MAP 2112 – Introdução à Lógica de Programação e Modelagem Computacional

1º Semestre - 2023

Prof. Dr. Luis Carlos de Castro Santos

lsantos@ime.usp.br

Chapter 7 Strings

7.1 A compound data type ("Um tipo composto")

Dos tipos de dados vistos até o momento: `int`, `float` e `string`; as strings são compostas de caracteres, por isso são chamadas de tipo de dados compostos.

Existem ocasiões em que são tratadas como objetos únicos e outros, quando desejamos acessar as partes menores (no caso, os caracteres).

```
fruit = "banana"

letter = fruit[1]

print (letter)
```

```
In [2]: runfile('C:/Users/win/Dropbox/USP/2023/
remote/MAP2112_2023/scripts/aula6_str1.py',
wdir='C:/Users/win/Dropbox/USP/2023/remote/
MAP2112_2023/scripts')
a
```

```
fruit = "banana"

letter = fruit[0]

print (letter)
```

```
In [3]: runfile('C:/Users/win/Dropbox/USP/2023/
remote/MAP2112_2023/scripts/aula6_str1.py',
wdir='C:/Users/win/Dropbox/USP/2023/remote/
MAP2112_2023/scripts')
b
```

A indexação no Python começa com a posição 0

"[x]" indica que se deseja extrair o conteúdo da posição "x" da string

7.2 Length ("comprimento")

O comando len retorna o comprimento do string:

```
In [4]: len(fruit)
Out[4]: 6
```

```
fruit = "banana"
length = len(fruit)
last = fruit[length]
```

```
In [10]: runfile('C:/Users/win/Dropbox/USP/2023/
remote/MAP2112_2023/scripts/aula6_str1.py',
wdir='C:/Users/win/Dropbox/USP/2023/remote/
MAP2112_2023/scripts')
Traceback (most recent call last):

  File C:\Program
Files\Spyder\pkgs\spyder_kernels\py3compat.py:356 in
compat_exec
    exec(code, globals, locals)

  File c:
\users\win\dropbox\usp\2023\remote\map2112_2023\scri
pts\aula6_str1.py:9
    last = fruit[length]

IndexError: string index out of range
```

Como a indexação começa com zero a última posição tem índice de comprimento -1

```
fruit = "banana"
length = len(fruit)
last = fruit[length-1]
print(last)
```

```
In [12]: runfile('C:/Users/win/Dropbox/USP/2023/
remote/MAP2112_2023/scripts/aula6_str1.py',
wdir='C:/Users/win/Dropbox/USP/2023/remote/
MAP2112_2023/scripts')
a
```

7.3 Traversal and the for loop

O processo de percorrer a string (“traversal” ou varredura) é um padrão recorrente em objetos compostos:

```
fruit = "banana"
index = 0
while (index < len(fruit)):
    letter = fruit[index]
    print (letter)
    index = index + 1
```

```
In [13]: runfile('C:/Users/win/Dropbox/USP/2023/
remote/MAP2112_2023/scripts/aula6_str2.py',
wdir='C:/Users/win/Dropbox/USP/2023/remote/
MAP2112_2023/scripts')
b
a
n
a
n
a
```

A indexação começa com zero e a última posição tem índice de comprimento -1 pois o sinal “<“ é usado no laço do while impedindo a varredura além do final da string

Existe um comando no python que implementa de forma mais fácil e compacta a varredura:

```
fruit = "banana"

for char in fruit:
    print (char)
```

```
In [14]: runfile('C:/Users/win/Dropbox/USP/2023/
remote/MAP2112_2023/scripts/aula6_str3.py',
wdir='C:/Users/win/Dropbox/USP/2023/remote/
MAP2112_2023/scripts')
b
a
n
a
n
a
```

Sendo “fruit” uma string (variável composta) a variável arbitrária “char” assume o valor de cada posição que compõe a string, que no caso são os caracteres.

Mais adiante veremos outros objetos armazenadores de dados e usos mais amplos do comando for.

7.4 String slices ("fatias de strings")

```
s = "Peter, Paul, and Mary"

print (s[0:5])
print (s[7:11])
print (s[17:21])
```

```
In [17]: runfile('C:/Users/win/Dropbox/USP/2023/
remote/MAP2112_2023/scripts/aula6_str4.py',
wdir='C:/Users/win/Dropbox/USP/2023/remote/
MAP2112_2023/scripts')
Peter
Paul
Mary
```

Usando a indexação e o operador "[ ]" pode-se extrair o conteúdo do string.
Os dois pontos ":" estabelecem a variação da fatia.

```
fruit = "banana"
print(fruit[:3])
print(fruit[3:])
```

```
In [19]: runfile('C:/Users/win/Dropbox/USP/2023/
remote/MAP2112_2023/scripts/aula6_str4.py',
wdir='C:/Users/win/Dropbox/USP/2023/remote/
MAP2112_2023/scripts')
ban
ana
```

A omissão da posição inicial ou final permite a varredura respectivamente a partir da posição inicial ou até a final.

