

Lista de Computação II (Python)

DCC / UFRJ

Pedro Maciel Xavier (monitor)
pedromxavier@poli.ufrj.br

January 11, 2020

Introdução

Essa lista de exercícios ainda se encontra em desenvolvimento. A intenção é que ela tenha um gabarito bem aberto, deixando muito das respostas para a criatividade do aluno. As questões são, em geral, grandes para se resolver e podem necessitar de alguma pesquisa adicional. Elas tem estrelinhas ★ indicando a dificuldade estimada. Alguns exercícios foram inspirados em outros propostos em materiais cujas fontes estão devidamente referenciadas no final. É importante você tire suas dúvidas e dê um retorno do que achou dos exercícios através do e-mail no cabeçalho.

Boa diversão!



Contents

1	Orientação a Objeto (class)	4
1.1	O Bidicionário ★	4
1.2	Ora Bolas! ★	4
1.3	Frações ★	5
1.4	Polinômios [3] ★★	6
1.5	Grupos ★★★★★	8
1.5.1	Subparte I: Produto	9
1.5.2	Subparte II: Quociente	9
1.5.3	Subparte III: Subgrupos	9
1.6	Quaterniões! ★★	10
2	Interface Gráfica (tkinter)	12
2.1	Preenchendo o espaço. !Incompleto! ★★★★★	12
2.2	Números Primos II !Incompleto! ★★★★★	12
2.3	O Triângulo de Sierpinsky ★★	12
2.4	Pôr-do-Sol [4] ★★	13
2.5	O Método de Monte Carlo ★★	14
2.6	A proporção áurea ★★	15
2.7	Música IV - O Piano ★★	16
3	Métodos numéricos (numpy)	17
3.1	Redes Neurais !Incompleto! ★★	17
3.2	Computação Quântica !Incompleto! ★★★★★	18
4	Plotagem de Gráficos (Matplotlib)	19
4.1	Ying-Yang ★★	19
5	Conexão e Redes (socket)	20

Revisão de Computação I

Não sei se vai ter isso aqui não em.

1 Orientação a Objeto (**class**)

1.1 O Bidicionário ★

Todos conhecemos os dicionários do Python, que guardam diversos objetos em pares da forma "chave":valor. Vejamos um exemplo:

```
1 >>> casa = {  
2     'quartos':4,  
3     'banheiros':5,  
4     'andares':3,  
5     'm^2':210  
6 }  
7  
8 >>> casa['quartos']  
9 4
```

Listing 1: "Dicionário Normal"

O objetivo deste exercício é criar um dicionário de mão dupla! Tudo que você precisa fazer é sobrescrever o método `__setitem__` em um novo tipo que herda as propriedades de um dicionário comum do Python, o **dict**. Basta completar o exemplo abaixo!

```
1 class Bidict(dict):  
2     def __init__(self, mapping={}):  
3         dict.__init__(self, mapping)  
4         ...  
5 >>> bd = Bidict()  
6 >>> bd["a"] = 4  
7 >>> bd[3] = "d"  
8 >>> print(bd)  
9 {"a":4, 4:"a", 3:"d", "d":3}
```

Lembrando que se um objeto não puder ser chave de um dicionário, ele também não poderá ser um valor do bidicionário!

1.2 Ora Bolas! ★

Mais um exercício pra esquentar: Crie uma classe chamada `Bola` que deve implementar objetos com as seguintes características:

1. O raio nominal da bola.
2. A pressão de ar máxima (em *bar*).
3. A pressão de ar atual. (em *bar*).
4. A condição da bola (furada ou não).
5. Uma onomatopeia correspondente ao barulho que a bola faz quando quica, e outra para caso ela fure.
6. A probabilidade da bola furar quando quica.

Além disso, tendo uma bola em mãos você deve poder:

1. Quicar! Caso ela não esteja vazia.
2. Encher em alguma quantidade de *bar*, passada como argumento da função, caso ela não esteja furada. Se você encher de mais ela deve furar!
3. Calcular o seu volume.

