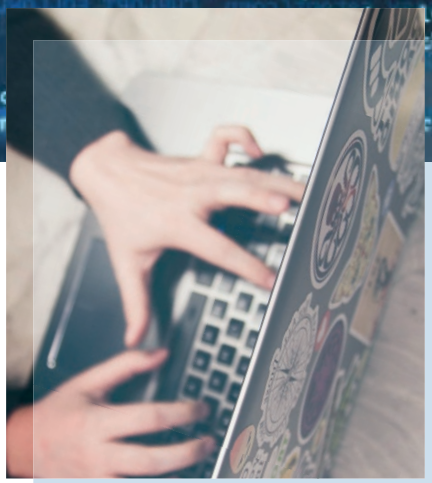
Mapas e dicionários em python

Conceito

Mapas e dicionários são estruturas de dados fundamentais em Python que nos permitem armazenar informações em pares de chave/valor. Essas estruturas são valiosas quando você precisa associar valores a identificadores únicos, como nomes, IDs, ou outras informações distintas.

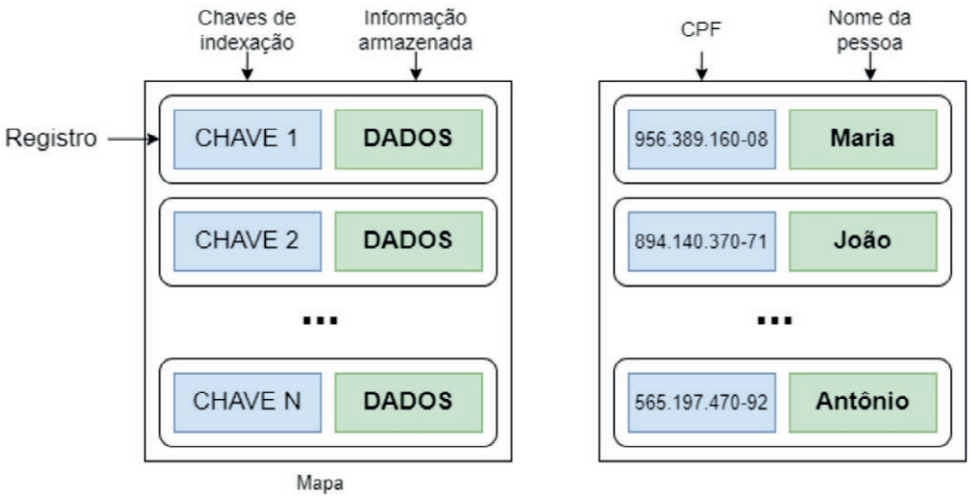


Em Python, essas estruturas são chamadas de dicionários e são representadas por chaves (keys) e valores (values). Cada chave mapeia para um valor correspondente. Aqui estão algumas características essenciais:

Chave Única: Cada chave em um dicionário é única, o que significa que não pode haver duplicatas. Isso garante que cada chave corresponda a um valor exclusivo.

Acesso Eficiente: Os dicionários são altamente eficientes para buscar, adicionar ou remover valores com base em suas chaves. Isso os torna ideais para situações em que você precisa de acesso rápido aos dados.

Essa estrutura permite operações rápidas de acesso, atualização e exclusão de dados, utilizando a chave para acessar diretamente o valor correspondente. Em contraste com listas e vetores, que utilizam números inteiros como índices para acessar informações, nos mapas, o índice de acesso pode ser um valor calculado ou informado, podendo até ser um objeto qualquer. Além disso, o valor da chave em um mapa é único para cada entrada.



Usabilidade

Os mapas são extremamente úteis em situações em que é necessário realizar consultas frequentes a dados e esses dados podem ser referenciados por alguma informação específica. Por exemplo, cada pessoa pode ser indexada pelo seu CPF, que é uma informação única para cada cidadão.

Outra aplicação comum é no processamento de dados de alunos em uma escola. Nesse caso, podemos usar um mapa em que a chave seja o número da sala e o valor seja uma lista de alunos. Dessa forma, podemos acessar com facilidade as informações necessárias a partir do número da sala.

Aplicação

Em Python, a estrutura de dados que implementa mapas é chamada de dicionário (dict). Os principais métodos e operações para um dicionário em Python são:

- dicionario[key] = value: Inclui no dicionário um valor indexado pela chave.
- del dicionario[key]: Remove o dado indexado pela chave no dicionário.
- list(dicionario.keys()): Retorna uma lista de chaves do dicionário.
- dicionario.get(key): Retorna o valor indexado pela chave.
- key in dicionario: Retorna verdadeiro se a chave existe no dicionário.
- len(dicionario) == 0: Retorna verdadeiro se o dicionário está vazio.

