

CC8210 – NCA210

Programação Avançada I

Prof. Reinaldo A. C. Bianchi
Prof. Isaac Jesus da Silva
Prof. Danilo H. Perico

Estruturas de Repetição

Repetições

- São utilizadas para executar **várias vezes** a mesma parte do programa
- Normalmente dependem de uma condição
- Repetições são a base de vários programas!

Estruturas de Repetição

Comando *while*

Comando **while**

- O comando **while** (enquanto) serve para executarmos alguma repetição **enquanto** uma condição for verdadeira (True)

```
• while <condição>:  
    #bloco que será repetido enquanto a condição for verdadeira
```

Comando **while**

```
x = 1
while x <= 10:
    print(x)
    x = x + 1
```

1
2
3
4
5
6
7
8
9
10

Exemplo

- Impressão do número 1 até o número digitado pelo usuário:

```
ultimo = int(input("Digite o último dígito da contagem: "))  
i = 1  
while i <= ultimo:  
    print(i)  
    i += 1
```

Equivalente a $i = i + 1$

Digite o último dígito da contagem: 5

1
2
3
4
5

- O lado direito do sinal de igual (=) é executado primeiro!
- O resultado é atribuído para a variável que estiver do lado esquerdo do sinal de igual (=)

Exemplo

- Impressão do número 1 até o número digitado pelo usuário:

```
ultimo = int(input("Digite o último dígito da contagem: "))  
  
i = 1  
  
while i <= ultimo:  
    print(i)  
    i += 1
```

i é um contador

um contador é uma
variável que conta o
número de ocorrências de
um evento: neste caso, o
número de repetições!

```
Digite o último dígito da contagem: 5  
1  
2  
3  
4  
5
```


Exemplo - combinando repetição com **if**

- Programa que imprime todos os números pares de 0 até um número digitado pelo usuário.

```
ultimo = int(input("Digite o último dígito da contagem: "))  
  
i = 0  
  
while i <= ultimo:  
    if i % 2 == 0:  
        print(i)  
    i += 1
```

Exemplo - combinando repetição com **if**

- Programa que imprime todos os números pares de 0 até um número digitado pelo usuário.

```
ultimo = int(input("Digite o último dígito da contagem: "))  
  
i = 0  
  
while i <= ultimo:  
    if i % 2 == 0:  
        print(i)  
    i += 1
```

```
Digite o último dígito da contagem: 6  
0  
2  
4  
6
```

while infinito

- Muitas vezes, queremos que nossos programas sejam executados infinitamente
- Nesses casos, podemos utilizar uma condição que nunca deixe de ser verdadeira (True)

while infinito

```
while True:  
    #bloco que sempre será executado,  
    #nunca sai do loop de repetição
```

Comando **break**

- Porém, mesmo quando utilizamos um **while** infinito, é possível que em determinadas situações o programa precise sair do loop de repetição.
- Esta interrupção pode ser alcançada com o comando **break**
- O comando **break** pode ser utilizado para interromper o while, independentemente da condição

Comando **break** - Exemplo

- Somatória de valores digitados pelo usuário até que o número 0 (zero) seja digitado; quando 0 for digitado o resultado da somatória é exibido:

```
somatoria = 0

while True:
    entrada = int(input("Digite um número a somar ou 0 para sair:"))
    if entrada == 0:
        break
    else:
        somatoria = somatoria + entrada

print("Somatória", somatoria)
```

```
Digite um número a somar ou 0 para sair:5
Digite um número a somar ou 0 para sair:6
Digite um número a somar ou 0 para sair:4
Digite um número a somar ou 0 para sair:0
Somatória 15
```

Repetições aninhadas

- Podemos combinar vários **while**, um dentro do outro!
- Com isso, conseguimos alterar automaticamente o valor de mais do que somente uma variável

