Educação Profissional Paulista

Técnico em Ciência de Dados

Secretaria da Educação

SÃO PAULO
GOVERNO DO ESTADO

Estrutura de controle de fluxo

Compreensão de listas

Aula 1

Código da aula: [DADOS]ANO1C2B2S13A1

Exposição

Objetivos da Aula

Revisar o conceito de listas e introduzir o conceito de compreensão de listas.

Competências da Unidade (Técnicas e Socioemocionais)

- Ser proficiente em linguagens de programação para manipular e analisar grandes conjuntos de dados.
- Usar técnicas para explorar e analisar dados, aplicar modelos estatísticos, identificar padrões, realizar inferências e tomar decisões fundamentadas em evidências.
- Colaborar efetivamente com outros profissionais, como cientistas de dados e engenheiros de dados; trabalhar em equipes multifuncionais, colaborando com colegas, gestores e clientes.

Recursos Didáticos

- Recurso audiovisual para exibição de vídeos e imagens.
- Acesso ao laboratório de informática e/ou à internet.
- Software Anaconda/Jupyter Notebook instalados ou similar.

Duração da Aula

50 minutos

Motivação: transação financeira

Você trabalha em uma empresa financeira e observou o código de um programador que criou um programa em Python para analisar as transações financeiras.

Sabe-se que transações positivas são depósitos (valores positivos) e valores negativos são retiradas de dinheiro.

Analise o código no próximo slide e responda:

1. Você sabe se o programador fez o código de acordo com o problema pedido?

2. O que você identificou nas linhas 2 e 8?

Momento de **reflexão**

Exposição

Motivação: transação financeira

```python
def calcular_total_transacoes_positivas(transacoes):
    transacoes_positivas = [valor for valor in transacoes if valor > 0]
    total_transacoes_positivas = sum(transacoes_positivas)
    quantidade_transacoes_positivas = len(transacoes_positivas)
    return total_transacoes_positivas, quantidade_transacoes_positivas

def calcular_total_transacoes_negativas(transacoes):
    transacoes_negativas = [valor for valor in transacoes if valor < 0]
    total_transacoes_negativas = sum(transacoes_negativas)
    return total_transacoes_negativas

# Lista de transações financeiras de um cliente
transacoes = [100, -50, 200, -20, 150, -30, 180]

# Chamando as funções para calcular os totais
total_positivas, quantidade_positivas = calcular_total_transacoes_positivas(transacoes)
total_negativas = calcular_total_transacoes_negativas(transacoes)

# Impressão dos resultados
print("Total das transações positivas:", total_positivas)
print("Quantidade de transações positivas:", quantidade_positivas)
print("Total das transações negativas:", total_negativas)
```

Elaborado especialmente para o curso com a ferramenta Jupyter Notebook.

