Relatório

- Problema:

Em um grupo de 5 pessoas, separar alguns dados específicos e descobrir quais delas estão com obesidade.

- Resolução:

Python 3 foi a linguagem escolhida para esse projeto.
Primeiro foi declarado uma lista com dicionários onde os dados das pessoas foram armazenados:

```
peoples = [{'name': 'Joao', 'bmi': 27},
           {'name': 'Cleiton', 'bmi': 21},
           {'name': 'Julia', 'bmi': 16},
           {'name': 'Carlos', 'bmi': 43},
           {'name': 'Daniela', 'bmi': 31}
]
```

Utilizaremos essa lista durante todo o projeto, separaremos e trataremos os dados utilizando a programação imperativa e depois a programação funcional.

Guardar o nome de cada pessoa

✓ Usando a programação imperativa:

Primeiro criaremos uma função para capturar os nomes dentro da lista e com o parâmetro *people_list* ela receberá a lista com os dados pessoais.
Dentro dessa função, um laço de repetição *for* tratará cada um dos dicionários e adicionará o item descrito como *name* de cada um em uma lista chamada de *names_list*.

```
def get_names(peoples_list):
    '''
    Function to get all names in a list with dictionaries
    :param peoples_list: list with peoples data
    :return: list with all names
    '''
    for people in peoples_list:
        names_list.append(people['name'])
    return names_list
```

Essa função será chamada no script a seguir:

```
#Geting names
names_list = []
names_list = get_names(peoples_list=peoples)
other_names_list = get_names(peoples_list=peoples)
print(f'Names: {names_list}')
print(f'Other names: {other_names_list}')
```

A partir desse script é possível descobrir o porquê da programação imperativa não ser a melhor opção para esse problema.

A variável *names_list* não é exclusiva da função, ou seja, ela pode sofrer alterações externas ao programa e isso faz com que a função não retorne os nomes corretamente quando ela for chamada mais de uma vez.

Observe o que o programa imprime quando é executado:

```
Names: ['Joao', 'Cleiton', 'Julia', 'Carlos', 'Daniela', 'Joao', 'Cleiton', 'Julia', 'Carlos', 'Daniela']
Other names: ['Joao', 'Cleiton', 'Julia', 'Carlos', 'Daniela', 'Joao', 'Cleiton', 'Julia', 'Carlos', 'Daniela']
```

Ambas as listas acabaram com o mesmo resultado, isso acontece porque quando a função foi chamada pela segunda vez, a lista *names_list* já tinha valores inseridos e os mesmos valores foram inseridos a ela de novo.

Esse problema é resolvido quando utilizamos a programação funcional.

✓ Utilizando a programação funcional:

Umas das características desse tipo de programação é a existência de múltiplas funções que trabalham de forma unificada para a resolução de um problema.

Veja o código:

```python
#Geting names using map()
names_list = list(map(lambda p: p['name'], peoples))
other_names_list = list(map(lambda p: p['name'], peoples))

print(f'Names: {names_list}')
print(f'Other names: {other_names_list}')
```

Utilizei funções *lambda*(uma pequena função anônima) junto com a função map, isso fez com que cada função fosse completamente independente uma da outra, o que evitou conflitos entre as variáveis de dentro e fora das funções.

Veja o resultado:

```
Names: ['Joao', 'Cleiton', 'Julia', 'Carlos', 'Daniela']
Other names: ['Joao', 'Cleiton', 'Julia', 'Carlos', 'Daniela']
```

Diferente do que aconteceu no método anterior, desta vez as duas listas (*names_list* e *other_names_list*) receberam os nomes das 5 pessoas presentes na lista *peoples* somente uma vez.

Descobrir quem tem obesidade

✓ Usando a programação imperativa:

Utilizaremos o mesmo método do problema passado, criaremos uma função com um laço de repetição *for* que tratará cada um dos dicionários, só que desta vez, quando o item *bmi*(IMC em inglês) for menor ou igual a 30(o que define obesidade grau I), o laço adicionará os dados da pessoa na lista *people_with_obesity*.

```
def search_obese(peoples_list):
    '''
    Function to get people with BMI higher than 30 in as list with dictionaries
    :param peoples_list: list with peoples data
    :return: list with obese people
    '''
    for people in peoples_list:
        if people['bmi'] >= 30:
            people_with_obesity.append(people)
    return people_with_obesity
```

A função será chamada assim:

```
#Geting people with obesity
people_with_obesity = []
people_with_obesity = search_obese(peoples_list=peoples)
print(f'Peoples with obesity: {people_with_obesity}')
```

A saída do código fica assim:

```
Peoples with obesity: [{'name': 'Carlos', 'bmi': 43}, {'name': 'Daniela', 'bmi': 31}]
```

O resultado está correto, porém, com o mesmo problema do seu antepassado: A falta de imutabilidade nos resultados.

