Algoritmos e Lógica de Programação

80 horas // 4 h/semana

Strings ou cadeias de Caracteres

Aula 8

Prof. Piva

Para começar...

- *Strings* são cadeias de caracteres que armazenam dados textuais e, portanto, podem armazenar informações para as mais diversas finalidades.
- O conteúdo de uma *string* pode representar um fato em si, ou uma informação. Por exemplo, se uma *string* armazena um valor igual a 120, isso é um dado, que somente será entendido conhecendo seu contexto.
- Agora, se uma String armazena a frase: "Neste mês vendemos 120 motores, e isso foi muito bom, pois significou um aumento de 30% nas vendas desse produto, em relação ao mesmo período do ano passado.". Temos, então, uma informação armazenada na **string**.

Para começar...

- Podemos, a partir desses exemplos simples, imaginar a importância desse tipo de variável *Strings*, na construção de algoritmos ou programas de computadores.
- A partir do estabelecimento de relações entre dados, contidos em *strings*, podemos gerar informação, que por sua vez poderá, a partir de outros relacionamentos, gerar conhecimento.
- A maioria dos mecanismos de busca, que conhecemos na Internet, funcionam manipulando extensas cadeias de caracteres, contidas em milhares de bases de dados nessa rede.
- Essas cadeias de caracteres, contidas em textos curtos ou longos, são denominadas *strings*.

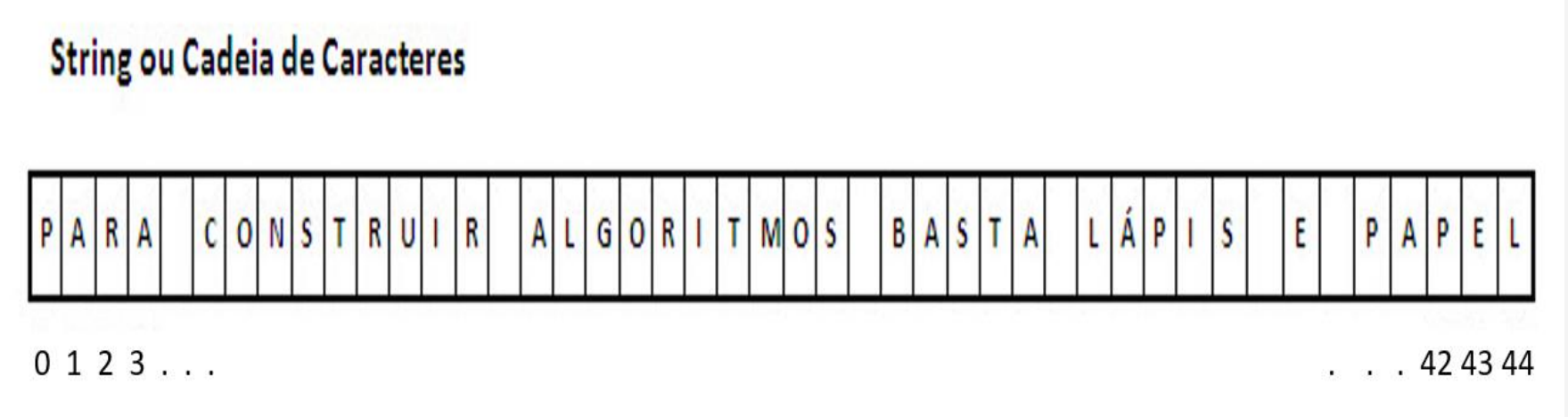
Strings

- Vamos aprender a manipular variáveis do tipo *string*, atribuindo valores a mesma, e recuperando seu conteúdo, usando linguagem algorítmica ou de programação de computador.
- Dependendo da linguagem de programação, *string* pode ser um tipo de dado primitivo, uma classe, ou mesmo um tipo criado pelo programador.
- A *string* referencia como “cadeia de caracteres” e, portanto, poder ser visualizada como uma lista linear ou vetor.
- Cada elemento da string é um caractere, e o agrupamento deles irá representar um dado ou uma informação.

Strings

- Em *strings*, os caracteres são armazenados da esquerda para a direita.