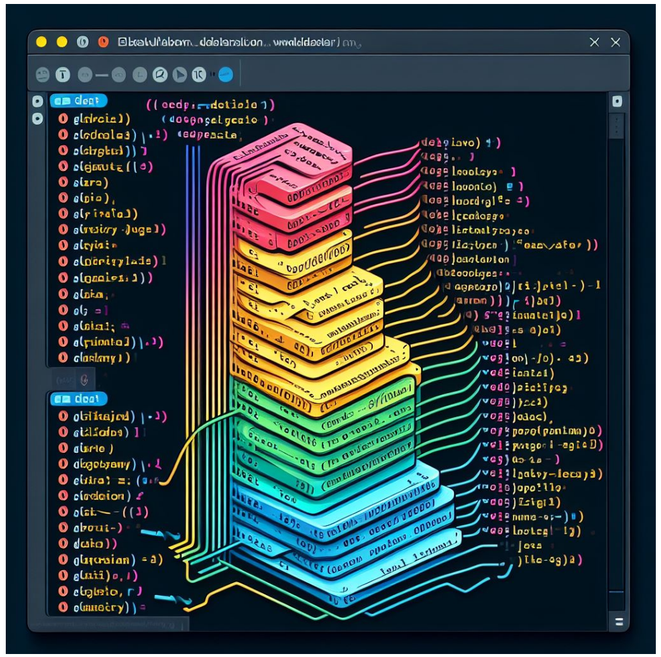
Exposição

Relembrando

	Listas
Definição	• Listas são estruturas de dados em Python que permitem armazenar uma coleção ordenada de elementos. • São mutáveis, o que significa que você pode adicionar, remover ou modificar elementos.
Sintaxe básica	• Declaradas usando colchetes []. • Exemplo: lista = [1, 2, 3, 4, 5].
Acesso aos elementos	• Índices começam em 0. • Exemplo: primeiro_elemento = lista[0].
Operações comuns	• Adição de elementos: lista.append(valor). • Remoção de elementos: lista.remove(valor) ou del lista[indice]. • Tamanho da lista: len(lista).
Versatilidade	• Aceita diferentes tipos de dados. • Pode conter listas dentro de listas (listas aninhadas).
Iteração e manipulação	• Fornece métodos poderosos para percorrer e manipular elementos. • Útil em várias operações, como filtragem, transformação e redução de dados.

Exposição

Compreensão de listas



Compreensão de lista ou ***List Comprehension*** em Python é uma construção sintática que permite criar listas de maneira concisa e eficiente.

Curiosidade:

Essa técnica é uma forma compacta de gerar, filtrar e transformar elementos de uma lista.

Elaborado especialmente para o curso com a ferramenta Microsoft Copilot.

Exposição

Compreensão de listas

Sua sintaxe básica é:

```
[expressao for elemento in lista_original if condicao]
```

Elaborado especialmente para o curso com a ferramenta Jupyter Notebook.

Tome nota:

- **expressao:** A expressão a ser aplicada a cada elemento da lista.
- **elemento:** A variável que representa cada elemento da lista original.
- **lista_original:** A lista da qual os elementos são retirados.
- **condicao (opcional):** Uma condição que determina se o elemento será incluído na nova lista.

Exposição

Exemplo:

1. Criação simples

```
1  quadrados = [x**2 for x in range(1, 6)]
2
3  print(quadrados)
```

```
[1, 4, 9, 16, 25]
```

```
[expressao for elemento in lista_original if condicao]
```

Elaborado especialmente para o curso com a ferramenta Jupyter Notebook.

Tome nota:

expressao: x**2 – A expressão que eleva cada elemento x ao quadrado.

elemento: for x in range(1, 6) - A variável x representa cada elemento na lista original, que varia de 1 a 5.

lista_original: range(1, 6) – A lista original da qual os elementos são retirados.

condicao: Não há condição neste exemplo.

Exposição

Exemplo:

Vamos reescrever o código e comparar.

```
1  quadrados = [x**2 for x in range(1, 6)]
2
3  print(quadrados)
```

```
[1, 4, 9, 16, 25]
```

```
1  quadrados = []
2  for x in range(1, 6):
3      quadrados.append(x**2)
4
5  print(quadrados)
```

```
[1, 4, 9, 16, 25]
```

Elaborado especialmente para o curso com a ferramenta Jupyter Notebook.

Exposição

Exemplo:

2. Filtragem de elementos

```
1  numeros_pares = [x for x in range(10) if x % 2 == 0]
2  print(numeros_pares)
```

```
[0, 2, 4, 6, 8]
```

```
[expressao for elemento in lista_original if condicao]
```

Elaborado especialmente para o curso com a ferramenta Jupyter Notebook.

Tome nota:

expressao: x – A expressão que representa cada elemento x.

elemento: for x in range(10) – A variável x representa cada elemento na lista original, variando de 0 a 9.

lista_original: range(10) – A lista original da qual os elementos são retirados.

condicao: if x % 2 == 0 – A condição que verifica se o número é par.

Exposição

Exemplo:

Vamos reescrever o código e comparar:

```python
numeros_pares = [x for x in range(10) if x % 2 == 0]
print(numeros_pares)
```

```
[0, 2, 4, 6, 8]
```

```python
numeros_pares = []
for x in range(10):
    if x % 2 == 0:
        numeros_pares.append(x)

print(numeros_pares)
```

```
[0, 2, 4, 6, 8]
```

Elaborado especialmente para o curso com a ferramenta Jupyter Notebook.

