

CC1612 - Fundamentos de Algoritmos

Centro Universitário FEI

Prof. Danilo H. Perico

Estruturas de Repetição

continuação

Relembrando... **while**

- Programa para calcular a somatória de 10 números que devem ser digitados pelo usuário.

```
n = 1
soma = 0

while n <= 10:
    x = int(input("Digite o %d número:" % n))
    soma = soma + x
    n += 1
print("Soma", soma)
```

```
Digite o 1 número:5
Digite o 2 número:6
Digite o 3 número:4
Digite o 4 número:7
Digite o 5 número:8
Digite o 6 número:9
Digite o 7 número:5
Digite o 8 número:2
Digite o 9 número:1
Digite o 10 número:3
Soma 50
```

Exercício

21. Escreva um programa que converta um número decimal (base 10) em binário (base 2). Leia o número decimal do usuário como um inteiro e, em seguida, use o pseudocódigo de divisão mostrado abaixo para realizar a conversão. Quando o algoritmo é concluído, o resultado contém a representação binária do número. Exiba o resultado.

Início

Deixe o **resultado** ser uma string vazia

Deixe **q** representar o número para converter

Repita

Defina **r** igual ao resto quando **q** é dividido por 2

Converta **r** para uma string e o adicione ao início do **resultado**

Divida **q** por 2, descartando qualquer resto, e armazene o resultado de volta em **q**

Até **q** ser igual a 0

Fim

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!

* iteração: ato de iterar**; repetição.

** iterar: tornar a fazer; repetir.

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**, pois 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.

Exercício

22. Este exercício examina o processo de identificação do valor máximo em uma coleção de inteiros. Cada um dos números inteiros será selecionado aleatoriamente entre os números 1 e 100. A coleção de inteiros pode conter valores duplicados, e alguns dos inteiros entre 1 e 100 podem não estar presentes.

Faça um programa que gera um número aleatório e, na sequência, compara com o maior número armazenado anteriormente. No final das 100 comparações, exiba qual foi o maior número gerado e quantas vezes o maior número foi atualizado no seu código.

```
from random import randrange  
numero = randrange(1, 101)
```

Cláusula **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.

```
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
```

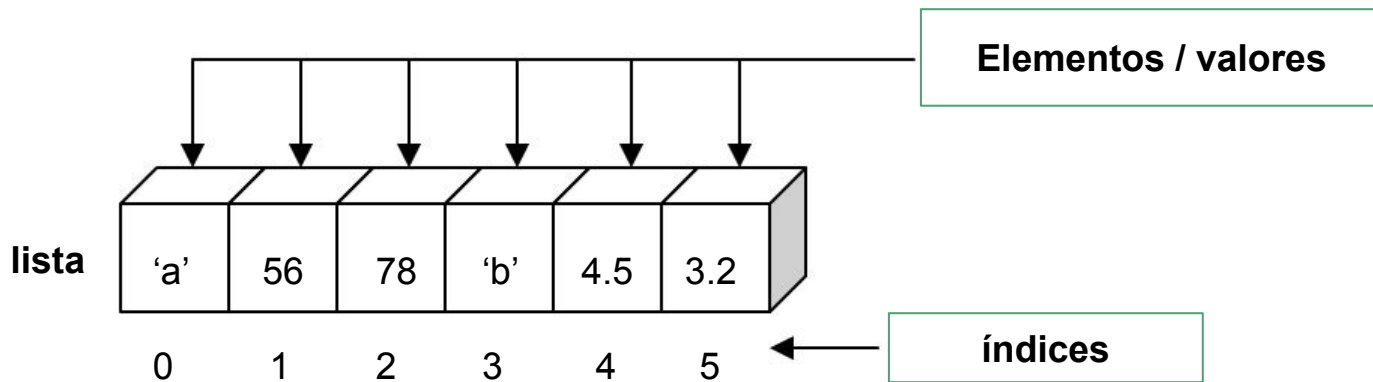

Comando **continue** na repetição

- O comando **continue** funciona de maneira parecida com o **break**, porém o **break** interrompe e sai do loop;
- Já o **continue** volta a realizar o loop desde o começo, não importando se existem mais comandos depois dele ou não
- O **continue** não sai do loop

