



# SKYRATS - GRUPO DE DESENVOLVIMENTO DE DRONES AUTÔNOMOS DA POLI-USP

SUBSISTEMA DE SOFTWARE

---

## Programação e Python

---

### **Autores**

Bárbara Bueno  
Mariana Watanabe

*Introdução à lógica de  
programação e à  
linguagem python*

2023

## Conteúdo

<b>1</b>	<b>Introdução a Python</b>	<b>2</b>
1.1	Sobre a linguagem . . . . .	2
<b>2</b>	<b>Comandos Básicos</b>	<b>3</b>
2.1	Data Types . . . . .	3
2.1.1	Integers . . . . .	3
2.1.2	Float . . . . .	3
2.1.3	String . . . . .	3
2.1.4	Boolean . . . . .	4
2.2	Data Structures . . . . .	5
2.2.1	Listas . . . . .	5
2.2.2	Tuplas . . . . .	7
2.2.3	Sets . . . . .	7
2.2.4	Dicionários . . . . .	8
2.3	Variables . . . . .	8
2.4	Input e Output . . . . .	9
2.5	Comentários . . . . .	10
2.6	Operators . . . . .	10
2.7	Conditionals . . . . .	11
2.8	Loops . . . . .	12
2.8.1	For Loop . . . . .	12
2.8.2	While Loop . . . . .	13
2.9	Interromper o looping . . . . .	13
2.10	Funções . . . . .	14
2.10.1	Built-in Functions . . . . .	14
2.10.2	Criando uma função . . . . .	15
2.10.3	Return . . . . .	16
2.10.4	Pass . . . . .	17
2.10.5	Recursão . . . . .	17
<b>3</b>	<b>Módulos e Bibliotecas</b>	<b>17</b>
3.1	Principais Bibliotecas . . . . .	19
<b>4</b>	<b>EP</b>	<b>20</b>

# 1 Introdução a Python

## 1.1 Sobre a linguagem

Idealizada e desenvolvida por Guido Van Rossum, matemático holandês, no início dos anos 90, o Python foi criado com o objetivo de otimizar a leitura de códigos e estimular a produtividade de quem os cria, seja este um programador ou qualquer outro profissional. [Adicionar referencia](#)

Assim, surgiu o Python, que é uma linguagem de programação de alto nível — ou High Level Language —, dinâmica, interpretada, modular, multiplataforma e orientada a objetos. Isso tudo significa :

- **Alto nível:** É uma linguagem que fornece uma abstração de nível superior para o programador, ou seja, uma sintaxe simples. Isso significa que o código escrito em Python é mais legível e fácil de entender do que em linguagens de baixo nível, como Assembly ou C, que são mais próximas da linguagem de máquina.
- **Dinâmica:** O tipo das variáveis é determinado em tempo de execução. Isso significa que o programador não precisa declarar o tipo das variáveis antes de usá-las.
- **Interpretada:** O código fonte é lido e executado linha por linha por um interpretador em tempo real. Isso torna a execução do código mais lenta do que em linguagens compiladas, mas fornece informações detalhadas de erros e problemas no código.
- **Multiplataforma:** Python pode ser executado em diferentes sistemas operacionais, como Windows, macOS e Linux.
- **Modular:** A modularidade é uma técnica de programação que permite dividir um programa em módulos menores e independentes. Cada módulo é um arquivo contendo código Python que pode ser importado em outros arquivos. Isso permite que os programadores escrevam código reutilizável, que pode ser compartilhado entre diferentes projetos.
- **Orientada a objetos:** Permite que o código seja mais modular, a partir da criação e manipulação de classes, métodos e objetos. A orientação a objetos é uma técnica bastante usada, que será explorada em um workshop futuro.

