

ICMC-USP

**Notas de Aula**  
**SME0332**

# **Fundamentos de Programação de Computadores**

**com Aplicações em python para Física e Bioinformática**

Roberto F. Ausas

August 29, 2024

Instituto de Ciências Matemáticas e de Computação  
Universidade de São Paulo

# Prefacio

Este documento está organizado por capítulos, cada um dos quais apresenta conceitos e ferramentas essenciais para desenvolver programas de computador na linguagem python. Os problemas e exemplos que serão desenvolvidos terão uma ênfase nas aplicações em física e bioinformática. Em cada capítulo serão apresentados problemas e atividades que o aluno precisará desenvolver, com as quais, este será avaliado ao longo do curso.

## Que tópicos este curso apresenta

Ao longo do curso iremos estudar os seguintes temas listados na sequência:

- Capítulo |01|** Um primeiro contato à Programação em python;
- Capítulo |02|** Variaveis e Caminhadas Aleatórias, Cálculos Monte Carlo;
- Capítulo |03|** Processos Iterativos e Relaxações;
- Capítulo |04|** Prprocessamento de Imagens com python;
- Capítulo |05|** Resolvendo Sistemas Dinâmicos com python;

*RFA*

# Contents

<b>Prefacio</b>	<b>ii</b>
<b>Contents</b>	<b>iii</b>
<b>1 UM PRIMEIRO CONTATO À PROGRAMAÇÃO EM python</b>	<b>1</b>
1.1 Preludio . . . . .	1
1.2 Noções iniciais de python . . . . .	2
1.2.1 Tipos de variáveis . . . . .	2
1.2.2 Estruturas condicionais . . . . .	3
1.2.3 Estruturas de repetição . . . . .	4
1.2.4 Funções . . . . .	5
1.2.5 A biblioteca numpy . . . . .	5
1.3 <a href="#">Lista 1: Rudimentos básicos de programação</a> . . . . .	6
<b>Alphabetical Index</b>	<b>11</b>

# UM PRIMEIRO CONTATO À PROGRAMAÇÃO EM python

# 1

## 1.1 Preludio

A principal ideia por trás de aprender uma linguagem de programação é a de automatizar certos cálculos que surgem em física ou engenharia, que não poderiam ser resolvidos manualmente, sem o auxílio de um computador.

Nas primeiras aulas precisamos introduzir alguns conceitos básicos de programação (que se aplicam a qualquer linguagem) e posteriormente precisamos introduzir a sintaxe específica da linguagem que vamos utilizar ao longo do curso, que é a linguagem python.

De maneira muito geral, as linguagens de programação, podem ser divididas em duas categorias:

- **Linguagens compiladas:** Dentre as primeiras temos linguagens tais como C, C++ e Fortran. A escrita de código neste tipo de linguagens de programação requer um domínio maior da sintaxe e das funcionalidades da linguagem. Como contrapartida, este tipo de linguagens produzem códigos que são muito rápidos pois tem sido otimizadas ao longo dos anos, e por tanto são usadas amplamente na Engenharia. De fato, a maioria dos códigos de cálculo usados na indústria (*Open Source* ou comerciais) estão feitos com elas. Ao **compilar** o código e gerar um arquivo binário otimizado, é possível tirar o maior proveito do poder de processamento de um computador.
- **Linguagens interpretadas:** Por outra parte, as linguagens interpretadas, como python e MatLab, são relativamente simples e intuitivas, pela sua flexibilidade na sintaxe, tornando o processo de desenvolvimento mais rápido e ágil. De maneira grosseira, o código vai sendo executado a medida que se **interpreta**. Porém, se não são tomados alguns recaudos no desenho do código, a performance computacional delas pode estar bastante aquém do necessário para resolver problemas de grande porte. Um exemplo prototípico disto, é quando utilizamos uma estrutura de repetição (como um `for`) no qual realizamos um grande número de operações em cada passo. Ao longo de curso iremos chamando a atenção sobre isto, tentando introduzir boas práticas de programação.

Como comparação, vejamos o exemplo de um código simples para criar um array com números de ponto flutuante de dupla precisão, popular ele com os números  $0, \dots, N$  e imprimir o resultado na tela, usando a linguagem compilada C (acima) e a linguagem interpretada python (embaixo).

