Iteração

MC102-2018s1-Aula06-180315-takeaway

Arthur J. Catto, PhD
arthur.catto@g.unicamp.br

15 de março de 2018

1 Iteração

Os recursos de programação que vimos até agora não nos habilitam a resolver qualquer classe de problemas.

Imagine, por exemplo, um programa que deva ler um inteiro positivo e depois imprimir essa quantidade de Xs.

Para resolver esse problema, poderíamos pensar em algo como...

```
numXs = int(input("Quantos Xs? "))
if numXs == 1:
   print("X")
elif numXs == 2:
   print("XX")
elif numXs == 3:
   print("XXX")
...
```

Isso funciona?

Quantos testes serão necessários?

O que fazer para adivinhar a resposta do usuário?

E se nossa solução fosse esta?

```
numXs = int(input("Quantos Xs? "))
ainda_faltam = numXs
if ainda_faltam > 0:
   print("X", end='')
   ainda_faltam -= 1
if ainda_faltam > 0:
   print("X", end='')
   ainda_faltam -= 1
if ...
```

E se fosse esta outra?

```
numXs = int(input("Quantos Xs? "))
ja_fiz = 0
if ja_fiz < numXs:
   print("X")
   ja_fiz += 1
elif ja_fiz < numXs:
```

```
    print("X")
    ja_fiz += 1
elif ...
```

A diferença, nesses dois últimos casos, é que um mesmo conjunto de comandos foi usado repetidas vezes.

Nesses dois casos, `ainda_faltam` e `ja_fiz` são chamadas *variáveis contadoras* — variáveis auxiliares cujos valores vão sendo incrementados ou decrementados passo a passo até que satisfaçam uma dada condição.

Uma vez satisfeita essa condição, nenhum outro `if` consegue ser executado.

O problema, no entanto, permanece.

Quantos `if`s são necessários?

A execução repetida de um conjunto de comandos é chamada *iteração* e é implementada por comandos específicos em todas as linguagens de alto nível.

Python, em particular, oferece duas estruturas, que serão examinadas em seguida.

1.1 O comando `while`

Um comando `while` executa repetidamente uma *suite* de comandos **enquanto** uma dada *condição* for verdadeira e tem a seguinte estrutura básica...

```
while condição:
    suite
```

Um comando `while` começa avaliando a *condição*.

- Se a *condição* for considerada `False`, o `while` termina sem qualquer efeito e o controle passa para o próximo comando na sequência.
- Se a *condição* for considerada `True`, a *suite* é executada, após o que a *condição* é reavaliada e o processo se repete.

Examinando a estrutura típica de um `while`...

```
while condição:
    suite
```

podemos tirar duas conclusões importantes:

- O código que antecede o `while` deve inicializar as variáveis que aparecem na *condição* para permitir sua avaliação inicial.
- Para evitar que o `while` execute eternamente, a *suite* deve atualizar os valores associados a algumas variáveis da *condição* de modo que, após um número finito de iterações, esta seja avaliada como `False`.

1.1.1 Exemplo: Exibir um número arbitrário de Xs

Uma vez entendido o funcionamento do `while`, será possível resolver o exemplo anterior?

```
In [ ]: numXs = int(input("Quantos Xs? "))
        ainda_faltam = numXs
        while ainda_faltam > 0:
            print("X")
            ainda_faltam -= 1
```

Note que...

- `ainda_faltam` é inicializado na linha 2, antes de ser usado na *condição* da linha 3.
- `ainda_faltam` é decrementado na linha 5, o que faz com que o resultado da avaliação da *condição* da linha 3 se torne `False` após um número finito de iterações.

```
In [ ]: numXs = int(input("Quantos Xs? "))
        ja_fiz = 0
        while :
           print("X")
```

Note que...

- `ja_fiz` é inicializado na linha 2, antes de ser usado na *condição* da linha 3.
- `ja_fiz` é incrementado na linha 5, o que faz com que o resultado da avaliação da *condição* da linha 3 se torne `False` após um número finito de iterações.

1.1.2 Exemplo: *Encontrar o mínimo múltiplo comum de dois inteiros positivos dados*

Problema. Ler dois inteiros positivos e exibir o menor inteiro que pode ser dividido por ambos, sem deixar resto.

Solução. Este problema pode ser resolvido por força bruta, se testarmos possíveis candidatos em ordem crescente.

```
In [20]: a = int(input('Primeiro número? '))
         b = int(input('Segundo número? '))
         mmc = 1
         while (mmc % a != 0) or (mmc % b != 0):
             mmc += 1
         print('O mínimo múltiplo comum de', a, 'e', b, 'é', mmc)
```

```
Primeiro número? 4
Segundo número? 5
O mínimo múltiplo comum de 4 e 5 é 20
```

```
In [ ]: (mmc é divisível por a) e (mmc é divisível por b)
```

```
In [ ]: a = int(input('Primeiro número? '))
        b = int(input('Segundo número? '))
        mmc = 1
        while (mmc % a != 0) or (mmc % b != 0):
            mmc += 1
        print('O mínimo múltiplo comum de', a, 'e', b, 'é', mmc)
```

1.1.3 Exemplo: *Achar o maior número ímpar entre 5 candidatos inteiros não-negativos*

Problema. Ler 5 inteiros não-negativos e mostrar o maior número ímpar dentre eles ou uma mensagem apropriada caso todos sejam pares.