Feita a bola, você deve criar uma classe `BolaQuadrada` que herde as propriedades de uma bola comum, mas tenha as adaptações necessárias para o seu formato. Ela deve, por exemplo, ter 50% de chance de quicar em uma tentativa.

Faça também a classe `BolaDeFesta`, que deve furar sempre que quicar. Dê a ela um som de estouro interessante.

1.3 Frações ★

Um número racional $r \in \mathbb{Q}$ é aquele que pode ser escrito como:

$$r = \frac{p}{q}; p, q \in \mathbb{Z}; q \neq 0$$

Desafio: Crie uma classe `Fracao` que implemente as seguintes operações e métodos:

1. Um método que simplifique a fração.
2. As operações:
 - `+` (`__add__`)
 - `-` (`__sub__`)
 - `*` (`__mul__`)
 - `**` (`__pow__`)
 - `/` (`__truediv__`)
 - `-` (`__neg__`)
 - `~` (`__invert__`).
3. `__repr__` que retorna "`p|q`".
4. `__float__`, que calcula a divisão em ponto-flutuante.

```

1 class Fracao(object):
2
3     def __init__(self, p, q):
4         assert type(p) is int
5         assert type(q) is int
6         assert q != 0
7         ...

```

1.4 Polinômios [3] ★★

Um polinômio de grau n é aquele da forma

$$p(x) = \sum_{i=0}^n a_i x^i = a_0 + a_1 x + \cdots + a_n x^n.$$

Desafio: Sabendo isso, faça uma classe `Polinomio` que implemente:

1. A definição de um coeficiente do polinômio através do método `__setitem__`, ou seja, `p[2] = 3` faria com que o coeficiente a_2 do polinômio p tivesse valor 3.
2. O cálculo do grau do polinômio p , que deve ser retornado quando chamamos `len(p)`.

3. A soma, a subtração e a multiplicação usual de polinômios usando os operadores da linguagem (+, -, *), que deve retornar um novo objeto da classe `Polinomio`.
4. E por fim, a avaliação da função num ponto x , usando o método especial `__call__`.

Desafio Bônus:

1. O método especial `__repr__` que deve retornar uma **string** que represente o polinômio $2 + x + 3x^2$ na forma `"2 + x + 3x^2"`, por exemplo.
 2. Duas funções, `Polinomio.integral` e `Polinomio.derivada` que retornem os respectivos polinômios resultantes destas operações.
 3. Divisão de polinômios (f/g e $f\%g$), que devem retornar respectivamente o quociente $q(x)$ e o resto $r(x)$ tais que $f(x) = g(x)q(x) + r(x)$.
-

Interlúdio: Beep

Um jeito fácil de fazer barulho no computador é usando a função `Beep`. Se você usa o Python no Windows boas notícias: você só vai precisar importar a função `Beep` do módulo `winsound`.

```
>>> Beep(440, 1000)
```

```
1 # windows
2 from winsound import Beep
```

Pra turminha do Linux, que precisa construir a função:

```
$ sudo apt-get install beep
```

```
1 # linux
2 import os
3 def Beep(f, ms):
4     os.system("beep -f%i -l%i" % (f, ms))
```

1.5 Quaterniões! ★★★

Não contente com os números complexos (\mathbb{C}) da forma $z = a + b\mathbf{i}$, a turma da matemática trouxe pra gente um ser ainda mais esquisito: o conjunto dos *Quaternions* (\mathbb{H}).

$$\begin{aligned}q \in \mathbb{H} &\iff q = a + b\mathbf{i} + c\mathbf{j} + d\mathbf{k} : a, b, c, d \in \mathbb{R} \\ \mathbf{i} \times \mathbf{j} &= -(\mathbf{j} \times \mathbf{i}) = \mathbf{k} \\ \mathbf{j} \times \mathbf{k} &= -(\mathbf{k} \times \mathbf{j}) = \mathbf{i} \\ \mathbf{k} \times \mathbf{i} &= -(\mathbf{i} \times \mathbf{k}) = \mathbf{j} \\ \mathbf{i} \times \mathbf{j} \times \mathbf{k} &= -1\end{aligned}$$