1.1	Preludio . . . . .	1
1.2	Noções iniciais de python	2
1.3	Lista 1: Rudimentos básicos de programação . . . .	6

```
#include <stdio.h>
#include <stdlib.h>
int main(void)
{
    int i, N = 10;
    double *array;
    array=(double *) malloc(N*sizeof(double));
    for(i=0; i < N; i++) {
        array[i] = (double) i;
        printf("%lf\n", array[i]);
    }
    free(array);
    return 0;
}
```

```
import numpy as np
N = 10
array = np.arange(N)
print(array)
```

Olhando para o exemplo, já vemos que uma linguagem interpretada como python se torna mais prática para um primeiro curso de programação, pois o objetivo é desenvolver a capacidade de programar algoritmos para resolver problemas práticos no computador, sem gastar muito tempo no desenvolvimento de código.

Neste curso iremos adotar a linguagem python, a qual tem-se tornado bastante popular nos últimos anos. Para facilitar a implementação dos programas para resolver problemas de interesse, contaremos com algumas bibliotecas específicas que facilitarão o trabalho:

- ▶ **numpy**: Para manipulação eficiente de matrizes e vetores, operações de álgebra linear computacional e vários métodos numéricos.
- ▶ **scipy**: Para métodos numéricos mais avançados ou específicos, não cobertos pela anterior.
- ▶ **matplotlib**: Para plotagens e geração de gráficos em 2D e 3D.

## 1.2 Noções iniciais de python

Ao longo do curso iremos incorporando diversas ferramentas e funções disponíveis em várias bibliotecas, mas antes vamos a introduzir alguns conceitos essenciais de programação específicos de python. Se sugere ir digitando em algum editor de código ou algum ambiente de programação.

### 1.2.1 Tipos de variáveis

Alguns dos tipos de variáveis mais usados são apresentados na sequência:

## Números

São os objetos mais simples. Podem ser inteiros, de ponto flutuante, complexos e booleanos, por exemplo:

```
1, -2, 3.1415, 6.02e23, 1 + 1j, True, False
```

## Strings

São basicamente, listas de caracteres e se escrevem entre aspas simples ou duplas:

```
"a", "USP", "21", "AbCdE", "---"
```

## Listas

Servem para agrupar vários objetos.

```
mylist = [1, 3.14, "USP", True, -10000, 1+1j, myfunc]
```

A lista é indexada simplesmente pela posição do objeto, começando desde 0, p.e.,

```
mylist[1] é 3.14
```

```
mylist[6] é myfunc
```

## Dicionários

É uma forma mais prática de definir listas com identificadores:

```
mydic = {"valor": 3.14, "minhauni": "USP", "lista": ["a", 2, 1+1j]}
```

Então,

```
mydic{"minhauni"} é "USP"
```

```
mydic{"lista"}[2] é 1 + 1j
```

### 1.2.2 Estruturas condicionais

Uma das estruturas de programação mais usadas é a estrutura condicional.

A sintaxe é simples:

```
if (Expressão lógica):
```

```
    .
```

```
    .
```

```
else:
```

```
    .
```

```
    .
```

ou em situações com mais de duas opções para decidir:

```
if (Expressão lógica 1):
```

```

    .
    .
elif (Expressão lógica 2):
    .
    .
else:
    .
    .

```

Notar os : no final das sentencias e a indentação dentro de cada bloco.

#### Advertência

A indentação é fundamental em python. Se esta não for respeitada o interpretador não saberá quais instruções ficam dentro da estrutura e por tanto o programa poderá ter comportamentos não esperados ou em alguns casos parar a execução .

### 1.2.3 Estruturas de repetição

Existem duas estruturas de repetição que são muito usadas. A primeira estrutura é o famoso for (o "para" em português):

```

for i in range(N):
    .
    .

```

A segunda estrutura de repetição que iremos usar as vezes é o while (o "enquanto" em português):

```

while (Expressão lógica):
    .
    .

