python-funcional

Quase um livro, quase um tutorial, quase qualquer coisa

Project maintained by dunossauro
Hosted on GitHub Pages — Theme by mattgraham

Retorne «< 1. Funções - Continue lendo »> 3. Consumindo Interáveis

2. Iteráveis e iteradores

O que são iteráveis? Basicamente e a grosso modo, iteráveis em python são todos os objetos que implementam o método `__getitem__` ou `__iter__` . Beleza, vamos partir do simples.

Quase todos os tipos de dados em python são iteráveis, por exemplo: listas, strings, tuplas, dicionários, conjuntos, etc...

Vamos aos exemplos, eles são sempre mágicos:

```
lista = [1, 2, 3, 4, 5]

# iteração
for x in lista:
    print(x)

# 1
# 2
# 3
# 4
# 5
```

Era só isso? Sim, nem doeu, fala a verdade.

Em python, o comando `for` nos fornece um iterador implícito. O que? Não entendi.

ou "foreach") no caso vai passando por cada elemento da sequência. Não é necessária a implementação de um index como na linguagem C, onde a iteração é explícita:

```
for (i = 0; i > 10; i++){
    sequencia[i];
}
```

2.1 `__getitem__`

O padrão de projeto iterator em python já vem implementado por padrão, como já foi dito antes. Basta que um objeto tenha os métodos `__iter__` ou `__getitem__` para que um laço possa ser utilizado.

Vamos exemplificar:

```
class iteravel:
    """
    Um objeto que implementa o `__getitem__` pode ser acessado por posição
    """
    def __init__(self, sequencia):
        self.seq = sequencia

    def __getitem__(self, posicao):
        """
        Por exemplo, quando tentamos acessar um elemento da sequência usando
        slice:
            >>> iteravel[2]

        O interpretador python chama o __getitem__ do objeto e nos retorna
            a posição solicitada

        um exemplo:
        IN: lista = [1, 2, 3, 4]
        IN: lista[0]
        OUT: 1
        """
        return self.seq[posicao]
```

Então, pode-se compreender, sendo bem rústico, que todos os objetos que implementam

2.2 `__iter__`

Agora os objetos que implementam `__iter__` tem algumas peculiaridades. Por exemplo, quando o iterável (vamos pensar no `for`) chamar a sequência, ela vai pedir o `__iter__` que vai retornar uma instância de si mesmo para o `for` e ele vai chamar o `__next__` até que a exceção `StopIteration` aconteça.

Uma classe que implementa `__iter__` :

```
class iteravel:
    def __init__(self, sequencia):
       self.data = sequencia

    def __next__(self):
        """
        Neste caso, como o método pop do objeto data é chamado
            (vamos pensar em uma lista) ele vai retornar o primeiro elemento
            (o de index 0) e vai remove-lo da lista

        E quando a sequência estiver vazia ele vai nos retornar um StopIterat
            O que vai fazer com que a iteração acabe.

        Porém, estamos iterando com pop, o que faz a nossa iteração ser a úni
            pois ela não pode ser repetida, dado que os elementos foram
            removidos da lista
        """
        if not self.sequencia:
           raise StopIteration
        return self.sequencia.pop()

    def __iter__(self):
        return self
```

Como é possível notar, o objeto com `__iter__` não necessita de um `__getitem__` e vise-versa. As diferenças partem do ponto em que um pode ser acessado por index/slice e outro não. Também um bom ponto é que nesse nosso caso, removemos os elementos da sequência, ou seja, ela se torna descartável.

Esse conceito, de ser descartado, pode parecer um pouco estranho no início, mas

dizer que nossa sequência se torna imutável, pois não existe uma maneira de mudar os valores contidos na lista do objeto.

Seguem dois links maravilhosos explicando sobre iteração em python:

- PEP0243
- Iteração em Python: do básico ao genial

O primeiro é a PEP sobre as estruturas dos iteráveis e o segundo um video do Guru Luciano Ramalho explicando tudo sobre iteradores.

Ah... Ia quase me esquecendo, se você não entendeu muita coisa sobre os dunders, você pode ler o Python data model. Obs: não me responsabilizo pelo programador melhor que você sairá desta página.

Embora esse tópico seja talvez o mais curto, ele vai ser de fundamental importância para o entendimento de um pouco de tudo nesse 'Curso'. É sério. Vamos entender como trabalhar com iteráveis de uma maneira bonita no próximo tópico.

Retorne «< 1. Funções - Continue lendo »> 3. Consumindo Interáveis