



FUNDAÇÃO EDSON QUEIROZ
UNIVERSIDADE DE FORTALEZA
CURSO DE ANÁLISE E DESENVOLVIMENTO DE SISTEMAS

ATIVIDADE FINAL - PROGRAMAÇÃO FUNCIONAL

CAIO NEPOMUCENO SANTOS - 2124641
CARLOS ALBERTO ALVES JUNIOR - 2124671
MAURICIO GARCIA VELOSO JUNIOR - 2124647
PEDRO MOREIRA MAMEDE - 2213870

FORTALEZA
2023

1. Papel dos Integrantes da Equipe e Disponibilização do código

Nome	Função
Caio Nepomuceno Santos	Testador
Carlos Alberto Alves Junior	Analista de Requisitos
Maurício Garcia Veloso Junior	Analista de Requisitos
Pedro Moreira Mamede	Desenvolvedor

O código e este documento estão disponíveis no repositório abaixo do github:
https://github.com/caionps/api_aluguel_carros

2. Requisitos Não Funcionais

A API tem como objetivo gravar os aluguéis realizados em uma locadora de carros. Além disso, é possível buscar aluguéis específicos, atualizá-los e deletá-los.

Os dados são todos salvos em um arquivo CSV para que seja possível, caso necessário, imprimi-los ou acessá-los todos de uma vez. As informações gravadas são: id do aluguel, data de início e fim do aluguel, os quilômetros rodados durante o período, a marca e o modelo do carro, o CPF do cliente, o nome do cliente e o valor cobrado do aluguel.

Também é função da API tratar os dados e retornar erros caso as informações fornecidas estejam erradas ou incompletas para determinado método HTTP.

3. Requisitos Funcionais

[RF01]

Deve realizar a validação do formato das datas, tanto de entrada como de saída. O padrão utilizado é o ISO 8601 (yyyy-mm-dd). Isso está definido na função “**validar_data()**”.

[RF02]

Deve indicar quando o dado do CPF é diferente de 11 dígitos. É através da função “**validar_cpf()**” que isso é realizado.

[RF03]

Realiza a validação de um objeto json, tanto da existência de todos os conjuntos de pares chave e valor, como de cada valor específico das chaves. Nela utilizamos o conceito de **Função de Continuação**. Nesta função, utilizamos outras funções declaradas anteriormente (validar_data() e

validar_cpf()) para fazer parte do trabalho de validação que está definida na função “**validar_aluguel**”.

[RF04]

Padronizar todas as palavras e colocá-las em caixa alta para só aí poderem ser salvas no arquivo .csv. Isto é realizado utilizando o conceito de **Monad List** na função “**caixa_alta()**”. Nela, aproveitamos outros conceitos da programação funcional que são o das **Funções Lambda**:

```
lambda x: str(x).upper()
```

e o das **Funções de Alta Ordem** no momento que utilizamos a função **map()** para aplicar a função lambda apenas nos valores que são strings.

[RF05]

Deve abrir um arquivo csv de dentro do diretório atual da aplicação. Isto é realizado na função “**abre_csv()**”.

[RF06]

Deve criar um número de id sequencial de acordo com os dados que já estiverem gravados no arquivo csv. Para isto foi utilizado o conceito da programação funcionam chamada de **Closure**, onde abrimos o arquivo csv através da função “**id_aux()**” e a partir da função interna “**gerar_id()**” geramos um novo id caso alguma nova inserção no arquivo seja realizada.

[RF07]

Deve unir dados e colocá-los na estrutura de dicionário para que depois seja transformado em um objeto json. Isto é realizado na função: “**criar_json()**”.

[RF08]

Deve criar um novo aluguel no arquivo csv ao receber uma requisição HTTP do método ‘**POST**’ caso os dados do json recebido estejam corretos. Para isto utilizou-se uma **List Comprehension** . Nela lemos os valores do json e os adicionamos a uma lista para que seja depois inserida no arquivo csv. Além disso, a closure é utilizada aqui para gerar o valor do novo id. Tudo isso está presente na função ‘**create()**’. Um exemplo de json a ser passado para a aplicação está abaixo:

```
{
    "dt_inicio_aluguel": "2023-08-01",
    "dt_fim_aluguel": "2023-08-05",
    "km_rodados_aluguel": 750,
    "carro_marca_modelo": "Fiat Argo",
    "nm_cliente": "Seu Fulano",
```

```
    "cpf_cliente": "74244814019",  
    "valor": 950  
}
```

[RF09]

Deve devolver um json com as informações do id requerido através do método HTTP '**GET**'. Caso o id não exista, deve retornar uma mensagem amigável de retorno. A função que realiza este procedimento é a '**read()**'. Um exemplo de json para a requisição é o abaixo:

```
{  
    "id_aluguel" : 2  
}
```

