E-BOOK
DO ESTUDANTE

Estruturas de Dados em Python

Juntando dados!

RESILIA

SUMÁRIO

CONTEXTUALIZANDO ---------- 2

LISTAS ---------- 3

TUPLAS ---------- 5

CONJUNTOS ---------- 6

DICIONÁRIOS ---------- 8

PARA REFLETIR ---------- 9

ATIVIDADE ---------- 10

PARA IR ALÉM ---------- 10

RESUMO DE ESTRUTURAS DE DADOS EM PYTHON ------- 11

Estruturas de Dados em Python

Juntando dados!

Em **programação**, uma estrutura de dados é um método para organizar e armazenar dados de forma eficiente. **Python**, uma linguagem de programação popular e de fácil aprendizado, oferece diversas estruturas de dados que permitem ao programador manipular e processar informações. Essas estruturas incluem listas, tuplas, dicionários, conjuntos e outros, que podem ser usados para uma ampla variedade de aplicações.

Conhecer as **estruturas de dados em Python** é essencial para quem deseja se tornar um profissional de programação eficiente e habilidoso. Ao dominar essas estruturas, é possível manipular grandes quantidades de dados com facilidade, realizar operações complexas de forma rápida e desenvolver soluções elegantes para uma variedade de problemas. Por isso, estudar as estruturas de dados em Python é uma tarefa importante e valiosa para qualquer profissional que deseja aprimorar suas habilidades e desenvolver soluções inovadoras.

Aqui, apresentaremos definições, características básicas, usos e implementações das principais estruturas de dados em Python, como listas, tuplas, conjuntos e dicionários.

CONTEXTUALIZANDO

As **estruturas de dados** em Python, como listas, tuplas, conjuntos e dicionários, são de extrema importância no mercado de trabalho de **análise de dados**. Essas estruturas permitem que os profissionais de dados organizem e manipulem grandes quantidades de informações de forma eficiente, garantindo a precisão e a agilidade no processamento dos dados.

As **listas** e **tuplas** são comumente utilizadas para armazenar dados em sequência. Elas são frequentemente usadas em tarefas de análise de dados, permitindo o armazenamento de informações de forma ordenada e a manipulação de dados de forma iterativa. Os **dicionário**s, por sua vez, são especialmente úteis na análise de dados, pois permitem a associação de valores a chaves, possibilitando a organização de dados em formatos mais complexos.

Além disso, os **conjuntos** são importantes para a análise de dados, pois permitem a manipulação de dados de forma rápida e eficiente. Eles possuem métodos que possibilitam a realização de operações como união, intersecção e diferença de conjuntos, sendo úteis para tarefas como a remoção de duplicatas e a identificação de padrões em dados. Em resumo, o conhecimento dessas estruturas de dados em Python é essencial para os profissionais de análise de dados, permitindo que eles desenvolvam soluções mais eficientes e precisas para lidar com grandes volumes de dados.

LISTAS

Em Python, uma **lista** é uma estrutura de dados que armazena vários itens em uma única variável. As listas são muito versáteis e podem ser usadas para armazenar e manipular uma grande variedade de dados, desde números e strings até outras listas e objetos complexos.

Para criar uma lista em Python, basta colocar os itens entre colchetes, separados por vírgulas. Por exemplo, para criar uma lista de números inteiros, podemos fazer o seguinte:

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

Uma vez criada a lista, podemos acessar seus elementos usando índices. Em Python, os índices começam em 0, então o primeiro elemento da lista tem o índice 0, o segundo elemento tem o índice 1, e assim por diante. Por exemplo, para acessar o terceiro elemento da lista "numeros", podemos fazer o seguinte:

```
print(numeros[2])
```

Além de acessar elementos individuais da lista, também podemos fazer operações em toda a lista. Por exemplo, podemos somar todos os elementos de uma lista de números da seguinte forma:

```
soma = sum(numeros)
print(soma)
```

Existem vários métodos disponíveis para manipular listas em Python. Alguns exemplos incluem:

- **append()**: adiciona um elemento ao final da lista;
- **insert()**: adiciona um elemento em uma posição específica da lista;
- **remove()**: remove o primeiro elemento da lista que corresponde a um valor específico;
- **pop():** remove e retorna o elemento na posição especificada da lista.

