Do, For, While, e melhor ainda: Nenhuma das opções

Maria Madrugo, com ajuda do StackExchange

Objectivo

No que se segue, vou tentar expôr alguns aspectos do uso de Loops em Mathematica que acho importantes ter em mente quando se programa em Mathematica. Para tal, uso como inspiração respostas no StackExchange, escritas por pessoas que certamente sabem bastante mais do assunto do que eu. Os links para estas respostas (caixinhas roxas) são dados, e tento fazer um apanhado das ideias principais.

Resumidamente, estes são os pontos principais que vão ser estudados aqui:

1. Porque é que se deve evitar While sempre que possível e quando optar por este comando;
2. Porque é que se deve usar Do em vez de For;
3. Porque é que se deve evitar Loops Do, While, For;
4. Que alternativas existem, nomeadamente as vantagens de uma approach orientada a listas.

Problemas e Soluções

Quando não usar While

While é um comando muito útil, mas que tem os seus riscos. Nomeadamente, o While é um dos comandos mais susceptível de dar problemas como Loops infinitos. Basta o comando de iteração não estar correcto ou a condição estar *off by one* e ficamos com um programa que não funciona e não imprime nada que nos possa ajudar a diagnosticar o problema. On this note,

- É muito comum obter um loop infinito quando temos um comando If dentro de um While. Como geralmente não usamos o caso falso, basta um erro de parentesis para estarmos a incrementar o iterador apenas no caso da condição não se verificar e entrarmos num loop infinito.

  Ex: While[a>1, If[EvenQ[a], b=b+1, a=a-1]] vs While[a>1, If[EvenQ[a], b=b+1]; a=a-1];

Resumindo:

Filosofia do While: Usar While apenas nos casos em que não sabemos quando queremos parar de iterar, como por exemplo se quisermos simular as iteradas da conjetura de Collatz (à partida não fazemos ideia quando chegamos a 1). Se queremos que o nosso iterador vá até n, se queremos iterar uma lista ou se queremos repetir o mesmo comando um número fixo de vezes há formas melhores e mais seguras de o fazer.

Filosofia do If e While: Cuidado com parentesis!

Evitar o For loop

Apesar de o Mathematica ter uma função For com funcionamento semelhante ao da maioria das linguagens, enquanto este é essencial e muito usado nessas, no caso do Mathematica é algo a evitar. Este link tem uma resposta muito útil a esta questão, cujos pontos essenciais eu tento resumir nestes bulletpoints:

- Em muitos dos casos, a notação do Do é mais fácil do que a do For. Em vez de ter a mecânica do costume para iteradores, o Do tem no final uma forma suscinta de indicar o iterador:
  - O comando For[i=0, i<=n, i++, (*code*)] pode ser escrito como Do[(*code*), {i,0,n}];
  - O comando For[i=0, i<=n, i=i+0.1, (*code*)] pode ser escrito como Do[(*code*), {i,0,n,0.1}];
  - O comando Do pode iterar sobre uma lista *listy* facilmente, fazendo Do[(*code*), {i,listy}]. É claro que o mesmo é possível no For, mas a notação fica muito mais pesada.
- O For tem duas "rasteiras" que o Do não tem:
  - A primeira diz respeito ao facto do For existir noutras linguagem, mas com uma notação ligeiramente diferente: geralmente os comandos são separados por pontos e virgulas, enquanto em Mathematica o facto de serem parâmetros da função For leva a serem separados por virgulas. É um erro fácil de cometer por hábito e subtil o suficiente para não saltar à vista quando se procura a falha no código.
  - Enquanto que no comando Do o iterador é localizado (isto é, não existe fora do Do, modificar o seu valor não tem impacto no restante código, e vice-versa), no comando For o iterador é uma variável global, o que pode causar problemas noutro local do código. Por exemplo, podemos ter vários Do diferentes com o mesmo nome para o iterador, mas se fizermos o mesmo para o For pode correr mal. Isto pode ser resolvido usando o comando Module, mas é mais uma preocupação que pode ser evitada escolhendo usar Do.
- O For é um comando pouco característico de Mathematica, pelo que o seu uso mantém iniciantes na zona de conforto proveniente das outras linguagens, dificultando a aprendizagem de funções e raciocínios essenciais ao paradigma especifico de Mathematica.
  - Usar o Do, nomeadamente a notação de iteradores, faz a ponte para outras funções importantes e características do Mathematica, nomeadamente a função Table. Esta função cria uma lista em função de um iterador, iterador este que funciona da mesma forma que no comando Do.
  - Na próxima secção, introduzimos uma abordagem mais direccionada a listas, que requer bastantes menos iteradores e que frequentemente simplifica o código. Ao mantermo-nos muito colados ao For, perdemos todos estes comandos e estratégias que facilitariam bastante a nossa vida.