7.5 String comparison ("comparação de strings")

Os operadores de comparação funcionam com strings

```
p = "alpha"

if (p == "alpha"):
    print("strings iguais")
else:
    print("strings diferentes")
```

```
In [21]: runfile('C:/Users/win/Dropbox/USP/2023/
remote/MAP2112_2023/scripts/aula6_str5.py',
wdir='C:/Users/win/Dropbox/USP/2023/remote/
MAP2112_2023/scripts')
strings iguais
```

```
p = "beta"

if (p < "alpha"):
    print("A palavra ,"+p+", vem antes alpha")
else:
    print("A palavra ,"+p+", vem depois alpha")
```

```
In [24]: runfile('C:/Users/win/Dropbox/USP/2023/
remote/MAP2112_2023/scripts/aula6_str6.py',
wdir='C:/Users/win/Dropbox/USP/2023/remote/
MAP2112_2023/scripts')
A palavra ,beta, vem depois alpha
```

A comparação se dá pela ordem alfabética do 1º caractere

```
p = "Beta"

if (p < "alpha"):
    print("A palavra ,"+p+", vem antes alpha")
else:
    print("A palavra ,"+p+", vem depois alpha")
```

```
In [25]: runfile('C:/Users/win/Dropbox/USP/2023/
remote/MAP2112_2023/scripts/aula6_str6.py',
wdir='C:/Users/win/Dropbox/USP/2023/remote/
MAP2112_2023/scripts')
A palavra ,Beta, vem antes alpha
```

O alfabeto das maiúsculas precede o das minúsculas

7.6 Strings are immutable ("strings são imutáveis")

O operador "[ ]" é usado para extrair o conteúdo de uma certa posição.

```
greeting = "Hello, world!"

print (greeting[0])
```

```
In [26]: runfile('C:/Users/win/Dropbox/USP/2023/
remote/MAP2112_2023/scripts/aula6_str7.py',
wdir='C:/Users/win/Dropbox/USP/2023/remote/
MAP2112_2023/scripts')
H
```

Tentar modificar o conteúdo de uma posição por atribuição gera um erro.

```
greeting = "Hello, world!"

greeting[0] = "J"

print (greeting[0])
```

```
In [27]: runfile('C:/Users/win/Dropbox/USP/2023/
remote/MAP2112_2023/scripts/aula6_str7.py',
wdir='C:/Users/win/Dropbox/USP/2023/remote/
MAP2112_2023/scripts')
Traceback (most recent call last):

  File C:\Program
Files\Spyder\pkgs\spyder_kernels\py3compat.py:356
in compat_exec
    exec(code, globals, locals)

  File c:
\users\win\dropbox\usp\2023\remote\map2112_2023\sc
ripts\aula6_str7.py:4
    greeting[0] = "J"

TypeError: 'str' object does not support item
assignment
```

O objeto string não aceita atribuição por posição

Para modificar o conteúdo de posições numa string é necessário usar variáveis intermediárias.

```
greeting = "Hello, world!"

newGreeting = "J" + greeting[1:]

greeting = newGreeting

print (greeting)
```

```
In [28]: runfile('C:/Users/win/Dropbox/USP/2023/
remote/MAP2112_2023/scripts/aula6_str8.py',
wdir='C:/Users/win/Dropbox/USP/2023/remote/
MAP2112_2023/scripts')
Jello, world!
```

Um objeto intermediário foi usado para concatenar o conteúdo novo, no caso a letra “J” com a fatia que seria mantida do objeto original.

Em seguida é feita uma re-atribuição da variável anterior substituindo o conteúdo original.

7.7 A find function ("Uma função de busca")

Considere a função exemplo.

```
def find(str, ch):
    index = 0
    while (index < len(str)):
        if (str[index] == ch):
            return index
        index = index + 1
    return -1

greeting = "Hello, world!"


char = "o"

p = find(greeting, char)

print("A primeira posição em que ",char," aparece é ",p)
```

Dado um string e um caractere a função encontra a primeira posição que o caractere aparece

Na primeira posição em que a condição é verdadeira o laço é quebrado

Caso a string não contenha o caractere a saída é -1

```
In [33]: runfile('C:/Users/win/Dropbox/USP/2023/remote/
MAP2112_2023/scripts/aula6_str9.py', wdir='C:/Users/win/
Dropbox/USP/2023/remote/MAP2112_2023/scripts')
A primeira posição em que  o  aparece é  4
```

7.8 Looping and counting ("contagem com loops")

O seguinte trecho de código conta quantas vezes o caractere "a" aparece no string "fruit".

```
fruit = "banana"
count = 0
for char in fruit:
    if (char == "a"):
        count = count + 1
print (count)
```

