



Python

String

André Tavares da Silva
andre.silva@udesc.br



String

- String em Python, denominado str, é usado para representar e manipular dados de texto ou, em outras palavras, uma sequência de caracteres, incluindo espaços, pontuação e diversos símbolos.
- A definição do valor de uma string é representado como uma sequência de caracteres delimitada por aspas simples ou duplas:

```
nome = 'Manoel "da Farmácia" Oliveira'
```

```
endereço = "rua da caixa d'água, 208"
```

```
cpf = '123.456.789-00'
```



String

texto

O	I	á	32	M	u	n	d	o	!
---	---	---	----	---	---	---	---	---	---

nome

T	A	D	S		U	D	E	S	C
---	---	---	---	--	---	---	---	---	---

- Uma string nada mais é do que uma cadeia de caracteres (vetor/sequência de caracteres).



String

- Assim como vetores, também podemos concatenar strings, fazer cópias de strings, acessar um elemento dela ou obter seu comprimento:

```
str1 = 'Olá ' + 'Mundo!'
```

```
str2 = str1 * 3
```

```
str3 = str1[9]
```

```
tamanho = len(str2)
```

String

- Mas diferente da lista, as strings são imutáveis. Isso significa que não podemos alterar ou manipular seu conteúdo diretamente.

```
>>> str1 = 'Olá Mundo!'
>>> str1[4] = 'm'
TypeError: 'str' object does not support item assignment

>>> id(str1)
139985731104816
>>> str1 = str1.replace("M", "n")
>>> id(str1)
139985709581968 # diferente!
```

String

- Assim como vetores, também podemos concatenar strings, fazer cópias de strings, acessar um elemento dela ou obter seu comprimento:

```
str1 = 'Olá ' + 'Mundo!'
str2 = str1 * 3
str3 = str1[9]
tamanho = len(str2)
```

- Outra operação útil em strings é verificar se uma substring está ou não contida em uma string:

```
if 'Olá' in str1:
    print('está')
if '$' not in str1:
    str1 = str1 + '$'
```

String – localizando substring

- Além de verificar se uma substring está contida em uma string, podemos identificar ONDE essa substring está. Para isso, utiliza-se o método `find`.
- Se a substring for encontrada, retorna a posição dela. Do contrário, retorna -1, indicando que a substring não foi localizada.

```
>>> str1 = 'Olá Mundo!'
>>> pos1 = str1.find("Mundo")
>>> print(pos1)
4
>>> print(str1.find("xyz"))
-1
```

String – criando lista de substring

- Podemos usar o método `split()` para dividir a string em partes e criar uma lista de strings com cada palavra contida na string original.
- Sintaxe: `<string>.split(<separador>, <maxsplit>)`, onde `<string>` é a string original, `<separador>` é o separador a ser usado (padrão é o espaço) e `<maxsplit>` é a quantidade máxima de divisões.

```
>>> str1 = 'Seja muito bem-vindo ao curso TADS'
>>> lista = str1.split()
>>> lista
['Seja', 'muito', 'bem-vindo', 'ao', 'curso', 'TADS']
```

Lista de String

- Como vimos anteriormente, podemos criar listas com diversos tipos de dados, inclusive strings.
- Podemos também utilizar o operador `in` para verificar se existe uma determinada string na lista:

```
>>> animais = ['vaca', 'cavalo', 'cachorro']
>>> 'gato' in animais
False
```

Python – Funções para listas

- Abaixo algumas funções Python para uso com listas:

Função	Explicação
<code>len(<lista>)</code>	Comprimento da lista
<code>min(<lista>)</code>	Menor item em uma lista
<code>max(<lista>)</code>	Maior item em uma lista
<code>sum(<lista>)</code>	Soma dos itens da lista

Python – Funções para listas

- Python é uma linguagem interpretada. Os programas criados na linguagem são lidos e analisados linha a linha e por isso acaba sendo mais lento que programas compilados.
- Como muitas operações em listas são bastante comuns, existem comandos (funções) escritas em linguagem C que o Python faz a chamada e são executadas com uma performance muito melhor.