Por exemplo, podemos usar o método append() para adicionar um novo número à lista "numeros":

```
numeros.append(6)
print(numeros)
```

Em análise de dados, as listas são frequentemente usadas para armazenar coleções de valores de uma determinada variável. Por exemplo, podemos usar uma lista para armazenar as vendas de um produto ao longo do tempo, onde cada elemento da lista representa as vendas em um determinado mês ou trimestre. Em seguida, podemos usar as operações e métodos de lista para analisar esses dados e obter *insights* úteis, como calcular a média de vendas ao longo do tempo ou identificar os meses ou trimestres com as vendas mais altas ou mais baixas.

Em resumo, as listas são uma parte fundamental da linguagem de programação Python e são amplamente usadas em análise de dados para armazenar e manipular coleções de valores. Ao aprender a trabalhar com listas em Python, você pode expandir sua capacidade de manipular dados e criar soluções mais sofisticadas para problemas de análise de dados.

Agora vamos conhecer uma excelente opção para estruturar dados em uma aplicação; seu uso oferece segurança, uma vez que não podem ser modificadas: as Tuplas.

TUPLAS

Na linguagem Python, uma **tupla** é outra estrutura de dados semelhante a uma lista, mas com uma importante diferença: as tuplas são **imutáveis**, ou seja, uma vez criadas, não podem ser modificadas. Isso significa que, ao contrário das listas, não podemos adicionar, remover ou modificar elementos de uma tupla depois que ela foi criada.

Para criar uma tupla em Python, podemos usar parênteses em vez de colchetes. Por exemplo, podemos criar uma tupla de números inteiros da seguinte forma:

```
numeros = (1, 2, 3, 4, 5)
```

Uma vez criada a tupla, podemos acessar seus elementos usando índices, da mesma forma que em uma lista. No entanto, como as tuplas são imutáveis, não podemos modificar esses elementos. Por exemplo, a seguinte linha de código resultará em um erro de atribuição:

```
numeros[2] = 10
```

Apesar de serem imutáveis, as tuplas ainda podem ser muito úteis na análise de dados. Por exemplo, podemos usar uma tupla para representar as coordenadas geográficas de um local, onde o primeiro elemento da tupla é a latitude e o segundo elemento é a longitude. Como as coordenadas de um local não mudam, faz sentido usar uma tupla em vez de uma lista, já que não precisaremos modificar esses valores posteriormente.

Além de acessar elementos individuais da tupla, também podemos fazer operações em toda a tupla. Por exemplo, podemos somar todos os elementos de uma tupla de números da seguinte forma:

```
soma = sum(numeros)
print(soma)
```

Embora as tuplas não tenham tantos métodos disponíveis quanto as listas, ainda existem alguns métodos úteis que podemos usar. Por exemplo, o método index() retorna o índice do primeiro elemento na tupla que corresponde a um valor específico, enquanto o método count() conta o número de ocorrências de um determinado valor na tupla:

```
numeros = (1, 2, 3, 4, 5, 3)
print(numeros.index(3))
print(numeros.count(3))
```

Em resumo, as tuplas são uma estrutura de dados imutável em Python que podem ser úteis na análise de dados para representar valores que não mudam, como coordenadas geográficas ou informações fixas sobre um objeto.

Embora as tuplas não possam ser modificadas depois de criadas, ainda podemos acessar seus elementos e fazer operações neles.

Nas situações em que necessitamos usar elementos únicos e imutáveis e realizar operações como união, intersecção e diferença, devemos usar os **conjuntos**. Vamos conferir essas estruturas agora.

CONJUNTOS

Um **conjunto** (set) é uma coleção não ordenada de elementos únicos e imutáveis. Isso significa que, ao contrário de uma lista ou tupla, um conjunto não permite elementos duplicados e não preserva a ordem em que os elementos foram adicionados.

