

TRATAMENTO DE EXCEÇÕES E OUTRAS OPERAÇÕES

Conceito de exceções

Tratamento de exceções



Quando trabalhamos com arquivos, é comum encontrarmos alguns problemas, como arquivo inexistente e falta de permissão para escrever em um. A maioria desses problemas só pode ser detectada durante a execução do programa.

Quando uma falha inesperada ocorre e o interpretador não consegue resolver o problema, dizemos que houve uma exceção.

Nesses casos, precisamos informar ao interpretador como tratar a exceção, para que o programa não seja interrompido.

Se a exceção é um problema inesperado, como tratá-la?

Ao desenvolver um programa, precisamos procurar na documentação da biblioteca, do módulo ou da própria linguagem de programação se as funcionalidades que vamos utilizar têm exceções mapeadas. Essas exceções são problemas que podem ocorrer, e é nossa tarefa tratá-las.

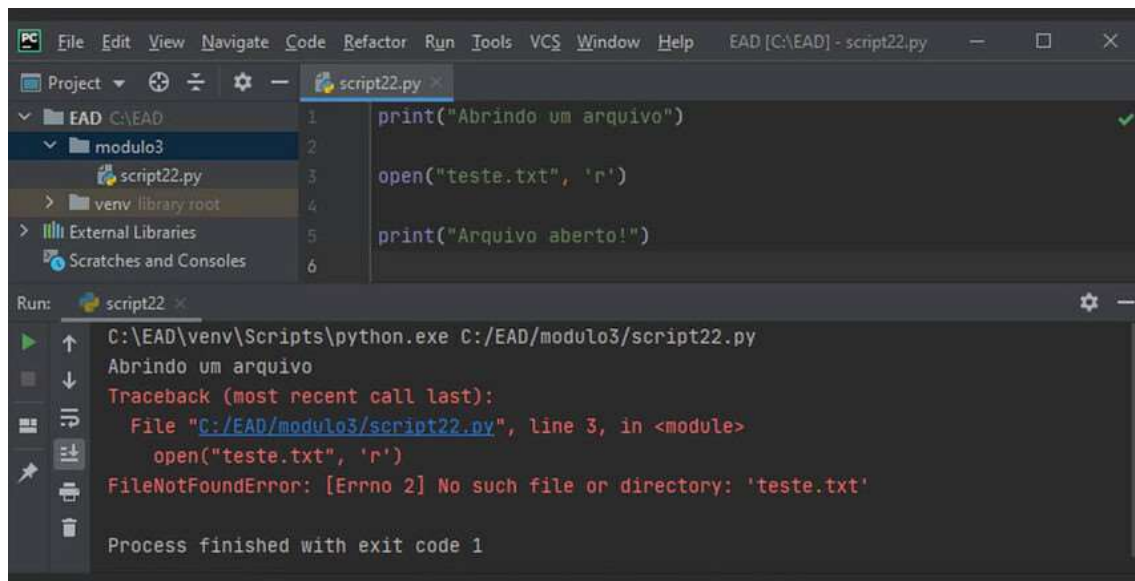
Você deve estar se perguntando: “O que seria ‘tratar uma exceção?’”. Isso nada mais é do que dizer ao Python o que fazer, ou quais instruções executar, quando ele encontrar um problema.



Para ilustrar, observe os seguintes passos:

1. Quando abrimos um arquivo em modo leitura e esse arquivo não existe, o Python lança uma exceção do tipo `FileNotFoundError`.
2. Se **não** avisarmos ao Python o que fazer quando isso ocorrer, o programa será interrompido.
3. Nesse caso, um tratamento para essa exceção poderia ser: exibir um pop-up ao usuário informando que o arquivo não existe.

Veja o que acontece quando uma exceção lançada não é tratada:



The screenshot shows an IDE window titled "EAD [C:\EAD] - script22.py". The left sidebar displays a project tree with folders "EAD C:\EAD" and "modulo3", and files "script22.py" and "venv\library root". The main editor shows the contents of "script22.py":

```
1 print("Abrindo um arquivo")
2
3 open("teste.txt", 'r')
4
5 print("Arquivo aberto!")
6
```

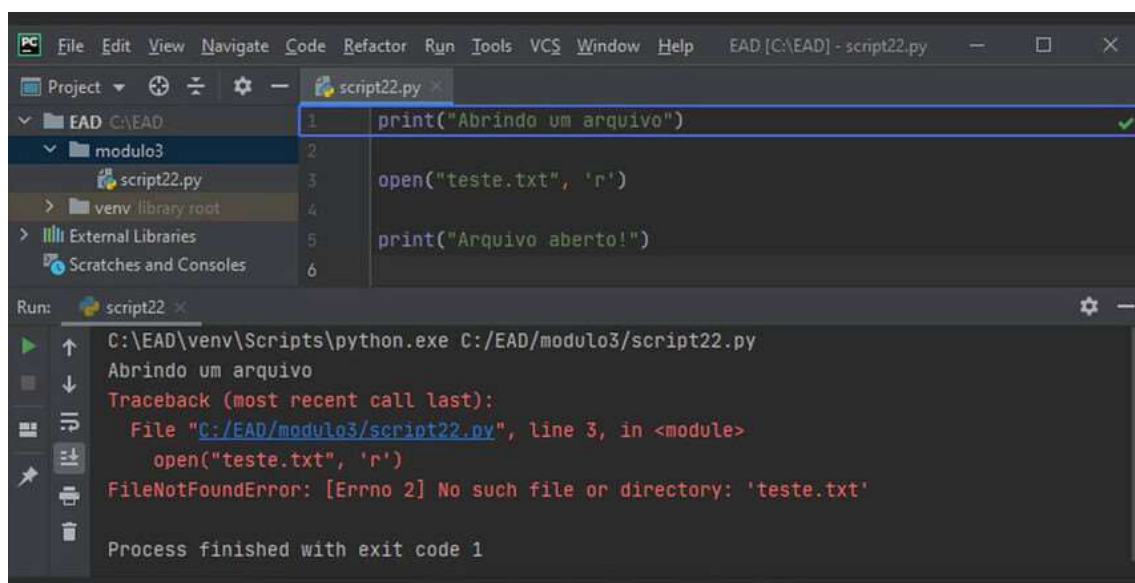
Below the editor, the "Run" console shows the command executed: `C:\EAD\venv\Scripts\python.exe C:/EAD/modulo3/script22.py`. The output is:

```
Abrindo um arquivo
Traceback (most recent call last):
  File "C:/EAD/modulo3/script22.py", line 3, in <module>
    open("teste.txt", 'r')
FileNotFoundError: [Errno 2] No such file or directory: 'teste.txt'

Process finished with exit code 1
```

Script 22 e saída.

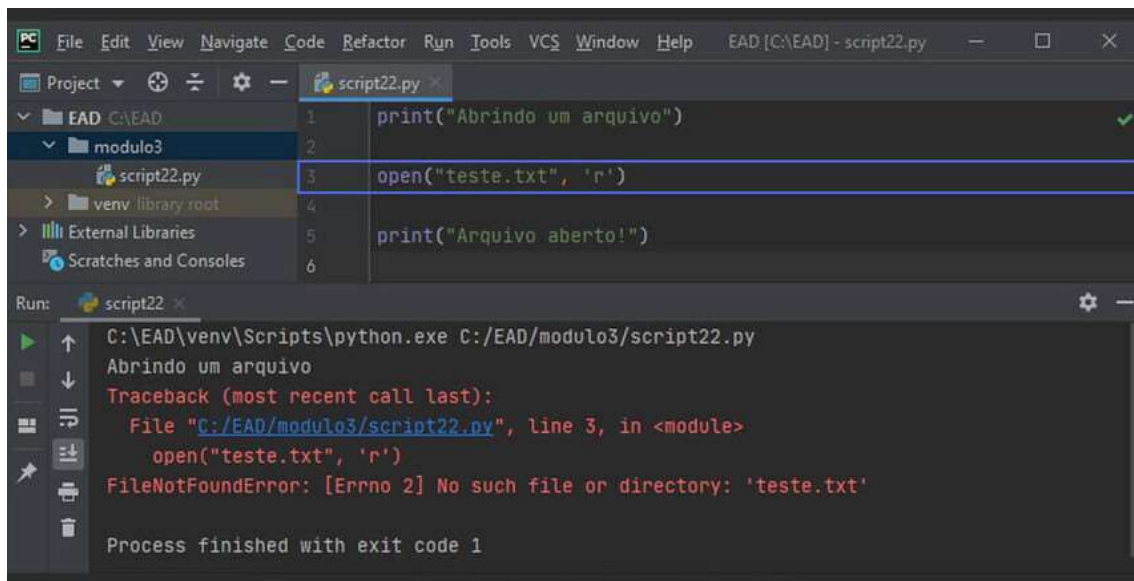
Temos o script 22 e sua saída.



This screenshot is identical to the one above, showing the same IDE window with the Python script "script22.py" and its execution output in the "Run" console. The script attempts to open a file named "teste.txt", which does not exist, resulting in a `FileNotFoundError`.

Impressão da string "Abrindo um arquivo".

Imprimimos, na linha 1, a string "Abrindo um arquivo". Essa impressão serve para acompanhar a execução do programa no console.



