

Olá, desejamos boas-vindas ao módulo **Lógica de programação com Python** do curso **Profissão: Cientista de Dados**.

O Módulo 5 do curso oferece uma introdução abrangente aos conceitos fundamentais da programação em Python, utilizando o Google Colab como principal ferramenta de desenvolvimento. O módulo é estruturado em várias aulas, cada uma focada em diferentes aspectos essenciais para iniciantes:

Introdução ao Python e Google Colab

O Google Colab é uma plataforma web que permite a execução de códigos Python em um ambiente virtual. VOcê aprenderá a acessar o Colab, criar e manipular notebooks, e executar seu primeiro código Python utilizando a função `print`.

Boas Práticas e Operadores

Você analisará a importância de documentar o código com comentários e aprenderá os operadores aritméticos, relacionais e lógicos. A prática contínua e a documentação são essenciais para a manutenção e clareza do código.

Estruturas Condicionais

Você aprenderá estruturas condicionais (`if`, `else`, `elif`) e operadores lógicos. Também verá a importância da indentação e dos testes de mesa para verificar o comportamento do código em diferentes cenários.

Este módulo fornece uma base sólida em Python e Google Colab, cobrindo desde a introdução e boas práticas até estruturas de dados e interação com o usuário. Vamos juntos avançar no seu caminho para se tornar um analista de dados de sucesso!

Comentários

Importante para **documentar** o código

Boas práticas

- Não comentar código errado
- Não comentar código testes (produção)

Comentário de uma linha ()

Usar para comentar uma linha do código

Comentário de múltiplas linhas

"""

Usar para o cabeçalho do código

Usar 3 aspas

"""

Operadores

Operadores Aritméticos

Operador	Símbolo	Exemplo
Adição	+	5 + 2
Subtração	-	5 - 2
Multiplicação	*	5 * 2
Divisão	/	5 / 2
Divisão inteira	//	5 // 2
Resto da divisão	%	5 % 2
Potenciação	**	5 ** 2

Operadores

Operadores Relacionais

Comparação	Símbolo	Exemplo
Menor	<	1 < 2
Menor ou igual	<=	1 <= 10
Maior	>	2 > 1
Maior ou igual	>=	2 >= 2
Igual	==	5 == 5
Diferente	!=	5 != 8

Operadores

Operadores Lógicos

<ul style="list-style-type: none">AndOrNot	Exemplo	Lógica	Resultado
	2 > 1 and 5 < 8	True and True	True
	2 > 1 and 5 > 8	True and False	False
	2 < 1 and 5 < 8	False and True	False
	2 < 1 and 5 > 8	False and False	False
	2 > 1 or 5 < 8	True or True	True
	2 > 1 or 5 > 8	True or False	True
	2 < 1 or 5 < 8	False or True	True
	2 < 1 or 5 > 8	False or False	False
	not 2 > 1	True	False
	not 1 < 2	False	True

Variáveis

Variáveis estão alocadas em espaços na memória

Ao criar uma variável

- Python aloca um espaço na memória
- Referência utilizando o nome atribuído à variável pelo programador no momento de sua criação
- Exemplo

""" Nome da variável é "x" que armazena o valor 10

em um espaço na memória """

```
>>> x = 10
```

```
>>> print(x)
```

```
10
```

Regras de variáveis

Nome da variável

- Começar com letra (a-z, A-Z) ou sublinhado (_)
- São Case Sensitive (Diferença entre letras maiúscula e minúscula)
- Exemplo

```
>>> A != a
```

Regras de variáveis

Boas práticas para o nome da variável

- Usar nomes que definam o que ela faz no programa
- Não usar nomes longos

Exemplo de soma de 2 números:

Recomendado	Não recomendado
soma = 2 + 5	a = 2 + 5
soma_total = 2 + 5	soma_dois_números_inteiros = 2 + 5
somaTotal = 2 + 5	somaDeDoisNumerosInteiros = 2 + 5

Tipos de variáveis

Python é uma linguagem de alto nível

- Não é necessário especificar o tipo de dados quando declarar a variável
- Python define o tipo de dados na declaração

Tipo	Exemplo
Int (Inteiro)	num = 10
Float (Real)	num = 10.5
String (Texto)	nome = "Rodrigo"

For

Estrutura de repetição

Executar comandos, enquanto percorre os itens de uma coleção

Sintaxe:

```
>>> for <variável> in <coleção>:  
>>>     <<comandos>>
```

Formas de repetição

- Simple
- Range
- Enumerate

For – Range

Exemplo

```
>>> for contador in range(0,4):  
>>>     print(contador)  
0  
1  
2  
3
```

```
>>> for contador in range(0,4):  
>>>     print(contador)  
>>>     if(contador == 2 ):  
>>>         exit()  
0  
1  
2
```

