Cadeias de caracteres

Aula 4

Diego Padilha Rubert

Faculdade de Computação
Universidade Federal de Mato Grosso do Sul

Algoritmos e Programação

Conteúdo da aula

Introdução

- cadeias de caracteres, também chamadas *strings*, são sequências de caracteres (letras)
- diversas funções que manipulam cadeias de caracteres estão disponíveis nativamente na linguagem Python

Introdução

- cadeias de caracteres, também chamadas *strings*, são sequências de caracteres (letras)
- diversas funções que manipulam cadeias de caracteres estão disponíveis nativamente na linguagem Python

Strings

- as strings estão presentes na maioria dos programas que criamos até agora

```
print("Programar é bacana!")
```

- podemos também ler strings do teclado ou armazená-las em variáveis
- ao definir strings em um programa, tanto faz utilizarmos aspas simples ou aspas duplas, não há distinção

Strings

- as strings estão presentes na maioria dos programas que criamos até agora

```
print("Programar é bacana!")
```

- podemos também ler strings do teclado ou armazená-las em variáveis
- ao definir strings em um programa, tanto faz utilizarmos aspas simples ou aspas duplas, não há distinção

Strings

- as strings estão presentes na maioria dos programas que criamos até agora

```
print("Programar é bacana!")
```

- podemos também ler strings do teclado ou armazená-las em variáveis
- ao definir strings em um programa, tanto faz utilizarmos aspas simples ou aspas duplas, não há distinção

Inicialização de strings

- podemos atribuir uma string a uma variável utilizando `=`:

```
nome = "Fulano Alves da Silva"
```

Leitura de strings

- strings são lidas no Python 3 através da função `input()`, que recebe opcionalmente um texto para ser apresentado ao usuário e lê todo o texto digitado até pressionarmos **enter**:

```
texto = input('Digite um texto: ')
```

- podemos escrever uma string que estejam armazenada em uma variável passando-a como único argumento para a função print:

```
print(texto)
```

- podemos também utilizar o conversor de tipo `%s`:

```
print("Você digitou o seguinte texto: %s" % texto)
```

Leitura de strings

- strings são lidas no Python 3 através da função `input()`, que recebe opcionalmente um texto para ser apresentado ao usuário e lê todo o texto digitado até pressionarmos **enter**:

```
texto = input('Digite um texto: ')
```

- podemos escrever uma string que estejam armazenada em uma variável passando-a como único argumento para a função print:

```
print(texto)
```

- podemos também utilizar o conversor de tipo `%s`:

```
print("Você digitou o seguinte texto: %s" % texto)
```

Leitura de strings

- strings são lidas no Python 3 através da função `input()`, que recebe opcionalmente um texto para ser apresentado ao usuário e lê todo o texto digitado até pressionarmos **enter**:

```
texto = input('Digite um texto: ')
```

- podemos escrever uma string que estejam armazenada em uma variável passando-a como único argumento para a função print:

```
print(texto)
```

- podemos também utilizar o conversor de tipo `%s`:

```
print("Você digitou o seguinte texto: %s" % texto)
```

Modificando strings

- não é possível alterar uma string, apenas podemos armazenar novas strings na mesma variável, seja a nova string baseada na anterior ou não

Operadores aritméticos de strings

- a linguagem Python oferece vários operadores e funções de strings, mas veremos apenas os mais utilizados
- operadores aritméticos:

Operador	Descrição
+	concatenação
+=	concatenação *in-place*

```
texto1 = "ABC"
texto2 = texto1 + "DEF"
texto1 += "def"
```

- os dois trechos abaixo têm o mesmo efeito:

```
texto = input()
print("Você digitou %s" % texto)
```

```
texto = input()
print("Você digitou " + texto)
```

Operadores aritméticos de strings

- a linguagem Python oferece vários operadores e funções de strings, mas veremos apenas os mais utilizados
- operadores aritméticos:

Operador	Descrição
+	concatenação
+=	concatenação *in-place*

```
texto1 = "ABC"
texto2 = texto1 + "DEF"
texto1 += "def"
```

- os dois trechos abaixo têm o mesmo efeito:

```
texto = input()
print("Você digitou %s" % texto)
```

```
texto = input()
print("Você digitou " + texto)
```

