Figure 1: Calvin e Hobbes de Bill Watterson

18 Reunião 18: 22/JUN/2021

18.1 Reuniões passadas

- listas
- criar listas
- acessar itens da lista através de **índices**
- percorrer listas
- função `len()`
- comando de repetição `for ... in range(ini, fim, passo):...`

18.2 Hoje

Figure 2: Fonte: Dreamstime.com

18.3 Exercício 1: soma e fatia

Uma **fatia** de uma lista é uma sublista especificada por um par de índices `ini` e `fim` que indicam início de fim (*exclusive*) da sublista. Por exemplo, para a lista

```
          [-1,  23, 'oi', True, 2.71, None, 45]
#indices   0    1    2      3     4     5   6
```

a fatia de:

- `2` a `5` é a sublista `['oi', True, 2.71]`
- `3` a `7` é a sublista `[True, 2.71, None, 45]`
- `0` a `1` é a sublista `[-1]`
- `4` a `4` é a sublista vazia

(a) Escreva uma função `soma()` que **recebe** um lista de números e retorna a soma de seus elementos. Por exemplo,

```
In [15]: soma([1,2,3])
Out[15]: 6

In [16]: soma([-1,2,-3])
Out[16]: -2
```

(b) Escreva função `fatia()` que **recebe** uma lista `lst` e indices `ini` e `fim` e **retorna** uma lista formada pela sublista de `lst` com índices entre `ini` e `fim`. Por exemplo,

```
In [12]: lst = [1, 'oi', True, 3.14, 19, None]

In [13]: fatia(lst, 1, 4)
Out[13]: ['oi', True, 3.14]

In [14]: fatia(lst, 2, len(lst))
Out[14]: [True, 3.14, 19, None]
```

