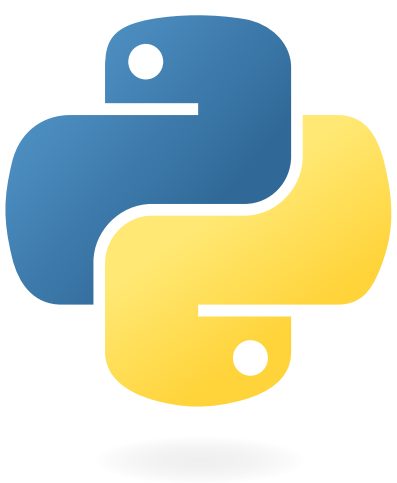
Introdução à Programação em



python™

https://evandro-crr.github.io/intro-python

Evandro Chagas Ribeiro da Rosa

Arquivos e Exceção

- Manipulação de Arquivos de Texto
- Tratamento de Exceções
- Arquivos JSON

Arquivos 

Um arquivo é uma **coleção de dados** armazenados. Ele pode conter diferentes tipos de informações, como texto, imagens, áudio ou dados binários.

Os arquivos são organizados de maneira sequencial ou estruturada, permitindo que programas acessem, leiam e modifiquem seu conteúdo de forma persistente, mesmo após o encerramento da aplicação.

Abrindo Arquivos

- Em Python, usamos a função `open()` para abrir arquivos.
- O modo de abertura pode ser:
  - `"r"` : leitura (padrão).
  - `"w"` : escrita (sobrescreve o conteúdo).
  - `"a"` : anexar (adiciona ao final do arquivo).
  - `"b"` : modo binário (usado para arquivos não textuais).

```
arquivo = open("dados.txt", "r")
```

Lendo Arquivos

- Existem diferentes formas de ler o conteúdo de um arquivo:

```
# Lendo o arquivo inteiro
conteudo = arquivo.read()

# Lendo uma linha
linha = arquivo.readline()

# Lendo todas as linhas em uma lista
linhas = arquivo.readlines()
```

- Após a leitura, sempre feche o arquivo com `arquivo.close()` ou utilize o gerenciador de contexto.

Escrevendo em Arquivos

- Para escrever em arquivos, devemos abri-los no modo de escrita `"w"` ou anexar `"a"`.

```
arquivo = open("saida.txt", "w")
arquivo.write("Linha de exemplo\n")
arquivo.close()
```

- Usar o modo `"a"` permite adicionar conteúdo ao final do arquivo sem sobrescrevê-lo.

Gerenciador de Contexto: `with`

- A maneira recomendada de manipular arquivos em Python é com o **gerenciador de contexto** (`with`), que garante o fechamento automático do arquivo.

```
with open("dados.txt", "r") as arquivo:
    conteudo = arquivo.read()
```

- Dessa forma, o arquivo é fechado automaticamente ao final do bloco `with` , mesmo que ocorra uma exceção.

Manipulação de Arquivos Binários

- Arquivos binários, como imagens, devem ser abertos com o modo `"b"`:

```
with open("imagem.png", "rb") as arquivo:
    dados_binarios = arquivo.read()
```

- Da mesma forma, para escrever em um arquivo binário, usamos `"wb"`.

Exemplo Completo

Aqui está um exemplo completo de leitura e escrita de arquivos:

```
# Escrevendo dados em um arquivo
with open("saida.txt", "w") as arquivo:
    arquivo.write("Primeira linha\n")
    arquivo.write("Segunda linha\n")

# Lendo o conteúdo do arquivo
with open("saida.txt", "r") as arquivo:
    conteudo = arquivo.read()
    print(conteudo)
```

Erro ao Abrir Arquivo Inexistente

Quando tentamos abrir um arquivo que não existe,
o Python lança uma exceção do tipo **FileNotFoundError**.

Exemplo de Código:

```
# Tentando abrir um arquivo que não existe
with open("dados.txt", "r") as arquivo:
    conteudo = arquivo.read()
```

Resultado:

```
Traceback (most recent call last):
  File "<python-input-3>", line 1, in <module>
    with open("dados.txt", "r") as arquivo:
         ~~~~^^^^^^^^^^^^^^^^^^
FileNotFoundError: [Errno 2] No such file or directory: 'dados.txt'
```

Como Evitar o Erro?