Desafio: Com regras de soma convencionais, similares aos números reais e complexos, mas com uma multiplicação que mais parece o produto externo entre vetores, implemente uma classe `Quaternion`, que é criada a partir dos seus 4 coeficientes reais e implementa:

1. O operador de soma `+` (`--add--`)
2. A subtração `-` (`--sub--`)
3. A multiplicação `*` (`--mul--`)
4. O método `--repr--` que imprima na tela, para $q = 3 + 4\mathbf{j} - 2\mathbf{k}$, a *string*
`(3.00) + (0.00)i + (4.00)j + (-2.00)k`

Bônus:

1. O módulo do quaterniões, através da função `abs(q)`, implementada pelo método especial `--abs--`
2. A divisão `/` (`--truediv--`)

Dica: Implemente os métodos `--invert--` e `--neg--` para auxiliar na definição de `--truediv--` e `--sub--`, respectivamente.

2 Interface Gráfica (`tkinter`)

2.1 Preenchendo o espaço. **!Incompleto!** ★★★★★

Curvas de Hilbert

2.2 Números Primos II **!Incompleto!** ★★★★★

Muitas figuras interessantes se formam a partir dos números primos.

Desafio: Vamos usar o mesmo programa da questão anterior [2], mas com uma singela modificação: Definimos um contador e, a cada passo, a pintura só ocorre se o número for primo. Em seguida, incrementamos o contador e seguimos adiante!

2.3 O Triângulo de Sierpinsky ★★★

A figura a seguir chama-se *Triângulo de Sierpinsky*:

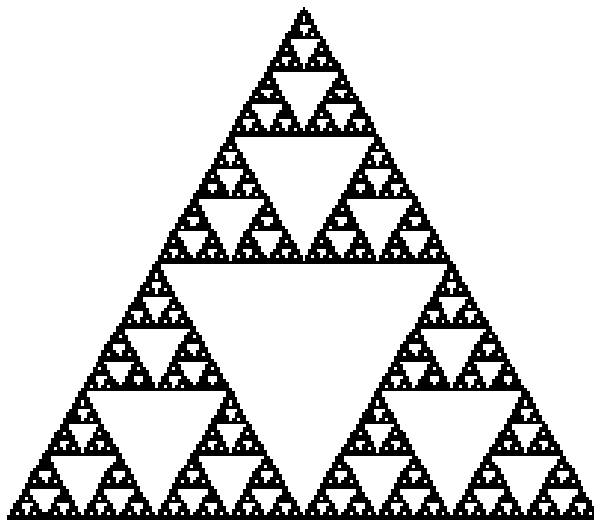


Figure 1: Triângulo de Sierpinsky

Ela é composta por triângulos equiláteros dispostos de forma que cada triângulo contém três outros em seu interior, cada um com $\frac{1}{2}$ do lado do triângulo original.

Desafio: Desenhe essa forma usando o `tkinter` ou o `turtle`. Crie uma função para compôr a figura de forma recursiva.

2.4 Pôr-do-Sol [4] ★★★

Todos os dias o Sol se põe no horizonte da mesma maneira.

Desafio: Faça uma visualização do pôr-do-Sol no `tkinter`, da forma que quiser. Se você não se sente muito inspirado, segue a receita do bolo:

1. Crie um Canvas. **Dica:** uma vez criado o Canvas, posicione ele com o método `pack`, passando as opções `expand=True` e `fill="both"` para que o Canvas ocupe toda a tela.
2. Faça primeiro o céu. Você pode criar um retângulo ou simplesmente alterar a propriedade `bg` do Canvas (cor do *background*).
3. Em seguida, desenhe o Sol. O seu movimento aparente no horizonte é bem descrito por um seno (ou cosseno).
4. Hora de fazer o horizonte. A linha fica bem no meio da tela, mas isso fica a seu critério. Um retângulo na parte inferior já é suficiente, mas você também pode compor diversos círculos ou triângulos para dar forma ao relevo, como em uma colagem.

```
1 import tkinter as tk
2
3 class Janela:
4     def __init__(self, root):
5         self.root = root
6
7         self.canvas = tk.Canvas(self.root)
8         self.canvas.pack(expand=True, fill="both")
9
10    ...
```

```
11  
12 root = tk.Tk()  
13 self = Janela(root)  
14 root.mainloop()
```

