[GIM] Curso de Férias - Python

Luiz Eduardo Barros e Victor Matheus Castro

Conteúdo do Dia

- Listas
- Tuplas
- Dicionários
- Set
- Funções

Listas

- Tipo de dado que agrega diversos valores relacionados;
- Armazenamento linear de elementos;
- Valores individuais são identificados por índices (números inteiros);
- Listas podem conter quaisquer valores, incluindo valores de tipos mistos e até outras listas.

Declaração de Listas

>>> vazio = [] Cria uma lista vazia
>>> numeros = [1, 2, 3] Cria uma lista com números
>>> opcoes = ['nao', 'sim', 'talvez'] Cria uma lista com strings
>>> modelos = [3.1, 3.11, 95, 98, 2000, "Ubuntu", "Mac"] Cria uma lista do tipo misto
>>> listas = [numeros, opcoes] Cria uma lista de listas
>>> print listas
$[[1, 2, 3], ['nao', 'sim', 'talvez']]$

Trabalhando com índices

- Índices são atribuidos sequencialmente a partir de zero;
- Para acessar um elemento específico, usamos o nome da lista seguido do índice entre colchetes;

Fatiamento de listas

- O fatiamento é especificado por dois índices i e j separados por dois pontos;
- A fatia conterá os elementos da posição i a j-1.

```
>>> numeros = [1, 2, 3, 4, 5]
>>> numeros[1:3]
[2, 3]
```

Cópia de Listas

- Podemos utilizar o operador de atribuição para copiar elementos entre listas;
- Por padrão, copiamos referências em memória, não os valores em si, ou seja, alterando um, alteramos os dois.
- Para copiar valores ao invés de referências a valores, usa-se o fatiamento de listas.

Adição e Remoção de Elementos

- Método **append**:

  Recebe um valor a ser adicionado no final da lista.

- Instrução **del**:

  Exclui um elemento (ou a lista inteira);

  O elemento excluído não ocupa mais a lista, fazendo com que os índices sejam reorganizados.

Outras Operações

Nome	Descrição	Exemplo
append	insere um elemento no final da lista	list.append(<elemento>)
count	retorna a quantidade de um determinado elemento	list.count(<elemento>)
extend	adiciona os elementos de uma lista dentro de outra	list.extend(<list>)
index	retorna o índice de um elemento da lista	list.index(<elemento>)
insert	insere um elemento em um determinado índice	list.insert(<índice>,<elemento>)
pop	retorna um elemento da lista o excluindo	list.pop(<índice>)
remove	remove um elemento da lista	list.remove(<elemento>)
reverse	inverte a ordem dos elementos	list.reverse()
sort	ordena os elementos	list.sort()

Laço for

- Estrutura de repetição utilizada quando conhecemos o intervalo de valores a percorrer;
- Muito útil para percorrer uma estrutura (ex: lista) em ordem: A cada ciclo substitui a variável da iteração por um elemento da estrutura;
- Sintaxe:

for <variável> in <lista>:
    #Bloco de Instrução

Tuplas

- Agrupamento de valores separados por vírgulas;
- Similar a uma lista só que é um tipo imutável;
- Os valores armazenados podem ser de tipos mutáveis;
- Usadas quando funções necessitam de número variável de argumentos.

Tuplas: Desempacotamento e Indexação

- Métodos de obter os elementos da tupla:

```
>>> c = 1, 3, 5
>>> e, f, g = c
>>> e
1
>>> g
5
```

```
>>> c = 1, 3, 5
>>> print c[0]
1
>>> c[1]
3
```

Atribuições de Tuplas

- Como fazer?

  $>>>$ x, y $=$ y, x

- Dessa forma não é necessário criar uma variável auxiliar para fazer a troca de valores entre x e y.

Operações sobre tuplas

Nome	Operador	Descrição	Exemplo
Adição	$+$	Gera uma nova tupla com a soma das duas	c + b
Comparação	$>, >=, <, <=, ==$	Compara-se com o primeiro elemento diferente	c $>$ b
Pegar pedaço	$[x : y]$	Retorna uma tupla na posição x até y-1	$c[0 : 2]$
Comprimento		Retorna o tamanho	len(b)
Deletar		Deleta a tupla	del c
Está contido		Verifica se c está na tupla	c in b

Transformações entre tupla e lista

- Para transformarmos uma tupla em uma lista utilizamos list(a)
- Para transformarmos uma lista em uma tupla utilizamos tuple(a)

Dicionários

- Mapeamento de valores, formado por pares;
- Cada par consiste em uma chave e seu conteúdo;
- Tipo mútavel;
- Usado quando interessa armazenar objetos em que sua manipulação é facilitada usando chaves para os representar: agendas, descrição de comandos etc.

```
>>> dici = {}
>>> dici["Luiz"] = "9877-5310"
>>> dici["Victor"] = "legal"
>>> dici["GIM"] = "irado"
>>> print dici
{'Luiz': '9877-5310', 'Victor': 'legal', 'GIM': irado}
```

Operações sobre dicionários

Nome	Descrição	Exemplo
Pegar	Retorna o valor de uma dada chave	dici.get(key)
Remover	Remove o par chave-conteúdo do dicionário	del dici[x]
Modificar conteúdo	Modifica o valor do conteúdo de uma determinada chave	dici[x] = y
Comprimento	Retorna o comprimento do dicionário	len(dici)