Exemplo

- Programa para calcular as tabuadas do número 1 até o número 10.

```
tabuada = 1
while tabuada <= 10:
    print()
    print("Tabuada do", tabuada)
    multiplicador = 0
    while multiplicador <= 10:
        print(tabuada, "x", multiplicador, "=", (tabuada*multiplicador))
        multiplicador += 1
    tabuada += 1
```

```
Tabuada do 1
1 x 0 = 0
1 x 1 = 1
1 x 2 = 2
1 x 3 = 3
1 x 4 = 4
1 x 5 = 5
1 x 6 = 6
1 x 7 = 7
1 x 8 = 8
1 x 9 = 9
1 x 10 = 10
```

```
Tabuada do 2
2 x 0 = 0
2 x 1 = 2
2 x 2 = 4
2 x 3 = 6
```


Estruturas de Repetição

Comando *for*

Comando for

- for é a estrutura de repetição mais utilizada

- Sintaxe:

```
for <referência> in <sequência>:  
    #bloco de código que será repetido  
    #a cada iteração
```

- Durante a execução, a cada iteração, a referência aponta para um elemento da sequência.
- Uma vantagem do for com relação ao while é que o contador não precisa ser explícito!

Comando **for** - Exemplo

- Calcular a somatória dos números de 0 a 99

```
somatoria = 0  
  
for x in range(0,100):  
    somatoria = somatoria + x  
print(somatoria)
```

4950

A função **range(i, f, p)** é bastante utilizada nos laços com **for**

Ela gera um conjunto de valores inteiros:

- Começando de **i**
- Até valores menores que **f**
- Com passo **p**

Se o passo **p** não for definido, o padrão de 1 será utilizado.

Comando **else** na repetição

- É possível a utilização do comando **else** nas estruturas de repetição
- Tanto no **while** quanto no **for**
- A cláusula **else** só é executada quando a condição do *loop* se torna falsa.
- Se você sair do *loop* com o comando **break**, por exemplo, ela não será executada.

Comando **else** na repetição

```
i = 0
while i < 11:
    print(i)
    i+=2
else:
    print("Os números pares de 0 a 10 foram exibidos")
```

```
0
2
4
6
8
10
Os números pares de 0 a 10 foram exibidos
```

```
for i in range(0,11,2):
    print(i)
else:
    print("Os números pares de 0 a 10 foram exibidos")
```

```
0
2
4
6
8
10
Os números pares de 0 a 10 foram exibidos
```

Comando **else** na repetição

- Exemplo com **break**: o **else** não é executado

```
1 i = 0
2
3 while i < 11:
4     print(i)
5     i+=2
6     if i == 8:
7         break
8 else:
9     print("Os números pares de 0 a 10 foram exibidos")
10
```

0
2
4
6


Comando **continue** na repetição

- O comando **continue** funciona de maneira parecida com o **break**, porém o break interrompe e sai do *loop*;
- Já o **continue** faz com que a próxima iteração comece a ser executada, não importando se existem mais comandos depois dele ou não
- O **continue** não sai do *loop*
- O **continue** faz com que a próxima iteração seja executada imediatamente

Comando **continue** na repetição

- Exemplo com **continue**

```
1 i = 0
2
3 while i < 12:
4     i+=2
5     if i == 8:
6         continue
7     print(i)
8 else:
9     print("Os números pares de 2 a 12 foram exibidos, com exceção do 8")
```



- Chama a próxima iteração
- Não executa os comandos da própria iteração: neste caso pula o *print()* com *i = 8*

```
2
4
6
10
12
Os números pares de 2 a 12 foram exibidos, com exceção do 8
```