The screenshot shows an IDE window with a project named 'EAD'. The file explorer on the left shows a directory structure with 'modulo3' containing 'script22.py'. The code editor shows the following Python code:

```
1 print("Abrindo um arquivo")
2
3 open("teste.txt", 'r')
4
5 print("Arquivo aberto!")
6
```

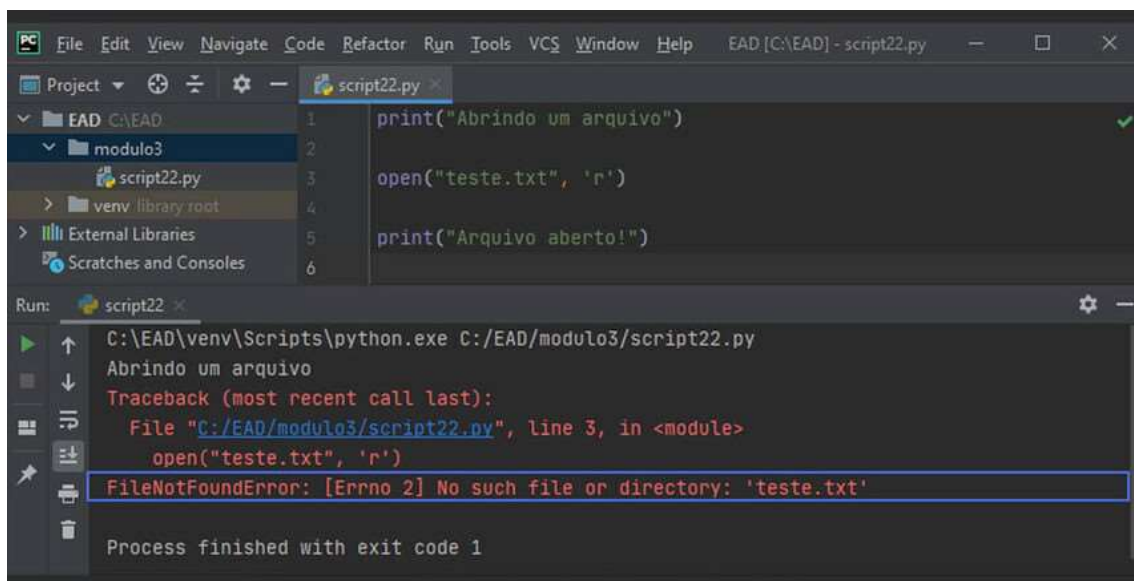
The Run console at the bottom shows the execution of the script. It displays the output 'Abrindo um arquivo' and then a traceback error:

```
Traceback (most recent call last):
  File "C:/EAD/modulo3/script22.py", line 3, in <module>
    open("teste.txt", 'r')
FileNotFoundError: [Errno 2] No such file or directory: 'teste.txt'
```

The process finished with exit code 1.

Abertura do arquivo teste.txt no modo leitura.

Abrimos, na linha 3, o arquivo teste.txt no modo leitura. Esse arquivo não existe no diretório modulo3 no qual estamos trabalhando, como podemos observar pela árvore de diretórios à esquerda da figura.

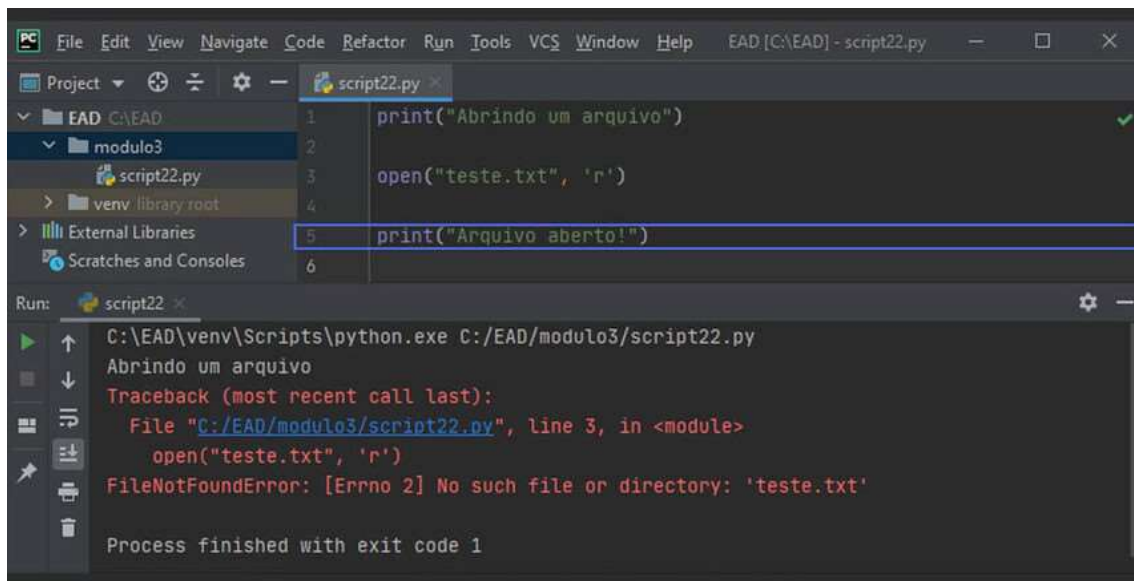


This screenshot is identical to the one above, showing the same code and execution output. In this version, the error message in the Run console is highlighted with a blue selection box:

```
FileNotFoundError: [Errno 2] No such file or directory: 'teste.txt'
```

Criação de um contador e aplicação do método *split*.

Recebemos um erro, ao executarmos o programa, destacado em vermelho no console. O nome do erro é `FileNotFoundError`, e a sua descrição é `No such file or directory` (em tradução literal, não existe tal arquivo ou diretório), seguido do nome do arquivo que apresentou o erro, teste.txt.



```
1 print("Abrindo um arquivo")
2
3 open("teste.txt", 'r')
4
5 print("Arquivo aberto!")
6
```

Run: script22

```
C:\EAD\venv\Scripts\python.exe C:/EAD/modulo3/script22.py
Abrindo um arquivo
Traceback (most recent call last):
  File "C:/EAD/modulo3/script22.py", line 3, in <module>
    open("teste.txt", 'r')
FileNotFoundError: [Errno 2] No such file or directory: 'teste.txt'

Process finished with exit code 1
```

Exceção gerada.

Essa exceção foi gerada, ou lançada, de acordo com o linguajar da computação, pois o Python não sabia qual caminho tomar ao encontrar esse problema. Observe que a linha 5 não foi executada, pois o programa parou sua execução assim que o problema foi encontrado.

Para resolver isso, precisamos tratar a exceção, ou melhor, uma possível exceção. Esse tratamento informa ao Python uma rota alternativa, caso ele encontre um problema.

Para tratar exceções, precisamos “envolver” o trecho de código passível de erro com a cláusula `try/except` ou `try/except/finally`. Veremos apenas a cláusula `try/except`.

Veja a sintaxe a seguir:

```
try:
    # código que pode gerar uma exceção
    ...
except Erro1 as erro:
    # código alternativo caso Erro1
    print(erro)
except Erro2 as erro:
    # código alternativo caso Erro2
    print(erro)
...
```

Sintaxe try/except.

O código crítico que desejamos executar deve estar no escopo do *try*, enquanto o código alternativo, que será executado em caso de erro, deve estar no escopo do **except**.

Uma mesma operação pode lançar mais de um tipo diferente de exceção, em que, para cada tipo, Erro1 e Erro2, devemos ter uma cláusula **except** específica.

No exemplo da imagem, a exceção está disponível por meio da variável *erro*, de onde podemos extrair mais informações, como veremos a seguir.

Praticamente todas as exceções em Python são herdadas da classe *Exception*, ou seja, ela é uma exceção muito genérica, lançada por diversos tipos de erros diferentes. Quanto mais genérica, mais abrangente é a exceção.

Atenção!

Não é uma boa prática utilizar exceções abrangentes, pois elas podem silenciar erros que não esperamos. O ideal é tratar as exceções utilizando a forma mais específica possível.

Confira algumas exceções específicas relacionadas à manipulação de arquivos e alguns motivos que podem gerar essas exceções:

PermissionError

Lançada quando não temos permissão para realizar uma operação.

FileExistsError

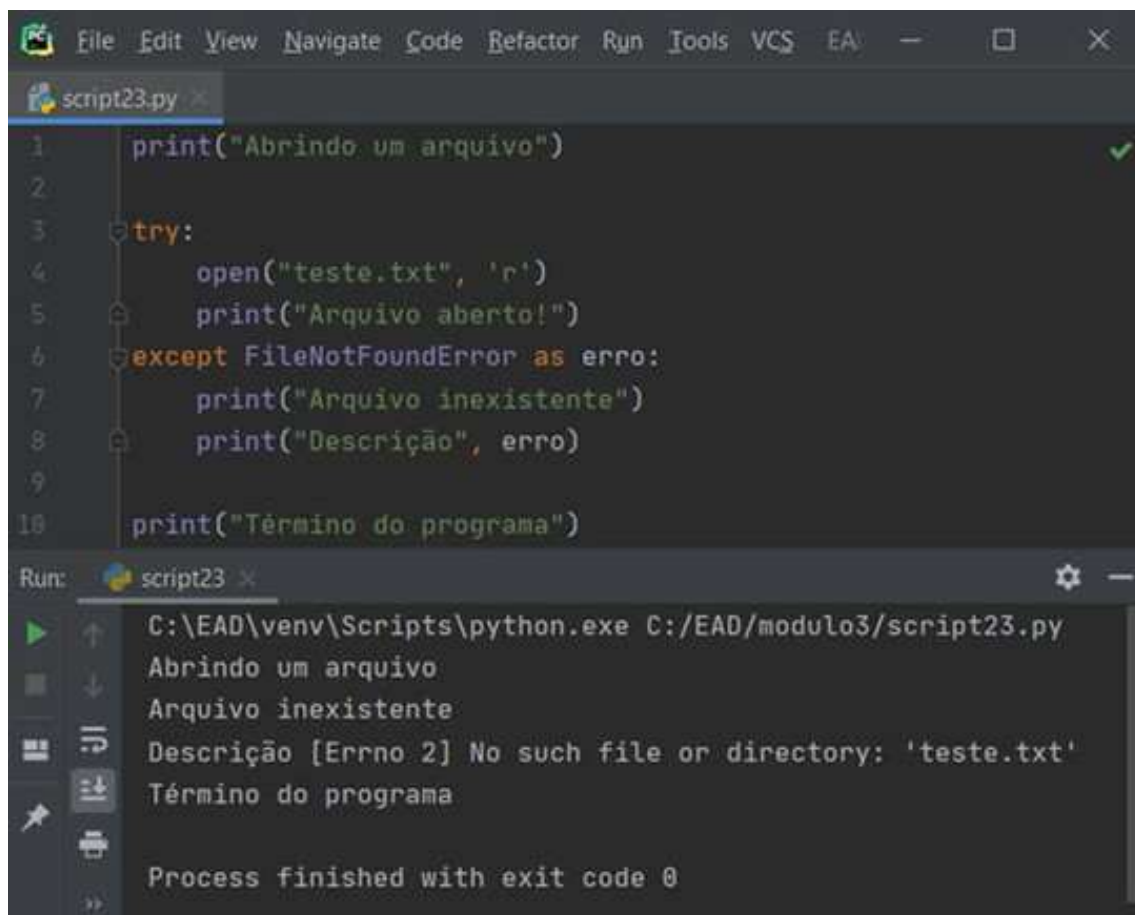
Lançada quando tentamos criar um arquivo ou diretório já existentes.

