nappacademy

Napp Academy

Orlando Saraiva Júnior

Comentários nos formulários...

nappacademy

Deixe um comentário sobre as atividades da semana. (semana 12)

Gostei das atividades. Porém gostaria de saber se terá conteúdo mais avançado sobre analise e manipulação de dados, principalmente de arquivos muito extensos e se teria alguma estratégia para lidar com grande quantidade de dados.

Iteráveis, iteradores e geradores

Iteráveis, iteradores e geradores

nappacademy

Iteração é fundamental para o processamento de dados. Ao percorrer conjunto de dados que não cabem na memória, precisamos ter como acessar os itens de modo *lazy* (preguiçoso), isto é, um item de cada vez, sob demanda.

É disto que trata o padrão *Iterator* (Iterador).

Vamos mostrar como este padrão está integrado à linguagem de modo que não precisamos implementar manualmente.

Iteráveis, iteradores e geradores

nappacademy

A palavra yield permite a construção de geradores que funcionam como iteradores. Python usa geradores em vários lugares.

Até a função range() devolve um objeto gerador, e não mais uma lista, como em python2. Se precisarmos de uma lista a partir de um range, precisamos ser explícitos.

Iteráveis, iteradores e geradores

nappacademy

Toda coleção em python é ***iterável***, e os interadores são usados internamente para dar suporte a:

- laços for;
- construção e extensão de tipos para coleção;
- percorrer arquivo-texto linha a linha em um laço;
- list, dict e set comprehensions;
- desempacotamento de tuplas;
- desempacotamento de parâmetros com * em chamadas de função

Iteráveis, iteradores e geradores

nappacademy

A classe Sentence extrai palavras de um texto pelo índice.

Todo programador python sabe que sequencias são iteráveis. Agora vamos descobrir exatamente por quê.

sentence1.py

Iteráveis, iteradores e geradores

nappacademy

Sempre que o interpretador precisa iterar por um objeto x, ele chama iter(x) automaticamente.

A função embutida iter:

1) Verifica se o objeto implementa **__iter__** e o chama para obter o iterador.
2) Se **__iter__** não estiver implementado, mas o **__getitem__** estiver, python criará um iterador que permite acessar os itens em sequencia, começando no índice 0 (zero)
3) Se isso falhar, Python levantará **TypeError**, normalmente dizendo que o objeto não é iterável.

Iteráveis, iteradores e geradores

nappacademy

Por isso qualquer sequencia em Python é iterável: todas elas implementam **__getitem__**. Todas as sequencias padrões também implementam **__iter__**.

Um objeto é considerado iterável não só quando implementa o método especial __iter__, mas também quando implementa o método __getitem__, desde que este aceite chaves int.

Iteráveis, iteradores e geradores

nappacademy

Iterável é qualquer objeto a partir do qual a função embutida iter pode obter um iterador. Objetos que implementem um método __iter__ que devolva um iterador são iteráveis. Sequencias sempre são iteráveis, assim como objetos que implementem um método __getitem__ que aceite índices a partir de 0.

Python obtém iteradores a partir de iteráveis.

simula_for.py

Iteráveis, iteradores e geradores

nappacademy

A interface padrão de um iterador tem dois métodos:

__next__

Devolve o próximo item disponível, levantando StopIteration quando não houver mais itens

__iter__

Devolve self; permite que iteradores sem usados em lugares em que se espera um iterável, por exemplo, em um laço for.

sentence1.py

Iteráveis, iteradores e geradores

nappacademy

Iterador é qualquer objeto que implemente o método **__next__**, sem argumentos, que devolva o próximo item de uma série ou levante **StopIteration** quando não houver mais itens. Os iteradores em Python também implementam o método __iter__, portanto também são iteráveis.

Padrão de Projeto Iterator clássico

nappacademy

A próxima classe Setence foi criada de acordo com o padrão de projeto Iterator clássico, seguindo o esquema do livro GoF.

Esta implementação tem como objetivo deixar clara a distinção fundamental entre um iterável e um iterador e como eles estão relacionaidos.

sentence2.py

Iteráveis, iteradores e geradores

nappacademy

Uma causa comum de erros na criação de iteráveis e iteradores é confundí-los.

Um iterável tem o método __iter__ que sempre instancia um novo iterador.

Iteradores implementam o método __next__ que devolve itens individuais e um método __iter__ que devolve self.

Um iterável jamais deve atuar como um iterador de si mesmo.

Função geradora

nappacademy

Uma implementação mais pythonica da mesma funcionalidade usa a função geradora para substituir a classe SequenceIterator.

sentence3.py

Função geradora

nappacademy

Qualquer função python que tenha a palavra reservada yield em seu corpo é uma função geradora: uma função que, quando chamada, devolve um objeto gerador.

Em outras palavras, uma função geradora é uma fábrica (factory) de geradores.

Uma implementação lazy

nappacademy

A interface do Iterador foi concebida para ser lazy: next(my_iterador_ produz um item de cada vez. O oposto ao lazy é eager (ávido).

As implementações até agora não foram lazy, pois __init__ cria uma lista de forma ávida com todas as palavras do texto, associando-a ao atributo self.word.

Isto implica em processar todo o texto, e a lista pode usar tanta memória quanto o próprio texto.

Uma implementação lazy

nappacademy

A função **re.finditer** é uma versão *lazy* de **re.findall**. Em vez de uma lista, ela devolve um gerador que produz instâncias **re.MatchObject** sob demanda.

Assim, esta versão é *lazy*, produzirá a próxima palavra somente quando necessário.

sentence4.py

Iteráveis, iteradores e geradores

nappacademy

A biblioteca padrão oferece muitos geradores, desde objetos-arquivo de texto puro, que oferecem iteração linha a linha, até a função os.walk, que produz nomes de arquivos enquanto percorre uma árvore de diretório, deixando as buscas recursivas no sistema de arquivos tão simples quanto um laço for.

Além disso, o módulo **itertools** possui um conjunto de iteradores.

iter.py

Performance

nappacademy

```
import sys
nums_squared_lc = [i * 2 for i in range(10000)]
sys.getsizeof(nums_squared_lc)
nums_squared_gc = (i ** 2 for i in range(10000))
print(sys.getsizeof(nums_squared_gc))
```

Performance

nappacademy

```
import cProfile
cProfile.run('sum([i * 2 for i in range(10000)])')
cProfile.run('sum((i * 2 for i in range(10000)))')

cProfile.run('sum([i * 2 for i in range(10000000)])')
cProfile.run('sum((i * 2 for i in range(10000000)))')
```

Acesso a grandes arquivos

nappacademy

Para acesso a grandes arquivos, geradores economizam recursos computacionais.

arquivo.py

Fonte dos dados

https://brasil.io/dataset/eleicoes-brasil/votacoes/

https://brasil.io/dataset/covid19/caso/