Digital College

FORMAÇÃO EM

DATA ANALYTICS

MÓDULO 3:

PYTHON PARA ANÁLISE DE DADOS

UNIDADE 1:

INICIANDO COM PYTHON

Sumário

1. Introdução	**03**
1.1. Por que usar Python?	03
2. Instalação e configuração do ambiente	**05**
2.1. Instalando o Anaconda Python	06
2.2. Configurando o ambiente de desenvolvimento	07
2.3. Escrevendo o primeiro programa em Python	08
3. Tipos de Dados e Variáveis	**09**
3.1. Tipos numéricos	09
3.2. Strings	11
3.3. Booleanos	13
4. Estrutura de Controle de Fluxo	**15**
4.1. Condições if, else e elif	15
4.2. Loops (While e For)	17
5. Coleções de Dados	**20**
5.1. Listas	21
5.2. Dicionários	24
5.3. Tuplas	26
6. Módulos, Pacotes e Funções	**28**
7. Manipulação de Arquivos	**34**
7.1. Manipulando arquivos TXT	34
7.2. Manipulando arquivos CSV	37
7.3. Manipulando arquivos Excel	40
7.4. Manipulando arquivos JSON	41
8. Exceções e tratamento de erros	**45**

1. Introdução

Python é uma linguagem de programação de alto nível, interpretada e de propósito geral. Foi criada por Guido van Rossum, em 1991, e é atualmente mantida por uma comunidade de desenvolvedores. A Python é conhecida por ser uma linguagem fácil de aprender e de usar, com uma sintaxe simples e legível. Além disso, possui uma vasta biblioteca padrão e muitas outras bibliotecas de terceiros, o que faz dela uma linguagem muito versátil e utilizada em diversas áreas, como ciência de dados, inteligência artificial, desenvolvimento web, automação de tarefas, entre outras. A Python é uma linguagem interpretada, o que significa que o código é executado linha por linha, sem necessidade de compilá-lo antes da execução, facilitando a prototipagem e a depuração de erros.

1.1. Por que usar Python?

Existem muitas razões pelas quais alguém pode querer usar Python em seus projetos de programação. Aqui estão algumas das principais razões:

- **Facilidade de aprendizado:** Python é considerada uma das linguagens de programação mais fáceis de aprender e usar, graças à sua sintaxe simples e legível. Isso faz com que seja uma ótima escolha para iniciantes em programação;

- **Versatilidade:** Python é uma linguagem de propósito geral, o que significa que pode ser usada para uma grande variedade de projetos, desde desenvolvimento web até ciência de dados e inteligência artificial;

- **Biblioteca padrão abrangente:** a biblioteca padrão da Python é muito completa e possui muitas funcionalidades prontas para uso, o que economiza tempo e esforço na programação de soluções para problemas comuns;

- **Grande comunidade:** Python tem uma das maiores comunidades de desenvolvedores do mundo, com muitos recursos, fóruns e bibliotecas disponíveis;

- **Suporte multiplataforma:** Python pode ser executada em uma variedade de plataformas, incluindo Windows, Mac e Linux, além de ser compatível com muitas outras linguagens e ferramentas;

- **Linguagem de script:** Python é uma linguagem interpretada, o que significa que o código pode ser executado diretamente, sem a necessidade de compilação. Isso facilita a prototipagem e o desenvolvimento rápido de soluções.

- **Fácil integração:** Python é fácil de integrar com outras linguagens e sistemas, permitindo que ela seja usada em conjunto com outras ferramentas e tecnologias.

2. Instalação e configuração do ambiente

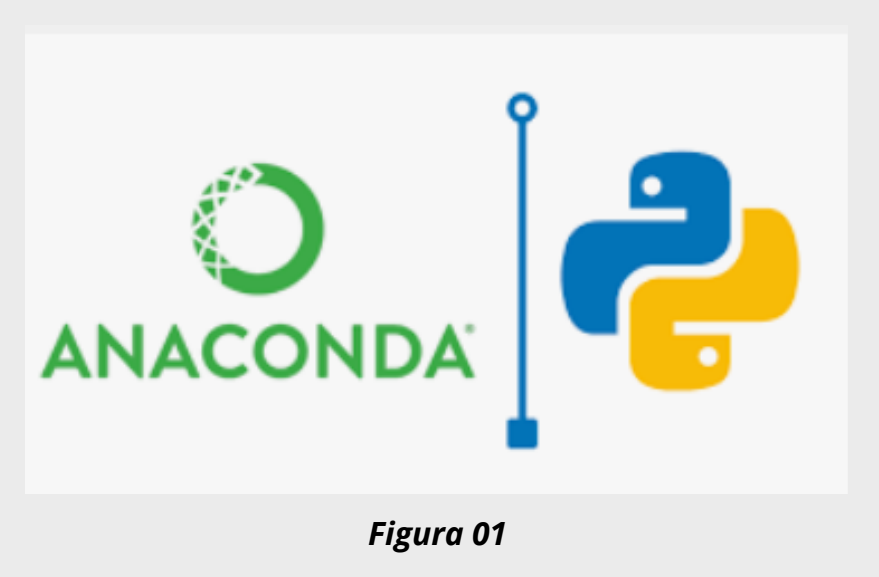


Figura 01

O Anaconda Python é uma distribuição de software gratuita e de código aberto que inclui Python e outras ferramentas de análise de dados e ciência de dados. Ele foi projetado para facilitar a instalação e o gerenciamento de pacotes e bibliotecas usados em projetos de análise de dados, tornando mais fácil para os desenvolvedores criar e executar aplicativos de análise de dados complexos.

O Anaconda Python vem com uma série de bibliotecas e ferramentas populares para análise de dados, como o NumPy, Pandas, Matplotlib, SciPy, Jupyter, entre outras. Além disso, ele oferece um gerenciador de pacotes fácil de usar, chamado Conda, que permite instalar e gerenciar bibliotecas e pacotes adicionais.