Usar **tratamento de exceção** para evitar que o
programaquebre ao tentar abrir arquivos inexistentes.

O que é uma Exceção?

- Em Python, **exceções** são erros detectados durante a execução de um programa.
- Quando uma exceção ocorre, o fluxo normal de execução do programa é interrompido.
- Exemplo de exceções:
  - `ZeroDivisionError` : divisão por zero.
  - `FileNotFoundError` : arquivo não encontrado.
  - `ValueError` : tipo de dado incorreto.

Lidando com Exceções: `try` e `except`

- O Python oferece o bloco `try` e `except` para tratar exceções.

```
try:
    # Código que pode gerar uma exceção
    x = 1 / 0
except ZeroDivisionError:
    # Código que será executado se ocorrer uma exceção
    print("Erro: Divisão por zero!")
```

Exemplo: Tratamento de Erros em Entrada de Dados

- Aqui está um exemplo de como usar `try` e `except` para lidar com entradas inválidas.

```
try:
    numero = int(input("Digite um número: "))
except ValueError:
    print("Erro: Entrada inválida, por favor insira um número válido.")
```

Múltiplos Blocos `except`

- Podemos tratar diferentes tipos de exceções com múltiplos blocos `except`.

```
try:
    numero = int(input("Digite um número: "))
    resultado = 10 / numero
except ValueError:
    print("Erro: Entrada inválida.")
except ZeroDivisionError:
    print("Erro: Divisão por zero!")
```

Bloco `else` e `finally`

- O bloco `else` é executado se nenhuma exceção for lançada.
- O bloco `finally` é sempre executado, independentemente de uma exceção ter ocorrido.

```
try:
    f = open("arquivo.txt")
    conteudo = f.read()
except FileNotFoundError:
    print("Erro: Arquivo não encontrado.")
else:
    print("Leitura do arquivo com sucesso!")
finally:
    f.close()
```

Lançando Exceções com `raise`

- Podemos lançar exceções explicitamente usando a palavra-chave `raise`.

```
def dividir(a, b):
    if b == 0:
        raise ZeroDivisionError("Divisão por zero não é permitida!")
    return a / b

try:
    dividir(10, 0)
except ZeroDivisionError as e:
    print(e)
```

Criando Exceções Personalizadas

- Podemos criar nossas próprias exceções definindo classes que herdam de `Exception`.

```
class ErroPersonalizado(Exception):
    pass

def validar(valor):
    if valor < 0:
        raise ErroPersonalizado("Valor negativo não permitido!")

try:
    validar(-10)
except ErroPersonalizado as e:
    print(e)
```

Tipos de Exceções em Python

Exceção	Descrição
`SyntaxError`	Ocorre quando há erro de sintaxe no código.
`IndexError`	Levantada quando se tenta acessar um índice de lista ou tupla inexistente.
`KeyError`	Ocorre quando se tenta acessar uma chave inexistente em um dicionário.
`ValueError`	Levantada quando um valor de tipo incorreto é passado para uma operação.
`TypeError`	Ocorre quando uma operação é aplicada a um objeto de tipo inadequado.
`FileNotFoundError`	Ocorre quando se tenta acessar um arquivo que não existe.
`ZeroDivisionError`	Levantada quando ocorre uma divisão por zero.
`NameError`	Ocorre quando uma variável é referenciada antes de ser definida.
`AttributeError`	Ocorre quando se tenta acessar um atributo que não existe.
`IOError`	Levantada em erros de entrada e saída, como ao tentar ler ou gravar um arquivo.

