

MONITÓRIAS DE PYTHON

AULA 1: INTRODUÇÃO

Julha Marcolan
Universidade de São Paulo (USP)
Instituto de Física de São Carlos (IFSC)



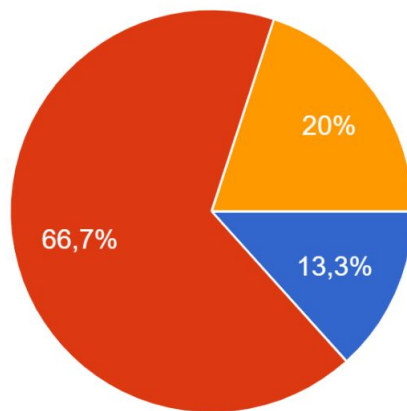
OBJETIVOS DA MONITORIA:

- Introduzir noções básicas sobre simulações.
- Introduzir noções básicas de programação em Python.
- Introduzir noções básicas de alguns pacotes científicos em Python: Numpy, SciPy, Matplotlib, etc.
- Introduzir noções básicas do uso das bibliotecas animate e PyGame.
- Auxiliar e instruir na busca de documentação.
- Auxiliar na documentação do código.
- Auxiliar na escolha dos projetos.
- Tirar dúvidas sobre a execução dos projetos.



Qual o seu nível de conhecimento em Python?

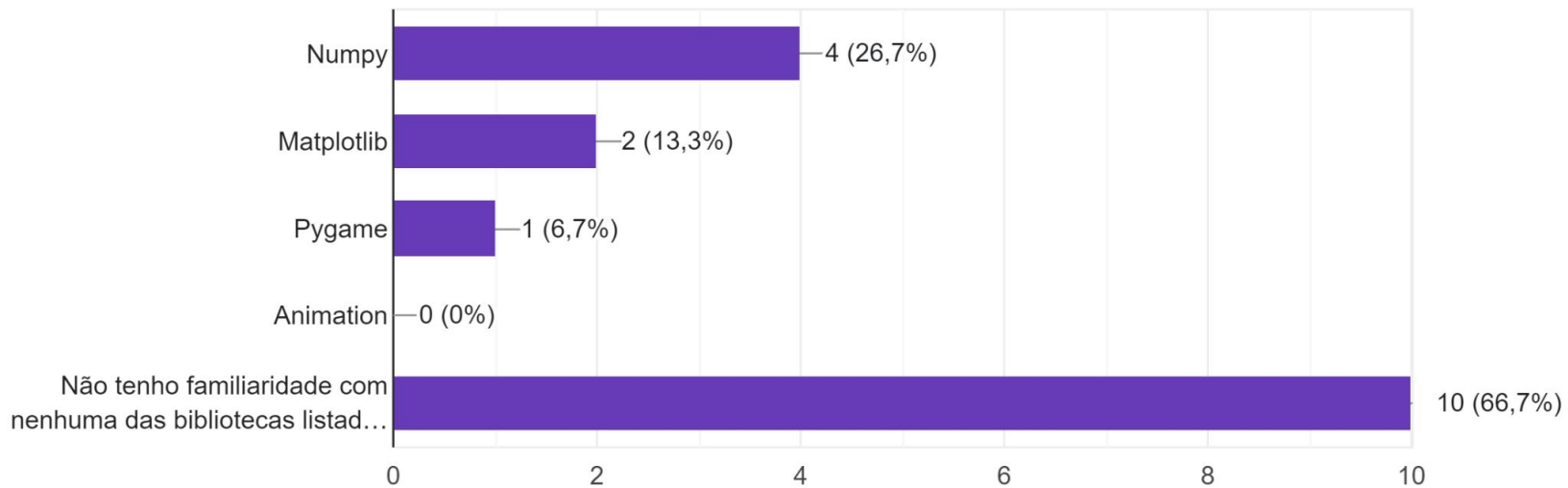
15 respostas



- Nunca usei Python
- Conhecimento básico (conheço conceitos como variáveis, loops, condicionais)
- Conhecimento intermediário (sei trabalhar com funções, listas, dicionários)
- Conhecimento avançado (sei trabalhar com classes, módulos, bibliotecas)

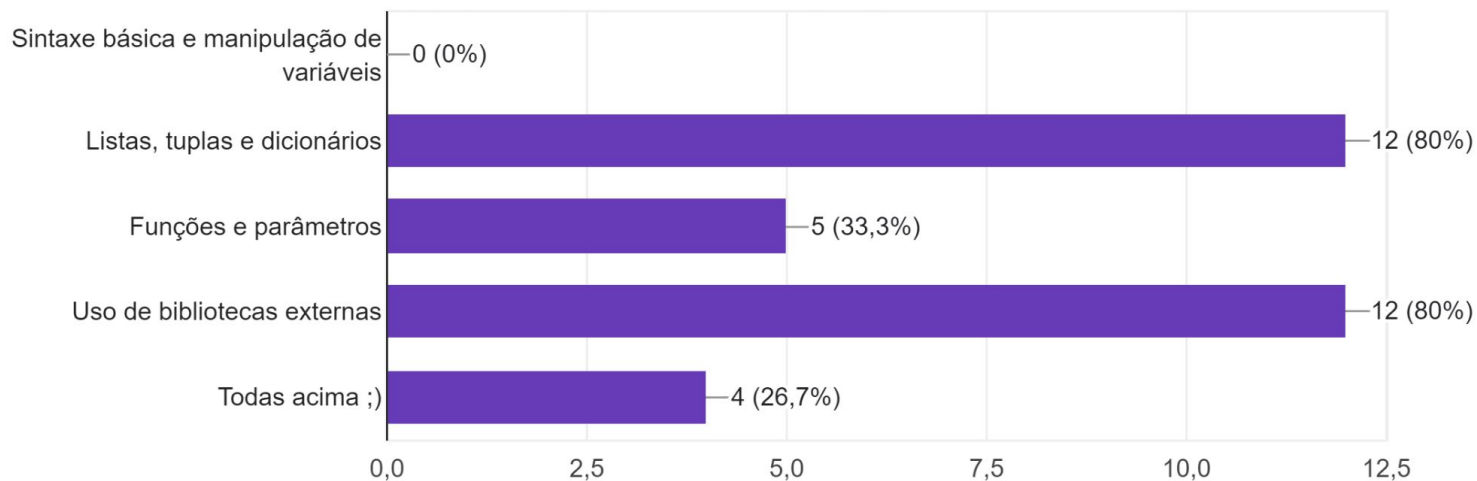
Você tem familiaridade com alguma das bibliotecas abaixo ?

15 respostas




Com relação ao Python, em quais dos seguintes tópicos você sente que precisa de mais ajuda?
(Você pode selecionar mais de uma opção)

15 respostas





CRONOGRAMA


- **AULA 1 (09/10):** Introdução: objetivos da monitoria + primeiros passos com python + Documentação.
- **AULA 2 (16/10):** Bibliotecas importantes: NumPy, SciPy, Matplotlib + Simulações (com exemplos)
- **AULA 3 (23/10):** Uso da Matplotlib.animation.
- **AULA 4 (30/10):** Uso da Matplotlib.animation.
- **AULA 5 (06/11):** Exercícios extras para fixação e aprofundamento + Apresentação de possibilidade de problemas que podem ser usados como projeto.
- **AULA 6 (13/11):** Dúvidas + Ajuda com projeto.
- **AULA 7 (20/11):** Ajuda com projeto.
- **AULA 8 (27/11):** Ajuda com projeto.


 **Fácil de Ler e Escrever:** A linguagem é mais próxima da linguagem humana do que da linguagem de máquina.


 **Linguagem versátil:** Permite ampla utilização.


 **Linguagem Interpretada:** Python é executada por um interpretador, o que significa que você pode executar seu código diretamente, sem necessidade de compilação prévia. Isso facilita a depuração e o desenvolvimento rápido.

 **Tipagem Dinâmica:** A linguagem é dinamicamente tipada, permitindo que você escreva menos código relacionado à declaração de tipos.

 **Multiplataforma:** Python pode ser executado em quase todos os sistemas operacionais, como Windows, macOS, Linux, e até mesmo em plataformas móveis, o que facilita o desenvolvimento multiplataforma.

 **Compatibilidade com Outros Sistemas:** Python pode se integrar facilmente com outras linguagens e tecnologias, como C, C++, Java, e .NET.

 **Ferramentas de Visualização:** Python também possui várias bibliotecas para visualização de dados, como Matplotlib, Seaborn e Plotly, que são amplamente utilizadas para análise e apresentação de dados.

 **Documentação Rica:** Python tem uma documentação abrangente e bem-organizada, que é muito útil tanto para iniciantes quanto para desenvolvedores experientes.

A eficiência de uma linguagem
SEMPRE depende da aplicação.



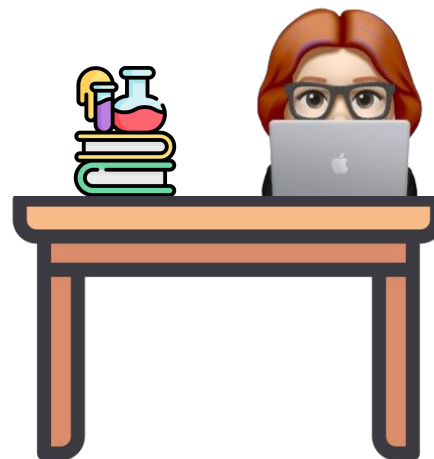


PLATAFORMA: [Google collaboratory](https://colab.research.google.com/)

Python

```
print("Hello, world!")
```

```
print("Hello  
, World!")
```





Em Python, as variáveis são criadas automaticamente quando um valor é atribuído a elas, sem a necessidade de declarar o tipo explicitamente.



Python é uma linguagem de tipagem dinâmica, o que significa que você não precisa especificar o tipo de uma variável ao defini-la, e o tipo pode mudar durante a execução do programa.

TIPOS DE VARIÁVEIS:



[EXEMPLOS 1:](#)

Inteiros (int**):** Representam números inteiros, positivos ou negativos, sem casas decimais.

Números de ponto flutuante (float**):** Representam números reais com casas decimais.

Números complexos (complex**):** Representam números complexos, na forma $a + bj$, onde a é a parte real e b é a parte imaginária.

Strings (str**):** Sequências de caracteres usadas para armazenar texto.

Booleanos (bool**):** Representam valores lógicos, ou seja, **True** (verdadeiro) ou **False** (falso).

Listas (list**):** Coleções ordenadas e mutáveis de itens. Itens podem ser de diferentes tipos.

Tuplas (tuple**):** Coleções ordenadas e imutáveis de itens.

Dicionários (dict**):** Coleções de pares chave-valor, onde cada chave está associada a um valor. As chaves são únicas e podem ser de qualquer tipo imutável.

Conjuntos (set**):** Coleções não ordenadas de itens únicos, sem elementos duplicados.

None: Representa a ausência de valor ou um valor nulo.



About Project Euler

Where should I start?

That depends on your background. There are two tables containing problems. The Recent problems table lists the ten most recently published problems, so if you are new to Project Euler then you may prefer to start with the Archives to get a feel for the different types/difficulties of our problems. The first one-hundred or so problems are generally considered to be easier than the problems which follow. In the archives table you will be able to see how many people have solved each problem; as a general rule of thumb the more people that have solved it, the easier it is. To assist further there is a difficulty rating system which may also help you decide where to start. You are able to sort the problems in the archives table on ID, Solved By, or Difficulty.



I've written my program but should it take days to get to the answer?

Absolutely not! Each problem has been designed according to a "one-minute rule", which means that although it may take several hours to design a successful algorithm with more difficult problems, an efficient implementation will allow a solution to be obtained on a modestly powered computer in less than one minute.

Em Python, há duas formas principais de escrever loops: **for** e **while**. Cada um é utilizado em diferentes situações:

1. Loop **for:** O loop **for** itera sobre uma sequência (como listas, tuplas, strings ou intervalos de números).

A sintaxe básica é:

```
Python
for elemento in sequência:
    # bloco de código
```

Exemplo:

```
Python
for i in range(5):
    print(i)
```

2. Loop **while:** O loop **while** executa enquanto uma condição for verdadeira. Ele é útil quando não se sabe antecipadamente o número de iterações.

A sintaxe básica é:

```
Python
while condição:
    # bloco de código
```

Exemplo:

```
Python
i = 0
while i < 5:
    print(i)
    i += 1
```

PROBLEMA 1:

Se listarmos todos os números naturais abaixo de 10 que são múltiplos de 3 ou 5, obtemos 3, 5, 6 e 9. A soma desses múltiplos é 23.

Encontre a soma de todos os múltiplos de 3 ou 5 abaixo de 1000.

Condicionais são usadas para executar diferentes blocos de código com base em uma condição. A estrutura mais comum é a declaração **if**, que avalia uma expressão booleana (verdadeira ou falsa) e executa o código correspondente se for verdadeira. Se a condição não for atendida, o Python pode seguir para outras condições com **elif** (else if) ou para o bloco **else**, que é executado se nenhuma das condições anteriores for verdadeira.

