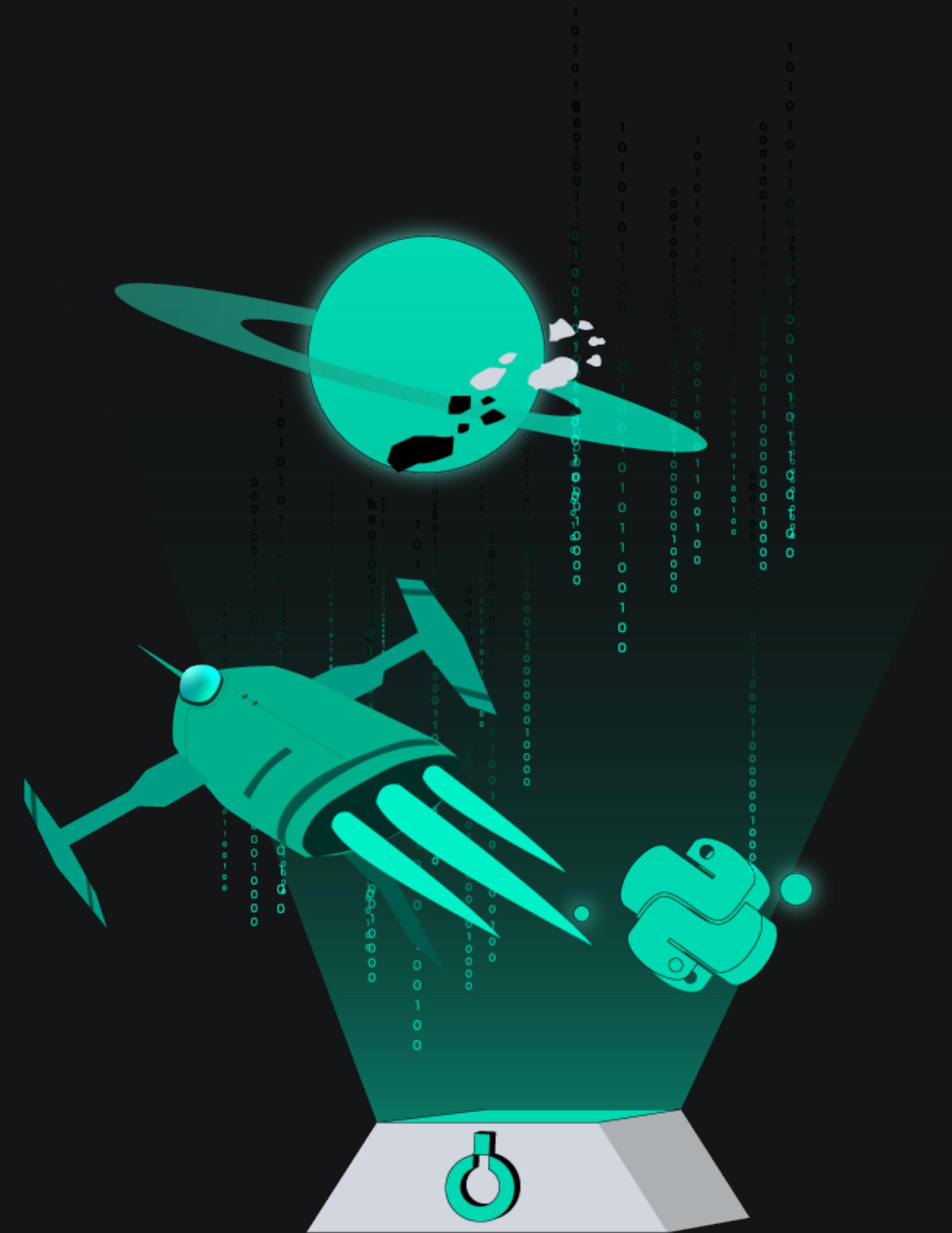


# código.py

## GALÁXIA EXTRA

**Código Limpo**



## Galáxia Extra

**Introdução aos pandas.**

### Mundo 1

Escolha bons nomes e evite comentários

Resumo

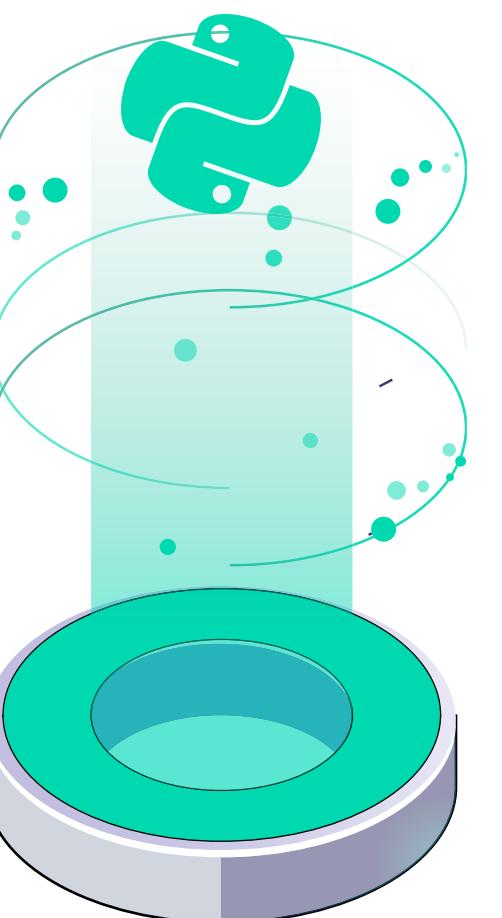
### Mundo 2

Funções e Métodos

Classes

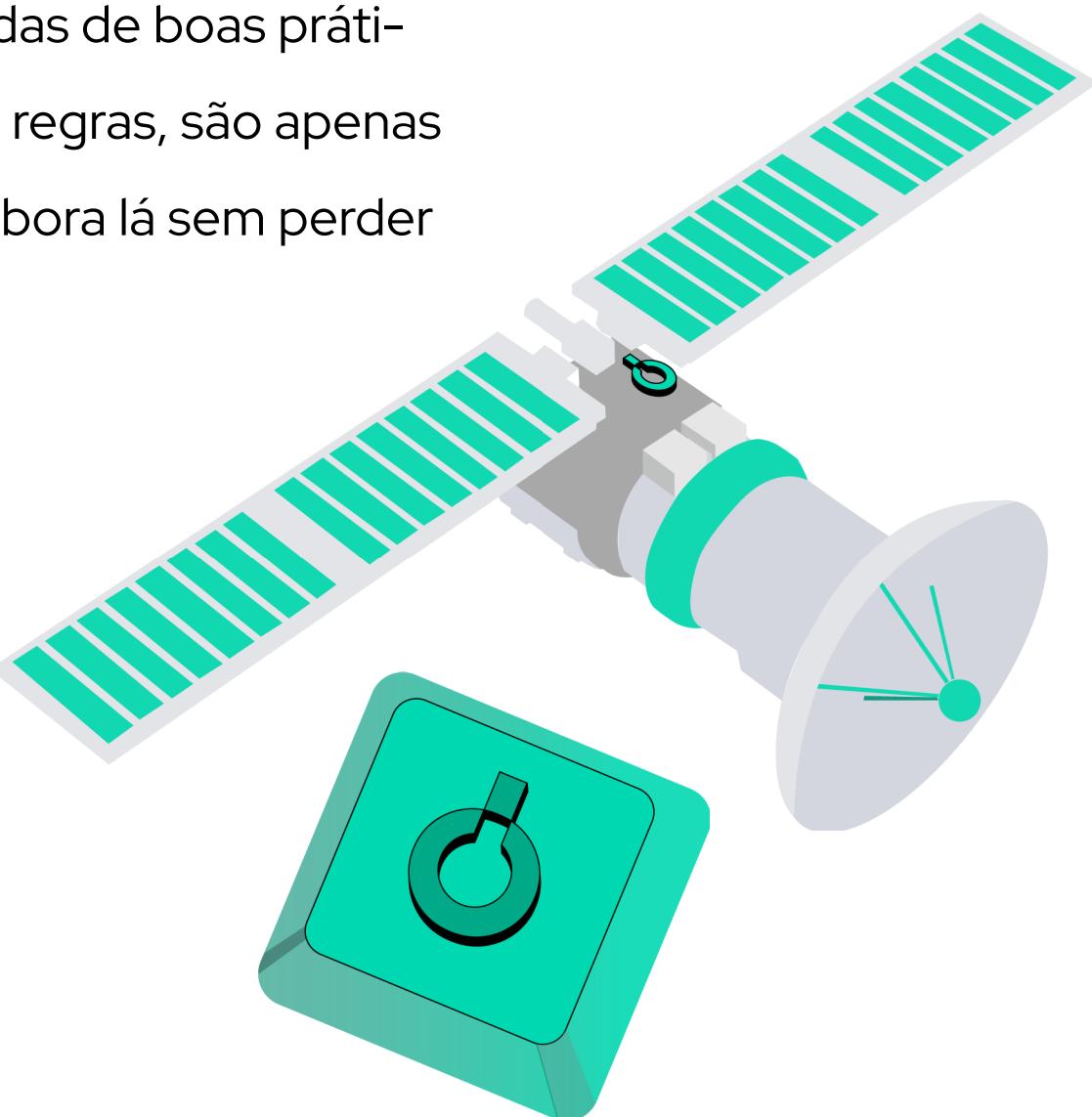
### Mundo 3

Assuma que a 1º versão do seu código é um lixo



## Introdução

Olá, seja bem vindo à Galáxia Extra sobre Código Limpo! Neste módulo iremos abordar a forma de você escrever seus códigos da forma mais profissional e otimizada possível. São chamadas de boas práticas de programação, como o nome já diz, não são regras, são apenas formas de tornar o seu código mais legível. Então bora lá sem perder tempo!



## Mundo 1

### 1.1. Escolha bons nomes e evite comentários

Escolha bons nomes para as suas [variáveis](#), [funções](#), [objetos](#), [class](#), etc.

