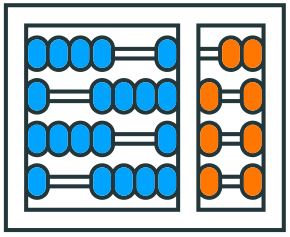

Instituto de Computação

UNIVERSIDADE ESTADUAL DE CAMPINAS

UNICAMP

MC102 – Aula 06

Strings

Algoritmos e Programação de Computadores

Zanoni Dias

2020

Instituto de Computação

Roteiro

Strings

Formatação de Strings

Operações, Funções e Métodos

Exercícios

Strings

Strings

- Uma String é uma sequência de caracteres.
- Em Python, Strings são representadas como listas imutáveis de caracteres.
- Podemos representar uma String como sequências de caracteres entre aspas simples (') ou aspas duplas (").
- Exemplo:

```
1 msg = "hello world"
2 print(msg)
3 # hello world
```

Caracteres Especiais

- O seguinte trecho de código apresenta erros.

```
print("Você respondeu "SIM".")
# SyntaxError: invalid syntax
print("\")
# SyntaxError: EOL while scanning string literal
```

- Isso acontece porque " e \ são caracteres reservados da linguagem.
- Para representar os caracteres " e \ precisamos utilizar o seguinte código.

```
print("Você respondeu \"SIM\".")
# Você respondeu "SIM".
print("\\")
# \
```

Caracteres Especiais

- Outros exemplos:

```
print("Joe\'s Car")
# Joe's Car
print("Quebra de\nLinha")
# Quebra de
# Linha
print("\tTabulação")
#         Tabulação
```

- Por padrão, um <tab> (`\t`) tem tamanho equivalente a oito espaços em branco.

Acessando Elementos de uma String

- Podemos acessar elementos de uma String como acessamos elementos de uma lista.

```
msg = "hello world"
print(msg[0])
# h
print(msg[1])
# e
print(msg[-1])
# d
print(msg[-5])
# w
print(msg[12])
# IndexError: string index out of range
```

Acessando Elementos de uma String

- Como Strings são listas imutáveis (assim como as tuplas), não é possível alterar uma posição da String.

```
1 msg = "hello world"
2 msg[0] = "y"
3 # TypeError: 'str' object does not support item assignment
```

Acessando Elementos de uma String

- Também podemos selecionar um trecho de uma String utilizando `string[start:stop:step]`.
- O trecho inicia na posição `start` (inclusive) e vai até a posição `stop` (exclusive), selecionando de `step` em `step` caracteres.
- Caso o parâmetro `step` não seja especificado, Python utilizará o valor `1` como padrão (assim como em listas ou tuplas).

```
1  msg = "hello world"
2  print(msg[3:8])
3  # lo wo
4  print(msg[:5])
5  # hello
6  print(msg[6:])
7  # world
8  print(msg[::2])
9  # hlowrd
10 print(msg[::-1])
11 # dlrow olleh
```

Formatação de Strings

Formatação de Strings

- Uma das formas de formatar uma String é utilizando a função `format`.
- A função `format` recebe como parâmetros um valor e uma string com a formatação desejada.
- Como resposta a função retorna uma string formatada.
- A string de formatação possui uma especificação para cada tipo de dados.
- Especificação completa em: https://docs.python.org/3/library/string.html#formatspec
- Iremos focar na formatação valores do tipo inteiro (`int`) e real (`float`).

