vetor = [1, 2, 3, 4, 5, 6, 7, 8, 9, 10]

tamanho = len(vetor) # Funcao pronta (interna) do Python.
print(tamanho)
10
for i in range(tamanho):
    print('i: %s' % vetor[i])

# Alternativa mais elegante.
for elemento in vetor:
    print('Elemento: %s' % elemento)

# Alternativa elegante enumerada.
for i, elemento in enumerate(vetor):
    print('Elemento %s: %s' % (i, elemento))

list(enumerate(vetor))
[(0, 1), (1, 2), (2, 3), (3, 4), (4, 5), (5, 6), (6, 7), (7, 8), (8, 9)]

nomes = ['joao', 'maria', 'jose']
for i, elemento in enumerate(nomes):
    print('Elemento %s: %s' % (i, elemento))
```

## WHILE \*

Python

```
while CONDICAO:  
    CORPO DO WHILE
```

Enquanto a `condição` for `verdadeira` o `while` é executado indefinidamente; caso contrário o `while` para sua execução.

Python

```
condicao = True  
i = 0  
while condicao:  
    print(i)  
    i += 1  
    if i > 10:  
        condicao = False
```

## Função \*\*

Python

```
def nome(parametro1, parametro2):  
    CORPO DA FUNCAO  
    return VALOR
```

## Exercício: encontrar posição de um elemento em um vetor \*\*

Dado um `vetor` e um `elemento` como entrada para uma função chamada de `find`, retorne a posição do elemento encontrado; caso não seja encontrado retorne `None` (nulo).

Python

```
def find(vetor, elemento):
```

## Exemplos e revisão

### Vetor (Arranjo)

FORMATO:

Python    C

```
# Em python o nome dessa estrutura de dado eh conhecida como lista
# Mas para fins de exemplo criamos uma variavel com o nome vetor.
# Indices do vetor comeca em 0.
vetor = [1, 2, 3, 4, 5, 6, 7, 8, 9, 10]
print(vetor[0])
1
```

GERENCIAMENTO:

Em um vetor podemos: adicionar, atualizar, excluir e ler elementos.

Adicionar \*

Adição modifica o vetor original, aumentando pelo menos um elemento novo.

Python

```
vetor = [1, 2, 3, 4, 5, 6, 7, 8, 9, 10]
vetor.append(11) # Metodo (funcao) de orientacao a objeto.
vetor
[1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11]

# Apenas para mostrar como eh no "paradigma estrutural".
def meu_append(vetor, novo_elemento):
    vetor.append(novo_elemento)
    # Funcao nao retorna nada, podemos chamar de "procedimento".

vetor = [1, 2, 3, 4, 5, 6, 7, 8, 9, 10]
meu_append(vetor, novo_elemento)
vetor
[1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11]
```

Atualizar \*

Atualização precisa modificar algum elemento já existente do vetor original.

Python

```
vetor = [1, 2, 3, 4, 5, 6, 7, 8, 9, 10]
vetor[2] = 11
vetor
[1, 2, 11, 4, 5, 6, 7, 8, 9, 10]
```

Remover \*

Python

```
vetor = [1, 2, 3, 4, 5, 6, 7, 8, 9, 10]
vetor.remove(5) # Qual algoritmo eh necessario ANTES de remover?
vetor
[1, 2, 3, 4, 6, 7, 8, 9, 10]
# TODO: exercicio: fazer o seu proprio remover.
def meu_remove(vetor, elemento):
    pass
    # Funcao nao retorna nada
```

Ler \*

Python

```
# Leitura NAO pode modificar o nosso vetor original
vetor = [1, 2, 3, 4, 5, 6, 7, 8, 9, 10]
a = vetor[2]
a
2
# Leitura direta do vetor para impressao
print(vetor[2])
```

ARMAZENAMENTO: \*\*

Memória RAM, registradores do processador (CPU), dispositivo de armazenamento secundário (HDDs, SSDs, etc).

Memória RAM \*\*

Um vetor de caracteres armazenados em memória RAM.

```
# Assumindo que cada caracter tem um byte.
```

```
vetor = ['1', '2', '3', '4', '5', '6', '7', '8', '9', '10']
```

Endereço	Dado
----------	------

100000	'1'
100001	'2'
100002	'3'
100003	'4'
100004	'5'
100005	'6'
100006	'7'
100007	'8'
100008	'9'
100009	'10'