

PROGRAMAÇÃO 1

Aula 3 – Listas

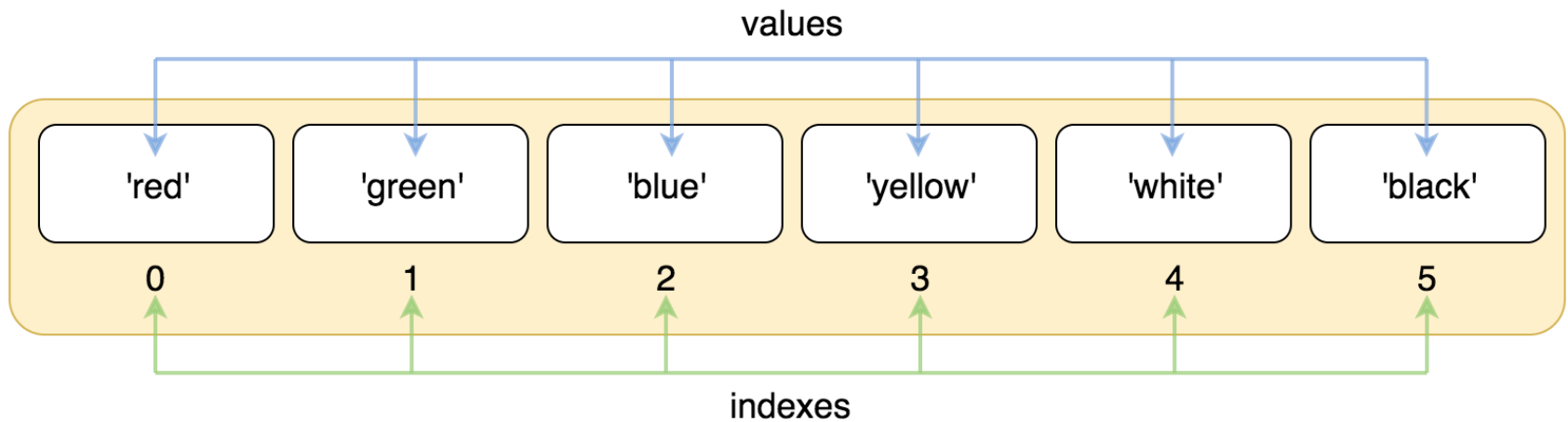
Prof. Emanuel Barreiros



Listas

- Listas são coleções de dados
- Elementos são dispostos em uma ordem particular
- Seus elementos não precisam estar relacionados de nenhuma forma
- Como listas geralmente armazenam vários itens, coloque seu nome no plural
- Em Python, colchetes indicam uma lista: []
- Elementos em uma lista aparecem separados por vírgula:
 - Tente: `campi = ['Garanhuns', 'Salgueiro', 'Serra Talhada', 'Arcoverde', 'Caruaru']`

Listas



Acessando elementos em listas

- Utilize o operador [] para acessar elementos pelo seu índice:

```
>>> campi = ['Garanhuns', 'Sagueiro', 'Serra  
Talhada', 'Caruaru']  
>>> campi[1]  
'Sagueiro'
```

- Serve para atribuição também:

```
>>> campi[1] = 'Salgueiro'  
>>> campi[1]  
'Salgueiro'
```

