Algoritmos e Estruturas de Dados I

Prof. Tiago Eugenio de Melo
tmelo@uea.edu.br

www.tiagodemelo.info

Observações

- O conteúdo dessa aula é parcialmente proveniente do Capítulo 4 do livro “Fundamentals of Python – From First Programs Through Data Structures”.
- As palavras com a fonte `Courier` indicam uma palavra-reservada da linguagem de programação.

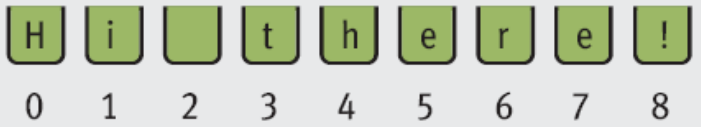
Objetivos

- Após o término dessa aula você deverá ser capaz de:
  - Acessar caracteres individuais em uma string.
  - Recuperar uma substring de uma string.
  - Pesquisar por uma substring em uma string.
  - Usar métodos para manipulação de strings.
  - Manipular arquivos de texto.

Strings

- Uma string é uma estrutura de dados **imutável**.
  - Estrutura de dados: consiste de “pedaços” menores de dados.
  - Tamanho da string: número de caracteres que ela contém.

```
>>> len("Hi there!")
9
>>> len("")
0
```



[FIGURE 4.1] Characters and their positions in a string

Acessando as strings

- Uma forma comum de acessar strings:

```
<a string>[<an integer expression>]
```

index is usually in range [0,len); can be negative

- Exemplos

```
>>> name = "Alan Turing"
>>> name[0]                         # Examine the first character
'A'
>>> name[3]                         # Examine the fourth character
'n'
>>> name[len(name)]                 # Oops! An index error!
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
IndexError: string index out of range
>>> name[len(name) - 1]             # Examine the last character
'g'
>>> name[-1]                        # Shorthand for the last one
'g'
```

Acessando as strings (cont.)

- Esse tipo de operador é útil quando você quer usar as posições, assim como os caracteres em uma string.
- Exemplo:

```
>>> data = "Hi there!"
>>> for index in xrange(len(data)):
        print index, data[index]

0 H
1 i
2
3 t
4 h
5 e
6 r
7 e
8 !
```

Fatiamento

- Uma parte da string pode ser acessada através do processo chamado fatiamento (***slicing***).
- Exemplo

```
>>> name = "myfile.txt"
>>> name[0:]                    # The entire string
'myfile.txt'
>>> name[0:1]                   # The first character
'm'
>>> name[0:2]                   # The first two characters
'my'
>>> name[:len(name)]            # The entire string
'myfile.txt'
>>> name[-3:]                   # The last three characters
'txt'
```

Usando o operador `in`

- Exemplo

```
>>> fileList = ["myfile.txt", "myprogram.exe", "yourfile.txt"]
>>> for fileName in fileList:
        if ".txt" in fileName:
            print fileName

myfile.txt
yourfile.txt
>>>
```

- Retorna `True` se o valor é encontrado e `False`, caso contrário.

Métodos em strings

- Python inclui um conjunto de operações em strings chamados de métodos que, por exemplo, contam a quantidade de palavras em uma simples sentença.

```
>>> sentence = raw_input("Enter a sentence: ")
Enter a sentence: This sentence has no long words.
>>> listOfWords = sentence.split()
>>> print "There are", len(listOfWords), "words."
There are 6 words.
>>> sum = 0
>>> for word in listOfWords:
        sum += len(word)

>>> print "The average word length is", sum / len(listOfWords)
The average word length is 4
```

Métodos em strings (cont.)

- Um método se comporta como uma função, mas com o comportamento um pouco diferente.
- Um método é sempre chamado com um dado valor chamado de **objeto**.

```
<an object>.<method name>(<argument-1>, …, <argument-n>)
```

- Métodos podem esperar argumentos (entrada) e retornar valores (saída).
- Um método conhece sobre o estado interno do objeto para o qual ele é chamado.
- Em Python, todos os valores são, de fato, objetos.

Métodos em strings (cont.)