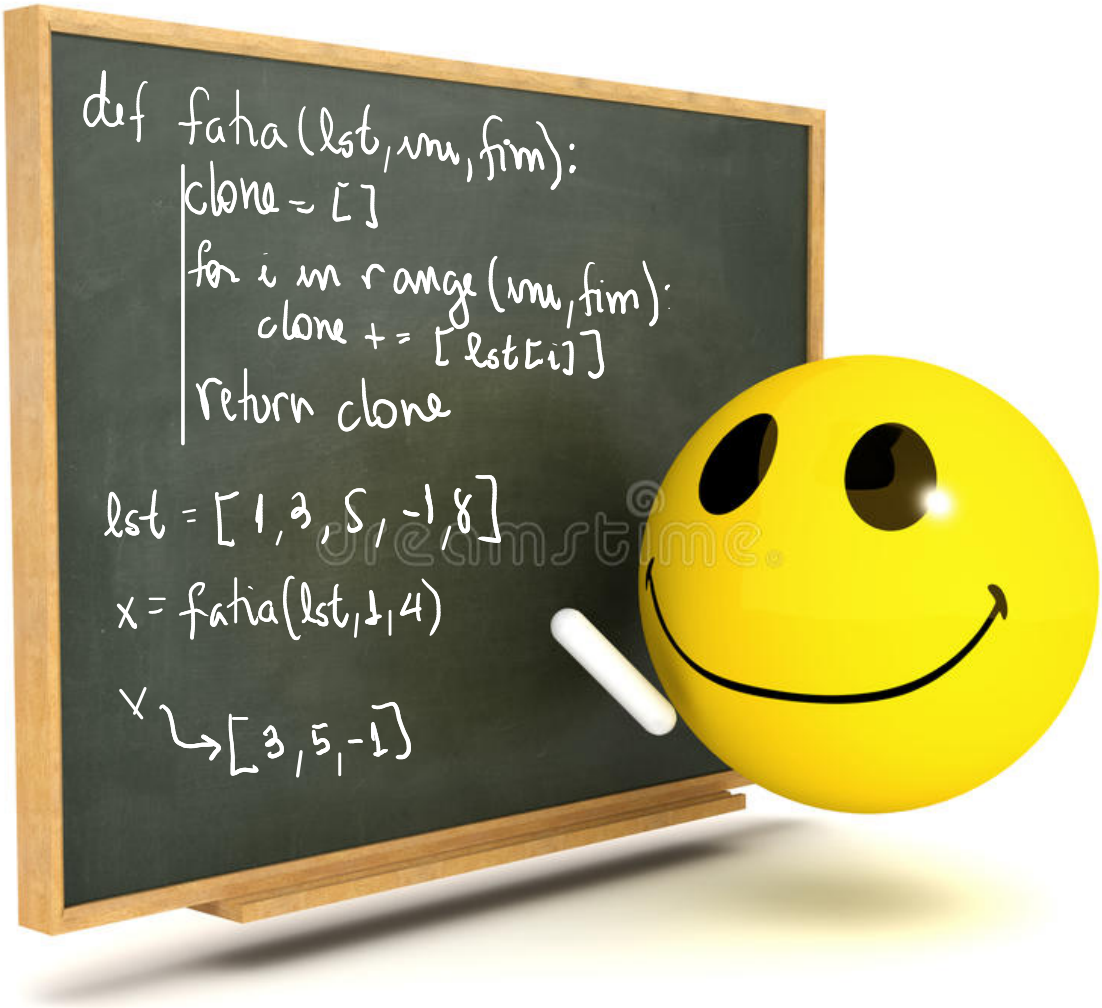


Figure 3: Fonte: Dreamstime.com

Solução

```
def main():
    '''() -> None
    Unidade de teste para as funções soma e fatia.
    '''
    print("Teste soma()")
    lst = [1, -1, 4]
    print(f"soma({lst}) = {soma(lst)}")
    lst = [-2.1, 1.3]
    print(f"soma({lst}) = {soma(lst)}")
    lst = []
    print(f"soma({lst}) = {soma(lst)}")

    print("\nTeste fatia()")
    lst = [-1,  23, 'oi', True, 2.71, None, 45]
    print(f"fatia({lst}, 0,{len(lst)}) = {fatia(lst, 0 ,len(lst))}")
    print(f"fatia({lst}, 2, 5)) = {fatia(lst, 2, 5)}")
    print(f"fatia({lst}, 3, 7)) = {fatia(lst, 3, 7)}")
    print(f"fatia({lst}, 4, 4)) = {fatia(lst, 4, 4)}")
```

```python
def soma(lst):
    '''(list) -> int
    RECEBE uma lista lst.
    RETORNA a soma dos itens na lista.
    In [15]: soma([1, 2, 3])
    Out[15]: 6
    In [16]: soma([-1, 2, -3])
    Out[16]: -2
    '''
    soma = 0
    n = len(lst)
    for i in range(0, n, 1):
        soma += lst[i]
    return soma
#-------------------------------------------------------------
def fatia(lst, ini, fim):
    '''(list, int, int) -> lst

    RECEBE uma lista lst e indices ini e fim.
    RETORNA uma lista formada pela sublista de lst com
        indices entre ini e fim.
    In [12]: lst = [1, 'oi', True, 3.14, 19, None]
    In [13]: fatia(lst, 1, 4)
    Out[13]: ['oi', True, 3.14]
    In [14]: fatia(lst, 2, len(lst))
    Out[14]: [True, 3.14, 19, None]
    '''
    sublista = []
    for i in range(ini, fim, 1):
        sublista = sublista + [lst[i]]
    return sublista

# inicio do programa
if __name__ == "__main__":
    main()
```

supérfluo o mesmo que range(n)

supérfluo, o mesmo que range(ini, fim)

18.4 Fatias em Python

Em Python uma fatia de uma lista é especificada por um intervalo de índices

```
lst[início:fim] # o mesmo que fatia(lst,início,fim)
```

e é um clone da sublista.

Podemos usar : para definir qualquer *fatia* de uma lista. Por exemplo,

```
lst1[1:3] retorna a lista [1, 2]
lst1[:] é uma abreviatura de lst1[início:len(lst1)]
```

abreviatura muito usada

```
def main():
    lst1 = [0, 1, 2, 3, 4]
    lst2 = lst1[:] # faz uma clone da lista
    lst2[1] = 7
    print(f"lst1 = {lst1}")
    print(f"lst2 = {lst2}")

main()
```

Mais exemplos

```
[8]: lst = [0, 1, 2, 3, 4]

In [9]: lst[0: 5]
Out[9]: [0, 1, 2, 3, 4]

In [10]: lst[1: 5]
Out[10]: [1, 2, 3, 4]

In [11]: lst[1: 4]
Out[11]: [1, 2, 3]

In [12]: lst[1: 1]
Out[12]: []
```

```
In [13]: lst[ : ]
Out[13]: [0, 1, 2, 3, 4]
```