O loop de for é usado para varrer as posições do string

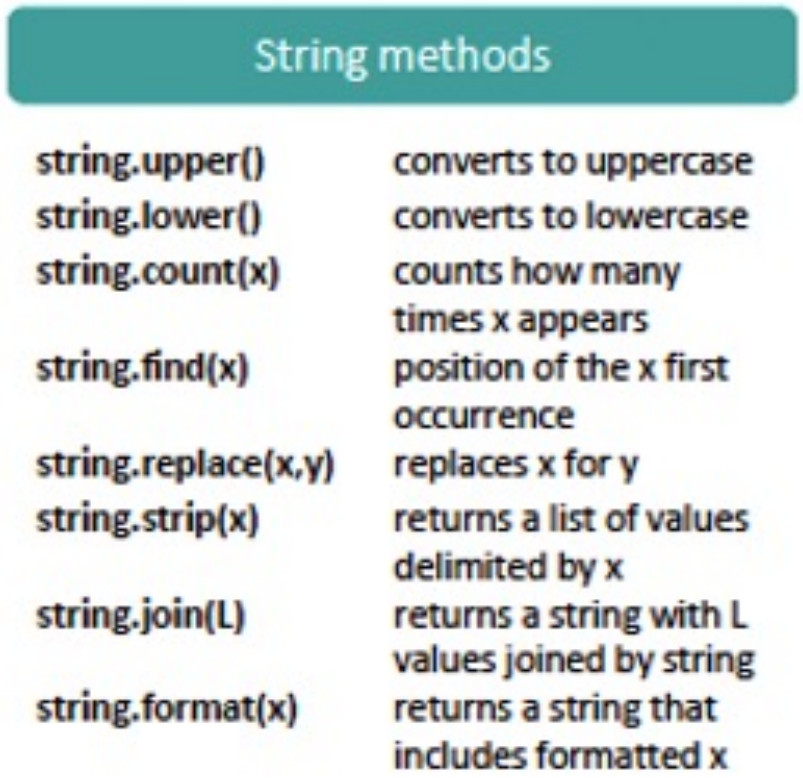
O contador avança a cada caractere encontrado

```
In [35]: runfile('C:/Users/win/Dropbox/USP/2023/remote/
MAP2112_2023/scripts/aula6_str10.py', wdir='C:/Users/win/
Dropbox/USP/2023/remote/MAP2112_2023/scripts')
3
```

Como exercício construa uma função que dado um string e um caractere conte quantas vezes esse caractere aparece no string.

As seções, 7.9 e 7.10 utilizam funções do Python 2 que foram removidas do Python 3.

No Python 3 utilizam-se os métodos, alguns deles estão listados na “folha-de-cola” (cheat sheet)



String methods

string.upper()	converts to uppercase
string.lower()	converts to lowercase
string.count(x)	counts how many times x appears
string.find(x)	position of the x first occurrence
string.replace(x,y)	replaces x for y
string.strip(x)	returns a list of values delimited by x
string.join(L)	returns a string with L values joined by string
string.format(x)	returns a string that includes formatted x

O uso é aplicado especificamente no objeto “string” veremos alguns exemplos em seguida

Uma lista mais completa de métodos para strings está disponível em :
https://www.w3schools.com/python/python_ref_string.asp

```
In [84]: txt = "Hello my friends"

In [85]: x = txt.upper()

In [86]: x
Out[86]: 'HELLO MY FRIENDS'
```

Todos caracteres são convertidos para maiúsculos

```
In [87]: txt = "Hello my FRIENDS"

In [88]: x = txt.lower()

In [89]: x
Out[89]: 'hello my friends'
```

Todos caracteres são convertidos para minúsculos

```
In [90]: txt = "I love apples, apple are my favorite fruit"

In [91]: x = txt.count("apple")

In [92]: x
Out[92]: 2
```

Todas as instâncias do string desejado são contadas no string original

```
In [93]: txt = "Hello, welcome to my world."

In [94]: x = txt.find("welcome")

In [95]: x
Out[95]: 7
```

O índice no qual o string desejado começa no string original é indicada

```
In [96]: txt = "I like bananas"

In [97]: x = txt.replace("bananas", "apples")

In [98]: x
Out[98]: 'I like apples'
```

Substitiu a string indicada por outra dentro da string original

```
In [101]: txt = "     banana     "

In [102]: x = txt.strip()

In [103]: x
Out[103]: 'banana'
```

Remove os espaços no começo e no final do string

```
In [106]: Nomes = ["John", "Peter", "Vicky"]

In [107]: x = "#".join(Nomes)

In [108]: x
Out[108]: 'John#Peter#Vicky'
```

Essa entidade é uma lista que será vista a seguir

O *join* irá unir os elementos da lista num único string separado por #

As chaves indicam as posições a serem preenchidas

```
In [104]: dados_pessoais = "Meu nome é {}, tenho {} anos"

In [105]: print(dados_pessoais.format("John",36))
Meu nome é John, tenho 36 anos
```

O *format* indica como preencher as posições indicadas pelas chaves

Chapter 8 Lists

Uma lista é um conjunto de valores identificados por um índice. O valores da lista são chamados de **elementos** . Listas se assemelham a strings mas seu conteúdo pode conter todos os tipos de elementos (inclusive outras listas).