Para criar um conjunto em Python, podemos usar chaves ({}) ou a função set(). Por exemplo:

```
frutas = {'maçã', 'banana', 'laranja'}
ou
frutas = set(['maçã', 'banana', 'laranja'])
```

As operações comuns que podemos fazer nos conjuntos incluem união, interseção e diferença. A **união** de dois conjuntos inclui todos os elementos presentes em ambos os conjuntos, enquanto a **interseção** inclui apenas os elementos que estão presentes em ambos os conjuntos. A **diferença** inclui apenas os elementos que estão presentes em um conjunto, mas não no outro:

```
conjunto1 = {1, 2, 3}
conjunto2 = {2, 3, 4}
uniao = conjunto1.union(conjunto2)#Saída: {1, 2, 3, 4}
intersecao = conjunto1.intersection(conjunto2) #Saída: {2, 3}
diferenca = conjunto1.difference(conjunto2) #Saída: {1}
```

Além dessas operações, também podemos verificar se um elemento está presente em um conjunto usando o operador in, e adicionar ou remover elementos do conjunto usando os métodos **add**() e **remove**(), respectivamente.

```
frutas = {'maçã', 'banana', 'laranja'}
print('maçã' in frutas)
frutas.add('uva')
print(frutas)
frutas.remove('banana')
print(frutas)
```

Os conjuntos também podem ser usados em análise de dados para remover duplicatas de uma lista de valores ou para realizar operações de conjunto em conjunto de dados. Por exemplo, podemos usar um conjunto para obter uma lista única de valores de uma coluna em um conjunto de dados.

Em resumo, os conjuntos em Python são úteis para trabalhar com coleções de elementos exclusivos e imutáveis. Eles fornecem operações convenientes para unir, interseccionar e subtrair conjuntos e podem ser usados para remover duplicatas de uma lista de valores ou realizar operações de conjunto em conjunto de dados na análise de dados.

Vimos que conjuntos são uma estrutura bastante interessante quando trabalhamos com dados imutáveis. Porém, caso desejemos usar uma forma diferente de organizar os dados, em vez dos índices, podemos usar os Dicionários. Vamos conferir

DICIONÁRIOS

Um **dicionário** (dictionary) é uma estrutura de dados que mapeia uma chave (key) a um valor (value). Os dicionários são úteis quando precisamos associar um valor a uma chave e queremos poder acessar esse valor de forma rápida e eficiente.

Para **criar** um dicionário em Python, usamos chaves ({}) e especificamos as chaves e valores separados por dois pontos (:). Por exemplo:

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

Podemos **acessar** o valor associado a uma chave usando o operador de colchetes ([]). Por exemplo:

```
print(dicionario['nome'])
```

Também podemos **adicionar** ou **atualizar** um valor associado a uma chave usando o operador de colchetes. Por exemplo:

```
dicionario['idade'] = 31
dicionario['email'] = 'joao@gmail.com'
```

As operações comuns que podemos fazer em dicionários incluem obter as chaves, os valores ou os itens (chave-valor) do dicionário usando os métodos **keys**(), **values**() e **items**(), respectivamente. Também podemos verificar se uma chave está presente no dicionário usando o operador in.

```
print(dicionario.keys())
print(dicionario.values())
print(dicionario.items())
print('cidade' in dicionario)
```

Além dessas operações, também podemos remover um item do dicionário, usando o método pop(), ou todos os itens do dicionário, usando o método clear():

```
dicionario.pop('idade')
print(dicionario)
dicionario.clear()
print(dicionario)
```

Os dicionários são amplamente utilizados em análise de dados para armazenar informações em forma de pares chave-valor, como dados em tabelas. Por exemplo, podemos usar um dicionário para armazenar informações sobre um cliente, como nome, idade, endereço e histórico de compras.

Em resumo, os dicionários em Python são úteis para associar valores a chaves e acessar esses valores de forma rápida e eficiente. Eles fornecem operações convenientes para obter as chaves, os valores ou os itens do dicionário e para adicionar, atualizar ou remover itens do dicionário. Os dicionários são amplamente utilizados em análise de dados para armazenar informações em forma de pares chave-valor, como dados em tabelas.