Quanto menos comentários você colocar, melhor. A ideia é que seu programa seja facilmente entendido apenas com estes nomes.

É importante que você escolha nomes auto explicativos, existe um ditado que diz “se você precisar comentar algo, é porque você falhou como programador”, utilize esse ditado como forma de inspiração para deixar seu código bonito, comentários poluem.

A primeira impressão que temos é que quanto mais comentários melhor. Porém isso está errado, pois comentários significam que você está explicando algo, se você precisa de explicação para entender seu código provavelmente o seu código não é auto explicativo, logo ele pode melhorar.

Abaixo daremos exemplo de um código que foi otimizado, e comentaremos algumas coisas.

### Código Sujo:

```
#calculadora juros compostos

#passe os argumentos valor inicial, taxa em número inteiro,
#tempo de investimento no mesmo periodo da taxa e aporte.

def funcao_juro(valor, taxa, tempo, aporte = 0):
    #lista de dataframes

    lista = []
    contador = 0

    lista.append(pd.DataFrame(data={'Acumulado': valor}, index= [0]))

    #criando um while que acumula os juros até o final do periodo

    while contador < tempo:

        if contador == 0:
            #calculando o valor ao final do periodo

            valor2 = valor * (1 + taxa/100) + aporte

            contador = contador + 1

        lista.append(pd.DataFrame(data={'Acumulado': valor2}, index= [contador]))
```

```
else:

    valor2 = valor2 * (1 + taxa/100) + aporte

    #arredondando valor do periodo

    valor2 = round(valor2, 0)

    contador = contador + 1

    lista.append(pd.DataFrame(data={'Acumulado': valor2}, index= [contador]))

#contador = contador - 1
#lista.append(100)
#lista = ["R$" + str(numero) for numero in lista]

return pd.concat(lista)
```

