Introdução à Ciência da Computação

Disciplina: 113913

Prof. Edison Ishikawa

Python 3.0
Aula 5
O Caminho do Programa

UnB | CIC

Sumário

- Dicionários
- Operações dos Dicionários
- Métodos dos Dicionários
- Aliasing (XXX) e Copiar
- Matrizes Esparsas
- Hint XXX
- Inteiros Longos
- Contando Letras
- Glossário

Dicionários

Dicionários são semelhantes a outros tipos compostos, entretanto eles podem usar qualquer tipo de dados como índice. Strings, listas e tuplas utilizam inteiros como índices. Como exemplo, um dicionário para traduzir palavras do inglês para o espanhol, será criado. Para esse dicionário, usaremos strings como índices.

Uma maneira de criar dicionários é começando com um dicionário vazio e depois adicionando elementos. Um dicionário vazio é denotado assim {}

```
>>> ing2esp = {}
>>> ing2esp['one'] = 'uno'
>>> ing2esp['two'] = 'dos'
```

Dicionários

Nos podemos imprimir o valor corrente de um dicionário da maneira usual:

```
>>> print (ing2esp)
{'one': 'uno', 'two': 'dos'}
```

Os elementos de um dicionário aparecem em uma lista separada por vírgulas. Cada entrada contêm um índice e um valor separado por dois-pontos. Em um dicionário, os índices são chamados de *chaves*, então os elementos são chamados de pares *chave-valor*.

Dicionários

Outra maneira de criar dicionários é fornecer uma lista de pares chaves-valor utilizando a mesma sintaxe da última saída.

```
>>> ing2esp = {'one': 'uno', 'two': 'dos', 'three': 'tres'}
```

Se nos imprimirmos o valor de ing2esp novamente, nos teremos uma surpresa:

```
>>> print (ing2esp)
{'one': 'uno', 'three': 'tres', 'two': 'dos'}
```

Dicionários

Os pares chave-valor não estão em ordem! Felizmente, não há motivos para se preocupar com a ordem, desde que os elementos do dicionário nunca sejam indexados com índices inteiros. Podemos usar as chaves para buscar os valores correspondentes:

```
>>> print (ing2esp['two'])
'dos'
```

A chave *'two'* retornou o valor *'dos'* mesmo pensando que retornaria o terceiro par chave-valor.

Operações com Dicionários

- O comando *del* remove um par *chave-valor* de um dicionário. Por exemplo, o dicionário abaixo contem os nomes de várias frutas e o número de cada fruta em no estoque:

```
>>> inventario = {'abacaxis': 430, 'bananas': 312, 'laranjas': 525, 'peras': 217}
>>> print (inventario)
{'laranjas': 525, 'abacaxis': 430, 'peras': 217, 'bananas': 312}
```

- Se alguém comprar todas as peras, podemos excluir a entrada do dicionário:

```
>>> del inventario['peras']
>>> print (inventario)
{'laranjas': 525, 'abacaxis': 430, 'bananas': 312}
```

Operações com Dicionários

- Ou, se nós esperamos por mais peras em breve, nos podemos simplesmente trocar o valor associado às peras:

```
>>> inventario['peras'] = 0
>>> print (inventario)
{'laranjas': 525, 'abacaxis': 430, 'peras': 0, 'bananas': 312}
```

- A função *len* também funciona com dicionários; retornando o número de pares chave-valor:

```
>>> len(inventario)
4
```

Métodos dos Dicionários

- Um **método** é semelhante a uma função: possui parâmetros e retorna valores, mas a sintaxe é diferente. Por exemplo, o método *keys* recebe um dicionário e retorna uma lista com as chaves, mas em vez de usarmos a sintaxe de função *keys(ing2esp)*, nós usamos a sintaxe de método *keysing2esp.keys()*:

```
>>> ing2esp.keys()
['one', 'three', 'two']
```

- Os parênteses indicam que o método não possui parâmetros.

Métodos dos Dicionários

- Ao invés de chamarmos um método, dizemos que ele é invocado, nesse caso, nós podemos dizer que nós estamos invocando *keys* do objeto ing2esp.
- O método *values* é parecido; retorna a lista de valores de um dicionário:

```
>>> ing2esp.values()
['uno', 'tres', 'dos']
```

- O método *items* retorna dois, na forma de uma lista com os pares. Os colchetes indicam que isso é uma lista. Os parênteses indicam que os elementos da lista são tuplas.

```
>>> ing2esp.items()
[('one','uno'), ('three','tres'), ('two','dos')]
```

Métodos dos Dicionários

- Se o método recebe de algum parâmetro, utiliza-se a mesma sintaxe das funções. Por exemplo, o método *get* recebe uma chave e retorna o valor associado se a chave existe no dicionário, e nada (None) caso contrário:

```
>>> ing2esp.get('one')
'uno'
>>> ing2esp.get('deux')
>>>
```

- Se você tentar chamar um método sem especificar em qual objeto, você obterá um erro. Nesse caso, a mensagem de erro não é muito útil:

```
>>> get('one')
NameError: get
```