```

Com elas podemos fazer qualquer tipo de cálculo em que precisamos iterar sobre os elementos de algum objeto ou repetir uma operação várias vezes.

### 1.2.4 Funções

As declaração de funções é fundamental para poder organizar um código, encapsulando uma serie de operações as quais pode ser necessário realizar muitas vezes. A sintaxe para declarar uma função é:

```
def minhafunc(arg1, arg2, ...):
    .
    .
    return var1, var2, ...
```

Novamente, notar os : no final da definição e a indentação dentro do bloco da função.

### 1.2.5 A biblioteca numpy

Esta é uma das bibliotecas que mais serão usadas ao longo do curso. Basicamente permite definir vetores, matrizes e tensores em geral, populados com números ou dados de um tipo homogêneo (i.e., todos números inteiros, ou todos números de ponto flutuante, etc.). Esta biblioteca é fundamental para poder realizar cálculos científicos com uma eficiencia razoável, possivelmente similar à da uma linguagem compilada. Alguns exemplos de uso na sequência:

```
import numpy as np

vec_ints = np.array([1,2,3,4], dtype=np.int32)
vec_doubles = np.array([1,2,3,4], dtype=np.float64)
x = np.linspace(start, end)
y = np.sin(x)
vetor_nulo = np.zeros(10, dtype=np.int32)
matriz_nula = np.zeros(shape=(5,3), dtype=np.float64)
```

A ideia era mostrar alguns exemplos simples para o aluno se familiarizar um pouco com a sintaxe. Na sequência colocaremos os conceitos em prática desenvolvindo códigos para resolver vários problemas.

#### Aviso importante

O material de estudo para desenvolver a primeira lista será a presente apostila e os jupyter notebooks desenvolvidos pelo professor em sala de aula, os quais foram disponibilizados no repositório da disciplina. Outras fontes de informação a ser consideradas são os sites das bibliotecas que iremos utilizar, tais como <https://numpy.org> e <https://matplotlib.org/>, assim como consultas dirigidas ao professor em forma presencial ou por e-mail (rfausas@icmc.usp.br).

**A lista na sequência deve estar pronta para o dia 24/09.**



### 1.3 Lista 1: Rudimentos básicos de programação

A melhor forma para aprender a programar é resolver problemas concretos. Na sequência temos a primeira lista de exercícios para desenvolver códigos em python. Os exercícios poderão ser feitos em grupos de no mínimo 2 e no máximo 2 integrantes, e a partir da data definida acima, o professor chamará aleatoriamente aos grupos para explicar os exercícios. Todos os membros do grupo deverão ser capazes de explicar qualquer um dos exercícios.

#### Exercícios preliminares para praticar e não serão avaliados

1. Fazer um script que define um vetor randômico de dimensão  $N$  e calcula a média dos valores das suas componentes, i.e.,

$$\bar{x} = \frac{1}{N} \sum_{i=1}^N x_i$$

Considerar  $N = 10, 100, 1000, 10000, 100000, 1000000$ .

2. Fazer um script que define uma matriz randômica de dimensão  $N \times N$  e calcula a média dos valores dos seus elementos, i.e.,

$$\bar{A} = \frac{1}{N \times N} \sum_{i=1}^N \sum_{j=1}^N A_{ij}$$

tomando  $N = 10, 100, 500, 1000$ .

3. Fazer um gráfico da função  $f(x) = x^m$  para diferentes valores de  $m$  considerando o intervalo  $x \in [1, 4]$ . Usar escala linear e escala loglog.
4. Fazer uma função que calcula o produto escalar de dois vetores randômicos  $\mathbf{a}$  e  $\mathbf{b}$  de  $\mathbb{R}^3$ , calcula a sua magnitude e determina o ângulo  $\theta$  que eles formam, usando a fórmula:

$$\mathbf{a} \cdot \mathbf{b} = \|\mathbf{a}\| \|\mathbf{b}\| \cos \theta$$

5. Considerar 10 dados que são jogados, podendo sair números de 1 até o 6. A soma dos valores será

$$S = \sum_{i=1}^{10} d_i$$

em que  $d_i$  é o que saiu em cada um dos dados. Jogar os 10 dados 100000 vezes e construir um histograma que mostre o comportamento de  $S$ . Um histograma seria um gráfico de frequência de um certo evento, ou seja, quantas vezes a soma deu 10, quantas vezes a soma deu 11, quantas vezes a soma deu 12, ..., quantas vezes a soma deu 60. Para isto precisará usar a função

```
counts, bins = np.histogram(s)
plt.stairs(counts, bins)
```

em que  $s$  será um vetor que guardou o resultados das 100000 realizações.

