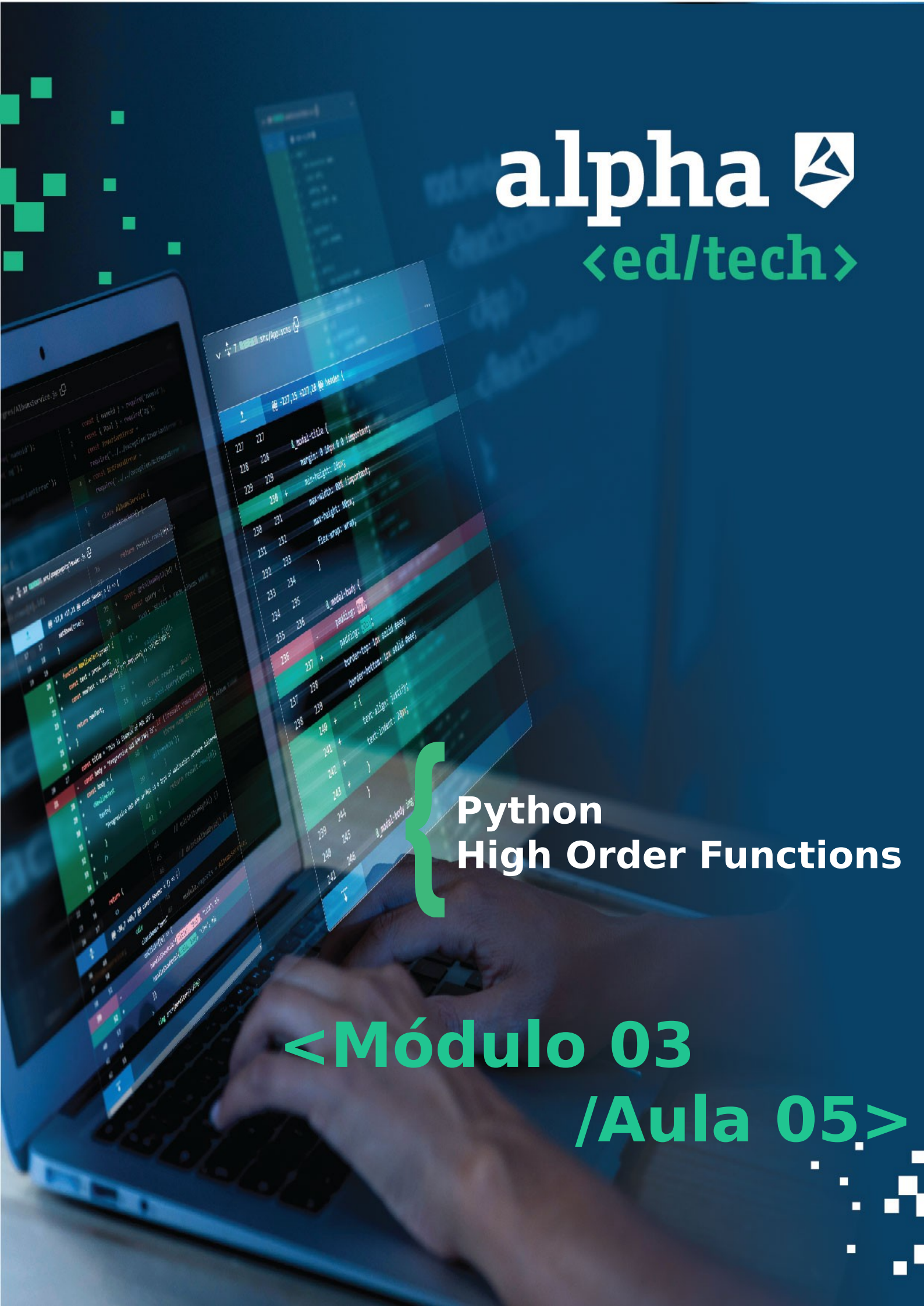
alpha
<ed/tech>
Python
High Order Functions
<Módulo 03
/Aula 05>

Funções de Alta Ordem HOF

Introdução

HOF ou High Order Functions:
- Passar comportamento como argumento para uma função.
- Passar comportamento como retorno de uma função.

```
from typing import Callable

def create_adder(x: float) -> Callable[[float], float]:
    def adder(y: float) -> float:
        return x + y


# função recebe uma função como argumento
# ou retorna uma função
add_1 = create_adder(1)
add_1 = create_adder(5)

print(add_1(10)) # 11
print(add_5(10)) # 15
print(add_5(add_1(10)) # 16
```

Lambda

- Funções anônimas
- Apenas uma linha
- Sem type hints (evite usar)

```
f = lambda a: a + 10
x = f(5)
print(x)
```

Map

- Bom conhecer mas não é necessário usar
- Problemas com tipagem se usar lambda
- Facilmente substituível com list comprehension

```
list_of_inputs = [1, 2, 3, 4, 5]

def function_to_apply(x: int) -> float:
    return x ** (1/2)

res = list(map(function_to_apply, list_of_inputs))
```

Funções de Alta Ordem HOF

```
list_of_inputs = [1, 2, 3, 4, 5]

res = list(map(lambda x: x ** (1/2), list_of_inputs))
```

```
list_of_inputs = [1, 2, 3, 4, 5]

res = [x ** (1/2) for x in list_of_inputs]
```

Filter

- Bom conhecer mas não é necessário usar
- Sem problemas com tipagem
- Facilmente substituível com list comprehension com if

```
list_of_inputs = [1, 2, 3, 4, 5]

def function_to_apply(x: int) -> bool:
    return x %2 == 0

# get only even numbers
res = list(filter(function_to_apply, list_of_inputs))
```

```
list_of_inputs = [1, 2, 3, 4, 5]

res = list(filter(lambda x: x % 2, list_of_inputs))
```

```
list_of_inputs = [1, 2, 3, 4, 5]

res = [x for x in list_of_inputs if x % 2 == 0]
```

Map e Filter compostos

- Problemas com tipagem se usar lambda no map
- Facilmente substituível com list comprehension com if

```
list_of_inputs = [1, 2, 3, 4, 5]

def square_root(x: int) -> float:
    return x * (1 / 2)

def is_even(x: int) -> bool:
    return x %2 == 0

res = list(square_root, filter(is_even, list_of_inputs)))
```

Funções de Alta Ordem HOF

```
list_of_inputs = [1, 2, 3, 4, 5]

res = [x ** (1 / 2) for x in list_of_inputs if x % 2 == 0]
```

Reduce

- Forma de "reduzir" uma lista para um único elemento
- Passe uma função que combina dois elementos, ele faz o resto
- Pode ter valor inicial predefinido
- Sem problemas com tipagem
- Não existe "tradução" para list comprehension

```
from functools import reduce

list_of_inputs = [1, 2, 3, 4, 5]

def function_to_apply(x: int, y: int) -> int:
    return x * y

res = reduce(function_to_apply, list_of_inputs)
```

```
from functools import reduce

list_of_inputs = [1, 2, 3, 4, 5]

res = reduce(lambda x, y: x * y, list_of_inputs)
```

Decorators

- Uma HOF que recebe uma função e retorna uma função
- Utilizar type hits é complexo (não é recomendável)

```
import time


def print_start(f):
    def new_f(x):
        print("Started...")
        return f(x)

    return new_f

def print_end(f):
    def new_f(x):
        y = f(x)
        print("Ended.")
        return y

    return new_f

@print_start
@print_end
def wait_and_multiply_by_2(x: float) -> float:
    time.sleep(5)
    return 2 * x


x = wait_and_multiply_by_2(3)
```