E-BOOK
DO ESTUDANTE

Introdução a Dados

Não tenha medo de Python!

RESILIA

SUMÁRIO

Introdução a Dados

Não tenha medo de Python!

Python é uma linguagem de programação de alto nível, interpretada e de uso geral. Ela é frequentemente utilizada para desenvolvimento de aplicações web, científicas, análise de dados, automação de tarefas, entre outras. Com uma **sintaxe** clara e intuitiva, Python permite que programadores desenvolvam soluções rapidamente, sem a necessidade de lidar com aspectos complexos de outras linguagens.

Uma das principais características do Python é a sua grande **comunidade** de desenvolvedores, que criam bibliotecas e módulos para diferentes necessidades. Dessa forma, é possível utilizar essas **bibliotecas** para resolver problemas de modo mais eficiente, sem precisar reinventar a roda. Além disso, Python é uma linguagem multiplataforma, o que significa que ela pode ser executada em diferentes sistemas operacionais, como Windows, Linux e MacOS.

Com base nesses pontos iremos, através deste e-book, apresentar os **principais conceitos** relacionados à lógica de programação com Python, de modo a dar suporte para lidar com as mais diversas tarefas que possam fazer uso dessa linguagem, incluindo operações com dados.

CONTEXTUALIZANDO

Python é atualmente uma das linguagens de programação mais utilizadas em análise e tratamento de dados. Isso se deve à sua facilidade de uso e ampla disponibilidade de **bibliotecas especializadas** para essas tarefas, como o Pandas, Numpy e Matplotlib. Com essas ferramentas, os usuários podem importar, manipular e visualizar dados em diferentes formatos, como arquivos CSV, Excel e SQL.

Uma das principais vantagens do Python é sua **flexibilidade** em lidar com diferentes tipos de dados, permitindo que os usuários analisem grandes quantidades de informações e obtenham insights valiosos. Além disso, a linguagem possui uma grande comunidade de desenvolvedores e usuários, o que facilita o compartilhamento de conhecimento e a criação de soluções colaborativas para problemas complexos de análise de dados.

Por fim, o Python também é amplamente utilizado em **aprendizado de máquina e inteligência artificial**, graças às bibliotecas como o Tensorflow, Scikit-Learn e PyTorch. Isso permite que os usuários construam modelos preditivos, classifiquem dados e realizem outras tarefas avançadas de análise de dados com facilidade e eficiência.

INTRODUÇÃO À PROGRAMAÇÃO COM PYTHON

Python é uma linguagem de programação interpretada, de alto nível, multiplataforma e de fácil aprendizado. Com sua sintaxe simples e clara, é uma das linguagens mais populares do mundo, utilizada em diversas áreas, desde desenvolvimento web e científico até inteligência artificial e aprendizado de máquina. Vamos começar falando sobre variáveis, tipos de dados e atribuições?

Uma **variável** em Python é um nome que faz referência a um valor. Em outras palavras, é um espaço na memória do computador onde podemos armazenar um valor e usá-lo posteriormente no nosso programa.

Em Python, não é necessário declarar o tipo de uma variável antes de usá-la. Quando uma variável é atribuída, o interpretador Python determina automaticamente o tipo de dados que está sendo atribuído. Vamos ver um exemplo:

```
idade = 25
nome = "Lucas"
altura = 1.75
```

Nesse exemplo, idade é uma variável do tipo inteiro, nome é uma variável do tipo string e altura é uma variável do tipo float.
Já as atribuições são usadas para associar um valor a uma variável. Em Python, usamos o sinal de igual (=) para realizar atribuições.

```
x = 10
y = 5
z = x + y
```

Nesse exemplo, a variável x recebe o valor 10, a variável y recebe o valor 5 e a variável z recebe o resultado da soma de x e y, ou seja, 15.
Python tem vários tipos de dados embutidos. Vamos ver alguns dos principais:

Números: incluem inteiros (int), números de ponto flutuante (float), ou decimais, e números complexos (complex).

```
a = 10 # inteiro
b = 3.14 # float
c = 2 + 3j # complexo
```

Strings: são sequências de caracteres. Podem ser criadas usando aspas simples ou duplas.

```
nome = "Lucas"
sobrenome = 'Silva'
```

Booleanos: têm apenas dois valores possíveis, True (verdadeiro) ou False (falso).

```
verdadeiro = True
falso = False
```

Listas: são coleções ordenadas e mutáveis de elementos. Podem conter elementos de tipos diferentes.

```
lista = [1, "dois", 3.0]
```

Tuplas: são coleções ordenadas e imutáveis de elementos. Podem conter elementos de tipos diferentes.

```
tupla = (1, "dois", 3.0)
```

Conjuntos: são coleções não ordenadas e sem elementos duplicados.

```
conjunto = {1, "dois", 3.0}
```

Dicionários: são coleções não ordenadas de pares chave-valor.

```
dicionario = {"nome": "Lucas", "idade": 25, "altura": 1.75}
```

Além desses tipos de dados, Python tem outros tipos embutidos, como bytes, bytearrays, range, entre outros. Também é possível criar seus próprios tipos de dados usando classes.