O Anaconda Python é usado por muitos profissionais de ciência de dados, pesquisadores acadêmicos e desenvolvedores de software em todo o mundo, devido à sua facilidade de uso e grande número de bibliotecas pré-instaladas. Ele pode ser executado em uma variedade de plataformas, incluindo Windows, Mac e Linux, e é frequentemente usado em conjunto com ambientes de desenvolvimento integrado (IDEs) como o Jupyter Notebook para criar aplicativos de análise de dados interativos e exploratórios.

2.1. Instalando o Anaconda Python

Para instalar o Anaconda Python, siga as seguintes etapas:

- Baixe o instalador apropriado do Anaconda Python para o seu sistema operacional (Windows, Mac ou Linux) a partir do site oficial da Anaconda: https://www.anaconda.com/products/individual
- Execute o instalador e siga as instruções na tela. Certifique-se de selecionar a opção "Adicionar Anaconda ao PATH do sistema" para que você possa acessar o Anaconda a partir do prompt de comando ou terminal;
- Quando a instalação estiver concluída, abra o terminal ou prompt de comando e digite "conda" para verificar se o Conda (gerenciador de pacotes do Anaconda) foi instalado corretamente;
- Agora você pode começar a criar ambientes virtuais e instalar pacotes do Python usando o Conda. Para criar um novo ambiente virtual, basta digitar "conda create --name NOME_DO_AMBIENTE python=X.X" (substitua "NOME_DO_AMBIENTE" pelo nome do ambiente que você deseja criar e "X.X" pela versão do Python que você deseja usar);
- Para ativar um ambiente virtual, digite "conda activate NOME_DO_AMBIENTE". Para desativar o ambiente, digite "conda deactivate".

Pronto! Agora você pode começar a usar o Anaconda Python para desenvolver seus projetos em Python e análise de dados.

2.2. Configurando o ambiente de desenvolvimento

Para configurar o ambiente de desenvolvimento Python com o VS Code, siga as seguintes etapas:

- Certifique-se de ter o Python e o VS Code instalados em seu sistema;
- Abra o VS Code e instale a extensão "Python" da Microsoft. Para fazer isso, pressione Ctrl + Shift + X para abrir a barra de extensões, pesquise "Python" e selecione a extensão "Python" da Microsoft. Clique em "Instalar" para instalar a extensão;
- Crie um ambiente virtual para seu projeto. Isso é recomendado para manter as dependências do seu projeto separadas do Python global. Para fazer isso, abra o terminal do VS Code (Menu Terminal -> Novo Terminal) e digite "python -m venv NOME_DO_AMBIENTE". Substitua "NOME_DO_AMBIENTE" pelo nome que você deseja dar ao seu ambiente;
- Ative o ambiente virtual digitando "NOME_DO_AMBIENTE\Scripts\activate" no terminal do VS Code (substitua "NOME_DO_AMBIENTE" pelo nome do ambiente que você acabou de criar);
- Crie um novo projeto no VS Code e abra a pasta raiz do projeto;
- Pressione Ctrl + Shift + P para abrir a barra de comando e pesquise por "Python: Select Interpreter". Selecione o ambiente virtual que você criou na etapa 3;

- Agora você pode começar a escrever código Python em seu projeto. Você também pode usar as funcionalidades da extensão Python para depuração, testes e outras tarefas;

- Lembre-se de sempre ativar seu ambiente virtual antes de iniciar o VS Code e trabalhar em seu projeto. Isso garantirá que todas as dependências do seu projeto estejam instaladas corretamente e que o VS Code esteja usando a versão correta do Python.

2.3. Escrevendo o primeiro programa em Python

Para escrever o primeiro programa em Python, siga as seguintes etapas:

- Abra o VS Code e crie uma nova pasta para seu projeto;
- Crie um novo arquivo e salve-o com o nome "primeiro_programa.py". Certifique-se de que a extensão do arquivo seja ".py" para que o VS Code reconheça que é um arquivo Python;
- Digite o seguinte código no arquivo:

```bash
print("Olá, mundo!")
```

Figura 02

- Salve o arquivo e pressione F5 para executar o programa;
- O resultado do programa deve aparecer na saída do terminal do VS Code, que deve exibir a mensagem "Olá, mundo!".

Parabéns, você escreveu e executou seu primeiro programa em Python! A partir daqui, você pode começar a aprender os conceitos básicos da linguagem Python e a explorar as muitas possibilidades que ela oferece.

3. Tipos de Dados e Variáveis

Em Python, existem vários tipos de dados que podem ser usados em variáveis, entre eles:

- Números inteiros (int): representam valores inteiros sem ponto decimal, por exemplo: 1, 2, 3, -4, -5, etc.;
- Números de ponto flutuante (float): representam valores numéricos com ponto decimal, por exemplo: 1.2, 3.5, -4.7, etc.;
- Números complexos (complex): representam números complexos com uma parte real e uma parte imaginária, por exemplo: 3 + 2j, -1 + 4j, etc.;
- Strings (str): representam uma sequência de caracteres, como um texto ou palavra. Strings são criados usando aspas simples (') ou aspas duplas ("");
- Booleanos (bool): representam valores verdadeiros ou falsos (True ou False).

Para armazenar esses tipos de dados, usamos variáveis em Python. Uma variável é um espaço na memória do computador que armazena um valor específico. Para criar uma variável, basta atribuir um valor a um nome de variável.

3.1. Tipos numéricos

Em Python, os tipos numéricos incluem inteiros (int), números de ponto flutuante (float) e números complexos (complex). Esses tipos de dados são frequentemente usados para fazer cálculos e armazenar valores numéricos.