Raciocínio

- Como ainda não sabemos como armazenar uma coleção de objetos, vamos ter que tomar decisões à medida em que formos lendo os candidatos.

Raciocínio

- Pense numa variável como sendo um *post-it* que pode ser aplicado a um objeto qualquer.
- Suponha que o *post-it* esteja grudado no maior ímpar já lido. Ao lermos um novo ímpar **maior do que aquele que está com o *post-it***, transferimos o *post-it* para ele.

Raciocínio - Repetimos esse raciocínio 5 vezes e, ao final, quem estiver com o *post-it* será a resposta desejada.

Pergunta: Quem vai estar com o *post-it* no início do programa?

- Como ainda não lemos número algum, uma saída é colar o *post-it* num ***"candidato imaginário e impossível"*** que seja superado pelo primeiro número ímpar que aparecer, qualquer que seja ele.

Como encontrar um ***"candidato imaginário e impossível"***?

- Como o enunciado nos diz que todos os candidatos serão não-negativos, qualquer inteiro negativo (p.ex. `-1`) pode servir como *"candidato impossível"*.
- Além disso, se ao final o *post-it* ainda estiver com ele, saberemos com certeza que todos os números lidos foram pares.

Com isso já podemos esboçar uma solução para o nosso problema.

Vamos usar o modelo de *engenharia reversa* e supor que o resultado desejado seja associado a uma variável `maior_impar`, inicializada com o valor do *candidato imaginário e impossível* `-1`.

Qual seria um possível *último comando*?

```
In [ ]: maior_impar = -1


        if maior_impar == -1:
```

```
In [ ]: maior_impar = -1
        i = 0
        while i < 5:
            cand = int(input('Próximo número? '))
            if (cand % 2 != 0) and (cand > maior_impar):
                maior_impar = cand
            i += 1
        if maior_impar != -1:
            print("maior ímpar =", maior_impar)
        else:
            print("Nenhum candidato ímpar.")
```

Podemos agora criar o loop usando uma variável contadora...

```
In [ ]: maior_impar = -1
        num_cands_lidos = 0
        while :


        if maior_impar != -1:
            print("maior ímpar =", maior_impar)
        else:
            print("Nenhum candidato ímpar.")
```

```
In [ ]: maior_impar = -1
        num_cands_lidos = 0
        while num_cands_lidos < 5:


            num_cands_lidos += 1
        if maior_impar != -1:
            print("maior ímpar =", maior_impar)
        else:
            print("Nenhum candidato ímpar.")
```

Agora podemos ler um candidato...

```
In [ ]: maior_impar = -1
        num_cands_lidos = 0
        while num_cands_lidos < 5:


            num_cands_lidos += 1
        if maior_impar != -1:
            print("maior ímpar =", maior_impar)
        else:
            print("Nenhum candidato ímpar.")
```

```
In [ ]: maior_impar = -1
        num_cands_lidos = 0
        while num_cands_lidos < 5:
            cand = int(input("Candidato " + str(num_cands_lidos) + "? "))

            num_cands_lidos += 1
        if maior_impar != -1:
            print("maior ímpar =", maior_impar)
        else:
            print("Nenhum candidato ímpar.")
```

E, finalmente, testar se o *post-it* deve ser passado para ele...

```
In [ ]: maior_impar = -1
        num_cands_lidos = 0
        while num_cands_lidos < 5:
            cand = int(input("Candidato " + str(num_cands_lidos) + "? "))

            num_cands_lidos += 1
        if maior_impar != -1:
            print("maior ímpar =", maior_impar)
        else:
            print("Nenhum candidato ímpar.")
```

```
In [ ]: maior_impar = -1
        num_cands_lidos = 0
        while num_cands_lidos < 5:
            cand = int(input("Candidato " + str(num_cands_lidos) + "? "))
            if cand % 2 == 1 and cand > maior_impar:
```

```
                maior_impar = cand
            num_cands_lidos += 1
        if maior_impar != -1:
            print("maior ímpar =", maior_impar)
        else:
            print("Nenhum candidato ímpar.")
```

E assim temos uma solução para o nosso problema...

```
maior_impar = -1
num_cands_lidos = 0
while num_cands_lidos < 5:
    cand = int(input("Candidato " + num_cands_lidos + "? "))
    if cand % 2 == 1 and cand > maior_impar:
        maior_impar = cand
    num_cands_lidos += 1
if maior_impar != -1:
    print("maior ímpar =", maior_impar)
else:
    print("Nenhum candidato ímpar.")
```

Embora correta, essa solução desperta pelo menos duas preocupações:

- Nem sempre será possível tomar decisões sem poder examinar simultaneamente todos os candidatos.
- Para controlar o loop, tivemos que criar e gerenciar uma variável `num_cands_lidos` que não nos interessava diretamente.