[RF10]

Realizar a correção ou alteração de algum registro no arquivo csv a partir de uma requisição do método 'HTTP' 'PUT' passando um json contendo o id de correção e o restante das informações. Os seus valores devem ser validados para que o método realize a alteração. Essa ação está definida na função: **update()**. Um exemplo é o json abaixo:

```
{  
    "id_aluguel" : 1  
    "dt_inicio_aluguel": "2023-08-01",  
    "dt_fim_aluguel": "2023-08-05",  
    "km_rodados_aluguel": 750,  
    "carro_marca_modelo": "Chevrolet Onix",  
    "nm_cliente": "Nome Editado",  
    "cpf_cliente": "74244814019",  
    "valor": 950  
}
```

[RF11]

Realizar o Delete através da requisição do método 'HTTP' 'DELETE'. No corpo da requisição deve ser especificado o valor da chave id_aluguel o qual será deletado do arquivo csv. Tudo isso está definido na função **delete()**. Exemplo:

```
{  
    "id_aluguel" : 3  
}
```

[RF12]

Deve possuir **testes unitários** para a aplicação. Os testes estão especificados no documento teste_app.py, onde a função **test_00_create()** realiza toda a parte de testes para as requisições 'POST', a função **test_01_read()** realiza toda a parte de testes para as requisições 'GET', a função **test_02_update()** realiza toda a parte de testes para as requisições 'PUT' e a função **test_03_delete()** realiza toda a parte de testes para as requisições 'DELETE'. O teste foi gerado utilizando a biblioteca unittest do python. É possível verificar a cobertura dos testes utilizando a biblioteca pytest, onde é possível verificar 80% de cobertura.

4. Auxílio do chatbot (Bard do Google)

O grupo utilizou a ia do Google chamada Bard. No chat, foram perguntados os conceitos de programação funcional a serem aplicados no trabalho como pedido nos arquivos e materiais da proposta da atividade no AVA. A cópia da conversa está abaixo:

Pergunta: na programação funcional, o que é uma função de continuação?

Resposta:

Na programação funcional, uma função de continuação é uma função que recebe como entrada uma função e um valor, e retorna uma função. A função retornada é então executada com o valor como entrada.

Funções de continuação são frequentemente usadas para implementar chamadas de procedimento atrasadas, que são chamadas de procedimento que são adiadas até que uma condição seja atendida. Por exemplo, a seguinte função de continuação adia a execução de uma função até que um valor seja recebido:

```
def continuacao(f, x):  
    def funcao_continuada(y):  
        return f(x, y)  
    return funcao_continuada
```

Esta função pode ser usada para implementar uma chamada de procedimento atrasada da seguinte forma:

```
def f(x, y):  
    print(x, y)
```

```
g = continuacao(f, 10)  
g(20)
```

A saída deste código é a seguinte:

10 20

Funções de continuação também podem ser usadas para implementar recursão tail-call, que é uma forma de recursão em que a chamada recursiva é o último comando executado em uma função. Por exemplo, a seguinte função de continuação implementa a recursão tail-call para calcular o fatorial de um número:

```
def fatorial(n):  
    def continuacao(f, x):  
        if x == 0:  
            return 1  
        else:  
            return f(lambda y: continuacao(f, x - 1) * y)  
    return continuacao(lambda y: y, n)
```

Esta função pode ser usada para calcular o fatorial de 5 da seguinte forma:

```
print(fatorial(5))
```

A saída deste código é a seguinte:

```
120
```

Funções de continuação são uma ferramenta poderosa que pode ser usada para implementar uma variedade de funcionalidades na programação funcional.

Pergunta: isso é o mesmo que continuation passing style?

Resposta: Sim, isso é o mesmo que continuation passing style (CPS). CPS é um estilo de programação em que funções recebem uma continuação como argumento. A continuação especifica o que deve ser feito com o valor retornado da função.

No exemplo da função `continuacao()`, a continuação é uma função que recebe um valor como entrada e retorna um valor. A função `f()` é então chamada com o valor retornado pela continuação como entrada.

No exemplo da função `fatorial()`, a continuação é uma função que recebe um valor como entrada e retorna um valor. A função `continuacao()` é chamada com a função `lambda y: y` como entrada. A função `lambda y: y` retorna o valor que é passado como entrada.

Portanto, em ambos os casos, a função `continuacao()` é uma função de continuação que recebe uma função e um valor como entrada, e retorna uma função. A função retornada é então executada com o valor como entrada.

A principal diferença entre CPS e o estilo de programação usual é que, em CPS, o controle é passado explicitamente na forma de uma continuação. Isso pode tornar o código mais legível e fácil de entender, especialmente para funções recursivas.