For – Range

Exemplo

```
lista1 = ["teste1","teste2"]
```

Exemplo de for simples

```
>>> for val in lista1:  
>>>     print(val)  
teste1  
teste2
```

Exemplo de for com Enumerate

```
>>> for i, val in enumerate(lista1):  
>>>     print(i, val)  
0, teste1  
1, teste2
```

For – Operadores Compostos

Operadores Compostos

- Muito utilizado em laços de repetição
- Código mais reduzido

Operador	Símbolo	Exemplo	Equivalente
Mais igual	+=	num += 1	num = num + 1
Menos igual	-=	num -= 2	num = num – 1
Vezez igual	*=	num *= 5	num = num * 5
Dividido igual	/=	num /= 2	num = num / 2
Módulo igual	%=	num %= 2	num = num % 2

For –Operadores Compostos

Exemplo

lista2 = [2, 3, 4]

```
>>> for val in lista2:
>>>     val += 1
>>>     print(val)
3
4
5
```

```
>>> for val in lista2:
>>>     val *= 2
>>>     print(val)
4
6
8
```


While

Estrutura de repetição

Executar comandos enquanto uma condição é verdadeira

Sintaxe:

```
>>> while <condição>:
```

```
>>>     <<comandos>>
```

```
>>> while <condição>:
```

```
>>>     <<comandos>>
```

```
>>> else:
```

```
>>>     <<comandos>>
```

While

Exemplo

```
# Exemplo de While
```

```
>>> val = 1
```

```
>>> while (val < 6):
```

```
>>>     val *=2
```

```
>>>     print(val)
```

```
2
```

```
4
```

```
8
```

```
# Exemplo de While com else
```

```
>>> while val < 6:
```

```
>>>     val *=2
```

```
>>> else:
```

```
>>>     print("Valor de val é: ", val)
```

```
Valor de val é: 8
```

- **Listas** são estruturas de dados que permitem **armazenar coleções de elementos**. Esses elementos podem ser números, textos, outros tipos de dados ou até mesmo outras listas. Elas são uma maneira flexível e poderosa de organizar dados em um programa.

Veja um exemplo simples usando Python:

```
frutas = ["maçã", "banana", "laranja", "uva"]
```

Neste exemplo, "frutas" é uma lista que contém quatro elementos, cada um representando um tipo de fruta.

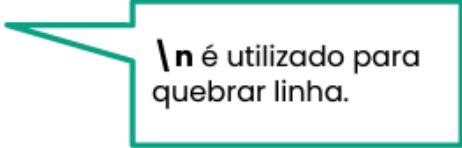
- Você pode acessar esses elementos individualmente usando sua posição na lista. Em Python, a contagem começa em 0, então para acessar a primeira fruta (maçã), você usaria `frutas[0]`.

Veja porque as listas são importantes na lógica de programação:

- **Armazenamento de múltiplos valores:** Listas permitem que você agrupe vários valores relacionados em uma única estrutura de dados. Isso é útil quando você precisa lidar com coleções de informações, como uma lista de nomes, números, produtos, etc.
- **Acesso e manipulação de elementos:** Você pode adicionar, remover, substituir e acessar elementos individualmente em uma lista, o que é fundamental para muitas tarefas de programação.
- **Iteração:** Você pode usar loops para percorrer todos os elementos de uma lista, o que é útil quando precisa realizar a mesma operação em cada item da lista.
- **Ordenação:** Você pode classificar os elementos em uma lista, o que é útil para organizar dados em ordem crescente ou decrescente.
- **Listas complexas:** Listas podem conter outras listas, permitindo a criação de estruturas de dados complexas para lidar com dados mais complexos.

Por exemplo, imagine que você está começando a coletar dados de feedback de clientes e deseja organizar esses dados em uma lista simples e um dicionário. Veja como seu código pode ficar:

```
# Organizando dados de feedback de clientes
# Lista de feedbacks
feedbacks = ["Ótimo serviço!", "Produto excelente", "Atendimento rápido"]
# Dicionário com informações de um cliente
cliente = {
    "nome": "Maria",
    "idade": 28,
    "feedback": "Muito satisfeita"
}
# Imprimir os feedbacks da lista
print("Feedbacks dos clientes:")
for feedback in feedbacks:
    print("- " + feedback)
# Imprimir informações do cliente
print("\nInformações do cliente:")
print("Nome:", cliente["nome"])
print("Idade:", cliente["idade"])
print("Feedback:", cliente["feedback"])
```



`\n` é utilizado para quebrar linha.