## 2 Comandos Básicos

### 2.1 Data Types

Você pode descobrir o tipo de variável colocando `type()`, como em:

---

```
1 x = 5
2 print(type(x))
3 # <class 'int'>
4
```

---

#### 2.1.1 Integers

O tipo `int` representa a classe dos números inteiros em Python. Números inteiros são todos aqueles que não contém uma parte decimal, ou seja, são uma unidade, por exemplo: 1, 10, -42.

Podemos definir um número inteiro (`int`) em Python simplesmente associando um número a uma variável

#### 2.1.2 Float

O tipo `float` representa a classe dos números reais em Python. Números reais são todos aqueles que contém parte decimal, por exemplo: 1.0, 50.8, -42.3, 0.

**Arredondamento:** No caso do tipo original da variável ser `float` e utilizarmos a função `int` acontecerá um truncamento, ou seja, a parte decimal do número será ignorada, mesmo se esse número for 1,9999, por exemplo.

---

```
1 numero = 1.9999
2
3 print(int(numero))
4 # 1
```

---

#### 2.1.3 String

Uma string no Python é uma sequência de caracteres, a qual pode ser formada com um par de aspas simples ou dupla.

---

```
1 nome_1 = "Mariana Watanabe"
2 nome_2 = 'Barbara Bueno'
```

---

Por as strings serem sequências de caracteres, podemos acessar um caractere em uma dada posição utilizando um índice. No exemplo a seguir, caso se queira obter o caractere na primeira posição da string nome, basta acessar o índice 0 da variável. No próximo tópico desse artigo, esse conceito de lista e índice ficará mais claro.

---

```
1 nome = 'Mariana'
2 print(nome[0]) # M
```

---

**Concatenação de strings:** Há casos em que é necessário juntar informações textuais. Para isso, utilizamos a concatenação, que é a junção do conteúdo de strings. Vamos fazer um exemplo no qual podemos ver como isso ocorre na prática:

---

```
1 nome = 'Cláudio'
2 sobrenome = 'Possani'
3
4 nome_completo = nome + ' ' + sobrenome
5
6 print(nome_completo) # Claudio Possani
```

---

#### 2.1.4 Boolean

As variáveis também podem ser do tipo boolean, assumindo os valores True ou False.

---

```
1 bateria_carregada = True
2 drone_armed = False
```

---

**Operador and:** Dados dois valores booleanos A e B, o operador lógico "and" resulta em True apenas quando A e B foram ambos True, e retorna False caso contrário. Exemplo, caso A for false, e B true, então um "print(A and B)" resultaria em false.

**Operador or:** Dados dois valores booleanos A e B, o operador lógico "or" resulta em False apenas quando A e B foram ambos False, e retorna True caso contrário.

**Operador not:** O operador lógico not muda o valor de seu argumento, ou seja, not True é False, e not False é True.

## 2.2 Data Structures

As Data Structures são um tipo mais avançado de armazenar os dados, podendo agrupá-los e manipulá-los através de diversos métodos. As principais são : listas, dicionários , tuplas e sets. [Adicionar referência](#)

### 2.2.1 Listas

As listas servem quando se quer criar e manipular um conjunto, cujos elementos podem ser tanto do mesmo tipo de variável quanto de variáveis diferentes. Os elementos têm uma ordem definida e podem ser acessados a partir de seu index/índice, que se inicia em 0. As listas apresentam diversos métodos de manipulação que serão apresentados no exemplo abaixo.

