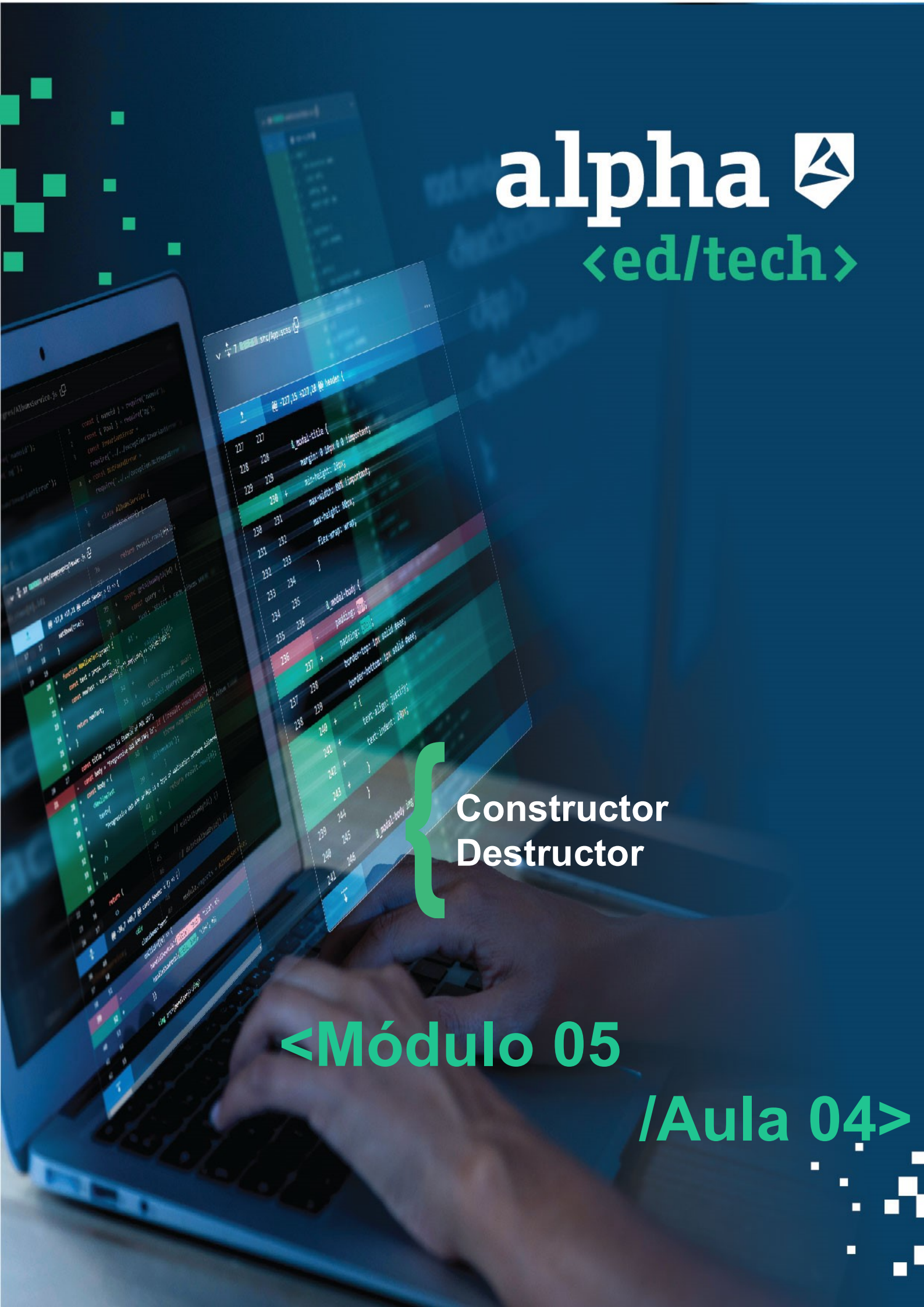
alpha
<ed/tech>
Constructor
Destructor
<Módulo 05
/Aula 04>

Constructor & Destructor

Método __init__

Em Python, você pode **implementar diferentes construtores para classes** usando métodos de classe alternativos, conhecidos como **métodos de classe** ou **métodos de fábrica**.

Utilizando o método `__init__` o objeto é inicializado quando é instanciado.

Para instanciar um objeto também pode ser chamado um **método construtor alternativo**.

Métodos de Classe Alternativos (Métodos de Fábrica)

```python
class MinhaClasse:

    def __init__(self, atributo1, atributo2):
        self.atributo1 = atributo1
        self.atributo2 = atributo2

    @classmethod
    def construtor_alternativo(cls, parametro):
        # Lógica do construtor alternativo
        # Pode retornar uma instância da classe
        # com base nos parâmetros fornecidos
        return cls("Valor Padrão", parametro)

# Uso do construtor padrão
objeto1 = MinhaClasse("Atributo1", "Atributo2")

# Uso do construtor alternativo
objeto2 = MinhaClasse.construtor_alternativo("Novo Valor")
```

Constructor & Destructor

Métodos de Instância com Padrões de Argumentos

```python
class MinhaClasse:
    # Atributos da Classe
    atribClasse: float

    def __init__(self, atributo1, atributo2):
        # Atributos da Instância (do Objeto)
        self.atributo1 = atributo1
        self.atributo2 = atributo2

    @classmethod
    def construtor_alternativo(cls, parametro):
        # Lógica do construtor alternativo
        # Pode retornar uma instância da classe
        # com base nos parâmetros fornecidos
        return cls("Valor Padrão", parametro)

    @classmethod
    def construtor_outro_alternativo(cls, param1, param2):
        # Outra lógica de construtor alternativo
        return cls(parametro1, parametro2)

# Uso do construtor padrão
objeto1 = MinhaClasse("Atributo1", "Atributo2")

# Uso do primeiro construtor alternativo
objeto2 = MinhaClasse.construtor_alternativo("Novo Valor")

# Uso do segundo construtor alternativo
objeto3 = MinhaClasse.construtor_outro_alternativo(“V1", “V2")
```

Ambas as abordagens permitem criar instâncias da classe com diferentes conjuntos de parâmetros, proporcionando flexibilidade na construção de objetos. Escolha a abordagem que melhor se adequa ao seu design e requisitos específicos.

Constructor & Destructor

Destrutores em Python

Em Python, o método `__del__` é usado como um "**destrutor**". Ele é chamado quando **um objeto é destruído ou liberado da memória**, geralmente no momento em que o coletor de lixo decide que o objeto não é mais referenciado e pode ser removido.

```python
class Exemplo:
    def __init__(self, nome):
        self.nome = nome
        print(f'Objeto {self.nome} criado.')

    def __del__(self):
        print(f'Objeto {self.nome} destruído.')

# Criando instâncias da classe
objeto1 = Exemplo('A')
objeto2 = Exemplo('B')

# Removendo referências aos objetos
del objeto1
del objeto2
```

Neste exemplo, quando um objeto da classe `Exemplo` é criado, o método `__init__` é chamado, e quando o objeto é destruído (por exemplo, quando `del` é usado para remover a referência a ele), o método `__del__` é chamado.

É importante notar que o uso direto de `__del__` não é recomendado em muitos casos, pois o momento exato da chamada do **destrutor** não é garantido devido ao funcionamento interno do coletor de lixo em Python. Em vez disso, é preferível usar o método `__enter__` e `__exit__` em conjunto com o bloco `with` para gerenciar recursos ou usar outras estratégias de gerenciamento de contexto.

```python
class Exemplo:
    def __enter__(self):
        print('Entrando no bloco "with".')
        return self
    def __exit__(self, exc_type, exc_value, traceback):
        print('Saindo do bloco "with".')
# Exemplo de uso
with Exemplo() as e:
    # código dentro do bloco "with“
    pass
```

Isso é mais seguro e previsível em comparação com o método `__del__`.

Constructor & Destructor

Assinatura de Métodos ou Construtores alternativos

Em Python, é possível definir vários **métodos construtores** (ou inicializadores) para uma mesma classe usando um conceito chamado "método de classe" e decoradores. Abaixo está um exemplo de como você pode fazer isso.

```python
class MinhaClasse:
    def __init__(self, parametro1, parametro2):
        self.parametro1 = parametro1
        self.parametro2 = parametro2

    @classmethod
    def construtor_alternativo(cls, valor):
        # Cria uma instância da classe usando um valor específico
        return cls("Valor fixo", valor)

# Uso do construtor padrão
objeto1 = MinhaClasse("Hello", "World")
print(objeto1.parametro1, objeto1.parametro2) # Saída: Hello World

# Uso do construtor alternativo
objeto2 = MinhaClasse.construtor_alternativo("Outro valor")
print(objeto2.parametro1, objeto2.parametro2)
# Saída: Valor fixo Outro valor
```

No exemplo acima, `__init__` é o construtor padrão da classe. Além disso, há um método de classe chamado `construtor_alternativo`, que cria uma instância da classe com valores específicos. O decorador `@classmethod` é usado para indicar que esse método pertence à classe, não à instância.
Ao chamar `MinhaClasse.construtor_alternativo("Outro valor")`, você cria uma instância da classe com valores diferentes dos passados para o construtor padrão.
Lembre-se de ajustar os nomes dos métodos e parâmetros conforme necessário para atender aos requisitos específicos da sua classe.

Outras linguagens, como por exemplo Java, permitem uma mesma função poder ser chamada com parâmetros diferentes. Chamando de assinatura de um método a composição do nome da função e os parâmetros, com quantidades e tipos. Para Python não é possível ter a mesma função com número de parâmetros diferentes, exceto se usar *args ou *kargs.

https://daltonmatos.com/2020/01/chamando-funcoes-python-com-assinatura-dinamica-baseada-em-typehint/
https://acervolima.com/python-obter-assinatura-de-funcao/
https://medium.com/pelando/dicas-de-como-documentar-suas-funções-em-python-161572f787f6