Strings
Algoritmos e Programação

Curso Integrado

Strings

- Uma string (=tipo str) é uma sequência de caracteres
- Em Python, podemos representar uma string colando-a entre aspas ou apóstrofes
- O uso desses 2 marcadores é útil quando queremos utilizar um deles dentro do string, como:

```
texto = "Esse string usa 'aspas' como demarcador"
print(texto)          Esse string usa 'aspas' como demarcador

novo_texto = 'Esse string usa "apóstrofes" como
demarcador'
print(novo_texto)     Esse string usa "apóstrofes" como demarcador
```

Strings

- Uma cadeia que não possui nenhum caractere, muitas vezes chamada de **string vazia**, é também considerado um string

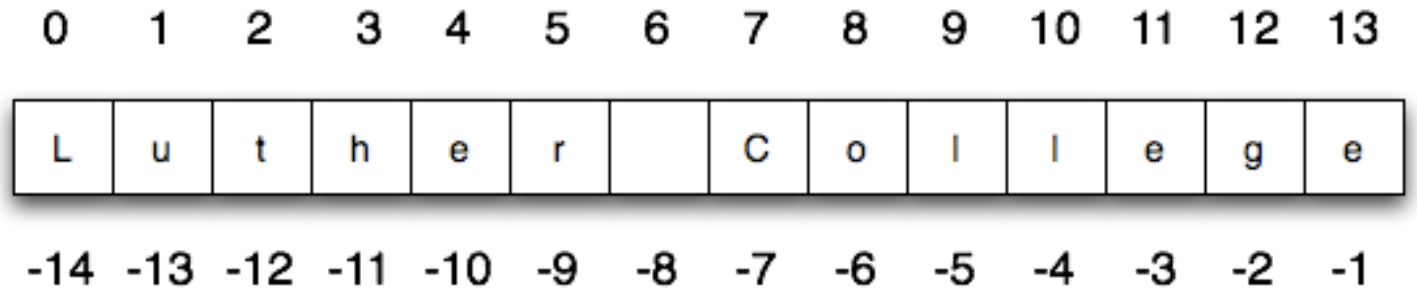
- É simplesmente uma sequência com zero caracteres e é representado por ‘’ ou “” (dois apóstrofes ou aspas sem nada entre eles)

- Uma string com um ou mais caracteres em branco, como ‘ ’ é diferente do string vazio ‘’

Strings: criação, indexação e comparação

- O **operador de indexação** seleciona um único caractere de uma string

- Os caracteres são acessados por sua posição ou valor do índice. Por exemplo, na sequência mostrada abaixo, os 14 caracteres são indexados da esquerda para a direita a partir posição 0 até a posição 13



Strings: criação, indexação e comparação

- As posições podem ser acessadas também da direita para a esquerda usando números negativos, onde -1 é o índice mais à direita e assim por diante

- Python usa índices negativos para percorrer a sequência da direita para a esquerda (do fim para o começo)

- Portanto, índices negativos não são inválidos

Strings: criação, indexação e comparação

- Um índice é inválido apenas quando ele cai fora da sequência, que pode ser um número positivo ou negativo maior que o tamanho da sequência

- Para uma fatia começando do primeiro caractere (índice zero), não é necessário escrever o zero explicitamente, simplificando assim a notação para fatia = texto[:5] para o intervalo [0:5]

- De forma semelhante, não é necessário escrever o valor do último índice para uma fatia que vá até o final, como por exemplo fatia = texto[-6:] para definir uma fatia com os 6 últimos caracteres da string texto

Strings: semelhanças com Listas

- Assim como listas:
  - um caractere de uma string pode ser acessado por meio de índice
  - o comprimento pode ser dado pela função len()
  - os caracteres de uma string podem ser fatiados e concatenados

Strings: semelhanças com Listas

- Exemplo de concatenação:

```
frase = ""   # string vazio
frase = frase + "Esse string usa "
meio  = '"apóstrofes" '
fim   = "como demarcador. "
frase = frase + meio + fim
print("A frase: ", frase)
print("Tem comprimento: ", len(frase))
```

Strings NÃO SÃO listas

- Apesar de serem criados usando um par de ” (aspas) ou ‘ (apóstrofes) como demarcadores, a função *len()*, a concatenação, a indexação e o fatiamento de strings são realizados de forma semelhante às listas
- A grande diferença é que strings são **imutáveis**, ou seja, não é possível alterar os seus elementos
- Teste o exemplo:

```
texto = "Esse texto é uma 'string'"
print(texto)
texto[2] = 'S'
print(texto)
```

Strings não são listas

- Como strings são IMUTÁVEIS, caso você precise de uma string diferente, é necessário criar uma nova utilizando, caso desejar, fatias (partes) de outras strings

