TÉCNICO LISBOA

Fundamentos de Programação - 2019/2020 Aula Prática 03 (30 minutos) Turno 2ª feira 10:30-12:00
Nome:
Número:
Data:
Curso:

Capítulo 3 - Funções

Um número inteiro, `n`, diz-se *triangular* se existir um inteiro `m` tal que `n = 1 + 2 + ... + (m - 1) + m`. Escreva uma função chamada `triangular` que recebe um número inteiro positivo `n`, e cujo valor é `True` apenas se o número for triangular. No caso de `n` ser `0` deverá devolver `False`. A função deve verificar a validade dos seus argumentos. Por exemplo,

```
>>> triangular(6)
True
>>> triangular(8)
False
>>> triangular(-4)
Traceback (most recent call last): <...>
ValueError: triangular: argumento invalido
```

Solução:

```
def triangular(n):
    if not (isinstance(n, int) and n > 0):
        raise ValueError('triangular: argumento invalido')
    elif n == 0:
        return False
    else:
        soma = 0
        i = 1
        while soma < n:
            soma = soma + i
            i = i + 1
        return soma == n
```

TÉCNICO LISBOA

Fundamentos de Programação - 2019/2020 Aula Prática 03 (30 minutos) Turno 6ª feira 14:00-15:30
Nome:
Número:
Data:
Curso:

Capítulo 3 - Funções

A função *arctg* pode ser calculada através da seguinte fórmula

$$arctg(x) = \sum_{i=1}^{n} \frac{(-1)^{i-1}\, x^{2i-1}}{2i-1}$$

Escreva uma função com o nome `arctg`, que recebe o número real `x` $\in$ [-1,1] para o qual se quer calcular o *arctg*, bem como o número de termos `n` da expressão a calcular, e devolve o *arctg* calculado de acordo com a fórmula anterior. A função deve verificar a validade dos seus argumentos.

```
>>> arctg(0.5, 100)
0.46364760900080615
>>> arctg(1.0, 100)
0.7828982258896382
>>> arctg(0.5, -2)
Traceback (most recent call last): <...>
ValueError: arctg: argumentos invalidos
```

Solução:

```
def arctg(x,n):
    if not (isinstance(x, float) and (-1 <= x <= 1.0) and \
            isinstance(n, int) and n > 0):
        raise ValueError('arctg: argumentos invalidos')
    soma = 0
    while n > 0:
        termo = (-1)**(n-1) * x**(2*n - 1) / (2*n - 1)
        soma = soma + termo
        n = n - 1
    return soma
```

TÉCNICO LISBOA

Fundamentos de Programação - 2019/2020 Aula Prática 03 (30 minutos) Turno 6ª feira 12:30-14:00
Nome:
Número:
Data:
Curso:

Capítulo 3 - Funções

A função *log(x) para x ∈]0,2]* pode ser calculada através da seguinte fórmula

$$log(x) = \sum_{i=1}^{n} \frac{(-1)^{i+1}(x-1)^{i}}{i}$$

Escreva uma função com o nome `log`, que recebe o número `x` para o qual se quer calcular o *log(x)*, bem como o número de termos `n` da expressão a calcular, e devolve o *log(x)* calculado de acordo com a fórmula anterior. A função deve verificar a validade dos seus argumentos.

```
>>> log(1.5, 50)
0.4054651081081644
>>> log(0.1, 50)
-2.3017962525010716
>>> log (-2, 50)
Traceback (most recent call last): <...>
ValueError: log: argumentos invalidos
```

Solução:

```
def log(x,n):
    if not (isinstance(x, float) and (0 < x <= 2) and \
            isinstance(n, int) and n > 0):
        raise ValueError('log: argumentos invalidos')
    soma = 0
    while n > 0:
        termo = (-1)**(n + 1) * (x - 1)**n / n
        soma = soma + termo
        n = n - 1
    return soma
```

TÉCNICO LISBOA

Fundamentos de Programação - 2019/2020 Aula Prática 03 (30 minutos) Turno 4ª feira 08:00-09:30
Nome:
Número:
Data:
Curso:

Capítulo 3 - Funções

Um número *primo* é um número inteiro maior do que 1 que apenas é divisível por 1 e por si próprio. Por exemplo, 5 é primo porque apenas é divisível por si próprio e por 1, ao passo que 6 não é primo pois é divisível por 1, 2, 3, e 6. Os números primos têm um papel muito importante tanto em Matemática como em Informática. 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 em Python chamada `primo` que recebe um número inteiro positivo e tem o valor `True` apenas se o seu argumento for primo. A função deve verificar a validade dos seus argumentos.

```
>>> primo(9)
False
>>> primo(11)
True
>>> primo(-4)
Traceback (most recent call last): <...>
ValueError: primo: argumento invalido
```

Solução:

```
import math
def primo(n):
    if not (isinstance(n, int) and n > 0):
        raise ValueError('primo: argumento invalido')
    i = 2
    max = math.sqrt(n)
    while i <= max:
        if n % i == 0:
            return False
        i = i + 1
    return True
```

TÉCNICO LISBOA

Fundamentos de Programação - 2019/2020 Aula Prática 03 (30 minutos) Turno 2ª feira 9:00-10:30
Nome:
Número:
Data:
Curso:

Capítulo 3 - Funções

Um número *d* é divisor de *n* se o resto da divisão de *n* por *d* for 0. Escreva uma função com o nome `num_divisores_impares` que recebe um número inteiro positivo `n`, e tem como valor o número de divisores de `n` ímpares. No caso de `n` ser 0 deverá devolver 0. A função deve verificar a validade dos seus argumentos. Por exemplo,

```
>>> num_divisores_impares(20)
2
>>> num_divisores_impares(13)
2
>>> num_divisores_impares(-4)
Traceback (most recent call last): <...>
ValueError: num_divisores_impares: argumento invalido
```

Solução 1:

```
def num_divisores_impares(n):
    if not (isinstance(n, int) and n >= 0):
        raise ValueError('num_divisores_impares: argumento
invalido')
    res = 0
    i = 1
    while i <= n:
        if n % i == 0 and i%2 != 0:
            res = res + 1
        i = i + 1
    return res
```

Solução 2:

```
def num_divisores_impares(n):
    if not (isinstance(n, int) and n >= 0):
        raise ValueError('num_divisores_impares: argumento
invalido')
    res = 0
    i = 1
    while i <= n:
        if n % i == 0:
            res = res + 1
        i = i + 2
    return res
```

TÉCNICO LISBOA

Fundamentos de Programação - 2019/2020 Aula Prática 03 (30 minutos) Turno 6ª feira 15:30-17:00
Nome:
Número:
Data:
Curso:

Capítulo 3 - Funções

Escreva uma função em Python que calcula o valor aproximado da série para um determinado valor de $x \in [0,inf]$:

$$\sum_{n=0}^{\infty} \frac{x^n}{n!} = e^x$$

O cálculo do valor aproximado da série deverá terminar quando o termo a adicionar for inferior a um certo `delta` (que é fornecido como segundo parâmetro da função). A função deve verificar a validade dos seus argumentos. Assuma a existência da função `factorial(n)`.

```
>>> serie_e(2.0, 0.0001)
7.388994708994708
>>> serie_e(1.0, 0.0001)
2.7182539682539684
>>> serie_e(2.5, "0.0001")
Traceback (most recent call last): <...>
ValueError: serie_e: argumentos invalidos
```

Solução:

```
def serie_e(x, delta):
    if not (isinstance(x, float) and x >= 0 and \
        isinstance(delta, float)):
        raise ValueError('serie_e: argumentos invalidos')
    soma = 0
    n = 0
    termo = 1
    while termo >= delta:
        soma = soma + termo
        n = n + 1
        termo = x**n / factorial(n)
    return soma
```

TÉCNICO LISBOA

Fundamentos de Programação - 2019/2020 Aula Prática 03 (30 minutos) Turno 5ª feira 10:30-12:00
Nome:
Número:
Data:
Curso:

Capítulo 3 - Funções

Escreva uma função em Python com o nome `prod_cubos` que recebe um número inteiro positivo, `n`, e tem como valor a soma dos cubos de todos os números inteiros de 1 até `n`. A função deve verificar a validade dos seus argumentos.

```
>>> prod_cubos(1)
1
>>> prod_cubos(3)
216
>>> prod_cubos(-4)
Traceback (most recent call last): <...>
ValueError: prod_cubos: argumento invalido
```

Solução:

```
def prod_cubos(n):
    if not (isinstance(n, int) and n > 0):
        raise ValueError('prod_cubos: argumento invalido')
    prod = 1
    while n != 0:
        prod = prod * (n**3)
        n = n - 1
    return prod
```