MC-102 — Aula 07
Comandos Repetitivos

Instituto de Computação – Unicamp

27 de Agosto de 2019

Introdução

- Vimos quais são os comandos de repetição em Python.
- Veremos mais alguns exemplos de sua utilização.

Exercício 1

- Faça um programa que lê um número *n* e imprima os valores entre 2 e n, que são divisores de n.

Exercício 1

```
n=int(input('Entre com o numero: '))
for i in range(2,n+1):
    if n % i == 0 :
        print(i)
```

o central aqui é usar o **range** para gerar os numeros que vao tentar dividir o **n**

Exercício 2

- Faça uma função que recebe 2 listas. Se elas nao tiverem o mesmo tamanho, a função deve retornar 0 senão ela retorna o produto escalar das duas listas

Exercício 2

- Há duas formas de percorrer uma lista. O primeiro é gerar os elementos da lista usando o **for**

```
for x in lista:
    faca algo com o x
```

- o segundo é gerar os indices dos elementos da lista usando o **for** e **range+len**

```
for i in range(len(lista)):
    faca algo com o lista[i]
```

Exercício 2

- De uma forma geral, gerar os indices é mais poderoso e em algumas linguagens de programação isso é tudo que é possivel.
- mas percorrer os elementos da lista é mais facil de entender e normalmente causa menos erros de programação. Utilize esse sempre que possivel, mas nem sempre é possivel!!
- utilize os indices quanto voce tem mais de uma lista e voce tem o conceito de "elementos correspondentes nas duas listas". Ou seja, o elemento na posição 3 da primeira lista **corresponde** ao elemento 3 da segunda lista.
- veremos num exercicio abaixo outra razao para usar os indices

Exercício 2

```
def produto_escalar(l1,l2):
  if len(l1) != len(l2):
      return 0
  s=0.0
  for i in range(len(l1)):
      s = s + l1[i]*l2[i]
  return s
```

Assim que voce descobre que a resposta deve ser 0, de um **return** desta resposta. O **return** termina a função e os comandos abaixo do **if** (neste caso **s=0.0**) não são executados.

Exercício 3

- Faça um programa que le 2 listas, uma de nomes (leitura até o usuário entrar com uma lista vazia) e a segunda lista de notas (o menso numero de notas que o numero de pessoas!), e imprima o nome das pessoas que tiveram nota < 5. **
- De novo, o conceito de elementos correspondentes das duas listas, nome e nota. Devemso usar o **for** do **range+len**
- vams usar o **leian** e o **leiafim** da aula passada

Exercício 3

```
def sleiafim():
  l = []
  while True:
     x = input('Nome: ')
     if x == '':
        return l
     l.append(x)

def nleian(n):
   l = []
   for i in range(n):
     d = float(input('Nota: '))
     l.append(d)
   return l

nomes = sleiafim()
n = len(nomes)
notas = nleian(n)

for i in range(n):
   if notas[i]<0.5:
      print(nome[i])
```

Exercício 4

- Faça um programa que le 2 listas, uma de nomes (leitura até o usuário entrar com uma lista vazia) e a segunda lista de notas (o menso numero de notas que o numero de pessoas!), e imprima o nome da pessoa com maior nota. **
- basicamente o mesmo problema que o anterior, mas buscando a posicao da maior nota e não so percorrendo as notas e buscando as menores que 0.5

Exercício 4

```
def sleiafim():
  l = []
  while True:
     x = input('Nome: ')
     if x == '':
        return l
     l.append(x)

def nleian(n):
   l = []
   for i in range(n):
     d = float(input('Nota: '))
     l.append(d)
   return l
```

Exercício 4

```
nomes = sleiafim()
n = len(nomes)
notas = nleian(n)

ma = 0.0
for i in range(n):
   if notas[i] > ma:
      ma = notas[i]
      pos = i

print('o aluno ',nomes[pos],'tirou a maior nota:',notas[pos])
```

Exercício 5

- Faça uma função que recebe uma lista e retorna **True** se a lista esta em ordem crescente
- este problema ilustra a 2a razão para usar os indices (**for** do **range+len**) para percorrer uma lista. Voce não pode só usar o elemento da lista, voce precisa usar seus "vizinhos" também.

Exercício 5

VERSAO INCORRETA

```
def ordenada(l):
   for i in range(len(l)):
     if l[i]>l[i+1]:
        return False
   return True
```

- O central é perceber que determinar se a lista **NAO** esta ordenada é facil - é so encontrar um elemento que é maior que o proximo na lista (é isso que o **if l[i]¿l[i+1]** esta fazendo). Assim que voce descobriu que a lista não esta ordenada, voce ja sabe a resposta e de o **return False** para terminar a função (não importa quanto da lista voce ainda tem que examinar!)
- A lista só esta ordenada se ao chegar ao final da lista voce descobriu que não é verdade que ela esta fora de ordem. Assim se o **for** terminar normalmente é porque o **if** nunca deu certo. Assim ao final do **for** voce sabe que a lista esta ordenada.

Exercício 5

Há um problema com o codigo acima, que não tem a ver com a ideia central. O programa funciona para não ordenadas, mas dá erro para ordenadas.

```
>>> ordenada([2,3,7,4,12])
False
>>> ordenada([2,3,7,14,112])
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
  File "<stdin>", line 3, in ordenada
IndexError: list index out of range
```

```
def ordenada(l):
   for i in range(len(l)-1):
     if l[i]>l[i+1]:
        return False
   return True
```

O problema é que para o ultimo elemento da lista, não da para compara-lo com o proximo.

Exercício 6

- Faça uma função que recebe uma lista e retorna True se a lista for palindromica - a lista é igual da esquerda para a direita e o reverso
- as listas [1,2,2,1] e [1,2,4,2,1] são palindromicas
- **'asdffdsa'** é um string (que é mais ou menos uma lista) palindromico
- a lista [1,2,3,4,3,2] não é palindromica

Exercício 6

```
def palindromico(l):
    n = len(l)
    meio = n // 2
    for i in range(meio):
        if not l[i] == l[n-i-1]:
            return False
    return True
```

- de novo o padrão de retorna False assim que algo der errado, e retorna True apenas ao final (nada deu errado)
- descobrir que vc tem que comparar o **l[i]** com o **l[n-i-1]**, e que o meio é **n//2** é através de tentativa e erro - isso é super difícil de acerta da primeira.