Formatação de Strings

- Formatando um número inteiro:

```
print(format(10, "d"))
# 10
```

- Formatando um número inteiro com sinais:

```
print(format(13, "+d"))
# +13
print(format(-7, "+d"))
# -7
```

Formatação de Strings

- Formatando um número real:

```
print(format(3.14159265359, "f"))
# 3.141593
```

- Formatando um número real com sinais:

```
print(format(3.14159265359, "+f"))
# +3.141593
```

- Formatando um número real com sinais e precisão:

```
print(format(3.14159265359, "+.10f"))
# +3.1415926536
```

Formatação de Strings

- Outra forma de formatar uma String é utilizando o método `format`.
- O método `format` gera uma nova String como resposta.
- O método recebe como parâmetros um sequência de valores que são utilizados para criar a String no formato desejado.
- O método `format` também segue uma especificação para formação de Strings.
- Especificação completa em: https://docs.python.org/3/library/string.html#formatstrings

Formatação de Strings

- Formatação com Strings:

```
frutas = "Frutas: {0}, {1} e {2}"
print(frutas.format("abacaxi", "banana", "caqui"))
# Frutas: abacaxi, banana e caqui
pets = "Quem é mais inteligente: {1} ou {0}?"
print(pets.format("gato", "cachorro"))
# Quem é mais inteligente: cachorro ou gato?
```

Formatação de Strings

- Formatação com número inteiros:

```
1 soma = "{0} + {1} = {2}"
2 print(soma.format(3, 4, 3 + 4))
3 # 3 + 4 = 7
4 valores = "Valor mínimo/médio/máximo: {0}/{1}/{2}"
5 print(valores.format(10, 35, 100))
6 # Valor mínimo/médio/máximo: 10/35/100
```

Formatação de Strings

- Formatação com número reais:

```
pi = "O valor de pi é: {0:.4f}"
print(pi.format(3.14159265359))
# O valor de pi é: 3.1416
notas = "A média das notas da turma foi {0:.2f}."
print(notas.format(8.7525))
# A média das notas da turma foi 8.75.
```

Formatação de Strings

- Formatação com vários tipos de dados:

```
cabeçalho = "{0}, {1} de {2} de {3}"
print(cabeçalho.format("Campinas", 7, "maio", 2020))
# Campinas, 7 de maio de 2020
temperatura = "{0:02d}/{1:02d}/{2}: {3:.1f}C"
print(temperatura.format(7, 5, 2020, 28.765))
# 07/05/2020: 28.8C
```

Operações, Funções e Métodos

Concatenação de Strings

- O operador `+` concatena duas Strings.

```
msg = "hello"
msg2 = "y" + msg[1:5] + "w"
print(msg2)
# yellow
```

- O operador `*` faz concatenações múltiplas da mesma String.

```
s = "abc"
print(s * 3)
# abcabcabc
```

Tamanho de uma String

- A função `len` retorna o tamanho (quantidade de caracteres) de uma String.

```
msg = "hello"
print(len(msg))
# 5
msg2 = "Hello World"
print(len(msg2))
# 11
msg3 = "Hello\nWorld"
print(len(msg3))
# 11
```

- Observe que qualquer tipo de caractere é contado pela função `len`, inclusive espaços, quebra de linhas ou tabulações.

Comparação de Strings

- O operador `==` verifica se duas Strings são iguais.
- O operador `!=` verifica se duas Strings são diferentes.

```
a = "Python"
b = "Py" + "thon"
c = "p" + "ython"
print(a == b)
# True
print(a == c)
# False
print(b != c)
# True
```

- O operador `in` verifica se uma String é parte de outra String.

```
print("thon" in "Python")
# True
print("thor" in "Python")
# False
```

Comparação de Strings

- O método **`startswith`** verifica se a String recebida como parâmetro é um prefixo da String base.

```
msg = "Hello World"
print(msg.startswith("Hello"))
# True
print(msg.startswith("World"))
# False
```

Buscando uma String

- O método `index` retorna a primeira posição em que uma String fornecida como parâmetro ocorre na String base.
- Se a String fornecida como parâmetro não está contida na String base, então é gerado um erro (similar ao que ocorre com listas).

```
1 bond = "My name is Bond, James Bond"
2 print(bond.index("Bond"))
3 # 11
4 msg = "Hello World"
5 print(msg.index("World"))
6 # 6
7 print(msg.index("Bond"))
8 # ValueError: substring not found
```

