Fundamentos da Programação

Listas

Aula 10

José Monteiro

(slides adaptados do Prof. Alberto Abad)

Na semana passada aprendemos...

- A definir funções em Python e como as utilizar
  - conceito de abstração procedimental
- Um primeiro tipo estruturado de dados, o `tuplo` , e operações sobre estes
- Ciclos contados, nomeadamente a instrução `for`
  - introduzimos a função embutida `range` para gerar sequências
- Operações sobre cadeias de carateres
- A escrita formatada de dados

Listas

- Em Python, uma lista (`list`) é uma sequência de elementos (como os tuplos), mas como uma diferença fundamental: as listas são **mutáveis**.
  - É possível alterar / eliminar / acrescentar elementos

`

::= [] | []

::= | ,

::= | | | `

- Tal como nos tuplos, o índice do primeiro elemento da lista é 0.
- Listas de um elemento: `[e]` (não há ambiguidade como no caso dos tuplos).

Operações com Listas

Operação	*Tipo dos argumentos*	*Valor*
l_1 `+` l_2	Listas	A concatenação das listas l_1 e l_2.
l `*` i	Lista e inteiro	A repetição i vezes da lista l.
l`[`i_1`:`i_2`]`	Lista e inteiros	A sub-lista de l entre os índices i_1 e $i_2 - 1$.
`del(`els`)`	Lista e inteiro(s)	Em que els pode ser da forma l`[`i`]` ou l`[`i_1`:`i_2`]`. Remove os elemento(s) especificado(s) da lista l.
e `in` l	Universal e lista	`True` se o elemento e pertence à lista l; `False` em caso contrário.
e `not in` l	Universal e lista	A negação do resultado da operação e `in` l.
`list(`a`)`	Tuplo ou dicionário ou cadeia de caracteres	Transforma o seu argumento numa lista. Se não forem fornecidos argumentos, o seu valor é a lista vazia.
`len(`l`)`	Lista	O número de elementos da lista l.

Operações com Listas

Exemplos:

```
lst1 = [2, 3, 5, 7]
lst2 = [0, 1, 1, 2, 3, 5, 8]
lst1 + lst2
lst1 * 5
lst2[2:5]
```

```
8 in lst2
8 in lst1
```

```
len(lst1)
list((8,9,4))
list('Fundamentos')
```

```
lst1[1] = 'FP'
lst1[2] = [1, 2, 3]
```

In [ ]:

Mais Operações com Listas

Operação	Tipo dos	Valor
`l[i] = e`	Lista, inteiro, universal	Elemento i de l é substituído pelo valor de e.
`l[i:j:k] = t`	Lista, inteiros, iterável	Elementos de i a $j-1$ de l com índices espaçados de k são substituídos pelos elementos de t (i, j, k são opcionais)
`del l[i:j:k]`	Lista, inteiros	Remove os elementos de l com índices entre i e $j-1$ espaçados de k (i, j, k são opcionais).
`l.append(e)`	Lista, universal	Acrescenta um elemento ao final da lista l com o valor de e.
`l.extend(t)` ou `l += t`	Lista, iterável	Acrescenta os elementos de t no final da lista l.
`l.clear()`	Lista	Remove todos os elementos da lista l.
`l.copy()`	Lista	Devolve uma cópia (*shallow*) da lista l.
`l *= n`	Lista, inteiro	Transforma l em n cópias de si mesma.
`l.insert(i, e)`	Lista, inteiro, universal	Insere um novo elemento em l na posição i com o valor e.
`l.pop(i)`	Lista, inteiro	Retorna o valor na posição i e remove-o da lista l. Sem parâmetro, remove o último elemento de l.
`l.remove(e)`	Lista, universal	Remove da lista l o primeiro elemento igual a e, `ValueError` se não existe.
`l.reverse()`	Lista	Coloca elementos de l por ordem inversa.

Referência: https://docs.python.org/3/library/stdtypes.html#typesseq-mutable

Mais Operações com Listas

Exemplos:

```
lst1 = [2, 3, 5, 7]
lst2 = [0, 1, 1, 2, 3, 5, 8]

del(lst2[3:5])
del lst2[3:5]
del lst2[-1]
lst1.append((4,6))
lst1.extend((4,6))

lst1.insert(1,True)
e = lst2.pop()
```

In [ ]:

Mais Operações com Listas

Exemplos:

```
lst1 = [2, 3, 5, 7]
lst2 = [0, 1, 1, 2, 3, 5, 8]

del(lst2[3:5])
del lst2[3:5]
del lst2[-1]

lst1.append((4,6))
lst1.extend((4,6))

lst1.insert(1,True)
```

In [ ]:

