TÉCNICO LISBOA

Fundamentos de Programação - 2018/2019 Aula Prática 07 (30 minutos) Turno 5ª feira 10:30-12:00
Nome:
Número:
Data:
Curso:

Capítulo 6.2 - Funções de ordem superior

Utilizando funcionais sobre listas, escreva a função `num_pred_digitos`, que recebe um predicado *pred* e uma lista *l* de dígitos e devolve o número composto pelos dígitos de *l* que satisfazem o predicado. Pode assumir que a lista *l* tem pelo menos um dígito que satisfaz o predicado *pred*. A sua função deve conter apenas uma instrução, a instrução `return`. Por exemplo:

```
>>> num_pred_digitos(lambda x : x > 3, [1, 2, 3, 4, 5])
45
>>> num_pred_digitos(lambda x : x > 3, [7, 2, 6, 4, 3])
764
```

Solução 1:

```
def num_pred_digitos(pred, l):
    return acumula(filtra(l, pred), lambda x,y: 10*x + y)
```

Solução 2 (usando funcionais do Python):

```
def num_pred_digitos(pred, l):
    return reduce(lambda x,y: 10*x + y, filter(pred, l))
```

Solução 3:

```
def num_pred_digitos(pred, l):
    return int(acumula(transforma(filtra(l,
        pred), str), lambda x,y: x+y)
```

TÉCNICO LISBOA

Fundamentos de Programação - 2018/2019 Aula Prática 07 (30 minutos) Turno 4ª feira 08:00-09:30
Nome:
Número:
Data:
Curso:

Capítulo 6.2 - Funções de ordem superior

Utilizando os funcionais sobre listas escreva a função `soma_multiplos_lista`, que recebe uma lista e um número *n*, e devolve o somatório dos elementos da lista que são múltiplos de *n*. A sua função deve conter apenas uma instrução, a instrução `return`. Por exemplo:

```
>>> soma_multiplos_lista([1, 2, 3, 4, 5, 6, 7, 8, 9], 3)
18
```

Solução:

```
def soma_multiplos_lista(lst, n):
    return acumula(lambda x, y: x + y, \
                   filtra(lambda x: x % n == 0, lst))
```

TÉCNICO LISBOA

Fundamentos de Programação - 2018/2019 Aula Prática 07 (30 minutos) Turno 6ª feira 14:00-15:30
Nome:
Número:
Data:
Curso:

Capítulo 6.2 - Funções de ordem superior

Utilizando os funcionais sobre listas escreva a função `conta_pares`, que recebe uma lista de inteiros e devolve o número de elementos pares da lista. A sua função deve conter apenas uma instrução, a instrução `return`. Por exemplo:

```
>>> conta_pares([1, 2, 3, 4, 5, 6])
3
```

Solução 1:

```
def conta_pares(lst):
    return len(filtra(lst, lambda x: x % 2== 0))
```

Solução 2:

```
def conta_pares(lst):
    return acumula(transforma(filtra(lst, \
        lambda x: x % 2 == 0), \
        lambda x: 1), \
        lambda x,y: x+y)
```

TÉCNICO LISBOA

Fundamentos de Programação - 2018/2019 Aula Prática 07 (30 minutos) Turno 2ª feira 09:00-10:30
Nome:
Número:
Data:
Curso:

Capítulo 6.2 - Funções de ordem superior

Utilizando os funcionais sobre listas escreva a função `num_occ`, que recebe uma lista e um elemento (que pode ser de qualquer tipo), e devolve o número de vezes que o elemento ocorre na lista. A sua função deve conter apenas uma instrução, a instrução `return`. Por exemplo:

```
>>> num_occ([1, 5, [5], 4, 5], 5)
2
>>> num_occ([1, 5, [5], 4, 5], [5])
1
```

Solução 1:

```
def num_occ(lst,el):
    return len(filtra(lambda x: x == el, lst))
```

Solução 2:

```
def num_occ_2(lst, el):
    return acumula(lambda x, y: x + y, \
                   transforma(lambda x: 1, \
                              filtra(lambda x: x == el, lst)))
```