## Código Limpo:

```
#Não esqueça da taxa no mesmo período do tempo!

def calculadora_de_montante_a_cada_periodo_juros_compostos(valor_inicial, taxa,
                                                               tempo_total_do_investimento, aporte = 0):

    montante_acumulado_por_periodo = []
    tempo_investindo = 0

    montante_acumulado_por_periodo.append(pd.DataFrame(data={'Acumulado': valor_inicial}, index= [0]))

    while tempo_investindo < tempo_total_do_investimento:

        if tempo_investindo == 0:

            valor_final_do_periodo = valor_inicial * (1 + taxa/100) + aporte

            tempo_investindo = tempo_investindo + 1

            montante_acumulado_por_periodo.append(pd.DataFrame(data={'Acumulado': valor_final_do_periodo},
                                                               index= [tempo_investindo]))
```

```
        else:

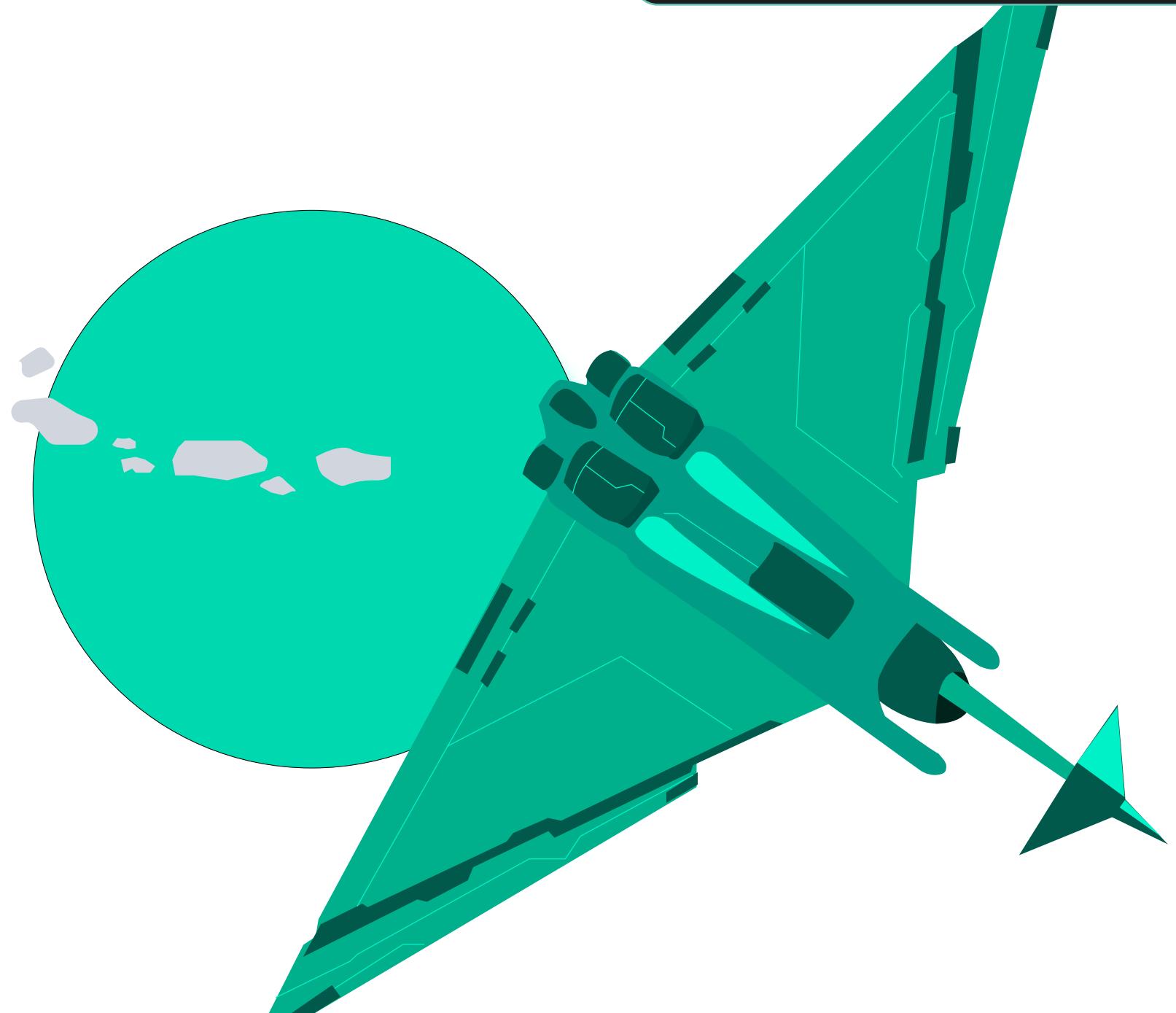
            valor_final_do_periodo = valor_final_do_periodo * (1 + taxa/100) + aporte

            valor_final_do_periodo = round(valor_final_do_periodo, 0)

            tempo_investindo = tempo_investindo + 1

            montante_acumulado_por_periodo.append(pd.DataFrame(
                data={'Acumulado': valor_final_do_periodo},
                index=[tempo_investindo]))

    return pd.concat(montante_acumulado_por_periodo)
```



Perceba como o código limpo é muito mais explicativo, não tenha medo de escolher nomes grandes. Quando bem usados, deixam bem claro o que cada coisa faz. Um nome grande e bom é MUITO melhor que um nome pequeno e ruim. Mas claro que é muito melhor você conseguir criar um nome pequeno e bom.