FileNotFoundError

Lançada quando tentamos abrir um arquivo ou diretório que não existem.

Todas essas exceções herdam da exceção mais abrangente `OSError`, que, por sua vez, herda de *Exception*.

Observe o exemplo a seguir, onde vamos tratar a exceção do exemplo anterior, utilizando a exceção específica mais indicada: `FileNotFoundError`:



The screenshot shows an IDE window with a file named `script23.py`. The code in the editor is as follows:

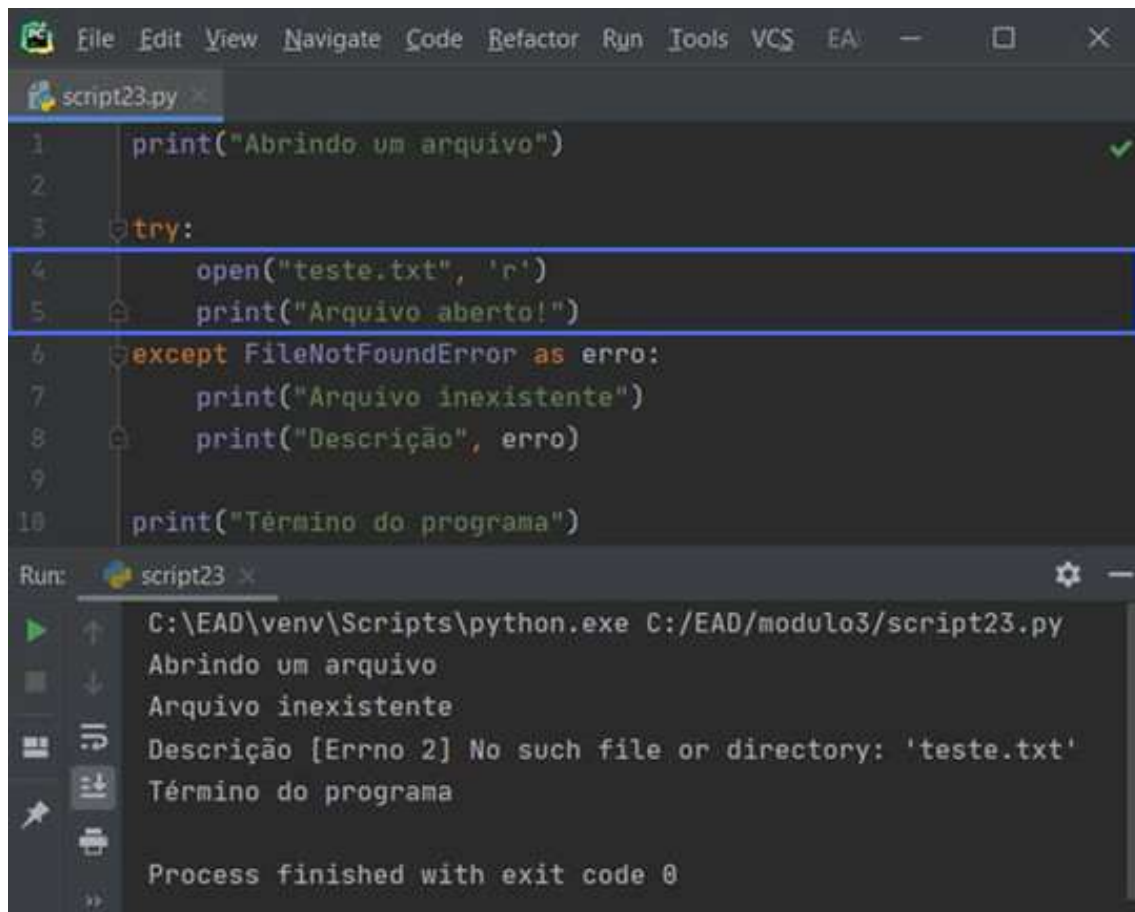
```
1 print("Abrindo um arquivo")
2
3 try:
4     open("teste.txt", 'r')
5     print("Arquivo aberto!")
6 except FileNotFoundError as erro:
7     print("Arquivo inexistente")
8     print("Descrição", erro)
9
10 print("Término do programa")
```

Below the editor, the 'Run' console shows the execution of the script. The output is:

```
Run: script23
C:\EAD\venv\Scripts\python.exe C:/EAD/modulo3/script23.py
Abrindo um arquivo
Arquivo inexistente
Descrição [Errno 2] No such file or directory: 'teste.txt'
Término do programa
Process finished with exit code 0
```

Script 23 e saída.

Temos o script 23 e sua saída.



The image shows a screenshot of an IDE window with a dark theme. The top menu bar includes File, Edit, View, Navigate, Code, Refactor, Run, Tools, VCS, and EA. The editor displays a file named 'script23.py' with the following Python code:

```
1 print("Abrindo um arquivo")
2
3 try:
4     open("teste.txt", 'r')
5     print("Arquivo aberto!")
6 except FileNotFoundError as erro:
7     print("Arquivo inexistente")
8     print("Descrição", erro)
9
10 print("Término do programa")
```

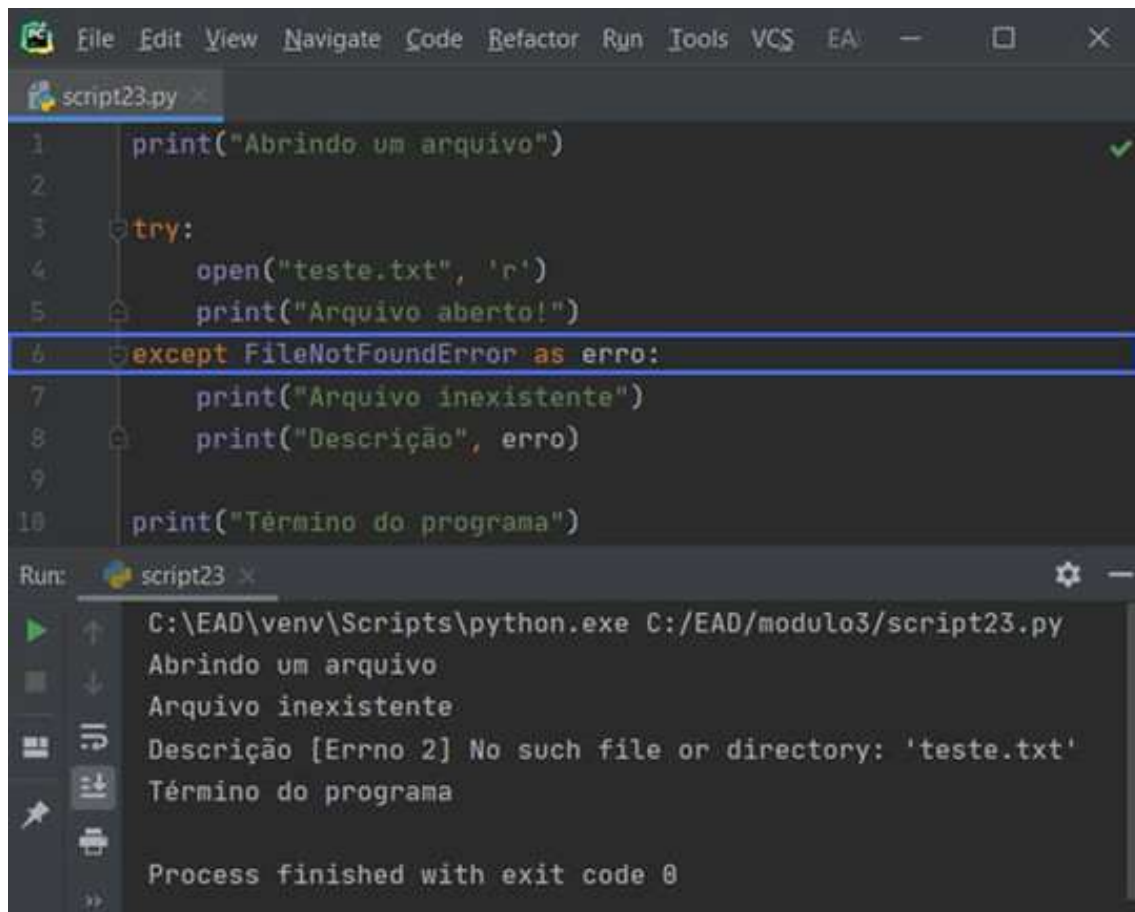
The lines 4 and 5 are highlighted with a blue selection bar. Below the editor is a 'Run' panel showing the execution of 'script23.py' using 'C:\EAD\venv\Scripts\python.exe'. The output is as follows:

```
C:\EAD\venv\Scripts\python.exe C:/EAD/modulo3/script23.py
Abrindo um arquivo
Arquivo inexistente
Descrição [Errno 2] No such file or directory: 'teste.txt'
Término do programa

Process finished with exit code 0
```

Indentação das linhas 4 e 5.

“Envolvemos” o código que pode gerar problema com o try. Para isso, indentamos as linhas 4 e 5.