---

```
1
2  # Criando uma lista
3  frutas = ["banana", "uva", "laranja"]
4
5  # Adicionando um elemento no final da lista
6  frutas.append("limao")
7
8  # Acessando um elemento da lista
9  # A contagem inicia em 0
10 primeiro_elemento = frutas[0]
11
12 # Pegando o indice de um elemento
13 index = frutas.index("uva")
14
15 # Conta as ocorrências de um elemento
16 no_bananas = frutas.count("banana")
17
18 # Adicionando um elemento em uma posição específica da lista
19 frutas.insert(2, "cereja")
20
21 # Removendo um elemento da lista
22 frutas.remove("uva")
23
24 # Remove um elemento de uma posição específica
25 # Se não passar argumento remove o último elemento
26 frutas.pop(1)
27
```

```
28 # Remove todos os elementos da lista em um certo intervalo
29 del frutas[2:4]
30
31 # Remove todos os elementos da lista
32 frutas.clear()
33
34 # Inverte a ordem dos elementos da lista
35 frutas.reverse()
36
37 # Ordena os elementos da lista (numericamente ou alfabeticamente)
38 frutas.sort()
39
40 # Imprimindo a lista
41 print(frutas)
```

---

A partir desses métodos, as listas podem ser usadas como outras data structures não necessariamente nativos do python, como stacks, queues, matrizes.

---

```
1
2 # Exemplo de matriz utilizando listas
3 matriz = [ [1,2,3],
4             [4,5,6],
5             [7,8,9]]
6
7 print(matriz[0][2])
8 # 3
9 # Pega o elemento da primeira linha (0) e da terceira coluna (2)
```

---

Por fim, listas também podem ser criadas sem a necessidade de listar todos os seus valores, utilizando um loop no lugar.

---

```
1
2 # Listando os números pares de 1 a 10
3 pares = [num * 2 for num in range(1,6)]
4 # pares = [2,4,6,8,10]
5
6 # Lista as combinações dos valores de x e y dados
7 # e impõe a condição de eles serem diferentes
8 list = [(x, y) for x in [1,2,3] for y in [3,1,4] if x != y]
```

---

```
9  #[(1, 3), (1, 4), (2, 3), (2, 1), (2, 4), (3, 1), (3, 4)]
```

---

### 2.2.2 Tuplas

Tuplas também servem para armazenar um conjunto de dados. Elas se diferem das listas em sua declaração (com parenteses em vez de colchetes) e no fato de serem imutáveis, ou seja, você não pode alterar nada de uma tupla após ela ser criada, o que inclui adicionar, remover e modificar elementos.

---

```
1
2  # Criando uma tupla vazia
3  tupla_vazia = ()
4
5  # Criando uma tupla com apenas um elemento : colocar a virgula no final
6  tupla_um_elemento = "solitario",
7
8  # Duas maneiras de se criar uma tupla
9  t1 = 1, 2, 3
10 t2 = (1, 2, 3)
11
12 # Acessando os elementos de uma tupla
13 primeiro = t1[0]
14
15 pri, seg, ter = t1  # pri = 1, seg = 2, ter = 3
16
17
```

---

### 2.2.3 Sets

Sets são conjuntos não ordenados, ou seja, cujos elementos não podem ser acessados por índices, e sem elementos repetidos. Assim, são úteis quando se quer remover as repetições de uma lista.

---

```
1
2  # Criando o set com chaves
3  # Observe que ele elimina os objetos repetidos
4  cesta = {"banana", "laranja", "cereja", "banana", "laranja"}
5  print(cesta)
6  # {'cereja', 'banana', 'laranja'}
```



```
7
8 # Criando o set com o método set()
9 cesta2 = set("banana", "laranja", "cereja", "banana", "laranja")
10 print(cesta2)
11 # {'cereja', 'banana', 'laranja'}
```

---

### 2.2.4 Dicionários

Dicionários são um jeito de armazenar pares chave/valor, contanto que cada chave seja única, não se repetindo no mesmo dicionário. Assim, os valores são acessados a partir de sua chave específica.

---

```
1
2 # Criando um dicionário que armazena o par nome/telefone
3 tel = {"babs" : 1234, "mari" : 5678, "lena" : 9012}
4 print(tel["mari"]) #5678
5
6 # Criando dicionários com loops
7 dic = {x: x**2 for x in (2, 4, 6)}
8 print(dic) # {2: 4, 4: 16, 6: 36}
9
10 # Iterando entre as chaves e valores
11 for k, v in tel.items():
12     print(k, v)
13 # babs 1234
14 # mari 5678
15 # lena 9012
```

---

## 2.3 Variables

As variáveis em python não precisam ser declaradas quanto ao seu tipo, pois a linguagem já reconhece qual seu tipo. Basta apenas atribuir-lhes um valor.