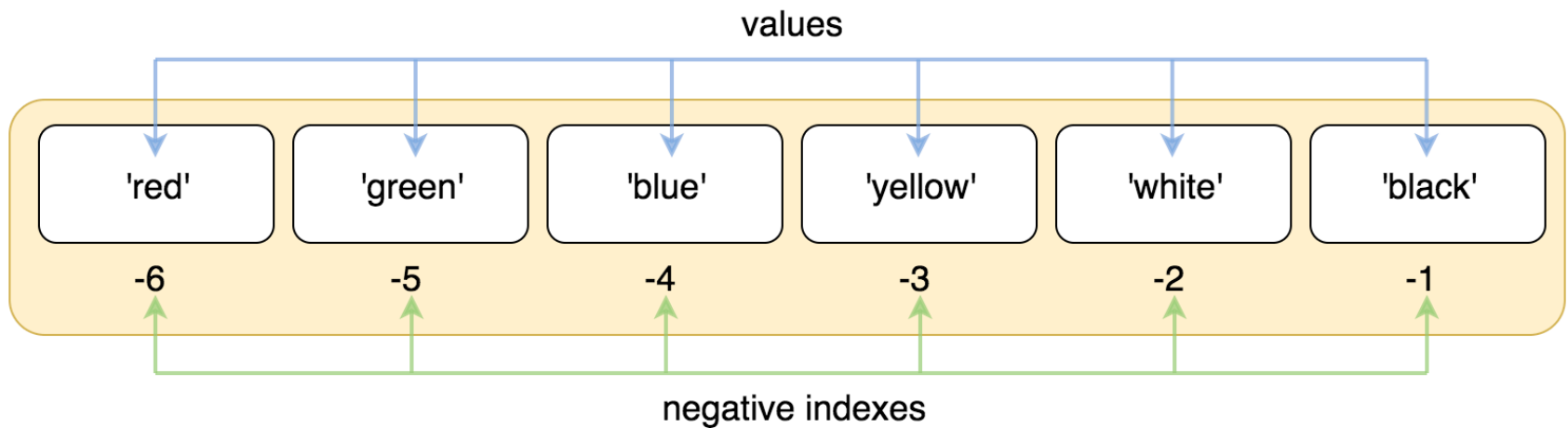
Listas

- Os índices começam em 0
- Python permite que utilizemos índices negativos, o que permite acessar itens do fim da lista:

```
>>> campi[-1]  
'Caruaru'
```

- -1 acessa o último item da lista, -2 o penúltimo e assim por diante

Listas



Manipulando a lista

- Podemos modificar itens a partir do índice
- Podemos também adicionar elementos ao final da lista:

```
>>> campi.append( 'Salgueiro' )
```

- Podemos criar listar dinamicamente:

```
>>> alunos = [ ]
```

```
>>> alunos.append( 'Bruno' )
```

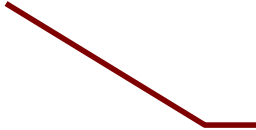
```
>>> alunos.append( 'Luis' )
```

```
...
```

Inserir elementos em posições específicas

- Você pode inserir elementos em posições específicas

```
>>> alunos.insert(0, 'Cláudio')
```



Inserir um elemento na primeira posição da lista.

Deletando elementos

- Se você sabe qual a posição do elemento, podemos usar a instrução `del`:

```
>>> del campi[0]
```

- Também podemos usar a função `pop()`:

```
>>> campus = campi.pop()
```

Remove o último elemento da lista e atribui à variável `campus`.

Deletando itens

- O método pop pode ser usado também para remover itens de qualquer posição:

```
>>> campus = campi.pop(0)
```

- Mas nem sempre você vai saber o índice do elemento a ser removido
- Nesses casos, você pode remover utilizando o valor a ser removido:

```
>>> campi.remove('Serra Talhada')
```

Exercícios

1. Crie um programa que, a partir de uma lista de nomes de pessoas, imprima mensagens 'Olá <NOME>, prazer em conhece-lo.' para cada nome na lista. Execute um print para cada nome na lista.
2. Crie uma lista com nomes e sobrenomes, e.g. ['Emanoel', 'Barreiros', 'Marcos', 'Silva', 'Carlos', 'Pereira']. Utilizando esta lista, obtenha cada nome e seu sobrenome, e imprima o nome completo em uma linha. Assuma que todos os nomes são compostos de um nome e um sobrenome, logo a lista deve ter uma quantidade par de nomes.

Ordenando listas

- Você pode precisar ordenar os elementos de uma lista
- O método `sort()` realiza a ordenação *in place* da lista:

```
>>> campi = ['Garanhuns', 'Salgueiro', 'Serra  
Talhada', 'Arcoverde', 'Caruaru']  
>>> campi.sort()  
>>> campi  
['Arcoverde', 'Caruaru', 'Garanhuns', 'Salgueiro',  
'Serra Talhada']
```

