Notas de Aula #08: Strings (Funções e Métodos)

String é uma das estruturas de dados primitivas. É constituída por uma coleção (cadeia) de caracteres, podendo ser tratadas na maioria das vezes como vetores.

LEN()

Retorna o tamanho (quantidade de caracteres) de uma *string*.

Sintaxe:

```
len (string)
```

No exemplo a seguir, solicitamos na linha 1 que o usuário informe o seu nome, onde em seguida na linha 14 será atribuído a variável qt_caracteres a o tamanho da *string* digitada na linha 1.

```
1 texto = input('Digite o seu nome: ')
2
3 qt_caracteres = len(texto)
4 print('O seu nome possui ', qt_caracteres, ' caracteres')
```

LOWER()

Retorna a *string* em letras minúsculas.

Sintaxe:

```
string.lower()
```

No exemplo a seguir, na linha 3 será impresso o conteúdo da variável texto em caracteres minúsculos (conforme exibido no comentário na mesma linha).

```
1 texto = '   OrdEm E Progresso   '
2
3 print(texto.lower())              # '   ordem e progresso   '
```

UPPER()

Retorna a *string* em letras maiúsculas.

Sintaxe:

```
string.upper()
```

No exemplo a seguir, na linha 3 será impresso o conteúdo da variável texto em caracteres maiúsculos (conforme exibido no comentário na mesma linha).

```
1 texto = '   OrdEm E Progresso   '
2
3 print(texto.upper())              # '   ORDEM E PROGRESSO   '
```

REPLACE()

Substitui uma *string* definida no argumento *string_antiga* por outra *string* definida no argumento *string_nova*.

O argumento *quantidade_ocorrencias* define a quantidade de substituições que deverão ser efetuadas. Não sendo informado serão efetuadas a troca de todas as ocorrências de *string_antiga* por *string_nova*.

Sintaxe:

```
string.replace(string_antiga, string_nova, quantidade_ocorrencias)
```

Vejamos o exemplo a seguir:

```
1 texto = 'OrdEm & ProgrEsso'
2
3 texto = texto.replace('&', 'E')
4 print(texto)              # 'OrdEm E ProgrEsso'
5
6 texto = texto.replace('E', 'e', 2)
7 print(texto)              # 'Ordem e ProgrEsso'
8
9 texto = texto.replace('E', 'e')
10 print(texto)             # 'Ordem e Progresso'
```

STRIP()

Remove os espaços em branco do início e do final de uma string.

Sintaxe:

```
string.strip()
```

No exemplo a seguir, na linha 4 será impresso o conteúdo da variável texto sem os espaços em branco do início e do final (conforme exibido no comentário na mesma linha).

```
1 texto = '   Ordem e Progresso   '
2
3 texto = texto.strip()
4 print(texto)              # 'Ordem e Progresso'
```

SPLIT()

Divide uma *string* e adiciona as *substrings* (partes de uma *string*) a uma lista de *strings*. Essa divisão toma como base um *separador* passado como argumento do método.

Se *separador não for especificado ou for None, um algoritmo de divisão diferente será aplicado: as execuções de espaço em branco consecutivo serão consideradas como um único separador, e o resultado não conterá strings vazias no início ou no final se a string tiver espaço em branco à esquerda ou à direita. Conseqüentemente, a divisão de uma string vazia ou de uma string consistindo em apenas espaços em branco com um separador None retorna [ ]* (lista vazia).

Sintaxe:

```
string.stplit(separador, máximo de ocorrências)
```

No exemplo a seguir, podemos observar que na linha 5 será impresso o conteúdo da lista texto_2 que irá conter uma lista com 3 posições, onde cada posição da lista é uma das palavras contidas na variável texto_1, pois como não foi informado o separador, o método split() assumiu como separador os espaços em branco contidos na *string*.

```
1 texto_1 = 'Ordem e Progresso'
2
3 texto_2 = texto_1.split()
4
5 print(texto_2)            # ['Ordem' , 'e', 'Progresso']
```

FORMAT()

Método utilizado para criar uma *string* através da substituição de campos chaves pelos argumentos do método format().

A depender do tipo de valor a ser formatado, o método format() possui parâmetros diferenciados.

Utilizando com Strings:

Estrutura base:

:[preencher][alinhar][largura].[precisão]

preencher	Qualquer caractere.
Alinhar	> ..... alinha a esquerda < ..... alinha a direita ^ ..... alinha ao centro
largura	Largura mínima do campo.
precisão	Largura máxima do campo.