---

```
1 x = 5
2 y = "John"
3 print(x)
4 print(y)
```

---

No entanto, você pode também forçar para que alguma variável seja do tipo que você deseja.

---

```
1 x = str(3)      # x will be '3'
2 y = int(3)      # y will be 3
3 z = float(3)    # z will be 3.0
```

---

## 2.4 Input e Output

Como observado nos exemplos acima, o jeito do python apresentar o output é pela função **print()**. Tudo que for passado dentro dela será printado no terminal na forma de string.

---

```
1 nome = Bárbara
2 print("Skyrats!!")
3 print("Olá" + nome)
4 print("Idade: " + 19)
5
```

---

Também é possível utilizar variáveis dentro da string por meio da formatação. Ela é utilizada colocando-se um **f** antes das strings e colocando as variáveis dentro de colchetes na posição em que deve ser substituída.

---

```
1 nome = "Bárbara"
2 idade = 19
3 print(f"Meu nome é {nome} e tenho {idade} anos")
4 # Meu nome é Bárbara e tenho 19 anos
```

---

Para pegar um input do usuário, utilizamos a função **input()**. O input é lido no formato de string (então mesmo que o usuário digite 2, não será um int e sim uma string). Além disso, como argumento ela recebe uma string que será apresentada para o usuário antes do input ser habilitado.

---

```
1 nome = input("Qual é o seu nome? ")
```

---

## 2.5 Comentários

Comentários são partes do código que não são executadas. Eles servem para adicionar informações que ajudam no entendimento do código, como explicar o que significa uma função ou variável, entre outras coisas.

Em python, comentários podem ser feitos de duas maneiras :

---

```
1  # Esse é um comentário de uma linha
2  # Podem ser usados quantos quiser
3
4
5  '''
6      Esse é um comentários
7      de várias linhas
8      Skyrats :)
9  '''
10
```

---

Exemplo de uso prático :

---

```
1
2  # Recebendo o nome do usuário
3  nome = input("Qual o seu nome? ")
4
5  # Usando o nome para cumprimentar o usuário
6  print("Olá " + nome)
7
```

---

## 2.6 Operators

**Operadores aritméticos:** Tais operadores são usados para os cálculos gerais do código e podem ser representados na tabela abaixo.

---

```
1  numero_1 = 5
2  numero_2 = 2
3
4  soma = numero_1 + numero_2
5  subtracao = numero_1 - numero_2
6  multiplicacao = numero_1 * numero_2
```

```
7 divisao = numero_1 / numero_2
8 divisao_inteira = numero_1 // numero_2
9 modulo = numero_1 % numero_2
10 exponenciacao = numero_1 ** numero_2
11
12 print(soma) # 7
13 print(subtracao) # 3
14 print(multiplicacao) # 10
15 print(divisao) # 2.5
16 print(divisao_inteira) # 2
17 print(modulo) # 1
18 print(exponenciacao) # 25
```

OPERAÇÃO	OPERADOR
SOMA	+
SUBTRAÇÃO	-
DIVISÃO	/
MULTIPLICAÇÃO	*
MODULO	%
DIVISAO INTEIRA	//
EXPONENCIAÇÃO	**

## 2.7 Conditionals

Utilizamos o comando `if` para verificar uma expressão e executar um bloco de código caso a condição definida seja verdadeira. É importante dizer que a instrução `if` pode ser utilizada sozinha, ou seja, apenas para executar algo se a condição for verdadeira. Observe que devemos utilizar o caractere dois pontos “:” ao final da instrução. Ainda, vale ressaltar que tudo dentro do comando “`if`” deve estar na indentação.

```
1 if (expressão_for_verdadeira):
2     executar_bloco_de_codigo()
```

Caso tal condição do `if` não seja executada, pode-se incluir uma condição “`else`” (senão)

que executa a ação caso o "if" não seja feito.