  Exemplo de uma *string* (uma frase) contendo 45 posições:



Variável String

- Matriz unidimensional (vetor) do tipo char (objeto)
- cada caractere de um string pode ser acessado individualmente
- vetor de tamanho n → posição varia de 0 a n-1

Ex:

```
s1 = “Exemplo1”
s2 = “123”
print(s1[0])
print(s2[2])
```

Saída:
E
3

Lendo Strings

- input() → a função input retorna um objeto do tipo string (sempre). Para obter outro tipo de objeto, temos que fazer a conversão.

  Ex:

```
nome = input(“Digite seu nome: “)
print(f“Bom dia {nome} !”)
print(type(nome))
```

Saída:
Digite seu nome: **Jose Maria**
Bom dia **Jose Maria** !
<class ‘str’>

String é um interável

s1 = 'Piva'
s2 = 'Jr.'

- Pode se usar operações como:
  - slicing ([], [:])

```
print(s2[0])
```

J

```
print(s1[1:3])
```

iv

  - concatenação (+)

```
print(s1+' '+s2)
```

Piva Jr.

  - repetição (*)

```
print(s1*3)
```

PivaPivaPiva

  - *membership* (in)

```
for i in s1:
    print(i)
```

P
i
v
a

Funções de manipulação de strings

str (num)	Converte um número em String
len (str)	Retorna o tamanho de uma String

```
n = 123456789
numero = str(n)
print(type(n))
print(type(numero))

print(len(numero))
```

```
<class 'int'>
<class 'str'>

9
```

Métodos de strings

str.count (s)	Retorna a quatidade de conjuntos *s* presentes na string
str.isalpha ()	Retorna False se a string contiver algum caracter que não seja letras
str.isdigit ()	Retorna False se a string contiver algum caracter que não seja número
str.lower ()	Retorna a string transformada em minúsculos
str.upper ()	Retorna a string transformada em maiúsculos
str.replace (old, new)	Substitui uma porção da string por outro conteúdo
str.strip ()	Retira espaços em branco no começo e no fim da string
str.title ()	Retorna a string capitalizada (iniciais em maiúscula)
str.split (delimit)	Separa uma string conforme um delimitador. É o inverso do join()
str.join (sequence)	Junta cada item da string com um delimitador especificado. É o inverso do split().

VAMOS PARA A PRÁTICA ?!!!

Python:
Strings

O operador de formatação

- Strings suportam o operador % que, dada uma string especial (template) e um valor, produz uma string *formatada*
- O formato geral é
  - *template % valor*
- O template é uma string entremeada por códigos de formatação
  - Um código de formatação é em geral composto do caractere % seguido de uma letra descritiva do tipo do valor a formatar (s para string, f para float, d para inteiro, etc)
- Exemplo:

```
>>> '====%d====' % 100
'====100===='
>>> '====%f====' % 1
'====1.000000===='
```

Anatomia das especificações de formato

- Caractere %
- Flags de conversão (opcionais):
  - - indica alinhamento à esquerda
  - + indica que um sinal deve preceder o valor convertido
  - “ ” (um branco) indica que um espaço deve preceder números positivos
  - 0 indica preenchimento à esquerda com zeros
- Comprimento mínimo do campo (opcional)
  - O valor formatado terá este comprimento no mínimo
  - Se igual a * (asterisco), o comprimento será lido da tupla
- Um “.” (ponto) seguido pela precisão (opcional)
  - Usado para converter as casas decimais de floats
  - Se aplicado para strings, indica o comprimento máximo
  - Se igual a *, o valor será lido da tupla
- Caractere indicador do tipo de formato

Tipos de formato

- d, i Número inteiro escrito em decimal
- o Número inteiro sem sinal escrito em octal
- u Número inteiro sem sinal escrito em decimal
- x Número inteiro sem sinal escrito em hexadecimal (minúsculas)
- X Número inteiro sem sinal escrito em hexadecimal (maiúsculas)
- e Número de ponto flutuante escrito em notação científica ('e' minúsculo)
- E Número de ponto flutuante escrito em notação científica ('E' maiúsculo)
- f, F Número de ponto flutuante escrito em notação convencional
- g Mesmo que e se expoente é maior que -4. Caso contrario, igual a f
- G Mesmo que E se expoente é maior que -4. Caso contrario, igual a F
- C Caractere único (usado com inteiro ou string de tamanho 1)
- r String (entrada é qualquer objeto Python que é convertido usando a função repr)