- Exemplo:

```
texto = "Esse é uma 'string'"
print(texto)
novo_texto = 'Mais' + texto[6:]
print(novo_texto)
```

Desempacotamento de strings

- Cadeias em caracteres:

```
a,b,c,d,e = "Hello"
print(a)
print(b)
print(c)
print(d)
print(e)
```

Métodos de Strings

- Strings também são objetos
- Cada instância de string tem seus próprios atributos e métodos
- O atributo mais importante do string é a sua coleção de caracteres
- Há uma grande variedade de métodos
- Teste o exemplo:

```
ss = "Hello, World"
print(ss.upper())
tt = ss.lower()
print(tt)
```

Métodos de Strings

- Outros métodos para você testar:

Método	Parâmetros	Descrição
upper	nenhum	Retorna um string todo em maiúsculas
lower	nenhum	Retorna um string todo em minúsculas
capitalize	nenhum	Retorna um string com o primeiro caractere em maiúscula, e o resto em minúsculas
strip	nenhum	Retorna um string removendo caracteres em branco do início e do fim
lstrip	nenhum	Retorna um string removendo caracteres em brando do início
rstrip	nenhum	Retorna um string removendo caracteres em brando do fim
count	item	Retorna o número de ocorrências de item
replace	old, new	Substitui todas as ocorrências do substring old por new
center	largura	Retorna um string centrado em um campo de tamanho largura
ljust	largura	Retorna um string justificado à esquerda em um campo de tamanho largura
rjust	largura	Retorna um string justificado à direita em um campo de tamanho largura
find	item	Retorna o índice mais à esquerda onde o substring item é encontrado
rfind	item	Retorna o índice mais à direita onde o substring item é encontrado
index	item	Como find, mas causa um erro em tempo de execução caso item não seja encontrado
rindex	item	Como rfind, mas causa um erro em tempo de execução caso item não seja encontrado

Métodos de Strings

- Teste o exemplo:

```
ss = "    Hello, World    "

els = ss.count("l")
print(els)

print("***"+ss.strip()+"***")
print("***"+ss.lstrip()+"***")
print("***"+ss.rstrip()+"***")

news = ss.replace("o", "***")
print(news)
```

Métodos de Strings

- Outro exemplo:

```
food = "banana bread"
print(food.capitalize())

print("*"+food.center(25)+"*")
print("*"+food.ljust(25)+"*")
print("*" +food.rjust(25)+"*")

print(food.find("e"))
print(food.find("na"))
print(food.find("b"))

print(food.rfind("e"))
print(food.rfind("na"))
print(food.rfind("b"))

print(food.index("e"))
```

Métodos de Strings

- Existem ainda muitos outros métodos úteis: isalnum(), isalpha(), isdigit(), islower(), isspace(), isupper(), lstrip(), replace(), strip(), entre outros → Faça os testes!!!!

- Teste com os valores Araraquara e depois 1234:

```
teste_str=input("Digite um valor: ")
print(teste_str, " capitalize",teste_str.capitalize())
print(teste_str," center",teste_str.center(20))
print(teste_str," count",teste_str.count("a"))
print(teste_str," find",teste_str.find("ara"))
print(teste_str," isalnum",teste_str.isalnum())
print(teste_str," isalpha",teste_str.isalpha())
print(teste_str," isdigit",teste_str.isdigit())
print(teste_str," replace",teste_str.replace("a","o"))
```

Métodos de Strings

- Faça o teste:

```
str_teste = 'Uma string é uma cadeia de caracteres.'
print(str_teste)
tam=len(str_teste)
print('Tamanho:',tam)
str_teste = str_teste.replace('uma', 'alguma')
print('Substituição:',str_teste)
quantoserres=str_teste.count('r')
print('Quantidade da letra "r": ',quantoserres)
posicaodoerre=str_teste.find('r')
print("Posição do primeiro 'r':",posicaodoerre)
```

Métodos de Strings

- Faça o teste:

```
quebra_da_string=str_teste.split()
print("Quebra de string:", quebra_da_string)
quebra_no_meio=str_teste.split('é')
print("Quebra da string na letra 'é':",quebra_no_meio)
juntando_string=' deve ser '.join(str_teste.split('é'))
print("Quebrando e juntando:", juntando_string)
print("Todas maiúsculas:",str_teste.upper())
print("Todas minúsculas:",str_teste.lower())
print("Só 1ª maiúscula:",str_teste.lower().capitalize())
print("Todas as iniciais maiúsculas:",str_teste.title())
print("Inversão de caracteres:",str_teste.swapcase())
```