Listas: Acrescentar Elementos

In [ ]:

```
lst1 = [1,2,3]
lst2 = lst1
print(id(lst1))
for i in range(10, 20):
    # lst1.append(i)
    lst1 = lst1 + [i]

print(lst1)
print(id(lst1))
print(id(lst2))
lst2
```

Atribuição em Listas

In [ ]:

```
lst1 = [2, 3, 5, 7]
lst2 = lst1

print(id(lst1))
print(id(lst2))

lst1[2] = 6
lst2[1] = 4

lst1 = 10

print(lst1, id(lst1))
print(lst2, id(lst2))
```

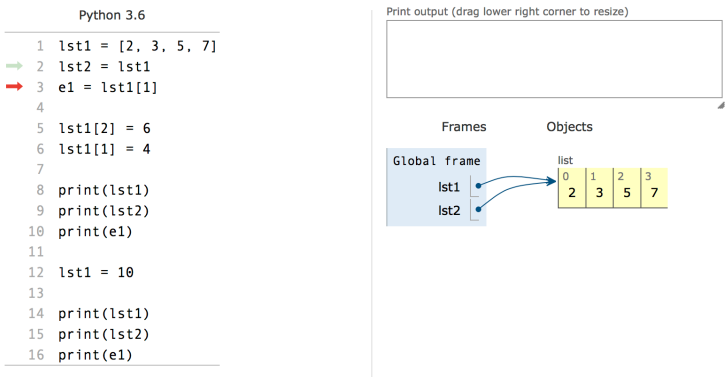
Atribuição em Listas

In [ ]:

```
lst1 = [2, 3, 5, 7]
lst2 = list(lst1)
lst3 = lst2[:]
lst4 = lst3.copy()

lst2[0] = -1
lst3[1] = -2
lst4[2] = -3

print(lst1, id(lst1))
print(lst2, id(lst2))
print(lst3, id(lst3))
print(lst4, id(lst4))
```

Atribuição em listas: Considerações sobre mutabilidade

Python Tutor



Passagem de Parâmetros

- Modo de passagem de parâmetros mais comuns em programação:
  - Por valor - A função recebe o valor do parâmetro concreto e mais nenhuma informação
  - Referência - A função recebe a posição em memória do parâmetro concreto

- Em Python é um pouco diferente:
  - Os parâmetros são passados por *cópia do valor da referência dos objetos* ou *assignment*.
  - *Assignment* é a operação de ligar (*binding*) um nome a um objeto.
  - Implicações:
    - Podemos alterar/mudar os objetos que sejam mutáveis.
    - Não podemos fazer *rebinding* da referência externa, ou seja, ligar o nome da variável do ambiente exterior da função a um outro objeto.

- Leitura adicional:
  - https://docs.python.org/3/faq/programming.html#how-do-i-write-a-function-with-output-parameters-call-by-reference

Passagem de Parâmetros / Retorno de Valores de uma Função

- return
- global
- listas

Passagem de Parâmetros

Exemplo parâmetros imutáveis

In [ ]:
```
def func1(a, b):
    print("DENTRO ANTES da troca:", a, b)
    a, b = 'new-value', b + 1                # a and b are local names
    print("DENTRO DEPOIS da troca:", a, b)   # assigned to new objects

a, b = 'old-value', 99
print("FORA ANTES da troca:", a, b)
func1(a, b)
print("FORA DEPOIS da troca:", a, b)
```

Passagem de Parâmetros

Exemplo parâmetros mutáveis 1

In [ ]:
```
def func2(l):
    print("DENTRO ANTES da troca:", l)
    l = ['outro', 101]
    l[0], l[1] = 'new-value', l[1] + 1     # 'a' references a mutable list
    print("DENTRO DEPOIS da troca:", l)

l = ['old-value', 99]
print("FORA ANTES da troca:", l)
func2(l)
print("FORA DEPOIS da troca:", l)
```

Passagem de Parâmetros

Exemplo parâmetros mutáveis 2

In [ ]:

```
def func3(l):
    print("DENTRO ANTES da troca:", l)
    l[0], l[1] = 'new-value', l[1] + 1     # 'a' references a mutable list
    print("DENTRO DEPOIS da troca:", l)
    l = ['last new-value', l[1] + 1]       # rebinding example
    print("DENTRO DEPOIS da assignment:", l)

l = ['old-value', 99]
print("FORA ANTES da troca:", l)
func3(l)
print("FORA DEPOIS da troca:", l)
```

Passagem de Parâmetros

Exemplo parâmetros mutáveis !?!?

