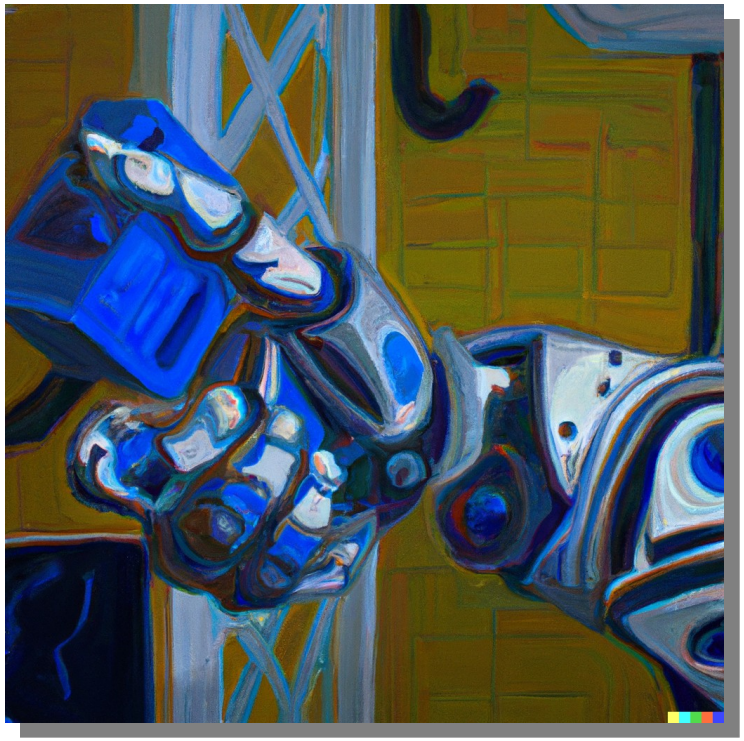
Universidade de Brasília
Departamento de Ciência da Computação



Semana 09

Strings

CIC0004
Algoritmos e Programação de Computadores

Prof. Pedro Garcia Freitas https://pedrogarcia.gitlab.io/ pedro.garcia@unb.br

Brasilia

1. Objetivos

Esta aula introduz o conceito de string (vetor de caracteres) e apresenta como tal conceito é implementado e explorado em linguagem Python.

2. Definição de Strings

- Em programação de computadores, uma string é tradicionalmente uma sequência de caracteres, seja como uma constante literal ou como algum tipo de variável.

2. Definição de Strings

- Em programação de computadores, uma string é tradicionalmente uma sequência de caracteres, seja como uma constante literal ou como algum tipo de variável.

```
char str[] = "Hello world!"
```

Index	0	1	2	3	4	5	6	7	8	9	10	11
str[]	H	e	l	l	o		w	o	r	l	d	\0
ASCII code	072	101	108	108	111	032	087	111	114	108	100	000

2. Definição de Strings

- Variáveis de string podem permitir que seus elementos sejam modificados e o comprimento alterado, ou pode ser fixa (algumas linguagens não implementam strings com side effects).

2. Definição de *Strings*

- Uma *string* é usada para armazenar e manipular textos como palavras, nomes e sentenças.
- Quando uma *string* aparece literalmente no código-fonte, ela é conhecida como uma ***string* literal** ou uma ***string* anônima**.

2. Definição de *Strings*

```
# character literal in single quote
v = 'n'
# character literal in double quotes
w = "a"
variable_str = input()
print(v)
print(w)
print(variable_str)
print("My literal string")
```

2. Definição de *Strings*

```
# character literal in single quote
v = 'n'
# character literal in double quotes
w = "a"
variable_str = input()
print(v)
print(w)
print(variable_str)
print("My literal string")
```

String literais

3. Propósito das *Strings*

- Um dos principais objetivos das strings é armazenar texto legível por humanos, como palavras e frases.
- As strings são usadas para comunicar informações de um programa de computador ao usuário do programa.

3. Propósito das *Strings*

- Um programa também pode receber entradas de strings do usuário.
- Além disso, as strings podem armazenar dados expressos como caracteres, mas que não são necessariamente destinados à leitura humana.

3. Propósito das *Strings*

- Exemplo de strings e seus propósitos:
  - A *string* `"upload de arquivo concluído"` pode ser usada no contexto de um software que exibe o status de um programa para os usuários finais.
  - No código-fonte do programa, essa mensagem provavelmente apareceria como uma string literal.

3. Propósito das *Strings*

- Exemplo de strings e seus propósitos:
  - Um texto inserido pelo usuário, como `"Consegui um novo emprego hoje"`, como uma atualização de status em um serviço de mídia social.
  - Em vez de uma *string* literal, o software provavelmente armazenaria essa *string* em um **banco de dados**.

3. Propósito das *Strings*

- Exemplo de strings e seus propósitos:
  - Dados alfabéticos, como **`"AGATGCCGT"`**, representando sequências de ácido nucleico do **DNA**.
  - Eis um exemplo de como uma string pode ser usada para conter uma informação sem necessariamente se destinar à leitura humana.

4. Histórico

- O uso da palavra "*string*" para significar **"uma sequência de símbolos ou elementos linguísticos em uma ordem definida"** surgiu da matemática, lógica simbólica e teoria linguística para falar sobre o comportamento formal de sistemas simbólicos, deixando de lado o significado dos símbolos.

Burchfield, R.W. (1986). "string". A Supplement to the Oxford English Dictionary. Oxford at the Clarendon Press.

5. String datatypes

- Um *string datatype* é um tipo de dado que modela uma string.
- As strings são um tipo de dado tão importante e útil que estão implementadas em praticamente todas as linguagens de programação.

5. String datatypes

- Em algumas linguagens, elas estão disponíveis como **tipos primitivos** e, em outras, como tipos compostos.
- A sintaxe da maioria das linguagens de programação de alto nível permite que uma string represente uma instância de um tipo de dado textual.

5. String datatypes

5.1 Comprimento de uma string:

- Embora as strings formais possam ter um comprimento finito arbitrário, o comprimento das strings em linguagens reais **muitas vezes é limitado a um máximo artificial**.

5. String datatypes

5.1 Comprimento de uma string:

- Em geral, existem dois tipos de tipos de dado *string*: ***strings* de comprimento fixo**, e ***strings* de comprimento variável**, cujo comprimento não é arbitrariamente fixo e que podem utilizar **quantidades variáveis de memória** dependendo dos requisitos reais em tempo de execução.

5. String datatypes

5.1 Comprimento de uma string:

- A maioria das strings nas linguagens de programação modernas são strings de comprimento variável.
- Claro, **mesmo as strings de comprimento variável têm um limite de comprimento: o tamanho da memória do computador** disponível.

5. String datatypes

5.1 Comprimento de uma string:

- O comprimento da *string* pode ser armazenado como um **inteiro separado** (o que pode impor um limite artificial ao comprimento) ou **implicitamente por meio de um caractere de terminação**, geralmente um valor de caractere com todos os bits iguais a zero.

5. String datatypes

5.1 Comprimento de uma string:

```
class String:
  text: list = []
  size: int = 0
```

Inteiro separado indicando a quantidade de caracteres

```
char str[] = "Hello world!\0"
```

Caractere de terminação

5. String datatypes

Como o interpretador oficial do Python (CPython) implementa o tipo string.

```
typedef struct {
    PyObject_VAR_HEAD
    long ob_shash;
    int ob_sstate;
    char ob_sval[1];

    /* Invariants:
     *     ob_sval contains space for 'ob_size+1' elements.
     *     ob_sval[ob_size] == 0.
     *     ob_shash is the hash of the string or -1 if not computed yet.
     *     ob_sstate != 0 iff the string object is in stringobject.c's
     *       'interned' dictionary; in this case the two references
     *       from 'interned' to this object are *not counted* in ob_refcnt.
     */
} PyStringObject;
```

Fonte: https://svn.python.org/projects/python/trunk/Include/stringobject.h

5. String datatypes

5.2 Codificação de caracteres (character encoding)

- Os tipos de dados de string historicamente alocavam um byte por caractere.
- Esses conjuntos de caracteres eram tipicamente baseados em **ASCII** ou **EBCDIC**.

5. String datatypes

5.2 Codificação de caracteres (character encoding)

- Se um texto gerado num sistema de codificação for exibido noutro sistema que usa uma codificação diferente, o texto geralmente é apresentado distorcido.

