Nomes Com Agua Potável

Um dos princípios é que nomes de classes métodos, variáveis, módulos, pacotes, e etc, sejam significativos, que apartir de seus nomes consigamos reconhece-los como uma parte da nossa historia, e aonde eles se situam nela, para que ao longo do código consigamos entender com eles transitam dentro da historia ou código.

Se eu lhe perguntasse o que isso embaixo que dizer

```
d = ""
```

Agora se eu desse o código

```
import re

def verify(p):

  number = str(p.number)
  d = re(r"\D*", '', number)

  if len(d) != 11:
      return False
  elif re.match( f"{d[0]}{'{11}'}", str(d)):
      return False
  else:
      return True
```

É uma das coisas que podemos ver dentro desse código é que ainda a uma incógnita, eventualmente de vai ser uma string, que só tem dígitos numéricos, isso graças a expressão regular que esta substituindo tudo que não e um digito (”\D*”), por nada, ou seja number, talvez quando entre possa ou não ser um atributo do do tipo numero, como integer, long e etc. mas possa ser uma Sting ou outro tipo, mas para não ficar claro:

- O que é p
- O que é o atributo number de p
- O que number é do tipo de dado numero ou não
- O por que d só tem dígitos, qual é a importância disso
- Por que o numero 11
- por que o eu não posso ter o mesmo numero repetido onze vezes
- é o mais importante o que essa função verifica

Veja que são muitas perguntas, veja um coisa, primeiro podemos dizer que já entendemos que o atributo number pode não ser numérico, mas veja que para programação number sub se entende que seja um dado do tipo numero, é isso causa mal entendido. então é melhor utilizar nomes que possam causar mal entendido

vamos reescrever o código

```
def verify_cpf(person: People) -> bool:
```

```
    cpf = str(person.register_number_document)
    only_number_cpf = re.sub(r"\D*", '', cpf)
    TOTAL_NUMBER_CPF = 11

    if len(only_number_cpf) != TOTAL_NUMBER_CPF:
        return False
    elif re.match(
        f"{only_number_cpf[0]}{'{11}'}",
        only_number_cpf
    ):
        return False
    else:
        return True
```

Veja que eventualmente nos temos o mesmo código, e agora ficou muito mais claro, com nomes mais claros, também adicionamos, anotações de que tipo de parâmetros esperamos agora fica claro que espero um tipo dado como argumento que é da classe ou herda da classe People. entretanto só adicionamos uma linha que é a constante “TOTAL_NUMBER_LIMIT”, além de algumas quebras.

O código não aumentou muito, todavia não é o que ocorre é abaixo nos temos um código que melhora o de cima ainda mais:

```
def verify_cpf(register_number_document) -> bool:

  cpf = str(register_number_document)
  not_digit_regex = r"\D*"

  only_number_cpf = re.sub(not_digit_regex, '', cpf)
  TOTAL_NUMBER_CPF = 11

  number_repeated_all_regex = f"{only_number_cpf[0]}{'{11}'}"

  if len(only_number_cpf) != TOTAL_NUMBER_CPF:
      return False
  elif re.match(
      number_repeated_all_regex,
      only_number_cpf
  ):
      return False
  else:
      return True
```

Compreenda que agora, eu tenho as padrões das expressões regulares em variáveis e mesmo que eu não tenha conhecimento de expressões regulares, como construir uma expressão agora tendo um conhecimento unicamente dos métodos, consigo saber o que esta sendo realizado.

E perceba que agora eu também não tenho person, nos parâmetros agora eu recebo o registro direto.