8.1 List values

Alguns exemplos de listas

```
lista_1 = [10, 20, 30, 40]

lista_2 = [ "spam", "bungee", "sparrow"]

lista_3 = ["hello", 2.0, 5, [10, 20]]

print(lista_1)

print(lista_2)

print(lista_3)
```

Listas são definidas usando o colchete (*bracket*)

Listas que contém listas são chamadas listas aninhadas (*nested*)

```
In [1]: runfile('C:/Users/win/Dropbox/USP/2023/remote/
MAP2112_2023/scripts/untitled0.py', wdir='C:/Users/win/
Dropbox/USP/2023/remote/MAP2112_2023/scripts')
[10, 20, 30, 40]
['spam', 'bungee', 'sparrow']
['hello', 2.0, 5, [10, 20]]
```

Listas que contém sequências numéricas, em especial de inteiros são muito comuns.
O python oferece um comando que auxilia nessa criação.

Comando range produz uma sequência que pode ser usado na construção de uma lista

```
lista_1 = list(range(1, 5))

lista_2 = list(range(10))

lista_3 = list(range(1, 10, 2))

print(lista_1)

print(lista_2)

print(lista_3)
```

Os valores iniciais e finais podem ser definidos

Se apenas um valor é definido ele é entendido como o no. de elementos a partir do zero.

Pode-se definir um valor inicial, final e um passo.

```
In [2]: runfile('C:/Users/win/Dropbox/USP/
2023/remote/MAP2112_2023/scripts/
aula6_lst2.py', wdir='C:/Users/win/Dropbox/
USP/2023/remote/MAP2112_2023/scripts')
[1, 2, 3, 4]
[0, 1, 2, 3, 4, 5, 6, 7, 8, 9]
[1, 3, 5, 7, 9]
```

Dentre as listas que podem ser criadas existe a lista “vazia”, ela define a classe do objeto poderá ser preenchido depois.

```
vocabulary = ["ameliorate", "castigate", "defenestrate"]

numbers = [17, 123]

empty = []

print(vocabulary, numbers, empty)
```

```
In [3]: runfile('C:/Users/win/Dropbox/USP/2023/remote/MAP2112_2023/
scripts/aula6_lst3.py', wdir='C:/Users/win/Dropbox/USP/2023/remote/
MAP2112_2023/scripts')
['ameliorate', 'castigate', 'defenestrate'] [17, 123] []
```

Os colchetes [ ] indicam que estamos tratando de listas.

8.2 Accessing elements (acessando elementos)

A sintaxe de acesso do elementos da lista é a mesma dos strings, ou seja o colchete [ ].

```
numbers = [17, 123]

print(numbers[0])

numbers[1] = 5

print(numbers)
```

O índice usado tanto para extração quanto atribuição de elementos

```
In [4]: runfile('C:/Users/win/Dropbox/USP/2023/remote/
MAP2112_2023/scripts/aula6_lst5.py', wdir='C:/Users/win/
Dropbox/USP/2023/remote/MAP2112_2023/scripts')
17
[17, 5]
```

Operações podem ser feitas “in place”

```
print(numbers[3-2])

print(numbers[1.0])
```

```
In [6]: runfile('C:/Users/win/Dropbox/USP/2023/remote/
MAP2112_2023/scripts/aula6_lst5.py', wdir='C:/Users/win/Dropbox/
USP/2023/remote/MAP2112_2023/scripts')
123
Traceback (most recent call last):

  ⋮

TypeError: list indices must be integers or slices, not float
```

Índices devem obrigatoriamente ser inteiros

Índices negativos acessam os elementos a partir do final da lista

```
numbers = [17, 123]

print(numbers[-1])

print(numbers[-2])

print(numbers[-3])
```

```
In [7]: runfile('C:/Users/win/Dropbox/USP/2023/remote/
MAP2112_2023/scripts/aula6_lst6.py', wdir='C:/Users/win/Dropbox/
USP/2023/remote/MAP2112_2023/scripts')
123
17
Traceback (most recent call last):

          ⋮

IndexError: list index out of range
```

Tentar acessar um índice fora dos limites da lista (independente da extremidade) produz um erro

8.3 List length (comprimento da lista)

Para facilitar a varredura, e evitar acessos indevidos, pode-se utilizar o comando “len” para interrogar a lista sobre seu comprimento.

```
horsemen = ["war", "famine", "pestilence", "death"]

i = 0
while i < len(horsemen):
    print (horsemen[i])
    i = i + 1
```

A indexação começa em zero e termina em len(horsemen)-1

```
In [8]: runfile('C:/Users/win/Dropbox/USP/2023/remote/
MAP2112_2023/scripts/aula6_lst8.py', wdir='C:/Users/win/
Dropbox/USP/2023/remote/MAP2112_2023/scripts')
war
famine
pestilence
death
```

O comprimento das listas aninhadas é definido pelo número de elementos

```
In [10]: len(["spam!", 1, ["Brie", "Roquefort", "Pol le Veq"], [1, 2, 3]])
Out[10]: 4
```

8.4 List membership (filiação a listas)

Operadores booleanos permitem testar se um elemento pertence a uma lista

```
horsemen = ["war", "famine", "pestilence", "death"]

print("pestilence" in horsemen)

print("debauchery" in horsemen)

print("pestilence" not in horsemen)
```

```
In [12]: runfile('C:/Users/win/Dropbox/USP/2023/remote/MAP2112_2023/
scripts/aula6_lst9.py', wdir='C:/Users/win/Dropbox/USP/2023/remote/
MAP2112_2023/scripts')
True
False
False
```

8.5 Lists and for loops (Lista e laços de *for*)

Os laços de while que vimos inicialmente (e que podem produzir loops infinitos) podem ser substituídos por laços de *for* quando o número de passagens no laço é determinado.