Por fim, é importante lembrar que em Python, tudo é um **objeto**. Isso significa que cada valor que atribuímos a uma variável é um objeto com seus próprios métodos e propriedades. Agora que você conheceu um pouco dessa linguagem, que tal aprofundarmos mais vendo sobre instruções de condição?

INSTRUÇÕES DE CONDIÇÃO

As instruções de condição em Python permitem que um programa execute diferentes caminhos de código com base em uma condição. Isso autoriza o programa a tomar decisões dinâmicas em tempo de execução, com base em entradas do usuário, valores de variáveis ou outros fatores.

A instrução mais básica de condição é o "**if**". O bloco de código dentro do "if" é executado apenas se a condição especificada for avaliada como verdadeira. Caso contrário, o bloco de código dentro do "if" é ignorado e a execução continua após o bloco de código. Veja o exemplo a seguir:

```
x = 10
if x > 5:
    print("x é maior que 5")
```

Nesse exemplo, o bloco de código dentro do "if" será executado porque a condição x > 5 é verdadeira.

Uma instrução "if" pode ser seguida por uma instrução "**else**", que é executada quando a condição especificada no "if" é avaliada como falsa. Veja o exemplo a seguir:

```
x = 2
if x > 5:
    print("x é maior que 5")
else:
    print("x é menor ou igual a 5")
```

Nesse exemplo, o bloco de código dentro do "else" será executado porque a condição x > 5 é falsa.

Em Python, também é possível usar uma instrução "**elif**" (abreviação de "else if") para testar múltiplas condições. O bloco de código dentro do "elif" é executado apenas se a condição especificada no "elif" for avaliada como verdadeira. Caso contrário, a execução passa para a próxima instrução "elif" ou "else", se houver. Veja o exemplo a seguir:

```
x = 7
if x > 10:
    print("x é maior que 10")
elif x > 5:
    print("x é maior que 5, mas menor ou igual a 10")
else:
    print("x é menor ou igual a 5")
```

Nesse exemplo, o bloco de código dentro do primeiro "elif" será executado porque a condição x > 5 é verdadeira, mas a condição x > 10 é falsa.

Além disso, é possível usar operadores lógicos para combinar múltiplas condições. Por exemplo, o operador "**and**" é usado para testar se ambas as condições são verdadeiras, enquanto o operador "**or**" é usado para testar se pelo menos uma das condições é verdadeira. Veja o exemplo a seguir:

```
x = 7
y = 8
if x > 5 and y > 5:
    print("x e y são maiores que 5")
if x > 10 or y > 10:
    print("pelo menos x ou y é maior que 10")
```

Nesse exemplo, o primeiro "if" será executado porque tanto x quanto y são maiores que 5. O segundo "if" não será executado porque nenhuma das condições x > 10 e y > 10 é verdadeira.

Para condições mais complexas, é possível usar operadores de comparação adicionais, como "**!=**" para "diferente de" ou "**>=**" para "maior ou igual a". Também é possível usar **parênteses** para controlar a ordem de avaliação das condições. Veja o exemplo a seguir:

```
x != 7
y >=8
```

Vimos como os blocos de condição são importantes para realizarmos ações que visam operações onde é necessário verificar qual ação deve ser executada sob uma determinada condição. Agora, veremos outro bloco igualmente importante: as instruções de loop.

INSTRUÇÕES DE LOOP

As instruções de loop em Python permitem que um programa execute um bloco de código várias vezes. Existem dois tipos principais de loop: o "**for**" e o "**while**". O loop "**for**" é usado para percorrer uma sequência, como uma lista, tupla ou string. Ele executa o bloco de código uma vez para cada elemento da sequência. Veja o código a seguir:

```
frutas = ["maçã", "banana", "cereja"]
for fruta in frutas:
      print(fruta)
```

Nesse exemplo, o bloco de código dentro do "for" é executado três vezes, uma vez para cada elemento da lista "frutas". A variável "fruta" assume o valor de cada elemento na lista em cada iteração.

