Lista_02_Solucoes

September 8, 2020

1 Lista de Exercícios 2 (soluções propostas)

*Alguns exercícios foram retirados do livro: The Python Workbook - Ben Stephenson.

Não é um gabarito fechado. As soluções encontradas aqui são apenas algumas possibilidades. Usem a criatividade!

1.1 QUESTÃO 1:

Crie uma função que recebe como parâmetro de entrada dois números inteiros positivos e encontre o seu **Máximo Divisor Comum** (MDC).

Essa função pode ser desenvolvida checando os divisores de cada número envolvido e verificando qual é o maior deles que divide ambos os números.

Um forma de otimizar o processo é começar a testar a dvisão pelo MENOR dos números envolvidos, por isso o teste para saber qual é o menor dos números passados como parâmetro de entrada.

[1]:
```
def mdc(n1,n2):
    if n1<n2:
        menor=n1
    else:
        menor=n2

    #menor=min(n1,n2) # Já existe no Python uma função que calcula o valor␣
↪mínimo!

    while n1%menor!=0 or n2%menor!=0:
        menor=menor-1

    return menor
```

[2]:
```
mdc(16,24)
```

[2]: 8

A quesão termina aqui, mas podemos resgatar a função feita em aula que lista todos os divisores de um número para confirmar e entender melhor esse resultado.

[3]:
```
def listaDivisores(x):
    divisores=[]
    for i in range(1,x+1):
        if x%i==0:
            divisores.append(i)
    return divisores
```

[4]:
```
n1=16
n2=24
print(listaDivisores(n1))
print(listaDivisores(n2))
print('O MDC entre {} e {} é {}!'.format(n1,n2,mdc(n1,n2)))
```

```
[1, 2, 4, 8, 16]
[1, 2, 3, 4, 6, 8, 12, 24]
O MDC entre 16 e 24 é 8!
```

1.2 QUESTÃO 2:

Agora, crie uma função que recebe como parâmetro de entrada dois números inteiros positivos e encontre o seu **Mínimo Múltiplo Comum** (MMC).

Lembrando que: se **um número é divisível por outro**, diferente de zero, então dizemos que ele é **múltiplo** desse outro.

Nesse caso, uma boa forma de otimizar o processo é começar a testar a dvisão pelo MAIOR dos números passados como parâmetro de entrada.

[5]:
```
def mmc(n1,n2):
    if n1<n2:
        maior=n2
    else:
        manior=n1

    #maior=max(n1,n2) # Já existe no Python uma função que calcula o valor␣
↪máximo!

    achou=False
    while achou==False:
        if maior%n1==0 and maior%n2==0:
            achou=True
        else:
            maior=maior+1

    return maior
```

[6]:
```
mmc(16,24)
```

[6]: 48

A questão termina aqui, mas e se quiséssemos criar uma função que recebe uma lista de números? A vantagem é que dessa forma o MMC poderia ser calculado para além de 2 números.

Em

```
...
if all(maior%nr==0 for nr in nums):
    ...
```

é o mesmo que,

```
...
if maior%nums[0]==0 and maior%nums[1]==1 and ... maior%nums[len(nums)-1]==1:
    ...
```

[7]:
```
def mmc2(nums):
    maior=max(nums)

    achou=False
    while achou==False:
        if all(maior%nr==0 for nr in nums):
            achou=True
        else:
            maior=maior+1

    return maior
```

[8]:
```
mmc2([5,6,8])
```

[8]: 120

1.3 QUESTÃO 3:

No iníco do curso, vimos como fazer conversão de números binários (base 2) para decimais (base 10). Portanto, crie uma função que recebe como parâmetro de entrada um número binário e calcula o número decimal correspondente.

[9]:
```
def binDec(numBin):
    decimal=0

    q=numBin
    i=0
    while q!=0:
        x=q%10    # Conseguimos ter acesso ao último dígito do número!
        decimal=decimal+(x*2**i)
        q=q//10   # Conseguimos ter acesso ao restante do número sem o últmo␣
↪dígito!
        i=i+1
```

```
    return decimal
```

[10]:
```
binDec(1001)
```

[10]: 9

Uma outra forma seria trabalhar com o parâmetro de entrada como sendo do tipo string. Consegue perceber em quais linhas o códgigo anterior sofreu alterações?

[11]:
```
def binDec2(numBin):
    decimal=0

    q=str(numBin)
    i=0
    while q!='':
        x=q[-1]    # Conseguimos ter acesso ao último dígito do número!
        decimal=decimal+(int(x)*2**i)
        q=q[:-1]   # Conseguimos ter acesso ao restante do número sem o últmo␣
↪dígito!
        i=i+1
    return decimal
```

[12]:
```
binDec2(1001)
```

[12]: 9

1.4 QUESTÃO 4:

Agora, escreva uma função que faça o inverso, ou seja, que faça a conversão de um número decimal (base 10) em número binário (base 2). Note que o parâmetro de entrada é um número inteiro e o de saída, uma string.