Métodos dos Dicionários

- O operador *in* (usado também em strings e listas) é um operador lógico que verifica se uma chave está no dicionário

```
>>> 'one' in ing2esp
True
>>> 'deux' in ing2esp
False
```

Aliasing e Copiar

- Aliasing é um recurso no qual duas variáveis referenciam a um mesmo objeto. Quando uma é alterada, a outra também é afetada.
- Se você quer modificar um dicionário e manter uma cópia original, então o método *copy* deve ser utilizado.
- **opposites** é um dicionário de antônimos:

```
>>> opposites = {'up': 'down', 'right': 'wrong', 'true': 'false'}
>>> alias = opposities
>>> copy = opposities.copy()
```

Aliasing e Copiar

- **opposites** e **alias** se referem ao mesmo objeto, **copy** se refere a um novo objeto igual ao dicionário **opposites**. Se você modificar o **alias**, **opposites** também será alterado.

```
>>> alias['right'] = 'left'
>>> opossites['right']
'left'
```

- Se você modificar o **copy**, **opposites** não será alterado.

```
>>> copy['right'] = 'privilege'
>>> opposites['right']
'left'
```

Matrizes Esparsas

- Seja a matriz esparsa abaixo:

$$\begin{bmatrix} 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 \\ 0 & 2 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 3 & 0 \end{bmatrix}$$

- Uma representação usando uma lista contém muitos zeros:

```
>>> matriz = [ [0,0,0,1,0],
               [0,0,0,0,0],
               [0,2,0,0,0],
               [0,0,0,0,0],
               [0,0,0,3,0] ]
```

Matrizes Esparsas

- Uma alternativa é usarmos um dicionário. Para as chaves, nós podemos usar tuplas que contêm os números da linha e a coluna. Abaixo, uma representação em um dicionário da mesma matriz:

```
>>> matriz = {(0,3): 1, (2, 1): 2, (4, 3): 3}
```

- Nós precisamos apenas de três pares *chave-valor*, cada um sendo um elemento diferente de zero da matriz. Cada chave é uma tupla, e cada valor é um número inteiro.

Matrizes Esparsas

- Para acessarmos um elemento da matriz, nós utilizamos o operador []:

```
>>> matriz[0,3]
1
```

- A sintaxe da representação de um dicionário não é a mesma das listas. Um índice apenas é usado (tupla de inteiros) ao invés de dois.
- Porém, se buscarmos um elemento zero, um erro será gerado, pois não existe entrada no dicionário para a palavra especificada.

```
>>> matriz[1,3]
KeyError: (1,3)
```

Matrizes Esparsas

- O método *get* resolve este problema:

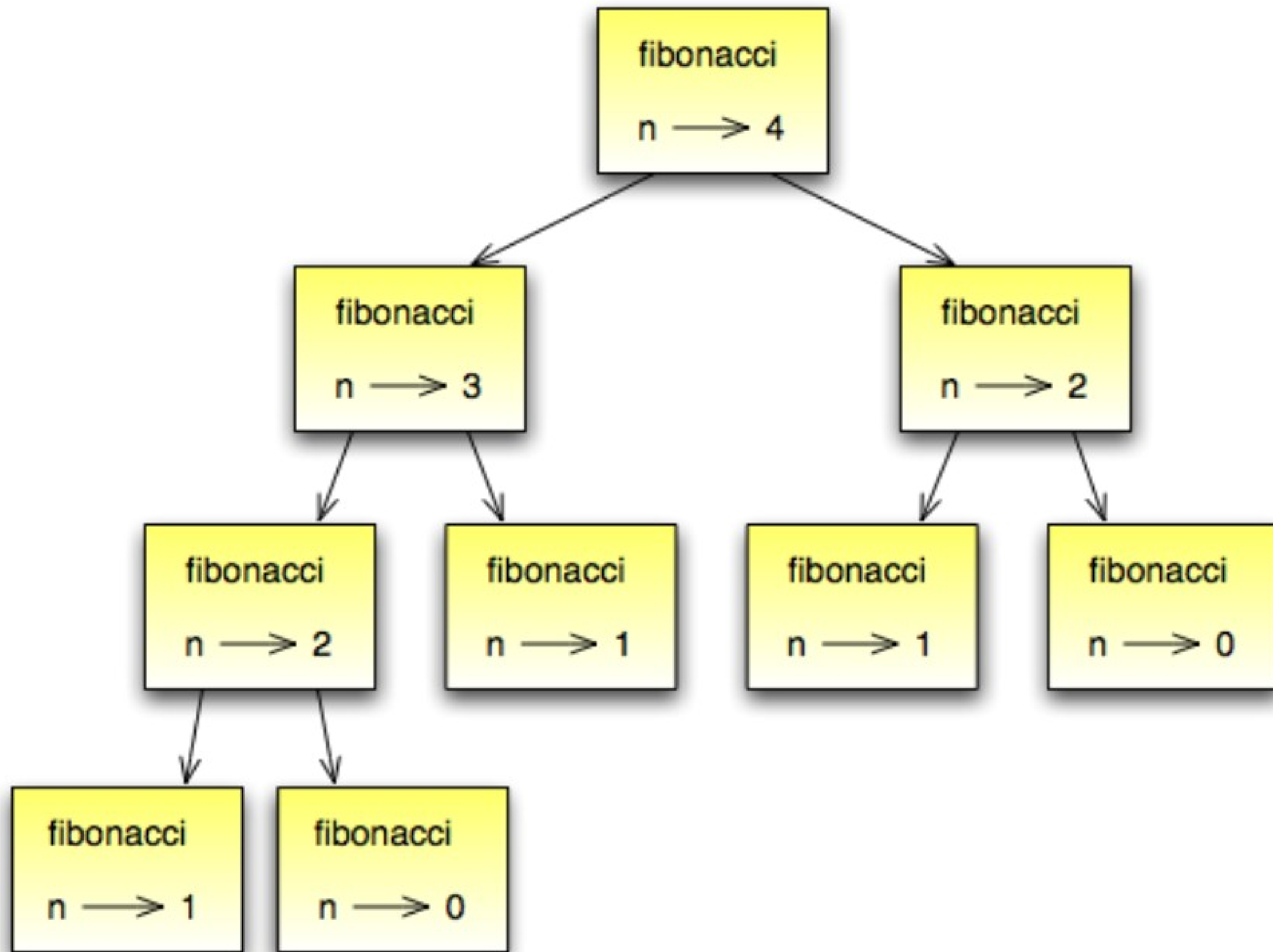
```
>>> matriz.get((0,3), 0)
1
```

- O primeiro parâmetro é a chave; o segundo é o valor que *get* retornará caso não exista a chave no dicionário:

```
>>> matriz.get((1,3), 0)
0
```

- *get* definitivamente melhora a semântica e a sintaxe do acesso a matrizes esparsas.

Hint



Hint

- É uma técnica que guarda os valores que já foram calculados armazenando-os em um dicionário, para que estes valores possam ser utilizados em uma outra oportunidade. Veja o exemplo abaixo:

```
>>> previous = {0:1, 1:1}
>>> def fibonacci(n):
        if previous.has_key(n):
                return previous[n]
        else:
                newValue = fibonacci(n-1) + fibonacci(n-2)
                previous[n] = newValue
                return newValue
```

Hint

- O dicionário chamado **previous** guarda os números de Fibonacci. Ele começa com apenas dois pares: 0 possui 1; e 1 possui 1.
- Sempre que **fibonacci** é chamada, ela verifica o dicionário para determinar se ele já possui o resultado. Se o resultado estiver ali, a função pode retornar imediatamente sempre precisar fazer mais chamadas recursivas. Se o resultado não estiver ali, ele é calculado no **newValue**. O valor de **newValue** é adicionado no dicionário antes da função retornar.

Hint

- Usando essa versão de **fibonacci**, nossa máquina consegue calcular **fibonacci(40)** em um piscar de olhos. Mas quando tentamos calcular **fibonacci(50)**, nós veremos um problema diferente:

```
>>> fibonacci(50)
OverflowError: integer addition
```

- A resposta, que você verá em um minuto, é 20.365.011.074. Esse número é muito grande para guardarmos como um inteiro do Python. Este problema chama-se **overflow** e, felizmente, tem uma solução simples.

Inteiros Longos

- Python possui um tipo chamado **long int** que permite trabalharmos com qualquer tamanho de inteiros. Existem duas maneiras de criarmos um valor **long int**. A primeira é escrever um inteiro seguido de um L no final:

```
>>> type(1L)
<class 'long int'>
```

Inteiros Longos

- A outra maneira é usarmos a função **long** que converte um valor para um **long int**. **long** pode receber qualquer valor númerico e até mesmo uma string de digitos:

```
>>> long(1)
1L
>>> long(3.9)
3L
>>> long('57')
57L
```

```
>>> previous = {0: 1L, 1:1L}
>>> fibonacci(50)
20365011074L
```

Contando letras

- Um histograma pode ser útil para comprimir um arquivo de texto.
- Dicionários fornecem uma maneira elegante de gerar um histograma:

```
>>> letterCounts = {}
>>> for letter in "Mississippi":
...     letterCounts[letter] = letterCounts.get(letter,0) + 1
...
>>> letterCounts
{'M': 1, 's': 4, 'p': 2, 'i': 4}
```

Contando letras

- Começamos com um dicionário vazio. Para cada letra da string, achamos o contador (possivelmente zero) e o incrementamos. No final, o dicionário contém pares de letras e as suas frequências.
- É mais atraente mostrarmos o histograma na ordem alfabética. Podemos fazer isso com os métodos **items** e **sort**:

```
>>> letterItems = letterCounts.items()
>>> letterItems.sort()
>>> print (letterItems)
[('M', 1), ('i', 4), ('p', 2), ('s', 4)]
```

Contando letras

- **Você já tinha visto o método items antes, mas sort é o primeiro método que você se depara para aplicar em listas. Existem muitos outros métodos de listas, incluindo append, extend, e reverse. Consulte a documentação do Python para maiores detalhes.**

Referências

- Aprenda Computação com Python 3.0, Versão 1. Allen Downey, Jeff Elkner and Chris Meyers