Exposição

Exemplo:

3. Transformação de elementos

```python
palavras = ['python', 'é', 'incrível']
maiusculas = [palavra.upper() for palavra in palavras]

print(maiusculas)
```

```
['PYTHON', 'É', 'INCRÍVEL']
```

```
[expressao for elemento in lista_original if condicao]
```

Elaborado especialmente para o curso com a ferramenta Jupyter Notebook.

Tome nota:

expressao: palavra.upper() – A expressão que converte cada palavra para maiúsculas.

elemento: for palavra in palavras – A variável palavra representa cada elemento na lista original palavras.

lista_original: palavras – A lista original da qual os elementos são retirados.

condicao: Não há condição neste exemplo.

Exposição

Exemplo:

Vamos reescrever o código e comparar:

```python
palavras = ['python', 'é', 'incrível']
maiusculas = [palavra.upper() for palavra in palavras]

print(maiusculas)
```

```
['PYTHON', 'É', 'INCRÍVEL']
```

```python
palavras = ['python', 'é', 'incrível']
maiusculas = []
for palavra in palavras:
    maiusculas.append(palavra.upper())

print(maiusculas)
```

```
['PYTHON', 'É', 'INCRÍVEL']
```

Elaborado especialmente para o curso com a ferramenta Jupyter Notebook.

Exposição

Exemplo:

4. Expressões complexas

```
pares_soma = [(x, y, x + y) for x in range(2) for y in range(2, 4) if (x + y) % 2 == 0]
print(pares_soma)
```

```
[(0, 2, 2), (1, 3, 4)]
```

```
[expressao for elemento in lista_original if condicao]
```

Elaborado especialmente para o curso com a ferramenta Jupyter Notebook.

Tome nota:

expressao: (x, y, x + y) – A expressão que cria uma tupla com x, y e a soma deles.

elemento: for x in range(2) for y in range(2, 4) – As variáveis x e y representam os elementos nas listas originais.

lista_original: Não há lista original neste caso; são usadas duas iterações.

condicao: if (x + y) % 2 == 0 – A condição que verifica se a soma de x e y é par.

Exposição

Exemplo:

4. Expressões complexas

```
pares_soma = [(x, y, x + y) for x in range(2) for y in range(2, 4) if (x + y) % 2 == 0]
print(pares_soma)
```

```
[(0, 2, 2), (1, 3, 4)]
```

```
pares_soma = []
for x in range(2):
    for y in range(2, 4):
        if (x + y) % 2 == 0:
            pares_soma.append((x, y, x + y))

print(pares_soma)
```

```
[(0, 2, 2), (1, 3, 4)]
```

Elaborado especialmente para o curso com a ferramenta Jupyter Notebook.

Exposição

© Getty Images

O que nós **aprendemos hoje?**

Hoje desenvolvemos:

1. O conhecimento sobre a criação de listas de forma eficiente, aplicando o conceito *List Comprehension* (compreensão de listas);

2. A revisão de conteúdo sobre listas;

3. A análise comparativa de vantagens e desvantagens na aplicação de compreensão de listas.

Saiba mais

Você lembra como fazer uma cópia de uma lista em Python?

FELIPE, A. *Python*: Append ou Extend? Alura, 23 jan. 2017. Adicionando elementos na lista. Disponível em: https://www.alura.com.br/artigos/adicionando-elementos-na-lista-do-python-append-ou-extend. Acesso em: 24 mar. 2024.

ORESTES, Y. *Como fazer uma cópia de uma lista no Python*. Alura, 6 set. 2018. Disponível em: https://www.alura.com.br/artigos/como-fazer-copia-de-lista-python. Acesso em: 24 mar. 2024.

Referências da aula

MENEZES, N. N. C. *Introdução à programação com Python*: algoritmos e lógica de programação para iniciantes. São Paulo: Novatec, 2019.

Educação Profissional Paulista

Técnico em Ciência de Dados

Secretaria da Educação

SÃO PAULO
GOVERNO DO ESTADO