STRING METHOD	WHAT IT DOES
`s.center(width)`	Returns a copy of `s` centered within the given number of columns.
`s.count(sub [, start [, end]])`	Returns the number of non-overlapping occurrences of substring `sub` in `s`. Optional arguments `start` and `end` are interpreted as in slice notation.
`s.endswith(sub)`	Returns `True` if `s` ends with `sub` or `False` otherwise.
`s.find(sub [, start [, end]])`	Returns the lowest index in `s` where substring `sub` is found. Optional arguments `start` and `end` are interpreted as in slice notation.
`s.isalpha()`	Returns `True` if `s` contains only letters or `False` otherwise.
`s.isdigit()`	Returns `True` if `s` contains only digits or `False` otherwise.

[TABLE 4.2] Some useful string methods, with the code letter `s` used to refer to any string

Métodos em strings (cont.)

STRING METHOD	WHAT IT DOES
`s.join(sequence)`	Returns a string that is the concatenation of the strings in the sequence. The separator between elements is `s`.
`s.lower()`	Returns a copy of `s` converted to lowercase.
`s.replace(old, new [, count])`	Returns a copy of `s` with all occurrences of substring `old` replaced by `new`. If the optional argument `count` is given, only the first `count` occurrences are replaced.
`s.split([sep])`	Returns a list of the words in `s`, using `sep` as the delimiter string. If `sep` is not specified, any whitespace string is a separator.
`s.startswith(sub)`	Returns `True` if `s` starts with `sub` or `False` otherwise.
`s.strip([aString])`	Returns a copy of `s` with leading and trailing whitespace (tabs, spaces, newlines) removed. If `aString` is given, remove characters in `aString` instead.
`s.upper()`	Returns a copy of `s` converted to uppercase.

[TABLE 4.2] Some useful string methods, with the code letter `s` used to refer to any string

Métodos em strings (cont.)

```
>>> s = "Hi there!"
>>> len(s)
9
>>> s.center(11)
' Hi there! '
>>> s.count('e')
2
>>> s.endswith("there!")
True
>>> s.startswith("Hi")
True
>>> s.find('the')
3
>>> s.isalpha()
False
>>> 'abc'.isalpha()
True
>>> "326".isdigit()
True
>>> words = s.split()
>>> words
['Hi', 'there!']
>>> "".join(words)
'Hithere!'
>>> " ".join(words)
'Hi there!'
>>> s.lower()
'hi there!'
>>> s.upper()
'HI THERE!'
>>> s.replace('i', 'o')
'Ho there!'
>>> " Hi there! ".strip()
'Hi there!'
>>>
```

Métodos em strings (cont.)

- Exemplo: extrair a extensão de um arquivo.

```
>>> "myfile.txt".split(".")
['myfile', 'txt']
>>> "myfile.py".split(".")
['myfile', 'py']
>>> "myfile.html".split(".")
['myfile', 'html']
>>>
```

- O operador [-1] extrai o último elemento da string.

```
filename.split(".")[-1]
```

Arquivos de texto

- Um arquivo de texto armazena dados em uma mídia de modo permanente.
- Quais são as vantagens do uso de arquivo como entrada de dados quando comparamos com a entrada de dados pelo teclado?
  - O conjunto de dados pode ser muito maior.
  - Os dados podem ser lido de maneira mais rápida e com menor chance de erros.
  - Os dados podem ser usados repetidamente com o mesmo programa ou por programas diferentes.

Formato do arquivo de texto

- Using um editor de texto como Notepad ou vim, você pode criar, visualizar e gravar dados em um arquivo de texto.

```
34.6 22.33 66.75
77.12 21.44 99.01
```

- Todos os dados de saída ou de entrada de um arquivo texto são strings.