Aqui está um exemplo prático de como usar os tipos numéricos em Python:

```
# Exemplo de uso de tipos numéricos em Python

# Criando variáveis numéricas
x = 5    # inteiro
y = 2.5  # float
z = 1 + 2j  # complexo

# Imprimindo valores das variáveis
print(x)  # 5
print(y)  # 2.5
print(z)  # (1+2j)

# Realizando operações numéricas
soma = x + y        # soma dois números
subtracao = x - y   # subtrai dois números
multiplicacao = x * y   # multiplica dois números
divisao = x / y     # divide dois números
potencia = x ** 2   # eleva um número a uma potência
modulo = x % 2      # retorna o resto da divisão por 2

# Imprimindo os resultados das operações
print(soma)         # 7.5
print(subtracao)    # 2.5
print(multiplicacao)  # 12.5
print(divisao)      # 2.0
print(potencia)     # 25
print(modulo)       # 1
```

Figura 03

3.2. Strings

Em Python, as strings são usadas para representar texto. Uma string é uma sequência de caracteres delimitada por aspas simples (' ') ou aspas duplas (" "). Aqui estão alguns conceitos e exemplos práticos sobre strings em Python:

- **Indexação:** os caracteres em uma string podem ser acessados usando índices, começando em 0 para o primeiro caractere.

```
# Exemplo de indexação de strings em Python

texto = "Olá, mundo!"
primeiro_caractere = texto[0]  # 'O'
segundo_caractere = texto[1]   # 'l'
ultimo_caractere = texto[-1]   # '!'
```

Figura 04

- **Concatenação:** é possível concatenar duas ou mais strings usando o operador de adição (+).

```
# Exemplo de concatenação de strings em Python

saudacao = "Olá"
nome = "João"
frase = saudacao + ", " + nome + "!"
print(frase)  # "Olá, João!"
```

Figura 05

- **Comprimento:** o comprimento de uma string (ou seja, o número de caracteres) pode ser obtido usando a função len().

```
# Exemplo de comprimento de strings em Python

texto = "Esta é uma string."
comprimento = len(texto)
print(comprimento)  # 18
```

Figura 06

- **Métodos de string:** as strings em Python possuem vários métodos que podem ser usados para realizar operações comuns, como transformar letras em maiúsculas ou minúsculas, separar strings em substrings ou remover espaços em branco.

```
# Exemplo de uso de métodos de strings em Python

texto = "Python é uma linguagem de programação"
maiusculo = texto.upper()                # 'PYTHON É UMA LINGUAGEM DE PROGRAMAÇÃO'
minusculo = texto.lower()                # 'python é uma linguagem de programação'
palavras = texto.split()                 # ['Python', 'é', 'uma', 'linguagem', 'de',
substituicao = texto.replace('Python', 'Java')   # 'Java é uma linguagem de progra
```

Figura 07

Neste exemplo, usamos alguns dos métodos de string mais comuns em Python. O método upper() transforma todas as letras em maiúsculas, enquanto o método lower() transforma todas as letras em minúsculas. O método split() divide uma string em substrings com base em um separador (por padrão, o espaço em branco). O método replace() substitui uma substring por outra em uma string.

3.3. Booleanos

Em Python, o tipo booleano é usado para representar valores lógicos verdadeiros ou falsos. Os valores booleanos são representados pelos objetos True e False. Aqui estão alguns conceitos e exemplos práticos sobre booleanos em Python:

Operadores de comparação: os operadores de comparação são usados para comparar dois valores e retornar um valor booleano. Por exemplo:

```
# Exemplo de operadores de comparação em Python

x = 5
y = 10

igual = x == y    # False
diferente = x != y   # True
maior = x > y     # False
menor = x < y     # True
maior_ou_igual = x >= y   # False
menor_ou_igual = x <= y   # True
```

Figura 08

Operadores booleanos: os operadores booleanos são usados para combinar valores booleanos. Os operadores booleanos mais comuns são and, or e not. Por exemplo:

```
# Exemplo de operadores booleanos em Python

x = 5
y = 10
z = 15

resultado1 = (x < y) and (y < z)    # True
resultado2 = (x > y) or (y > z)     # False
resultado3 = not (x < y)            # False
```

Figura 09

Valores truthy e falsy: em Python, além dos valores booleanos explícitos True e False, alguns outros valores também são considerados truthy (verdadeiros) ou falsy (falsos) em um contexto booleano. Valores falsy incluem False, None, 0, "" (uma string vazia), [] (uma lista vazia), () (uma tupla vazia) e {} (um dicionário vazio). Qualquer outro valor é considerado truthy. Por exemplo:

```
# Exemplo de valores truthy e falsy em Python

if 0:
    print("Este código nunca será executado.")
else:
    print("Este código será executado.")   # "Este código será executado."
```

Figura 10

Neste exemplo, o valor 0 é considerado falsy em um contexto booleano, portanto o código dentro do bloco if nunca será executado. Em vez disso, o código dentro do bloco else será executado.

4. Estrutura de Controle de Fluxo

As estruturas de controle de fluxo em Python permitem que você controle o fluxo de execução do programa com base em certas condições. As estruturas de controle de fluxo mais comuns em Python são as instruções if, else e elif, bem como os loops while e for.

4.1. Condições if, else e elif

As condições if, else e elif são instruções em Python que permitem que você execute diferentes blocos de código com base em uma determinada condição. Aqui estão algumas definições e exemplos práticos para cada uma dessas instruções:

If: a instrução "if" é usada para executar um bloco de código se uma determinada condição for verdadeira. Por exemplo:

```
# Exemplo de instrução if em Python

x = 5

if x > 0:
    print("x é positivo")
```

Figura 11

Neste exemplo, o programa imprime "x é positivo" porque a condição x > 0 é verdadeira.

Else: a instrução “else” é usada para executar um bloco de código se a condição do if for falsa. Por exemplo:

```
# Exemplo de instrução if-else em Python

x = -2

if x > 0:
    print("x é positivo")
else:
    print("x é negativo ou igual a zero")
```

Figura 12

Neste exemplo, o programa imprime "x é negativo ou igual a zero" porque a condição x > 0 é falsa.

Elif: a instrução “elif” é usada para testar várias condições. Por exemplo:

```
# Exemplo de instrução if-elif-else em Python

x = 0

if x > 0:
    print("x é positivo")
elif x < 0:
    print("x é negativo")
else:
    print("x é igual a zero")
```

Figura 13

Neste exemplo, o programa imprime "x é igual a zero" porque a condição x > 0 é falsa, a condição x < 0 também é falsa, e a condição final do else é executada.