Python – Métodos de listas

- Abaixo alguns métodos de listas em Python:

Função	Explicação
<code>lista.append(<item>)</code>	Inclui item ao final da lista
<code>lista.count(<item>)</code>	Retorna o número de ocorrências de item
<code>lista.insert(n, <item>)</code>	Insere um item na posição n
<code>lista.pop()</code>	Remove o último elemento da lista
<code>lista.remove(<item>)</code>	Remove a primeira ocorrência do item na lista
<code>lista.reverse()</code>	Inverte a ordem dos itens da lista
<code>lista.sort()</code>	Ordena a lista na ordem crescente dos itens

Python – Cópia de lista

- Como devemos proceder caso seja necessário copiar o conteúdo de uma lista? Vejamos o que acontece no código abaixo:

```
>>> animais = ['vaca', 'cavalo', 'cachorro']
>>> b = animais # b é um apelido (alias) de animais
>>> b.append('porco')
>>> animais
['vaca', 'cavalo', 'cachorro', 'porco']
```

Python – fatia de lista (slices)

- Para selecionar partes específicas de uma lista em Python, usamos a funcionalidade de fatias.
- São definidas pelo uso de colchetes ([]) e de dois pontos (:) para especificar a faixa de elementos que se quer selecionar.
- A sintaxe é a seguinte: <lista>[<início>:<fim>:<passo>], onde <início> é o índice inicial da seleção, <fim> é o índice final e <passo> a quantidade de elementos a serem pulados.

Python – fatia de lista (slices)

- Vejamos o exemplo abaixo:

```
>>> numeros = [0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10]
>>> numeros[:5]
[0, 1, 2, 3, 4]
>>> pares = numeros[::2]
>>> pares
[0, 2, 4, 6, 8, 10]
>>> impares = numeros[1::2]
>>> impares
[1, 3, 5, 7, 9]
>>> numeros[3:10:3]
[3, 6, 9]
```

Python – Cópia de lista

- Agora sim, copiando uma lista:

```
>>> animais = ['vaca', 'cavalo', 'cachorro']
>>> b = animais[:] # copia animais do início ao fim
>>> b.append('porco')
>>> animais
['vaca', 'cavalo', 'cachorro']
>>> b
['vaca', 'cavalo', 'cachorro', 'porco']
```




Python

String - format

André Tavares da Silva
andre.silva@udesc.br



String - format

- O uso básico do método `format` tem a seguinte forma:

```
>>> print('Meu {} disse "{}!"'.format('amigo', 'olá'))  
Meu amigo disse "Olá"!
```
- As chaves e seus conteúdos (chamados campos de formatação) são substituídos pelos objetos passados para o método `str.format()`. Um número nas chaves pode ser usado para referenciar a posição do objeto passado no método `str.format()`:

```
>>> print('{0} e {1}'.format('cédulas', 'moedas'))  
cédulas e moedas  
>>> print('{1} e {0}'.format('cédulas', 'moedas'))  
moedas e cédulas
```



String - format

- Se argumentos nomeados são passados para o método `str.format()`, seus valores serão referenciados usando o nome do argumento:

```
>>> print('Este {objeto} é {adjetivo}.'.format(  
    objeto='óculos', adjetivo='muito feio'))  
Este óculos é muito feio.
```
- Argumentos posicionais e nomeados podem ser combinados à vontade:

```
>>> print('Este {1} com {0} é {adjetivo}.'.format('capa',  
    'celular', adjetivo='lindo'))  
Este celular com capa é lindo.
```



String – format com dicionário

- Se uma string de formatação é muito longa, e não se deseja quebrá-la, pode ser bom fazer referência aos valores a serem formatados por nome, em vez de posição. Isto pode ser feito passando um dicionário usando colchetes `[]` para acessar as chaves:

```
>>> tabela = {'IdNome': 412, 'Cidade': 8142, 'Rua': 8637678}  
>>> print('Nome: {0[IdNome]:d} Cidade: {0[Cidade]:d} '  
    'Rua: {0[Rua]:d}'.format(tabela))  
Nome: 412 Cidade: 8142 Rua: 8637678
```
- Ou passando o dicionário `tabela` nomeados com a notação `**`:

```
>>> print('N: {IdNome:d} C: {Cidade:d} R:{Rua:d}'.format(**tabela))  
Nome: 412 Cidade: 8142 Rua: 8637678
```

String – literais formatados

- Para usar strings literais formatadas, comece uma string com **f** ou **F**, antes de abrir as aspas ou aspas duplas. Dentro dessa string, pode-se escrever uma expressão Python entre caracteres **{ e }**, que podem se referir a variáveis, ou valores literais:

```
>>> ano = 2023
>>> disciplina = 'Algoritmos'
>>> f'No ano de {ano} aprendi {disciplina}.'
'No ano de 2023 aprendi Algoritmos.'

>>> chance = 0.4967
>>> 'Ano {: -9} tem {:2.2%} de chance.'.format(ano,chance)
'Ano      2023 tem 49.67% de chance.'
```