Operadores aritméticos de strings

- a linguagem Python oferece vários operadores e funções de strings, mas veremos apenas os mais utilizados
- operadores aritméticos:

Operador	Descrição
+	concatenação
+=	concatenação *in-place*

```
texto1 = "ABC"
texto2 = texto1 + "DEF"
texto1 += "def"
```

- os dois trechos abaixo têm o mesmo efeito:

```
texto = input()
print("Você digitou %s" % texto)
```

```
texto = input()
print("Você digitou " + texto)
```

Funções de strings

- para uma string `S`, temos as seguintes funções:

Função	Descrição
`len(S)`	devolve o comprimento de **S**
`S.lower()`	devolve **S** em caixa baixa
`S.upper()`	devolve **S** em caixa alta
`S.replace(old, new)`	devolve **S** trocando **old** por **new**
`S.split(sep)`	"quebra" **S** e devolve várias strings, usa **sep** (opcional) como separador, por padrão **sep** é qualquer espaço

- exemplos:

```
texto = input()
print('Você digitou "%s" com comprimento %d' % (texto, len(texto)))
print("Texto maiúsculo: " + S.upper())
print("Texto sem espaços: " + S.replace(" ", "")
```

Funções de strings

- para uma string `S`, temos as seguintes funções:

Função	Descrição
`len(S)`	devolve o comprimento de `S`
`S.lower()`	devolve `S` em caixa baixa
`S.upper()`	devolve `S` em caixa alta
`S.replace(old, new)`	devolve `S` trocando `old` por `new`
`S.split(sep)`	"quebra" `S` e devolve várias strings, usa `sep` (opcional) como separador, por padrão `sep` é qualquer espaço

- exemplos:

```
texto = input()
print('Você digitou "%s" com comprimento %d' % (texto, len(texto)))
print("Texto maiúsculo: " + S.upper())
print("Texto sem espaços: " + S.replace(" ", "")
```

Fatiando strings (*slices*)

- na linguagem Python, podemos facilmente obter **fatias** (*slices*) de strings utilizando o operador `[]`. Para um string `s` e números inteiros `x` e `y`, os slices mais comuns são:

Uso	Descrição
`s[x]`	devolve uma string apenas com a letra na posição `x` em `s`
`s[-x]`	devolve uma string apenas com a letra na posição `x` de trás para frente em `s`
`s[x:y]`	devolve uma string com as letras da posição `x` até a posição **anterior** a `y` em `s`

- para uma string de tamanho `n`, as posições são numeradas de `0` a `n-1` (ex: para 10 letras, posições 0 a 9)
- se `x` é vazio considera-se `0`, se `y` é vazio considera-se o tamanho da string

Fatiando strings (*slices*)

- na linguagem Python, podemos facilmente obter **fatias** (*slices*) de strings utilizando o operador `[]`. Para um string `S` e números inteiros `x` e `y`, os slices mais comuns são:

Uso	Descrição
`S[x]`	devolve uma string apenas com a letra na posição `x` em `S`
`S[-x]`	devolve uma string apenas com a letra na posição `x` de trás para frente em `S`
`S[x:y]`	devolve uma string com as letras da posição `x` até a posição **anterior** a `y` em `S`

- para uma string de tamanho `n`, as posições são numeradas de `0` a `n-1` (ex: para 10 letras, posições 0 a 9)
- se `x` é vazio considera-se `0`, se `y` é vazio considera-se o tamanho da string

Fatiando strings (*slices*)

- na linguagem Python, podemos facilmente obter **fatias** (*slices*) de strings utilizando o operador `[]`. Para um string `S` e números inteiros `x` e `y`, os slices mais comuns são:

Uso	Descrição
`S[x]`	devolve uma string apenas com a letra na posição `x` em `S`
`S[-x]`	devolve uma string apenas com a letra na posição `x` de trás para frente em `S`
`S[x:y]`	devolve uma string com as letras da posição `x` até a posição **anterior** a `y` em `S`

- para uma string de tamanho `n`, as posições são numeradas de `0` a `n-1` (ex: para 10 letras, posições 0 a 9)
- se `x` é vazio considera-se `0`, se `y` é vazio considera-se o tamanho da string