Você também pode usar operadores lógicos, como and, or e not, para combinar ou negar condições. Por exemplo:

```
# Exemplo de uso de operadores lógicos em Python

x = 5
y = 10

if x > 0 and y > 0:
    print("x e y são positivos")
elif x < 0 or y < 0:
    print("pelo menos x ou y é negativo")
else:
    print("x e y são iguais a zero")
```

Figura 14

Neste exemplo, o programa imprime "x e y são positivos" porque a condição x > 0 and y > 0 é verdadeira. Se a condição do if fosse falsa, o programa testaria a condição do elif e, se fosse verdadeira, imprimiria "pelo menos x ou y é negativo". Caso contrário, o programa executaria o bloco de código do else.

4.2. Loops (While e For)

Os loops *while* e *for* são estruturas de controle de fluxo em Python que permitem executar um bloco de código várias vezes, com base em uma determinada condição ou intervalo de valores. Aqui estão algumas definições e exemplos práticos para cada um desses loops:

- **while:** o loop while é usado para executar um bloco de código enquanto uma determinada condição for verdadeira. Por exemplo:

```
# Exemplo de loop while em Python

i = 1

while i <= 5:
    print(i)
    i += 1
```

Figura 15

Neste exemplo, o loop *while* imprime os números de 1 a 5, porque a condição i <= 5 é verdadeira até que i seja igual a 6, quando a condição se torna falsa e o loop termina.

- **for:** o loop *for* é usado para iterar sobre uma sequência de valores, como uma lista, uma tupla ou uma string. Por exemplo:

```
# Exemplo de loop for em Python

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

for fruta in frutas:
    print(fruta)
```

Figura 16

Neste exemplo, o loop *for* imprime cada elemento da lista "frutas", um por vez.

Você também pode usar a função range para criar uma sequência de números inteiros para iterar com o loop *for*. Por exemplo:

```
# Exemplo de loop for com a função range em Python

for i in range(1, 6):
    print(i)
```

Figura 17

Neste exemplo, o loop for imprime os números de 1 a 5, porque a função range(1, 6) gera uma sequência de números inteiros de 1 a 5.

Você também pode usar o comando break dentro de um loop para interromper a execução do loop prematuramente, se uma determinada condição for atendida. Por exemplo:

```
# Exemplo de uso do comando break em um loop em Python

i = 1

while i <= 5:
    if i == 3:
        break
    print(i)
    i += 1
```

Figura 18

Neste exemplo, o loop while imprime os números de 1 a 2 e, em seguida, interrompe a execução quando i é igual a 3, porque a condição i == 3 é atendida.

5. Coleções de Dados

Em Python, existem três tipos principais de coleções de dados: listas, dicionários e tuplas. Aqui estão algumas definições e exemplos práticos sobre cada um desses tipos de coleções:

- **Listas:** são coleções ordenadas de elementos. Os elementos podem ser de qualquer tipo de dados e podem ser adicionados, modificados ou removidos facilmente;

- **Dicionários:** dicionários são coleções de pares chave-valor. Cada valor é associado a uma chave única e imutável que pode ser usada para acessar esse valor posteriormente;

- **Tuplas:** as tuplas são coleções imutáveis de elementos, ou seja, não podem ser modificadas após a criação. As tuplas são usadas para armazenar um conjunto de valores que não deve ser alterado.

Cada tipo de coleção de dados tem suas próprias propriedades e métodos. Por exemplo, as listas podem ser usadas para armazenar uma coleção ordenada de elementos, os dicionários são úteis para armazenar informações associadas a uma chave específica e as tuplas são usadas para armazenar informações que não podem ser alteradas após a criação. Ao escolher qual tipo de coleção usar, é importante considerar as necessidades específicas do seu projeto e como cada tipo de coleção pode ser usado para resolver o problema em questão.

5.1. Listas

Em Python, as listas são uma estrutura de dados que permitem armazenar uma coleção ordenada de valores. Aqui estão algumas definições e exemplos práticos sobre listas:

Definição: uma lista em Python é definida usando colchetes [ ] e separando cada elemento com uma vírgula. Os elementos podem ser de qualquer tipo de dado, inclusive outras listas.

```
# Exemplo de criação de uma lista em Python

lista_numeros = [1, 2, 3, 4, 5]
lista_misturada = [1, "dois", True, [6, 7, 8]]
```

Figura 19

Acessando elementos: você pode acessar elementos individuais de uma lista usando um índice entre colchetes []. O primeiro elemento da lista tem índice 0, o segundo tem índice 1, e assim por diante. Também é possível acessar elementos da lista contando de trás para frente, usando índices negativos.

```
# Exemplo de acesso a elementos de uma lista em Python

lista_numeros = [1, 2, 3, 4, 5]
primeiro_numero = lista_numeros[0]  # retorna 1
ultimo_numero = lista_numeros[-1]  # retorna 5
```

Figura 20

Modificando elementos: você pode modificar elementos individuais de uma lista atribuindo um novo valor a um índice específico.

```
# Exemplo de modificação de elementos de uma lista em Python

lista_numeros = [1, 2, 3, 4, 5]
lista_numeros[0] = 6
```

Figura 21

Adicionando elementos: você pode adicionar elementos a uma lista usando o método append().

```
# Exemplo de adição de elementos a uma lista em Python

lista_numeros = [1, 2, 3, 4, 5]
lista_numeros.append(6)
```

Figura 22

Removendo elementos: você pode remover elementos de uma lista usando os métodos pop() (para remover um elemento com base em seu índice) ou remove() (para remover um elemento com base em seu valor).

```
# Exemplo de remoção de elementos de uma lista em Python

lista_numeros = [1, 2, 3, 4, 5]
lista_numeros.pop(0)  # remove o primeiro elemento da lista
lista_numeros.remove(3)  # remove o valor 3 da lista
```

Figura 23

Verificando a presença de um elemento: você pode verificar se um elemento está presente em uma lista usando o operador in.

```
# Exemplo de verificação da presença de um elemento em uma lista em Python

lista_numeros = [1, 2, 3, 4, 5]
if 3 in lista_numeros:
    print("O número 3 está na lista!")
```

Figura 24

Iterando sobre uma lista: você pode iterar sobre os elementos de uma lista usando um loop *for*.

```
# Exemplo de iteração sobre uma lista em Python

lista_numeros = [1, 2, 3, 4, 5]
for numero in lista_numeros:
    print(numero)
```

Figura 25

Esses são apenas alguns exemplos básicos de como trabalhar com listas em Python. As listas têm muitas outras funcionalidades e métodos que podem ser úteis para diferentes situações de programação.