Pseudocódigo

ENQUANTO ($a < b$) **FAÇA**

Comandos para condição verdadeira

FIM-ENQUANTO

ou

WHILE ($a < b$) **DO**

Comandos para condição verdadeira

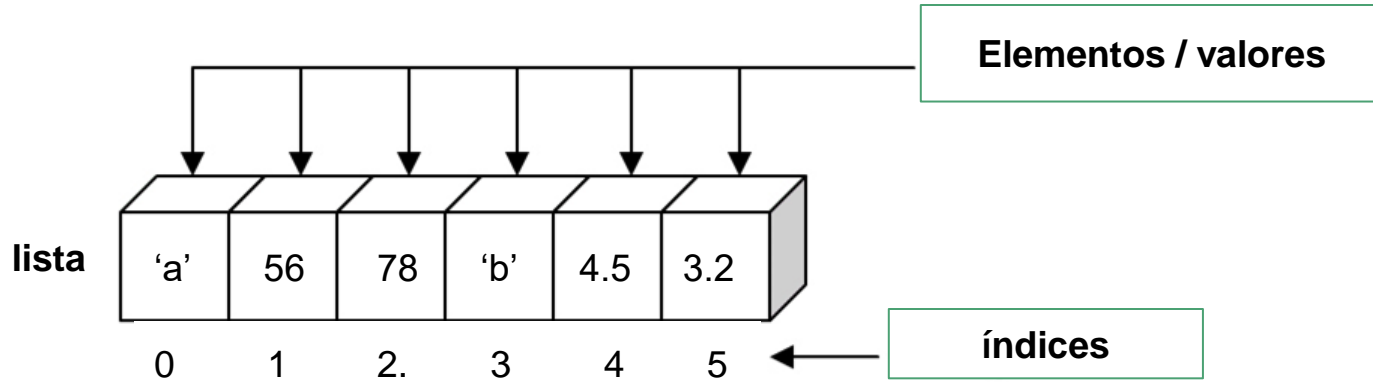
END-WHILE

Lista

Lista

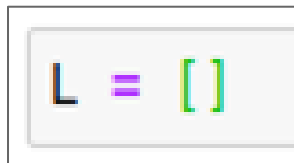
- Uma lista é uma variável que armazena um conjunto de valores.
- Lista é um tipo de variável que permite o armazenamento de valores com tipos homogêneos ou heterogêneos:
 - Homogêneos = do mesmo tipo.
 - Heterogêneo = de tipos diferentes.
- Os valores armazenados em uma lista são acessados por um índice.

Lista heterogênea



Lista

- Para indicar que uma variável é uma lista, o símbolo de colchetes [] é utilizado para delimitar o conjunto
- Sintaxe - criando uma lista chamada L:



```
L = []
```

A code snippet showing the creation of an empty list in Python. The variable 'L' is followed by an equals sign and a pair of empty square brackets '[]'.

- L é uma lista vazia

Lista - Exemplo

- Criando uma lista chamada z com 3 números inteiros

```
z = [5, 7, 1]  
print(z)
```

```
[5, 7, 1]
```

- Dizemos que z tem tamanho 3

Lista - Adicionando elementos no fim da lista

- Podemos ainda adicionar novos elementos no fim da lista
- Para isto, utilizamos o método *append(item)*
- Exemplo:

```
z = [32, 7, 1]  
print(z)
```

```
[32, 7, 1]
```

```
z.append("oi")  
print(z)
```

```
[32, 7, 1, 'oi']
```

Lista - Adicionando elemento em qualquer lugar

- Podemos ainda adicionar novos elementos em qualquer lugar da lista
- Para isto, utilizamos o método *insert(índice, item)*
- Exemplo:

```
z = [32, 7, 1]  
print(z)
```

```
[32, 7, 1]
```

```
z.insert(1, "oi")  
print(z)
```

```
[32, 'oi', 7, 1]
```


Lista - Removendo da lista pelo índice

- Podemos remover um elemento da lista
- Para isto, utilizamos o método *pop(índice)*
- Exemplo:
- Se nenhum índice for especificado, ele retorna o ultimo elemento.

```
z = ["a", "b", "c", "d", "e"]  
print(z)
```

```
['a', 'b', 'c', 'd', 'e']
```

```
z.pop(1)  
print(z)
```

```
['a', 'c', 'd', 'e']
```

Lista - Removendo da lista pelo elemento

- Podemos remover um elemento da lista
- Para isto, utilizamos o método *remove(item)*
- Exemplo:

```
z = ["a", "b", "c", "d", "e"]  
print(z)
```

```
['a', 'b', 'c', 'd', 'e']
```

```
z.remove("d")  
print(z)
```

```
['a', 'b', 'c', 'e']
```

```
z = [1,2,3,1,4,5,1]  
z.remove(1)  
print(z)
```

