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**.

Uma solução

Ir 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
```

Uma solução - algoritmo

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.

Por que 4 elementos?

- Por que definir uma função que só funciona com tuplas de 4 elementos? Por que não 3? Ou 5? Por que não ser mais geral e permitir que a função funcione para qualquer tamanho da tupla?
- Nós ainda não tínhamos visto os recursos que permitiriam implementar essa generalidade, que são as estruturas de repetição. Se analisarmos as soluções que mostrei aqui, elas só servem para um número fixo de 4 elementos.
- Vamos mudar agora o enunciado.

Faça uma função chamada **filtra_pares** que receba uma tupla não vazia de inteiros como argumento, e retorne uma nova tupla contendo apenas os elementos pares da tupla original, na mesma ordem em que se encontravam.

Tratando o problema mais geral

Como poderíamos fazer para alterar a solução anterior para resolver este problema mais geral?
A solução força bruta (enumerar todas as possibilidades) não generaliza facilmente. Já a que apresentamos aqui, possui uma estrutura mais fácil de generalizar. Vamos olhar para ela de novo.

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.

Tratando o problema mais geral - preparando para qualquer número de elementos

Podemos mudar para:

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 próximo elemento da tupla é par. Se for, anote esse elemento na resposta.
4. Verifique se o valor do próximo elemento da tupla é par. Se for, anote esse elemento na resposta.
5. Verifique se o valor do próximo elemento da tupla é par. Se for, anote esse elemento na resposta.

Observaram que as três últimas linhas são idênticas? O que muda é a posição da tupla que "próximo elemento" representa. Vamos mudar mais um pouquinho.

Tratando o problema mais geral - preparando para qualquer número de elementos

Podemos mudar para:

1. Crie um uma variável para registrar a resposta, inicialmente vazia.
2. Crie uma variável para identificar o próximo elemento a ser verificado. Inicie o valor dessa variável de maneira a indicar que o próximo elemento a ser tratado é o primeiro elemento da tupla de entrada.
3. Verifique se o valor do próximo elemento da tupla é par. Se for, anote esse elemento na resposta.
4. Verifique se o valor do próximo elemento da tupla é par. Se for, anote esse elemento na resposta.
5. Verifique se o valor do próximo elemento da tupla é par. Se for, anote esse elemento na resposta.
6. Verifique se o valor do próximo elemento da tupla é par. Se for, anote esse elemento na resposta.

Agora as 4 linhas de verifique e anote são idênticas. Fica mais fácil de imaginar como processar tuplas de tamanhos variados, não?

Tratando o problema mais geral - preparando para qualquer número de elementos

Antes, vamos ver como fica esse algoritmo em Python.

```
def filtra_pares_v2a(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 = ()
    proximo = 0

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

    return pares
```

Que tal? O que vocês acham desse código? Ele faz a mesma coisa que o código original? Pense um pouco antes de continuar.

Tratando o problema mais geral - preparando para qualquer número de elementos

Não...

- A palavra próximo tem um significado embutido para nós humanos: se eu olhei o índice 1 vou olhar o 2, se o último que olhei é o índice 2, o próximo é o 3, e assim por diante.
- Para o programa, precisaremos traduzir esse significado em uma ação de incrementar o valor da variável que indica o próximo valor a ser avaliado.

Tratando o problema mais geral - preparando para qualquer número de elementos

```
def filtra_pares_v2b(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 = ()
    proximo = 0

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

    return pares
```

Tratando o problema mais geral - qualquer número de elementos

Agora estamos bem perto de conseguir generalizar o código para uma quantidade qualquer de elementos na tupla de entrada. Seria algo como:

1. Crie um uma variável para registrar a resposta, inicialmente vazia.
2. Crie uma variável para identificar o próximo elemento a ser verificado. Inicie o valor dessa variável de maneira a indicar que o próximo elemento a ser tratado é o primeiro elemento da tupla de entrada.
3. Repita:
    1. Verifique se o valor do próximo elemento da tupla é par. Se for, anote esse elemento na resposta.
    2. Incremente o indicador do proximo elemento a ser verificado.

    Enquanto tem elemento novo para verificar.
4. Quando isso deixar de acontecer, ou seja, quando não tem elemento novo para verificar, retorne a resposta construída.

Tratando o problema mais geral - qualquer número de elementos

- A maneira de implementar em Python ações de um algoritmo com a estrutura "repita um conjunto de ações enquanto uma determinada condição é satisfeita" é o comando `while`, que estudaremos nesta aula.
- Mas para traduzir esse algoritmo para Python, precisamos ainda pensar como dizer que "tem elemento novo para verificar".
- Que expressão booleana construída a partir dos dados do nosso algoritmo:
  - o parâmetro `t`
  - as variáveis `pares` e `proximo`

  podemos usar para exprimir essa condição?
- Pause o vídeo e pense em como resolver este problema.

Tratando o problema mais geral - qualquer número de elementos

- Ter elementos novos para verificar significa que ainda não verificamos o último, indexado pelo tamanho de t menos 1, ou seja, `proximo` ainda representa um índice válido (menor ou igual a `len(t)-1`).

Tratando o problema mais geral - qualquer número de elementos

- Ter elementos novos para verificar significa que ainda não verificamos o último, indexado pelo tamanho de t menos 1, ou seja, `proximo` ainda representa um índice válido (menor ou igual a `len(t)-1`).
- Como sempre estamos fazendo verifica-anota-incrementa `proximo`, quando `proximo` chegar ao valor `len(t)` significa que acabaram os elementos a serem verificados.

Tratando o problema mais geral - qualquer número de elementos

- Ter elementos novos para verificar significa que ainda não verificamos o último, indexado pelo tamanho de t menos 1, ou seja, `proximo` ainda representa um índice válido (menor ou igual a `len(t)-1`).
- Como sempre estamos fazendo verifica-anota-incrementa `proximo`, quando `proximo` chegar ao valor `len(t)` significa que acabaram os elementos a serem verificados.
- A condição a ser avaliada e que diz que precisamos continuar o ciclo verifica-anota-incrementa `proximo` é então

  `proximo < = len(t) -1`,

  ou, mais simplesmente,

  `proximo < len(t)`.

Tratando o problema mais geral - qualquer número de elementos

Em Python, nosso algoritmo fica então assim:

```
def filtra_pares_v3(t):
    ''' funcao que dada uma tupla não vazia de inteiros, retorna uma tupla com os
    inteiros pares da tupla original, mantida a ordem.
    tuple --> tuple'''
    pares = ()
    proximo = 0
    while proximo <len(t):
        if t[proximo]%2 == 0:
            pares = pares + (t[proximo],)
        proximo = proximo + 1
    return pares
```

E é como desenvolver funções usando essa estrutura (comando while) que vamos estudar nesta aula.

Bom Estudo!