5.2. Dicionários

Em Python, dicionários são coleções de pares chave-valor. Cada valor é associado a uma chave única e imutável que pode ser usada para acessar esse valor posteriormente. Aqui estão algumas definições e exemplos práticos sobre dicionários em Python.

Criação de um dicionário:

Para criar um dicionário, utilizamos as chaves { } e separamos cada par chave-valor com uma vírgula.

```
# Exemplo de criação de um dicionário em Python

dicionario_pessoa = {"nome": "João", "idade": 30, "cidade": "São Paulo"}
```

Figura 26

Acesso aos valores do dicionário:

Para acessar os valores de um dicionário, basta especificar a chave correspondente ao valor desejado dentro dos colchetes [ ].

```
# Exemplo de acesso aos valores de um dicionário em Python

print(dicionario_pessoa["nome"])  # retorna "João"
```

Figura 27

Adição de novos elementos ao dicionário:

Podemos adicionar um novo par chave-valor a um dicionário simplesmente atribuindo o valor a uma nova chave.

```
# Exemplo de adição de um novo elemento a um dicionário em Python

dicionario_pessoa["profissão"] = "Engenheiro"
```

Figura 28

Remoção de elementos do dicionário:

Podemos remover um elemento de um dicionário utilizando a palavra chave "del" e especificando a chave correspondente.

```
# Exemplo de remoção de um elemento de um dicionário em Python

del dicionario_pessoa["idade"]
```

Figura 29

Métodos úteis dos dicionários:

Alguns dos métodos mais comuns dos dicionários em Python incluem "keys()", que retorna uma lista de todas as chaves do dicionário, e "values()", que retorna uma lista de todos os valores do dicionário.

```
# Exemplo de uso dos métodos "keys()" e "values()" em Python

print(dicionario_pessoa.keys())  # retorna ["nome", "cidade", "profissão"]
print(dicionario_pessoa.values())  # retorna ["João", "São Paulo", "Engenheiro"]
```

Figura 30

Os dicionários são úteis para armazenar informações associadas a uma chave específica. Eles podem ser usados para armazenar informações de contato, configurações de usuário ou qualquer outra informação que precise ser acessada de forma rápida e eficiente.

5.3. Tuplas

Em Python, tuplas são coleções de valores imutáveis e ordenados. Ao contrário das listas, as tuplas não podem ser modificadas após a sua criação. Aqui estão algumas definições e exemplos práticos sobre tuplas em Python:

Criação de uma tupla:

Para criar uma tupla, utilizamos os parênteses ( ) e separamos cada elemento com uma vírgula.

```
# Exemplo de criação de uma tupla em Python

tupla_frutas = ("maçã", "banana", "laranja", "uva")
```

Figura 31

Acesso aos elementos da tupla:

Podemos acessar os elementos de uma tupla utilizando seus índices. Os índices começam em 0.

```
# Exemplo de acesso aos elementos de uma tupla em Python

print(tupla_frutas[0])  # retorna "maçã"
```

Figura 32

Conversão de lista em tupla:

Podemos converter uma lista em uma tupla utilizando a função "tuple()".

```
# Exemplo de conversão de lista em tupla em Python

lista_numeros = [1, 2, 3, 4, 5]
tupla_numeros = tuple(lista_numeros)
```

Figura 33

Desempacotamento de tupla:

Podemos atribuir cada elemento de uma tupla a uma variável separadamente, em um processo conhecido como desempacotamento.

```
# Exemplo de desempacotamento de tupla em Python

tupla_pessoa = ("João", 30, "São Paulo")
nome, idade, cidade = tupla_pessoa
```

Figura 34

Métodos úteis das tuplas:

Alguns dos métodos mais comuns das tuplas em Python incluem "count()", que conta o número de vezes que um elemento aparece na tupla, e "index()", que retorna o índice da primeira ocorrência de um elemento na tupla.

```
# Exemplo de uso dos métodos "count()" e "index()" em Python

print(tupla_frutas.count("banana"))  # retorna 1
print(tupla_frutas.index("laranja"))  # retorna 2
```

Figura 35

As tuplas são úteis quando precisamos armazenar uma coleção de elementos que não pode ser modificada posteriormente, como dias da semana ou coordenadas geográficas. Elas também podem ser usadas para retornar múltiplos valores de uma função.

6. Módulos, Pacotes e Funções

Em Python, funções são blocos de código que podem ser chamados repetidamente em diferentes partes de um programa. As funções permitem que o código seja reutilizado e organizado de uma maneira mais clara e modular. Aqui estão algumas definições e exemplos práticos sobre funções em Python:

Criação de uma função:

Para criar uma função, utilizamos a palavra-chave "def", seguida pelo nome da função, parênteses ( ) e dois-pontos :. O bloco de código da função é definido com indentação.

```
# Exemplo de criação de uma função em Python

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

Figura 36

Chamada de uma função:

Para chamar uma função, utilizamos o nome da função, seguido por parênteses ( ) e os argumentos da função, se houver.

```
# Exemplo de chamada de uma função em Python

x = 3
y = 4
resultado = soma(x, y)
print(resultado)  # retorna 7
```

Figura 37

Argumentos de uma função:

As funções podem ter argumentos, que são valores passados para a função quando ela é chamada. Os argumentos podem ser obrigatórios ou opcionais e podem ter um valor padrão.

