ebac-logo

Módulo 05 | Python: Programação Funcional

Caderno de **Aula**

Desenvolvedor Enzo Schitini

Tópicos

1. Função lambda;
2. Função map;
3. Função filter;
4. Função reduce.

Aulas

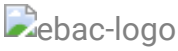
0. Paradigmas de Programação

Estilos de programação.

1. Imperativa;
2. Funcional;
3. Orientada a objetos.

O Python é uma linguagem multi-paradigma.

Este módulo é sobre **programação funcional**, que é o estilo que busca manipular dados com muitas e pequenas funções.



1. Função lambda

1.1. Definição

Função anônima (sem nome) com bloco de código super enxuto e que pode ser salva em uma variável. Em geral é utilizada com outros métodos funcionais como `map`, `filter`, e `reduce`.

```
variavel = lambda params: expressão
```

Exemplo: Função `lambda` para extrair provedor de e-mail.

```
extrair_provedor_email = lambda email: email.split(sep='@')[-1]

email = 'andre.perez@gmail.com'
print(email)

provedor_email = extrair_provedor_email(email)
print(provedor_email)
```

Exemplo: Função `lamba` com estruturas condicionais.

```
numero_e_par = lambda numero: True if numero % 2 == 0 else False

numeros = range(0, 10)

for numero in numeros:
  if numero_e_par(numero) == True:
    print(f'O número {numero} é par!')
```

1.2. Função de alta ordem

São funções que recebem outras funções para parâmetro ou retornam outra função.

Exemplo: Juros compostos dinâmico.

- Definição.

```
def retorno(juros: float):
  return lambda investimento: investimento * (1 + juros)
```

- Instanciação.

```
retorno_5_porcento = retorno(juros=0.05)
retorno_10_porcento = retorno(juros=0.10)
```

```
valor_final = retorno_5_porcento(investimento=1000)
print(valor_final)

valor_final = retorno_10_porcento(investimento=1000)
print(valor_final)
```

- Uso.

```
anos = 10
valor_inicial = 1000
valor_final = valor_inicial

for ano in range(1, anos+1):
  valor_final = retorno_5_porcento(investimento=valor_final)

valor_final = round(valor_final, 2)
print(valor_final)
```

```
anos = 10
valor_inicial = 1000
valor_final = valor_inicial

for ano in range(1, anos+1):
  valor_final = retorno_10_porcento(investimento=valor_final)

valor_final = round(valor_final, 2)
print(valor_final)
```

2. Função map

2.1. Definição

Aplica uma função em todos os elementos de uma coleção (`list`, `dict`, etc.) e retorna **todos** os elementos transformados.

```
 variavel = map(função, coleção)
```

```
numeros = [1, 2, 3]

numeros_ao_cubo = map(lambda num: num ** 3, numeros)

print(list(numeros_ao_cubo))
```

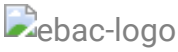
2.2. Função de alta ordem

Exemplo: Função `lambda` para extrair provedor de e-mail (1 parâmetro).

```
emails = ['andre.perez@gmail.com', 'andre.perez@live.com', 'andre.perez@yahoo.com']
extrair_provedor_email = lambda email: email.split(sep='@')[-1]


provedores = []
for email in emails:
  provedor = extrair_provedor_email(email)
  provedores.append(provedor)

print(provedores)


provedores = map(extrair_provedor_email, emails)
print(provedores)


provedores = list(map(extrair_provedor_email, emails))
print(provedores)


provedores = map(lambda email: email.split(sep='@')[-1], emails)
...
print(list(provedores))
```

- **Exemplo**: Investimento (Mais de 1 parâmetros).

```
anos = [10, 10, 10]
taxas_juros = [0.05, 0.10, 0.15]
valores_iniciais = [1000, 1000, 1000]

def retorno(valor_inicial: float, taxa_juros: float, anos: int) -> float:
  valor_final = valor_inicial
  for ano in range(1,anos+1):
    valor_final = valor_final * (1+taxa_juros)
  return round(valor_final, 2)

cenarios = list(map(retorno, valores_iniciais, taxas_juros, anos))
print(cenarios)
```



3. Função filter

3.1. Definição

Aplica uma função lógica (que retorna um booleano) em todos os elementos de uma coleção (`list`, `dict`, etc.) e retorna **apenas** aqueles que resultaram em verdadeiro (`True`).

```
 variavel = filter(função, coleção)
```

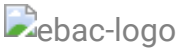
```
numeros = [1, 2, 3]

numeros_par = filter(lambda num: num % 2 == 0, numeros)

print(list(numeros_par))
```

3.2. Função de alta ordem

Exemplo: Função `lambda` para extrair provedor de e-mail.

```
emails = ['andre.perez@gmail.com', 'andre.perez@live.com', 'andre.perez@yahoo.com']
provedor_da_google = lambda email: 'gmail' in email


emails_google = []
for email in emails:
  if provedor_da_google(email) == True:
```

```
    emails_google.append(email)

print(emails_google)


emails_google = filter(provedor_da_google, emails)
print(emails_google)


emails_google = list(filter(provedor_da_google, emails))
print(emails_google)


emails_google = filter(lambda email: 'gmail' in email, emails)
...
print(list(emails_google))
```



4. Função reduce

4.1. Definição

Aplica uma função a todos os elemento de uma coleção, dois a dois, e retorna **apenas** um elemento.

```
 variavel = reduce(função, coleção)


numeros = [1, 2, 3]

from functools import reduce

soma = reduce(lambda x, y: x + y, numeros)
print(soma)

     6
```

4.2. Função de alta ordem

Exemplo: Encontrar maior número em uma lista.

```
def maior_entre(primeiro: int, segundo: int) -> int:
  return primeiro if primeiro >= segundo else segundo
```

```
primeiro = 11
segundo = 11

print(maior_entre(primeiro=primeiro, segundo=segundo))
```

```
from random import random

print(random())
```

```
     0.8674143678027258
```

```
from random import random

numeros = [round(100 * random()) for _ in range(0, 100)]
print(numeros)
```

```
from random import random

def soma(v1:int) -> int:
    return v1 + 5

numeros = [(soma(3)) for _ in range(0, 100)]
print(numeros)
```

```
maior_numero = reduce(maior_entre, numeros)
print(maior_numero)
```

```
maior_numero = reduce(lambda primeiro, segundo: primeiro if primeiro >= segundo else segur
print(maior_numero)
```

4.3. Compossibilidade

Exemplo: Combinação de métodos funcionais.

```
from random import random

numeros = [round(100 * random()) for _ in range(0, 100)]
print(numeros)
```

- Eleve os números ao quadrado.

```
numeros_ao_quadrado = map(lambda numero: numero ** 2, numeros)
```

- Filtra os números ímpares.

```
numeros_impares = filter(lambda numero: numero % 2 != 0, numeros_ao_quadrado)
```

- Soma todos os números.

```
soma_numeros = reduce(lambda x, y: x + y, numeros_impares)
print(soma_numeros)
```

- Todos os métodos de uma vez.

```
soma_numeros = reduce(lambda x, y: x + y, filter(lambda numero: numero % 2 != 0, map(lambd
print(soma_numeros)
```

PROGRAMAS

```
print("Ok")
```

```
Ok
```

```
def ok():
    print("Nome")

o = [ok() for _ in range(0, 5)]
```

```
Nome
Nome
Nome
Nome
Nome
```