INSTITUTO FEDERAL
São Paulo
Câmpus São Carlos

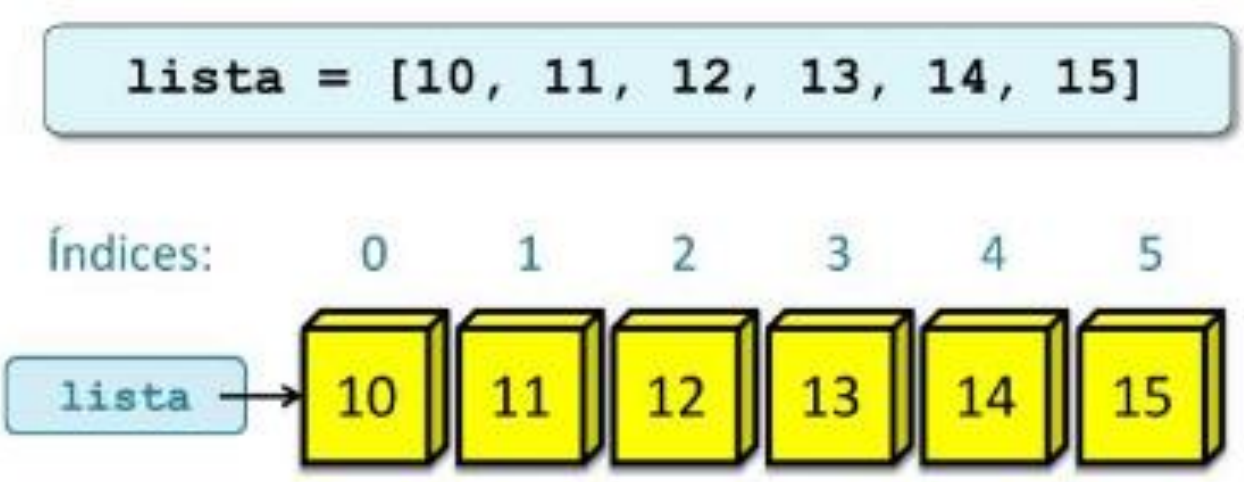
Listas
AP1S1 – Algoritmos e Programação

Curso Superior de Tecnologia em Análise e Desenvolvimento de Sistemas

INSTITUTO FEDERAL
São Paulo
Câmpus São Carlos

Listas

- Uma lista é uma estrutura de dados composta por itens organizados de forma linear, na qual cada um pode ser acessado a partir de um índice, que representa sua posição na coleção (iniciando em zero)



- Em Python, essa estrutura de dados pode armazenar, simultaneamente, objetos de diferentes tipos, como strings, números e até mesmo outras listas

Listas

- Em Python, uma **lista é representada como uma sequência de objetos** separados por vírgula e dentro de colchetes [].Exemplo:

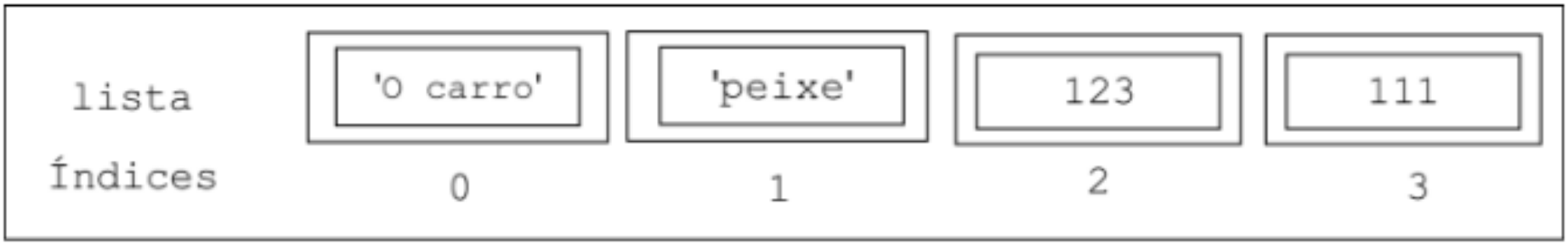
Lista = [1356, “João Cardoso”, “25/08/1978”, 3560.45]

- Uma lista vazia é representada por colchetes sem nenhum conteúdo → []

- Cada valor na lista é identificado por um índice. O valores que formam uma lista são chamados **elementos** ou **itens**.