```
# Exemplo de função com argumentos obrigatórios e opcionais em Python

def saudacao(nome, sobrenome, formalidade=False):
    if formalidade:
        return f"Olá, Sr. {nome} {sobrenome}"
    else:
        return f"Oi, {nome}"

print(saudacao("João", "Silva"))  # retorna "Oi, João"
print(saudacao("Maria", "Souza", formalidade=True))  # retorna "Olá, Sr. Ma
```

Figura 38

Retorno de uma função:

As funções podem retornar valores com a palavra-chave "return". Se uma função não tiver um comando "return", ela retornará "None" por padrão.

```
# Exemplo de função com retorno em Python

def verifica_par(numero):
    if numero % 2 == 0:
        return True
    else:
        return False

print(verifica_par(4))  # retorna True
print(verifica_par(5))  # retorna False
```

Figura 39

Escopo de uma função:

As variáveis definidas dentro de uma função só existem dentro dessa função, a menos que sejam definidas como globais.

```
# Exemplo de escopo de variáveis em uma função em Python

def funcao():
    x = 5
    print(x)

funcao()  # retorna 5
print(x)  # gera um erro, já que x não existe fora da função
```

Figura 40

As funções são uma parte importante da programação em Python, permitindo que o código seja reutilizado e organizado de forma modular. Elas podem ter argumentos e retornar valores, e podem ser usadas para executar tarefas complexas com eficiência e clareza.

6.1. Módulos e Pacotes

Em Python, um módulo é um arquivo contendo definições e instruções que podem ser usadas em outros arquivos. Módulos permitem reutilizar código existente e aumentar a modularidade do programa. Aqui estão algumas definições e exemplos práticos sobre módulos em Python:

Importação de um módulo:

Para usar um módulo em um programa Python, é necessário importá-lo usando a palavra-chave "import", seguida do nome do módulo. Por exemplo, para importar o módulo "math":

```python
# Exemplo de importação de um módulo em Python

import math

resultado = math.sqrt(4)
print(resultado)  # retorna 2.0
```

Figura 41

Renomeação de um módulo:

Um módulo pode ser renomeado durante a importação usando a palavra-chave "as". Isso pode ser útil para evitar conflitos de nomes ou para tornar o nome do módulo mais fácil de digitar.

```python
# Exemplo de renomeação de um módulo em Python

import math as m

resultado = m.sqrt(4)
print(resultado)  # retorna 2.0
```

Figura 42

Importação de funções específicas de um módulo:

Também é possível importar funções específicas de um módulo usando a palavra-chave "from", seguida do nome do módulo e da função desejada. Isso pode tornar o código mais eficiente em termos de espaço, já que apenas as funções necessárias são importadas.

```python
# Exemplo de importação de funções específicas de um módulo em Python

from math import sqrt

resultado = sqrt(4)
print(resultado)  # retorna 2.0
```

Figura 43

Criação de um módulo:

Qualquer arquivo Python pode ser usado como um módulo, desde que ele contenha definições e instruções Python válidas. Para usar um arquivo como um módulo, basta importá-lo em outro arquivo Python, usando o seu nome de arquivo (sem a extensão .py).

```python
# Exemplo de criação de um módulo em Python

# arquivo meu_modulo.py

def minha_funcao():
    print("Olá, mundo!")

# arquivo principal.py

import meu_modulo

meu_modulo.minha_funcao()  # retorna "Olá, mundo!"
```

Figura 44

Pacotes em Python:

Os pacotes em Python são uma forma de organizar e hierarquizar os módulos em diretórios. Um pacote é simplesmente um diretório que contém um arquivo init.py. Os módulos dentro do pacote podem ser importados usando o nome do pacote seguido do nome do módulo.

```
# Exemplo de uso de pacotes em Python

# diretório pacote
#   - arquivo __init__.py
#   - arquivo modulo.py

# arquivo __init__.py
from .modulo import minha_funcao

# arquivo principal.py
import pacote

pacote.minha_funcao()  # retorna "Olá, mundo!"
```

Figura 45

Os módulos e pacotes são recursos essenciais em Python para reutilizar código, aumentar a modularidade e facilitar a manutenção do código. O uso correto de módulos e pacotes pode tornar o código mais legível, organizado e eficiente.

7. Manipulação de Arquivos

A manipulação de arquivos é uma funcionalidade essencial em muitos projetos em Python. Nesta seção, apresentamos alguns exemplos de como abrir, ler, escrever e fechar arquivos em Python.

7.1. Manipulando arquivos TXT

Abrindo e fechando arquivos

Para abrir um arquivo em Python, utilizamos a função open(). Essa função retorna um objeto que permite acessar o conteúdo do arquivo.

```
arquivo = open('exemplo.txt', 'r')
```

Figura 46

O primeiro argumento é o nome do arquivo que desejamos abrir. O segundo argumento é o modo de acesso ao arquivo. Existem diferentes modos, como:

r: leitura (read);
w: escrita (write);
a: adição (append);
x: exclusivo (exclusive).

Dependendo do modo escolhido, podemos ter diferentes comportamentos ao abrir o arquivo. Por exemplo, ao abrir um arquivo em modo de escrita ('w'), se o arquivo já existir, seu conteúdo será apagado.

Após finalizar a manipulação do arquivo, é importante fechá-lo utilizando o método close().

```
arquivo.close()
```

Figura 47

Esse método retorna todo o conteúdo do arquivo como uma única string. Também é possível ler o arquivo linha a linha, utilizando o método readline().

```
linha1 = arquivo.readline()
linha2 = arquivo.readline()
```

Figura 48

Esse método retorna uma string contendo a próxima linha do arquivo.
Outra forma de ler as linhas de um arquivo é utilizando o método readlines(). Esse método retorna uma lista de strings em que cada elemento corresponde a uma linha do arquivo.

```
linhas = arquivo.readlines()
```

Figura 49

Escrevendo arquivos

Para escrever em um arquivo aberto, podemos utilizar o método write().

```
arquivo.write('Olá, mundo!')
```

Figura 50

Esse método escreve o conteúdo passado como argumento no final do arquivo. Se desejamos escrever em uma nova linha, podemos adicionar o caractere de quebra de linha \n.