O exemplo anterior:

```
horsemen = ["war", "famine", "pestilence", "death"]

i = 0
while i < len(horsemen):
    print (horsemen[i])
    i = i + 1
```

Pode ser re-escrito na forma:

```
for horseman in horsemen:
    print (horseman)
```

A variável horseman (no singular) assume o valor de cada elemento da lista horsemen (no plural)

```
In [14]: runfile('C:/Users/win/Dropbox/USP/2023/re
scripts/aula6_lst10.py', wdir='C:/Users/win/Dropbo
MAP2112_2023/scripts')
war
famine
pestilence
death
```

Dessa forma não há necessidade de controlar índices .

O uso de *for* com listas pode ter diversas formas

```
for number in range(20):
    if (number % 2 == 0):
        print (number)
```

```
In [15]: runfile('C:/Users,
scripts/aula6_lst11.py', w
MAP2112_2023/scripts')
0
2
4
6
8
10
12
14
16
18
```

O valor de number varia de 0 a 19 e apenas os pares são apresentados

```
for fruit in ["banana", "apple", "quince"]:
    print ("I like to eat " + fruit + "s!")
```

```
I like to eat bananas!
I like to eat apples!
I like to eat quinces!
```

8.6 List operations (Operações sobre listas)

As operações sobre listas são análogas as que vimos sobre strings

```
a = [1, 2, 3]

b = [4, 5, 6]

c = a + b

print(c)
```

```
In [17]: runfile('C:/Users/win/D
scripts/aula6_lst12.py', wdir='C
MAP2112_2023/scripts')
[1, 2, 3, 4, 5, 6]
```

O operador soma (+) concatena as *listas*

```
print(a*2)
```

```
In [19]: runfile('C:/Users/win
scripts/aula6_lst12.py', wdir=
MAP2112_2023/scripts')
[1, 2, 3, 1, 2, 3]
```

O operador multiplicação (*) repete elementos das *listas*

8.7 List slices ("fatias de listas")

As opções são análogas as que vimos para strings:

```
lista = ["a","b","c","d","e","f"]

print(lista[1:3])

print(lista[:4])

print(lista[3:])

print(lista[:])
```

Lembre-se que a variação termina uma posição antes do último índice da fatia

```
In [20]: runfile('C:/Users/win/Dropbox/USP/2023/remote/
scripts/aula6_lst13.py', wdir='C:/Users/win/Dropbox/USP
MAP2112_2023/scripts')
['b', 'c']
['a', 'b', 'c', 'd']
['d', 'e', 'f']
['a', 'b', 'c', 'd', 'e', 'f']
```

Usando a indexação e o operador "[ ]" pode-se extrair o conteúdo da string. Os dois pontos ":" estabelecem a variação da fatia.

A omissão da posição inicial ou final permite a varredura respectivamente a partir da posição inicial ou até a final.

8.8 Lists are mutable (Listas são mutáveis)

Diferentemente das strings as listas tem como objetivo armazenar informações nos elementos e em seu processamento alterá-los.

```
fruit = ["banana","apple","quince"]

fruit[0] = "pear"

fruit[-1] = "orange"

print(fruit)
```

```
In [21]: runfile('C:/Users/win/Dropbox/USP/20
scripts/aula6_lst14.py', wdir='C:/Users/win/D
MAP2112_2023/scripts')
['pear', 'apple', 'orange']
```

A atribuição combinada com a localização do elemento atualiza a lista.

```
lista = ["a","b","c","d","e","f"]

lista[1:3] = ["x","y"]

print(lista)
```

```
In [22]: runfile('C:/Users/win/Dropbox/USP/2023
scripts/aula6_lst15.py', wdir='C:/Users/win/Dro
MAP2112_2023/scripts')
['a', 'x', 'y', 'd', 'e', 'f']
```

A atribuição de vários elementos pode ser feita usando o operador [:] da mesma forma que na extração.

Perceba que a varredura vai até até o elemento [2], o último índice não está incluído

Pode-se remover elementos atribuindo listas vazias as respectivas posições:

```
lista = ["a","b","c","d","e","f"]

lista[1:3] = []

print(lista)
```

```
In [24]: runfile('C:/Users/win/Dropbox/l
scripts/aula6_lst16.py', wdir='C:/Users/
MAP2112_2023/scripts')
['a', 'd', 'e', 'f']
```

Os elementos são removidos e a lista é re-dimensionada.

Pode-se introduzir elementos na lista de forma análoga:

```
lista = ["a","d","f"]

lista[1:1] = ["b","c"]

lista[4:4] = ["e"]

print(lista)
```

```
In [26]: runfile('C:/Users/win/Dropbox/USP/2023/r
scripts/aula6_lst17.py', wdir='C:/Users/win/Dropb
MAP2112_2023/scripts')
['a', 'b', 'c', 'd', 'e', 'f']
```

Os elementos são adicionados e a lista é re-dimensionada.