Escrevendo em um arquivo-texto

- Dados podem ser gravados em um arquivo-texto usando o objeto **`file`**.
- Para abrir um arquivo:

```
>>> f = open("myfile.txt", 'w')
```

  - Se o arquivo não existe, então ele é criado.
  - Se o arquivo já existe, então Python abre o arquivo. Quando os dados são escritos no arquivo e o arquivo é fechado. Neste caso, qualquer dado existente nesse arquivo será apagado.

```
>>> f.write("First line.\nSecond line.\n")
```

```
>>> f.close()
```

Failure to close output file can result in data being lost

Escrevendo números em um arquivo

- O método **write** espera uma string como um argumento.
  - Outros tipos de dados devem ser primeiro convertidos para strings antes de serem escritos para saída do arquivo (ex: usando `str`).

```
import random
f = open("integers.txt", 'w')
for count in xrange(500):
    number = random.randint(1, 500)
    f.write(str(number) + "\n")
f.close()
```

Lendo texto de um arquivo

- Você abre um arquivo para escrever de uma maneira similar:

```
>>> f = open("myfile.txt", 'r')
```

  - Se o *pathname* não é acessível do diretório atual, ocorre uma exceção (erro) em Python.

- Existem algumas maneiras de ler dados de um arquivo. Exemplo: método `read`.

```
>>> text = f.read()
>>> text
'First line.\nSecond line.\n'
>>> print text
First line.
Second line.
```

Lendo texto de um arquivo (cont.)

- Depois que a entrada de dados é finalizada, `read` retorna uma string vazia.

```
>>> f = open("myfile.txt", 'r')
>>> for line in f:
        print line

First line.

Second line.
>>> f = open("myfile.txt", 'r')
>>> while True:
        line = f.readline()
        if line == "":
            break
        print line

First line.

Second line.
```

Lendo números de um arquivo

- Exemplos

```
f = open("integers.txt", 'r')
sum = 0
for line in f:
    line = line.strip()
    number = int(line)
    sum += number
print "The sum is", sum
```

```
f = open("integers.txt", 'r')
sum = 0
for line in f:
    wordlist = line.split()
    for word in wordlist:
        number = int(word)
        sum += number
print "The sum is", sum
```

Lendo números de um arquivo

METHOD	WHAT IT DOES
`open(pathname, mode)`	Opens a file at the given pathname and returns a **`file`** object. The **`mode`** can be **`'r'`**, **`'w'`**, **`'rw'`**, or **`'a'`**. The last two values, **`'rw'`** and **`'a'`**, mean read/write and append, respectively.
`f.close()`	Closes an output file. Not needed for input files.
`f.write(aString)`	Outputs **`aString`** to a file.
`f.read()`	Inputs the contents of a file and returns them as a single string. Returns `''` if the end of file is reached.
`f.readline()`	Inputs a line of text and returns it as a string, including the newline. Returns `''` if the end of file is reached.

[TABLE 4.3] Some **`file`** operations

Acessando e manipulando arquivos e diretórios do disco

- Quando projetando programas em Python que acessam arquivos é recomendável incluir o tratamento de erros e exceções.
- Por exemplo, antes de tentar abrir um arquivo para entrada de dados, você deveria verificar se o arquivo existe.
  - A função **`os.path.exists`** permite essa verificação.

Acessando e manipulando arquivos e diretórios do disco (cont.)

- Exemplo: imprimir todos os nomes de dados no atual diretório com a extensão **.py**.

```
import os
currentDirectoryPath = os.getcwd()
listOfFileNames = os.listdir(currentDirectoryPath)
for name in listOfFileNames:
    if ".py" in name:
        print name
```

Acessando e manipulando arquivos e diretórios do disco (cont.)

os MODULE FUNCTION	WHAT IT DOES
`chdir(path)`	Changes the current working directory to **path**.
`getcwd()`	Returns the path of the current working directory.
`listdir(path)`	Returns a list of the names in directory named **path**.
`mkdir(path)`	Creates a new directory named **path** and places it in the current working directory.
`remove(path)`	Removes the file named **path** from the current working directory.
`rename(old, new)`	Renames the file or directory named **old** to **new**.
`rmdir(path)`	Removes the directory named **path** from the current working directory.

[TABLE 4.4] Some file system functions

Acessando e manipulando arquivos e diretórios do disco (cont.)

os.path MODULE FUNCTION	WHAT IT DOES
`exists(path)`	Returns **`True`** if **`path`** exists and **`False`** otherwise.
`isdir(path)`	Returns **`True`** if **`path`** names a directory and **`False`** otherwise.
`isfile(path)`	Returns **`True`** if **`path`** names a file and **`False`** otherwise.
`getsize(path)`	Returns the size of the object names by **`path`** in bytes.

[TABLE 4.5] More file system functions