SCC 124 - Introdução à Programação para Engenharias

Strings

python

Professor: André C. P. L. F. de Carvalho, ICMC-USP
Pos-doutorando: Isvani Frias-Blanco
Monitor: Henrique Bonini de Britto Menezes

Aula de hoje

- Introdução
- Operações para o tipo srings
- Indexação de strings
- Alteração de strings
- Funções e métodos para strings

Expressões lógicas

- Essenciais em linguagens de programação
- Definem como programa será executado
  - Comandos *if* e *while*
- Pode incluir vários operadores lógicos
  - Ex.: *XOR = (A and !B) or (!A and B)*
- Expressões podem ser simplificadas pela manipulação de seus operadores
  - Levar negação para o termo mais simples

Relações lógicas equivalentes

Propriedades	Expressão 1 ⇔	Expressão 2
Comutatividade	A or B	B or A
	A and B	B and A
Associatividade	A or (B or C)	(A or B) or C
	A and (B and C)	(A and B) and C
Distributividade	A or (B and C)	(A or B) and (A or C)
	A and (B or C)	(A and B) or (A and C)
De Morgan	!(A or B)	!A and !B
	!(A and B)	!A or !B
Anulação	!!A	A

Introdução

- Arquivos com texto são gerados com frequência crescente
  - Internet
  - Bioinformática
  - Armazenamento de documentos
- Várias tarefas atuais de programação manipulam textos

Operadores para strings

- Python usa o tipo string para manipular textos
- Strings podem ser:
  - Comparados com operadores ==, !=, <, >, <= e >=
  - Concatenados com o operador +
  - Repetidos com o operador *

Exemplo

```
>>> palavra = 'Help' + 'A'
>>> palavra
'HelpA'
>>> '<' + palavra*5 + '>'
'<HelpAHelpAHelpAHelpAHelpA>'
```


Maiúsculas ≠ Minúsculas

```
a= "Help"
b = "help"
if a != b:
   print ("São diferentes")
```

```
São diferentes
```


Operadores para strings

- Dois literais *strings* adjacentes são automaticamente concatenados
  - Sem precisar do operador +
  - Funciona apenas com dois literais strings
    - Não com expressões *string* arbitrárias

```
>>> palavra = 'Help' 'A'
>>> palavra
'HelpA'
```


Indexação de strings

- *Strings* podem ser indexados
  - Primeiro elemento possui índice 0
  - Não existe um tipo *caracter*
    - *Caracter* é um *string* de tamanho 1
  - É possível utilizar intervalos para indexar um ou mais *caracteres*
    - *Substrings*

Fatiamento de strings

- *Substrings* podem ser indicados pelo uso da notação de fatias
  - Valores inicial e final separados por ":"
    - Fatia não inclui caracter do último índice
      - *Ex.: palavra[0:2] # não inclui palavra[2]*
  - Valores extremos podem ser negativos
    - *Ex.: palavra[1:-1] # não inclui palavra[0] nem palavra[-1]*

Fatiamento de strings

- Valores extremos podem ser omitidos
  - São usados valores default (0 e posição do último elemento)
  - Mas o sinal ":" não (é obrigatório)
- É possível usar um terceiro argumento
  - Passo do fatiamento
    - Por default, igual a 1
    - Ex.: palavra [0:2:2]

Exemplo

```
>>> palavra = 'Help' 'A'
>>> palavra[4]

>>> palavra[0:2]

>>> palavra[2:4]

>>> palavra[0:4:2]
```


Exemplo

```
>>> palavra = 'Help' 'A'
>>> palavra[4]
'A'
>>> palavra[0:2]
'He'
>>> palavra[2:4]
'lp'
>>> palavra[0:4:2]
'Hl'
```


Indexação

- Quando valor do índice é negativo, a contagem começa pela direita, com -1

H	e	l	p	A
0	1	2	3	4
-5	-4	-3	-2	-1

```
>>> palavra = 'Help' 'A'
>>> palavra[-1] # último caracter

>>> palavra[-2] # penúltimo

>>> palavra[-2:] # dois últimos caracteres

>>> palavra[:-2] # tudo menos os dois últimos caracteres

>>> palavra[-0] # -0 eh igual a 0
```