8.9 List deletion (Deletando listas)

A forma oficial de deletar elementos da lista, ao invés de usar listas vazias, é utilizar o comando *del*

```
a = ["one", "two", "three", "four"]

del a[1]

del a[-1]

print(a)
```

```
In [28]: runfile('C:/Users/win/Dropbox/
scripts/aula6_lst19.py', wdir='C:/Users
MAP2112_2023/scripts')
['one', 'three']
```

Pode-se remover múltiplos elementos na lista usando ":"

```
lista = ["a","b","c","d","e","f"]

del lista[1:3]

print(lista)
```

```
In [27]: runfile('C:/Users/win/Dropbox/USP/2023/
scripts/aula6_lst15.py', wdir='C:/Users/win/Drop
MAP2112_2023/scripts')
['a', 'd', 'e', 'f']
```

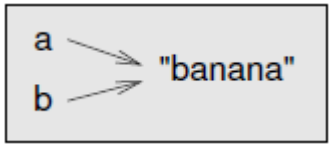
Os elementos são removidos e a lista é re-dimensionada.

8.10 Objects and values (Objetos e Valores)

No caso de strings a associação entre objetos e valores se dá na seguinte forma:

```
In [2]: a = "banana"

In [3]: b = "banana"

In [4]: id(a)
Out[4]: 1527379885360

In [5]: id(b)
Out[5]: 1527379885360
```

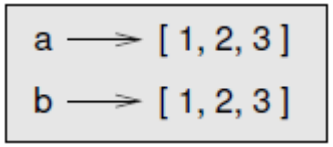
O comando id() apresenta o identificador único associado ao objeto. Como a e b tem o mesmo conteúdo o id é idêntico



Para listas:

```
In [6]: a = [1, 2, 3]

In [7]: b = [1, 2, 3]

In [8]: id(a)
Out[8]: 1527392740672

In [9]: id(b)
Out[9]: 1527392861184
```

No caso das listas os objetos são diferentes. As listas são mutáveis, mesmo com o conteúdo idêntico não apontam para o mesmo objeto.



8.11 Aliasing

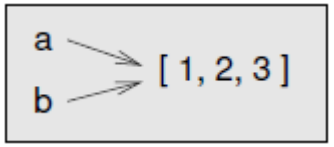
Quando os objetos são associad0s diretamente (aliasing) os id´s se tornam idênticos:

```
In [10]: a = [1, 2, 3]

In [11]: b = a

In [12]: b[0] = 5

In [13]: print(a)
[5, 2, 3]

In [14]: id(a)
Out[14]: 1527392861760

In [15]: id(b)
Out[15]: 1527392861760
```

Quando “b” é associado a “a” ele se torna o objeto lista “a”

Quando o contéudo de “b” é modificado, como ele está “aliased” com “a” ambos apontam para o mesmo conteúdo de lista

Os id´s são idênticos



Essa possibilidade do python deve ser usada com muito cuidado. Por um lado adiciona flexibilidade mas pode levar a comportamentos indesejados.

8.12 Cloning lists (Clonando listas)

Para evitar o aliasing a melhor forma é utilizar a fatia (slice) com o operador [:]

```
In [16]: a = [1, 2, 3]

In [17]: b = a[:]

In [18]: print(b)
[1, 2, 3]

In [19]: id(a)
Out[19]: 1527393673024

In [20]: id(b)
Out[20]: 1527393671616
```

O operador [:] clona o conteúdo e não o objeto

Os id´s são diferentes

Essa prática evita o risco de confusão e uso inadvertido

8.13 List parameters (Listas como parâmetros)

Listas podem ser argumentos de funções e essas funções podem modificar não apenas o conteúdo mas modificar o comprimento das listas

```
def head(list):
    return list[0]

numbers = [1, 2, 3]

print(head(numbers))
```

```
In [21]: runfile('C:/Users/win/
scripts/aula6_lst23.py', wdir='
MAP2112_2023/scripts')
1
```

Apenas o conteúdo é apresentado

```
def deleteHead(list):
    del list[0]

numbers = [1, 2, 3]

deleteHead(numbers)

print(numbers)
```

```
In [22]: runfile('C:/Users/win/Dro
scripts/aula6_lst24.py', wdir='C:/
MAP2112_2023/scripts')
[2, 3]
```

A lista perdeu um elemento e reduziu de tamanho

Listas podem criadas como saídas de funções que manipulam outras listas

```
def tail(list):
    return list[1:]

numbers = [1, 2, 3]

rest = tail(numbers)

print(rest)
```

```
In [24]: runfile('C:/Users/win/Dropbox/
scripts/aula6_lst25.py', wdir='C:/User
MAP2112_2023/scripts')
[2, 3]
```

A lista de saída tem um tamanho menor que a lista original

8.14 Nested lists (Listas aninhadas)

A indexação das listas aninhadas se dá do nível mais alto para o mais baixo, por exemplo:

```
lista = ["hello", 2.0, 5, [10, 20]]

print(lista[3])

elem = lista[3]

print(elem)

print(elem[1])

print(lista[3][1])
```