A sintaxe básica é:

Python

```
if condição:
    # bloco executado se a condição for verdadeira
elif outra_condição:
    # bloco executado se a outra_condição for
    verdadeira
else:
    # bloco executado se nenhuma condição anterior
    for verdadeira
```

Exemplo:

Python

```
x = 10

if x > 0:
    print("x é positivo")
elif x == 0:
    print("x é zero")
else:
    print("x é negativo")
```

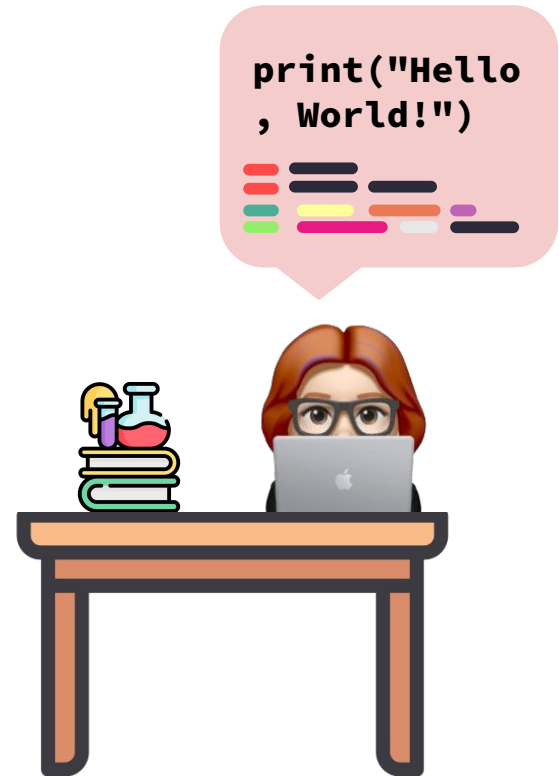
PROBLEMA 2:

Cada novo termo na sequência de Fibonacci é gerado somando os dois termos anteriores. Começando com 1 e 2, os primeiros 10 termos serão:

1, 2, 3, 5, 8, 13, 21, 34, 55, 89...

Considerando os termos na sequência de Fibonacci cujos valores não excedem quatro milhões, encontre a soma dos termos de valor par.

- [Documentação oficial do Python.](#)
- [Real Python.](#)
- [Geeks for geeks.](#)
- [Material Pyladies.](#)
- Chat GPT - com moderação e sabedoria.



DOCUMENTAÇÃO

- **Seja conciso:** Documente o essencial, sem excessos.
- **Use linguagem simples:** Torne a documentação acessível para qualquer desenvolvedor.
- **Mantenha a docstring atualizada:** Se a função mudar, atualize a documentação.
- **Consistência:** Mantenha o mesmo estilo de docstring em todo o código

Python

```
def minha_funcao(param1, param2):  
    """  
    Descrição da função.  
    """  
    # Corpo da função
```

Python

```
"""
```

Breve descrição da função.

Parâmetros:

param1 (tipo): Descrição do primeiro parâmetro.
param2 (tipo): Descrição do segundo parâmetro.

Retorno:

tipo: Descrição do que a função retorna.

Exceções (opcional):

TipoDeErro Descrição dos erros que a função
pode levantar.

```
"""
```



```
def sum_3_or_5(N):  
    """  
    Calculates the sum of all numbers less than N that are divisible by 3 or 5.  
  
    Parameters:  
        N (int): The upper limit of the range (exclusive).  
  
    Returns:  
        int: The sum of all numbers less than N that are divisible by 3 or 5.  
    """  
    sum = 0  
    for i in range(N): # Iterate from 0 to N-1 (exclusive)  
        if i % 3 == 0 or i % 5 == 0: # If the number is divisible by 3 or 5  
            sum += i # Add the number to sum  
    return sum # Return the final sum
```

```
def sum_even_fibonacci(limit):  
    """  
    Calculates the sum of even-valued terms in the Fibonacci sequence up to a  
    specified limit.  
  
    Parameters:  
  
    limit (int): The maximum value for terms in the Fibonacci sequence.  
    The function will compute the sum of even-valued terms that do not exceed  
    this value.  
  
    Returns:  
    int: The sum of all even-valued Fibonacci terms whose values are less than  
    or equal to the specified limit.  
    """  
    a, b = 1, 2  
    sum_even = 0  
    while a <= limit:  
        if a % 2 == 0:  
            sum_even += a  
        a, b = b, a + b  
    return sum_even
```



Até a próxima aula!



juliamarcolan@usp.br



https://github.com/julhamarcolan/monitorias_python.git