Tipo String

- Quando valor do índice é negativo, a contagem começa pela direita, com -1

H	e	l	p	A
0	1	2	3	4
-5	-4	-3	-2	-1

```
>>> palavra = 'Help' 'A'
>>> palavra[-1] # último caracter
'A'
>>> palavra[-2] # penúltimo
'p'
>>> palavra[-2:] # dois últimos caracteres
'pA'
>>> palavra[:-2] # tudo menos os dois últimos caracteres
'Hel'
>>> palavra[-0] # -0 eh igual a 0
'H'
```


Exercício

```
>>> palavra = 'Help' 'A'
>>> palavra[:2] #  Os primeiros dois caracteres

>>> palavra[2:] # Tudo menos os dois primeiros caracteres

>>> palavra[1:-1] # Tudo menos o primeiro e o ultimo

>>> palavra[::-1] # Inverte ordem dos caracteres
```


Exercício

```
>>> palavra = 'Help' 'A'
>>> palavra[:2] #  Os primeiros dois caracteres
'He'
>>> palavra[2:] # Tudo menos os dois primeiros caracteres
'lpA'
>>> palavra[1:-1] # Tudo menos o primeiro e o ultimo
'elp'
>>> palavra[::-1] # Inverte ordem dos caracteres
'ApleH'
```

Indexação de strings

- Valor positivo (negativo) do índice em uma fatia pode superar tamanho do string
  - Valor é convertido para o limite real
    - Não pode ser feito quando um único elemento é indexado (ao invés de uma faixa)
- String vazio é retornado quando:
  - Índice inferior é maior que tamanho do string
  - Índice superior é menor que índice inferior

Exemplo

```
palavra = 'Help' 'A'
>>> palavra[1:100]
'elpA'
>>> palavra[10:]
''
>>> palavra[2:1]
''
```


Exemplo

```
>>> palavra = 'Help' 'A'
>>> palavra[:8]
'HelpA'
>>> palavra[-100:]
'HelpA'
>>> palavra[10] # error
Traceback (most recent call last):
  File "<stdin>", line 1, in ?
IndexError: string index out of range
>>> palavra[-10] # error
Traceback (most recent call last):
  File "<stdin>", line 1, in ?
IndexError: string index out of range
```


Exercício

```
>>> EcoRI = 'gaattc'
>>> EcoRI[0]

>>> EcoRI[-1]

>>> EcoRI[1:3]

>>> EcoRI[3:]
```

```
>>> EcoRI[:]

>>> EcoRI[:-1]

>>> EcoRI[1:100]

>>> EcoRI[3:1]

>>> EcoRI[100:101]
```


Exercício

```
>>> EcoRI = 'gaattc'
>>> EcoRI[0]
'g'
>>> EcoRI[-1]
'c'
>>> EcoRI[1:3]
'aa'
>>> EcoRI[3:]
'ttc'
```

```
>>> EcoRI[:]
'gaattc'
>>> EcoRI[:-1]
'gaatt'
>>> EcoRI[1:100]
'aattc'
>>> EcoRI[3:1]
''
>>> EcoRI[100:101]
''
```


Alteração de Strings

- Python não permite mudar o valor dos elementos de um string
  - Tipo imutável
  - Diferente da linguagem C
- Mas é fácil e eficiente combinar partes de strings para criar um novo string

Exemplo

```
>>> palavra = 'Help' 'A'
>>> palavra[0] = 'x'
Traceback (most recent call last):
  File "<stdin>", line 1, in ?
TypeError: object doesn't support item assignment
>>> palavra[:1] = 'Splat'
Traceback (most recent call last):
  File "<stdin>", line 1, in ?
TypeError: object doesn't support slice assignment
```