12

Documentação do Código

Documentar o código com comentários é essencial para a manutenção e clareza. Comentários ajudam outros desenvolvedores (e você mesmo no futuro) a entender a lógica e o propósito do código. Use comentários para explicar partes complexas ou importantes do código.

Uso de Nomes Descritivos para Variáveis

Utilize nomes descritivos para variáveis que reflitam seu propósito. Isso facilita a leitura e compreensão do código. Evite nomes genéricos como `x` ou `temp` a menos que sejam usados em contextos muito claros.

Interação com o Usuário

Utilize os comandos `print` e `input` para interagir com o usuário. Certifique-se de fornecer mensagens claras e informativas ao solicitar entradas e ao exibir resultados.

Conversão de Tipos

Ao trabalhar com entradas do usuário, é importante converter os tipos de dados conforme necessário. Por exemplo, se você espera um número, converta a entrada de string para um tipo numérico usando `int()` ou `float()`.

Exemplo de utilização no mercado de trabalho:

Automação de Tarefas: Analistas de dados frequentemente escrevem scripts em Python para automatizar tarefas repetitivas, como a limpeza de dados ou a geração de relatórios. A documentação e o uso de nomes descritivos são cruciais para a manutenção desses scripts.

Interação com Usuários: Em projetos onde é necessário coletar dados de usuários, como em pesquisas ou formulários, a interação clara e a conversão correta de tipos garantem que os dados sejam coletados e processados corretamente.

Exemplo de código executável (em Python) que ilustra boas práticas:

```
# Solicita o nome do usuário e o exibe
nome_usuario = input("Por favor, insira seu nome: ")
print(f"Olá, {nome_usuario}! Bem-vindo ao nosso sistema.")
```

```
# Solicita a idade do usuário e converte para inteiro
idade_usuario = int(input("Por favor, insira sua idade: "))

# Verifica se o usuário é maior de idade
if idade_usuario >= 18:
    print("Você é maior de idade.")
else:
    print("Você é menor de idade.")
```

Vamos discutir o código passo-a-passo:

`input("Por favor, insira seu nome: ")`: Solicita ao usuário que insira seu nome e armazena a entrada na variável `nome_usuario`.

`print(f"Olá, {nome_usuario}! Bem-vindo ao nosso sistema.")`: Exibe uma mensagem de boas-vindas ao usuário utilizando a variável `nome_usuario`.

`int(input("Por favor, insira sua idade: "))`: Solicita ao usuário que insira sua idade, converte a entrada de string para inteiro e armazena na variável `idade_usuario`.

`if idade_usuario >= 18:`: Verifica se o usuário é maior de idade.

`print("Você é maior de idade.")`: Exibe uma mensagem se o usuário for maior de idade.

`else:`: Bloco de código que será executado se a condição anterior não for atendida.

`print("Você é menor de idade.")`: Exibe uma mensagem se o usuário for menor de idade.

Aprenda a identificar e corrigir os erros mais comuns e torne seu código mais eficiente e confiável.

01

Esquecer de Indentar o Código Corretamente.

Descrição: A indentação é crucial em Python para definir blocos de código. Esquecer de indentar corretamente pode levar a erros de sintaxe.

Tipo de Erro: Erro de sintaxe.

Exemplo:

```
if True:  
print("Indentação incorreta")
```

Correção:

```
if True:  
    print("Indentação correta")
```

Passo 1: Verifique se todos os blocos de código estão corretamente indentados.

Passo 2: Use espaços ou tabulações de forma consistente.

Passo 3: Utilize um editor de código que destaque a indentação.

02

Usar o Operador de Atribuição em vez do Operador de Igualdade.

Descrição: Confundir o operador de atribuição (`=`) com o operador de igualdade (`==`) pode levar a resultados inesperados.

Tipo de Erro: Erro lógico.

Exemplo:

```
x = 10
if x = 10:
    print("Erro de operador")
```

Correção:

```
x = 10
if x == 10:
    print("Operador correto")
```


Passo 1: Revise o código para garantir que `=` é usado para atribuição e `==` para comparação.

Passo 2: Teste o código para verificar se as comparações estão corretas.

03

Não Converter Tipos de Dados ao Usar `input`.

Descrição: O comando `input` retorna uma string. Esquecer de converter para o tipo de dado apropriado pode causar erros.