Fatiando strings (*slices*)

- exemplos:

```
# aqui escrevemos algumas fatias da
# string digitada pelo usuário
texto = input()
print(texto[3]) # posição 3
print(texto[-2]) # penúltma posição
print(texto[2:6]) # posições 2 até 5
print(texto[:5]) # posições 0 até 4
print(texto[2:]) # posições 2 até a última

# abaixo obtemos as posições x até y,
# com valores definidos pelo usuário
x, y = int(input()), int(input())
print (texto[x:y+1])
```

Operadores relacionais de strings

- operadores relacionais:

Operador	Descrição
==	igual a (maiúsculas e minúsculas diferem)
!=	diferente de (maiúsculas e minúsculas diferem)
<	menor que (lexicograficamente, tabela ASCII)
>	maior que (lexicograficamente, tabela ASCII)
<=	menor que ou igual a (lexicograficamente, ASCII)
>=	maior que ou igual a (lexicograficamente, ASCII)

Tabela ASCII

Bin	Dec	Sim	Bin	Dec	Sim	Bin	Dec	Sim
0010 0000	32		0100 0000	64	@	0110 0000	96	`
0010 0001	33	!	0100 0001	65	A	0110 0001	97	a
0010 0010	34	"	0100 0010	66	B	0110 0010	98	b
0010 0011	35	#	0100 0011	67	C	0110 0011	99	c
0010 0100	36	$	0100 0100	68	D	0110 0100	100	d
0010 0101	37	%	0100 0101	69	E	0110 0101	101	e
0010 0110	38	&	0100 0110	70	F	0110 0110	102	f
0010 0111	39	'	0100 0111	71	G	0110 0111	103	g
0010 1000	40	(	0100 1000	72	H	0110 1000	104	h
0010 1001	41	)	0100 1001	73	I	0110 1001	105	i
0010 1010	42	*	0100 1010	74	J	0110 1010	106	j
0010 1011	43	+	0100 1011	75	K	0110 1011	107	k
0010 1100	44	,	0100 1100	76	L	0110 1100	108	l
0010 1101	45	-	0100 1101	77	M	0110 1101	109	m
0010 1110	46	.	0100 1110	78	N	0110 1110	110	n
0010 1111	47	/	0100 1111	79	O	0110 1111	111	o
0011 0000	48	0	0101 0000	80	P	0111 0000	112	p
0011 0001	49	1	0101 0001	81	Q	0111 0001	113	q
0011 0010	50	2	0101 0010	82	R	0111 0010	114	r
0011 0011	51	3	0101 0011	83	S	0111 0011	115	s
0011 0100	52	4	0101 0100	84	T	0111 0100	116	t
0011 0101	53	5	0101 0101	85	U	0111 0101	117	u
0011 0110	54	6	0101 0110	86	V	0111 0110	118	v
0011 0111	55	7	0101 0111	87	W	0111 0111	119	w
0011 1000	56	8	0101 1000	88	X	0111 1000	120	x
0011 1001	57	9	0101 1001	89	Y	0111 1001	121	y
0011 1010	58	:	0101 1010	90	Z	0111 1010	122	z
0011 1011	59	;	0101 1011	91	[	0111 1011	123	{
0011 1100	60	<	0101 1100	92	\	0111 1100	124	|
0011 1101	61	=	0101 1101	93	]	0111 1101	125	}
0011 1110	62	>	0101 1110	94	^	0111 1110	126	~
0011 1111	63	?	0101 1111	95	_			

Um detalhe importante

- a forma que sabemos até o momento para lermos valores do teclado (usuário) é através da função `input()`, convertendo o texo para o tipo desejado caso necessário
- contudo, essa função tem uma limitação: ela aceita ler apenas um valor por linha, para lermos diversos números precisamos utilizá-la várias vezes e digitar *valor, enter, valor, enter, ...*:

```
num1 = int(input())
num2 = int(input())
num3 = int(input())
```

