Computação I - Python
Aula 7 - Estrutura de Repetição com Teste de Parada: while

Exemplo

Apresentado por: Anamaria Martins Moreira

Produção DCC-UFRJ

Metodologia de referência https://doi.org/10.5753/wei.2016.9683

Problema da filtragem

- Vamos olhar de novo para um exercício que fizemos nas aulas anteriores.
- Vejamos o problema da filtragem de números pares que foi usado para exercitar a estrutura condicional e o uso de tuplas.

Faça uma função chamada **filtra_pares** que receba uma tupla com quatro elementos inteiros como argumento, e retorne uma nova tupla contendo apenas os elementos pares da tupla original, na mesma ordem em que se encontravam. Esse tipo de operação onde se selecionam elementos de um conjunto inicial que satisfazem uma determinada propriedade é bastante comum em computação, e se chama **filtragem**.

Primeira solução - força bruta

- Testar todas as possibilidades de combinação de números pares na tupla de 4 elementos.
- Algo como: se todos são pares, devolve a tupla original, se o primeiro e o segundo e o terceiro são pares e o quarto, não, devolve a tupla contendo os 3 primeiros elementos. etc.
- $1 + 4 + 6 + 4 + 1$ (como vão os conhecimentos de combinatória?)

Primeira solução - força bruta - parte 1

```
def filtra_pares_v0(t):
    ''' funcao que dada uma tupla com 4 inteiros, retorna uma tupla com os
    inteiros pares da tupla original, mantida a ordem.
    tuple --> tuple'''

    if t[0]%2 == 0 and t[1]%2 == 0 and t[2]%2 == 0 and t[3]%2 == 0:
        return t

    elif t[0]%2 == 0 and t[1]%2 == 0 and t[2]%2 == 0:
        return (t[0], t[1], t[2])
    elif t[0]%2 == 0 and t[1]%2 == 0 and t[3]%2 == 0:
        return (t[0], t[1], t[3])
    elif t[0]%2 == 0 and t[2]%2 == 0 and t[3]%2 == 0:
        return (t[0], t[2], t[3])
    elif t[1]%2 == 0 and t[2]%2 == 0 and t[3]%2 == 0:
        return (t[1], t[2], t[3])

    elif t[0]%2 == 0 and t[1]%2 == 0:
        return (t[0], t[1])
    elif t[0]%2 == 0 and t[2]%2 == 0:
        return (t[0], t[2])
    elif t[0]%2 == 0 and  t[3]%2 == 0:
        return (t[0], t[3])
```

Primeira solução - força bruta - parte 2

```
elif t[1]%2 == 0 and t[2]%2 == 0:
    return (t[1], t[2])
elif t[1]%2 == 0 and  t[3]%2 == 0:
    return (t[1], t[3])

elif t[2]%2 == 0 and  t[3]%2 == 0:
    return (t[2], t[3])

elif t[0]%2 == 0:
    return (t[0],)
elif t[1]%2 == 0 :
    return (t[1],)
elif t[2]%2 == 0 :
    return (t[2],)
elif t[3]%2 == 0:
    return (t[3],)

else:
    return ()
```

Primeira solução - força bruta - parte 1

Observe que não testo todos os elementos em todos os elifs. A quantidade de elementos testada vai diminuindo. Por que isso é possível?

```
def filtra_pares_v0(t):
    ''' funcao que dada uma tupla com 4 inteiros, retorna uma tupla com os
    inteiros pares da tupla original, mantida a ordem.
    tuple --> tuple'''

    if t[0]%2 == 0 and t[1]%2 == 0 and t[2]%2 == 0 and t[3]%2 == 0:
        return t

    elif t[0]%2 == 0 and t[1]%2 == 0 and t[2]%2 == 0:
        return (t[0], t[1], t[2])
    elif t[0]%2 == 0 and t[1]%2 == 0 and t[3]%2 == 0:
        return (t[0], t[1], t[3])
    elif t[0]%2 == 0 and t[2]%2 == 0 and t[3]%2 == 0:
        return (t[0], t[2], t[3])
    elif t[1]%2 == 0 and t[2]%2 == 0 and t[3]%2 == 0:
        return (t[1], t[2], t[3])

    elif t[0]%2 == 0 and t[1]%2 == 0:
        return (t[0], t[1])
    elif t[0]%2 == 0 and t[2]%2 == 0:
        return (t[0], t[2])
    elif t[0]%2 == 0 and  t[3]%2 == 0:
        return (t[0], t[3])
```

Segunda solução - mais esperta

Uma solução bem mais esperta vai construindo a tupla resultado aos poucos.

```
def filtra_pares_v1(t):
    ''' funcao que dada uma tupla com 4 inteiros, retorna uma tupla com os
    inteiros pares da tupla original, mantida a ordem.
    tuple --> tuple'''
    pares = ()

    if t[0]%2 == 0:
        pares = pares + (t[0],)
    if t[1]%2 == 0:
        pares = pares + (t[1],)
    if t[2]%2 == 0:
        pares = pares + (t[2],)
    if t[3]%2 == 0:
        pares = pares + (t[3],)

    return pares
```

Segunda solução - mais esperta

O algoritmo dessa segunda solução era algo como:

1. Verifique se o valor do primeiro elemento da tupla é par. Se for, anote esse elemento na resposta.
2. Verifique se o valor do segundo elemento da tupla é par. Se for, anote esse elemento na resposta.
3. Verifique se o valor do terceiro elemento da tupla é par. Se for, anote esse elemento na resposta.
4. Verifique se o valor do quarto elemento da tupla é par. Se for, anote esse elemento na resposta.

Com um pequeno detalhe adicional. No papel, podemos criar o espaço para a resposta na hora que aparece o primeiro número par. No programa, isso também é possível, mas o código ficaria muito maior. Por isso, já criamos esse "espaço" para a resposta antes de começar o processo de verificação dos elementos.

Segunda solução - mais esperta

O algoritmo então fica:

1. Crie um registro (um lugar) para registrar a resposta, inicialmente vazia.
   construa o valor da resposta fazendo as seguintes ações em sequência
2. Verifique se o valor do primeiro elemento da tupla é par. Se for, anote esse elemento na resposta.
3. Verifique se o valor do segundo elemento da tupla é par. Se for, anote esse elemento na resposta.
4. Verifique se o valor do terceiro elemento da tupla é par. Se for, anote esse elemento na resposta.
5. Verifique se o valor do quarto elemento da tupla é par. Se for, anote esse elemento na resposta.

Segunda solução - mais esperta

Usando o conceito de variáveis, o algoritmo então fica:

1. Crie um uma variável para registrar a resposta, inicialmente vazia. *construa o valor da resposta fazendo as seguintes ações em sequência*
2. Verifique se o valor do primeiro elemento da tupla é par. Se for, anote esse elemento na resposta.
3. Verifique se o valor do segundo elemento da tupla é par. Se for, anote esse elemento na resposta.
4. Verifique se o valor do terceiro elemento da tupla é par. Se for, anote esse elemento na resposta.
5. Verifique se o valor do quarto elemento da tupla é par. Se for, anote esse elemento na resposta.

Segunda solução - mais esperta

Esta úlltima solução, além de mais curtinha, tem a grande vantagem de possuir um algoritmo facilmente adaptável para tratar um número qualquer de elementos na tupla de entrada.