Isso porque quando chamarmos a função duas vezes esta é a saída:

```
people_with_obesity = []
people_with_obesity = search_obese(peoples_list=peoples)
other_people_with_obesity = search_obese(peoples_list=peoples)
print(f'Peoples with obesity: {people_with_obesity}')
print(f'Other peoples with obesity: {other_people_with_obesity}')
```

```
Peoples with obesity: [{'name': 'Carlos', 'bmi': 43}, {'name': 'Daniela', 'bmi': 31}, {'name': 'Carlos', 'bmi': 43}, {'name': 'Daniela', 'bmi': 31}]
Other peoples with obesity: [{'name': 'Carlos', 'bmi': 43}, {'name': 'Daniela', 'bmi': 31}, {'name': 'Carlos', 'bmi': 43}, {'name': 'Daniela', 'bmi': 31}]
```

Percebe-se que a variável compartilhada entre a função e o resto do programa mais uma vez faz com que os valores sejam repetidos.

✓ Utilizando a programação funcional:

Dessa vez, em vez de usar uma função *lambda* declararemos uma função do jeito comum, porém com variáveis exclusivas para evitar efeitos colaterais no código.

Veja a seguir:

```
def search_obese(peoples_list):
    '''
    Function to search people with obesity in a list with dictionaries
    :param peoples_list: list with peoples data
    :return: list with obeses
    '''
    obese = []
    if peoples_list['bmi'] >= 30:
        obese.append(peoples_list)
    return obese
```

Perceba que a lista *obese* é declarada dentro da função, desse jeito ela não vai sofrer nenhuma alteração fora dela, o que garante maior segurança ao código.

A função será chamada neste script:

```
#Geting people with obesity using list()
people_with_obesity = list(filter(search_obese, peoples))
other_people_with_obesity = list(filter(search_obese, peoples))
print(f'Peoples with obesity: {people_with_obesity}')
print(f'Other peoples with obesity: {other_people_with_obesity}')
```

Utilizaremos também a função *filter*, que filtra os elementos de uma sequência baseando-se em uma função (no caso a *search_obese* que criamos).

A saída ficou assim:

```
Peoples with obesity: [{'name': 'Carlos', 'bmi': 43}, {'name': 'Daniela', 'bmi': 31}]
Other peoples with obesity: [{'name': 'Carlos', 'bmi': 43}, {'name': 'Daniela', 'bmi': 31}]
```

Mais uma vez, o valor desejado foi recebido 2 vezes sem sofrer alterações, isso se deve a imutabilidade presente na programação funcional.

Encontrar maior valor de IMC entre as pessoas

✓ Usando a programação imperativa:

Desta vez, ao invés de usar funções usaremos o laço de repetição *for* diretamente no código. Observe:

```
#Geting higher BMI
bmi_list = []
for people in peoples:
    if people['bmi'] >= 30:
        bmi_list.append(int(people['bmi']))
higher_bmi = 0
for bmi in bmi_list:
    if higher_bmi < bmi:
        higher_bmi = bmi
print(f'Higher BMI: {higher_bmi}')
```

O primeiro laço *for* irá verificar o *bmi* (IMC) de cada pessoa e colocará os valores maiores que 30 dentro da lista *bmi_list*.

O segundo laço *for* descobrirá qual é o maior valor de IMC entre os obesos.

A saída do programa fica assim:

```
Higher BMI: 43
```

O script em si não possui nenhum problema ou erro, porém mais uma vez as listas *bmi_list* e *highter_bmi* podem sofrer alterações em outras partes do código o que pode comprometer os resultados.

✓ Utilizando a programação funcional:

Utilizaremos funções *lambda* mais uma vez, além da função *reduce*. O script irá guardar o imc de cada pessoa na lista *bmi_list* e depois será usado pela função *reduce* para calcular qual o maior valor de imc entre as amostras.

```
#Geting higher BMI using reduce()
bmi_list = list(map(lambda p: p['bmi'], peoples))
higher_bmi = reduce(lambda a, b: a if a > b else b, bmi_list)
print(f'Higher BMI: {higher_bmi}')
```

Sendo assim, o script imprime isso:

```
Higher BMI: 43
```

Embora o resultado seja o mesmo obtido quando utilizamos a programação imperativa, a estrutura deste código está muito mais segura e estável.

Conclusão

A programação funcional trata a computação como uma sequência de funções e não como uma sequência de ações que mudam o estado ou a estrutura do programa. Isso traz várias vantagens como melhor legibilidade e manutenção do código, código simples e conciso.

Além de tudo isso, podemos criar funções que não sofrem interferências de meios externos (variáveis de fora), evitando assim efeitos colaterais. Um bom exemplo é a função *lambda*, que é perfeita quando precisamos de uma função sem nome por um curto período de tempo.