The screenshot shows an IDE window with a file named `script23.py`. The code is as follows:

```
1 print("Abrindo um arquivo")
2
3 try:
4     open("teste.txt", 'r')
5     print("Arquivo aberto!")
6 except FileNotFoundError as erro:
7     print("Arquivo inexistente")
8     print("Descrição", erro)
9
10 print("Término do programa")
```

Below the code editor is a 'Run' console window. It shows the command executed: `C:\EAD\venv\Scripts\python.exe C:/EAD/modulo3/script23.py`. The output of the program is:

```
Abrindo um arquivo
Arquivo inexistente
Descrição [Errno 2] No such file or directory: 'teste.txt'
Término do programa

Process finished with exit code 0
```

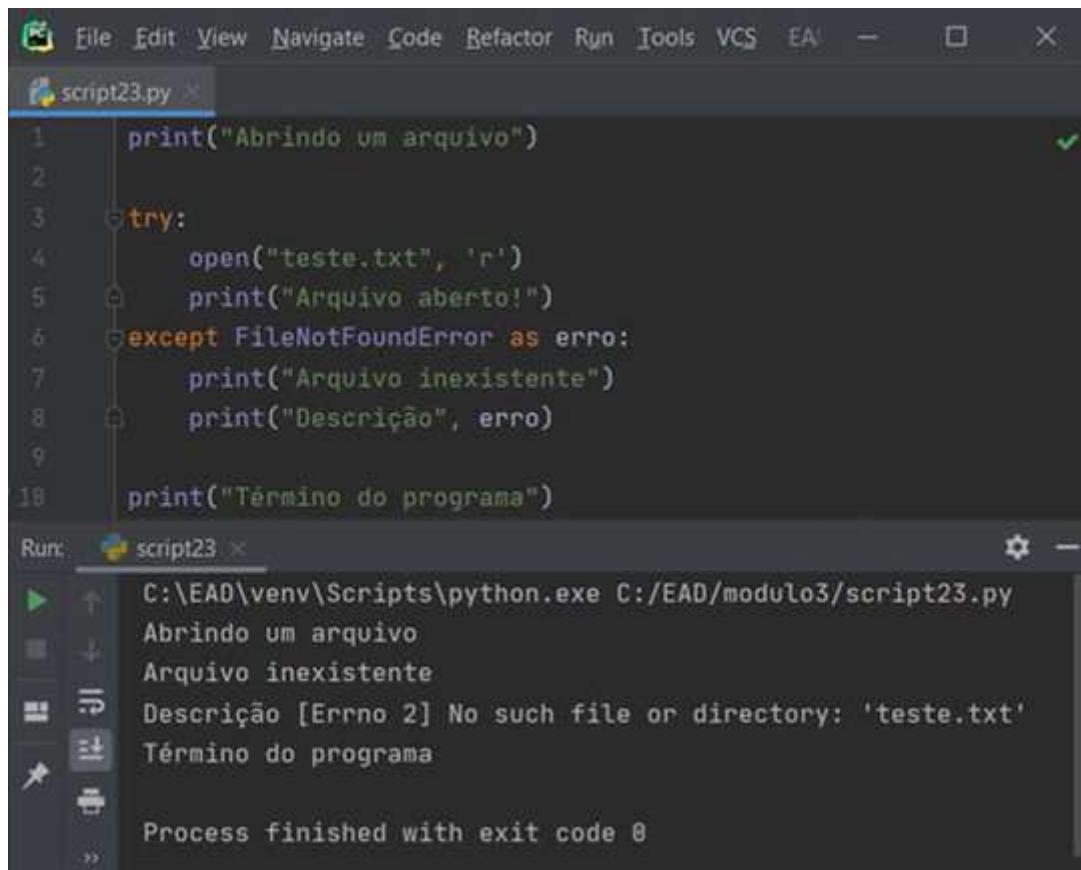
Indentação das linhas 7 e 8.

Precisamos, para tratar o erro, explicitar qual o tipo que vamos tratar. Nesse caso, `FileNotFoundError`, como descrito na linha 6. Indentamos as linhas 7 e 8 para indicar que elas fazem parte do escopo da exceção explicitada na linha 6.

Durante a execução do programa, ao executar a linha 4, o Python encontra um erro, pois tentamos abrir o arquivo `teste.txt` para leitura, mas ele não existe. Como este código está dentro do escopo do `try`, o interpretador interrompe imediatamente a execução do código contido nesse escopo e inicia a execução do código do `except` correspondente ao erro `FileNotFoundError`. Ou seja, a execução salta da linha 4 para a linha 7.

Na linha 7, imprimimos a mensagem "Arquivo inexistente" e, na linha 8, imprimimos o problema encontrado, disponível na variável **erro**.

Observe a sequência de execução pelas saídas no console:

The image shows a screenshot of a code editor window with a dark theme. The editor displays a Python script named 'script23.py'. The code is as follows:

```
1 print("Abrindo um arquivo")
2
3 try:
4     open("teste.txt", 'r')
5     print("Arquivo aberto!")
6 except FileNotFoundError as erro:
7     print("Arquivo inexistente")
8     print("Descrição", erro)
9
10 print("Término do programa")
```

Line 4 is highlighted with a green checkmark. Below the editor is a 'Run' console window. It shows the command executed: 'C:\EAD\venv\Scripts\python.exe C:/EAD/modulo3/script23.py'. The output of the script is displayed below the command:

```
Abrindo um arquivo
Arquivo inexistente
Descrição [Errno 2] No such file or directory: 'teste.txt'
Término do programa
```

At the bottom of the console, it says 'Process finished with exit code 0'.

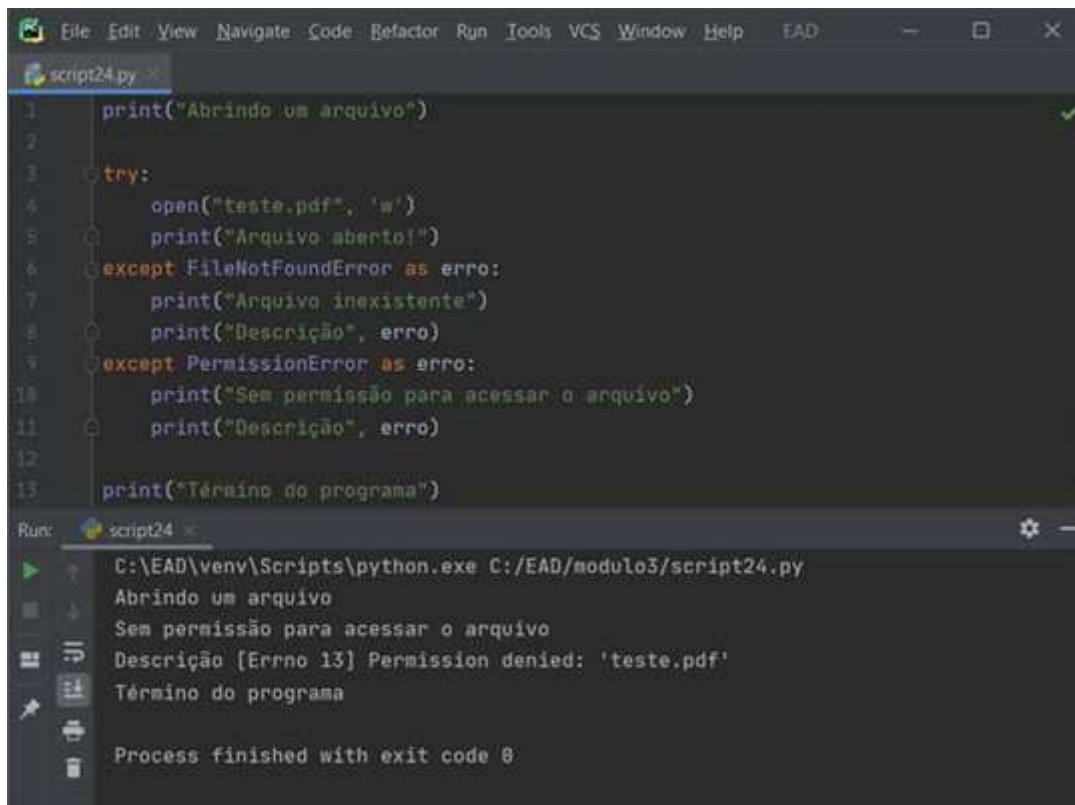
Início da execução do código do *except*.

Saiba que o Python só consegue tratar a exceção caso o erro esteja mapeado em algum *except*. Se o interpretador não encontrar o *except* adequado, será gerado um erro, e o programa será interrompido.

Um problema clássico que ocorre quando lidamos com arquivos é tentar alterar o conteúdo de um arquivo quando ele está aberto em outro programa. No caso do sistema operacional Windows 10, é lançada uma exceção sobre permissão de acesso.

A seguir, vamos criar mais um *except* para tratar o caso de não termos permissão para abrir um arquivo, mostrando o tratamento do problema levantado no parágrafo anterior.

Neste exemplo, tentamos abrir o arquivo teste.pdf para **escrita**, linha 4, porém ele já está aberto em outro programa:



The screenshot shows an IDE window titled 'script24.py'. The code is as follows:

```
1 print("Abrindo um arquivo")
2
3 try:
4     open("teste.pdf", 'w')
5     print("Arquivo aberto!")
6 except FileNotFoundError as erro:
7     print("Arquivo inexistente")
8     print("Descrição", erro)
9 except PermissionError as erro:
10    print("Sem permissão para acessar o arquivo")
11    print("Descrição", erro)
12
13 print("Término do programa")
```

Below the code editor is a 'Run' console window. It shows the command executed: `C:\EAD\venv\Scripts\python.exe C:/EAD/modulo3/script24.py`. The output is:

```
Abrindo um arquivo
Sem permissão para acessar o arquivo
Descrição [Errno 13] Permission denied: 'teste.pdf'
Término do programa

Process finished with exit code 0
```

Script 24 e sua saída.

Observe que o fluxo de execução do programa saltou da linha 4 para a linha 10. Na linha 10, temos o início do tratamento da exceção `PermissionError`, que foi justamente a exceção lançada pelo Python, impressa pela linha 11, e que pode ser verificada no console.

O Python direciona o fluxo de execução para o trecho onde é realizado o tratamento da exceção lançada.

Operações adicionais em arquivos

Além das opções para leitura e escrita em arquivos, o Python disponibiliza um conjunto de operações adicionais, como renomear e apagar arquivo, além de operações em diretórios, como listar arquivos de diretórios, criar diretórios etc.

A partir de agora, apresentaremos algumas dessas operações.

