02

# DSLs e o Interpreter

## Interpretando uma expressão aritmética

Muitas vezes temos problemas que são bem representados por uma árvore. Suponha que devamos fazer uma calculadora científica, que é composta por diversas operações, desde as mais simples até as mais complexas.

Por exemplo, imagine que seu sistema deve conseguir resolver a expressão "(2+3)-4/2". Ele deve ser responsável por entender essa string e interpretá-la.

É, com certeza um trabalho complicado. Você consegue pensar em uma implementação pra isso que seja simples? Podemos até utilizar recursos poderemos das expressões regulares suportados pelo Python, mas ainda assim não seria uma tarefa tão trivial.

## Entendendo a árvore de interpretação

Dado que temos muitas expressões diferentes em uma calculadora, como por exemplo, adição, subtração, etc, precisamos achar uma maneira simples de lidar com essa complexidade, e podermos criar novas expressões.

Imagine então a representação dessas operações como classes. Dessa maneira, teríamos as classes `Soma`, `Subtracao`, inclusive `Numero`. Na hora de utilizarmos todas elas, teremos algo assim:

```
expressao_conta = Soma(Numero(10), Numero(20))
```

Podemos ter expressões ainda mais complicadas:

```
expressao_esquerda = Substracao(Numero(10), Numero(5))
expressao_direita = Soma(Numero(2), Numero(10))
expressao_conta = Soma(expressao_esquerda, expressao_direita)
```

Veja que uma subtração recebe no construtor duas outras "expressões", que podem ser um número ou mesmo uma outra expressão.

Vamos ver a implementação disso, em código:

```
class Substracao(object):

    def __init__(self, expressao_esquerda, expressao_direita):
        self.__expressao_esquerda = expressao_esquerda
        self.__expressao_direita = expressao_direita

class Soma(object):

    def __init__(self, expressao_esquerda, expressao_direita):
        self.__expressao_esquerda = expressao_esquerda
        self.__expressao_direita = expressao_direita
```

Veja que a classe `Soma` e a classe `Subtracao` cada uma guarda outras duas expressões dentro. O que faz sentido, já que a subtração subtrai o número da esquerda com o da direita. Análogo para a soma.

Precisamos também implementar a classe `Numero`, que representa um simples número:

```
class Numero(object):

    def __init__(self, numero):
        self.__numero = numero
```

Excelente. Já temos a nossa estrutura de dados pronta. Ela consegue bem representar qualquer expressão que queremos. Basta instanciar os objetos corretos.

O próximo passo agora é interpretar essa "árvore". Já que todos os elementos devem ser interpretados; podemos garantir isso colocando o método `avalia()` em cada classe. Vamos começar pela classe `Numero`. Interpretar um `Numero` é fácil: basta retornar ele próprio.

```
class Numero(object):

    def __init__(self, numero):
        self.__numero = numero

    def avalia(self):
        return self.__numero
```

Já o da soma ou da subtração, é um pouco mais complicado. Precisamos "interpretar" o lado da esquerda primeiro, depois o da direita, e aí sim fazer a conta:

```
class Substracao(object):

    def __init__(self, expressao_esquerda, expressao_direita):
        self.__expressao_esquerda = expressao_esquerda
        self.__expressao_direita = expressao_direita

    def avalia(self):
        return (self.__expressao_esquerda.avalia()
            - self.__expressao_direita.avalia())

class Soma(object):

    def __init__(self, expressao_esquerda, expressao_direita):
        self.__expressao_esquerda = expressao_esquerda
        self.__expressao_direita = expressao_direita

    def avalia(self):
        return (self.__expressao_esquerda.avalia()
            + self.__expressao_direita.avalia())
```

Veja que cada `avalia()` agora sabe interpretar sua própria expressão, e expressões podem avaliar outras expressões, podemos tentar calcular o resultado de uma expressão inteira.

```
if __name__ == '__main__':

    expressao_esquerda = Substracao(Numero(10), Numero(5))
    expressao_direita = Soma(Numero(2), Numero(10))
    expressao_conta = Soma(expressao_esquerda, expressao_direita)

    resultado = expressao_conta.avalia()
    print resultado
```

Veja que nossa árvore consegue interpretar, e calcular o resultado final. Quando temos expressões que devem ser avaliadas, e a transformamos em uma estrutura de dados, e depois fazemos com que a própria árvore se avalie, damos o nome de **Interpreter**.

O padrão é bastante útil quando temos que implementar interpretadores para DSLs, ou coisas similares. É um padrão bem complicado, mas bastante interessante.