Listas

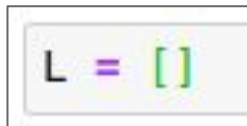
Lista

- Lista é um tipo de **variável** que permite o armazenamento de vários valores
- Uma lista armazena um conjunto de valores
- As listas podem conter valores do mesmo tipo ou de tipos diferentes
- Os valores armazenados em uma lista são acessados por um índice



Lista

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



```
L = []
```

A code snippet showing the creation of an empty list. 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 - Acesso aos elementos

- Exemplo:

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

[5, 7, 1]

- Para acessarmos o primeiro número da lista *z*, utilizamos a notação: ***z[0]***
- Ou seja, da lista *z* queremos pegar o valor armazenado no índice 0.

```
z = [5, 7, 1]
print(z[0])
print(z[1])
print(z[2])
```

5
7
1

Lista

- Utilizando o nome de uma lista com o índice desejado, podemos também modificar o conteúdo armazenado.
- Exemplo: Alterando o valor do primeiro elemento da lista z

```
z = [5, 7, 1]
```

```
z[0] = 32
```

```
print(z)
```

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

Lista - Cópia

- A cópia de uma lista para uma nova variável requer alguma atenção!
- Por exemplo, se quisermos copiar a lista `z` para uma nova variável chamada `z1`, o mais natural seria o seguinte:

`z1 = z`

- Porém, quando fazemos isso no Python, criamos duas variáveis que referenciam a mesma lista!
- É como se déssemos dois nomes para a mesma lista

Lista - Cópia

- Exemplo:
- Quando alteramos o elemento na lista `z`, a alteração ocorre também na lista `z1`

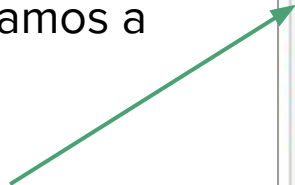
```
z = [4,5,3,6]
z1 = z
print("antes da alteração na lista z")
print(z)
print(z1)
z[1] = 98
print("depois da alteração na lista z")
print(z)
print(z1)
```

```
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, 98, 3, 6]
```

Lista - Cópia

- Para criarmos uma cópia independente, utilizamos a sintaxe:

z1 = z[:]



```
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 - Fatiamento

- No Python, podemos também fatiar as listas
- Ou seja, pegar somente partes de uma lista
- Exemplo:

```
p = [1,2,3,4,5,6]  
print(p[0:5])
```

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

```
print(p[:4])
```

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

```
print(p[1:3])
```

```
[2, 3]
```

```
print(p[-1])
```

```
6
```

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:

```
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 - 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

Lista - Pesquisando na lista

- Podemos pesquisar se um elemento está na lista
- Para isso, verificamos do primeiro ao último comparando com o que queremos encontrar.
- Para percorrer listas, utilizamos a estrutura de repetição: *while* ou *for*
- A estrutura *for* é otimizada para trabalhar com listas

Lista - Pesquisando na lista

- Exemplo: Procurar o elemento “c” na lista z

```
z = ["a", "b", "c", "d", "e"]
for elemento in z:
    if elemento == "c":
        print("Elemento encontrado!")
        break
    else:
        print("Elemento não encontrado!")
```

Elemento encontrado!

Lista - Pesquisando na lista

- Porém, se a ideia é somente falar se o elemento está ou não na lista, podemos utilizar uma estrutura mais simples:

```
z = ["a", "b", "c", "d", "e"]  
if "c" in z:  
    print("Encontrado!")  
else:  
    print("Não encontrado!")
```

Encontrado!