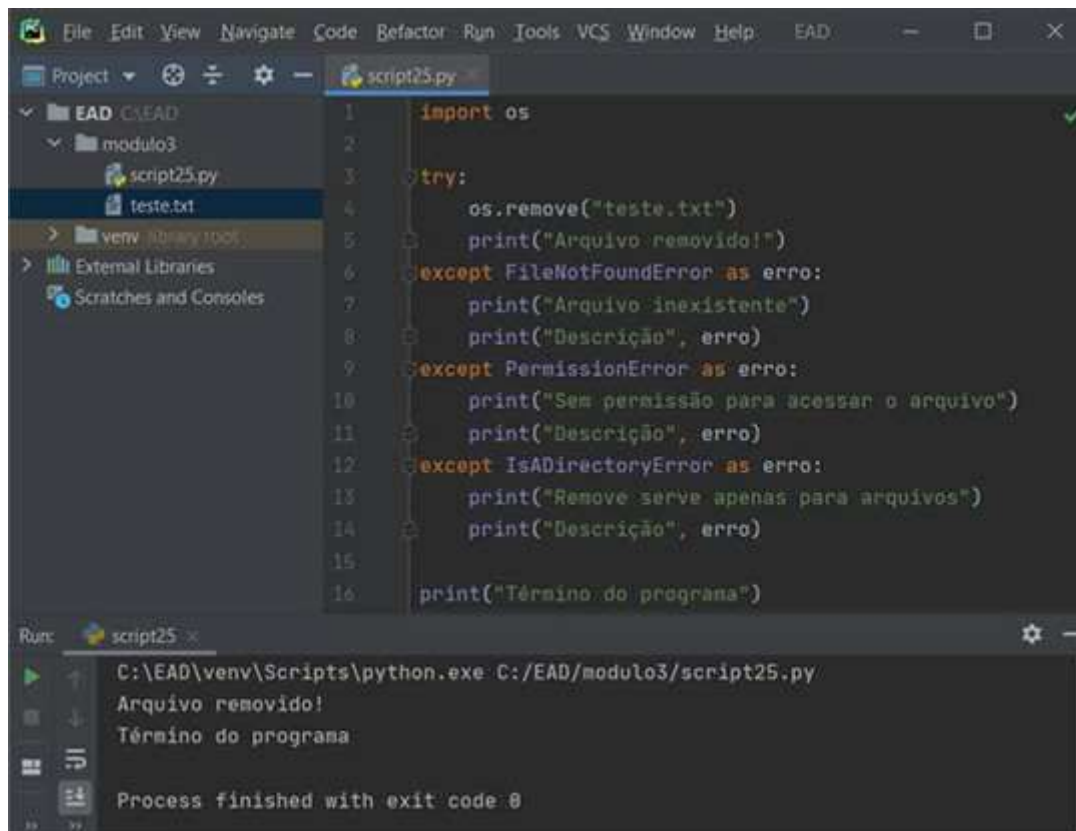
Vamos iniciar pela operação de remover um arquivo, que está disponível por meio da função **remove** do módulo **os** do Python.

A função **remove** tem a seguinte sintaxe:

```
os.remove(caminho)
```

Nesse exemplo, temos o nome do módulo **os**, seguido de um ponto e o nome da função **remove**. Como parâmetro, a função espera o caminho para um **arquivo**. Para remover diretório, devemos utilizar outra função, **rmdir**.

Veja a função **remove** na imagem abaixo. Iniciamos o script com a importação do módulo **os**, na linha 1. Aqui vamos remover o arquivo **teste.txt**, que se encontra no mesmo diretório do nosso script. Observe a árvore de diretórios à esquerda:



The screenshot shows an IDE window with a project structure on the left and a Python script in the center. The project structure includes a folder 'EAD' containing 'modulo3', which contains 'script25.py' and 'teste.txt'. The script 'script25.py' is open in the editor, showing the following code:

```
1 import os
2
3 try:
4     os.remove("teste.txt")
5     print("Arquivo removido!")
6 except FileNotFoundError as erro:
7     print("Arquivo inexistente")
8     print("Descrição", erro)
9 except PermissionError as erro:
10    print("Sem permissão para acessar o arquivo")
11    print("Descrição", erro)
12 except IsADirectoryError as erro:
13    print("Remove serve apenas para arquivos")
14    print("Descrição", erro)
15
16 print("Término do programa")
```

Below the editor, the 'Run' console shows the execution of the script. The command executed is 'C:\EAD\venv\Scripts\python.exe C:/EAD/modulo3/script25.py'. The output is 'Arquivo removido!' followed by 'Término do programa'. The console also indicates 'Process finished with exit code 0'.

Script 25 e sua saída.

Na linha 2, utilizamos a função **remove**, passando como argumento o caminho do arquivo que desejamos remover. Como estamos no mesmo diretório, utilizamos apenas o nome do arquivo. Pronto! Isso é suficiente para remover um arquivo.

Dentre as exceções lançadas ao usar a função **remove**, destacamos as seguintes:

FileNotFoundError

Ocorre quando o arquivo não existe.

PermissionError

Ocorre quando não temos permissão para alterar o arquivo.

IsADirectoryError

Ocorre quando tentamos remover um diretório usando a função `remove`, em vez de `rmdir`.

Observe a saída do console, onde tudo ocorreu conforme esperado e nenhuma exceção foi lançada.

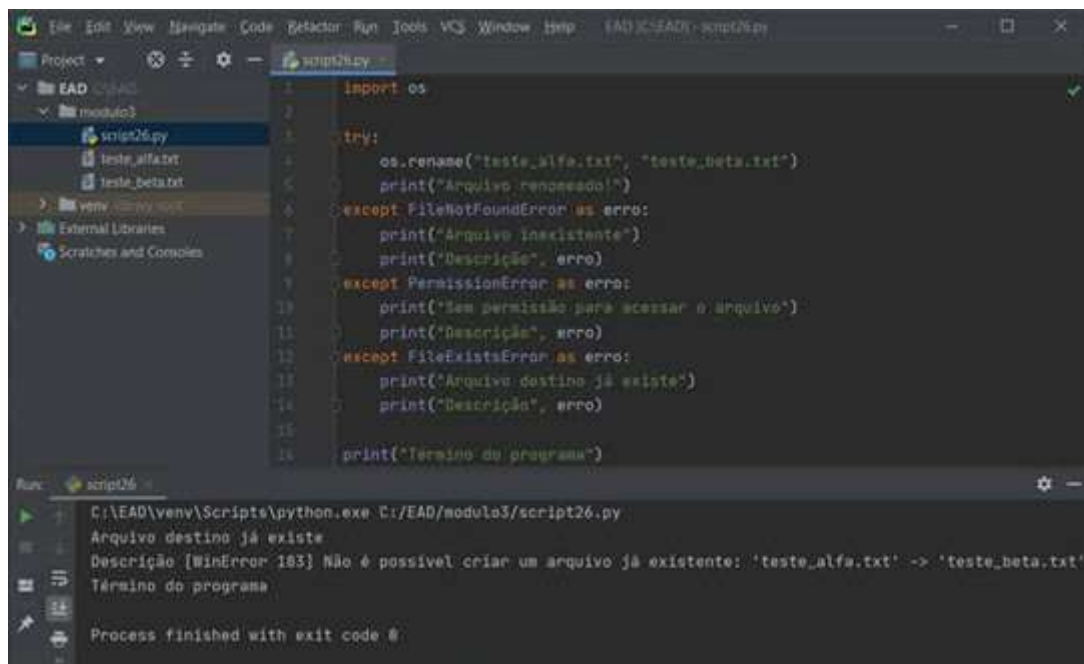
A segunda operação, também muito comum, é a de renomear um arquivo. Essa operação também está disponível no módulo `os`, mas por meio da função `rename`.

A função `rename` tem a seguinte sintaxe:

```
os.rename(origem, destino)
```

Nesse exemplo, temos o nome do módulo **os**, seguido de um ponto e o nome da função `rename`. Como parâmetro, a função espera o caminho para o arquivo que desejamos renomear, **origem**, e o novo nome do arquivo, **destino**.

Veja agora o exemplo em que descrevemos o uso dessa função:



The screenshot shows an IDE window with a Python script named `script26.py` and its execution output in the Run console.

```
1 import os
2
3 try:
4     os.rename("teste_alfa.txt", "teste_beta.txt")
5     print("Arquivo renomeado!")
6 except FileNotFoundError as erro:
7     print("Arquivo inexistente")
8     print("Descrição", erro)
9 except PermissionError as erro:
10    print("Sem permissão para acessar o arquivo")
11    print("Descrição", erro)
12 except FileExistsError as erro:
13    print("Arquivo destino já existe")
14    print("Descrição", erro)
15
16 print("Fim do programa")
```

The Run console output shows the following messages:

```
Run: C:\EAD\venv\Scripts\python.exe C:\EAD\modulo3\script26.py
Arquivo destino já existe
Descrição [WinError 183] Não é possível criar um arquivo já existente: 'teste_alfa.txt' -> 'teste_beta.txt'
Fim do programa
Process finished with exit code 0
```

Script 26 e sua saída.

Na linha 1, importamos o módulo **os**, no qual será utilizada a função `rename`.

Na linha 4, chamamos a função *rename* com os parâmetros `teste_alfa.txt` (origem) e `teste_beta.txt` (destino). Caso tudo ocorra bem, ao final da operação, teremos apenas o arquivo destino.

Veja agora algumas exceções que podem ser lançadas quando utilizamos a função *rename*. Não estamos tratando todas as opções possíveis, mas apenas as mais comuns:

FileNotFoundError

Ocorre quando a origem não existe.

FileExistsError

Ocorre quando o arquivo de destino já existe.

PermissionError

Ocorre quando não temos permissão para alterar o arquivo de origem ou para escrita do destino.

Na imagem **Script 26 e sua saída**, veja a árvore de diretórios à esquerda. Temos tanto o arquivo `teste_alfa.txt` quanto o arquivo `teste_beta.txt`.

Observe a execução do script pelo console e veja que ele saltou da linha 4 para a linha 13. Isso ocorreu porque, como o arquivo `teste_beta.txt` já existia, a exceção `FileExistsError` foi lançada.