Exemplos

```
>>> "Numero inteiro: %d" % 55
'Numero inteiro: 55'
>>> "Numero inteiro com 3 casas: %3d" % 55
'Numero inteiro com 3 casas:  55'
>>> "Inteiro com 3 casas e zeros a esquerda: %03d" % 55
'Inteiro com 3 casas e zeros a esquerda: 055'
>>> "Inteiro escrito em hexadecimal: %x" % 55
'Inteiro escrito em hexadecimal: 37'
>>> from math import pi
>>> "Ponto flutuante: %f" % pi
'Ponto flutuante: 3.141593'
>>> "Ponto flutuante com 12 decimais: %.12f" % pi
'Ponto flutuante com 12 decimais: 3.141592653590'
>>> "Ponto flutuante com 10 caracteres: %10f" % pi
'Ponto flutuante com 10 caracteres:   3.141593'
>>> "Ponto flutuante em notacao cientifica: %10e" % pi
'Ponto flutuante em notacao cientifica: 3.141593e+00'
>>> "String com tamanho maximo definido: %.3s" % "Pedro"
'String com tamanho maximo definido: Ped'
```

Exemplo: Imprimindo uma tabela

```
Itens    = ["Abacate", "Limão", "Tangerina", "Melancia”, "Laranja da China"]
precos = [2.13, 0.19, 1.95, 0.87, 12.00]

len_precos = 10 # Coluna de precos tem 10 caracteres

# Achar a largura da coluna de itens
len_itens = len(itens[0])
for it in itens : len_itens = max(len_itens,len(it))

# Imprimir tabela de precos
print ("-"*(len_itens+len_precos))
print ("%-*s%*s" % (len_itens, "Item", len_precos, "Preço”))
print ("-"*(len_itens+len_precos))
for i in range(len(itens)):
        print ("%-*s%*.2f" % (len_itens, itens[i],len_precos, precos[i]))
```

Exemplo: resultados

Item	Preço
Abacate	2.13
Limão	0.19
Tangerina	1.95
Melancia	0.87
Laranja da China	12.00

O Módulo String

- Manipulação de strings é uma atividade frequente em programas Python
- Existe um módulo chamado string que contém uma grande quantidade de funcionalidades para trabalhar com strings
  - Para usá-las:

    ```
    from string import *
    ```
- Entretanto, strings pertencem à classe str e a maior parte do que existe no módulo string aparece como métodos da classe str

Strings: método *find*

- find (*substring, inicio, fim*)
  - Retorna o índice da primeira ocorrência de *substring*
  - *inicio* e *fim* são opcionais e indicam os intervalos de índices onde a busca será efetuada
    - Os defaults são 0 e o comprimento da string, respectivamente
  - Caso *substring* não apareça na string, é retornado -1
  - Observe que o operador in pode ser usado para dizer se uma substring aparece numa string

Strings: método *find* (exemplo)

```
>>> s = "quem parte e reparte, fica com a maior parte"
>>> s.find("parte”)
5
>>> s.find("reparte”)
13
>>> s.find("parcela”)
-1
>>> "parte" in s
True
>>> s.find("parte",6)
15
>>> s.find("parte",6,12)
-1
```

Strings: método *join*

- join(*sequência*)
  - Retorna uma string com todos os elementos da *sequência* concatenados
    - Obs: Os elementos da sequência têm que ser strings
  - A string objeto é usada como separador entre os elementos
  - Ex.:

```
>>> "/".join(("usr","bin","python"))
'usr/bin/python'
>>> "Q".join((1,2,3,4,5))
...
TypeError: sequence item 0: expected string, int found
>>> "Q".join(('1','2','3','4','5'))
'1Q2Q3Q4Q5'
```