Fonte: https://www.rapidtables.com/code/text/ascii-table.html

5. String datatypes

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

5. String datatypes

5.3 Mutabilidade de *strings*

- Algumas linguagens, como **C++**, **Perl** e **Ruby**, normalmente permitem que o conteúdo de uma *string* seja alterado após sua criação; essas são chamadas de **strings mutáveis**.

5. String datatypes

5.3 Mutabilidade de *strings*

- Em outras linguagens, como **Java**, **JavaScript**, **Lua**, **Python** e **Go**, o **valor é fixo** e uma **nova string é criada** se alguma alteração for feita; essas são chamadas de **strings imutáveis**.

5. String datatypes

5.3 Mutabilidade de *strings*

- Algumas dessas linguagens com *strings* imutáveis também fornecem outro tipo que é mutável, como o ***StringBuilder*** do Java e .NET, o Java ***StringBuffer***,.

5. String datatypes

5.3 Mutabilidade de *strings*

- Existem vantagens e desvantagens na imutabilidade: embora as strings imutáveis demandam mais memória (cada operação gera uma nova cópia), elas são mais simples e completamente seguras em termos de concorrência.

6. Exemplo de *strings* em Python

6.1 Declaração de *strings*

```
# create a string using double quotes
string1 = "Python programming"

# create a string using single quotes
string2 = 'Python programming'

# create a string using triple quotes
string3 = '''Both ' and " available!'''
```

6. Exemplo de *strings* em Python

6.1 Declaração de *strings*

exemplo01.py

```
1  # create string type variables
2
3  # create a string using double quotes
4  s1 = "Double quotes string"
5
6  # create a string using single quotes
7  s2 = 'Single quotes string'
8
9  # create a string using triple quotes
10 s3 = '''Both ' and " available!'''
11
12 for s in (s1,s2,s3):
13     print(s)
```

6. Exemplo de *strings* em Python

6.1 Declaração de *strings*

```
>>> %Run exemplo01.py
Double quotes string
Single quotes string
Both ' and " available!

>>>
```

6. Exemplo de *strings* em Python

6.2 Acesso de caracteres pertencentes à *string*

Podemos acessar os caracteres em uma *string* de três maneiras:

- Indexação
- Indexação negativa
- *Slicing*

6. Exemplo de *strings* em Python

6.2 Acesso de caracteres pertencentes à *string*

Indexação (*indexing*): Uma maneira é tratar as strings como uma lista e usar valores de índice.

```
greet = 'hello'
# access 2nd index element
print(greet[1]) # "e"
```

6. Exemplo de *strings* em Python

6.2 Acesso de caracteres pertencentes à *string*

Indexação negativa (*negative indexing*): Semelhante a uma lista, o Python permite a indexação negativa para suas strings. Por exemplo,

```
greet = 'hello'
# access 4th element reversely
print(greet[-4]) # "e"
```

6. Exemplo de *strings* em Python

6.2 Acesso de caracteres pertencentes à *string*

Indexação negativa (*negative indexing*): Semelhante a uma lista, o Python permite a indexação negativa para suas strings. Por exemplo,

```
greet = 'hello'
# access 1th element reversely
print(greet[-1]) # "o"
```

6. Exemplo de *strings* em Python

6.2 Acesso de caracteres pertencentes à *string*

Slicing: Python permite o acesso a um intervalo de caracteres em uma string usando o "operador de fatiamento" ":"

```
greet = 'hello'
# access 1th element reversely
print(greet[1:4]) # "ell"
```

6. Exemplo de *strings* em Python

6.2 Acesso de caracteres pertencentes à *string*

Slicing: Python permite o acesso a um intervalo de caracteres e[m] [a]través do "operador de [f]atia"

6. Exemplo de *strings* em Python

6.3 Mutabilidade/Imutabilidade de *strings*

- Em Python, as strings são imutáveis. Isso significa que os caracteres de uma string não podem ser alterados. Por exemplo,

```
message = 'Hola Amigos'
message[0] = 'X'
```

6. Exemplo de *strings* em Python

6.3 Mutabilidade/Imutabilidade de *strings*

TypeError: 'str' object does not support item assignment

nutáveis.
res de uma
dos. Por
ex o,

```
message = 'Hola Amigos'
message[0] = 'X'
```