Ordenando listas

- Após a execução do método `sort ()` a lista será ordenada permanentemente
- Se quisermos ordenar a lista mas sem efeito colateral na lista em si, podemos usar a função `sorted ()`:

```
>>> campi = ['Garanhuns', 'Saqueiro', 'Serra Talhada',  
             'Arcoverde', 'Caruaru']
```

```
>>> sorted(campi)  
['Arcoverde', 'Caruaru', 'Garanhuns', 'Salgueiro',  
 'Serra Talhada']
```

```
>>> campi  
['Garanhuns', 'Saqueiro', 'Serra Talhada', 'Arcoverde',  
 'Caruaru']
```

Invertendo uma lista

- Utilizando-se o método `reverse()` podemos inverter a ordem atual de uma lista:

```
>>> campi = ['Garanhuns', 'Sagueiro',  
             'Serra Talhada', 'Caruaru']  
>>> campi.reverse()  
>>> campi  
['Caruaru', 'Serra Talhada', 'Salgueiro',  
 'Garanhuns']
```

Tamanho de uma lista

- O tamanho de uma lista pode ser obtido a partir da função `len()`:

```
>>> campi = ['Garanhuns', 'Sagueiro',  
             'Serra Talhada', 'Caruaru']
```

```
>>> len(campi)
```

```
4
```

TRABALHANDO COM LISTAS

Percorrendo uma lista com um laço

- O laço for é particularmente útil para isso. Exemplo 1:

```
1  campi = ['Arcoverde', 'Garanhuns', 'Salgueiro', 'Serra Talhada', 'Caruaru']
2  for campus in campi:
3      print(campus)
```

- A cada iteração o Python atribui um elemento diferente da lista à variável `campus`
- Diferente de outras linguagens, o laço `for` em Python só funciona sobre *iteráveis*
- Como o laço `for` define um bloco, observe sempre a indentação!

Criando listas numéricas

- A função `range(. . .)` facilita a criação de listas numéricas. Exemplo:

```
1  for valor in range(5):  
2      print(valor)
```

- Saída esperada:

0
1
2
3
4

Criando listas numéricas

- Python inicia a lista com o valor 0 e vai até o valor informado -1, por isso só imprime até o valor 4
- Você também pode informar um valor inicial:

```
4   for valor in range(1, 5):  
5       print(valor)
```

- Saída esperada:

1
2
3
4

Criando listas numéricas

- Também é possível informar o passo do incremento para geração da lista:

```
7   for valor in range(1, 10, 2):  
8       print(valor)
```

- Será gerada uma sequência de números iniciando em 1, com incremento 2, com valor máximo menor que 10. Saída esperada:

1
3
5
7
9

Detalhes sobre *range*

- Range na realidade não é uma função, é um gerador de sequências, gerando o próximo valor quando necessário
- Como vantagem, não importa a quantidade de elementos gerados, ela sempre vai consumir a mesma quantidade de memória, pois não gera todos os números da sequência de antemão

Listas a partir do *range*

- Usando a função `list(...)` você pode criar uma lista a partir de um gerador *range*:

```
10  numeros = list(range(1, 5))  
11  print(numeros)
```

- Saída esperada: `[1, 2, 3, 4]`

Exercício

3. Escreva um programa que crie uma lista com os quadrados dos números de 1 a 10.

Operadores *in* e *not in*

- Servem para checar a existência ou não de um valor na lista
- O operador `in` checa se um elemento está na lista:

```
>>> campi = ['Garanhuns', 'Salgueiro',  
             'Serra Talhada', 'Caruaru']
```

```
>>> 'Caruaru' in campi
```

```
True
```

O operador `not in` verifica se o elemento não está na lista:

```
>>> 'Caruaru' not in campi
```

```
False
```


COMPREENSÃO DE LISTAS E FATIAMENTO

Compreensão de listas

- Permite a geração/tratamento
- Usa muito menos código e às vezes até parece mágica...

```
13 quadrados = [valor**2 for valor in range(1, 11)]  
14 print(quadrados)
```

- Não se preocupe tanto com isso agora, mas saiba que existe e não se desespere quando encontrar alguma por aí...