Buscando uma String

- O método `find` também retorna a primeira posição em que uma String fornecida como parâmetro ocorre na String base.
- Se a String fornecida como parâmetro não está contida na String base, então é retornado o valor `-1`.

```
1 bond = "My name is Bond, James Bond"
2 print(bond.find("Bond"))
3 # 11
4 msg = "Hello World"
5 print(msg.find("World"))
6 # 6
7 print(msg.find("Bond"))
8 # -1
```

Manipulação de Strings

- O método **`strip`** remove todos os espaços em branco (incluindo quebras de linhas e tabulações) no início e no fim da String.

```
msg = " \n  Hello World \t"
print(msg.strip())
# Hello World
```

Manipulação de Strings

- O método `split` divide uma String em uma lista de acordo com um padrão de caracteres (separador).
- Por padrão, o separador é igual a qualquer sequência de espaços em branco (incluindo quebras de linhas e tabulações).

```
1 str1 = "  MC102  Algoritmos\t\tProgramação\nComputadores  "
2 dados = str1.split()
3 print(dados)
4 # ['MC102', 'Algoritmos', 'Programação', 'Computadores']
5 str2 = "abacaxi, banana, caqui, damasco"
6 frutas = str2.split(", ")
7 print(frutas)
8 # ['abacaxi', 'banana', 'caqui', 'damasco']
```

Manipulação de Strings

- O método `join` junta uma lista de Strings usando a String base como concatenador.

```
frutas = ['abacaxi', 'banana', 'caqui', 'damasco']
txt = ", ".join(frutas)
print(txt)
# abacaxi, banana, caqui, damasco
```

- A função `list()` pode ser utilizada para transformar uma String em uma lista de caracteres.

```
str = "aeiou"
lista = list(str)
print(lista)
# ['a', 'e', 'i', 'o', 'u']
```

Manipulação de Strings

- O método **`replace`** cria uma nova String onde todas as ocorrências de um padrão de caracteres numa String dada são trocas por outro.

```
x = "Algoritmos e Programação de Computadores"
y = x.replace("a", "_")
print(y)
# Algoritmos e Progr_m_ção de Comput_dores
y = x.replace("Algoritmos", "$" * len("Algoritmos"))
print(y)
# $$$$$$$$$ e Programação de Computadores
x = "a,b,c,d,e"
y = x.replace(",", "")
print(y)
# abcde
```

Outros Métodos

- `capitalize()`: converte o primeiro caractere para maiúsculo.

```
print("meu teste".capitalize())
# Meu teste
```

- `lower()`: converte a String para letras minúsculas.

```
print("Meu TESTE".lower())
# meu teste
```

- `upper()`: converte a String para letras maiúsculas.

```
print("mEU tESte".upper())
# MEU TESTE
```

Outros Métodos

- `isnumeric()`: testa se todos os caracteres são dígitos.

```
print("1234".isnumeric())
# True
print("teste123".isnumeric())
# False
```

- `isalpha()`: testa se todos os caracteres são letras.

```
print("MeuTeste".isalpha())
# True
print("teste123".isalpha())
# False
```

- `isalnum()`: testa se todos os caracteres são letras ou dígitos.

```
print("teste123".isalnum())
# True
print("Meu teste".isalnum())
# False
```

Exemplo

- Exemplo:

```
1  sc = input("Entre com uma sequência de caracteres: ")
2
3  if sc.isalpha():
4    print(sc, "possui apenas letras")
5  elif sc.isnumeric():
6    print(sc, "possui apenas dígitos")
7  elif sc.isalnum():
8    print(sc, "possui letras e dígitos")
9  else:
10   print(sc, "não possui apenas letras e dígitos")
```

Laços e Strings

- Podemos utilizar o comando `for` para percorrer uma String.
- Exemplo:

```
s = "abc"
for c in s:
  print(c)
# a
# b
# c
```

- Outro exemplo:

```
for c in "Algoritmos":
  if c in "AEIOUaeiou":
    print("A String possui a vogal:", c)
# A String possui a vogal: A
# A String possui a vogal: o
# A String possui a vogal: i
# A String possui a vogal: o
```

Exercícios

Exercícios

1. Escreva um programa que, dada uma sequência de números inteiros (todos fornecidos na mesma linha, separados por espaços), imprima a média desses números.
2. Escreva um programa que, dada uma String representando um texto, imprima o número de palavras existentes. Observação: você deve remover os sinais de pontuação (“.”, “,”, “:”, “;”, “!” e “?”) antes de realizar a contagem das palavras.