Além disso, CPS pode ser usado para implementar uma variedade de funcionalidades que são difíceis ou impossíveis de implementar no estilo de programação usual. Por exemplo, CPS pode ser usado para implementar chamadas de procedimento atrasadas, recursão tail-call, e programação assíncrona.

Pergunta: qual a diferença para função de alta ordem?

Resposta: Funções de alta ordem e funções de continuação são duas técnicas de programação funcional que permitem que funções sejam usadas como valores. No entanto, existem algumas diferenças importantes entre as duas técnicas.

Funções de alta ordem são funções que podem tomar outras funções como argumentos ou retornar funções como valores. Por exemplo, a função `filter()` é uma função de alta ordem que toma uma função como argumento e retorna uma nova função que aplica a função passada como argumento a cada elemento de uma sequência.

Funções de continuação são funções que recebem uma continuação como argumento. A continuação especifica o que deve ser feito com o valor retornado da função.

A principal diferença entre funções de alta ordem e funções de continuação é que funções de alta ordem não têm um controle explícito sobre o fluxo de execução. Em vez disso, elas dependem de outras funções para controlar o fluxo de execução.

Por exemplo, a função `filter()` não tem controle sobre o fluxo de execução. Ela simplesmente aplica a função passada como argumento a cada elemento de uma sequência. O fluxo de execução é controlado pela função que chama a função `filter()`.

Funções de continuação, por outro lado, têm controle explícito sobre o fluxo de execução. Elas recebem uma continuação como argumento, que especifica o que deve ser feito com o valor retornado da função.

Por exemplo, a função `continuacao()` é uma função de continuação que recebe uma função e um valor como entrada, e retorna uma função. A função retornada é então executada com o valor como entrada.

Portanto, a principal diferença entre funções de alta ordem e funções de continuação é que funções de alta ordem não têm um controle explícito sobre o fluxo de execução, enquanto funções de continuação têm controle explícito sobre o fluxo de execução.

Aqui está uma tabela que resume as principais diferenças entre funções de alta ordem e funções de continuação:

Característica	Função de alta ordem	Função de continuação
Funções como argumentos	Sim	Sim
Funções como valores retornados	Sim	Sim
Controle explícito sobre o fluxo de execução	Não	Sim

Em alguns casos, funções de alta ordem e funções de continuação podem ser usadas de forma intercambiável. No entanto, em outros casos, uma técnica é mais adequada do que a outra.

Pergunta: então ao utilizar o filter() eu estou utilizando uma função de alta ordem

Resposta: Sim, ao utilizar a função filter() você está utilizando uma função de alta ordem. A função filter() toma uma função como argumento e retorna uma nova função que aplica a função passada como argumento a cada elemento de uma sequência.

No exemplo a seguir, a função filter() é usada para retornar uma lista de todos os números pares de uma lista de números:

Python

```
def pares(x):  
    return x % 2 == 0
```

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

```
pares_nums = filter(pares, nums)
```

```
print(pares_nums)
```

Use o código com cuidado. Saiba mais

A saída deste código é a seguinte:

```
[2, 4]
```

Neste exemplo, a função pares() é usada como argumento para a função filter(). A função filter() então cria uma nova função que aplica a função pares() a cada elemento da lista nums. A nova função é então usada para retornar uma lista de todos os números pares da lista nums.

Portanto, a função filter() é uma função de alta ordem porque ela toma uma função como argumento.

foto do perfil

o map também é uma função de alta ordem?

Sim, o map() também é uma função de alta ordem. O map() toma uma função como argumento e retorna uma nova função que aplica a função passada como argumento a cada elemento de uma sequência.

No exemplo a seguir, o map() é usado para retornar uma lista dos quadrados de uma lista de números:

Python

```
def quadrado(x):  
    return x * x
```



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

```
quadrados = map(quadrado, nums)
```

```
print(quadrados)
```

Use o código com cuidado. Saiba mais

A saída deste código é a seguinte:

```
[1, 4, 9, 16, 25]
```

Neste exemplo, a função `quadrado()` é usada como argumento para a função `map()`. A função `map()` então cria uma nova função que aplica a função `quadrado()` a cada elemento da lista `nums`. A nova função é então usada para retornar uma lista dos quadrados dos números da lista `nums`.

Portanto, o `map()` é uma função de alta ordem porque ele toma uma função como argumento.

Aqui está uma lista de outras funções de alta ordem comuns:

```
reduce()
```

```
fold()
```

```
sort()
```

```
filter()
```

```
map()
```

```
zip()
```

```
unzip()
```

```
lambda()
```

Essas funções são frequentemente usadas em programação funcional para implementar padrões comuns.

Pergunta: o que é um monad?

Resposta: Em programação funcional, um monad é uma estrutura que combina fragmentos de programa (funções) e envolve seus valores de retorno em um tipo com computação adicional.