6. Exemplo de *strings* em Python

6.3 Mutabilidade/Imutabilidade de *strings*

- Portanto, toda operação feita com *strings* como operandos produz uma nova *string*.

```
message = 'Hola Amigos'
message = 'X' + message[1:]
print(message)
```

6. Exemplo de *strings* em Python

6.3 Mutabilidade/Imutabilidade de *strings*

exemplo02.py

```
1  message = 'Hola Amigos'
2  message = 'X' + message[1:]
3  print(message)
```

Shell

```
>>> %Run exemplo02.py
 Xola Amigos
>>>
```

6. Exemplo de *strings* em Python

6.4 *Multiline strings*

Também é possível criar uma *string* multilinha em Python. Para isso, utilizamos três aspas duplas """ ou três aspas simples '''.

6. Exemplo de *strings* em Python

6.4 *Multiline strings*

```
# multiline string
message = """
Never gonna give you up
Never gonna let you down
"""
print(message)
```

6. Exemplo de *strings* em Python

6.4 *Multiline strings*

exemplo03.py

```
1  # multiline string
2  message = """
3  Never gonna give you up
4  Never gonna let you down
5  """
6
7  print(message)
```

Shell

```
>>> %Run exemplo03.py

  Never gonna give you up
  Never gonna let you down

>>>
```

6. Exemplo de *strings* em Python

6.5 Comparação de *strings*

Utilizamos o operador == para comparar duas strings. Se as duas strings forem iguais, o operador retorna True. Caso contrário, retorna False. Por exemplo,

6. Exemplo de *strings* em Python

6.5 Comparação de *strings*

exemplo04.py

```
1  s1 = "Hello, world!"
2  s2 = "I love Python."
3  s3 = "Hello, world!"
4
5  # compare str1 and str2
6  print(f"'{s1}' = '{s2}'?", s1 == s2)
7
8  # compare str1 and str3
9  print(f"'{s1}' = '{s3}'?", s1 == s3)
```

Shell

```
>>> %Run exemplo04.py
  'Hello, world!' = 'I love Python.'? False
  'Hello, world!' = 'Hello, world!'? True

>>>
```

6. Exemplo de *strings* em Python

6.6 Junção de *strings* (concatenação)

Em Python, podemos juntar (concatenar) duas ou mais *strings* usando o operador **+**.

```
greet = "Hello, "
name = "Jack"
# using + operator
result = greet + name
print(result)
```

6. Exemplo de *strings* em Python

6.6 Junção de *strings* (concatenação)

exemplo05.py

```
1  greet = "Hello, "
2  name = "Jack"
3  # using + operator
4  result = greet + name
5  print(result)
```

Shell

```
>>> %Run exemplo05.py
 Hello, Jack
>>>
```

6. Exemplo de *strings* em Python

6.7 Iterar através de uma string em Python

Podemos iterar através de uma string usando um loop **`for`** ou **`while`** em Python. Por exemplo,

```
greet = 'Hello'
# iterating through greet string
for letter in greet:
    print(letter)
```

6. Exemplo de *strings* em Python

6.7 Iterar através de uma string em Python

exemplo06.py

```
1  greet = 'Hello'
2
3  for letter in greet:
4      print(letter)
```

Shell

```
>>> %Run exemplo06.py
H
e
l
l
o

>>>
```

6. Exemplo de *strings* em Python

6.8 Comprimento/tamanho de uma *string*

Em Python, usamos o método **`len()`** para encontrar o comprimento de uma string. Por exemplo,

```
greet = 'Hello'
# count length of greet string
print(len(greet))
```

6. Exemplo de *strings* em Python

6.8 Comprimento/tamanho de uma *string*

exemplo07.py

```
1  greet = 'Hello'
2
3  # count length of greet string
4  print(len(greet))
```

Shell

```
>>> %Run exemplo07.py
  5
>>>
```

6. Exemplo de *strings* em Python

6.9 Teste da existência de uma substring

Podemos testar se uma *substring* existe ou não em uma *string* usando a palavra-chave **`in`**.

```
print('a' in 'program') # True
print('at' not in 'battle') # False
```

6. Exemplo de *strings* em Python

6.9 Teste da existência de uma substring

exemplo08.py

```
1 print('a' in 'program')
2 print('at' not in 'battle')
```

Shell

```
>>> %Run exemplo08.py

  True
  False

>>>
```

6. Exemplo de *strings* em Python

6.10 Busca de *substring*

- O método `find()` retorna o índice da primeira ocorrência da substring (se encontrada). Se não for encontrada, **retorna -1**.

