Laboratório de Programação 1

Aula 08

Mário Hozano
professor@hozano.com

Ciência da Computação
UFAL - Arapiraca

Relembrando a aula anterior...

- Para que serve o comando *for*?
- O que são sequências ?
- Qual a sintaxe do comando *for*?
- Para que serve a função *range* ?
- Como utilizar a função *range* ?
- Quais são as diferenças entre os comandos *for* e *while* ?

Roteiro da aula

- *Strings*
- Acessando os itens de uma *String*
- A função *len()*
- Índices negativos
- *Substrings*
- Métodos de *Strings*
- O operador *in* e os operadores relacionais

Strings

- Como vimos, uma *string* é uma sequência de caracteres
- Os itens de uma sequência podem ser acessados através de índices que indicam o item a ser recuperado
- Os índices devem expressos entre colchetes logo a pós a sequência a ser acessada
- O exemplo a seguir retornará o item com índice 2 da string “banana”

```
>>> seq = “banana”
>>> seq[2]
'n'
```

Strings

- Como vimos, uma *string* é uma sequência de caracteres
- Os itens de uma sequência podem ser acessados através de índices que indicam o item a ser recuperado
- Os índices devem expressos entre colchetes logo a pós a sequência a ser acessada
- O exemplo a seguir retornará o item com índice 2 da string “banana”

```
>>> seq = “banana”
>>> seq[2]
'n'
```

Por quê o item recuperado tem valor 'n' e não 'a' ?

Strings - Índices

- Em Linguagens de Programação, é comum que os items de sequências sejam iniciados pelo índice 0 (zero)
- Neste caso, a primeira letra de uma *string* possui índice 0, a segunda possui índice 1, a terceira índice 2...

```
>>> seq = “banana”
>>> seq[0]
'b'
>>> seq[1]
'a'
>>> seq[2]
'n'
```

Strings - Índices

- Em Linguagens de Programação, é comum que os items de sequências sejam iniciados pelo índice 0 (zero)
- Neste caso, a primeira letra de uma *string* possui índice 0, a segunda possui índice 1, a terceira índice 2...

```
>>> seq = “banana”
>>> seq[0]
'b'
>>> seq[1]
'a'
>>> seq[2]
'n'
```

O que acontece se tentarmos acessar o item com índice 6 ?

Strings - Índices

```
>>> seq = “banana”
>>> seq[6]
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
IndexError: string index out of range
```

O erro indica que o índice dado (6) está for a da faixa de valores possíveis (0..5)

Strings - Índices

```
>>> seq = “banana”
>>> seq[6]
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
IndexError: string index out of range
```

O erro indica que o índice dado (6) está for a da faixa de valores possíveis (0..5)

Como saber o tamanho de uma *string* ?

Strings – A função *len()*

- Muitas vezes é necessário saber o tamanho de uma sequência para evitar acessos a itens com índices fora da faixa
- Python provê a função *len()* para saber o tamanho de uma sequência
- Esta função retorna o tamanho de uma sequência dada

```
>>> seq = “banana”
>>> len(seq)
6
>>> len([12, 22, 32])
3
```

Strings – A função *len()*

- Assim, o último item de uma sequência não vazia é o item com índice igual ao tamanho da sequência - 1

```
>>> seq = “UFAL”
>>> tamanho =  len(seq)
>>> seq[tamanho - 1]
'L'
```

Strings – Índices Negativos

- Python, também permite que sejam indicados índices com valores negativos

- Neste caso, os itens percorrem a sequência de trás pra frente, onde o índice -1 representa o último item, o índice -2 o penúltimo e assim sucessivamente

```
>>> seq = “UFAL”
>>> seq[-1]
'L'
>>> seq[-2]
'A'
```

Strings – Substrings

- Um segmento de uma *string* é considerado como uma fatia ou uma *substring*
- Uma faixa de índices pode ser indicada com a sintaxe *[m:n]* que representa os índices de *m* à *n*, incluindo o primeiro e excluindo o último
- O caractere (:) é conhecido como operador *slice*

```
>>> seq = “universidade”
>>> seq[3:6]
'ver'
>>> seq[3:8]
'versi'
```

Strings – Substrings

- Em uma faixa de índices do tipo *[m:n]*, caso *m* seja omitido, a faixa irá considerar o início da *string* dada
- Por outro lado, caso *n* seja omitido, a faixa irá considerar o fim da *string* dada

```
>>> seq = “universidade”
>>> seq[7:]
'idade'
>>> seq[:4]
'univ'
```

Strings – Substrings

- Em uma faixa de índices do tipo *[m:n]*, caso *m* seja omitido, a faixa irá considerar o início da *string* dada
- Por outro lado, caso *n* seja omitido, a faixa irá considerar o fim da *string* dada

```
>>> seq = “universidade”
>>> seq[7:]
'idade'
>>> seq[:4]
'univ'
```

Qual seria o retorno do comando *seq[:]* ?