---

```
1 if (expressão_for_verdadeira):
2     executar_primeiro_bloco_de_codigo()
3 else:
4     executar_segundo_bloco_de_codigo()
```

---

Ainda, caso queira acrescentar mais de uma condição além do "else", pode-se usar um, ou mais "elif".

---

```
1 if (expressão_for_verdadeira):
2     executar_primeiro_bloco_de_codigo()
3 elif (segunda_expressão_for_verdadeira):
4     executar_segundo_bloco_de_codigo()
5 else:
6     executar_terceiro_bloco_de_codigo()
```

---

## 2.8 Loops

### 2.8.1 For Loop

Na linguagem de programação Python, os laços de repetição "for" também são chamados de "loops definidos" porque executam a instrução um certo número de vezes. For loops são úteis quando você deseja executar o mesmo código para cada item em uma determinada sequência

---

```
1 numbers = [1, 2, 3, 4, 5]
2 for number in numbers:
3     print(number)
4
```

---

Observa-se no caso acima que a lista de numbers tem um final, portanto cabe o uso de "for".

Também pode-se definir um número de vezes para o código se repetir a partir do uso do range, que começa em zero (caso o início não seja especificado) e vai até o número anterior ao último determinado.

---

```
1 for i in range(5):
```

---

```
2     print(i)
3     # Isso printará os números de 0 a 4
4
5     for i in range(1,6):
6         print(i)
7     # Isso printará os números de 1 a 5
8
```

---

### 2.8.2 While Loop

No caso do while loop, um certo comando é feito continuamente até que a condição imposta se torne falsa. É essencial, portanto, que nesse ciclo do while haja uma condição que torne em algum momento a condição imposta como falsa e interrompa o looping, caso contrário, ele continuará para sempre.

---

```
1 i = 1
2 while i < 6:
3     print(i)
4     i += 1
5
```

---

Observa-se no caso acima que o incremento do i fará com que alguma hora saia do looping.

## 2.9 Interromper o looping

Dentro do looping while pode-se interromper o laço por meio da condição "if" e também o uso de "break".

---

```
1 for <item> in <conjunto_de_itens>:
2     <bloco_de_codigo>
3     if <condicao_verdadeira>:
4         <outras_instrucoes>
5         break
```

---

De forma mais ilustrativa, podemos ter o código:

---

```
1 numbers=[1,2,3,4,5]
2 for number in numbers:
```

```
3     print(number)
4     if number==3:
5         print('chegou no numero 3')
6         break
```

OBS: Percebe-se que foi usado "==" para definir que algo é igual a algo, pois um único

## 2.10 Funções

Em Python, uma função é um bloco de código que executa uma tarefa específica e pode ser reutilizado em diferentes partes do programa. Assim, quando se quer executar um mesmo tipo de ação diversas vezes, criamos uma função para não precisar escrever a lógica por trás dessa tarefa todas as vezes.

### 2.10.1 Built-in Functions

O Python apresenta algumas funções já integradas, ou seja, que você não precisa declará-las e escrever sua lógica para usá-las. Algumas delas são: [1]

- **print()** - Essa função é usada para imprimir texto ou variáveis no console.
- **len()** - Essa função retorna o comprimento de um objeto, como uma string ou uma lista.
- **input()** - Essa função é usada para obter entrada do usuário pelo console.
- **range()** - Essa função retorna uma sequência de números, que pode ser usada em loops ou para criar listas.
- **type()** - Essa função retorna o tipo de dados de um objeto.
- **int()** - Essa função converte uma string ou float em um inteiro.
- **float()** - Essa função converte uma string ou inteiro em um float.
- **str()** - Essa função converte um objeto em uma string.
- **list()** - Essa função converte um objeto em uma lista.
- **max()** - Essa função retorna o maior item em um iterável ou o maior de dois ou mais argumentos.
- **min()** - Essa função retorna o menor item em um iterável ou o menor de dois ou mais argumentos.
- **sum()** - Essa função retorna a soma de todos os itens em um iterável.