ou

```
num1, num2, num3 = int(input()), int(input()), int(input())
```

Um detalhe importante

- a forma que sabemos até o momento para lermos valores do teclado (usuário) é através da função `input()`, convertendo o texo para o tipo desejado caso necessário
- contudo, essa função tem uma limitação: ela aceita ler apenas um valor por linha, para lermos diversos números precisamos utilizá-la várias vezes e digitar *valor, enter, valor, enter, ...*:

```
num1 = int(input())
num2 = int(input())
num3 = int(input())
```

ou

```
num1, num2, num3 = int(input()), int(input()), int(input())
```

Um detalhe importante

- a forma que sabemos até o momento para lermos valores do teclado (usuário) é através da função `input()`, convertendo o texo para o tipo desejado caso necessário
- contudo, essa função tem uma limitação: ela aceita ler apenas um valor por linha, para lermos diversos números precisamos utilizá-la várias vezes e digitar *valor, enter, valor, enter, ...*:

```
num1 = int(input())
num2 = int(input())
num3 = int(input())
```

ou

```
num1, num2, num3 = int(input()), int(input()), int(input())
```

Um detalhe importante

- se quisermos ler vários valores em uma única linha, podemos utilizar uma receita bastante conhecida:
  1. lemos todos os valores de uma vez como se fossem um único texto
  2. separamos os valores como se fossem “palavras” do texto
  3. convertemos essas “palavras” para os tipos desejados

```
# esse código lê três valores em uma linha,
# um inteiro, seguido de um float, seguido de outro inteiro
valor1, valor2, valor3 = input().split()
valor1, valor2, valor3 = int(valor1), float(valor2), int(valor3)
```

Exercícios

1. Leia duas strings no console do Python e faça testes com os operadores aritméticos e relacionais, funções e fatias (*slices*).
2. Considere o trecho de código abaixo:

```
print("Na primeira aula, escrevi "Programar é bacana!" na tela")
```

Teste essa instrução no console ou em um programa Python. Você verá que ocorre um erro porque estamos tentando escrever textualmente aspas duplas dentro das aspas duplas que definem uma string. De que formas podemos resolver isso?

Exercícios

1. Leia duas strings no console do Python e faça testes com os operadores aritméticos e relacionais, funções e fatias (*slices*).
2. Considere o trecho de código abaixo:

```
print("Na primeira aula, escrevi "Programar é bacana!" na tela")
```

   Teste essa instrução no console ou em um programa Python. Você verá que ocorre um erro porque estamos tentando escrever textualmente aspas duplas dentro das aspas duplas que definem uma string. De que formas podemos resolver isso?

Exercícios

3. Dadas duas cadeias de caracteres `cadeia1` e `cadeia2`, concatenar `cadeia2` no final de `cadeia1`. A cadeia resultante deve estar armazenada em `cadeia1` e ser mostrada na tela.
4. Dada uma cadeia de caracteres `cadeia` e uma palavra `p`, buscar a primeira ocorrência de `p` em `cadeia`. Se `p` ocorre em `cadeia`, mostrar a posição da primeira ocorrência; caso contrário, mostrar o valor `-1`. A palavra `p` pode ter desde apenas uma letra até várias letras. Pesquise sobre a função de strings `find`.

Exercícios

3. Dadas duas cadeias de caracteres `cadeia1` e `cadeia2`, concatenar `cadeia2` no final de `cadeia1`. A cadeia resultante deve estar armazenada em `cadeia1` e ser mostrada na tela.
4. Dada uma cadeia de caracteres `cadeia` e uma palavra `p`, buscar a primeira ocorrência de `p` em `cadeia`. Se `p` ocorre em `cadeia`, mostrar a posição da primeira ocorrência; caso contrário, mostrar o valor `-1`. A palavra `p` pode ter desde apenas uma letra até várias letras. Pesquise sobre a função de strings `find`.

Exercícios

5. Dadas duas cadeias de caracteres `cadeia1` e `cadeia2`, compará-las e devolver um valor menor que zero se `cadeia1` é lexicograficamente menor que `cadeia2`, o valor zero se `cadeia1` é igual ou tem o mesmo conteúdo que `cadeia2`, ou um valor maior que zero se `cadeia1` é lexicograficamente maior que `cadeia2`. Seu programa deve ignorar a caixa das letras, ou seja, as cadeias `teste` e `Teste` devem ser consideradas iguais por exemplo.