Trabalho de algoritmos: Dicionário python

Membros: Larissa Fernandes 2022101873, Carlos Eduardo Ferreira 2022101225, Orlando Silva 2022101395, Pedro Henrico 2022101127, Matheus de Souza 2022101315, Guilherme Ribeiro 2022101582

Introdução

Os dicionários são coleções de itens e seus elementos são armazenados de forma não ordenada. Sendo assim seus elementos tem chave e valor, dessa forma irá ter uma chave ({}) que vai servir para posicionar certo tipo de elemento no dicionário e terá um valor que aceita vário tipos como: listas, dicionários, inteiros, strings, dentre outros.

Sua sintaxe básica é: {‘chave’: ‘valor’}. Utiliza-se {} para delimitar o dicionário e a chave é separada do valor por dois pontos (:).

Características

Usamos chaves ({}) para construir nossos dicionários. Neste caso, falamos para o Python, que a chave ({}) possuí o valor (:).

É possível inserir um item em um dicionário com chave e valor simplesmente fazendo a operação direta: chave = {valor:00}

O tipo da chave é imutável, ou seja, uma vez determinado que uma chave seja string, ela não poderá ser alterada.

Os elementos de um dicionário são armazenados de forma não ordenada, pois os elementos de cada chave são exclusivos daquela chave.

Porém, é possível duplicar o mesmo valor á chaves diferentes.

Dicionários também são mutáveis. isso significa que o dicionário pode ser modificado pela referência a uma associação no

lado esquerdo de um comando de atribuição. Podemos redefinir o valor de uma chave ou deletar uma chave valor, com o comando "del".

É possível também deletar um único valor dentro cda chave, com o comando pop(), que irá receber a chave e o valor indicado e irá exclui-lo.

Também é possível juntar dois dicionários com o comando update(), onde dois dicionários diferentes se unirão, subscrevendo o dicionário original.

Sintaxe

Os dicionários são coleções de itens e seus elementos são armazenados de forma não ordenada.

Seus elementos contém uma chave e valor, isto é:

Uma chave que vai servir para indexar (posicionar) determinado elemento no dicionário.

Um valor que contém… Bem, um valor :laughing: Este valor aceita diversos tipos: listas, outros dicionários, inteiros, strings e etc.

Sua sintaxe básica é: {'chave': 'valor'}. Utiliza-se {} para delimitar o dicionário e a chave é separada do valor por dois pontos :.

Exemplo de sua sintaxe:

dicio = {'chave': 'valor'}

print(type(dicio))

Quando utilizar type() e a saída for essa abaixo, pode ter certeza que é um dicionário.

Vantagens e Desvantagens

Vantagens

* Ao procurar um item em uma lista enorme, a busca irá passar por todos os itens dessa lista, já no dicionário basta procurar pela chave, o retorno será muito mais rápido.
* Código legível, ao trabalhar com as chaves junto aos valores no dicionário é adicionado uma camada de documentação ao código, ficando mais fácil depurar.

Desvantagens

Por mais que dicionário em python seja uma das ferramentas mais úteis, ainda existem desvantagens, como:

* Dicionários não são ordenados o que em alguns casos acaba mais atrapalhando que ajudando. Porém em versões recentes do python já temos a resolução desse problema.
* Outra desvantagem que podemos comentar seria a ocupação de espaço extra na memória, já que para cada valor é necessário vincular a uma chave.

Aplicações

O dicionário python serve para gerenciar complexidade ao mesmo tempo que dá flexibilidade,de forma extremamente produtiva, dando um poder muito grande ao programador e também ao usuário (o que pode ser questionável). Em algumas implementações pode realmente ajudar na produtividade, em outras pode prejudicar. Dessa forma o dicionário não serve para sistemas pequenos ou simples demais. Mas se torna mais importante em sistemas que as pessoas desenvolvem hoje em dia, que têm complexidade excessiva e principalmente redundância excessiva. O AD (application dictionary) tem a ver com manter todas informações sobre a aplicação em um só lugar, todas mesmo, até a documentação. Você mantém a documentação dentro da aplicação e "garante" que a documentação muda junto com a aplicação

Logo o Código fica mais gerenciável e o software com mais qualidade, reduzindo o uso de ferramentas.

Desenvolvimento de exemplo utilizando o item apresentado

Perguntar o nome de uma pessoa, idade e seu time de futebol.

torcedor = {} #Declaração da variável torcedor do tipo Dicionário (Dict)

torcedor['Nome'] = input("Digite o seu nome: ") #Através do input inserir o nome da pessoa na chave 'Nome'

torcedor['Idade']= int(input("Digite a sua idade: ")) #Inserir a idade na chave 'Idade'

torcedor['Time'] = input("Digite o seu time do coração: ") #Inserir o time na chave 'Time'

print(torcedor.keys()) #Printar as chaves do dicionário através do método .keys()

print(torcedor.values()) #Printar os valores do dicionário através do método .values()

print(torcedor.items()) #Printar as chaves e valores do dicionário através do método .items()

Conclusão

Concluímos que em comparação com listas, tuplas e outros modos, dicionário possui a característica de poder armazenar vários tipos de dados, entre eles listas, outros dicionários, strings, etc.. então depende do que o usuário quer desenvolver e a maneira que vai manipular.

Referências

<https://pt.stackoverflow.com/questions/347190/quando-e-por-que-usar-dicionários-de-dados>

<https://pythonacademy.com.br/blog/dicts-ou-dicionarios-no-python>

<https://www.digitalvidya.com/blog/python-dictionary/>

<https://penseallen.github.io/PensePython2e/11-dicionarios.html>