Lista - Pesquisando na lista

- Contudo, nem sempre encontrar o elemento é suficiente.
- Muitas vezes, precisamos saber qual é a sua posição na lista.
- Exemplo:

```
z = ["a", "b", "c", "d", "e"]
for indice in range(len(z)):
    if z[indice] == "c":
        print("Elemento encontrado no índice %d" % indice)
        break
else:
    print("Elemento não encontrado!")
```

Elemento encontrado no índice 2

Exercício

23. Faça um programa que mostra o menor valor dentro da lista $T = [1, 7, 2, 4]$.
24. As temperaturas de uma cidade foram armazenadas na lista $temperaturas = [-10, -8, 0, 1, 2, 5, -2, -4]$. Faça um programa que imprime a menor e a maior temperatura, assim como a média.
25. Neste exercício, você criará um programa que lê palavras do usuário até que o usuário entre com uma linha em branco. Após o usuário digitar uma linha em branco, seu programa deve exibir cada palavra digitada pelo usuário exatamente uma vez. As palavras devem ser exibidas na mesma ordem em que foram inseridas. Por exemplo, se o usuário inserir:

Exercício

25. first
second
first
third
second

A saída deve ser:

```
first
second
third
```

Exercícios

26. Faça um Programa que leia 20 números inteiros e armazene-os num vetor. Armazene os números pares no vetor **par** e os números **ímpares** no vetor ímpar. Imprima os três vetores.
27. Faça um programa que carregue uma lista com os modelos de cinco carros (exemplos: FUSCA, GOL, VECTRA etc). Carregue uma outra lista com o consumo desses carros, isto é, quantos quilômetros cada um desses carros faz com um litro de combustível. Calcule e mostre:
- a) O modelo do carro mais econômico;
 - b) Quantos litros de combustível cada um dos carros cadastrados consome para percorrer uma distância de 1000 quilômetros e quanto isto custará, considerando um que a gasolina custe R\$ 4,09 o litro.

Exercício

28. Crie um programa que leia números inteiros do usuário até que uma linha em branco seja inserida. Uma vez que todos os números inteiros tenham sido lidos, seu programa deve exibir todos os números negativos, seguidos por todos os zeros, seguidos por todos os números positivos. Dentro de cada grupo, os números devem ser exibidos na mesma ordem em que foram inseridos pelo usuário. Por exemplo, se o usuário inserir os valores 3, -4, 1, 0, -1, 0 e -2, seu programa deverá exibir três linhas:

-4, -1, -2

0, 0

3, 1

Exercício

29. Para ganhar o prêmio principal em uma determinada loteria, é preciso combinar todos os 6 números em seu bilhete com os 6 números entre 1 e 49 que são sorteados pelo organizador da loteria. Escreva um programa que gere uma seleção aleatória de 6 números para um bilhete de loteria. Assegure-se de que os 6 números selecionados não contenham duplicatas. Exiba os números gerados.

Exercício

30. O crivo de Eratóstenes é uma técnica que foi desenvolvida há mais de 2.000 anos atrás para encontrar facilmente todos os números primos entre 2 e algum limite. Uma descrição do algoritmo é a seguinte:

Anote todos os números de 0 até um limite

Descarte o 0 e o 1 porque eles não são primos

Defina ***p*** igual a 2

Enquanto ***p*** for menor que o limite **faça**

 Descarte todos os múltiplos de ***p*** (mas não o próprio ***p***)

 Defina ***p*** igual ao próximo número da lista que não foi descartado

Relate todos os números que não foram descartados como primos

Exercício

30. Este algoritmo está baseado no fato de ser relativamente fácil desconsiderar todos os números n em um pedaço de papel. Essa também é uma tarefa fácil para um computador - um loop for pode simular esse comportamento quando um terceiro parâmetro é fornecido para a função *range()*. Quando um número é descartado, sabemos que ele não é mais primo, mas ainda ocupa espaço no pedaço de papel, e ainda deve ser considerado ao computar números primos posteriores.

Exercício

30. Como resultado, você não deve descartar um número removendo-o da lista. Em vez disso, você deve descartar um número substituindo-o por 0. Em seguida, assim que o algoritmo for concluído, todos os valores diferentes de zero na lista serão primos.

Crie um programa em Python que use esse algoritmo para exibir todos os números primos entre 2 e um limite digitado pelo usuário. Se você implementar o algoritmo corretamente, você deve ser capaz de exibir todos os números primos menores que 1.000.000 em apenas alguns segundos.