[13]:
```
def decBin(numDec):

    base=2
    binario='' # Definimos a variável binario como uma string vazia!

    q=numDec
    r=q%base
    binario=str(r)+binario
    q=q//base

    while q>0:
        r=q%base
        binario=str(r)+binario
        q=q//base
```

```
    return binario
```

[14]:
```
decBin(9)
```

[14]:
```
'1001'
```

1.5 QUESTÃO 5:

Crie uma função que começa selecionando um número inteiro aleatório entre 1 e 100. Armazene esse valor inicial como sendo o maior número inteiro encontrado até o momento. Em seguida, gere aleatoriamente mais 99 números inteiros, verificando sempre se o novo valor é maior do que o máximo encontrado anteriormente. Caso seja maior, a função contabiliza que ocorreu uma atualização do valor máximo. Ao final, sua função deverá retornar o valor máximo encontrado e a quantidade de veses que o valor máximo foi atualizado durante o processo.

[15]:
```
import random as rd
```

[16]:
```
def maiorInteiro(num):

    maximo=rd.randint(1,num)
    cont=0

    for i in range(num-1):
        novo_valor=rd.randint(1,num)
        if novo_valor>maximo:
            maximo=novo_valor
            cont=cont+1

    return maximo,cont
```

[17]:
```
maiorInteiro(100)
```

[17]:
```
(97, 4)
```

1.6 QUESTÃO 6:

Crie uma função que encontre o número mínimo de vezes que você tem que jogar uma moeda antes de poder ter três lançamentos consecutivos que resultam no mesmo resultado (ou todos os três são CARA (C) ou todos os três são COROA (K))?

Tente fazer uma simulação para saber quantos lançamentos são necessários fazer em média para obter tal feito? Para isso, você pode executar a função criada umas 10 vezes e calcular a média dos resultados.

[18]:
```
def lancamento():
    moeda=rd.randint(1,2)
    if moeda==1:
        return 'C'
```

```
    else:
        return 'K'
```

[19]:
```
def moedas():

    #ops = ['C', 'K'] # Aqui trabalha com a ideia de listas!
    resultado = ''

    while True:
        #moeda=rd.choice(ops)
        moeda=lancamento()
        resultado=resultado+moeda

        if len(resultado) >= 3:
            if ( resultado[-3] == resultado[-2] == resultado[-1] ):
                break
            else:
                continue
    return resultado, len(resultado)
```

[20]:
```
moedas()
```

[20]:
```
('KKCCC', 5)
```

[21]:
```
def simulaMoedas(rep):
    resultados=[]
    soma=0

    for i in range(rep):
        resultados.append(moedas())
        soma=soma+resultados[i][1]

    print(*resultados, sep="\n") # Para o print sair com cada elemento da lista␣
 ↪em linhas diferentes!

    media = soma/rep

    return media
```

[22]:
```
simulaMoedas(10)
```

```
('KKK', 3)
('KCKCCC', 6)
('CKCCC', 5)
('KKCKCKKCCC', 10)
('KKCKCKKK', 8)
('CKCCKKK', 7)
```

```
('KCKCCC', 6)
('CCC', 3)
('CCKKK', 5)
('KKK', 3)
```

[22]:
```
5.6
```

Podemos explorar ainda mais essa simulação! Vamos deixar isso para as próxima aulas.

1.7 QUESTÃO 7:

Escreva uma função que determina se uma senha é válida ou não. Uma boa senha é defnida segundo os seguintes critérios:

- possui pelo menos 8 caracteres
- contém pelo menos uma letra maiúscula
- contém pelo menos uma letra minúscula
- contém pelo menos um número

Sua função deve retornar verdadeiro (True) se a senha informada for válida ou falso (False), caso contrário.

[23]:
```
def validaSenha(senha):

    tem_maiuscula=False
    tem_minuscula=False
    tem_num=False

    alfabeto='abcdefghijklmnopqrstuvwxyz'
    numeros='0123456789'

    for letra in senha:
        if letra in alfabeto:
            tem_minuscula=True
        elif letra in alfabeto.upper():
            tem_maiuscula=True
        elif letra in numeros:
            tem_num=True

    if len(senha)>=8 and tem_minuscula and tem_maiuscula and tem_num:
        return True
    else:
        return False
```

[24]:
```
validaSenha('anacarolina29')
```

[24]:
```
False
```

[25]:
```
validaSenha('anaCarolina29')
```

[25]:
```
True
```

1.8 QUESTÃO 8:

Uma **data mágica** é uma data em que o dia multiplicado pelo mês é igual ao ano de dois dígitos. Por exemplo, 10 de junho de 1960 é uma data mágica porque junho é o sexto mês e 6 vezes 10 é 60, que é igual ao ano de dois dígitos.