Para refletir

- Quais são as principais diferenças entre listas, tuplas, conjuntos e dicionários em Python, e quando devo usar cada uma dessas estruturas de dados em um projeto de análise de dados?
- Como posso manipular eficientemente grandes conjuntos de dados em Python usando listas, tuplas, conjuntos e dicionários?
- Qual é a melhor forma de armazenar dados não estruturados em Python para que eu possa acessá-los e analisá-los facilmente usando as estruturas de dados disponíveis?

Atividade: Contando elementos em uma lista

Nesta atividade, você irá criar um programa em Python que recebe uma lista e imprime um dicionário contendo a contagem de cada elemento presente na lista. Crie um programa que recebe uma lista e imprime um dicionário com a contagem de cada elemento presente na lista.

Passo a passo:

1. Crie uma lista com alguns elementos;
2. Crie um conjunto a partir da lista, para eliminar possíveis elementos repetidos;
3. Crie um dicionário vazio para armazenar a contagem de cada elemento;
4. Percorra o conjunto criado no passo 2 com um loop "for";
5. Para cada elemento no conjunto, use o método "count" da lista original para contar quantas vezes ele aparece na lista;
6. Armazene a contagem no dicionário criado no passo 3, usando o elemento como chave e a contagem como valor;
7. Imprima o dicionário com a contagem de elementos.

Para ir além

- Vimos que uma lista em Python é uma coleção ordenada e mutável de valores, onde cada valor é identificado por um índice. É denotada por colchetes [ ]. No site abaixo, você encontra mais exemplos de implementação dessa estrutura.
  <https://www.devmedia.com.br/como-trabalhar-com-listas-em-python/37460>

- Vimos que uma tupla em Python é uma coleção ordenada e imutável de valores, onde cada valor é identificado por um índice. É denotada por parênteses ( ). No site abaixo, você encontra mais exemplos de códigos de tuplas em Python.
  <https://algoritmosempython.com.br/cursos/programacao-python/tuplas/>

- Vimos que um conjunto em Python é uma coleção não ordenada e mutável de valores únicos e não duplicados. É denotado por chaves { }. No site abaixo, você encontra exemplos mais diversificados de uso de dicionários em Python.
  <https://pense-python.caravela.club/19-extra/05-conjuntos.html>

- Vimos que um dicionário em Python é uma coleção não ordenada e mutável de pares chave-valor, onde cada valor é identificado por uma chave única. É denotado por chaves { }. No site abaixo você encontra exemplos de diversificados de dicionários em python.
  <https://docs.python.org/pt-br/3/c-api/dict.html>

RESUMO DE ESTRUTURAS DE DADOS EM PYTHON

Lista
Em Python, uma lista é uma estrutura de dados que pode armazenar vários itens em uma única variável.

Criação de Lista

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

Adição de Elementos

```
numeros.append(6)
print(numeros)
```

Tupla
Uma tupla é outra estrutura de dados semelhante a uma lista, mas com uma diferença: as tuplas são imutáveis.

Criação de uma Tupla

```
numeros = (1, 2, 3, 4, 5)
```

Somando Elementos

```
soma = sum(numeros)
print(soma)
```

Acessando Index e Contando Elementos

```
numeros = (1, 2, 3, 4, 5, 3)
print(numeros.index(3))
print(numeros.count(3))
```

Conjunto
Um conjunto (set) é uma coleção não ordenada de elementos únicos e imutáveis.

Criação de Conjuntos

```
frutas = {'maçã', 'banana', 'laranja'}
#ou
frutas = set(['maçã', 'banana', 'laranja'])
```

Adição e Remoção de Elementos

```
frutas = {'maçã', 'banana', 'laranja'}
print('maçã' in frutas)
frutas.add('uva')
print(frutas)
frutas.remove('banana')
print(frutas)
```

Dicionário
Um dicionário (dictionary) é uma estrutura de dados que mapeia uma chave (key) a um valor (value).

Criação de Dicionário

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

Acessando Elementos

```
print(dicionario['nome'])
```

Adicionando ou Atualizando Elementos

```
dicionario['idade'] = 31
dicionario['email'] = 'joao@gmail.com'
```

Até a próxima e
#confianoprocesso

RESILIA