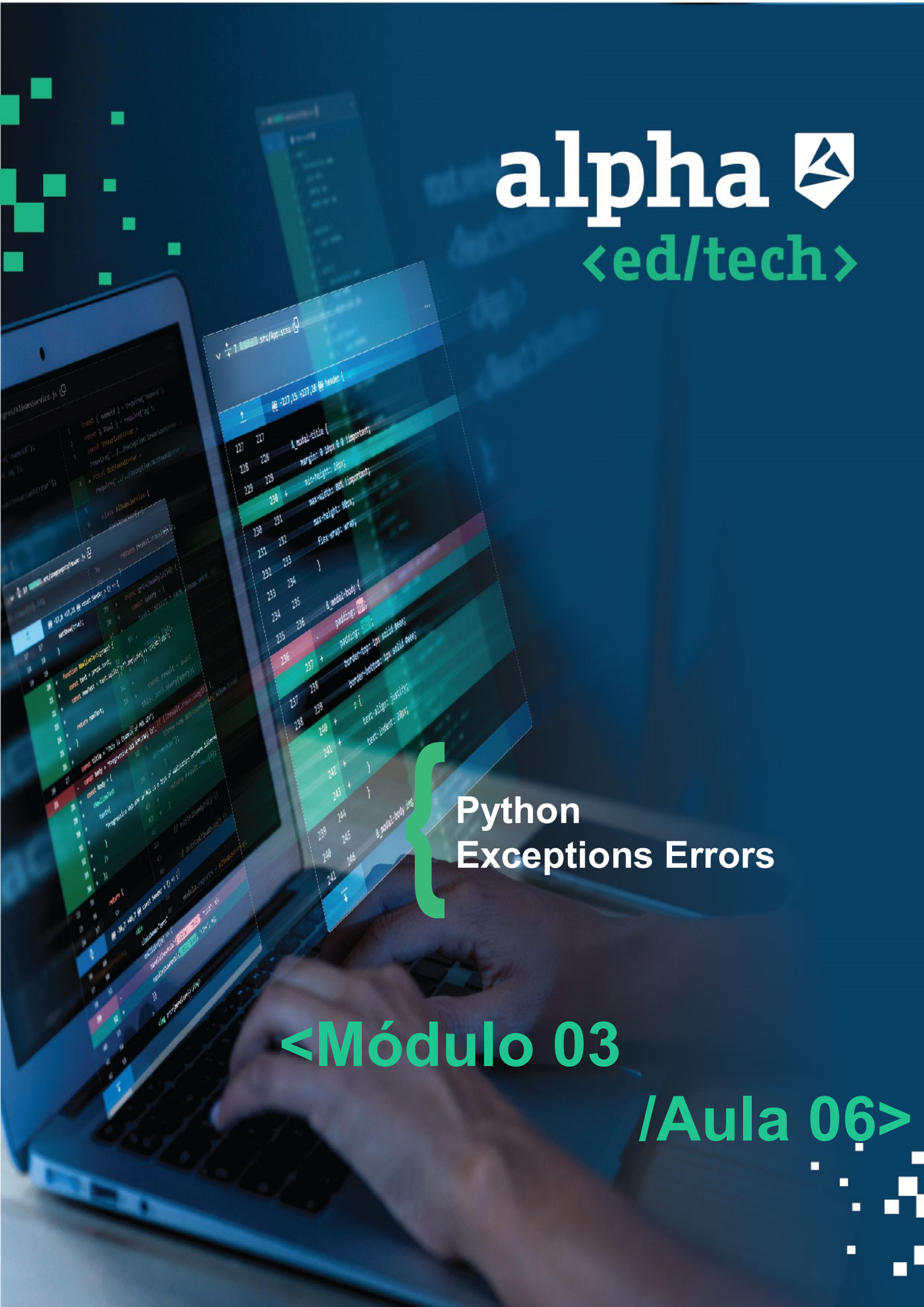
alpha
<ed/tech>
Python
Exceptions Errors
<Módulo 03
/Aula 06>

Exceptions Erros

Python como a maioria das linguagens de programação permite que o desenvolvedor capture erros.
Através dos objetos de exceção, é possível tentar corrigir o erro sem a necessidade de interromper a execução do programa ou mesmo mandar uma mensagem de erro que o usuário final possa compreender.

Em Python, o tratamento de erros e exceções é realizado usando blocos `try`, `except`, `else` e `finally`.