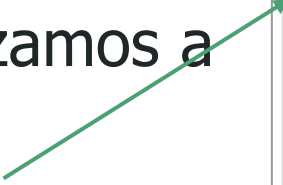
```
[2, 3, 1, 4, 5, 1]
```

Lista - Cópia

- Para criarmos uma cópia independente, utilizamos a sintaxe:

`z1 = z[:]`

- Outra opção é utilizar o método `list.copy()`.
 - Return a shallow copy of the list.
 - Equivalent to `a[:]`.



```
z = [4,5,3,6]
z1 = z[:]
print("antes da alteração na lista z")
print(z)
print(z1)
z[1] = 98
print("depois da alteração na lista z")
print(z)
print(z1)
```

```
antes da alteração na lista z
[4, 5, 3, 6]
[4, 5, 3, 6]
depois da alteração na lista z
[4, 98, 3, 6]
[4, 5, 3, 6]
```

Lista - Tamanho da lista

- Como temos os métodos para incluir e remover dados das listas, nem sempre sabemos qual é o tamanho exato que a lista tem
- Para descobrirmos o tamanho da lista, utilizamos o método *len(lista)*
- Exemplo:

```
a = [3, 4, 5]  
print(len(a))
```

```
a.append(9)  
a.append(11)  
print(len(a))
```

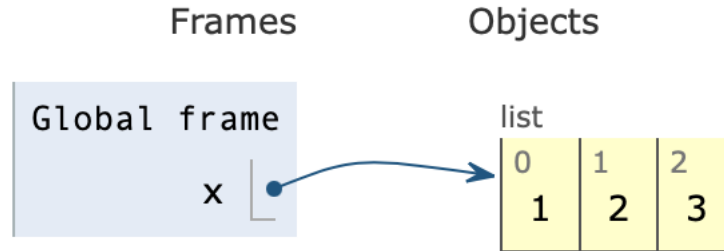
```
3  
5
```

Python Tutor

- Podemos usar o Python Tutor para verificar graficamente o comportamento dos nossos programas.
 - Python Tutor helps people overcome a fundamental barrier to learning programming: understanding what happens as the computer runs each line of code.
 - You can use it to write Python, Java, C, C++, JavaScript, and Ruby code in your web browser and see its execution visualized step by step.
- <http://www.pythontutor.com>
 - Ver exemplo "hello" em Python