18.5 Somas em Python

Python possui a função nativa **`sum()`** que calcula e retorna a soma dos itens em uma lista

Exemplos

```
In [17]: lst = [1.1, 2, 3, -1, 4]

In [18]: sum(lst)
Out[18]: 9.1

In [19]: sum(lst[1:])
Out[19]: 8

In [20]: sum(lst[2:])
Out[20]: 6

In [21]: sum(lst[3:])
Out[21]: 3

In [22]: sum(lst[4:])
Out[22]: 4

In [23]: sum(lst[5:])
Out[23]: 0

In [24]: sum([])
Out[24]: 0
```

18.6 Exercício 2: fatia de soma máxima

Dados `n > 0` e uma sequência com de `n` números inteiros, determinar uma fatia de soma máxima da lista formada pelos números da sequência.

Exemplos

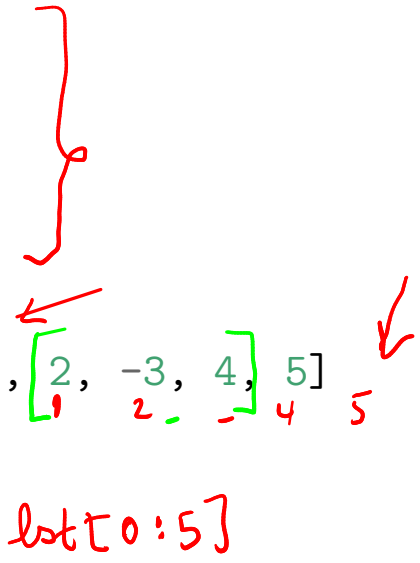
```
Digite n: 12
Digite o 1o. número: 5
Digite o 2o. número: 2
Digite o 3o. número: -2
Digite o 4o. número: -7
Digite o 5o. número: 3
Digite o 6o. número: 14
Digite o 7o. número: 10
Digite o 8o. número: -3
Digite o 9o. número: 9
Digite o 10o. número: -6
Digite o 11o. número: 4
Digite o 12o. número: 1
lista = [5, 2, -2, -7, 3, 14, 10, -3, 9, -6, 4, 1]
fatia de soma máxima= [3, 14, 10, -3, 9]
soma= 33
ini = 4
fim = 9
```

feita

```
Digite n: 5
Digite o 1o. número: 1
Digite o 2o. número: 2
Digite o 3o. número: -3
Digite o 4o. número: 4
Digite o 5o. número: 5
lista = [1, 2, -3, 4, 5]
fatia de soma máxima= [1, 2, -3, 4, 5]
soma= 9
ini = 0
fim = 5

Digite n: 5
Digite o 1o. número: -1
Digite o 2o. número: 2
Digite o 3o. número: -3
Digite o 4o. número: 4
Digite o 5o. número: 5
lista = [-1, 2, -3, 4, 5]
fatia de soma máxima= [4, 5]
soma= 9
ini = 3
fim = 5

Digite n: 5
Digite o 1o. número: 1
Digite o 2o. número: -2
Digite o 3o. número: 3
Digite o 4o. número: -4
Digite o 5o. número: 5
lista = [1, -2, 3, -4, 5]
fatia de soma máxima= [5]
soma= 5
ini = 4
fim = 5
```



Proposta de solução

(a) Escreva uma funçao `leia_seq()` que **lê** um inteiro positivo `n` e uma sequência de `n` números inteiros e retorna a lista formada por esses números.

(b) Escreva uma função `fatia_max()` que **recebe** uma lista de números inteiros e **retorna** o indices do início e do fim de uma fatia de soma máxima e o valor da soma máxima. Por exemplo,

```
In [18]: fatia_max([1,-2, 3, -2, 5])
Out[18]: (2, 5, 6)

In [19]: fatia_max([10,-2,-5,-2, 10])
Out[19]: (0, 5, 11)
```

(c) Escreva uma função `main()` que lê um inteiro `n > 0` e uma seqüência `n` números inteiros e calcula a maior soma de uma de suas fatias.