**Exercícios que serão avaliados**

1. Fazer uma função que:

- Pega dois vetores randômicos  $\mathbf{a}$  e  $\mathbf{b}$  de dimensão  $n$ , e dois escalares randômicos  $\alpha$  e  $\beta$  e calcula um vetor  $\mathbf{c}$  tal que

$$\mathbf{c} = \alpha \mathbf{a} + \beta \mathbf{b}$$

- Pega uma matriz randômica  $\mathbf{A}$  de  $n \times n$  e calcula a sua  $m$ -essima potência

$$\mathbf{A}^m = \underbrace{\mathbf{A} \dots \mathbf{A}}_{m \text{ vezes}}$$

(tomar valores de  $m = 2, 3, 4$ ).

Em todos os casos medir o tempo necessário para realizar as operações para diferentes dimensões  $n$ . Plotar o tempo de cálculo como função da dimensão  $n$  usando a escala linear padrão e a escala  $\log \log$ . No segundo ponto, colocar no mesmo gráfico os resultados para os diferentes valores de  $m$ . Tirar conclusões.

**Nota:** Para que os resultados sejam interessantes, no primeiro ponto, tomar valores de  $n = 10^5, 10^6, \dots, 10^8$ . Já, no segundo ponto, tomar valores de  $n = 500, 1000, 1500, 2000$ .

2. Fazer uma função que recebe uma matriz randômica  $\mathbf{A}$  de dimensão  $m \times r$  e outra matriz randômica  $\mathbf{B}$  de dimensão  $r \times n$ , verifica as suas dimensões e realiza a multiplicação delas no sentido usual de álgebra linear, para retornar uma matriz  $\mathbf{C}$  de dimensão  $m \times n$ . A multiplicação deve ser feita usando todos os `for` que sejam necessários. Considerando matrizes quadradas (i.e.,  $m = r = n$ ), medir o tempo de cálculo como função da dimensão e comparar com o tempo necessário fazendo  $\mathbf{A} @ \mathbf{B}$ . Para que os resultados sejam interessantes tomar dimensões de matriz 50, 100, 150, 200, 250, como anteriormente. Novamente, plotar o tempo de cálculo como função da dimensão na escala  $\log \log$ .

3. **Mapeo logístico:** Considerar uma sequência de números gerada da seguinte forma:

$$x_n = a x_{n-1} (1 - x_{n-1}), \quad n = 1, 2, \dots, N$$

Considerar  $N = 5000$ ,  $x_0 = 0.1$  e diferentes valores de  $a$  entre 0 e 4 (p.e.,  $a = 1, 2, 3.8$  e 4). Calcular a média e a variância da sequência:

$$\bar{x} = \frac{1}{N} \sum_{i=0}^N x_i, \quad \sigma = \frac{1}{N-1} \sum_{i=0}^N (x_i - \bar{x})^2$$

Programar-lo na mão e usando funções de `numpy` já prontas. Comparar os resultados.

4. No problema 3, plotar a sequência de valores obtida em cada

caso considerando os diferentes valores de  $a$  pedidos. Fazer os gráficos usando legendas, labels, e outros atributos que achar interessante, para melhor ilustrar os resultados.

5. Continuando com a sequência do problema 3, fazer um código que gera o diagrama de bifurcações, que é um gráfico que mostra os valores que assume a sequência, como função dos valores de  $a$ . O resultado deveria ser algo do tipo mostrado na figura ao lado, que no eixo horizontal tem os valores de  $a$  usados para gerar a sequência e no eixo vertical todos os possíveis valores que toma a sequência para o correspondente valor de  $a$ . Se sugere usar pontinhos bem pequenos para gerar o gráfico. Explicar os resultados se auxiliando com os gráficos do exercício anterior.