“Fatiando” listas

- Podemos recortar listas de forma bastante flexível em Python
- A forma mais simples prevê que você informe um índice inicial e onde a fatia deve parar (limite exclusivo)
- Sintaxe do comando dessa forma:

```
16 jogadores = ['severino', 'inacio', 'astolfo', 'raimundo', 'bartolomeu']  
17 print(jogadores[0:3])
```

- Saída esperada: ['severino', 'inacio', 'astolfo']

Outros casos e detalhes de indexação

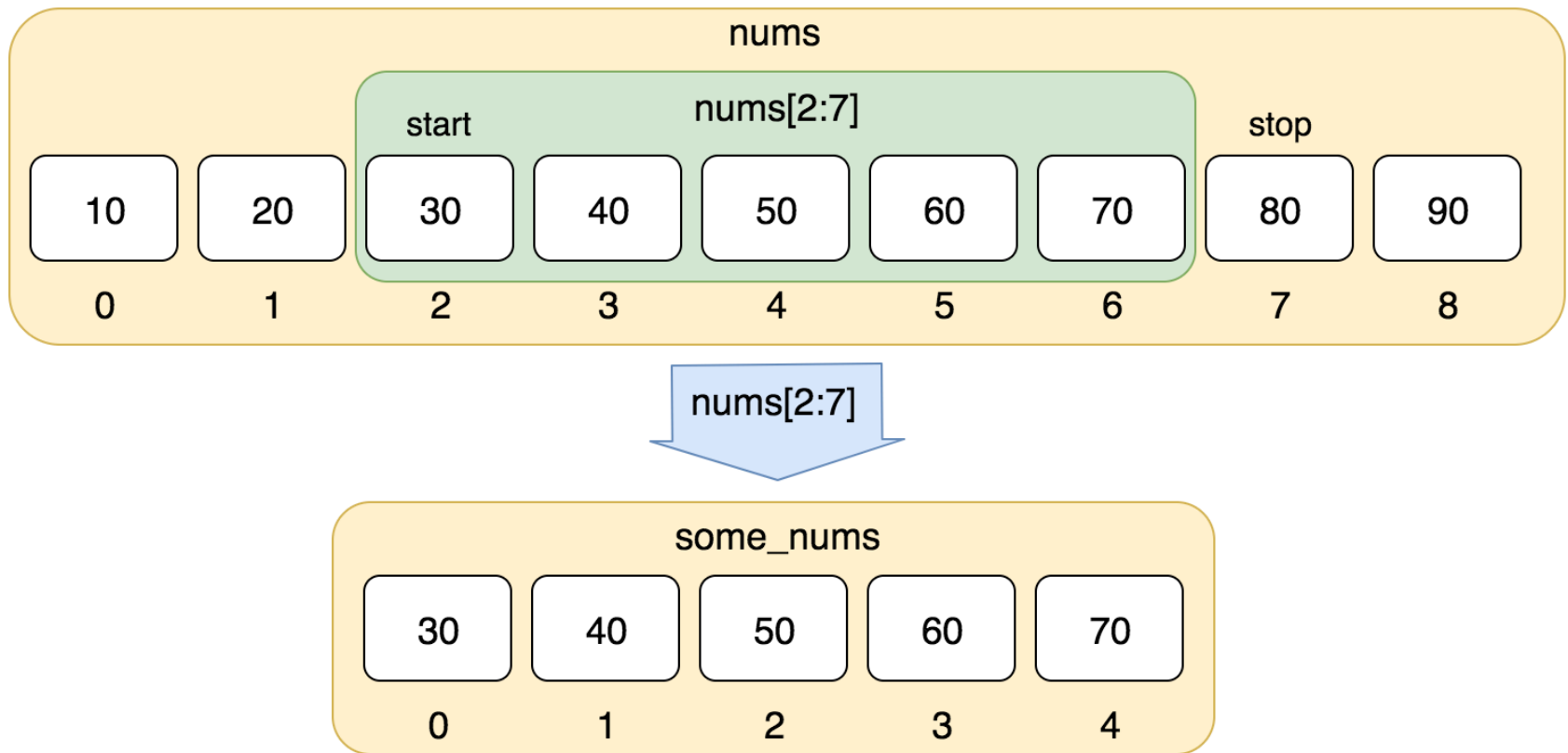
Comando	Significado
<code>l[inicio:fim]</code>	Itens de índice <code>inicio</code> até <code>fim-1</code>
<code>l[inicio:]</code>	Itens do índice <code>inicio</code> até o fim da lista
<code>l[:fim]</code>	Itens do início da lista até <code>fim-1</code>
<code>l[:]</code>	Uma cópia de toda a lista
<code>l[-1]</code>	Último elemento da lista
<code>l[-2:]</code>	Os dois últimos elementos da lista
<code>l[:-2]</code>	Toda a lista com exceção dos dois últimos itens