Solução

```
def main():
    '''() -> None
    Programa que lê n > 0 e uma sequência n
    números inteiros e calcula a maior soma de
    uma de suas fatias.
    '''
    lst = leia_seq()
    print(f"lista = {lst}")
    ini, fim, soma = fatia_max(lst)
    print(f"fatia de soma máxima= {lst[ini:fim]}")
    print(f"soma= {soma}")
    print(f"ini = {ini}")
    print(f"fim = {fim}")

#-------------------------------------------------------------
def leia_seq():
    '''() -> list
```

```
    LÊ um número inteiro positivo n e uma sequências
       com n números inteiros.
    RETORNA a lista com os números da sequência na ordem em que
        foram lidos.
    '''
    lst = []
    n = int(input("Digite n: "))
    for i in range(n):
        num = int(input(f"Digite o {i+1}do. número: "))
        lst += [num]
    return lst

#-----------------------------------------------------------
def fatia_max(lst):
    '''(list) -> int, int, int
    RECEBE uma lista de números inteiros.
    RETORNA o indices do início e do fim de uma fatia de soma
        máxima e o valor da soma máxima
    '''
    n = len(lst)
    smax = 0
    ini = 0
    fim = 0
    for i in range(n): # o mesmo que range(0, n, 1):
        for j in range(i+1,n+1): # o mesmo que range(i+1, n+1, 1):
            s = sum(lst[i:j])
            if  s > smax:
                smax = s
                ini  = i
                fim  = j
    return ini, fim, smax

# inicio do programa
if __name__ == "__main__":
    main()
```

18.7 Apelidos versus Clones

Considere as seguintes listas

```
lstA = [True, 5, 3.14, None, 'fim']
lstB = lstA
n = len(lstA)
lstC = lstA[0:n:1] # fatia [ini:fim:passo]
lstD = lstA[0:n] # [ini:fim] passo é 1
lstE = lstA[0:n] # [:fim ] ini é 0 e passo é 1
lstF = lstA[ : ] # [:] ini é 0, fim é len(lstA) e passo é 1
lstG = lstA[0:n:2]  #  ...
```

Apelidos

Variáveis são apelidos para *coisas*. Acima, `lstA` e `lstB` são *apelidos*, nomes associados a uma **mesma lista**.

Duas lista são iguais se têm os mesmos itens na mesma ordem

```
In [30]: X = ['a', True, 4, 3.1, None]

In [31]: Y = ['a', True, 4, 3.1, None]

In [32]: X == Y
Out[32]: True
```

Escrevendo em Python:

```
if lstA == lstB:
   print("listas têm os mesmos valores, na mesma ordem")
else:
   print("listas tem valores diferentes")
```

Operador `is`

`X is Y` é verdadeiro se `X` e `Y` são apelidos para a mesma coisa. É evidente que se `X is Y` é `True`, então `X == Y` é `True`.

```
In [25]: X = [1,2,3]

In [26]: Y = X

In [27]: Y
Out[27]: [1, 2, 3]

In [28]: X == Y
Out[28]: True

In [29]: X is Y
Out[29]: True
```

Já se duas coisas são iguais elas podem não ser a mesma coisa. Ou seja, `X == Y` pode ser `True`, apesar de `X is Y` ser `False`.

```
In [33]: X = ['a', True, 4, 3.1, None]

In [34]: Y = ['a', True, 4, 3.1, None]

In [35]: X == Y
Out[35]: True

In [36]: X is Y
Out[36]: False

In [37]: X[0] = 111

In [38]: X
Out[38]: [111, True, 4, 3.1, None]

In [39]: Y
Out[39]: ['a', True, 4, 3.1, None]
```

Escrevendo em Python

```
if lstA == lstB:
```

```
    if lstA is lstB:
        print("Uau, lstA e lstB são a mesma coisa!")
    else:
        print("lstA e lstB têm os mesmos valores, mas não são a me
else:
    print("listas tem valores diferentes")
```

18.8 Operador `in list`

O operador **`in`** pode ser usado para verificar se um item esta na lista

`item in lst` → é fácil escrevermos uma função que faço (mais ou menos) isso

é **`True`** se **`item`** é um elemento da lista **`lst`** e **`False`** em caso contrário.

Por exemplo

```
5 in [2, 3, 5, 7, 11]  == True
True in [True]  == True
True in [False] == False
True in [1, 2, 5, None] == False
None in [1, 2, 5, None] == True
```

```
if 5 in [3, 2, 7, 5, True]:
    print('5 está na lista')
else:
    print('5 não está na lista')
```

```
if False not in [3, 2, 7, 5, True]:
    print('False não está na lista')
else:
    print('False está na lista')
```