Operadores de Listas

- A linguagem Python dispõe de vários métodos e operadores para auxiliar na manipulação de listas
- O primeiro e mais básico é o operador de acesso aos seus itens a partir dos índices



- É possível acessar cada um dos elementos utilizando a sintaxe

nome_da_variável_lista[índice]

Listas

- Um lista em uma outra lista é dita aninhada (*nested*) e a lista mais interna é chamada frequentemente de sublista (*sublist*)

```
vocabulario = ["iteracao", "selecao", "controle"]
numeros = [17, 123]
vazia = []
lista_mista = ["ola", 2.0, 5*2, [10, 20]]
print(lista_mista)
nova_lista = [numeros, vocabulario, vazia]
print(nova_lista)
outra_lista = ['O carro','peixe',123,111]
print(outra_lista)
outra_nova_lista = ['pedra',outra_lista]
print(outra_nova_lista)
```

Acesso aos elementos

- A sintaxe para acessar um elemento de uma lista é o operador de indexação ([] – não confundir com a lista vazia)
- A expressão dentro dos colchetes especifica o índice
- Lembrar que o índice do primeiro elemento é 0
- Qualquer expressão que tenha como resultado um número inteiro pode ser usada como índice
- Índices negativos indicarão elementos da direita para a esquerda ao invés de da esquerda para a direita, iniciando em -1

Acesso aos elementos

- Exemplos:

```
numeros = [17, 123, 87, 34, 66, 8398, 44]
print(numeros[2])
print(numeros[9-8])
print(numeros[-2])
print(numeros[len(numeros)-1])
```

- Resultados:

```
87
123
8398
44
```

Acesso aos elementos

- Exemplos:

```
vocabulario = ["iteracao", "selecao", "controle"]
numeros = [17, 123]
vazia = []
lista_mista = ["ola", 2.0, 5*2, [10, 20]]
nova_lista = [numeros, vocabulario, vazia]

print(lista_mista[0])    → ola
print(lista_mista[3])    → [10, 20]
print(nova_lista[1])     → ["iteracao", "selecao", "controle"]
print(nova_lista[1][2]) → controle
```

Comprimento de Listas

- O comprimento de uma lista, ou o número de itens que a compõem, pode ser obtido a partir da função **len()**

- Exemplo: o código abaixo indica que a variável nova_lista contém 2 elementos:

```
uma_lista = [3, 67, "gato", [56, 57, "cachorro"], [ ], 3.14, False]
print(len(uma_lista))
print(len(uma_lista[2]))
print(len(uma_lista[4]))
print(len(uma_lista[6])) → ?????
```

Pertinência em uma lista

- in e not in são operadores booleanos ou lógicos que testam a pertinência (*membership*) em uma sequência

- Exemplo:

```
frutas = ["maca", "laranja", "banana", "cereja"]

print("maca" in frutas)          → TRUE
print("pera" in frutas)          → FALSE
print("morango" not in frutas)   → TRUE
```

Concatenação

- O operador + concatena listas
- Exemplo:

```
frutas = ["maca", "laranja", "banana", "cereja"]
print([1, 2] + [3, 4])
print(frutas + [6, 7, 8, 9])
```

```
[1, 2, 3, 4]
['maca', 'laranja', 'banana', 'cereja', 6, 7, 8, 9]
```

```
lista_orig = [45,32,88]
lista_orig = lista_orig + ["cachorro"]
print(lista_orig)
```

```
[45, 32, 88, 'cachorro']
```

Append

- O método append acrescenta um novo item no final da lista

- Com append, a lista original é simplesmente modificada
  - É possível ver isto observando o id de lista_orig
  - O id da lista original é o mesmo antes e depois de executarmos append

- É possível acrescentar um item no final da lista usando o operador + de concatenação
  - É necessário ter cuidado!!! Com a concatenação uma nova lista é criada e o id da nova lista é diferente do id da lista original

A função id()

- Python possui uma função nativa (*build-in*) que recebe um objeto como argumento e retorna o seu id

- A função é convenientemente chamada de id e tem um único parâmetro, o objeto que você está interessado em descobrir o id
- O id é usualmente um número inteiro muito grande, pois corresponde a um endereço na memória
- Exemplo:

```
uma_lista = [4,5,6]
print(id(uma_lista))
```

Append

- Caso o elemento a ser adicionado seja uma lista, o append cria uma sublista

- Exemplo:

```
lista_orig = [45,32,88]
print(id(lista_orig))          → 46940448
lista_orig.append("gato")
lista_orig.append(["cachorro"])
print(lista_orig)              → [45,32,88,'gato',['cachorro']]
print(id(lista_orig))          → 46940448
lista_orig = lista_orig + ["sapo"]
print(lista_orig)              → [45,32,88,'gato',['cachorro'],'sapo']
print(id(lista_orig))          → 46926304
```