O loop "**while**" é usado para executar um bloco de código enquanto uma condição especificada for verdadeira. Ele pode ser útil quando o número de iterações não é conhecido com antecedência. Veja o código a seguir:

```
i = 1
while i < 6:
    print(i)
    i += 1
```

Nesse exemplo, o bloco de código dentro do "while" é executado cinco vezes, uma vez para cada valor de "i" de 1 a 5. A variável "i" é incrementada em cada iteração, e o loop é interrompido quando "i" atinge o valor 6.

Dentro de um loop, é possível usar a instrução "**break**" para interromper o loop prematuramente, caso uma determinada condição seja atendida. Além disso, é possível usar a instrução "**continue**" para pular para a próxima iteração do loop, ignorando o resto do código dentro do bloco de loop atual.

```
for i in range(1, 11):
    if i == 5:
        break
    print(i)
```

Nesse exemplo, o bloco de código dentro do "for" é executado cinco vezes, uma vez para cada valor de "i" de 1 a 4. Quando "i" é igual a 5, a instrução "break" é executada, interrompendo o loop prematuramente.

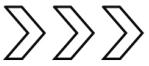
Os loops "**for**" e "**while**" também podem ser aninhados, permitindo que um loop seja executado dentro de outro loop. Veja o código a seguir:

```
for i in range(1, 4):
    for j in range(1, 4):
        print(i, j)
```

Nesse exemplo, o bloco de código dentro do primeiro "for" é executado três vezes, uma vez para cada valor de "i" de 1 a 3. Dentro desse bloco, o bloco de código dentro do segundo "for" é executado três vezes para cada valor de "i", uma vez para cada valor de "j" de 1 a 3. O resultado é uma matriz de valores de "i" e "j" de 1 a 3.

Vimos como os loops em Python podem ser úteis para automatizarmos tarefas repetindo determinadas operações. Desse modo, isso abre uma porta para nosso próximo conteúdo: Funções.

FUNÇÕES



As funções em Python são blocos de código que podem ser reutilizados em diferentes partes do programa. Elas recebem argumentos como entrada e podem retornar valores como saída.

Para definir uma função em Python, utiliza-se a palavra-chave “**def**” seguida do **nome da função** e dos **parâmetros entre parênteses**. A definição da função é seguida por um bloco de código indentado, que é o corpo da função.

Por exemplo, a função abaixo recebe dois números como argumentos e retorna a soma deles: Veja o código a seguir:

```
def soma(a, b):
    return a + b
```

Para chamar a função acima, basta passar dois valores como argumentos:

```
resultado = soma(3, 5)
print(resultado)
```

Além de receber argumentos, as funções em Python também podem ter valores padrão para os parâmetros. Isso significa que, se um argumento não for passado na chamada da função, o valor padrão será utilizado. Veja o código a seguir:

```
def potencia(base, expoente=2):
    return base ** expoente

print(potencia(3))
print(potencia(3, 3))
```

Outra característica importante das funções em Python é que elas podem receber um número variável de argumentos. Isso é feito usando o operador "*****" para os argumentos posicionais e "******" para os argumentos nomeados. Veja o código a seguir:

```
def print_args(*args, **kwargs):
    print("Argumentos posicionais:")
    for arg in args:
        print(arg)
    print("Argumentos nomeados:")
    for key, value in kwargs.items():
        print(f"{key}: {value}")

print_args(1, 2, 3, nome="João", idade=30)
```

O código acima exibe:

```
Argumentos posicionais:
1
2
3

Argumentos nomeados:
nome: João
idade: 30
```

Finalmente, é importante mencionar que as funções em Python podem retornar qualquer tipo de objeto, incluindo outras funções. Isso permite que sejam criadas funções de ordem superior, que recebem ou retornam outras funções. Veja o código a seguir:

```
def criar_soma(x):
    def soma(y):
        return x + y
    return soma

soma_3 = criar_soma(3)
soma_5 = criar_soma(5)

print(soma_3(4))
print(soma_5(4))
```

No exemplo acima, a função "**criar_soma**" retorna outra função que soma o argumento recebido com o valor x passado na criação da função.

Ao longo deste texto, foram abordadas algumas premissas sobre a lógica de programação em Python e como fazer operações com instruções de condição, loop e funções. Agora, é seguir em frente e continuar aperfeiçoando seus estudos usando esta poderosa linguagem de programação.

Para refletir

- Qual é a importância da linguagem Python em relação a outras linguagens de programação simples, como Ruby e Perl?
- Como a simplicidade e a clareza de sintaxe da linguagem Python a tornam uma escolha popular para iniciantes e para projetos com prazos apertados?
- Qual é a importância da comunidade de desenvolvedores Python e como ela contribui para a evolução contínua da linguagem?