TÉCNICO LISBOA

Fundamentos de Programação - 2018/2019 Aula Prática 07 (30 minutos) Turno 6ª feira 15:30-17:00
Nome:
Número:
Data:
Curso:

Capítulo 6.2 - Funções de ordem superior

Utilizando funcionais sobre listas escreva uma função de ordem superior, `conta_p`, que recebe um número inteiro positivo *n* e um predicado *p*, e devolve o número de inteiros positivos menores ou iguais a *n* que satisfazem o predicado *p*. Por exemplo:

```
>>> conta_p(87, lambda x: x % 100 == 0)
0
>>> conta_p(487, lambda x: x % 100 == 0)
4
```

Solução 1:

```
def conta_p(num, pred):
    return len(filtra(pred, list(range(1, num+1))))
```

Solução 2:

```
def conta_p(num, pred):
   return acumula(transforma(filtra(pred, \
       list(range(1, num+1))), \
       lambda x : 1), \
       lambda x,y : x+y)
```

TÉCNICO LISBOA

Fundamentos de Programação - 2018/2019 Aula Prática 07 (30 minutos) Turno 6ª feira 12:30-14:00
Nome:
Número:
Data:
Curso:

Capítulo 6.2 - Funções de ordem superior

Usando funcionais sobre listas, escreva a função `soma_impares`, que recebe uma lista de inteiros e devolve a soma dos elementos ímpares da lista recebida. A sua função deve conter apenas uma instrução, a instrução `return`. Por exemplo:

```
>>> soma_impares([1, 2, 3, 4, 5])
9
```

Solução:

```
def soma_impares(lst):
    return acumula(lambda x,y : x + y, \
         filtra(lambda x : x % 2 != 0, lst))
```

TÉCNICO LISBOA

Fundamentos de Programação - 2018/2019 Aula Prática 07 (30 minutos) Turno Xª feira XX:XX-XX:XX
Nome:
Número:
Data:
Curso:

Capítulo 6.2 - Funções de ordem superior

Utilizando funcionais sobre listas, escreva a função `concatena`, que recebe um predicado *pred* e uma lista *l* e concatena todas as listas em *l* que satisfazem o predicado. Pode assumir que a lista *l* não é vazia e só tem listas como elementos. A sua função deve conter apenas uma instrução, a instrução `return`. Por exemplo:

```
>>> concatena(lambda l : len(l) >= 2, [[1], [2, 3], [4, 5]])
[2, 3, 4, 5]
>>> concatena(lambda l : 2 in l, [[7, 2], [6, 4, 3], [5, 2]])
[7, 2, 5, 2]
```

Solução 1:

```
def concatena(pred, l):
    return acumula(lambda x,y : x + y, \
                   filtra(pred, l))
```

Solução 2:

```
def concatena(pred, l):
    return reduce(lambda x,y : x + y, \
                  filter(pred, l))
```

TÉCNICO LISBOA

Fundamentos de Programação - 2018/2019 Aula Prática 07 (30 minutos) Turno 2ª feira 10:30-12:00
Nome:
Número:
Data:
Curso:

Capítulo 6.2 - Funções de ordem superior

Utilizando funcionais sobre listas, escreva a função `todos_pares_lista`, que recebe um predicado unário e uma lista de inteiros positivos, e devolve verdadeiro caso todos números pares da lista satisfaçam o predicado e falso em caso contrário. Pode assumir que a lista dada tem pelo menos um elemento. A sua função deve conter apenas uma instrução, a instrução `return`. Por exemplo:

```
>>> todos_pares_lista(lambda x: x > 5, [3, 4, 5, 6])
False
>>> todos_pares_lista(lambda x: x >= 4, [3, 4, 5, 6])
True
```

Solução 1:

```
def todos_pares_lista(pred, lst):
    return acumula(lambda x,y : x and y,\
                   transforma(pred, \
                     filtra(lambda x : x % 2 == 0, lst))
```

Solução 2:

```
def todos_pares_lista(pred, lst):
    return reduce(lambda x,y : x and y,\
                   map(pred, filter(lambda x : x % 2 == 0, lst))
```