Dica

Para os casos em que desejamos renomear sobrescrevendo o arquivo destino, caso ele exista, podemos utilizar a função *replace*, também do módulo `os`.

Manipulação de diretórios

Criando e removendo diretórios

Trabalhar com arquivos significa trabalhar com diretórios. Vejamos as principais funcionalidades relacionadas à manipulação de diretórios em Python começando pela criação e remoção de um diretório.

Para criar um diretório, utilizamos a função `mkdir` do módulo `os`, enquanto, para remover um diretório, utilizamos a função `rmdir`, também do módulo **os**.

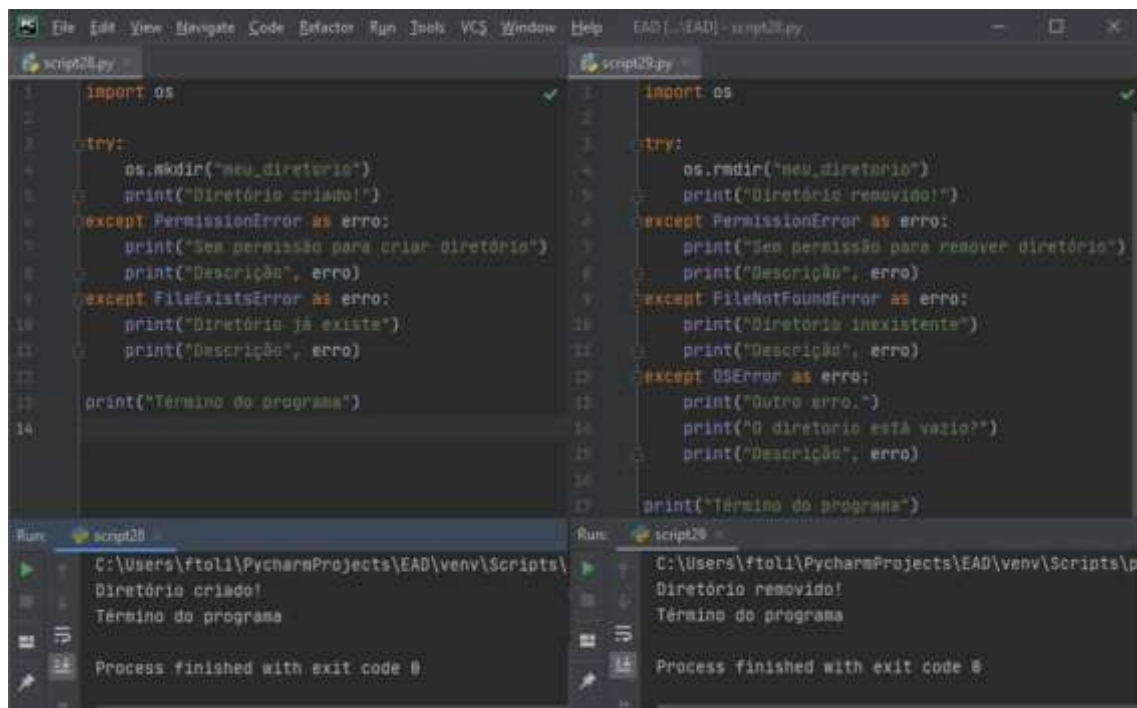
A sintaxe dessas duas funções são as seguintes:

`os.mkdir(caminho)`

`os.rmdir(caminho)`

Nesse exemplo, temos o nome do módulo **os**, seguido de um ponto e o nome da função `mkdir` ou `rmdir`. Como parâmetro, a função espera o caminho para o diretório.

Veja agora como utilizamos essas duas funções:



```
script28.py
1 import os
2
3 try:
4     os.mkdir("meu_diretorio")
5     print("Diretório criado!")
6 except PermissionError as erro:
7     print("Sem permissão para criar diretório")
8     print("Descrição", erro)
9 except FileExistsError as erro:
10    print("Diretório já existe")
11    print("Descrição", erro)
12
13 print("Término do programa")
14

Run: script28
C:\Users\ftoli\PycharmProjects\EAD\venv\Scripts\
Diretório criado!
Término do programa
Process finished with exit code 0

script29.py
1 import os
2
3 try:
4     os.rmdir("meu_diretorio")
5     print("Diretório removido!")
6 except PermissionError as erro:
7     print("Sem permissão para remover diretório")
8     print("Descrição", erro)
9 except FileNotFoundError as erro:
10    print("Diretório inexistente")
11    print("Descrição", erro)
12 except OSError as erro:
13    print("Outro erro.")
14    print("O diretório está vazio?")
15    print("Descrição", erro)
16
17 print("Término do programa")
18

Run: script29
C:\Users\ftoli\PycharmProjects\EAD\venv\Scripts\p
Diretório removido!
Término do programa
Process finished with exit code 0
```

Scripts 28 e 29 e suas respectivas saídas.

No script 28, importamos o módulo **os** na linha 1 e, na linha 4, utilizamos a função `mkdir("meu_diretorio")`. O diretório `meu_diretorio` foi criado na mesma pasta onde o `script28` se encontra. Considere as seguintes condições:

- Caso não tenhamos permissão para criar o diretório, será lançada a exceção **PermissionError**.
- Caso o diretório já exista, a exceção **FileExistsError** é lançada.

No `script29`, na linha 4, utilizamos a função `rmdir` para remover o diretório `meu_diretorio`. Considere as seguintes condições:

- Caso não tenhamos permissão para remover o diretório, será lançada a exceção **PermissionError**.

- Caso o diretório não exista, a exceção **FileNotFoundError** é lançada.

Para os casos em que o diretório a ser removido não esteja vazio, será lançada a exceção **OSError**. Essa exceção é mais abrangente.

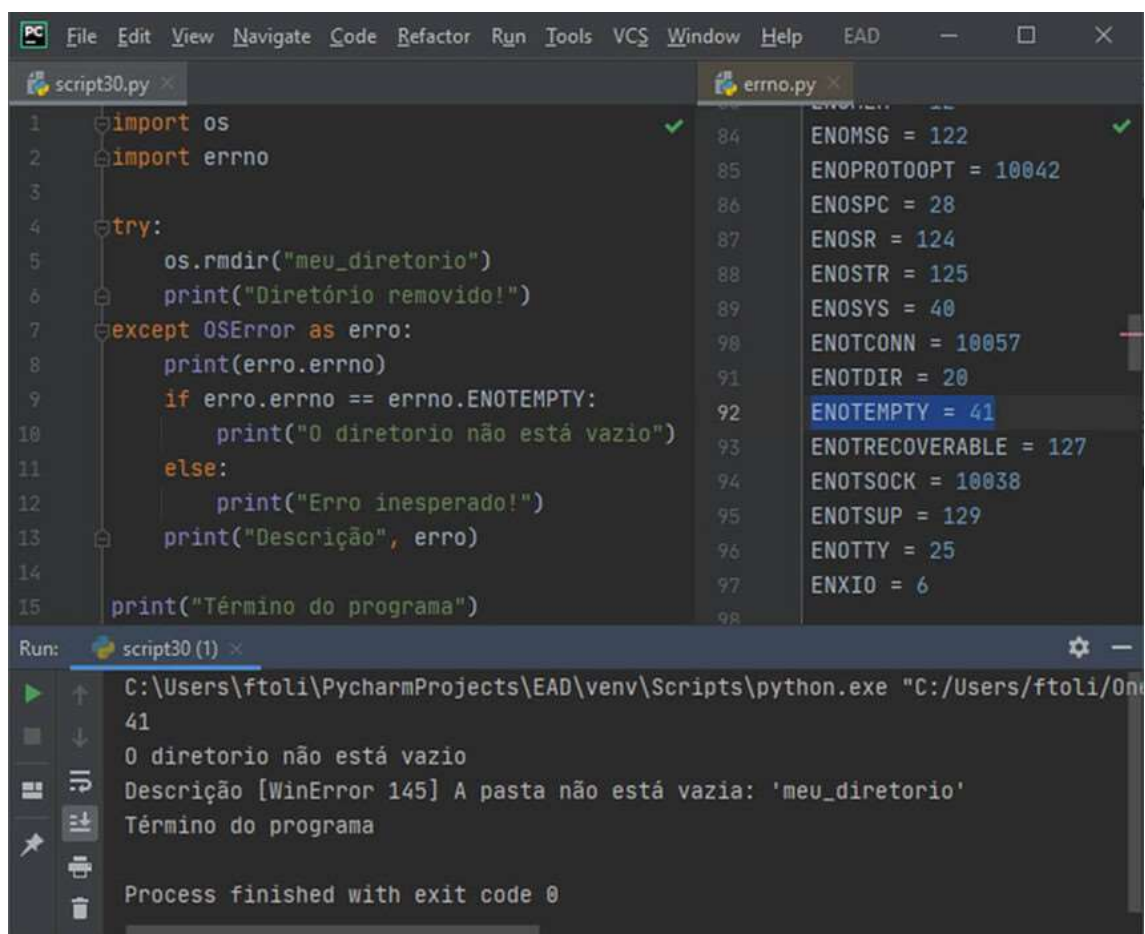
Não temos como garantir, a princípio, que a exceção lançada ocorre especificamente pelo fato de o diretório não estar vazio.

Nessas situações, precisamos analisar mais o erro, principalmente o seu número, para verificar o que realmente aconteceu.

O número do erro está disponível no atributo `errno` do objeto erro.

Os códigos dos possíveis erros estão no módulo `errno` do Python e podem ser utilizados no tratamento das exceções para descobrir o que realmente deu errado.

Veja esse problema mais de perto:



The screenshot shows a PyCharm IDE with two tabs: `script30.py` and `errno.py`. The `script30.py` tab is active, showing the following code:

```
1 import os
2 import errno
3
4 try:
5     os.rmdir("meu_diretorio")
6     print("Diretório removido!")
7 except OSError as erro:
8     print(erro.errno)
9     if erro.errno == errno.ENOTEMPTY:
10         print("O diretorio não está vazio")
11     else:
12         print("Erro inesperado!")
13     print("Descrição", erro)
14
15 print("Término do programa")
```

The `errno.py` tab is also visible, showing a list of error codes. The value `ENOTEMPTY = 41` is highlighted.