Atividade: Beba mais da fonte

Que tal fazer uma pesquisa?

Busque saber:

- Como a linguagem Python pode ser usada para desenvolver jogos simples ou aplicativos de desktop;
- Quais são as bibliotecas e frameworks mais adequados para esses projetos;
- Quais são as principais diferenças em relação a análise de dados com Python.

Para ir além

- Vimos uma introdução sobre a linguagem Python. No entanto, ainda há muito mais para descobrir! No site abaixo, podemos encontrar mais informações relacionadas a como essa linguagem funciona.

  <https://medium.com/@ebsouza/python-essencial-muito-al%C3%A9m-da-l%C3%B3gica-de-programa%C3%A7%C3%A3o-c832c96d5ea7>

- Você descobriu como os blocos de condição se comportam e como podem ser usados. No site abaixo, há exemplos mais complexos sobre o uso dessa instrução.

  <https://www.devmedia.com.br/estruturas-de-condicao-em-python/37158>

- Conhecemos a importância dos laços de repetição com alguns exemplos práticos. No site abaixo, podemos conhecer outras aplicações de uso prático dos loops em Python.

  <https://pythonacademy.com.br/blog/estruturas-de-repeticao>

- Vimos como as funções são úteis para a construção de blocos de tarefas que podem ser usadas várias vezes. No site abaixo, temos mais exemplos práticos e complexos de funções em Python.

  <https://algoritmosempython.com.br/cursos/programacao-python/funcoes/>

RESUMO DE PYTHON

INTRODUÇÃO A PROGRAMAÇÃO EM PYTHON

Uma variável em Python é um nome que faz referência a um valor.Já as atribuições são usadas para associar um valor a uma variável.

Tipos de Dados em Python

```
a = 10 # inteiro
b = 3.14 # float
c = 2 + 3j # complexo
```

```
nome = "Lucas"
sobrenome = 'Silva'
```

```
verdadeiro = True
falso = False
```

```
lista = [1, "dois", 3.0]
```

```
conjunto = {1, "dois", 3.0}
```

Declaração de variáveis

```
x = 10
y = 5
z = x + y
```

INSTRUÇÕES DE LOOP

As instruções de loop em Python permitem que um programa execute um bloco de código várias vezes.

Loop For

```
frutas = ["maçã", "banana", "cereja"]
for fruta in frutas:
      print(fruta)
```

Loop While

```
i = 1
while i < 6:
    print(i)
    i += 1
```

Loop For com Break

```
for i in range(1, 11):
    if i == 5:
        break
    print(i)
```

Loop dentro de loop

```
for i in range(1, 4):
    for j in range(1, 4):
        print(i, j)
```

INSTRUÇÕES DE CONDIÇÃO

As instruções de condição em Python permitem que um programa execute diferentes caminhos de código com base em uma condição.

Condição "If"

```
x = 10
if x > 5:
    print("x é maior que 5")
```

Condição "If, Else"

```
x = 2
if x > 5:
    print("x é maior que 5")
else:
    print("x é menor ou igual a 5")
```

Condição "If, Else, Elif"

```
x = 7
if x > 10:
    print("x é maior que 10")
elif x > 5:
    print("x é maior que 5, mas menor ou igual
a 10")
else:
    print("x é menor ou igual a 5")
```

Condição com Operadores Lógicos

```
x = 7
y = 8
if x > 5 and y > 5:
    print("x e y são maiores que 5")
if x > 10 or y > 10:
    print("pelo menos x ou y é maior que 10")
```

FUNÇÕES

As funções em Python são blocos de código que podem ser reutilizados em diferentes partes do programa.

Definição de função

```
def soma(a, b):
    return a + b
```

```
def potencia(base, expoente=2):
    return base ** expoente

print(potencia(3))
print(potencia(3, 3))
```

Função com vários argumentos

```
def print_args(*args, **kwargs):
    print("Argumentos posicionais:")
    for arg in args:
        print(arg)
    print("Argumentos nomeados:")
    for key, value in kwargs.items():
        print(f"{key}: {value}")

print_args(1, 2, 3, nome="João", idade=30)
```

Funções que retornam funções

```
def criar_soma(x):
    def soma(y):
        return x + y
    return soma

soma_3 = criar_soma(3)
soma_5 = criar_soma(5)

print(soma_3(4))
print(soma_5(4))
```

Até a próxima e
#confianoprocesso

RESILIA