Monads são frequentemente usados para lidar com efeitos colaterais, como I/O, mutação de estado e acesso a recursos compartilhados. Eles também podem ser usados para simplificar a composição de funções.

Definição formal

Um monad é um tipo abstrato com os seguintes requisitos:

Unidade: Há uma função `return()` que leva um valor de tipo `A` e retorna um monad de tipo `M[A]`.

Aplicabilidade: Há uma função `bind()` que leva um monad de tipo `M[A]` e uma função `f` que leva um valor de tipo `A` e retorna um monad de tipo `M[B]`. A função `bind()` retorna um monad de tipo `M[B]`.

Exemplos

Alguns exemplos de monads incluem:

Lista: O monad lista é um monad que representa uma lista de valores. A função `return()` retorna uma lista vazia. A função `bind()` aplica uma função a cada elemento da lista.

Python

```
def f(x):  
    return x * x
```

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

```
quadrados = list(map(f, nums))
```

Use o código com cuidado. Saiba mais

Opção: O monad opção é um monad que representa um valor opcional. A função `return()` retorna uma opção `Some()` com o valor especificado. A função `bind()` aplica uma função ao valor se a opção for `Some()`.

Python

```
def f(x):  
    return x * x
```

```
num = 10
```

```
quadrado = Option(num).bind(f)
```

Use o código com cuidado. Saiba mais

IO: O monad IO é um monad que representa uma operação de I/O. A função `return()` retorna uma operação de I/O que não faz nada. A função `bind()` aplica uma função ao resultado de uma operação de I/O.

Python

```
def f(x):  
    return x * x
```

```
num = IO(print("Digite um número: ")).bind(lambda y: IO(input())).bind(f)
```

Use o código com cuidado. Saiba mais

Aplicações

Monads são uma ferramenta poderosa que pode ser usada para implementar uma variedade de funcionalidades na programação funcional. Alguns exemplos de aplicações de monads incluem:

Lidando com efeitos colaterais: Monads podem ser usados para lidar com efeitos colaterais, como I/O, mutação de estado e acesso a recursos compartilhados. Por exemplo, o monad IO pode ser usado para implementar operações de I/O que produzem resultados observáveis.

Simplificando a composição de funções: Monads podem ser usados para simplificar a composição de funções. Por exemplo, o monad lista pode ser usado para implementar a operação de concatenação de listas.

Implementando padrões comuns: Monads podem ser usados para implementar padrões comuns, como mapeamento, filtragem e redução.

Conclusão

Monads são uma ferramenta poderosa que pode ser usada para implementar uma variedade de funcionalidades na programação funcional. Eles podem ser usados para lidar com efeitos colaterais, simplificar a composição de funções e implementar padrões comuns.

Pergunta: O que seria então uma closure?

Resposta: Uma closure é uma função que referencia variáveis livres no contexto léxico. Uma closure ocorre normalmente quando uma função é declarada dentro do corpo de outra, e a função interior referencia variáveis locais da função exterior.

Em outras palavras, uma closure é uma função que pode acessar variáveis que estão fora do seu escopo.

Exemplo

O seguinte código define uma closure que referencia a variável x:

```
Python
def outer(x):
    def inner():
        print(x)
    return inner
```

```
x = 10
```

```
closure = outer(x)
```

```
closure()
```

Use o código com cuidado. Saiba mais

A saída deste código é a seguinte:

```
10
```

Neste exemplo, a função outer() declara a função inner() dentro do seu corpo. A função inner() referencia a variável x, que é uma variável local da função outer().

Quando a função outer() é chamada, ela retorna a função inner(). A função inner() é então chamada, e ela pode acessar a variável x.

Funções de alta ordem

Closures são frequentemente usadas em funções de alta ordem. Uma função de alta ordem é uma função que pode tomar outras funções como argumentos ou retornar funções como valores.

No exemplo a seguir, a função `map()` é uma função de alta ordem que toma uma função como argumento e retorna uma nova função que aplica a função passada como argumento a cada elemento de uma sequência.

Python

```
def quadrado(x):  
    return x * x
```

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

```
quadrados = map(quadrado, nums)
```

```
print(quadrados)
```

Use o código com cuidado. Saiba mais

A saída deste código é a seguinte:

```
[1, 4, 9]
```

Neste exemplo, a função `quadrado()` é uma função de alta ordem. A função `map()` toma a função `quadrado()` como argumento e retorna uma nova função que aplica a função `quadrado()` a cada elemento da lista `nums`.

A nova função é então usada para retornar uma lista dos quadrados dos números da lista `nums`.

Conclusão

Closures são uma ferramenta poderosa que pode ser usada em uma variedade de contextos. Elas podem ser usadas para lidar com efeitos colaterais, simplificar a composição de funções e implementar padrões comuns.