### 2.10.2 Criando uma função

Toda função, ao ser criada, é iniciada pela palavra chave **def**, seguida do nome da função e do parênteses com os argumentos. Embaixo e seguindo a indentação, você escreve o comportamento da função. Para chamar a função, basta passar seu nome com os argumentos, assim como se faz com a built-in functions. [Adicionar referencia](#)

---

```
1
2 # Função sem argumentos
3 def say_hello():
4     print("Hello!!")
5
6 say_hello()
7 # Hello!!
8
9 # Função com argumento
10 def say_hello(name):
11     print("Hello " + name + "!!")
12
13 say_hello("Bárbara")
14 # Hello Bárbara!!
15
```

---

As funções podem ter vários argumentos e até mesmo argumentos padrões, para caso não sejam passados ao ser chamadas.

---

```
1
2 # Função com vários argumentos
3 def say_hello(name1, name2, food):
4     print("Hello " + name1)
5     print("I am " + name2)
6     print("Do you want to eat " + food + "?")
7
8 say_hello("Mari", "Bárbara", "ice cream")
9 # Hello Mari
10 # I am Bárbara
11 # Do you want to eat ice cream?
12
13 # Função com argumento padrão
14 def say_hello(name = "you"):
```



```
15     print("Hello " + name + "!!")
16
17 say_hello()  # name não foi passado, então assume o valor padrão 'you'
18 # Hello you!!
19
```

---

### 2.10.3 Return

As funções podem retornar valores a partir do uso do **return**. Assim, o resultado de uma função pode ser armazenado em uma variável ou utilizado como argumento de outra função.

---

```
1
2 # Função que retorna o quadrado de um número
3 def square(x):
4     return x**2
5
6 # Armazenando o valor retornado em uma variável
7 value1 = square(5)
8 print(value1)
9 # 25
10
11 # Usando o valor retornado como argumento de uma função
12 print(square(3))
13 # 9
14
```

---

Quando uma função chega em um **return**, ela encerra a sua ação. Assim, o **return** pode ser utilizado como um método de "stop" da função.

---

```
1
2 def square(x):
3     if( type(x) == str):
4         print("Não é possível efetuar a conta")
5         return
6     return x**2
7
8 square(2) # 4
9 square("palavra") # Não é possível efetuar a conta
```

10

---

#### 2.10.4 Pass

Um função não pode ter o seu corpo vazio. Se por algum motivo você não quer colocar um corpo para sua função (seja temporariamente ou para ela realmente não fazer nada), você deve usar o **pass**

---

```
1
2 def nada():
3     pass
4
5 nada()
```

---

#### 2.10.5 Recursão

Recursão é um algoritmo de programação que ocorre quando uma função chama ela mesma. Para isso, é necessário haver uma condição de parada, ou ela ira ficar chamando recursivamente a si mesma infinitamente.

---

```
1
2 def rec_factorial(n):
3     if n == 1 or n == 0:
4         return n
5     else:
6         return n * rec_factorial(n-1)
7
8 rec_factorial(1) # 1
9 rec_factorial(4) # 24
```

---

## 3 Módulos e Bibliotecas

Um módulo em python, é um conjunto de código relacionado, salvo em um arquivo com a extensão **.py**. Dentro dele, podem estar definidas diversas funções, classes e variáveis. Você pode criar seus próprios módulos ou importar módulos prontos do python. Nesse exemplo, vamos criar um módulo com a função `say_hello()` e salvar em um arquivo chamado **hello.py**

---

```
1
2 def say_hello(name):
3     print("Hello " + name)
4
```

---

Para chamar um módulo em seu código, basta usar o **import** com o nome do arquivo (caso ele não esteja na mesma pasta, adicionar o caminho até ele). Para usar uma função do módulo, utiliza-se o nome do módulo importado seguido de um ponto e o nome da função.