- O método `index()` retorna o índice de uma substring dentro da string (se encontrada). Se a substring não for encontrada, ele **gera uma exceção**.

6. Exemplo de *strings* em Python

6.10 Busca de *substring*

exemplo09.py

```
1  string = "This string has a substring!"
2
3  print(string.find("substring"))
4  print(string.index("substring"))
```

Shell

```
>>> %Run exemplo09.py
  18
  18

>>>
```

6. Exemplo de *strings* em Python

6.10 Busca de *substring*

exemplo10.py

```
1  string = "This string has a substring!"
2  substr = "not_exist"
3  print("find:", string.find(substr))
4  print("index:", string.index(substr))
```

Shell

```
>>> %Run exemplo10.py
  find: -1
  Traceback (most recent call last):
    File "/home/pedro/Dropbox/UnB/APC/2023.1/T05/meus
  slides/Aula09/exemplo10.py", line 4, in <module>
      print("index:", string.index(substr))
  ValueError: substring not found

>>>
```

6. Exemplo de *strings* em Python

6.11 Substituição de *substring*

O método **`replace()`** substitui cada ocorrência correspondente de uma substring por outra string.
Sintaxe:

```
str.replace(old, new [, count])
```

- **old** - a antiga substring que queremos substituir
- **new** - a nova substring que substituirá a antiga substring
- **count** (opcional) - o número de vezes que você deseja substituir a antiga substring pela nova string

Nota: Se o parâmetro count não for especificado, o método `replace()` substituirá todas as ocorrências da antiga substring pela nova string.

uma s por outra string.

Sintaxe:

```
str.replace(old, new [, count])
```

6. Exemplo de *strings* em Python

6.11 Substituição de *substring*

exemplo12.py

```
1  string = "Multi str str str str!"
2  old = "str"
3  new = "NEW"
4  print("old:", string)
5  print("new:", string.replace(old, new))
```

Shell

```
>>> %Run exemplo12.py
  old: Multi str str str str!
  new: Multi NEW NEW NEW NEW!
>>>
```

6. Exemplo de *strings* em Python

6.12 Conversão capitular

- O método `upper()` converte todos os caracteres minúsculos em uma string em caracteres maiúsculos e retorna a nova string.
- O método `lower()` converte todos os caracteres maiúsculos em uma string em caracteres minúsculos e retorna a nova string.

6. Exemplo de *strings* em Python

6.12 Conversão capitular

exemplo13.py

```
1  title = "This is a Title Case String!"
2  low = title.lower()
3  up = title.upper()
4  print("title=", title)
5  print("low=", low)
6  print("up=", up)
```

Shell

```
>>> %Run exemplo13.py

  title -> This is a Title Case String!
  low -> this is a title case string!
  up -> THIS IS A TITLE CASE STRING!

>>>
```

6. Exemplo de *strings* em Python

6.13 Particionamento de *strings*

- O método `partition()` divide a string na primeira ocorrência da string de argumento e retorna uma tupla contendo a parte antes do separador, a string de argumento e a parte após o separador.
- O método `split()` divide uma string no separador especificado e retorna uma lista de substrings.

6. Exemplo de *strings* em Python

6.13 Particionamento de *strings*

exemplo14.py

```
1  cars = 'BMW-Tesla-Range Rover'
2
3  print("cars -> ", cars)
4  print("split -> ", cars.split('-'))
5  print("partition -> ", cars.partition('-'))
6
```

Shell

```
>>> %Run exemplo14.py

  cars ->  BMW-Tesla-Range Rover
  split ->  ['BMW', 'Tesla', 'Range Rover']
  partition ->  ('BMW', '-', 'Tesla-Range Rover')

>>>
```

6. Exemplo de *strings* em Python

6.14 Format *strings*

Existem pelo menos 3 formas de formatar strings em Python:

- Old-fashion style
- Método `String.format()`
- f-strings

Fonte: https://pyformat.info/

6. Exemplo de *strings* em Python

6.14 Format *strings*

Old

```
'%s %s' % ('one', 'two')
```

New

```
'{} {}'.format('one', 'two')
```

Output

```
one two
```

Fonte: https://pyformat.info/

6. Exemplo de *strings* em Python

6.14 Format *strings*

Old

```
'%d %d' % (1, 2)
```

New

```
'{} {}'.format(1, 2)
```

Output

```
1 2
```

Fonte: https://pyformat.info/

6. Exemplo de *strings* em Python

6.14 Format *strings*

This operation is not available with old-style formatting.