```
arquivo.write('Olá, mundo!\n')
```

Figura 51

Também podemos utilizar o método writelines() para escrever uma lista de strings no arquivo.

```
linhas = ['Linha 1\n', 'Linha 2\n', 'Linha 3\n']
arquivo.writelines(linhas)
```

Figura 52

Exemplo completo

A seguir, apresentamos um exemplo completo de como abrir um arquivo, ler seu conteúdo, adicionar novas linhas e fechá-lo.

```
# Abrir arquivo em modo de leitura
arquivo = open('exemplo.txt', 'r')

# Ler o conteúdo do arquivo
conteudo = arquivo.read()

# Imprimir o conteúdo do arquivo
print(conteudo)

# Fechar o arquivo
arquivo.close()

# Abrir arquivo em modo de escrita (sobrescrevendo o conteúdo anterior)
arquivo = open('exemplo.txt', 'w')

# Escrever novas linhas no arquivo
arquivo.write('Nova linha 1\n')
arquivo.write('Nova linha 2\n')

# Fechar o arquivo
arquivo.close()
```

Figura 53

7.2. Manipulando arquivos CSV

A manipulação de arquivos CSV (Comma Separated Values) com Python é uma tarefa muito comum em diversas aplicações. O CSV é um formato de arquivo de texto simples que é frequentemente utilizado para armazenar e compartilhar dados tabulares. Em geral, cada linha do arquivo representa um registro e as colunas são separadas por vírgulas.

Para manipular arquivos CSV com Python, é necessário utilizar a biblioteca padrão CSV. Essa biblioteca fornece classes e funções para ler e gravar arquivos CSV de forma eficiente e fácil. Veja abaixo alguns exemplos práticos:

Lendo um arquivo CSV

Para ler um arquivo CSV com Python, é necessário criar um objeto reader da classe csv.reader e passar o arquivo como parâmetro. Em seguida, basta iterar sobre as linhas do arquivo e processar os dados conforme necessário. Veja um exemplo:

```
import csv

with open('exemplo.csv', 'r') as arquivo:
    leitor = csv.reader(arquivo)
    for linha in leitor:
        print(linha)
```

Figura 54

Nesse exemplo, o arquivo exemplo.csv é aberto para leitura, e o objeto csv.reader é criado com base nesse arquivo. Em seguida, cada linha do arquivo é lida e impressa na tela.

Escrevendo um arquivo CSV

Para escrever um arquivo CSV com Python, é necessário criar um objeto writer da classe csv.writer e passar o arquivo como parâmetro. Em seguida, basta escrever as linhas do arquivo utilizando o método writerow(). Veja um exemplo:

```
import csv

dados = [
    ['Nome', 'Idade', 'Cidade'],
    ['João', '25', 'São Paulo'],
    ['Maria', '30', 'Rio de Janeiro'],
    ['Pedro', '20', 'Belo Horizonte']
]

with open('exemplo.csv', 'w', newline='') as arquivo:
    escritor = csv.writer(arquivo)
    for linha in dados:
        escritor.writerow(linha)
```

Figura 55

Nesse exemplo, a variável “dados” contém os dados que serão escritos no arquivo CSV. O arquivo exemplo.csv é aberto para escrita e o objeto csv.writer é criado com base nesse arquivo. Em seguida, cada linha dos dados é escrita no arquivo utilizando o método writerow().

Manipulando dados do arquivo CSV

A biblioteca CSV também fornece algumas funções úteis para manipular os dados do arquivo CSV. Veja alguns exemplos:

```
import csv

# Lendo um valor específico do arquivo CSV
with open('exemplo.csv', 'r') as arquivo:
    leitor = csv.reader(arquivo)
    for linha in leitor:
        print(linha[0])  # Imprime o primeiro valor de cada linha

# Filtrando dados do arquivo CSV
with open('exemplo.csv', 'r') as arquivo:
    leitor = csv.DictReader(arquivo)
    for linha in leitor:
        if linha['Cidade'] == 'São Paulo':
            print(linha['Nome'])

# Adicionando dados ao arquivo CSV
dados = [['Ana', '27', 'Florianópolis'], ['Lucas', '22', 'Porto Alegre']]
with open('exemplo.csv', 'a', newline='') as arquivo:
    escritor = csv.writer(arquivo)
    for linha in dados:
        escritor
```

Figura 56

7.3. Manipulando arquivos Excel

Para manipular arquivos Excel com Python, é necessário utilizar uma biblioteca externa chamada "openpyxl". Essa biblioteca permite a leitura e escrita de arquivos no formato Excel, tanto no formato antigo (.xls) quanto no formato atual (.xlsx).

Para utilizar a biblioteca, é necessário instalá-la primeiro. Isso pode ser feito via pip, o gerenciador de pacotes do Python. Para instalar o "openpyxl", basta executar o seguinte comando no terminal:

```
pip install openpyxl
```

Figura 57

Com a biblioteca instalada, é possível abrir um arquivo Excel e manipulá-lo em Python. Por exemplo, para abrir um arquivo Excel chamado "exemplo.xlsx" e imprimir o valor da célula A1, pode-se fazer o seguinte:

```
import openpyxl

# abre o arquivo Excel
workbook = openpyxl.load_workbook('exemplo.xlsx')

# seleciona a primeira planilha
worksheet = workbook.active

# lê o valor da célula A1
valor_a1 = worksheet['A1'].value

# imprime o valor da célula A1
print(valor_a1)
```

Figura 58

Para escrever em uma célula do arquivo Excel, basta atribuir um valor a essa célula. Por exemplo, para escrever o valor "42" na célula B1, pode-se fazer o seguinte:

```
import openpyxl

# abre o arquivo Excel
workbook = openpyxl.load_workbook('exemplo.xlsx')

# seleciona a primeira planilha
worksheet = workbook.active

# escreve o valor 42 na célula B1
worksheet['B1'] = 42

# salva as alterações no arquivo Excel
workbook.save('exemplo.xlsx')
```

Figura 59

Dessa forma, o valor da célula B1 será atualizado para "42" no arquivo Excel "exemplo.xlsx".

7.4. Manipulando arquivos JSON

A manipulação de arquivos JSON com Python é bastante simples e pode ser realizada utilizando a biblioteca padrão JSON. Para trabalhar com esse padrão em Python, é necessário carregar o arquivo JSON para um objeto Python e, em seguida, manipular esses dados como qualquer outra variável Python.

Para carregar um arquivo JSON em Python, podemos utilizar a função json.load(), que lê o arquivo JSON e o converte para um objeto Python. Por exemplo, se tivermos um arquivo JSON chamado dados.json com o seguinte conteúdo:

```
{
    "nome": "João",
    "idade": 30,
    "email": "joao@example.com",
    "telefones": [
        {
            "tipo": "residencial",
            "numero": "123456789"
        },
        {
            "tipo": "celular",
            "numero": "987654321"
        }
    ]
}
```

Figura 60

Podemos carregar o arquivo JSON e acessar seus dados em Python com o seguinte código:

```
import json

# Abre o arquivo JSON
with open('dados.json') as arquivo:
    # Carrega o arquivo JSON para um objeto Python
    dados = json.load(arquivo)

# Acessa os dados do arquivo JSON como variáveis Python
nome = dados['nome']
idade = dados['idade']
email = dados['email']
telefones = dados['telefones']
```

Figura 61

No exemplo acima, utilizamos a função json.load() para carregar o arquivo dados.json e armazenamos os dados em um objeto Python chamado “dados”. Em seguida, acessamos os dados do arquivo JSON como variáveis Python, como nome, idade, e-mail e telefones.

Para escrever dados em um arquivo JSON, podemos utilizar a função json.dump(). Por exemplo, se quisermos salvar os dados de um dicionário Python em um arquivo JSON chamado dados.json, podemos utilizar o seguinte código:

```
import json

# Dados a serem salvos em um arquivo JSON
dados = {
    'nome': 'Maria',
    'idade': 25,
    'email': 'maria@example.com',
    'telefones': [
        {
            'tipo': 'residencial',
            'numero': '111111111'
        },
        {
            'tipo': 'celular',
            'numero': '222222222'
        }
    ]
}

# Salva os dados em um arquivo JSON
with open('dados.json', 'w') as arquivo:
    json.dump(dados, arquivo)
```

Figura 62

Neste exemplo, criamos um dicionário Python chamado “dados” com informações sobre uma pessoa e, em seguida, salvamos esses dados em um arquivo JSON chamado dados.json utilizando a função json.dump(). Note que o segundo parâmetro da função open() é 'w', que significa que estamos abrindo o arquivo em modo de escrita.

8. Exceções e tratamento de erros

Em Python, as exceções são usadas para lidar com erros ou comportamentos inesperados que podem ocorrer durante a execução do programa. Quando um erro ocorre, uma exceção é gerada, interrompendo a execução do programa, a menos que a exceção seja tratada adequadamente.

O tratamento de exceções em Python é realizado usando blocos try-except. O bloco try contém o código que pode gerar uma exceção. O bloco except é usado para lidar com a exceção e executar algum código alternativo. Se não houver exceções, o bloco try é executado completamente e o bloco except é ignorado.

A sintaxe básica é a seguinte:

```
try:
    # código que pode gerar exceções
except TipoDeExcecao:
    # código para tratar a exceção
```

Figura 63

Por exemplo, se quisermos tentar abrir um arquivo que não existe, podemos usar um bloco try-except para tratar a exceção FileNotFoundError. O código ficaria assim:

```
try:
    arquivo = open('arquivo_nao_existente.txt', 'r')
except FileNotFoundError:
    print('O arquivo não existe')
```

Figura 64

Assim, se o arquivo não existir, o bloco except será executado e a mensagem 'O arquivo não existe' será exibida. Se o arquivo existir, o bloco except será ignorado e o programa continuará normalmente.

Além disso, é possível usar a cláusula else para executar um código se não houver exceções e a cláusula finally para executar um código após a execução do bloco try, independentemente de haver exceções ou não. A sintaxe completa seria a seguinte:

```
try:
    # código que pode gerar exceções
except TipoDeExcecao:
    # código para tratar a exceção
else:
    # código para executar se não houver exceções
finally:
    # código para executar após a execução do bloco try
```

Figura 65

O tratamento adequado de exceções pode melhorar muito a robustez e a confiabilidade de um programa Python.

Referências

BANIN, Sérgio Luiz. **Python 3 - Conceitos e Aplicações - Uma Abordagem Didática [BV:MB]**. São Paulo: Érica, 2018.

FORBELLONE, André L. V.; Eberspacher, Henri F. **Lógica de Programação a construção de algoritmos e estruturas de dados [BV:PE]**. 3. ed. São Paulo: Editora Pearson, 2005.

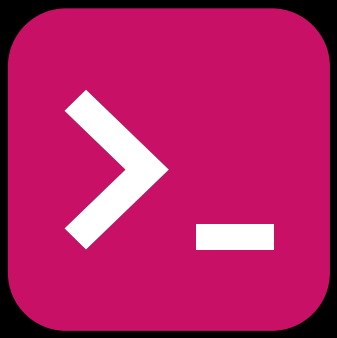
LIMA, Janssen dos Reis. **Consumindo a API do Zabbix com Python [BV:PE].** Rio de Janeiro: Editora Brasport, 2016.

PERKOVIC, Ljubomir. **Introdução à Computação Usando Python - Um Foco no Desenvolvimento de Aplicações [BV:MB]**. 1ª ed. Rio de Janeiro: LTC, 2016.

SEBESTA,Robert W. **Conceitos de Linguagens de Programação**. 11. ed. Porto Alegre: Grupo A, 2011.

TUCKER, Allen; NOONAN, Robert. **Linguagens de Programação: Princípios e Paradigmas**. Porto Alegre: Grupo A.

VLADISHEV, A. **Consumindo a API do Zabbix com Python**. Rio de Janeiro: Brasport.



Digital College

ENSINO DE HABILIDADES DIGITAIS

@digitalcollegebr

/school/digitalcollegebr

/digitalcollegebr

digitalcollege.com.br