In [ ]:

```
def addtoall(t, x):
    for l in t:
        l.append(x)

t = ([],[], [])
addtoall(t, 2)
addtoall(t, 17)
addtoall(t, 4)
print(t)
```

Listas

Exemplo: Verificar IMEI válido

- O IMEI (International Mobile Equipment Identity) é um número de 15 dígitos, geralmente exclusivo, para identificar telefones móveis (marcar *#06#* para o obter). O dígito mais à direita é um dígito de verificação ou *checksum*.
- O algoritmo Luhn ou a fórmula Luhn, também conhecido como algoritmo *módulo 10*, é uma fórmula de *checksum* simples usada para validar uma variedade de números de identificação, como números de cartão de crédito, números IMEI, etc.
- O algoritmo de Luhn verifica um número em relação ao seu dígito de verificação. O número deve passar no seguinte teste:
  - A partir do dígito mais à direita, que é o dígito de verificação, e movendo para a esquerda, dobrar o valor de cada segundo dígito. Se o resultado dessa operação de duplicação for maior que 9 (por exemplo, 8 × 2 = 16), adicionar os dígitos do número (por exemplo, 16: 1 + 6 = 7, 18: 1 + 8 = 9) ou, alternativamente, o mesmo resultado pode ser encontrado ao subtrair 9 (por exemplo, 16: 16 - 9 = 7, 18: 18 - 9 = 9).
  - Calcular a soma de todos os dígitos, incluindo o dígito de verificação
  - Se o módulo 10 total é igual a 0 então o número é **válido** de acordo com a fórmula de Luhn; senão é **não válido**.

Account number	7	9	9	2	7	3	9	8	7	1	**x**
Double every other	7	**18**	9	**4**	7	**6**	9	**16**	7	**2**	**x**
Sum digits	7	**9**	9	4	7	6	9	**7**	7	2	**x**

Listas: Exemplo

Exemplo: Verificar IMEI válido - Proposta solução 1:

- Criar uma lista com os digitos do IMEI, alterar os elementos e fazer a soma

In [ ]:

```
def is_valid_imei(imei):
    """
    checks if an imei is valid.
    """

    if (not isinstance(imei, str)) or len(imei) != 15:
        raise ValueError("Doesn't look like a serial number!")

    imei = list(imei)
    for i in range(len(imei)):
        imei[i] = int(imei[i])

    for i in range(len(imei) - 2, -1, -2):
        imei[i] = imei[i] * 2

    # print(imei)

    for i in range(len(imei)):
        if imei[i] >= 10:
            imei[i] -= 9

    # print(imei)

    acc = 0
    for i in range(len(imei)):
        acc = acc + imei[i]

    return acc % 10 == 0

is_valid_imei("353270079684223")
```

Listas: Exemplo

Exemplo: Verificar IMEI válido - Proposta de solução 2

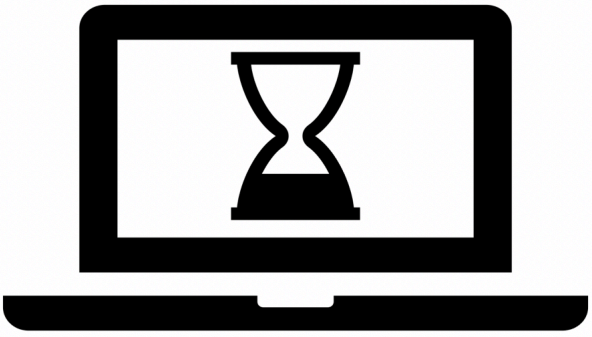
- Criar uma lista e obter a soma sem alterar os elementos

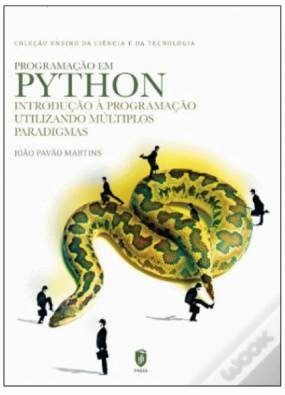
In [ ]:

```
def is_valid_imei2(imei):
    """
    checks if an imei is valid.

    """

    if (not isinstance(imei, str)) or len(imei) != 15:
        raise ValueError("Doesn't look like a serial number!")

    imei = list(imei)
    acc = 0
    factor = 1
    for i in range(len(imei) - 1, -1, -1):
        double = int(imei[i]) * factor
        acc += ((double - 9) if double > 9 else double)
        factor = 1 if factor == 2 else 2


    return acc % 10 == 0

is_valid_imei2("353270079684223")
```

Listas - Tarefas próxima aula

- Trabalhar matéria apresentada hoje:
  - Experimentar todos os programas dos slides
- Projeto!!
- Nas aulas de problemas ==> tuplos e listas





In [ ]: