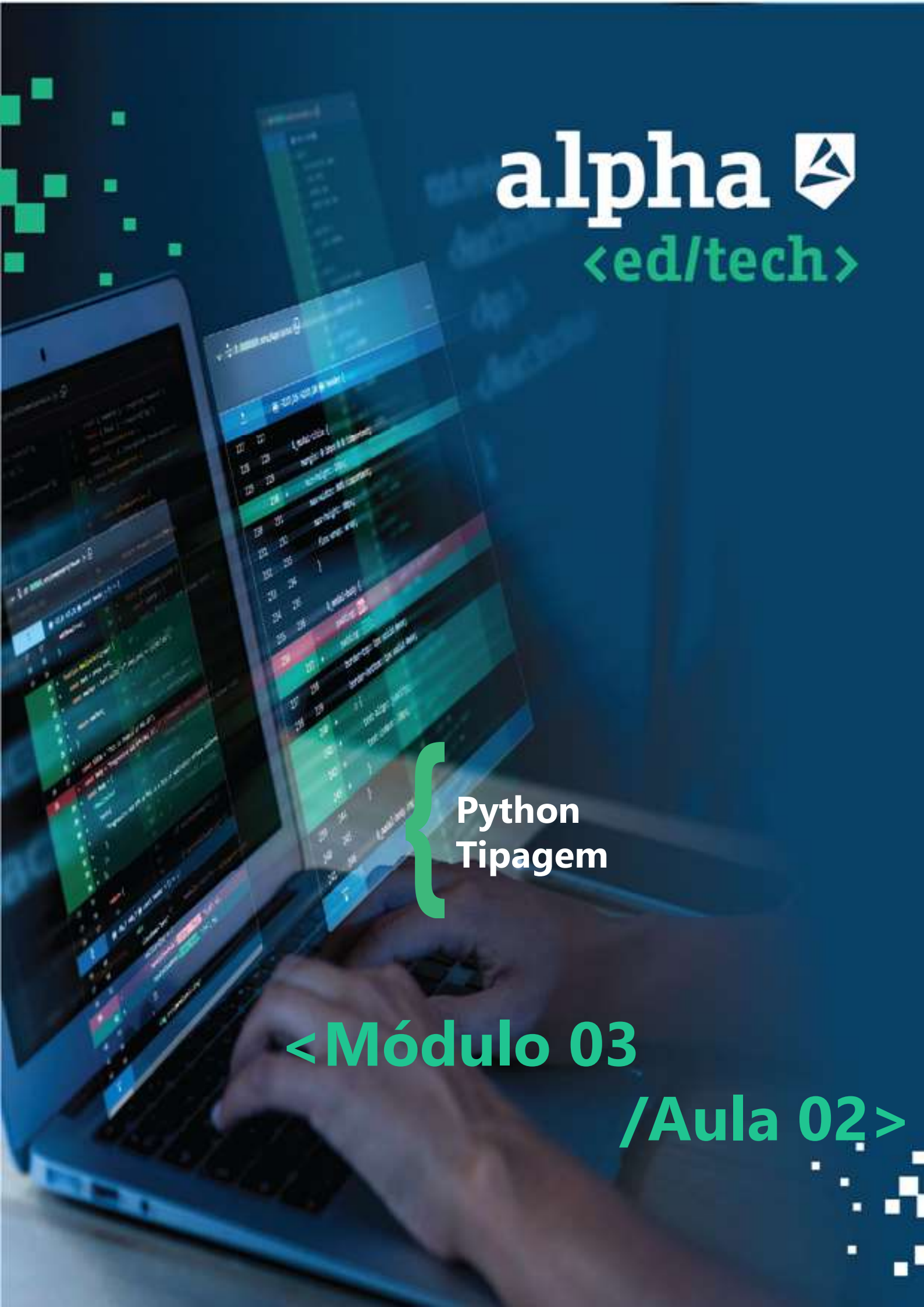
alpha
<ed/tech>
Python
Tipagem
<Módulo 03
/Aula 02>

Funções e Tipos

Introdução

Bem-vindos ao capítulo sobre **Funções e Tipos** em Python, ou simplesmente **Tipagem**. Aprofundar seus conhecimentos, pois abordaremos os **tipos de dados**, que formam a base de como a informação é armazenada e manipulada em Python. Além disso, abordaremos os *Type Hints* e a ferramenta *MyPy* para garantir que o código seja não apenas funcional, mas também tipado de maneira correta.

Tipagem Estática e Tipagem Dinâmica

A **tipagem** (**utilização de tipos de variáveis**) em Python pode ser usada através de dois métodos: **tipagem estática** e **tipagem dinâmica**.