Extend

- O método Extend recebe como parâmetro um objeto iterável e vai adicionar elemento por elemento ao final da lista
- O id da lista permanece o mesmo

```
lista_orig = [45,32,88]
print(id(lista_orig))          → 47809240
lista_orig.extend(["gato"])
print(id(lista_orig))          → 47809240
lista_orig.extend("cachorro")
print(id(lista_orig))          → 47809240
print(lista_orig)
→[45, 32, 88, 'gato', 'c', 'a', 'c', 'h', 'o', 'r', 'r', 'o']
```

Append x Extend

- Exemplos:

```
lista1 = [1, 2, 3]
lista1.append(4)
print(lista1)
```
→ `[1, 2, 3, 4]`

```
lista2 = [1, 2, 3]
lista2.append([4, 5])
print(lista2)
```
→ `[1, 2, 3, [4, 5]]`

```
lista3 = [1, 2, 3]
lista3.extend([4, 5])
print(lista3)
```
→ `[1, 2, 3, 4, 5]`

Repetição

- O operador * repete os itens em uma lista um dado número de vezes
- Exemplos:

```
print([0] * 4)
print([1, 2, ["ola", "tudo bem?"]]*2)
```

- Resultados:

```
[0, 0, 0, 0]
[1, 2, ['ola', 'tudo bem?'], 1, 2, ['ola', 'tudo bem?']]
```

Observações Importantes

- Os operadores de concatenação e repetição criam novas listas (diferentes *ids*) a partir dos elementos das listas dadas
- Por exemplo, se você concatena uma lista de 2 itens com uma lista de 4 itens, você obterá uma nova lista com 6 itens e não uma lista com duas sublistas

```
L1=[1,2]      L2=[3,4,5,6]      Lista=L1+L2
print(Lista)  → [1,2,3,4,5,6]

L0=[]         L0.append(L1)     L0.append(L2)
print(L0)     → [[1,2],[3,4,5,6]]


L1=[1,2]      Lista=L1*3        print(Lista) → [1,2,1,2,1,2]
```

Fatias de Listas

- A operação de fatiar (*slice*) pode ser aplicada sobre listas
- O primeiro índice indica o ponto do início da fatia e o segundo índice é um a mais do final da fatia (o elemento com esse índice não faz parte da fatia)
- Teste:

```
uma_lista = ['a', 'b', 'c', 'd', 'e', 'f']
print(uma_lista[1:3])        → ['b', 'c']
print(uma_lista[:4])         → ['a', 'b', 'c', 'd']
print(uma_lista[3:])         → ['d', 'e', 'f']
print(uma_lista[:])          → ['a', 'b', 'c', 'd', 'e', 'f']
```

Listas são mutáveis

- Listas são **mutáveis** (*mutable*)
- Isto significa que podemos alterar um item em uma lista acessando-o diretamente como parte do comando de atribuição
- Usando o operador e indexação (colchetes) à esquerda de um comando de atribuição, podemos atualizar um dos itens de uma lista
- Teste: frutas = ["banana", "maca", "cereja"]

  print(frutas) → ['banana', 'maca', 'cereja']

  frutas[0] = "pera"

  frutas[-1] = "laranja"

  print(frutas) → ['pera', 'maca', 'laranja']

Listas são mutáveis

- Uma atribuição a um elemento de uma lista é chamada de **atribuição a um item** (*item assignment*)
- Combinando uma atribuição com o operador de fatiamento pode-se atualizar vários elementos de uma só vez
- Exemplo:
```
uma_lista = ['a', 'b', 'c', 'd', 'e', 'f']
uma_lista[1:3] = ['x', 'y']
print(uma_lista)    → ['a', 'x', 'y', 'd', 'e', 'f']
```
- Pode-se remover elementos de uma lista atribuindo a lista vazia a eles:
```
uma_lista[1:3] = [ ]
print(uma_lista)    → ['a', 'd', 'e', 'f']
```

Listas são mutáveis

- Python oferece uma maneira alternativa para remover elementos que é mais legível
- O comando del remove um elemento de uma lista usando a sua posição
- O comando del também manipula índices negativos e produz um erro de execução se o índice estiver fora do intervalo da lista
- Pode-se também usar uma fatia como argumento para del. Como é usual, fatias selecionam todos os elementos até, mas não incluindo, o segundo índice.