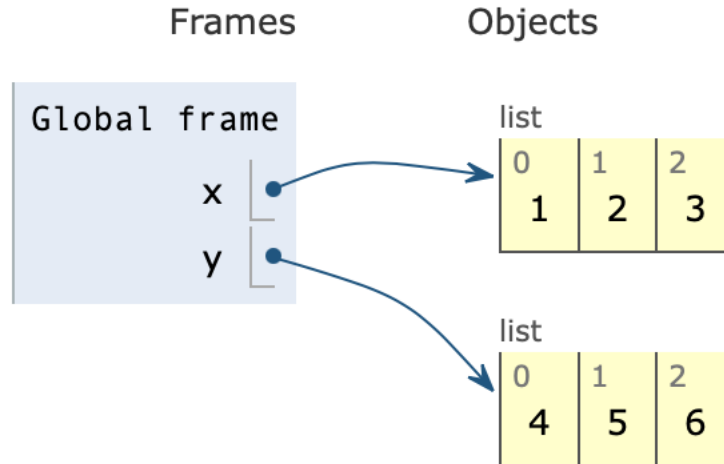
Python Tutor

- `x = [1, 2, 3]`



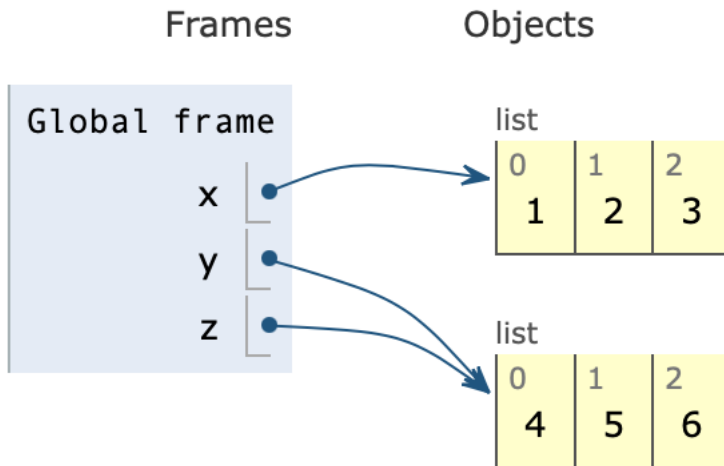
Python Tutor

- `x = [1, 2, 3]`
- `y = [4, 5, 6]`



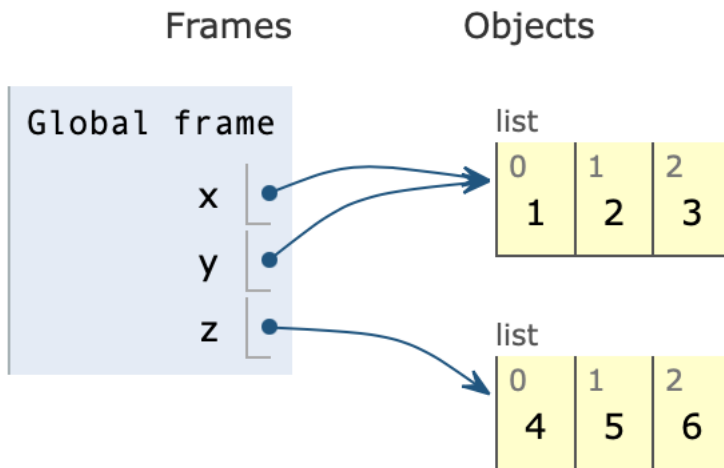
Python Tutor

- `x = [1, 2, 3]`
- `y = [4, 5, 6]`
- `z = y`



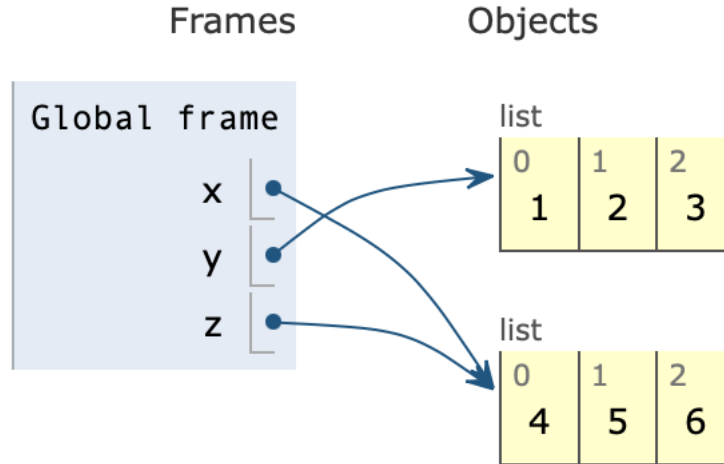
Python Tutor

- `x = [1, 2, 3]`
- `y = [4, 5, 6]`
- `z = y`
- `y = x`



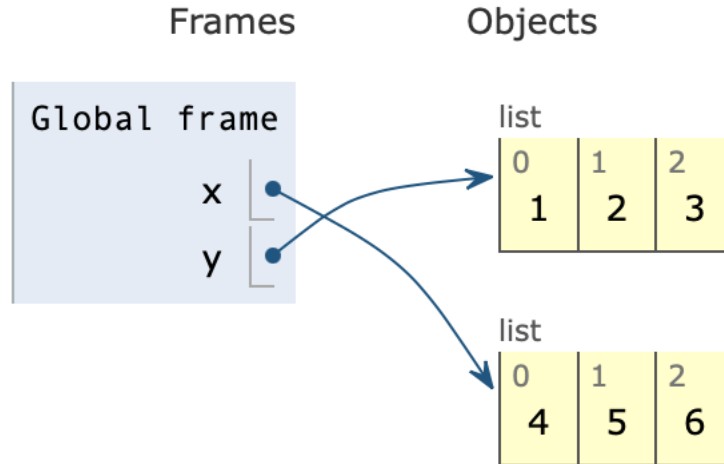
Python Tutor

- `x = [1, 2, 3]`
- `y = [4, 5, 6]`
- `z = y`
- `y = x`
- `x = z`



Python Tutor

- `x = [1, 2, 3]`
- `y = [4, 5, 6]`
- `z = y`
- `y = x`
- `x = z`
- `del (z)`



Acessar cada um dos elementos da Lista

Para acessar cada um dos elementos do vetor usamos o comando *for*

- Sintaxe:

for <nome da variável> **in** <nome do vetor>:

- Exemplos:

```
for x in vetSaldo:  
    print(x)
```

```
for x in vetNomes:  
    print(x)
```

Acessar itens na Lista

Para acessar um item em uma Lista é necessário indicar em qual posição está o item que será acessado

- Sempre temos que escrever o nome da Lista e a posição entre colchetes
- **IMPORTANTE:** o primeiro item do vetor sempre está na posição zero
- Sintaxe:

<nome da Lista> [<posição>]

Acessando itens no vetSaldo

vetSaldo	
0	10.00
1	999.99
2	87.60
3	159.90
4	230.00

Exemplos:

```
vetSaldo = [10.00, 999.99, 87.60, 159.90, 230.00]
```

- Mostrar terceiro item:

```
print (vetSaldo[2])
```
- Atribuir primeiro item a uma variável:

```
valor = vetSaldo[0]
```

Acessando itens no vetEstoque

vetEstoque

0	123
1	64
2	100
3	26
4	555
5	975
6	87
7	3

Exemplos:

`vetEstoque = [123, 64, 100, 26, 555, 975, 87, 3]`

- Mostrar último item:

`print(vetEstoque[7])`

- Atribuir penúltimo item a uma variável:

`numero = vetEstoque[6]`

Substituindo itens na Lista

- Para substituir um item em uma Lista é necessário indicar em qual posição do item que será substituído
- Sempre temos que escrever o nome do vetor, a posição entre colchetes, o sinal de atribuição e o valor a ser colocado no vetor
- Sintaxe:

<nome do vetor> [<posição>] = <valor>;

Substituindo itens no vetSaldo

vetSaldo	
0	10.00
1	999.99
2	87.60
3	159.90
4	230.00

Exemplos:

`vetSaldo = [10.00, 999.99, 87.60, 159.90, 230.00]`

- Substituir terceiro item:

`vetSaldo[2] = 123.45`

vetSaldo	
0	10.00
1	999.99
2	123.45
3	159.90
4	230.00

Acessando itens no vetEstoque

vetEstoque

0	123
1	64
2	100
3	26
4	555
5	975
6	87
7	3

Exemplos:

vetEstoque = [123, 64, 100, 26, 555, 975, 87, 3]

- Substituir o último item:

vetEstoque[7] = 21

vetEstoque

0	123
1	64
2	100
3	26
4	555
5	975
6	87
7	21

Exercícios

Exercício 01

- Crie uma Lista de números inteiros de tamanho 5.
Alimente essa Lista com entradas fornecidas pelo usuário.
Exiba como saída o conteúdo do índice 3 dessa Lista.
(Usar estrutura de repetição)

Exercício 02

- Faça um algoritmos que armazene 10 números inteiros informados pelo usuário em uma Lista. Exiba como saída o MAIOR numero dessa Lista.

OBRIGATÓRIO O USO DE LAÇO DE REPETIÇÃO PARA LEITURA DA LISTA.

Exercício 03

- Crie uma Lista de números inteiros $V[5]$. Inicialize esse vetor com números fixos e aleatórios (a sua escolha). Exiba como saída a média dos valores desse vetor.
OBRIGATÓRIO O USO DE LAÇO DE REPETIÇÃO PARA LEITURA DA LISTA.

Exercício 04

- Faça um algoritmo que leia 2 vetores $A[10]$ e $B[10]$.
A seguir, crie um vetor C que seja a intersecção de A com B e mostre este vetor C .
Obs.: Intersecção é quando um valor estiver nos dois vetores. Considere que não há elementos duplicados em cada um dos vetores.