O elemento [3] de lista é uma lista

Se criamos uma variável intermediária com o elemento [3] da lista a posição [1] é acessada usando elem[1]

Pode-se usar a indexação dupla o 1º índice é o nível mais alto da lista, o 2º o nível seguinte

```
In [25]: runfile('C:/Users/win/Dropbox/USP/2023/
scripts/aula6_lst26.py', wdir='C:/Users/win/Drop
MAP2112_2023/scripts')
[10, 20]
[10, 20]
20
20
```

8.15 Matrices (Matrizes)

Listas aninhadas podem ser usadas para representar matrizes. Por exemplo a matriz :

$$\begin{bmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \\ 7 & 8 & 9 \end{bmatrix}$$

Pode ser representada por:

```
In [26]: matrix = [[1, 2, 3],[4, 5, 6], [7, 8, 9]]

In [27]: matrix[1]
Out[27]: [4, 5, 6]

In [28]: matrix[0][2]
Out[28]: 3
```

Os índices podem ser utilizados para acessar a linhas e os elementos

As operações matriciais na prática são realizadas usando módulos específicos de maior flexibilidade e desempenho (Numpy)

De forma análoga às strings existem os métodos para listas, alguns deles estão indicados na “folha-de-cola” (cheat sheet)

List methods	
list.append(x)	adds x to the end of the list
list.extend(L)	appends L to the end of the list
list.insert(i,x)	inserts x at i position
list.remove(x)	removes the first list item whose value is x
list.pop(i)	removes the item at position i and returns its value
list.clear()	removes all items from the list
list.index(x)	returns a list of values delimited by x
list.count(x)	returns a string with list values joined by S
list.sort()	sorts list items
list.reverse()	reverses list elements
list.copy()	returns a copy of the list

O uso é aplicado especificamente no objeto “list” veremos alguns exemplos em seguida.

Existe uma lista mais completa de métodos para listas disponível em :
https://www.w3schools.com/python/python_lists_methods.asp

```
In [30]: fruits = ['apple', 'banana', 'cherry']

In [31]: fruits.append("orange")

In [32]: fruits
Out[32]: ['apple', 'banana', 'cherry', 'orange']
```

Adiciona elemento no final da lista

```
In [40]: fruits = ['apple', 'banana', 'cherry']

In [41]: cars = ['Ford', 'BMW', 'Volvo']

In [42]: fruits.extend(cars)

In [43]: fruits
Out[43]: ['apple', 'banana', 'cherry', 'Ford', 'BMW', 'Volvo']
```

Adiciona uma lista no final de outra lista

```
In [44]: fruits = ['apple', 'banana', 'cherry']

In [45]: fruits.insert(1, "orange")

In [46]: fruits
Out[46]: ['apple', 'orange', 'banana', 'cherry']
```

Adiciona elemento numa posição dada

```
In [47]: fruits = ['apple', 'banana', 'cherry']

In [48]: fruits.remove("banana")

In [49]: fruits
Out[49]: ['apple', 'cherry']
```

remove elemento dado o conteúdo

```
In [50]: fruits = ['apple', 'banana', 'cherry']

In [51]: fruits.pop(1)
Out[51]: 'banana'

In [52]: fruits
Out[52]: ['apple', 'cherry']
```

remove elemento dada a posição

Usando a atribuição pode-se armazenar o elemento em outra variável

```
In [54]: fruits = ['apple', 'banana', 'cherry', 'orange']

In [55]: fruits.clear()

In [56]: fruits
Out[56]: []
```

Os elementos são removidos resultando numa lista vazia.

```
In [57]: fruits = ['apple', 'banana', 'cherry']

In [58]: x = fruits.index("cherry")

In [59]: x
Out[59]: 2
```

Indica o índice do elemento usando o conteúdo

```
In [63]: fruits = ['apple', 'banana', 'cherry', 'cherry']

In [64]: x = fruits.count("cherry")

In [65]: x
Out[65]: 2
```

Conta quantas vezes o conteúdo aparece dentre os elementos

```
In [66]: cars = ['Ford', 'BMW', 'Volvo']

In [67]: cars.sort()

In [68]: cars
Out[68]: ['BMW', 'Ford', 'Volvo']
```

Ordena os elementos em ordem alfabética se forem strings ou numérica se for o caso

```
In [69]: lista = [3, 4, 9 ,1, 2]

In [70]: lista.sort()

In [71]: lista
Out[71]: [1, 2, 3, 4, 9]
```

Ordena os inteiros

```
In [72]: lista = [2, 1.0, "apple"]

In [73]: lista.sort()
Traceback (most recent call last):

  Cell In[73], line 1
    lista.sort()

TypeError: '<' not supported between instances of 'str' and 'float'
```

Quando a lista contém tipos diferentes temos um erro

```
In [75]: lista = [14, 5, 2.0, 5.0]

In [76]: lista.sort()

In [77]: lista
Out[77]: [2.0, 5, 5.0, 14]
```

