TÉCNICO LISBOA

Fundamentos de Programação - 2019/2020 Aula Prática 08 (30 minutos) Turno 5ª feira 10:30-12:00
Nome:
Número:
Data:
Curso:

Capítulo 7 - Recursão e Iteração

Um número primo é um número inteiro maior do que 1 que apenas é divisível por 1 e por si próprio. Um método simples, mas pouco eficiente, para determinar se um número, `n`, é primo consiste em testar se `n` é múltiplo de algum número entre `2` e $\sqrt{n}$. Usando este processo, escreva uma função recursiva de cauda chamada `primo`, que recebe um número inteiro e tem o valor `True` apenas se o seu argumento for primo. Por exemplo:

```
>>> primo(3)
True
>>> primo(8)
False
```

Solução:

```
def primo(n):
    from math import sqrt
    def primo_aux(i):
        if i < 2:
            return True
        elif n % i == 0:
            return False
        else:
            return primo_aux(i - 1)
    return primo_aux(int(sqrt(n)))
```

TÉCNICO LISBOA

Fundamentos de Programação - 2019/2020 Aula Prática 08 (30 minutos) Turno 4ª feira 08:00-09:30
Nome:
Número:
Data:
Curso:

Capítulo 7 - Recursão e Iteração

Escreva a função recursiva de cauda chamada `cria_tuplo_multiplos` que recebe dois números inteiros positivos `n` e `m`, e devolve um tuplo com os `m` primeiros múltiplos desse número `n`. Considere que 0 é múltiplo de todos os números. Por exemplo:

```
>>> cria_tuplo_multiplos(6, 10)
(0, 6, 12, 18, 24, 30, 36, 42, 48, 54)
```

Solução 1:

```
def cria_tuplo_multiplos(n, m):
    def cria_aux(tup,max) :
        if len(tup) == max:
            return tup
        return cria_aux(tup + (tup[-1] + n,), max)
    return cria_aux((0,), m)
```

Solução 2:

```
def cria_tuplo_multiplos(n, m):
    def cria_aux(tup, mul) :
        if mul == m:
            return tup
        return cria_aux(tup + (n*mul,), mul+1)
    return cria_aux((), 0)
```

TÉCNICO LISBOA

Fundamentos de Programação - 2019/2020 Aula Prática 08 (30 minutos) Turno 6ª feira 15:30-17:00
Nome:
Número:
Data:
Curso:

Capítulo 7 - Recursão e Iteração

Usando recursão de cauda, escreva a função `num_divisores` que recebe um número inteiro positivo *n*, e devolve o número de divisores de *n*. No caso de *n* ser 0 deverá devolver 0. Por exemplo:

```
>>> num_divisores(20)
6
>>> num_divisores(13)
2
```

Solução:

```
def num_divisores(n):
    def num_div_aux(i, res):
        if i == 0:
            return res
        elif n % i == 0:
            return num_div_aux(i - 1, res + 1)
        else:
            return num_div_aux(i - 1, res)
    return num_div_aux(n, 0)
```

TÉCNICO LISBOA

Fundamentos de Programação - 2019/2020 Aula Prática 08 (30 minutos) Turno 6ª feira 14:00-15:30
Nome:
Número:
Data:
Curso:

Capítulo 7 - Recursão e Iteração

Usando recursão de cauda, escreva a função `prod_divisores` que recebe um número inteiro positivo *n*, e devolve o produto de todos os divisores de *n*. No caso de *n* ser 0 deverá devolver 1. Por exemplo:

```
>>> prod_divisores(6)
36
>>> prod_divisores(8)
64
```

Solução:

```
def prod_divisores(n):
    def prod_div_aux(d, res):
        if d == 0:
            return res
        elif n % d == 0:
            return prod_div_aux(d - 1, res * d)
        else:
            return prod_div_aux(d - 1, res)
    return prod_div_aux(n, 1)
```

TÉCNICO LISBOA

Fundamentos de Programação - 2019/2020 Aula Prática 08 (30 minutos) Turno 6ª feira 12:30-14:00
Nome:
Número:
Data:
Curso:

Capítulo 7 - Recursão e Iteração

Usando recursão de cauda, escreva a função `substitui_occ_lista` que recebe uma lista e dois valores, `a` e `b`, e devolve uma nova lista, obtida a partir do original substituindo todas as ocorrências de `a` por `b`. Por exemplo:

```
>>> substitui_occ_lista([(2, 3), 'a', 3, True, 5], 'a', 2)
[(2, 3), 2, 3, True, 5]
>>> substitui_occ_lista([(2, 3), 'a', 3, True, 5], False, 4)
[(2, 3), 'a', 3, True, 5]
>>> substitui_occ_lista([], False, 4)
[]
```

Solução:

```
def substitui_occ_lista(t, de, para):
    def troca_aux(t, res):
        if not t:
            return res
        elif t[0] == de:
            return troca_aux(t[1:], res + [para])
        else:
            return troca_aux(t[1:], res + [t[0]])
    return troca_aux(t, [])
```

TÉCNICO LISBOA

Fundamentos de Programação - 2019/2020 Aula Prática 08 (30 minutos) Turno 2ª feira 09:00-10:30
Nome:
Número:
Data:
Curso:

Capítulo 7 - Recursão e Iteração

Usando recursão de cauda, escreva a função `soma_impares_tuplo` que recebe uma lista de inteiros e devolve a soma de todos os elementos ímpares do tuplo. Por exemplo:

```
>>> soma_impares_tuplo((1,2,3,4,5,6,7))
16
>>> soma_impares_tuplo(())
0
```

Solução:

```
def soma_impares_tuplo(lst):
    def soma_aux(lst, res):
        if not lst:
            return res
        elif lst[0] % 2 != 0:
            return soma_aux(lst[1:], res + lst[0])
        else:
            return soma_aux(lst[1:], res)
    return soma_aux(lst, 0)
```

TÉCNICO LISBOA

Fundamentos de Programação - 2019/2020 Aula Prática 08 (30 minutos) Turno 2ª feira 10:30-12:00
Nome:
Número:
Data:
Curso:

Capítulo 7 - Recursão e Iteração

Usando recursão de cauda, escreva a função `conta_pares_lista` que recebe uma lista de inteiros e devolve o número de elementos pares na lista. Por exemplo:

```
>>> conta_pares_lista([4, 5, 6])
2
>>> conta_pares_lista([3, 5, 7])
0
>>> conta_pares_lista([3])
0
```

Solução:

```
def conta_pares_lista(t):
    def conta_pares_aux(t, res):
        if not t:
            return res
        elif t[0] % 2 == 0:
            return conta_pares_aux(t[1:], res + 1)
        else:
            return conta_pares_aux(t[1:], res)
    return conta_pares_aux(t, 0)
```