Exercícios

3. Escreva um programa que, dada uma String `texto` e uma String `palavra`, ache todas as posições de ocorrência da `palavra` no `texto`. O seu programa deve desconsiderar se as letras são maiúsculas ou minúsculas.
4. Um palíndromo é uma palavra ou frase que pode ser lida da mesma forma tanto da esquerda para a direita como da direita para a esquerda (desconsiderando os espaços em branco). Considere que a entrada não possui sinais de pontuação ou acentos. Escreva um programa que, dada uma String, verifique se ela é um palíndromo.

Exercício 1

1. Escreva um programa que, dada uma sequência de números inteiros (todos fornecidos na mesma linha, separados por espaços), imprima a média desses números.

```
texto = input("Entre com uma sequência de números: ")
numeros = texto.split()
soma = 0

for n in numeros:
  soma = soma + int(n)
media = soma / len(numeros)

print("A média é: ", format(media, ".2f"))
```

Exercício 2

2. Escreva um programa que, dada uma String representando um texto, imprima o número de palavras existentes. Observação: você deve remover os sinais de pontuação (“.”, “,”, “:”, “;”, “!” e “?”) antes de realizar a contagem das palavras.

```
1 texto = input("Entre com um texto: ")
2 pontuacao = [".", ",", ":", ";", "!", "?"]
3
4 # remove os sinais de pontuação
5 for p in pontuacao:
6   texto = texto.replace(p, " ")
7
8 numPalavras = len(texto.split())
9 print("Número de palavras:", numPalavras)
```

Exercício 3

3. Escreva um programa que, dada uma String **texto** e uma String **palavra**, ache todas as posições de ocorrência da **palavra** no **texto**. O seu programa deve desconsiderar se as letras são maiúsculas ou minúsculas.

```
1  texto = input("Entre com um texto: ")
2  palavra = input("Entre com uma palavra: ")
3  texto = texto.lower()
4  palavra = palavra.lower()
5
6  removido = 0
7  while palavra in texto:
8    posicao = texto.find(palavra)
9    print(removido + posicao)
10   texto = texto[posicao + 1:]
11   removido = removido + (posicao + 1)
```

Exercício 4

4. Um palíndromo é uma palavra ou frase que pode ser lida da mesma forma tanto da esquerda para a direita como da direita para a esquerda (desconsiderando os espaços em branco). Considere que a entrada não possui sinais de pontuação ou acentos. Escreva um programa que, dada uma String, verifique se ela é um palíndromo.

```
1  txt = input("Entre com uma palavra ou frase: ")
2  txt = txt.lower()
3  txt = txt.replace(" ", "")
4
5  i = 0
6  j = len(txt) - 1
7
8  palindromo = True
9
10 ...
```

Exercício 4 - Continuação

```
...

while i < j:
  if txt[i] != txt[j]:
    palindromo = False
    break
  else:
    i = i + 1
    j = j - 1

if palindromo:
  print("Palíndromo")
else:
  print("Não palíndromo")
```

Exercício 4

4. Um palíndromo é uma palavra ou frase que pode ser lida da mesma forma tanto da esquerda para a direita como da direita para a esquerda (desconsiderando os espaços em branco). Considere que a entrada não possui sinais de pontuação ou acentos. Escreva um programa que, dada uma String, verifique se ela é um palíndromo.

```
1 txt = input("Entre com uma palavra ou frase: ")
2 txt = txt.lower()
3 txt = txt.replace(" ", "")
4
5 if txt == txt[::-1]:
6   print("Palindromo")
7 else:
8   print("Nao´palindromo")
```