Outros Métodos de Strings

- Faça o teste:

```
print("Todas são MAIÚSCULAS?",'MAIÚSCULAS'.isupper())
print("Todas são MAIúscULAS?",'MAIúscULAS'.isupper())
print("Todas são minúsculas?",'minúsculas'.islower())
print("Todas são Minúsculas?",'Minúsculas'.islower())
print("A sequência é alfa-númerica?",'abcdef224466'.isalnum())
print("A sequência é alfa-númerica?",'abcdef$224466'.isalnum())
print("A sequência contém somente letras?",'abcdef'.isalpha())
print("A sequência contém somente letras?",'abcdef24'.isalpha())
print("A sequência contém somente dígitos?",'224466'.isdigit())
print("A sequência contém somente dígitos?",'224466 Progressão
Aritmética'.isdigit())
```

Outros Métodos de Strings

- Faça o teste:

```
print("A sequência contém somente espaços?','   '.isspace())
print("A sequência contém somente espaços?','''.isspace())
print("Incluindo espaços à direita:",'A string.'.ljust(15),"----")
print("Incluindo espaços à esquerda:",'A string.'.rjust(15),"----")
print("Centralizando a string:",'A string.'.center(15),"----")
s = "-"
seq = ("abacaxi", "banana", "caqui")
print (s.join( seq ))
```

Comparação de Strings

- Os operadores de comparação também funcionam com strings
- Para ver se duas strings são iguais, basta escrever uma expressão booleana usando o operador de igualdade. Exemplo:

```
word = "banana"
if word == "banana":
    print("Yes, we have bananas!")
else:
    print("Yes, we have NO bananas!")
```

- Outras operações de comparação são úteis para colocar palavras em *ordem lexicográfica,* isto é semelhante à ordem alfabética usada em um dicionário, exceto que todas as letras maiúsculas vêm antes de todas as letras minúsculas.

Varredura com for por item

- Um grande número de computações envolvem o processamento de item de um conjunto de cada vez

- Para strings, isto significa processar um caractere de cada vez, onde o processo envolve: começar do início, selecionar um caractere de cada vez, fazer alguma coisa com ele, e continuar até o final. Este padrão de processamento é chamado um percurso
- Anteriormente foi visto que o comando for pode iterar sobre os itens de uma sequência, por exemplo, em uma lista de nomes

Varredura com for por item

- Exemplo:

```
for um_nome in ["Joe", "Amy", "Brad", "Angelina",
"Zuki", "Thandi", "Paris"]:
    convite = "Oi " + um_nome + ".  Venha para a
festa nesse sabado!"
    print(convite)
```

- Lembre-se que a variável do laço um_nome assume cada valor da sequência de nomes
- O corpo é executado uma vez para cada nome

Varredura com for por item

- O mesmo processamento era verdade para a sequência de números inteiros criada pela função range. Exemplo:

```
for um_valor in range(10):
    print(um_valor)
```

- Como uma string é simplesmente uma sequência de caracteres, o laço for itera sobre cada caractere automaticamente → Iteração por item:

```
for um_char in "Venha para a festa":
    print(um_char)
```

Varredura com for por índice

- Também é possível utilizar a função range para gerar sistematicamente os índices dos caracteres
- O laço for pode então ser usado para iterar sobre essas posições
- Estas posições podem ser utilizados em conjunto com o operador de indexação para acessar os caracteres individuais na sequência. Exemplo:

```
fruta = "apple"
for idx in range(5):
    currentChar = fruta[idx]
    print(currentChar)
```

Varredura com for por índice

- A fim de tornar a iteração mais genérica, pode-se usar a função len para fornecer o limite para range
- Este é um padrão muito comum para percorrer qualquer sequência por posição. Exemplo:

```
fruta = "apple"
for idx in range(len(fruta)):
    print(fruta[idx])
```

Varredura com while

- O laço while também pode controlar a geração dos valores de índices
- Nesse caso o programador é responsável por configurar a condição inicial, certificando-se que a condição é correta e certificando-se de que algo muda dentro do corpo para garantir que a condição se tornará falsa e a repetição acabará

```
fruta = "apple"
position = 0
while position < len(fruta):
    print(fruta[position])
    position = position + 1
```

Operadores in e not in

- um string é um substring de si mesmo, e o string vazio é um substring de qualquer outro string

- O operador not in retorna o resultado lógico oposto de in

- Teste o exemplo:

```
print('a' in 'a')
print('apple' in 'apple')
print('' in 'a')
print('' in 'apple')
```

Operadores in e not in

- Outro exemplo: dada uma palavra, retirar suas vogais e mostra-la na tela

```
palavra=input("Digite uma palavra: ")
vogais = "aeiouAEIOU"
str_sem_vogais = ""
for cada_letra in palavra:
    if cada_letra not in vogais:
        str_sem_vogais = str_sem_vogais + cada_letra
```