Python é conhecido por ser uma linguagem de tipagem dinâmica, o que significa que você não precisa declarar explicitamente o tipo de uma variável ao atribuir um valor a ela.

```
x = 10  # x é do tipo int (inteiro)
y = "Olá!"  # y é do tipo str (string)
```

Em Python, você não precisa declarar que `x` é um inteiro ou `y` é uma string. **O interpretador Python infere automaticamente os tipos de dados com base nos valores atribuídos às variáveis**.

Tipagem Estática em Python

Python tem suporte para a **tipagem estática** através do módulo ***`typing`***.

Embora a **tipagem estática não altere a natureza dinâmica** dos tipos em tempo de execução, é **utilizada para ajudar os desenvolvedores a entenderem e a manterem o código**.

O módulo **`typing`** oferece uma maneira de indicar intenções de tipos **sem alterar o comportamento real da tipagem dinâmica do Python**.

Exemplo de Tipagem Estática com `typing`

```
from typing import List, Tuple

def soma(a: int, b: int) -> int:
    return a + b

def junta_texto(texto1: str, texto2: str) -> str:
    return texto1 + texto2

def processa_lista(lista: List[int]) -> Tuple[int, int]:
    return min(lista), max(lista)

# Exemplo de uso
resultado = soma(5, 3)  # resultado será do tipo int
texto_unido = junta_texto("Olá, ", "mundo!")  # texto_unido será do tipo str
minimo, maximo = processa_lista([1, 2, 3, 4, 5])  # minimo e maximo serão int

print(resultado, texto_unido, minimo, maximo)
```

Neste exemplo, as funções têm indicações de tipos usando **anotações de tipo** (`: int`, `: str`, `List[int]` etc.), fornecendo informações sobre os **tipos esperados para os argumentos e retornos**. No entanto, essas anotações de tipo são apenas sugestões para ferramentas de análise estática e IDEs, não alterando a natureza dinâmica do Python.

A **tipagem estática** pode ser **útil para documentação de código**, tornando-o mais **legível** e **compreensível**, especialmente em grandes projetos colaborativos.

No entanto, a interpretação das anotações de tipo é feita apenas por ferramentas externas, como o `**mypy**`, um verificador de tipo estático opcional para Python.

Em resumo, em Python, você pode utilizar a **tipagem dinâmica** para a maioria das situações, enquanto a **tipagem estática** (com `typing`) pode ser usada **opcionalmente** para ajudar na legibilidade e na manutenção do código, principalmente em projetos maiores.

mypy

O `**mypy**` é uma ferramenta de verificação de tipos estáticos para Python.
Ele pode ser utilizado para verificar se o seu código está de acordo com as anotações de tipos definidas no código-fonte, ajudando a **identificar possíveis erros de tipagem** antes da execução do programa.

Instalação para usar o mypy

Certifique-se de ter o `mypy` instalado. Você pode instalar o `mypy` via `pip`.

```
pip install mypy
```

Ou gerenciador de extensões.
Ou pip manager.

Adição de Anotações de Tipo

Em seu código Python, você precisará adicionar anotações de tipo utilizando a sintaxe do módulo `typing`.

```
def soma(a: int, b: int) -> int:
    return a + b

resultado = soma(5, "3")  # Exemplo de erro de tipagem
print(resultado)
```

Executar o mypy

Após adicionar as anotações de tipo, você pode **executar o `mypy`** passando o nome do arquivo ou diretório onde seu código está localizado.

```
mypy exemplo03_02_03.py
```

Se o `**mypy**` encontrar problemas de tipagem no seu código, ele irá fornecer **mensagens de erro indicando as linhas específicas** onde ocorrem as discrepâncias entre as anotações de tipo e o código real.

mypy

Opções Adicionais

Se o `mypy` encontrar problemas de tipagem no seu código, ele irá fornecer mensagens de erro indicando as linhas específicas onde ocorrem as discrepâncias entre as anotações de tipo e o código real

Verificação Recursiva: Para verificar todos os arquivos em um diretório recursivamente, use o comando:

```
mypy seu_diretorio/ \
```

Ignorar Erros: Às vezes, você pode querer ignorar certos erros ou arquivos ao verificar com o `mypy`.
Para isso, você pode adicionar comentários especiais no seu código, como
`# type: ignore`
para informar ao `mypy` para ignorar a verificação naquela linha específica.

Configuração: O `mypy` também suporta a criação de arquivos de configuração (`mypy.ini` ou `setup.cfg`) para personalizar o comportamento das verificações. Isso inclui configurações para definir caminhos a serem ignorados, ajustar níveis de severidade de erros e muito mais.

Utilizar o `mypy` pode ajudar a melhorar a qualidade do seu código Python, garantindo consistência nos tipos e reduzindo possíveis erros relacionados à tipagem. Integrar a verificação de tipos estáticos pode ser particularmente útil em projetos grandes ou em equipes colaborativas.