New

```
'{1} {0}'.format('one', 'two')
```

Output

```
two one
```

Fonte: https://pyformat.info/

6. Exemplo de *strings* em Python

6.14 Format *strings*

Old

```
'%10s' % ('test',)
```

New

```
'{:>10}'.format('test')
```

Output

```
      test
```

Fonte: https://pyformat.info/

6. Exemplo de *strings* em Python

6.14 Format *strings*

Old

```
'%-10s' % ('test',)
```

New

```
'{:10}'.format('test')
```

Output

```
test
```

Fonte: https://pyformat.info/

6. Exemplo de *strings* em Python

6.14 Format *strings*

This operation is not available with old-style formatting.

New

```
'{:_<10}'.format('test')
```

Output

```
t e s t _ _ _ _ _ _
```

Fonte: https://pyformat.info/

6. Exemplo de *strings* em Python

6.14 Format *strings*

This operation is not available with old-style formatting.

New

```
'{:^10}'.format('test')
```

Output

```
   test   
```

Fonte: https://pyformat.info/

6. Exemplo de *strings* em Python

6.14 Format *strings*

Old

```
'%d' % (42,)
```

New

```
'{:d}'.format(42)
```

Output

```
4 2
```

Fonte: https://pyformat.info/

6. Exemplo de *strings* em Python

6.14 Format *strings*

Old

```
'%f' % (3.141592653589793,)
```

New

```
'{:f}'.format(3.141592653589793)
```

Output

```
3.141593
```

Fonte: https://pyformat.info/

6. Exemplo de *strings* em Python

6.14 Format *strings*

Old

```
'%04d' % (42,)
```

New

```
'{:04d}'.format(42)
```

Output

```
0042
```

Fonte: https://pyformat.info/

6. Exemplo de *strings* em Python

6.14 Format *strings*

Old

```
'%06.2f' % (3.141592653589793,)
```

New

```
'{:06.2f}'.format(3.141592653589793)
```

Output

```
0 0 3 . 1 4
```

Fonte: https://pyformat.info/

6. Exemplo de *strings* em Python

6.14 Format *strings*

```
data = {'first': 'Hodor', 'last': 'Hodor!'}
```

Old

```
'%(first)s %(last)s' % data
```

New

```
'{first} {last}'.format(**data)
```

Output

```
H o d o r   H o d o r !
```

6. Exemplo de *strings* em Python

6.14 Format *strings*

New

```
'{first} {last}'.format(first='Hodor', last='Hodor!')
```

Output

```
H o d o r   H o d o r !
```

Fonte: https://pyformat.info/

7. Exercício

Escreva uma função que recebe como entrada um <nome> e imprime na tela do computador "Hola, <nome>" emoldurada conforme ilustrado abaixo. A largura da moldura deve adaptar-se de acordo com o tamanho da *string* de entrada. Por exemplo, se o nome for "Ivo", a largura do quadro impresso na tela será diferente da largura do quadro impresso para o nome "Maersalalhasbaz".

```
+-----------+
|           |
| Hola, Ivo |
|           |
+-----------+
```

```
+----------------------+
|                      |
| Hola, Maersalalhasbaz |
|                      |
+----------------------+
```


THANK
YOU
GRACIAS
ARIGATO
SHUKURIA
DANKSCHEEN
TASHAKKURATU
YAQHANYELAY
SUKSAMA
EKHMET
MEHRBANI
GRAZIE
MAAKE
KOMAPSUMNIDA
GOZAIMASHITA
EFCHARISTO
JUSPAXAR
PALDIES
BOLZÏN
MERCI
BÏYAN
SHUKRIA
TINGKI

Dúvidas?