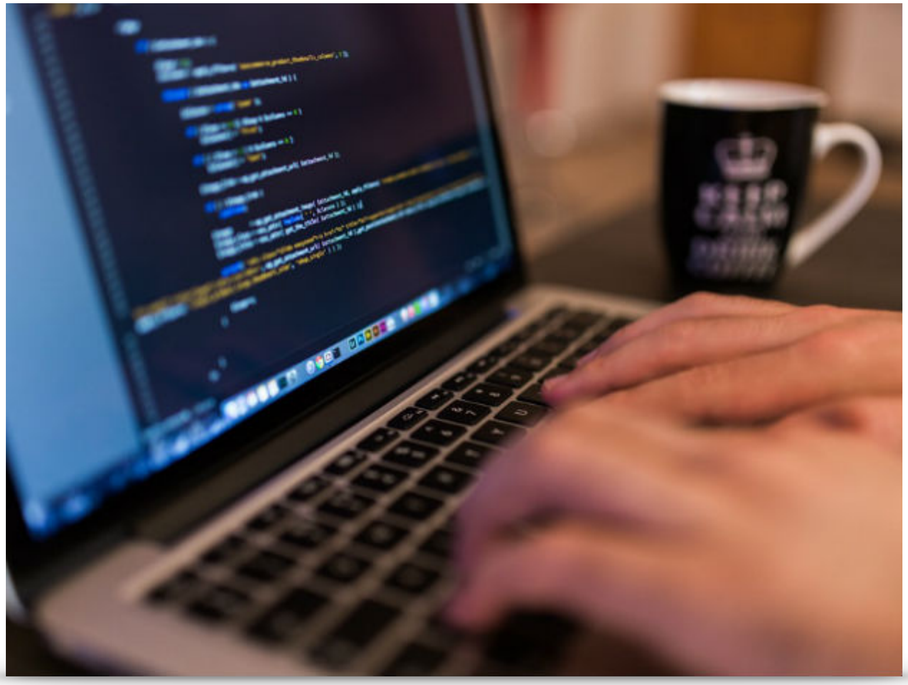
Profa. Dra. Raquel C. de Melo-Minardi
Departamento de Ciência da Computação
Instituto de Ciências Exatas
Universidade Federal de Minas Gerais



MÓDULO 2 – PROGRAMAÇÃO
Sequências – Listas

TIPOS DE VARIÁVEIS OU ESTRUTURAS DE DADOS EM PYTHON

- As **sequências** podem ser de dois tipos:
  - **sequências imutáveis**: são objetos ordenados e finitos
    - ***strings***: cadeias de caracteres
    - ***tuples***: dois ou mais elementos de qualquer tipo dentro de parênteses e separados por vírgula
  - **sequências mutáveis**
    - ***lists***: conhecidas em outras linguagens como vetores ou arranjos
    - ***sets***: coleções não ordenadas e não indexadas escritas entre chaves

LISTAS

- Durante o desenvolvimento de software, independentemente de plataforma e linguagem, é comum a necessidade de lidar com **listas**
  - Por exemplo, elas podem ser empregadas para armazenar nucleotídeos de uma sequência de DNA ou aminoácidos de uma proteína
- 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**)

LISTAS

- Durante o desenvolvimento de software, independentemente de plataforma e linguagem, é comum a necessidade de lidar com **listas**
  - Por exemplo, elas podem ser empregadas para armazenar nucleotídeos de uma sequência de DNA ou aminoácidos de uma proteína
- 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**)

```
proteinas = ['1A6M', '2MM1', '1CHO', '2PTI', '1A6N']
tamanhos = [153, 155, 233, 245, 150]
heterogeneo = ['1A6M', 153, '2MM1', 155, '1CHO', 233, '1A6N', 245]
```

- Como exemplificado na terceira linha, uma em Python pode ter diversos tipos de dados, ou seja, é uma variável composta heterogênea

CRIANDO LISTAS

- Em Python, uma **lista** é representada como uma **sequência de objetos separados por vírgula** e **dentro de colchetes** `[]`, assim, uma lista vazia, por exemplo, pode ser representada por colchetes sem nenhum conteúdo:

```
proteina = []
proteina = ['ALA', 'LYS', 'GLY', 'GLU', 'ALA']
proteina1 = ['A', 'K', 'G', 'E', 'A']
proteina2 = ['A', 'K', 'G', 'E', 'A']
uniao = [proteina1, proteina2, 'K', 'G', 'E']
```

ACESSANDO ITENS DE LISTAS

- Listas em Python sempre iniciam a indexação por “0”. Veja alguns exemplos de como elas podem ser acessados diretamente

```
proteina = ['ALA', 'CYS', 'ASP', 'GLU', 'HIS']
tamanhos = [153, 155, 233, 245, 150]
```

```
print(proteina[0]) # Imprime 'ALA'
print(proteina[1]) # Imprime 'CYS'
print(tamanhos[0])  # Imprime 153
```

TAMANHO DE UMA LISTA

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

```
proteina = ['ALA', 'CYS', 'ASP', 'GLU', 'HIS']
print(len(proteina)) # Imprime 5
```

SOMA E MULTIPLICAÇÃO (CONCATENAÇÃO) DE LISTAS

- É possível **concatenar** listas por meio do operador de adição **+** e **multiplicá-las** por um **inteiro**, o que gerará várias cópias dos seus itens