---

```
1
2 import hello
3 hello.say_hello("Bárbara") # Hello Bárbara
4
```

---

Caso você importe uma única função, não é preciso utilizar o nome do módulo

---

```
1
2 from hello import say_hello
3 # também poderia ser :
4 # import hello.say_hello()
5
6 hello.say_hello("Bárbara") # Hello Bárbara
7
```

---

Módulos já contidos no python não precisam ter o caminho especificado e podem ser chamados do mesmo modo, pelo seu nome

---

```
1 import time
2 import random
3
```

---

Além dos módulos, o python apresenta também as **bibliotecas**, que são um conjunto de módulos, geralmente com funções mais elaboradas. Muitas vezes será preciso instalar um biblioteca para usá-la. Para isso, utilizamos, no terminal, o comando **pip install** e o nome da biblioteca

---

```
1 # Digitar no terminal
```

---

```
2 pip install pygame
3 pip install
4
```

---

Para usar essas bibliotecas utilizamos também o **import**. Além disso, tanto com bibliotecas quanto com módulos, podemos escolher um nome para se referir a eles no código, utilizando o **as**

---

```
1 import numpy as np # importa a biblioteca numpy chamando-a de np
2 import cv2 as cv # importa a biblioteca cv2 chamando-a de cv
3
4 # Quando for utilizar funções da biblioteca,
5 # utilizar o nome escolhido no import
6 cv.imread("./Img.png")
```

---

### 3.1 Principais Bibliotecas

O python apresenta diversas bibliotecas úteis, mas algumas são mais conhecidas e mais utilizadas, principalmente pela Skyrats, como as mencionadas a seguir:

- **numpy**: O NumPy é uma biblioteca com funções para se trabalhar com computação numérica. É bastante útil para executar várias tarefas matemáticas como integração numérica, diferenciação, interpolação, extrapolação e muitas outras.
- **matplotlib**: A Python Matplotlib é utilizada para visualização de dados e plotagem gráfica.
- **pygame**: PyGame é um módulo usado na programação de jogos 2D. Ele fornece ferramentas simples para gerenciar ambientes gráficos complexos, com movimentos e sons. Nós usamos em nosso projeto do tello, para gerar interfaces gráficas e utilizar sons.
- **rospy**: Rospy é uma API que permite que o python acesse e manipule tópicos, serviços e parâmetros do ROS.
- **opencv**: Opencv é uma biblioteca de processamento de imagens amplamente utilizada por nossa equipe. Com ela é possível modificar imagens, aplicar filtros, utilizar detecções de formas e objetos, entre várias outras coisas. Ela é bastante documentada e apresenta diversas funções.

## 4 EP

Para colocar em prática o conhecimento aprendido, você deverá fazer um código em python de **detecção de um quadrado verde**, utilizando para isso, a biblioteca Opencv. Assim, por estratégias que você decidirá, o seu código deve identificar, na imagem da câmera (do próprio pc ou externa), um quadrado verde e desenhar uma moldura identificadora em volta dele.



Figura 1: Moldura vermelha em torno de quadrado verde identificado

**BÔNUS:** Como você faria para o código avisar toda vez que aparecesse o quadrado verde na tela, sem que ele print durante todo o tempo em que ele está na tela? Exemplo: quando o quadrado surgir na tela ele deve avisar uma única vez. Após o quadrado sumir da tela e aparecer novamente, ele deve avisar mais uma única vez.

## Referências

- [1] Python built-in functions. <https://docs.python.org/3/library/functions.html>.