Escreva uma função que determina se uma data é ou não uma data mágica.

[26]:
```
def dataMagica(data):

    if len(data)!=10:
        return 'Digite uma data válida! Formato da data: dd/mm/aaaa.'

    dia=int(data[:2])
    mes=int(data[3:5])
    decada=int(data[8:])

    if dia*mes==decada:
        return True
    else:
        return False
```

[27]:
```
dataMagica('10/06/1960')
```

[27]:
```
True
```

1.9 QUESTÃO 9:

Dizemos que dois números são amigos se cada um deles é igual a soma dos **divisores próprios** do outro. Os divisores próprios de um número positivo n são todos os divisores inteiros positivos de n exceto o próprio n.

Crie uma função que verifica se dois números são amigos (retorna True) ou não (retorna False).

[28]:
```
def divisoresProprios(n):

    soma=0
    for i in range(1,n-1): # Vai até n-1, pois n não é um divisor próprio!
        if n%i==0:
            soma=soma+i
    return soma
```

[29]:
```
divisoresProprios(284)
```

[29]: 220

[30]:
```
divisoresProprios(220)
```

[30]: 284

[31]:
```
def numerosAmigos(n1,n2):
    if divisoresProprios(n1)==n2 and n1==divisoresProprios(n2):
        return True
    else:
        return False
```

[32]:
```
numerosAmigos(284,220)
```

[32]: True

1.10 QUESTÃO 10:

O CPF é formado por 11 dígitos numéricos que seguem a máscara "###.###.###-##", a **verificação do CPF** acontece utilizando os 9 primeiros dígitos e, com um cálculo simples, verificando se o resultado corresponde aos dois últimos dígitos (depois do sinal "-").

VALIDAÇÃO DO PRIMEIRO DÍGITO

- multiplicar os 9 primeiros dígitos pela sequência decrescente de números de 10 à 2 e somar os resultados;
- multiplicar esse resultado por 10 e dividir por 11;
- se o RESTO for igual ao primeiro dígito verificador (primeiro dígito depois do '-'), a primeira parte da validação está correta (se o resto da divisão for igual a 10, nós o consideramos como 0).

VALIDAÇÃO DO SEGUNDO DÍGITO

- multiplicar os 10 primeiros dígitos pela sequência decrescente de números de 11 à 2 e somar os resultados;
- multiplicar esse resultado por 10 e dividir por 11;
- se o RESTO for igual ao segundo dígito verificador (segundo dígito depois do '-'), a validação está correta!

Sua tarefa é criar uma função que verifica se um CPF é válido. O parâmetro de entrada dessa função é uma string da seguinte forma: "###.###.###-##" e esse formato deve ser verificado antes de efetuar os cálculos da validação.

Atente-se ao fato de que alguns CPFs passam nessa validação, mas ainda são inválidos. É o caso dos CPFs com dígitos repetidos ("111.111.111-11", "222.222.222-22", ...). Adicione essa verficação!

[33]:
```
def verificaCPF(cpf):

```

```
    if cpf[3]=='.' and cpf[7]=='.' and cpf[11]=='-' and len(cpf)==(11+3):

        num1=cpf[:3]+cpf[4:7]+cpf[8:11]
        d1=cpf[12]
        d2=cpf[13]
        num2=num1+d1

        if num2+d2==11*num2[0]:
            return False

        else:
            soma1=0
            i=10
            for n in num1:
                soma1=soma1+i*int(n)
                i=i-1
            resto1=soma1*10%11
            if resto1==10: resto1=0

            soma2=0
            j=11
            for n in num2:
                soma2=soma2+j*int(n)
                j=j-1
            resto2=soma2*10%11
            if resto2==10: resto2=0

            if resto1==int(d1) and resto2==int(d2):
                return True
            else:
                return False

    else:
        return 'ERRO! Você deve digitar um CPF no seguinte formato "###.###.
↪###-##"'
```

[34]:
```
verificaCPF('529.982.247-25')
```

[34]: True

[35]:
```
verificaCPF('111.111.111-11')
```

[35]: False

[36]:
```
verificaCPF('52998224725')
```

[36]: 'ERRO! Você deve digitar um CPF no seguinte formato "###.###.###-##"'