- O código a seguir traz alguns exemplos dessas operações:

```
proteina1 = ['A', 'K', 'G', 'E', 'A']
proteina2 = ['A', 'K', 'G', 'E', 'A']
concatenacao = proteina1 + proteina2
print(concatenacao)
# imprime ['A', 'K', 'G', 'E', 'A', 'A', 'K', 'G', 'E', 'A’]
multiplicacao = proteina1 * 2
print(multiplicacao)
# imprime ['A', 'K', 'G', 'E', 'A', 'A', 'K', 'G', 'E', 'A']
```

VERIFICAÇÃO DE PERTENCIMENTO EM LISTAS

- Em alguns casos, é preciso verificar se um determinado valor está contido em uma lista
  - Utilizamos o operador `in`, que indicará `True` se objeto pertencer ao conjunto, e `False` caso contrário:

```
print('A' in proteina) # Imprime True
```

VALORES MÍNIMOS, MÁXIMOS E SOMA EM LISTAS

- O Python oferece as funções `min()`, `max()` e `sum()`, através das quais é possível encontrar, respectivamente, o menor valor, o maior valor ou ainda realizar a soma de todos os elementos de uma lista

MÉTODOS DE LISTAS

- Até então apresentamos operadores e funções que recebem uma lista como argumento, retornam um resultado, mas não efetuam alterações na sua estrutura

- A partir de agora apresentaremos métodos pertencentes ao tipo **lista**

INCLUSÃO DE ITENS NO FIM DA LISTA

- O `append()` **adiciona** um novo elemento no **final** da lista

	[0]	[1]	[2]	[3]	[4]	[5]	[6]	[7]	[8]
Antes do *append*	4	5	3	2	7	8	1	0	
Após o *append(9)*	4	5	3	2	7	8	1	0	**9**

```
lista = [4, 5, 3, 2, 7, 8, 1, 0]
lista.append(9)
```

INCLUSÃO DE ITENS EM LOCAIS ARBITRÁRIOS DA LISTA

- Para inserir itens em uma **posição específica**, utilizamos o método `insert()` que, além do elemento a ser inserido, recebe também o índice que ele deve assumir

	[0]	[1]	[2]	[3]	[4]	[5]	[6]	[7]	[8]
Antes do *insert*	4	5	3	2	7	8	1	0	
Após o *insert(6, 9)*	4	5	3	2	7	8	**9**	1	0

```
lista = [4, 5, 3, 2, 7, 8, 1, 0]
lista.insert(6, 9)
```

REMOÇÃO DE ITENS DE UMA LISTA

- O método `pop()` remove o **último** item da **lista** e o retorna como resultado da operação

	[0]	[1]	[2]	[3]	[4]	[5]	[6]	[7]
Antes do *pop*	4	5	3	2	7	8	1	**0**
Após o *pop*	4	5	3	2	7	8	1	

```
lista = [4, 5, 3, 2, 7, 8, 1, 0]
lista.pop()
```

REMOÇÃO DE ITENS DE UMA LISTA

- O método `pop()` remove o **último** item da **lista** e o retorna como resultado da operação
  - Caso seja necessário remover um **índice específico**, basta **informá-lo como argumento**

	[0]	[1]	[2]	[3]	[4]	[5]	[6]	[7]
Antes do *pop*	4	5	3	**2**	7	8	1	0
Após o *pop*	4	5	3	7	8	1	0	

```
lista = [4, 5, 3, 2, 7, 8, 1, 0]
lista.pop(3)
```

REMOÇÃO DE ITENS DE UMA LISTA PELO VALOR

- Há situações em que o índice de elementos que se deseja remover é desconhecido e desejamos remover o item a partir do seu valor e para tal usamos o método `remove(item)`
  - Ele removerá o primeiro item com o determinado valor

	[0]	[1]	[2]	[3]	[4]	[5]	[6]	[7]
Antes do *remove*	4	5	3	**2**	7	8	1	0
Após o *remove*	4	5	3	7	8	1	0	

```
lista = [4, 5, 3, 2, 7, 8, 1, 0]
lista.remove(2)
```

ORDENAÇÃO DE ITENS DE UMA LISTA

- Há também funções prontas para **inverter** e **ordenar** arranjos:
  - `reverse()`: inverte um arranjo

```
lista = [4, 5, 3, 2, 7, 8, 1, 0, 11]
lista.reverse()
print(lista)# Imprime 11, 0, 1, 8, 7, 2, 3, 5, 4
```

  - `sort()`: ordena um arranjo em ordem crescente

```
lista = [4, 5, 3, 2, 7, 8, 1, 0, 11]
lista.sort()
print(lista)# Imprime 0, 1, 11, 2, 3, 4, 5, 7, 8
```

TAMANHO DE UMA LISTA

- O método `count()` retorna o número de ocorrências de determinado objeto, passado como parâmetro, em uma lista

```
lista = [4, 5, 3, 2, 7, 8, 1, 0, 11]
print(lista.count(4)) # Imprime 1
```

TUPLAS OU LISTAS? VALE A PENA USAR TUPLAS?

TUPLAS OU LISTAS? VALE A PENA USAR TUPLAS?

- Tuplas são **mais rápidas** que listas
  - Se você precisa apenas definir um conjunto de **valores constantes** e depois percorrê-lo e consultá-lo de alguma forma, opte por tuplas

TUPLAS OU LISTAS? VALE A PENA USAR TUPLAS?

- Tuplas são **mais rápidas** que listas
  - Se você precisa apenas definir um conjunto de **valores constantes** e depois percorrê-lo e consultá-lo de alguma forma, opte por tuplas
  - *“Tupla congela uma lista e lista descongela uma tupla”*