O nome da função já deixa bem claro do que se trata a função, bem diferente do “`funcao_juro`” que poderia ser, literalmente, qualquer coisa.

Ao invés de nomes ambíguos como “`valor1`” ou “`contador`” escolha nomes claros sobre o que se tratam suas variáveis, nomes como “`tempo_investindo`” e “`valor_final_do_periodo`”.

## 1.2. Resumo

1. Escreva nomes que se auto explicam
2. Não polua seu código (principalmente utilizando comentários)
3. Não explique o que você está fazendo com comentários, isso deveria estar claro apenas com seu código.
4. Não use nomes aleatórios, use nomes que façam sentido e procuráveis, que tenham palavras em comum com sua funcionalidade



## Mundo 2

### 2.1. Funções e Métodos

Por mais que seja um pouco subjetivo, coloque na cabeça que uma função deve fazer apenas uma coisa. Isso faz ela ficar mais otimizada e na hora da manutenção fica bem claro o que a função está fazendo.

A função não deve ser muito grande, e nem muito curta, ela deve ter um tamanho ideal e esse tamanho você que deve arbitrar.

Acontece que muitas vezes, por fazer uma função grande demais, e que faça coisas completamente diferentes, pode acontecer de uma parte que não tem nada a ver com a outra falhar. Isso geraria uma falha em toda a função, sendo que apenas a parte final está parando, ou seja, tente manter dentro da função um único trabalho.

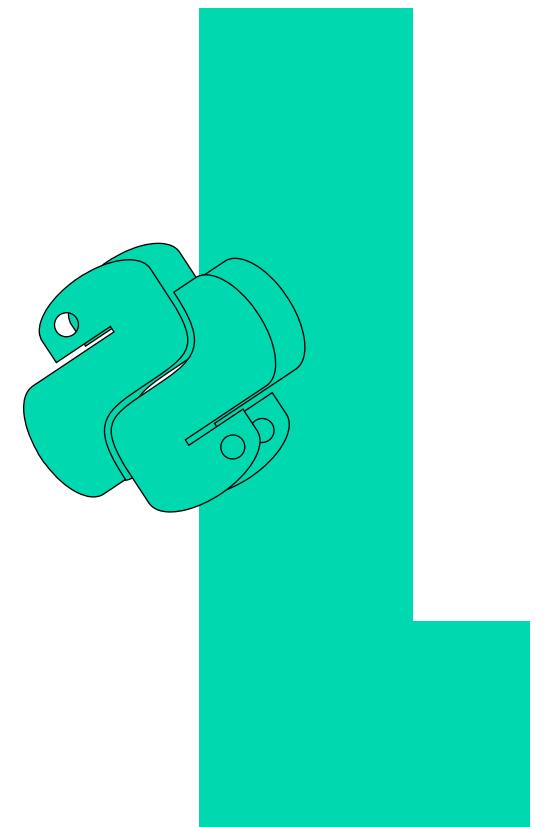
Ela também não deve ter coisas ocultas dentro dela.

### 2.2. Classes

Quando criar classes que interagem entre si, é ideal que as classes estejam próximas umas às outras, isso porque a visualização e a manutenção fica muito melhor.

As classes devem ser pequenas e devem representar apenas um único objeto. Pense numa estrutura hierárquica onde existe uma classe mãe que chama e ativa outras classes.

As classes seguem os mesmos princípios dos métodos e funções.



## Mundo 3

### 3.1. Assuma que a 1º versão do seu código é um lixo

Quando você está começando a programar, a primeira coisa e a mais importante é fazer seu código funcionar. É muito difícil para um iniciante, pensar em uma lógica de programação, pensar em otimização e um código limpo enquanto programa. Quando você finaliza a aplicação e volta para revisar, você tem uma visão maior do todo e consegue atacar exatamente onde acha que pode melhorar.

Tenha a cultura de refatorar, você perceberá que ao passar do tempo seu conhecimento vai crescendo muito rápido. Então logo você verá formas simples e fáceis de fazer a mesma coisa que você fez.

Se você escrever um código ruim você pode ter certeza que com o passar do tempo sua principal função será consertar bugs dos seus programas.

Use e abuse dos tratamentos de erros e deixe sempre o seu código o mais otimizado possível. Quanto mais otimizado melhor.