Existe uma farta documentação de erros capturados pelo Python no link:

https://docs.python.org/pt-br/3/tutorial/errors.html

Existem erros de sintaxe e erros de exceções.

Erros de sintaxe são aqueles que cometemos ao escrever nosso código.

Por exemplo, quando esquecemos de colocar dois pontos após a definição de uma função.

Exceções, ou **Erros Lógicos,** ocorrem quando o programa é executado de uma forma que possa gerar um problema não previsto durante seu desenvolvimento, mesmo com a sintaxe perfeita.

Por exemplo, tentar entrar com strings de texto no lugar de números em cálculos aritméticos, acessar recursos inexistentes de rede ou bancos de dados, ou realizar operações proibidas, como a divisão por zero.

As **exceções** são eventos especiais – geralmente erros – que ocorrem em tempo de execução.
Quando um erro desses ocorre, o Python cria um objeto do tipo Exception.
Este objeto deve ser manipulado corretamente, caso contrário apenas imprimirá na tela um *traceback* do erro com alguns detalhes técnicos a respeito, além de parar inesperadamente a execução do script (trava o programa ou o finaliza) – o que não ajuda o usuário em praticamente nada.

https://docs.python.org/pt-br/3/library/exceptions.html

Erros comuns:

AttributeError: Quando tentamos acessar um atributo ou método que não existe em um objeto. Quando tentamos acessar uma variável que não existe.

IndexError: Quando tentamos acessar um elemento de uma sequência com um índice maior que o total de seus elementos menos um.

NameError: Quando tentamos acessar uma variável que não existe.

TypeError: Quando tentamos usar um valor de forma inadequada, como por exemplo tentar indexar um sequência com algo diferente de um número inteiro ou de um fatiamento.

Try Except

```
try:
    # Código que pode gerar um erro
    resultado = 10 / 0 # Tentativa de divisão por zero
except:
    # Bloco executado se ocorrer um erro
    print("Erro detectado!")
```

O código dentro do bloco `try` é executado.
Se ocorrer um erro durante a execução desse bloco, o controle é transferido para o bloco `except`.
Neste caso, a tentativa de dividir por zero gera uma exceção, e o bloco `except` é executado.

Tratando exceções específicas

```
try:
    # Tentativa de converter uma string para inteiro
    numero = int("texto")
except ValueError:
    print("Erro: Valor inválido para conversão.")
```

O bloco `except` só será executado se uma exceção do tipo `ValueError` ocorrer.
Isso ajuda a lidar com diferentes tipos de erros de forma mais específica.

Utilizando `else` para código sem erros

```
try:
    numero = int(input("Digite um número: "))
except ValueError:
    print("Erro: Valor inválido para conversão.")
else:
    print("Você digitou um número válido.")
```

O bloco `else` é executado apenas se nenhum erro ocorrer dentro do bloco `try`. Isso permite separar o código que pode gerar erros do código que deve ser executado apenas se não houver erros.

Usando `finally` para código de limpeza

```
try:
    # Código para ler o conteúdo do arquivo
    arquivo = open("arquivo.txt", "r")
Except FileNotFoundError:
    print("Erro: Arquivo não encontrado.")
finally:
    # Garante que o arquivo seja fechado,
    # independentemente de ocorrer um erro ou não.
    arquivo.close()
```

O bloco `finally` é executado sempre, independentemente de ocorrer uma exceção ou não. Pode ser usado para realizar ações de limpeza, como fechar arquivos ou conexões de banco de dados.

Capturando múltiplas exceções

```
try:
    resultado = x / y
except ZeroDivisionError:
    print("Erro: Divisão por zero.")
except ValueError:
    print("Erro: Valor inválido.")
except Exception as e:
    print(f"Erro inesperado: {e}")
```

Permite capturar diferentes tipos de exceções e lidar com elas de maneira específica. O último bloco `except` (com `Exception`) captura qualquer exceção não especificada anteriormente.

Lançamentos de exceções

Existe o comando `raise` que permite forçar uma ocorrência de um determionado tipo de exceção. O programador cria a sua própria exception de acordo com sua necessidade.

```
x = -1
```

```
if x < 0:
    raise Exception("O número deve ser positivo ou nulo")
```

Esses são conceitos básicos de tratamento de erros e exceções em Python.

Eles ajudam a tornar seus programas mais robustos, permitindo que você lide de maneira adequada com situações imprevistas.