Tipo de Erro: Erro de tempo de execução.

Exemplo:

```
idade = input("Digite sua idade: ")  
if idade > 18:  
    print("Maior de idade")
```

Correção:

```
idade = int(input("Digite sua idade: "))  
if idade > 18:  
    print("Maior de idade")
```

Passo 1: Identifique onde o ``input`` é usado para capturar dados do usuário.

Passo 2: Converta a entrada para o tipo de dado apropriado (por exemplo, ``int`` ou ``float``).

04

Esquecer de Inicializar Variáveis.

Descrição: Usar variáveis antes de inicializá-las pode levar a erros de tempo de execução.

Tipo de Erro: Erro de tempo de execução.

Exemplo:

```
total = total + 1  
print(total)
```

Correção:

```
total = 0
```

```
total = total + 1  
print(total)
```

Passo 1: Verifique se todas as variáveis são inicializadas antes de serem usadas.

Passo 2: Inicialize as variáveis com valores apropriados.

05

Não Usar Parênteses Corretamente em Expressões Lógicas.

Descrição: Esquecer de usar parênteses para agrupar expressões lógicas pode levar a resultados inesperados.

Tipo de Erro: Erro lógico.

Exemplo:

```
idade = 20  
renda = 3000  
if idade > 18 and renda > 2000 or idade < 25:  
    print("Aprovado")
```

Correção:

```
idade = 20
```

```
renda = 3000
if (idade > 18 and renda > 2000) or idade < 25:
    print("Aprovado")
```

Passo 1: Revise as expressões lógicas para garantir que a precedência dos operadores está correta.

Passo 2: Use parênteses para agrupar expressões e clarificar a lógica.

O que é o Google Colab e por que ele é utilizado no curso?

O Google Colab é uma plataforma web que permite a execução de códigos Python em um ambiente virtual. Ele é utilizado no curso por ser uma ferramenta acessível e prática para iniciantes, permitindo a criação e manipulação de notebooks, além de facilitar a execução de códigos sem a necessidade de instalação de software adicional.

Qual a importância de salvar o trabalho fora do Google Colab?

É importante salvar o trabalho fora do Google Colab devido às limitações de tempo das máquinas virtuais. Isso garante que você não perca seu progresso e possa continuar seu trabalho posteriormente sem interrupções.

Quais são os tipos de dados nativos em Python abordados no curso?

Os tipos de dados nativos em Python abordados no curso incluem inteiros, floats, strings e booleanos. Cada tipo de dado é explicado em detalhes, com exemplos práticos de como utilizá-los e manipulá-los.

O que são variáveis e como elas são utilizadas em Python?

Variáveis são espaços na memória que armazenam dados. Em Python, elas são utilizadas para guardar valores que podem ser manipulados ao longo do código. O curso ensina como declarar variáveis, boas práticas de nomenclatura e como identificar tipos de dados utilizando a função `type`.

Quais são as boas práticas recomendadas para a utilização de variáveis?

As boas práticas recomendadas incluem o uso de nomes descritivos para variáveis, a diferenciação entre maiúsculas e minúsculas, e a documentação do código com comentários. Essas práticas ajudam a manter o código organizado e fácil de entender.

O que são strings e quais operações podem ser realizadas com elas?

Strings são sequências de caracteres utilizadas para representar texto. O curso aborda diversas operações com strings, como concatenação, formatação, fatiamento e métodos como ``upper``, ``find`` e ``replace``. Essas operações são ilustradas com exemplos práticos.

Como são utilizadas as variáveis booleanas e operadores lógicos em Python?

Variáveis booleanas armazenam valores ``True`` ou ``False``. O curso ensina como utilizar operadores lógicos (``and``, ``or``, ``not``) para realizar comparações e tomar decisões no código. Exemplos práticos, como um sistema de controle de acesso, são utilizados para ilustrar esses conceitos.

O que são estruturas condicionais e como elas são utilizadas?

Estruturas condicionais (``if``, ``else``, ``elif``) são utilizadas para executar diferentes blocos de código com base em condições específicas. O curso destaca a importância da indentação e dos testes de mesa para verificar o comportamento do código em diferentes cenários.

Como são utilizados os laços de repetição em Python?

Laços de repetição (``for``, ``while``) são utilizados para percorrer coleções de dados. O curso ensina como utilizar o comando ``range``, operadores compostos e o ``enumerate`` para obter índices ao percorrer listas. Exemplos práticos são fornecidos para ilustrar o uso desses laços.