String – literais formatados

- Para usar strings literais formatadas, comece uma string com **f** ou **F**, antes de abrir as aspas ou aspas duplas. Dentro dessa string, pode-se escrever uma expressão Python entre caracteres **{ e }**, que podem se referir a variáveis, ou valores literais:

```
>>> ano = 2023
>>> disciplina = 'Algoritmos'
>>> f'No ano de {ano} aprendi {disciplina}.'
'No ano de 2023 aprendi Algoritmos.'

>>> chance = 0.4967
>>> 'Ano {: -9} tem {:2.2%} de chance.'.format(ano,chance)
'Ano      2023 tem 49.67% de chance.'
```

<num>	- inteiro com sinal
<int.dec>	- float (real)
u	- inteiro sem sinal
o	- octal
b	- número binário
x	- hexadecimal (minúsculo)
X	- hexadecimal (maiúsculas)
e	- notação exponencial
%	- porcentagem

String – literais formatados

- Para usar strings literais formatadas, comece uma string com **f** ou **F**, antes de abrir as aspas ou aspas duplas. Dentro dessa string, pode-se escrever uma expressão Python entre caracteres **{ e }**, que podem se referir a variáveis, ou valores literais:

```
>>> ano = 2023
>>> disciplina = 'Algoritmos'
>>> f'No ano de {ano:o} aprendi {disciplina}.'
'No ano de 3747 aprendi Algoritmos.'
>>> f'No ano de {ano:X} aprendi {disciplina}.'
'No ano de 7E7 aprendi Algoritmos.'
>>> f'No ano de {ano:b} aprendi {disciplina}.'
'No ano de 11111100111 aprendi Algoritmos.'
```

String – literais formatados

- Lembrando que também podemos usar strings literais formatadas usando o operador **%** (percentual):

```
>>> ano = 2023
>>> disciplina = 'Algoritmos'
>>> f'No ano de {ano:d} aprendi {disciplina}.'
'No ano de 2023 aprendi Algoritmos.'

>>> 'No ano de {: -6} aprendi {:s}.'.format(ano,disciplina)
'No ano de      2023 aprendi Algoritmos.'

>>> 'No ano de %5d aprendi %s.' % (ano,disciplina)
'No ano de  2023 aprendi Algoritmos.'
```

String – métodos

- A função `str()` retorna representações de valores legíveis para as pessoas, enquanto `repr()` gera representações que o interpretador Python consegue ler:

```
>>> str(1/7)
'0.14285714285714285'
>>> x = 10 * 3.25
>>> y = 200 * 200
>>> s = 'x é ' + repr(x) + ', e y é ' + repr(y) + '...'
>>> s
'x é 32.5, e y é 40000...'
>>> texto = repr('olá\n')
>>> print(texto)
'olá\n'
>>> repr((x, y, ('xis', 'y'))))
"(32.5, 40000, ('xis', 'y'))"
```

String – métodos

- Para concatenar valores e strings em uma nova string, podemos usar o operador soma (+) ou usando o operador `join()`:

```
>>> 'a' + 'b' + 'c'
'abc'
>>> strings = ['do', 're', 'mi']
>>> ''.join(strings)
'doremi'
>>> ', '.join(strings)
'do, re, mi'
```

String – métodos

- Para concatenar valores e strings em uma nova string, podemos usar o operador soma (+):

```
>>> 'a' + 'b' + 'c'
'abc'
>>> strings = ['do', 're', 'mi']
>>> strings[0] + strings[1] + strings[2]
'doremi'
```

String – fatias

- Lembrando** que podemos selecionar partes de uma string:

```
>>> s = 'um dois três quatro cinco'
>>> s[8:12]
'três'
>>> s[:2]
'u ostê utocno'
>>> s[13]
'q'
>>> s[::-1]
'ocnic ortauq sêrt siod mu'
>>> s[11:7:-1]
'sêrt'
```


Tabela ASCII

- É um sistema de representação de letras, algarismos e sinais de pontuação e de controle, através de um sinal codificado em forma de código binário (cadeias de bits formada por vários 0 e 1), desenvolvido a partir de 1960, que representa um conjunto de 128 sinais: 95 sinais gráficos (letras do alfabeto latino, algarismos arábicos, sinais de pontuação e sinais matemáticos) e 33 sinais de controle, utilizando 7 bits para representar todos os seus símbolos.
- É usada pela maior parte da indústria de computadores para a troca de informações.
- Os valores de 0 a 31 e o 127 armazenam sinais como: nulo, quebra de linha, sinal sonoro (*bell*), espaço atrás (*backspace*), del, tabulação, etc.
- Valores de 32 a 126 armazenam os sinais gráficos.