Strings – Características de mutação

- A alteração de um item de uma *string* gera um erro
- Isto acontece porquê em python as *strings* são imutáveis
- Uma vez declarada ela não pode mudar o valor de seus itens, mas é possível criar uma nova *string* a partir de uma já criada

```
>>> seq = 'UFAL'
>>> seq[3] = 'R'
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
TypeError: 'str' object does not support item assignment
>>> seq2 = seq[:3] + 'R'
>>> seq2
'UFAR'
```

Strings – Métodos de *strings*

- Um **método** é similar à uma função, ele recebe argumentos e retorna valores
- A sintaxe para a chamada de um método (**invocação**) deve ser precedida de um objeto (variável ou valor) separado por um ponto (.)
- Em aulas passadas conhecemos o método *append()* que é utilizado com listas

```
>>> lista = ['A', 'B', 'C']
>>> lista.append('D')
>>> lista
['A', 'B', 'C', 'D']
```

Strings – Métodos de *strings*

- Os métodos normalmente fornecem mecanismos para manipular os objetos dados
- A método *append()* permite adicionar itens em um objeto do tipo lista
- Assim como as listas, *strings* também possuem métodos de manipulação
- A lista com os métodos que podem ser chamados em um objeto do tipo *string* pode ser vista utilizando o comando *dir*, como a seguir

```
>>> dir('texto')
```

Strings – Métodos de *strings*

- O método *upper()* retorna uma nova *string* com todas as letras maiúsculas baseado no objeto dado

```
>>> seq = 'ufal'
>>> seq.upper()
'UFAL'
```

- O método *lower()* retorna uma nova *string* com todas as letras minúsculas baseado no objeto dado

```
>>> 'MARIA'.lower()
'maria'
```

Strings – Métodos de *strings*

- O método *upper()* retorna uma nova *string* com todas as letras maiúsculas baseado no objeto dado

```
>>> seq = 'ufal'
>>> seq.upper()
'UFAL'
```

- O método *lower()* retorna uma nova *string* com todas as letras minúsculas baseado no objeto dado

```
>>> 'MARIA'.lower()
'maria'
```

O que faz o método *capitalize()* ?

Strings – O operador de formatação

- Em algumas situações a apresentação de dados na tela requer uma formatação dos dados
- Existem casos em que é necessário a apresentação de números com 2 casas decimais, textos com alinhamento à esquerda ou à direita etc.
- O operador de formatação (%) pode ser utilizado nestes casos através da sintaxe a seguir

"<*string de formatação*>" % (<*valores a formatar*>)

Strings – O operador de formatação

- A ***String* de formatação** é uma *string* comum contendo marcadores de formatação
- Um marcador de formatação indica o local da *string* em que um valor será inserido, bem como sua formatação
- Um marcador de formatação é representado pelo caractere % seguido de uma letra que representa o tipo do valor que será formatado

Marcador	Tipo
%d	Número Inteiro
%s	*String*
%f	Ponto flutuante (float)
%c	Caractere

Strings – O operador de formatação

- Os “**valores a formatar**” expressam quais valores serão atribuídos aos marcadores de formatação da *string* dada respeitando a ordem da esquerda para direita e os tipos informados

- Os valores devem ser separados por vírgulas e podem ser expressos dentro de parênteses (tuplas)

- Conversões entre pontos flutuantes e inteiros são promovidas na operação de formatação caso sejam possíveis

Strings – O operador de formatação

```
>>> "Em julho vendemos %d carros" % carros
'Em julho vendemos 52 carros'
```

```
>>> "Em %d dias ganhamos %f mil %s." % (34, 6.1, 'reais')
'Em 34 dias ganhamos 6.100000 mil reais.'
```

```
>>> "Em %d dias ganhamos %.2f mil %s." % (34,6.1,'reais')
'Em 34 dias ganhamos 6.10 mil reais.'
```

```
>>> "%20s" % 'UFAL'
'                UFAL'
```

Strings – O operador *in*

- Muitas vezes é necessário verificar se uma *string* é *substring* de outra
- Em python, o operador in facilita esta verificação
- A expressão <seq1> in <seq2> retorna *True* se <seq1> é substring de <seq2>, como a seguir

```
>>> seq1 = 'idade'
>>> seq2 = 'universidade'
>>> seq1 in seq2
True
>>> seq2 in seq1
False
```

Strings – Comparação de *strings*

- Alguns operadores relacionais podem ser usados na comparação de strings
- O operador == verifica se duas *strings* dadas são iguais
- Os operadores >, >=, < e <= utilizam a ordem alfabética para avaliar expressões de comparação

```
>>> seq1 = 'beatriz'
>>> seq2 = 'ana'
>>> seq1 > seq1
False
>>> seq1 >= 'beatriz'
True
>>> seq2 == 'ana'
True
```

Exercícios

1. Crie um programa que escreva uma frase indicada pelo usuário de trás pra frente.

2. Crie um programa que verifique se uma palavra dada pelo usuário é um palindromo.

3. Crie um programa que verifica se em uma frase dada pelo usuário todas as letras são minúsculas

4. Crie um programa que verifica se em uma frase dada pelo usuário todas as palavras possuem iniciais maiúsculas.

5. Crie um programa que inverta as letras entre maiúsculas e minúsculas de uma frase dada pelo usuário.