Listas são mutáveis

- Exemplos:

```
a = ['um', 'dois', 'três']
del a[1]
print(a) → ['um', 'três']

lista = ['a', 'b', 'c', 'd', 'e', 'f']
del lista[1:5]
print(lista) → ['a', 'f']
```

Listas são mutáveis

- Podemos inserir elementos em uma lista espremendo-os em uma fatia vazia na posição desejada ou por meio do método "insert". Exemplo:

```
uma_lista = ['a', 'd', 'f']
print(uma_lista)  →['a', 'd', 'f']
uma_lista[1:1] = ['b', 'c']
print(uma_lista)  →['a', 'b', 'c', 'd', 'f']
uma_lista[4:4] = ['e']
print(uma_lista)  →['a', 'b', 'c', 'd', 'e', 'f']
uma_lista.insert(0, 'z')
print(uma_lista)  →['z', 'a', 'b', 'c', 'd', 'e', 'f']
uma_lista.insert(3, 'x')
print(uma_lista) →['z', 'a', 'b', 'x', 'c', 'd', 'e', 'f']
```

Encontrando a posição de um elemento qualquer de uma lista

- O método index() é usada para encontrar a primeira posição de um elemento qualquer em uma lista
- Por exemplo, para encontrar a posição do elemento ‘x’ na lista L:

```
L = ['z', 'x', 'b', 'x', 'c', 'd', 'e', 'x']
print(L.index('x')) → 1
```

- O método count() é usado para determinar a quantidade de ocorrências de um elemento em uma lista

```
L = ['z', 'x', 'b', 'x', 'c', 'd', 'e', 'x']
print(L.count('x')) → 3
```

Criando listas de inteiros

- Listas que contém números inteiros consecutivos são bastante comuns.
- O operador **range** fornece uma maneira simples de criar uma lista de inteiros.
- Exemplos:

```
L=list(range(1, 11))
print(L) → [1,2,3,4,5,6,7,8,9,10]
```

- O operador **range** pega dois argumentos (dois inteiros) e devolve uma lista que contém todos os inteiros do primeiro até o segundo, incluindo o primeiro mas não incluindo o segundo!

Desempacotamento de listas em variáveis

- Números inteiros:

```
x, y, z = [1, 2, 3]
print(x)
print(y)
print(z)
```

- Cadeias de Caracteres:

```
n1,n2 = ("marcos", "joao")
print(n1, n2)
```

- Descartar valores → _ (underscore)

```
x,y,_ = [1,2,3]
print(x)
print(y)
```

Desempacotamento de sequências

- Descartar valores

```
notas = [9, 7, 8, 5, 10,20,30]
n1, *n2 = notas
print(n1)          → 9
print(n2)          → [7,8,5,10,20,30]
first, *_, last = notas
print(first)       → 9
print(last)        → 30
```

Percorrendo uma lista

- Utilizando o comando while:

```
notas = [9, 7, 8, 5, 10, 6]
soma=0
idx=0
while idx < len(notas):
    soma = soma + notas[idx]
    idx = idx + 1
media=soma/len(notas)
print("O valor da média das notas é: ", media)
```

Listas e o Laço for

- É possível **percorrermos uma lista** (*list traversal*) iterando através de itens ou iterando através de índices

- Exemplo:

```
frutas = ["pera", "laranja", "banana", "cereja"]
for uma_fruta in frutas:      #varredura por item
    print(uma_fruta)
```

- Leitura: para cada fruta (variável: uma_fruta) na lista de frutas (lista: frutas), escreva o nome da fruta (variável: uma_fruta)

Listas e o Laço for

- Outro exemplo: Cálculo da média de notas

```
notas = [9, 7, 8, 5, 10, 6]
soma=0
for elemento in notas:
    soma = soma + elemento
media=soma/len(notas)
print("O valor da média das notas é: ", media)
```

Listas e o Laço for

- Podemos também usar o índice para acessar os itens iterativamente

- A função range retorna uma sequência de inteiros

```
frutas = ["pera", "laranja", "banana", "cereja"]
for idx in range(len(frutas)):
                                #varredura por índice
    print(frutas[idx])
```

Listas e o Laço for

- Outro exemplo: Cálculo da média de notas

```
notas = [9, 7, 8, 5, 10, 6]
soma=0
for idx in range(len(notas)):
    soma = soma + notas[idx]
media=soma/len(notas)
print("O valor da média das notas é: ", media)
```

Listas e o Laço for

- Outro exemplo: Exiba todos os números entre 1 e 5 elevados ao quadrado utilizando iteração por índice

```
numeros = []
for n in range(1,6):
    numeros.append(n)    #monta a lista
print(numeros)
for i in range(len(numeros)):
    numeros[i] = numeros[i]**2
print(numeros)
```

Exercícios

1. Crie um programa que leia inicialmente uma sequência de N notas de alunos fornecidas pelo usuário e ao final mostre a sequência e sua média aritmética.
2. Crie um programa que leia inicialmente uma sequência de N alturas de pessoas e ao final mostre a sequência, a maior altura e sua posição na sequência.
3. Crie um programa que leia inicialmente uma sequência de N números inteiros e ao seu final mostre a sequência original, a soma de seus elementos que forem pares e a multiplicação dos elementos que forem impares.
4. Crie um programa que leia inicialmente uma sequencia de N números inteiros e mostre ao final 2 listas: uma sem repetição e outra dos elementos repetidos.