Repetições e Contagens

- Neste exemplo o programa conta o número de vezes que uma letra em particular aparece em uma string:

```
palavra=input("Digite uma palavra: ")
caracter=input("Digite um caracter a ser contado: ")
contador = 0
for letra in palavra:
    if letra == caracter:
            contador = contador + 1
print("A letra %s aparece %d vezes na palavra %s" % (caracter,
contador, palavra))
```

Classificação de Caracteres

- Muitas vezes, é útil examinar um caractere e testar se ele é maiúsculo ou minúsculo, ou se é uma letra ou um dígito
- O módulo string fornece várias constantes que são úteis para esses fins
- Um exemplo é a constante string.digits é equivalente a “0123456789”, que pode ser usada para verificar se um caractere é um dígito usando o operador in
- A constante string.ascii_lowercase contém todas as letras ASCII que o sistema considera serem minúsculas
- Da mesma forma, string.ascii_uppercase contém todas as letras maiúsculas e string.punctuation contem todos os caracteres considerados símbolos de pontuação

Classificação de Caracteres

- Teste o código para verificar o conteúdo das constantes:

```
import string
print(string.ascii_lowercase)
print(string.ascii_uppercase)
print(string.digits)
print(string.punctuation)
```

Classificação de Caracteres

- O código abaixo verifica se o valor digitado é um número inteiro positivo e ilustra o uso da constante string.digits:

```
Import string
nro_ok=False
while not nro_ok:
    nro=input("Digite um número inteiro positivo: ")
    nro_ok=True
    for i in nro:
        if i not in string.digits:
            nro_ok=False
    if not nro_ok:
        print("O valor digitado não é um número inteiro
positivo!")
nro=int(nro) #faz a conversão
print("O valor digitado é: ",nro)
```

Classificação de Caracteres

- O código abaixo verifica se o valor digitado é um número inteiro positivo utilizando o método isdigit():

```
nro_ok=False
while not nro_ok:
    nro=input("Digite um número inteiro positivo: ")
    nro_ok=nro.isdigit()
    if not nro_ok:
        print("O valor digitado não é um número inteiro positivo!")

nro=int(nro) #faz a conversão
print("O valor digitado é: ",nro)
```

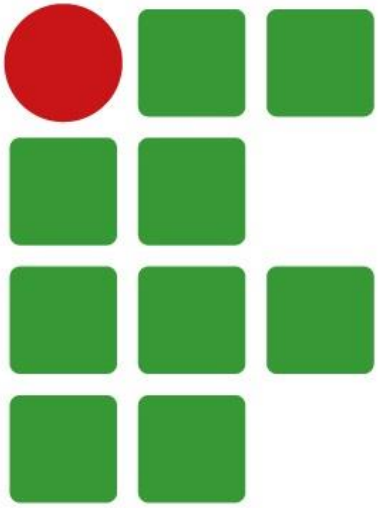
Exercícios

1. Escreva um programa que remove a primeira ocorrência de uma letra de uma string. A string e a letra devem ser fornecidas pelo usuário.
2. Escreva um programa que remove todas as ocorrências de uma letra de uma string. A string e a letra devem ser fornecidas pelo usuário.
3. Escreva um programa que verifique se duas strings fornecidas pelo usuário são iguais e mostre o total de caracteres de cada uma delas. Diferencie letras maiúsculas das minúsculas.
4. Escreva um programa que reconhece se uma string é um palíndromo. Exemplo: arara, ovo, reter.
5. Faça um programa que recebe uma frase e retorna o número de palavras que a frase contém.
6. Faça um programa que solicite o nome do usuário e imprima-o na vertical e em formato de escada. Ex.:

```
F
FU
FUL
FULA
FULAN
FULANO
```

Exercícios

7. Faça um programa que permita ao usuário digitar o seu nome e em seguida o mostre de trás para frente utilizando somente letras maiúsculas.
8. Dada uma string com uma frase informada pelo usuário (incluindo espaços em branco), conte a quantidade de espaços em branco e a quantidade de vezes que aparecem as vogais a, e, i, o, u.
9. Um anagrama é uma palavra que é feita a partir da transposição das letras de outra palavra ou frase. Por exemplo, “Iracema” é um anagrama para “America”. Escreva um programa que decida se uma string é um anagrama de outra string, ignorando os espaços em branco. O programa deve considerar maiúsculas e minúsculas como sendo caracteres iguais, ou seja, “a” = “A”.
10. Escreva um programa que solicite ao usuário a entrada de um número inteiro positivo ou negativo e mostre a quantidade de dígitos desse número.



INSTITUTO FEDERAL

São Paulo

Câmpus São Carlos