Boas Práticas

- Evite capturar exceções genéricas com `except Exception`.
- Seja específico ao capturar exceções para garantir que apenas erros esperados sejam tratados.
- Use `finally` para garantir a liberação de recursos (como arquivos abertos ou conexões).

```
from json import JSONDecodeError, load

try:
    arquivo = open("dados.json", "r")
    dados = load(arquivo)
    print("Dados carregados:", dados)

except (FileNotFoundError, JSONDecodeError):
    print("Erro: O arquivo não foi"\
          "encontrado ou está corrompido!")

finally:
    print("Fechando o arquivo.")
    arquivo.close()
```

Por que Usar Tratamento de Exceções?

- **Robustez**: Tratamento de exceções permite que programas lidem com erros de forma controlada, evitando que o sistema "quebre".
- **Prevenção de Falhas**: Sem exceções, erros inesperados, como arquivo inexistente ou dados corrompidos, podem causar falhas graves no programa.
- **Usuário Informado**: Ao capturar exceções, podemos fornecer *mensagens claras* e úteis para o usuário, ao invés de permitir que erros técnicos interrompam a execução.

Manipulação de Arquivo JSON

O que é JSON?

- **JSON** (JavaScript Object Notation) é um formato de troca de dados.
- É amplamente utilizado para comunicação entre sistemas e armazenamento de dados.
- Em Python, a biblioteca padrão `json` permite manipular arquivos e dados em JSON.

Exemplo de JSON:

```
{
  "nome": "Carlinhos",
  "idade": 47,
  "cidades": ["São José", "Florianópolis"]
}
```

Salvando Dados em JSON com `json.dump()`

- Para salvar dados em formato JSON, usamos o método `json.dump()`.
- Convertendo um dicionário Python em um arquivo JSON:

```python
import json

dados = {
    "nome": "Maria",
    "idade": 74,
    "cidades": ["Curitiba", "Porto Alegre"]
}

# Salvando em um arquivo JSON
with open("dados.json", "w") as arquivo:
    json.dump(dados, arquivo)
```

Carregando Arquivos JSON com `json.load()`

- Para carregar dados de um arquivo JSON, usamos o método `json.load()`.
- Convertendo o JSON de volta para um dicionário Python:

```
import json

# Lendo o arquivo JSON
with open("dados.json", "r") as arquivo:
    dados_carregados = json.load(arquivo)

print(dados_carregados)
# {'nome': 'Maria', 'idade': 74, 'cidades': ['Curitiba', 'Porto Alegre']}
```

Convertendo Strings JSON com `json.dumps()` e `json.loads()`

Para converter diretamente entre strings JSON e dicionários:

- `json.dumps()` : converte um dicionário em uma string JSON.
- `json.loads()` : converte uma string JSON em um dicionário.

```
import json
# Convertendo dicionário para string JSON
dados_str = json.dumps(dados)
print(dados_str)
# {"nome": "Maria", "idade": 25, "cidades": ["Curitiba", "Porto Alegre"]}
# Convertendo string JSON de volta para dicionário
dados_dict = json.loads(dados_str)
print(dados_dict)
# {'nome': 'Maria', 'idade': 25, 'cidades': ['Curitiba', 'Porto Alegre']}
```

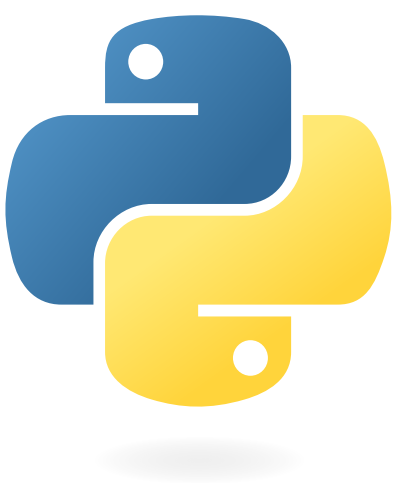
Formatação com `indent`

- Podemos formatar o JSON de forma legível com o parâmetro `indent` no método `json.dump()`:

```
with open("dados_formatado.json", "w") as arquivo:
    json.dump(dados, arquivo, indent=4)
```

- Isso gera um arquivo mais fácil de ler, com indentação e quebras de linha.

Introdução à Programação em



python™

https://evandro-crr.github.io/intro-python

Evandro Chagas Ribeiro da Rosa