Vejamos os exemplos a seguir.

```
1  texto = 'IFRN'
2
3  # alinha a direita com 30 espaços em branco
4  print('{0:>30}'.format(texto))
5
6  # alinha a direita com 30 símbolos.
7  print('{0:.>30}'.format(texto))
8
9  # alinha ao centro usando 15 espaços em branco a esquerda e 15 a direita
10 print('{0:^15}'.format(texto))
11
12 # imprime só as duas primeiras letras
13 print('{0:.2}'.format(texto))
```

Utilizando com Números:

Estrutura base:

:[preencher][alinhar][sinal][largura].[precisão][tipo]

preencher	Qualquer caractere.
alinhar	>..... alinha a esquerda <..... alinha a direita ^ ..... alinha ao centro
sinal	
largura	Largura mínima do campo.
precisão	Largura máxima do campo.
tipo	Define o tipo de saída.

A seguir temos os principais tipos de saída:

Tipo	Significado
d	Inteiro.
i	Inteiro.
f	Float.
F	Float.
o	Inteiros em octal.
x	Inteiros em hexadecimal minúscula.
X	Inteiros em hexadecimal maiúscula.
e	Float em formato exponencial minúsculo.
E	Float em formato exponencial maiúsculo.
g	Mesmo que “e” se expoente é maior do que -4 ou inferior a precisão, “f” de outra forma.
G	Mesmo que “E” se expoente é maior do que -4 ou inferior a precisão, “F” de outra forma.
c	Um único caractere.
s	String (converte qualquer objeto python usando str ()).
b	Inteiro em binário.

A seguir temos alguns exemplos. Note a explicação de cada um nos comentários no código.

```
valor = 135

# Imprime o conteúdo da variável valor inserindo espaços em branco a esquerda
# quando a quantidade de caracteres da variável for menor que 4
print('{0:4}'.format(valor))    #' 135'

# Imprime o conteúdo da variável valor inserindo espaços em branco a esquerda
# quando a quantidade de caracteres da variável for menor que 3
print('{0:3}'.format(valor))    #'135'

# Imprime o conteúdo da variável valor convertido em binário
print('{0:b}'.format(valor))    #'10000111'

pi = 3.141619

# Imprime o conteúdo da variável pi com 2 casas decimais
print('{0:.2f}'.format(pi))     #'3.14'

# Imprime o conteúdo da variável pi com 2 casas decimais e colocando o sinal
# de positivo (+)
print('{0:+.2f}'.format(pi))    #'+3.14'
```

A seguir temos um exemplo do método format() onde o mesmo recebe mais de um argumento.

```
print('Calculando a área de uma circunferência\n')

raio = input('Informe o Raio da Circunferência: ')

raio = float(raio)
area = 3.14 * raio ** 2

print('Área da circunferência\n')
print(' RAIO = {0:0.1f} | ÁREA = {1:4.2f}'.format(raio, area))
```

Trabalhando com substrings

"*Slice*" é uma maneira prática de se referir a sub-partes de seqüências - normalmente strings e listas.

Sintaxe:

```
string[posicao_inicial, posicao_final, step]
```

A seguir temos alguns exemplos. Note a explicação de cada um nos comentários no código.

```
texto = 'Instituto Federal de Educação, Ciência e Tecnologia do RN'

#Imprimindo a posição 16 da string
print('Exemplo 1: ' + texto[16] + '\n')

#Imprimindo da posição 8 até a posição 12
print('Exemplo 2: ' + texto[8:13] + '\n')

#Imprimindo da posição 8 em diante
print('Exemplo 3: ' + texto[8:] + '\n')

#Imprimindo até a posição 10
print('Exemplo 4: ' + texto[:10] + '\n')

#Imprimindo toda a string
print('Exemplo 5: '+ texto[:] + '\n')

#Imprimindo toda a string
print('Exemplo 6: ' + texto[:10] + texto [10:] + '\n')

#Imprimindo da posição 8 até a posição 12 saltando de 2 em 2 posições
print('Exemplo 7: ' + texto[8:13:2] + '\n')

#Imprimindo a última posição da string
print('Exemplo 8: ' + texto[-1] + '\n')

#Imprimindo a posição 7 a partir da última posição da string
print('Exemplo 9: ' + texto[-7] + '\n')

#Imprimindo da posição  até a posição 2 a partir da última posição
print('Exemplo 10: ' + texto[-7:-3] + '\n')

#Imprimindo da posição 18 até a posição 12
#invertendo a ordem de impressão
print('Exemplo 11: ' + texto[18:13:-1] + '\n')
```