Coisas que podem causar confusão

A primeira coisa que pode causar confusão na hora de nomear algo é que, existe uma dificuldade em nomear num primeiro momento, é o nome deve ser claro de forma que consigamos identificar o objetivo, o que temos que fugir é:

1. Evite utilizar nomes que estão ligados a alguma convenção dentro programação ou envolvendo tecnologia, principalmente se a variável não tem como objetivo evidenciar aquela características ou pode causar confusão

   como referencia algo como number, list, array essas palavras no nome podem causas confusão principalmente se você tem lá, “objects_list” e na verdade, não é um objeto, para usar esses nomes e necessário um certo cuidado

2. Outra coisa é a utilização de nomes semelhantes que podem causar muita confusão entre o que é o que principalmente quando eles são grande como

- copy_markers_html_internal
- copy_mark_tag_html_internal

Se você olhar assim uma embaixo da outra da para notar que são diferentes, mas e dentro de algum código ou e depois de algumas linhas de código, será que da para se dizer o mesmo. Uma outra coisa a nomes parecidos como no exemplo a seguir:

A função seguinte renomeia as colunas de uma tabela então ela recebe um dataframe (tabela) e uma lista com nomes para as colunas, e o objetivo e pegar o primeiro nome tabela e substituir pelo primeiro nome da lista, e assim subsequentemente até o final da lista e retornar a nova tabela.

```
def rename_columns(
  dataframe: DataFrame,
  columns: list):

  TOTAL_COLUMNS = len(columns)
  dataframe_columns = dataframe.columns

  for index in range(TOTAL_COLUMNS):

    column = columns[index]
    dataframe_column = dataframe_columns[index]

    dataframe = dataframe.WithColumnRename(
      dataframe_column , column)

  return dataframe
```

Primeiramente podemos observar que os nomes que temos parecidos columns (parâmetro) e column (a variável que recebe o nome da coluna da lista) e temos dataframe_columns (a lista de colunas da base) e dataframe_column (a variável que recebe o nome da coluna do tabela). e isso pode causar muita confusão dependendo do código o que é o dentro do código poderíamos corrigir isso da seguinte forma

```
import pyspark from DataFrame
import pyspark.sql.functions from col

def rename_columns(
  dataframe: DataFrame,
  list_columns_name: list) -> DataFrame:

  TOTAL_COLUMNS = len(list_columns_name)
  dataframe_columns = dataframe.columns

  for index in range(TOTAL_COLUMNS):

    new_name = list_columns_name[index]
    old_name = dataframe_columns[index]

    dataframe = dataframe.withColumnRename(
      old_name , new_name
)

 return dataframe
```

Agora nos temos para o referencia as variáveis que vão receber os nomes das colunas pelos objetivos que elas tem uma e para ser o nome antigo outra e para ser o nome novo da coluna e eu mudei o nome do parâmetro para list_columns_name, agora esta claro o que é esse parâmetro é o que a dentro dele

3. Evite utilizar números para referencias atributos, como por exemplo item1, item2, item3 por assim por diante
4. Evite a utilização de artigos como A, O, Um, Uma, caso esteja escrevendo o código em português no caso do inglês seria ‘the’ e ‘a’
5. Essa eu considero uma sugestão é a utilização de nomes pronunciáveis, pois e mais fácil para o nosso cérebro, relacionar nomes que conseguimos pronunciar
6. A utilização de nomes passiveis por busca, ou seja nomes que seja possíveis de facilmente se localizar dentro do código