Nome	Descrição	Exemplo
Valores	Retorna uma lista de valores do dicionário	dici.values()
Chaves	Retorna uma lista de chaves do dicionário	dici.keys()
Itens	Retorna uma lista de tuplas com chave-valor	dici.items()
Testa chave	Retorna true se essa chave existe no dicionário, false caso contrário	dici.has$_k ey$("$Luiz$")
Copia de conteúdo	Copia o conteúdo do objeto para outro	dici2 = dici.copy()
Limpar	Limpa o dicionário	dici.clear()
Retira chave-conteúdo	Deleta e retorna o que foi deletado	dici.pop("GIM")

Set

- Representa um conjunto de valores não ordenados e sem repetição;
- Podemos criar um conjunto a partir de uma lista, tupla, dicionário: set(arg);
- Exemplo de uso: itens cosméticos de um personagem em um jogo, operações dentro de uma sala (incluir ou tirar um aluno).

Operações sobre Set

Nome	Descrição	Exemplo
Adiciona	Adiciona um novo elemento ao conjunto	s.add(x)
Comprimento	Retorna a cardinalidade do conjunto	len(s)
Pertence	Testa se x está em s	x in s
Não Pertence	Testa se x não está em s	x not in s
Remove último	Retorna o elemento tirado do conjunto	s.pop()
Remove x	Retira o elemento, gera erro caso ele não esteja presente	s.remove(x)
Remove x	Retira o elemento caso ele exista	s.discard(x)
É subconjunto	Testa se todos os elementos de s estão em t	s.issubset(t)
Cópia de valor	Copia o conteúdo de s	s.copy()

Nome	Operador	Descrição	Exemplo
União	—	Gera novo conjunto com elementos de ambos s e t	s.union(t)
Diferença	-	Gera novo conjunto com elementos que estão em s, mas não em t	s.difference(t)
Intersecção	&	Gera novo conjunto com elementos em comum aos dois conjuntos	s.intersection(t) ou s&t

Variáveis locais e globais

- Uma variável criada fora de uma função, chamada variável global, pode ser acessada em qualquer lugar do programa, como as variáveis vistas até agora;
- Quando uma variável é criada dentro de uma função, ela recebe o nome de variável local, e seu conteúdo só pode ser acessado dentro da própria função e tentar acessá-la fora do seu escopo resultará em um erro.

Funções

- Função é um bloco de código identificado por um nome, que pode ser "chamado"em algum ponto de um programa;
  def nome_da_funcao (parametros):
  comandos
- Pode receber dados de entrada, chamados parâmetros;
- Pode devolver um valor, chamado o retorno da função.

Funções como parâmetros

- Em Python, é possível passar funções como parâmetro para outra função;

```
>>> def read_reverse(n, reverse):
...         list = []
...         i = 0
...         while i < n:
...                 a = input("Digite um numero: ")
...                 list.append(a)
...                 i += 1
...         reverse(list)
...
>>> read_reverse(6, reverse_print)
Digite um numero: 6
Digite um numero: 4
Digite um numero: 8
Digite um numero: 2
Digite um numero: 68
Digite um numero: 23
23 68 2 8 4 6
```

Parâmetros com Valores Default

- Às vezes, podemos desejar que um parâmetro tenha sempre o mesmo valor, exceto se for explicitamente informado que ele deve possuir outro valor;
- Em Python, chamamos isso de parâmetros com valores default
- A não ser que desejemos que o parâmetro tenha um valor diferente, podemos omitir a passagem do valor para o parâmetro.
- Parâmetros com valores default devem vir OBRIGATORIAMENTE após os outros parâmetros;

Argumentos nomeados

- Um argumento não-nomeado NÃO deve suceder um argumento nomeado;
- Quando não nomeamos alguns argumentos, por padrão eles são atribuídos sequencialmente aos parâmetros da função.

```
>>> def print_numbers(a, b, c):
...         print "%d %d %d"%(a, b, c)
...
>>> print_numbers(10, 20, 30)
10 20 30
>>> print_numbers(b = 10, c = 20, a = 30)
30 10 20
```

Construções Lambda

- Funções lambda são pequenas funções anônimas (não usa-se def) definidas em uma única linha;
- São usadas quando se precisa de uma função de pequeno tamanho e uso pontual.

```
>>> aumento = lambda x,y: (x*y/100 + x)
>>>
>>> aumento(100, 5)
105
```

Funções com Listas Arbitrárias de Argumentos

- Em certas ocasiões, pode ser necessário passar um número arbitrário de parâmetros para uma função, ou seja, passar parâmetros variados;
- Neste caso, há duas soluções diferentes: usar tuplas ou usar dicionários.
- **IMPORTANTE:**
  O Parâmetro que irá receber um número arbitrário de elementos deve SEMPRE vir por último.

Tuplas

```
>>> def mean(*numbers):
...         sum = 0
...         n = 0
...
...         for i in numbers:
...                 sum += i
...                 n += 1
...
...         return float(sum)/n
...
>>>
>>> mean(1,2,3,4,5,6,7,8,9,10)
5.5
```

Dicionários

```
>>> def list_phone(**numbers):
...         print "Lista de contatos: "
...         for contact in numbers:
...                 print (contact, numbers[contact])
...
...
>>> list_phone(Cersei = 91009878, Jaime = 91009879)
Lista de contatos:
Jaime 91009878 Cersei 91009878
```