Exemplo

```
>>> palavra = 'Help' 'A'
>>> 'x' + palavra[1:]
'xelpA'
>>> 'Splat' + palavra[4]
'SplatA'
>>> palavra[:2] + palavra[2:]
'HelpA'
>>> palavra[:3] + palavra[3:]
'HelpA'
```

s[:i] + s[i:] == s

Funções e métodos

- Python é uma linguagem orientada a objetos
  - Classe define estrutura de dados e métodos para essa estrutura
  - Objeto é uma instanciação (variável) de uma classe
- Método é uma função que pertence a um objeto (obj)
  - É chamado pelo termo *obj.metodo*
  - Diferentes objetos definem diferentes métodos
  - Facilita manipulação de objetos

Métodos gerais

- Não restritos a strings
- *len (tex)*
  - Retorna o tamanho (número de caracteres) do string *tex*
- *subtex in tex*
  - Verifica se substring subtex pertence ao (ocorre no) string *tex*
  - *Substring pode ter apenas um caracter*

Exemplo

```
>>> dna = 'gcatgacgttattacgactctgtcacgcc'
>>> len(dna)
29
>>> 'tg' in dna
True
>>> 'n' in dna
False
```


Métodos para strings

- *tex.find (subtex)*
  - Retorna posição de primeira ocorrência do *substring subtex* no *string tex*
- *tex.count (subtex)*
  - Conta número de vezes que *subtex* ocorre no *string tex*
- *tex.replace (strvelho, strnovo)*
  - Troca em *tex* ocorrência de *strvelho* por *strnovo*

Exemplo

```
>>> dna = 'gcatgacgttattacgactctgtcacgcc'
>>> dna.count("a")

>>> dna.count("tg")

>>> dna.find("tg")

>>> dna.replace('a', 'A')
```


Exemplo

```
>>> dna = 'gcatgacgttattacgactctgtcacgcc'
>>> dna.count("a")
6
>>> dna.count("tg")
2
>>> dna.find("tg")
3
>>> dna.replace('a', 'A')
'gcAtgAcgttAttAcgActctgtcAcgcc '
```


Métodos para strings

- *tex.strip (subtex)*
  - Remove substring *subtex* do início e final de um string *tex*
    - Parâmetro do tipo *string* (s*tring* a ser removido)
  - Não modifica o string *tex*
    - Gera um novo *string*

Exemplo

```
>>> 'str' 'ing'                # Isso esta ok
'string'
>>> 'str'.strip('s') + 'ing'  # Isso esta ok
'tring'
>>> a = "stringss"
>>> a.strip('s') + 'ing'      # Isso esta ok
'tringing'
>>> a.strip() + 'ing'         # Isso esta ok
'stringssing'
>>> 'str'.strip() 'ing'       # Isso não eh valido (termo 1 não é literal)
  File "<stdin>", line 1, in ?
    'str'.strip() 'ing'
                  ^
SyntaxError: invalid syntax
```


Manipulando strings

- Converter strings para ints
  - Usar ordem em que aparece no código Unicode (preserva sequência ASCII)

```
>>> ord ('A')
65
>>> ord ('B')
66
>>> ord('a')
97
>>> chr (97)
'a'
```

```
>>> str (65 +3)
'68'
>>> ord('D') - ord('A')
3
>>> ord('a') – ord ('A')
32
>>> chr (ord('a') + 3)
'd'
```


Manipulando strings

- Procurando strings em listas

```
>>> vogais = 'aeiouAEIOU'
>>> numeros = '0123456789'
>>> consoantes = 'bcdfghjklmnpqrstvwxyz'
>>> n = 'e'
>>> n in vogais
True
>>> m = chr(ord (n) + 1) # letras após 'e' em ASCII'
>>> m
'f'
```

Exercícios

- Escrever um programa para verificar se um texto tem mais vogais ou consoantes
- Trocar no texto cada caracter por:
  - Letra anterior no alfabeto, se for letra
  - Número seguinte, se for número
- Calcular a percentagem da sequência GC de uma sequência de DNA
- Gerar complemento de uma sequência de DNA

Conclusão

- Introdução
- Operações para o tipo srings
- Indexação de strings
- Alteração de strings
- Métodos para strings

Perguntas