Floats e inteiros podem ser ordenados sem problema

```
In [78]: fruits = ['apple', 'banana', 'cherry']

In [79]: fruits.reverse()

In [80]: fruits
Out[80]: ['cherry', 'banana', 'apple']
```

A ordem dos elementos é invertida

```
In [81]: fruits = ['apple', 'banana', 'cherry', 'orange']

In [82]: x = fruits.copy()

In [83]: x
Out[83]: ['apple', 'banana', 'cherry', 'orange']
```

A lista é copiada

Python 3 Beginner's Reference Cheat Sheet

Alvaro Sebastian
http://www.sixthresearcher.com

Main data types

boolean = *True* / *False*
integer = 10
float = 10.01
string = "123abc"
list = [value1, value2, ...]
dictionary = { key1:value1, key2:value2, ...}

Numeric operators

+	addition
-	subtraction
*	multiplication
/	division
**	exponent
%	modulus
//	floor division

Comparison operators

==	equal
!=	different
>	higher
<	lower
>=	higher or equal
<=	lower or equal

Boolean operators

and	logical AND
or	logical OR
not	logical NOT

Special characters

#	coment
\n	new line
\<char>	scape char

String operations

string[i]	retrieves character at position i
string[-1]	retrieves last character
string[i:j]	retrieves characters in range i to j

List operations

list = []	defines an empty list
list[i] = x	stores x with index i
list[i]	retrieves the item with index I
list[-1]	retrieves last item
list[i:j]	retrieves items in the range i to j
del list[i]	removes the item with index i

Dictionary operations

dict = {}	defines an empty dictionary
dict[k] = x	stores x associated to key k
dict[k]	retrieves the item with key k
del dict[k]	removes the item with key k

String methods

string.upper()	converts to uppercase
string.lower()	converts to lowercase
string.count(x)	counts how many times x appears
string.find(x)	position of the x first occurrence
string.replace(x,y)	replaces x for y
string.strip(x)	returns a list of values delimited by x
string.join(L)	returns a string with L values joined by string
string.format(x)	returns a string that includes formatted x

List methods

list.append(x)	adds x to the end of the list
list.extend(L)	appends L to the end of the list
list.insert(i,x)	inserts x at i position
list.remove(x)	removes the first list item whose value is x
list.pop(i)	removes the item at position i and returns its value
list.clear()	removes all items from the list
list.index(x)	returns a list of values delimited by x
list.count(x)	returns a string with list values joined by S
list.sort()	sorts list items
list.reverse()	reverses list elements
list.copy()	returns a copy of the list

Dictionary methods

dict.keys()	returns a list of keys
dict.values()	returns a list of values
dict.items()	returns a list of pairs (key,value)
dict.get(k)	returns the value associtated to the key k
dict.pop()	removes the item associated to the key and returns its value
dict.update(D)	adds keys-values (D) to dictionary
dict.clear()	removes all keys-values from the dictionary
dict.copy()	returns a copy of the dictionary

Legend: x,y stand for any kind of data values, s for a string, n for a number, L for a list where i,j are list indexes, D stands for a dictionary and k is a dictionary key.

Python 3 Beginner's Reference Cheat Sheet

Alvaro Sebastian
http://www.sixthresearcher.com

Built-in functions

print(x, sep='y')	prints x objects separated by y
input(s)	prints s and waits for an input that will be returned
len(x)	returns the length of x (s, L or D)
min(L)	returns the minimum value in L
max(L)	returns the maximum value in L
sum(L)	returns the sum of the values in L
range(n1,n2,n)	returns a sequence of numbers from n1 to n2 in steps of n
abs(n)	returns the absolute value of n
round(n1,n)	returns the n1 number rounded to n digits
type(x)	returns the type of x (string, float, list, dict ...)
str(x)	converts x to string
list(x)	converts x to a list
int(x)	converts x to a integer number
float(x)	converts x to a float number
help(s)	prints help about x
map(function, L)	Applies function to values in L

Conditional statements

```
if <condition> :
    <code>
else if <condition> :
    <code>
...
else:
    <code>

if <value> in <list>:
```

Data validation

```
try:
    <code>
except <error>:
    <code>
else:
    <code>
```

Working with files and folders

```
import os
os.getcwd()
os.makedirs(<path>)
os.chdir(<path>)
os.listdir(<path>)
```

Loops

```
while <condition>:
    <code>

for <variable> in <list>:
    <code>

for <variable> in
range(start,stop,step):
    <code>

for key, value in
dict.items():
    <code>
```

Loop control statements

break	finishes loop execution
continue	jumps to next iteration
pass	does nothing

Running external programs

```
import os
os.system(<command>)
```

Functions

```
def function(<params>):
    <code>
    return <data>
```

Modules

```
import module
module.function()

from module import *
function()
```

Reading and writing files

```
f = open(<path>,'r')
f.read(<size>)
f.readline(<size>)
f.close()

f = open(<path>,'r')
for line in f:
    <code>
f.close()

f = open(<path>,'w')
f.write(<str>)
f.close()
```

Legend: x,y stand for any kind of data values, s for a string, n for a number, L for a list where i,j are list indexes, D stands for a dictionary and k is a dictionary key.

Fim Aula 06