Strings: métodos *lower* e *upper*

- lower()
  - Retorna a string com todos os caracteres maiúsculos convertidos para minúsculos
- upper()
  - Retorna a string com todos os caracteres minúsculos convertidos para maiúsculos
  - Ex.:

```
>>> print ("Esperança".upper())
ESPERANÇA
>>> print ("Pé de Laranja Lima".lower())
pé de laranja lima
```

Strings: método *replace*

- replace(*velho*,*novo*,*n*)
  - Substitui as instâncias da substring *velho* por *novo*
  - Se *n* for especificado, apenas *n* instâncias são trocadas
  - Caso contrário, todas as instâncias são trocadas
  - Ex.:

```
>>> s = "quem parte e reparte, fica com a maior parte"
>>> s.replace("parte","parcela”)
'quem parcela e reparcela, fica com a maior parcela'
>>> s.replace("parte","parcela",2)
'quem parcela e reparcela, fica com a maior parte'
```

Strings: método *split*

- split(*separador*)
  - Retorna uma lista com as substrings presentes entre cópias da string *separador*
  - Faz o contrário do método join
  - Se *separador* não for especificado, é assumido sequências de caracteres em branco, tabs ou newlines
  - Ex.:

```
>>> s = "xxx yyy zzz  xxx  yyy zzz"
>>> s.split()
['xxx', 'yyy', 'zzz', 'xxx', 'yyy', 'zzz']
>>> s.split('xxx’)
['', ' yyy zzz  ', '  yyy zzz']
```

Strings: método *strip*

- strip(*ch*)
  - Retorna a string sem caracteres iniciais ou finais que estejam na string *ch*
  - Se *ch* não for especificada, retira caracteres em branco
  - Pode-se também usar rstrip() para retirar caracteres à direita (final) ou lstrip() para retirar caracteres à esquerda (início)
  - Ex.:

```
>>> "   xxx afdsfa  ".strip()
'xxx afdsfa'
>>> "xxx yyy zzz xxx".strip("xy ”)
'zzz'
>>> "  xxx   ".rstrip()
'  xxx'
```

Strings: método *translate*

- translate(*trans*)
  - Retorna uma cópia da string onde os caracteres são substituídos de acordo com a tabela de tradução *trans*
  - *trans* é uma string com 256 caracteres, um para cada possível código de oito bits
    - Ex.: se *trans* tem 'X' na posição 65 (correspondente ao caractere ASCII 'A'), então, na string retornada, todos os caracteres 'A' terão sido substituídos por 'X'
  - Na verdade, as tabelas de tradução são normalmente construídas com a função maketrans do módulo string

Função string.maketrans

- maketrans (*velho*, *novo*)
  - retorna uma tabela de tradução onde os caracteres em *velho* são substituídos pelos caracteres em *novo*
  - Ex.:

```
>>> from string import maketrans
>>> trans = maketrans('qs', 'kz’)
>>> s = "que surpresa: quebrei a cara"
>>> s.translate(trans)
'kue zurpreza: kuebrei a cara'
```

EXERCÍCIOS

Strings...

EXERCÍCIO 1

Elabore um algoritmo para ler/receber, separadamente, o primeiro nome, o nome do meio e o sobrenome de uma pessoa, e mostre o nome completo, correspondente.

EXERCÍCIO 2

Faça um algoritmo que solicite uma data no formato de uma string – dd/mm/aaaa. Mostre essa data no formato AAAAMMDD

EXERCÍCIO 3

Faça um algoritmo para ler nove caracteres numéricos em uma string. Mostre o conteúdo dessa string colando pontos e virgula, respectivamente nas posições inteiras e decimais.

Exemplo:

Digitado> 987654321

Mostrado> 9.876.543,21

EXERCÍCIO 4

Elabore um algoritmo para determinar quantas vogais existem dentro de uma determinada frase (que deve ser recebida do usuário).

EXERCÍCIO 5

Faça um algoritmo para determinar quantas palavras existem em uma determinada frase
Obs: tanto a palavra, quanto a frase, devem ser informadas pelo usuário.

EXERCÍCIO 6

Faça um algoritmo para determinar se um determinado vetor, digitado pelo usuário, é um palíndromo.

Palíndromo: lido da direita para a esquerda, ou vice versa, representam a mesma coisa.
Ex: AMA