Filosofia do For (traduzida do link acima): "O meu maior argumento contra usar for é que atrapalha a aprendizagem, ao encorajar código lento, difícil de ler e susceptível a erros".

Disclaimer: Existem situações ocasionais nas quais é prático usar For. Não vou falar delas aqui, mas o link acima fala do assunto se estiverem interessados. Também tem mais exemplos, com a vantagem de virem a cores.

Evitar Loops em geral; Alternativas

Até agora, limitámo-nos a comparar os diferentes iteradores possíveis. Nesta secção, espero mostrar que existem muitas situações nas quais a implementação mais fácil e limpa não usa iteradores de todo. Para isso, uso em parte como inspiração esta entrada do Stack Exchange.

De forma a não sobrecarregar este pdf, vou apenas dar alguns exemplos de código no qual Loops são uma complicação desnecessária. Detalhes sobre como trabalhar com listas, nomeadamente com as funções Map e Apply, não serão assumidos como pré-requisito, em vez disso estarão no pdf *Listas e Programação Funcional*. A sua leitura é aconselhada para complementar este pdf, não só para aprender como usar listas e programação funcional em Mathematica, mas também para ter mais exemplos e informação sobre a sua utilidade.

Vejamos então dois exemplos simples, ambos partindo duma lista *listy*. No primeiro, queremos ver a soma dos elementos da lista. Eis várias formas de o fazer:

- Module[{i,total=0}, For[i=1, i<=Length[listy], i++, total=total+listy[[i]]], total]. Aqui, estamos a usar um For para iterar na lista e a guardar a informação na variável total, e a usar o Module para fazer das variáveis locais.
- total=0; Do[total=total+listy[[i]], {i,1,Length[listy]}]; total. Essencialmente o mesmo que o exemplo anterior, mas é claro que o código está muito mais limpo, até por não ser preciso usar Module.
- Total[listy]. That's it! O Mathematica tem uma função para fazer isto directamente[1]

Neste segundo exemplo, queremos somar 1 a todos os elementos duma lista. Omitimos os códigos com Do e For, uma vez que são semelhantes ao anterior, mas sem a variável auxiliar. Usando apenas a forma como Mathematica trabalha com listas, podemos somar 1 a todos os elementos da seguinte forma:

- listy+1.

Estas são apenas duas de muitas operações de listas implementadas no Mathematica, que facilitam muito a escrita de código que manipule listas. Com estas, deixa de ser preciso andar sempre com loops, iteradores e indexação atrás. Além disso, as funções Map e Apply permitem aplicar qualquer função a listas, a primeira elemento a elemento e a segunda tomando a lista como conjunto de parâmetros. Uma explicação detalhada destas operações e comandos encontra-se, tal como indicado acima, neste pdf *Listas e Programação Funcional*

[1]Na verdade, isto é um acontecimento muito comum em Mathematica. Se queres implementar alguma coisa razoavelmente comum, há uma boa probabilidade de já existir essa função, pelo que é aconselhável que tentes fazer uma pesquisa rápida antes de o tentares fazer à mão. Podes encontrar exemplos, juntamente com outras dicas gerais, aqui