Exercícios

5. Crie um programa que leia inicialmente duas sequências de N elementos cada uma e ao final mostre as duas sequências originais e a sequência resultante da soma de seus elementos. Exemplo:

   a=[5, 9, 0] b=[12, 34, 101] soma=[17, 43, 101]

6. Crie um programa que dada uma sequência de N elementos fornecidos pelo usuário mostre a sequência original e a sequência elevada ao cubo.
7. Crie um programa que leia inicialmente uma sequência de N números inteiros fornecidos pelo usuário e mostre ao final da leitura a sequência original e a sequência invertida.
8. Crie um programa que leia inicialmente uma sequência de N elementos inteiros positivos fornecidos pelo usuário e substitua seus elementos de valor ímpar por -1 e os pares por +1. Ao final imprima a sequência original e a sequência alterada.

Exercícios

9. Crie um programa que leia inicialmente duas sequências de N elementos cada uma. Ao final mostre as duas sequências originais e diga se as listas são iguais ou não. *Duas listas são consideradas iguais quando possuem os mesmos elementos na mesma ordem.*
10. Crie um programa que leia inicialmente duas sequências de N elementos cada uma. Ao final mostre as duas sequências originais e diga se as listas possuem os mesmos elementos ou não. *Neste caso, as duas listas devem possuir os mesmos elementos em qualquer ordem.*
11. Faça uma função que receba uma lista de números armazenados de forma crescente (faça a consistência) , e dois valores (limite inferior e limite superior), e exiba a sublista cujos elementos são maiores ou iguais ao limite inferior e menores ou iguais ao limite superior. Exemplo: lista inicial=[12,14,15,16,18,20,24,26,28,32,34,38] limite inferior=13 limite superior = 26 lista exibida: [14,15,16,18,20,24,26]

Exercícios

12. Faça um programa que percorre uma lista com o seguinte formato: [['Brasil', 'Italia', [10, 9]], ['Brasil', 'Espanha', [5, 7]], ['Italia', 'Espanha', [7,8]]]. Essa lista indica o número de faltas que cada time fez em cada jogo. Na lista acima, no jogo entre Brasil e Itália, o Brasil fez 10 faltas e a Itália fez 9. O programa deve imprimir na tela: a) o total de faltas do campeonato b) o time que fez mais faltas c) o time que fez menos faltas.

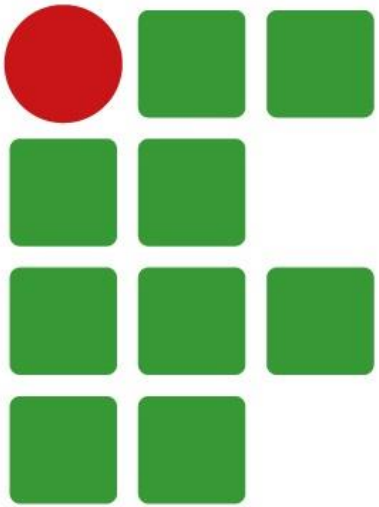
13. Foram anotadas as idades e alturas dos alunos de uma turma e armazenados em uma lista cujos elementos são sublistas com dois elementos: o primeiro é a idade do aluno e o segundo a sua altura. Mostre quantos alunos com mais de 13 anos possuem altura inferior à média de altura desses alunos.

Exercícios

14. O Zodíaco chinês é composto por animais com ciclo de 12 anos. Uma maneira simplificada de identificá-lo é verificando-se apenas o ano de seu nascimento do seguinte modo:

ano do nascimento % 12	Signo
0	Macaco
1	Galo
2	Cão
3	Porco
4	Rato
5	Boi
6	Tigre
7	Coelho
8	Dragão
9	Serpente
10	Cavalo
11	Carneiro

Faça um programa que tenha a lista de signos e outra lista que receba o nome e o ano de nascimento dos membros de sua família. Ao final mostre as duas listas e signo correspondente de cada membro.



INSTITUTO FEDERAL
São Paulo
Câmpus São Carlos