Below the code editor, the Run console shows the output of the script:

```
Run: script30 (1) x
C:\Users\ftoli\PycharmProjects\EAD\venv\Scripts\python.exe "C:/Users/ftoli/On
41
O diretorio não está vazio
Descrição [WinError 145] A pasta não está vazia: 'meu_diretorio'
Término do programa

Process finished with exit code 0
```

Script 30 e saída.

Temos o script 30, sua saída e modulo `errno.py`.

```
1 import os
2 import errno
3
4 try:
5     os.rmdir("meu_diretorio")
6     print("Diretório removido!")
7 except OSError as erro:
8     print(erro.errno)
9     if erro.errno == errno.ENOTEMPTY:
10         print("O diretório não está vazio")
11     else:
12         print("Erro inesperado!")
13     print("Descrição", erro)
14
15 print("Término do programa")
```

```
84 ENOMSG = 122
85 ENOPROTOOPT = 10042
86 ENOSPC = 28
87 ENOSR = 124
88 ENOSTR = 125
89 ENOSYS = 40
90 ENOTCONN = 10057
91 ENOTDIR = 20
92 ENOTEMPTY = 41
93 ENOTRECOVERABLE = 127
94 ENOTSOCK = 10038
95 ENOTSUP = 129
96 ENOTTY = 25
97 ENXIO = 6
98
```

Run: script30 (1) x

```
C:\Users\ftoli\PycharmProjects\EAD\venv\Scripts\python.exe "C:/Users/ftoli/On
41
O diretorio não está vazio
Descrição [WinError 145] A pasta não está vazia: 'meu_diretorio'
Término do programa

Process finished with exit code 0
```

Importação dos módulos os e errno.

Importamos, à esquerda, no script30, os módulos os e errno nas linhas 1 e 2.

The screenshot shows the PyCharm IDE interface. The top pane displays two files: `script30.py` and `errno.py`. `script30.py` contains the following code:

```
1 import os
2 import errno
3
4 try:
5     os.rmdir("meu_diretorio")
6     print("Diretório removido!")
7 except OSError as erro:
8     print(erro.errno)
9     if erro.errno == errno.ENOTEMPTY:
10         print("O diretório não está vazio")
11     else:
12         print("Erro inesperado!")
13     print("Descrição", erro)
14
15 print("Término do programa")
```

The `errno.py` file shows a list of error codes, with `ENOTEMPTY = 41` highlighted. The bottom pane shows the output of the script:

```
Run: script30 (1) x
C:\Users\ftoli\PycharmProjects\EAD\venv\Scripts\python.exe "C:/Users/ftoli/On
41
O diretório não está vazio
Descrição [WinError 145] A pasta não está vazia: 'meu_diretorio'
Término do programa

Process finished with exit code 0
```

Tentativa de remoção do diretório meu_diretorio.

Tentamos remover, na linha 5, o diretório meu_diretorio utilizando a função rmdir. Como o diretório não está vazio, a exceção OSError é lançada, e a execução do programa salta para a linha 8.

The screenshot shows the PyCharm IDE with two files open: `script30.py` and `errno.py`. The `script30.py` file contains the following code:

```
1 import os
2 import errno
3
4 try:
5     os.rmdir("meu_diretorio")
6     print("Diretório removido!")
7 except OSError as erro:
8     print(erro.errno)
9     if erro.errno == errno.ENOTEMPTY:
10         print("O diretório não está vazio")
11     else:
12         print("Erro inesperado!")
13     print("Descrição", erro)
14
15 print("Término do programa")
```

The `errno.py` file shows a list of error codes, with `ENOTEMPTY = 41` highlighted. The Run console at the bottom shows the output of the script:

```
Run: script30 (1) x
C:\Users\ftoli\PycharmProjects\EAD\venv\Scripts\python.exe "C:/Users/ftoli/On
41
O diretório não está vazio
Descrição [WinError 145] A pasta não está vazia: 'meu_diretorio'
Término do programa

Process finished with exit code 0
```

Impressão do número do erro.

Imprimimos, na linha 8, o número do erro por meio do atributo `errno` da variável `erro`. Observe que o valor impresso foi 41. O erro 41 faz parte da numeração interna de erros do Python, que pode ser verificado no módulo `errno` à direita da imagem, mapeado como `ENOTEMPTY` (não vazio).

The screenshot shows a PyCharm IDE with two tabs: `script30.py` and `errno.py`. The `script30.py` tab is active, showing a Python script that attempts to remove a directory. Line 9 is highlighted with a blue box, containing the condition `if erro.errno == errno.ENOTEMPTY:`. The `errno.py` tab is also visible, showing a list of error codes. The code in `script30.py` is as follows:

```
1 import os
2 import errno
3
4 try:
5     os.rmdir("meu_diretorio")
6     print("Diretório removido!")
7 except OSError as erro:
8     print(erro.errno)
9     if erro.errno == errno.ENOTEMPTY:
10         print("O diretório não está vazio")
11     else:
12         print("Erro inesperado!")
13     print("Descrição", erro)
14
15 print("Término do programa")
```

The `Run` window at the bottom shows the output of the script:

```
Run: script30 (1) x
C:\Users\ftoli\PycharmProjects\EAD\venv\Scripts\python.exe "C:/Users/ftoli/On
41
O diretório não está vazio
Descrição [WinError 145] A pasta não está vazia: 'meu_diretorio'
Término do programa
Process finished with exit code 0
```

Comparação entre o erro gerado e o código do erro.

Comparamos, na linha 9, o erro gerado pelo programa com o código do erro `ENOTEMPTY` (erro 41). Caso o resultado da comparação seja verdadeiro, teremos certeza de que o erro ocorreu, pois o diretório não está vazio. Caso contrário, precisaremos analisá-lo novamente. O `WinError 145` é o erro nativo que o Windows retornou. Esse erro foi mapeado pelo Python para o erro 41.

Como a exceção `OSError` é mais abrangente que as outras exceções que estudamos, ela deve ficar por último. Caso contrário, nunca alcançaremos as exceções mais específicas.

Listando conteúdo de diretórios

Outra tarefa muito comum quando estamos tratando com arquivos é listar os arquivos presentes em um diretório.

Para isso, podemos utilizar a função `scandir` do módulo `os`. Sua sintaxe é a seguinte:

```
os.scandir(caminho)
```

Nesse exemplo, temos o nome do módulo **os**, seguido de um ponto e o nome da função **scandir**. Como parâmetro, a função espera o caminho para o **diretório**.

Como resultado, teremos um iterável (iterator) que retorna objetos do tipo **os.DirEntry**, que podem ser arquivos ou diretórios. Dentre os atributos e métodos desse tipo de objetos, destacamos:

Name

Nome do diretório ou arquivo.

Path

Caminho completo do diretório ou arquivo.

is_dir()

Retorna verdadeiro se o objeto é um diretório.

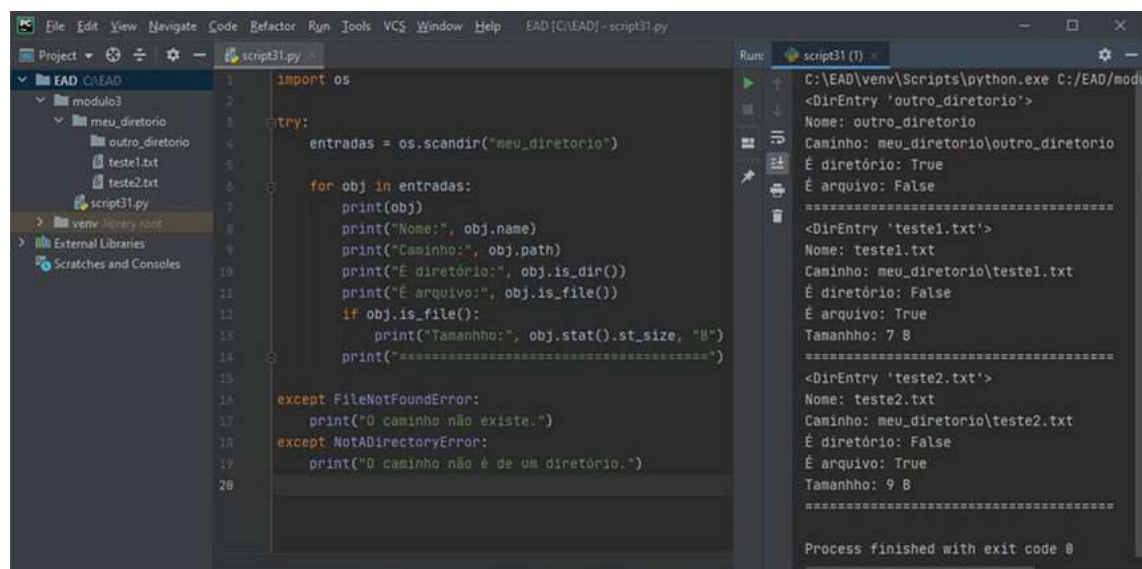
is_file()

Retorna verdadeiro se o objeto é um arquivo.

stat()

Retorna alguns atributos do arquivo ou diretório, como tamanho.