Aqui está um exemplo de implementação de um dicionário em Python, com dados de pessoas indexados pelo CPF delas:

```
pessoas = { '12345678900': 'Alice',
'98765432100': 'Bob', '55555555500': 'Charlie'
}
```

Para acessar o nome de uma pessoa pelo CPF, você pode usar a seguinte sintaxe:

```
cpf = '12345678900'

nome = pessoas[cpf]

print(f'O nome da pessoa com CPF {cpf} é
{nome}.')
```

Saída na console:

```
O nome da pessoa com CPF 12345678900 é Alice.

Process finished with exit code 0
```

Outro Exemplo de Dicionário em Python

Para entender melhor, vejamos um exemplo de um dicionário em Python que armazena informações sobre uma pessoa:

```
pessoa = { 'nome': 'João', 'idade': 30,
'cidade': 'São Paulo' }
```

Neste caso, 'nome', 'idade' e 'cidade' são as chaves, e 'João', 30 e 'São Paulo' são os valores correspondentes. Você pode acessar os valores associados a cada chave da seguinte maneira:

```
print(pessoa['nome'])# Saída: 'João'

print(pessoa['idade']) # Saída: 30

print(pessoa['cidade']) # Saída: 'São Paulo'
```

Saída na console

```
João
30
São Paulo

Process finished with exit code 0
```

Adicionar, Atualizar e Remover Itens

Uma das grandes vantagens dos dicionários é a capacidade de adicionar, atualizar e remover itens de forma simples e eficiente. Vejamos como fazer isso:

Adicionar um Item:

O operador [] é usado para adicionar um valor associado a uma chave no dicionário. Se a chave não existir, ela será criada; caso contrário, o valor associado à chave existente será atualizado.
Exemplo:

```
pessoa = { 'nome': 'João', 'idade': 30,
'cidade': 'São Paulo' }

pessoa['profissao'] = 'Engenheiro' #adiciona
novo índice

print(pessoa['nome'])# Saída: 'João'
print(pessoa['idade']) # Saída: 30
print(pessoa['cidade']) # Saída: 'São Paulo'
print(pessoa['profissao']) # Saída: Engenheiro
```

Agora o dicionário pessoa contém uma nova chave 'profissao' com o valor 'Engenheiro'.

```
João
30
São Paulo
Engenheiro

Process finished with exit code 0
```

Atualizar um Item:

```
pessoa['idade'] = 31
```

Isso atualiza o valor da chave 'idade' de 30 para 31.

Remover um Item:

O operador del permite remover um item do dicionário com base em sua chave.

```
del pessoa['cidade']Isso remove a chave
'cidade' e seu valor associado do dicionário.
```

Retornar o valor associado a uma chave:

O método get() permite acessar o valor associado a uma chave no dicionário. Caso a chave não exista, você pode fornecer um valor padrão.
Exemplo:

```
pessoa = { ‘nome’: ‘João’, ‘idade’: 30,
‘cidade’: ‘São Paulo’ }

pessoa[‘profissao’] = ‘Engenheiro’ #adiciona
novo índice

print(pessoa[‘nome’])# Saída: ‘João’
print(pessoa[‘idade’]) # Saída: 30
print(pessoa[‘cidade’]) # Saída: ‘São Paulo’
print(pessoa[‘profissao’]) # Saída: Engenheiro

nome = pessoa.get('nome', 'Não encontrado')
cidade = pessoa.get('cidade', 'Não encontrado')

cpf = pessoa.get('cpf','Não encontrado!') # Não
existe índice cpf

print('Nome: ',nome)

print('cpf: ', cpf)
```

Saída na Console:

```
João
30
São Paulo
Engenheiro
Nome:  João
cpf:  Não encontrado!
Process finished with exit code 0
```

Verificação de Existência

Você pode verificar se uma chave específica existe em um dicionário usando o operador in:

```
if 'idade' in pessoa:

print('A chave "idade" existe no dicionário.')
```

Saída na Console:

```
A chave "idade" existe no dicionário.

Process finished with exit code 0
```

Iterando Sobre um Dicionário

É possível percorrer todos os pares chave/valor em um dicionário utilizando um loop for:

```
for chave, valor in pessoa.items():

print(f'{chave}: {valor}')
```

Isso produzirá a seguinte saída na console:

```
nome: João
idade: 30
cidade: São Paulo
profissao: Engenheiro

Process finished with exit code 0
```

Retornar uma lista de chaves do dicionário:

O método keys() retorna uma lista das chaves presentes no dicionário.
Exemplo:
usando list(dicionário.keys())

```
print(list(pessoa.keys()) )
```

Isso produzirá a seguinte saída na console:

```
['nome', 'idade', 'cidade', 'profissao']

Process finished with exit code 0
```

Verificar se o dicionário está vazio:

Você pode usar a função len() para verificar o tamanho do dicionário. Se o tamanho for 0, o dicionário está vazio.
Exemplo:

```
pessoa = { 'nome': 'João', 'idade': 30,
'cidade': 'São Paulo' }
aluno={}

if len(aluno) == 0:
    print('O dicionário aluno está vazio.')
else:
    print("tem: ", aluno['nome'])

if len(pessoa) == 0:
    print('O dicionário pessoa está vazio.')
else:
    print ("O dicionário Pessoa tem: ",
pessoa['nome'])
```

Isso produzirá a seguinte saída na console:

```
O dicionário aluno está vazio.

O dicionário Pessoa tem:  João

Process finished with exit code 0
```

Estes exemplos ilustram as operações básicas que você pode realizar em um dicionário em Python. Eles são amplamente utilizados para gerenciar dados de maneira eficiente e flexível.

Conclusão

Os dicionários em Python são estruturas de dados poderosas que oferecem uma maneira eficiente de armazenar e acessar informações por meio de chaves exclusivas. Eles são amplamente utilizados na programação Python e desempenham um papel fundamental em muitos aplicativos.