7. Evite escrever nomes como alguém que esta inventando um tipo de criptografia, é aonde os nomes que você criou e necessário os decifra-los para entender o código.
8. Evitar a utilização de prefixo em variáveis de membros, como m_employee, é esse sinceramente é algo que eu não costumo ver, na verdade a primeira vez que eu vi foi no livro do Bob.
9. Evitar usar trocadilhos, gírias ou outras coisas da linguagem coloquial, por motivos obvieis, primeiro os nomes não são claros, segundo pode levar a interpretações erradas, e talvez pessoa não conheça o termo usado.
10. Mantenha uma palavra contexto abstrato ao investe de utilizar múltiplas palavras como **receive** e **get**, se as duas significam que você recebe uma informação ao invés de ter as duas no código pra uma e para outra mantenha um para representar o mesmo contexto, o que vai deixar o código mais uniforme.
11. Use nome que tenham a ver com programação informática, termos matemáticos, que são coisas mais presentes na vida do programador, tornando o código mais fácil de leitura.
12. Um conselho que no livro e nos recomentado é a utilização de nomes que tenham a ver com com o domínio do problema, ou seja nomes ou jargões que estão relacionados aquela área de negocio, estritamente onde o problema se insere, isso e um bom ponto pois permite que alguém que tenha duvida, consiga perguntar para alguém de negocio que tenha mais fluência com o problema para explicar. entretanto existe um restrição ai, nos não queremos que o programador que for mexer sempre tenha que perguntar sobre o conceito X e Y para o cliente, então essa solução só deve ser usada quando você realmente não tiver uma solução de nome
13. Evite utilização de redundâncias na hora de nomear classes por exemplo vamos tomar como exemplo a ultima versão do código de **rename_columns,** como base. Então se ao invés de eu nomear as variáveis de **new_name e old_name**, eu as chame-se de new_name_column e old_name_column, a primeiro momento podem parecer um nome bom, mas se você olhar bem a minha função de rename_columns, o que nos podemos identificar pelo nome que o objetivo e renomear colunas, e se você olhar de novo para os nomes new_name_column e old_name_column, o “column” no nome e redundante, pois já é parte do assunto da função. Isso é um coisa que costuma muito acontecer, então é necessário um certo cuidado

Interface e implementações:

Uma das coisas que e abortada dentro do livro do Uncle Bob, é sobre uma conversão é a utilização do “I” na frente do nome das interfaces, deste de que eu comecei a trabalhar eventualmente com orientação a objetos, do que eu diria a forma certa de utilização, interfaces e implementações, estão muito presente até quando se utiliza de algum design pattern. um exemplo seria uma Interface produto com o nome de “IProduct” e a classe que a implementa “Product”, para o Uncle Bob e mais simples só nome essa classe de interface como “ShapeProduct” do que eventualmente utilizar esse “I”.

Nomes de Classe e Métodos

Classes

O correto seria a utilização de substantivos, que são nomes utilizados para classificar e nomear algo abstrato ou concreto, assim como Conta, Produto, Web, Endereço, se você esta acostumado com modelagem isso fazer fácil pois nesse caso as classes tomam o papel do entidade. Um outro ponto abordado e evitar nomes que são verbos como Gerente, Processador, Dados ou Info

Métodos

Para os métodos nos devemos ter sempre acompanhado do nome um verbo, que evidencie a ação do método, como set_name, post_message, add_account. (em outras linguagens seria como setNamem postMessage, addAccount

uma das coisas que ele comenta que se construtores tiverem carregados, utiliza as funções de factory com métodos abstratos para abstrair a complexidade, um ponto que ele evidencia e que esses métodos abstratos devem ser claro no

que eles realizam.

```
#  isso é melhor que
money = coin.convertDolrToRealBrl(12.45)

# que isso
money = coin(12.45)
```

Interações e Mapas Mentais

Uma coisa que eventualmente é abortado e a necessidade de o leitor do código ter que traduzir os valores mentalmente, e isso não e bom, uma dos fatores abortados dentro deste assunto e a utilização de variáveis com um carácter, onde a pessoa tende a criar uma mapa mental para o que determinada variável faz ou tem como objetivo.

Um dos pontos onde o Uncle Bob, aceita esse utilização e dentro lações de interação assim como o for, ou atuando como um contador da interação, por se tratar de uma tradição e se o escopo e pequeno e eventualmente não houver possam entrar em conflito com ele. Eventualmente discordo um pouco dessa pratica, pois acredito que seja, melhor utilizar esses casos dentro do inteirador onde você terá apenas um carácter com ultimo caso, quando estritamente não houver nome melhor para o mesmo.