Exemplos

- Assumindo a lista: `l = [13, 14, 15, 16, 17, 18, 19]`

Comando	Resultado
<code>l[2:6]</code>	<code>[15, 16, 17, 18]</code>
<code>l[5:]</code>	<code>[18, 19]</code>
<code>l[:3]</code>	<code>[13, 14, 15]</code>
<code>l[:]</code>	<code>[13, 14, 15, 16, 17, 18, 19]</code>
<code>l[-1]</code>	<code>[19]</code>
<code>l[-2:]</code>	<code>[18, 19]</code>
<code>l[:-2]</code>	<code>[13, 14, 15, 16, 17]</code>

Exemplo visual



Utilizando o *step*

- Você pode determinar o passo da seleção

Comando	Significado
<code>l[inicio:fim:passo]</code>	Itens de índice <code>inicio</code> até <code>fim-1</code>
<code>l[inicio::passo]</code>	Itens do índice <code>inicio</code> até o fim da lista
<code>l[:fim:passo]</code>	Itens do início da lista até <code>fim-1</code> , a cada passo itens
<code>l>::passo]</code>	Uma cópia de toda a lista, mas a cada passo itens

- Quando `inicio` é omitido, o Python assume o valor 0 pra ele
- Quando o `fim` é omitido, o Python assume o tamanho da lista pra ele
- Quando `passo` é omitido, o Python assume o valor 1 pra ele

Exemplos

- Assumindo a lista: `l = [13, 14, 15, 16, 17, 18, 19]`

Comando	Resultado
<code>l[1:-1:2]</code>	<code>[14, 16, 18]</code>
<code>l[1::3]</code>	<code>[14, 17]</code>
<code>l[:-3:2]</code>	<code>[13, 15]</code>
<code>l[::2]</code>	<code>[13, 15, 17, 19]</code>

Utilizando o *step*

- O valor do passo pode ser *negativo*:

Comando	Significado
<code>l[:: -1]</code>	Toda a lista em ordem reversa
<code>l[1 :: -1]</code>	Os primeiros dois itens da lista em ordem reversa
<code>l[: -3 :: -1]</code>	Os dois últimos itens da lista em ordem reversa
<code>l[-3 :: -1]</code>	Toda a lista com exceção dos dois últimos itens, em ordem reversa

- O significado de início e fim invertem quando usa-se o passo negativo, pois a contagem agora é no sentido inverso
- O valor padrão para *início* passa a ser o último elemento da lista
- O valor padrão para *fim* passa a ser o início da lista (inclusive)
- Início tem que ser um índice posterior ao fim, já que a contagem é reversa

Exemplos

- Assumindo a lista: `l = [13, 14, 15, 16, 17, 18, 19]`

Comando	Resultado
<code>l[::-1]</code>	<code>[19, 18, 17, 16, 15, 14, 13]</code>
<code>l[1::-1]</code>	<code>[14, 13]</code>
<code>l[:-3:-1]</code>	<code>[19, 18]</code>
<code>l[-3::-1]</code>	<code>[17, 16, 15, 14, 13]</code>
<code>l[-2:2:-1]</code>	<code>[18, 17, 16]</code>
<code>l[2:-2:-1]</code>	<code>[]</code>
<code>l[3:5:-1]</code>	<code>[]</code>

Exercício

4. Faça um programa que, utilizando um laço for, imprima os três primeiros itens de uma lista
5. Faça um programa que, utilizando um laço for, imprima os 5 últimos elementos de uma lista, na ordem em que aparecem e em ordem inversa.

Tuplas

- Enquanto listas são coleções mutáveis, tuplas são coleções imutáveis de dados
- Uma vez criadas, seu conteúdo não pode ser alterado
- São definidas de forma semelhante a listas, mas usando parênteses em vez dos colchetes
- Fora isso, funcionam da mesma forma que lista, inclusive podem ser iteradas utilizando um laço for