Agora veja como utilizar essa função:



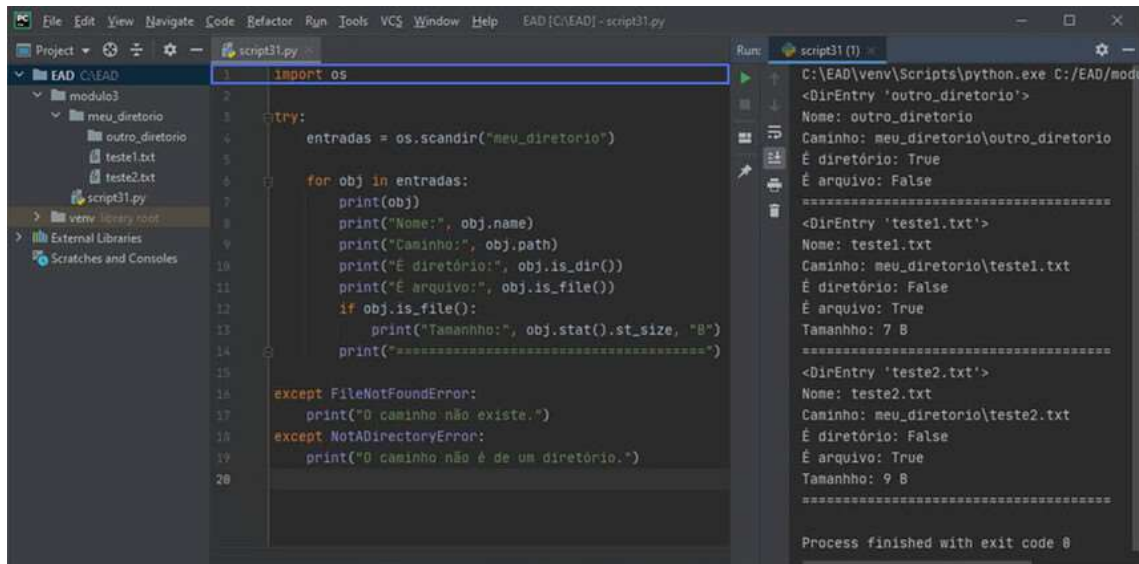
```
1 import os
2
3 try:
4     entradas = os.scandir("meu_diretorio")
5
6     for obj in entradas:
7         print(obj)
8         print("Nome:", obj.name)
9         print("Caminho:", obj.path)
10        print("É diretório:", obj.is_dir())
11        print("É arquivo:", obj.is_file())
12        if obj.is_file():
13            print("Tamanho:", obj.stat().st_size, "B")
14        print("=====")
15
16 except FileNotFoundError:
17     print("O caminho não existe.")
18 except NotADirectoryError:
19     print("O caminho não é de um diretório.")
20
```

Run: script31 (1)

```
<DirEntry 'outro_diretorio'>
Nome: outro_diretorio
Caminho: meu_diretorio\outro_diretorio
É diretório: True
É arquivo: False
=====
<DirEntry 'teste1.txt'>
Nome: teste1.txt
Caminho: meu_diretorio\teste1.txt
É diretório: False
É arquivo: True
Tamanho: 7 B
=====
<DirEntry 'teste2.txt'>
Nome: teste2.txt
Caminho: meu_diretorio\teste2.txt
É diretório: False
É arquivo: True
Tamanho: 9 B
=====
Process finished with exit code 0
```

Script 31 e saída.

Temos o script 31 e sua saída. Vamos percorrer os arquivos e diretórios da pasta **meu_diretorio**. A árvore de diretórios pode ser verificada à esquerda da imagem.



The screenshot shows an IDE with a project named 'EAD C/HEAD'. The file explorer on the left shows a directory structure with 'meu_diretorio' containing 'outro_diretorio', 'teste1.txt', and 'teste2.txt'. The script 'script31.py' is open in the editor. The code imports the 'os' module and uses 'os.scandir()' to list the contents of 'meu_diretorio'. The output window on the right shows the results of the script execution, displaying details for each directory entry, including its name, path, whether it's a directory or file, and its size.

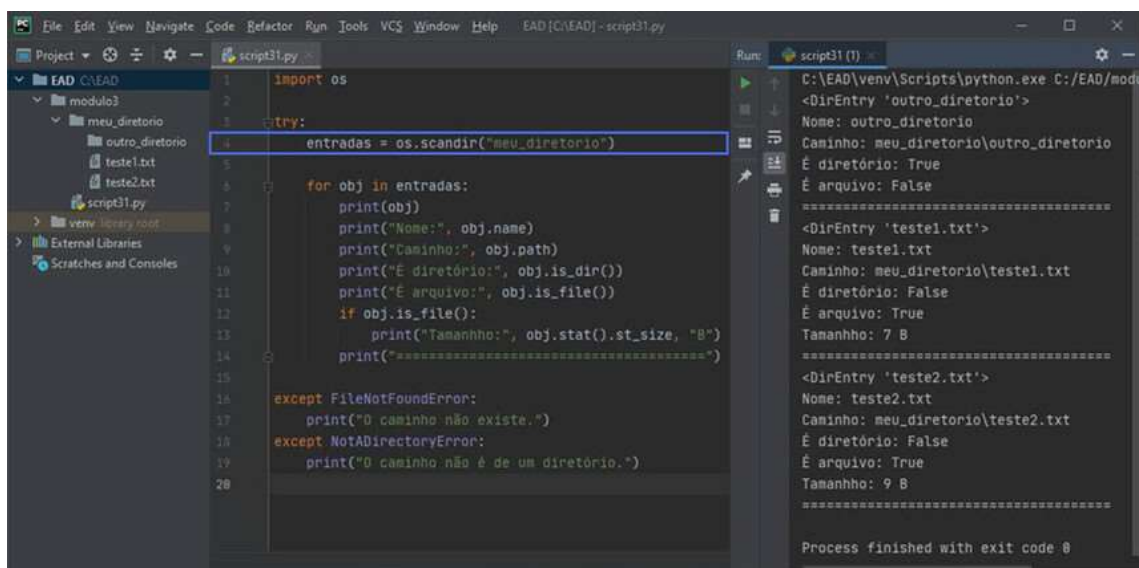
```
1 import os
2
3 try:
4     entradas = os.scandir("meu_diretorio")
5
6     for obj in entradas:
7         print(obj)
8         print("Nome:", obj.name)
9         print("Caminho:", obj.path)
10        print("É diretório:", obj.is_dir())
11        print("É arquivo:", obj.is_file())
12        if obj.is_file():
13            print("Tamanho:", obj.stat().st_size, "B")
14        print("=====")
15
16 except FileNotFoundError:
17     print("O caminho não existe.")
18 except NotADirectoryError:
19     print("O caminho não é de um diretório.")
20
```

Run: script31 (1)

```
C:\EAD\venv\Scripts\python.exe C:/EAD/modu
<DirEntry 'outro_diretorio'>
Nome: outro_diretorio
Caminho: meu_diretorio\outro_diretorio
É diretório: True
É arquivo: False
=====
<DirEntry 'teste1.txt'>
Nome: teste1.txt
Caminho: meu_diretorio\teste1.txt
É diretório: False
É arquivo: True
Tamanho: 7 B
=====
<DirEntry 'teste2.txt'>
Nome: teste2.txt
Caminho: meu_diretorio\teste2.txt
É diretório: False
É arquivo: True
Tamanho: 9 B
=====
Process finished with exit code 0
```

Importação dos módulos os.

Importamos, na linha 1, o módulo os, onde se encontra a função scandir.



This screenshot is identical to the one above, showing the same Python script and its output. The code imports the 'os' module and uses 'os.scandir()' to list the contents of 'meu_diretorio'. The output window on the right shows the results of the script execution, displaying details for each directory entry, including its name, path, whether it's a directory or file, and its size.

```
1 import os
2
3 try:
4     entradas = os.scandir("meu_diretorio")
5
6     for obj in entradas:
7         print(obj)
8         print("Nome:", obj.name)
9         print("Caminho:", obj.path)
10        print("É diretório:", obj.is_dir())
11        print("É arquivo:", obj.is_file())
12        if obj.is_file():
13            print("Tamanho:", obj.stat().st_size, "B")
14        print("=====")
15
16 except FileNotFoundError:
17     print("O caminho não existe.")
18 except NotADirectoryError:
19     print("O caminho não é de um diretório.")
20
```

Run: script31 (1)

```
C:\EAD\venv\Scripts\python.exe C:/EAD/modu
<DirEntry 'outro_diretorio'>
Nome: outro_diretorio
Caminho: meu_diretorio\outro_diretorio
É diretório: True
É arquivo: False
=====
<DirEntry 'teste1.txt'>
Nome: teste1.txt
Caminho: meu_diretorio\teste1.txt
É diretório: False
É arquivo: True
Tamanho: 7 B
=====
<DirEntry 'teste2.txt'>
Nome: teste2.txt
Caminho: meu_diretorio\teste2.txt
É diretório: False
É arquivo: True
Tamanho: 9 B
=====
Process finished with exit code 0
```

Aplicação da função scandir.

Utilizamos, na linha 4, a função scandir utilizando o diretório “meu_diretorio” como argumento. Armazenamos o retorno dessa função na variável entradas.

The screenshot shows an IDE with a project named 'EAD CNEAD'. The file explorer on the left shows a directory structure with 'meu_diretorio' containing 'outro_diretorio', 'teste1.txt', and 'teste2.txt'. The script 'script31.py' is open in the editor. The code is as follows:

```
1 import os
2
3 try:
4     entradas = os.listdir("meu_diretorio")
5
6     for obj in entradas:
7         print(obj)
8         print("Nome:", obj.name)
9         print("Caminho:", obj.path)
10        print("É diretório:", obj.is_dir())
11        print("É arquivo:", obj.is_file())
12        if obj.is_file():
13            print("Tamanho:", obj.stat().st_size, "B")
14        print("=====")
15
16 except FileNotFoundError:
17     print("O caminho não existe.")
18 except NotADirectoryError:
19     print("O caminho não é de um diretório.")
20
```

The Run window on the right shows the output of the script:

```
C:\EAD\venv\Scripts\python.exe C:/EAD/mod
<DirEntry 'outro_diretorio'>
Nome: outro_diretorio
Caminho: meu_diretorio\outro_diretorio
É diretório: True
É arquivo: False
=====
<DirEntry 'teste1.txt'>
Nome: teste1.txt
Caminho: meu_diretorio\teste1.txt
É diretório: False
É arquivo: True
Tamanho: 7 B
=====
<DirEntry 'teste2.txt'>
Nome: teste2.txt
Caminho: meu_diretorio\teste2.txt
É diretório: False
É arquivo: True
Tamanho: 9 B
=====
Process finished with exit code 0
```

Iteração de cada entrada.

Iteramos, na linha 6, cada entrada e, da linha 7 a 13, imprimimos algumas de suas propriedades.

This screenshot is identical to the first one, showing the same IDE environment, code, and output. It highlights the iteration process in the code and the corresponding output in the console.

Saída no console à direita.

Observe a saída no console à direita.