Python

ASCII

(American Standard Code for Information Interchange)

André Tavares da Silva
andre.silva@udesc.br

Tabela ASCII

- Como cada byte possui 8 bits, o bit não utilizado pela tabela ASCII pode ser utilizado de formas diferentes:
 - O padrão UTF-8 utiliza o bit excedente do primeiro byte para indicar que o Code point tem um valor que excede os valores da tabela ASCII (acima de 127) e necessitará de mais bytes para ser representado.
 - A Microsoft utilizou este bit excedente para codificação de caracteres adicionais no Windows Code Page.
 - Outra utilização do bit excedente é informar a paridade em transmissões assíncronas de baixa velocidade.
- A existência de um bit excedente em cada byte cria oportunidades para utilizar os 7 bits da Tabela ASCII em diferentes codificações não padronizadas, algumas vezes chamadas de "Tabela ASCII", que erroneamente passa a ideia que a Tabela ASCII foi oficialmente ampliada para utilizar 8 bits, fato que nunca ocorreu.

ASCII control characters			ASCII printable characters					
00	NULL	(Null character)	32	space	64	@	96	.
01	SOH	(Start of Header)	33	!	65	A	97	a
02	STX	(Start of Text)	34	"	66	B	98	b
03	ETX	(End of Text)	35	#	67	C	99	c
04	EOT	(End of Trans.)	36	\$	68	D	100	d
05	ENQ	(Enquiry)	37	%	69	E	101	e
06	ACK	(Acknowledgement)	38	&	70	F	102	f
07	BEL	(Bell)	39	'	71	G	103	g
08	BS	(Backspace)	40	(72	H	104	h
09	HT	(Horizontal Tab)	41)	73	I	105	i
10	LF	(Line feed)	42	*	74	J	106	j
11	VT	(Vertical Tab)	43	+	75	K	107	k
12	FF	(Form feed)	44	,	76	L	108	l
13	CR	(Carriage return)	45	-	77	M	109	m
14	SO	(Shift Out)	46	.	78	N	110	n
15	SI	(Shift In)	47	/	79	O	111	o
16	DLE	(Data link escape)	48	0	80	P	112	p
17	DC1	(Device control 1)	49	1	81	Q	113	q
18	DC2	(Device control 2)	50	2	82	R	114	r
19	DC3	(Device control 3)	51	3	83	S	115	s
20	DC4	(Device control 4)	52	4	84	T	116	t
21	NAK	(Negative acknowl.)	53	5	85	U	117	u
22	SYN	(Synchronous idle)	54	6	86	V	118	v
23	ETB	(End of trans. block)	55	7	87	W	119	w
24	CAN	(Cancel)	56	8	88	X	120	x
25	EM	(End of medium)	57	9	89	Y	121	y
26	SUB	(Substitute)	58	:	90	Z	122	z
27	ESC	(Escape)	59	;	91	[123	{
28	FS	(File separator)	60	<	92	\	124	
29	GS	(Group separator)	61	=	93]	125	}
30	RS	(Record separator)	62	>	94	^	126	~
31	US	(Unit separator)	63	?	95	_		
127	DEL	(Delete)						

String - format

- Em Python, se quisermos ver qual o código ASCII de um caractere, podemos usar a função `ord()`:

```
>>> ord("t")
116
>>> str1 = "Teste"
>>> for i in str1:
>>>     print(ord(i))
84
101
115
116
101
```

String - format

- Outra forma de exibir o código ASCII de um caractere é transformando ele para um byte:

```
>>> list(b"Teste")
[84, 101, 115, 116, 101]

>>> list("Teste".encode('ascii')) #outra conversão
[84, 101, 115, 116, 101]
```

String - format

- Em Python, se quisermos ver qual o código ASCII de um caractere, podemos usar a função `ord()`:

```
>>> str1 = "Teste"
>>> lista = [ord(c) for c in str1]
>>> lista
[84, 101, 115, 116, 101]
```

Bibliografia

- RIBEIRO, João A. **Introdução à Programação e aos Algoritmos**. Grupo GEN, 2019.
- MANZANO, José Augusto Navarro G.; OLIVEIRA, Jayr Figueiredo D. **Algoritmos - Lógica para Desenvolvimento de Programação de Computadores**. Editora Saraiva, 2019.
- AGUILAR, Luis J. **Fundamentos de Programação**. Grupo A, 2008.
- CORMEN, Thomas. **Algoritmos - Teoria e Prática**. Grupo GEN, 2012.
- ZIVIANI, Nivio. **Projeto de Algoritmos: com Implementações em Pascal e C** - 3a edição revista e ampliada. Cengage Learning Brasil, 2018.
- ALVES, William P. **Linguagem e Lógica de Programação**. Editora Saraiva, 2013.