6. **Algo de grafos:** Considerar uma rede de distribuição com a mostrada na figura ao lado. Esta pode ser um exemplo de uma rede elétrica ou hidráulica e é basicamente o que se chama um *grafo*). Notar que em geral ela estará caracterizada por um certo número de *nós* (ou uniões), um certo número de *arestas* e alguma informação sobre a conectividade entre pontos.

- ▶ Fazer uma estrutura de dados que sirva para descrever essa rede. Idealmente, a estrutura deveria incluir algum tipo de matriz ou *array* que indica como os nós e as arestas estão relacionados. Adicionalmente, a estrutura deve conter um array para descrever as coordenadas  $(x, y)$  de cada nó. Construir um exemplo inventando as coordenadas e plotar a rede.
- ▶ Fazer uma função que deleta um nó e todas as arestas que emanam dele.
- ▶ Fazer uma outra função em python que permita apagar (ou *deletar*) uma aresta da rede. Notar que se a aresta possui um nó que não pertence a nenhuma outra aresta, esse nó também precisa ser deletado.
- ▶ Finalmente, fazer uma função que insere uma nova aresta na rede para conectar dois pontos já existentes.

7. **O jogo da vida de Conway:** O Jogo da vida é uma grade ortogonal bidimensional de células quadradas, cada uma das quais está em um dos dois estados possíveis: viva ou morta. Cada célula interage com seus oito vizinhos, que são as células adjacentes horizontalmente, verticalmente ou diagonalmente. A cada passo no tempo, ocorrem as seguintes transições:

- ▶ Qualquer célula viva com menos de dois vizinhos vivos morre;
- ▶ Qualquer célula viva com dois ou três vizinhos vivos continua viva para a próxima geração;
- ▶ Qualquer célula viva com mais de três vizinhos vivos morre;
- ▶ Qualquer célula morta com exatamente três vizinhos vivos torna-se uma célula viva.

A tarefa é fazer um programa de python que implementa o jogo

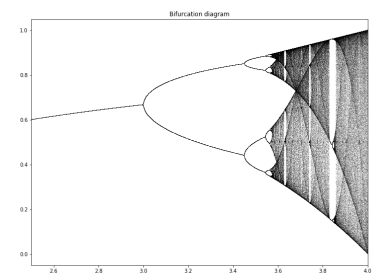


Figure 1.1: Diagrama de bifurcações.

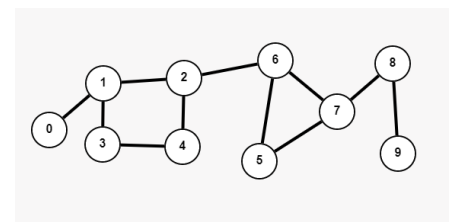


Figure 1.2: Exemplo de uma rede.

da vida:

- Uma grade com  $100 \times 100$  células e condições iniciais randômicas. que dependam de dois parâmetros  $p_0$  e  $p_1$  sendo as probabilidades de iniciar morta ou viva, respetivamente. Testar diferentes valores;
  - Uma grade menor e condições iniciais como as descritas no [https://en.wikipedia.org/wiki/Conway%27s\\_Game\\_of\\_Life](https://en.wikipedia.org/wiki/Conway%27s_Game_of_Life), de forma a reproduzir alguns padrões clássicos conhecidos como **Still lifes**, **Oscillators** e **Spaceshifts**.
- O programa dever estar estruturado da seguinte forma:

```
# Grid size
N = 100

# Create an initial random grid
p0, p1 = 0.8, 0.2
grid = np.random.choice([0, 1], N*N, p=[p0, p1]).reshape(
    N, N)

def update(frameNum, img, grid):
    # Programar as regras de atualizacao
    .
    .
    .
    plt.title(f"Game of Life - Frame {frameNum}")
    return img,

fig, ax = plt.subplots()
img = ax.imshow(grid, interpolation='nearest')
ani = animation.FuncAnimation(fig, update, fargs=(img,
    grid,))
plt.show()
```

Explicar que o que cada parte do código faz.



# Alphabetical Index

preface, ii