2.5 O Método de Monte Carlo ★★

O Método de *Monte Carlo* funciona da seguinte forma: Se temos uma determinada região e queremos calcular sua área, basta fazer com que ela esteja contida em uma outra região cuja área é conhecida. Em seguida, sorteamos pontos aleatórios $P_i = (x_i, y_i)$ e contamos quantos pontos caem dentro da região que estamos avaliando. Assim:

$$\frac{A_{figura}}{A_{total}} \approx \frac{P_{dentro}}{P_{total}} \rightarrow A_{figura} \approx \frac{P_{dentro}}{P_{total}} \times A_{total}$$

Queremos então calcular a área de um círculo cujo raio é 1. Sabemos de antemão que valor da área é π , e vamos então usar o método acima para estimar o seu valor numérico.

Faça duas janelas usando o `tkinter`. A primeira mostra onde os pontos estão sendo posicionados, colorindo os que caem dentro de **azul** e os que caem fora de **vermelho**. Na segunda tela, mostre em tempo real a aproximação para a área em azul conforme os pontos vão sendo posicionados.

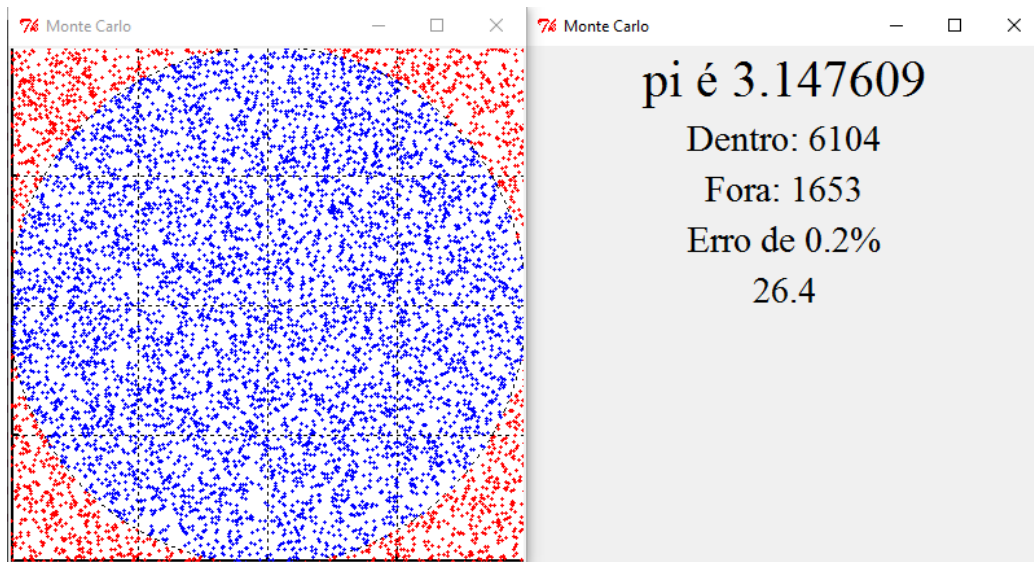


Figure 2: Cálculo de π pelo método de Monte Carlo

2.6 A proporção áurea ★★

0 1 1 2 3 5 8 13 21 34 55 89 ...

Muitos conhecem esse conjunto de números, a famosa *Sequência de Fibonacci*, que tem seu n -ésimo número definido por:

$$\begin{cases} F_n = F_{n-1} + F_{n-2} \\ F_0 = 0 \text{ e } F_1 = 1 \end{cases}$$

Se tomamos a razão entre dois números de *Fibonacci* consecutivos obtemos, no limite, um número que já era conhecido pelos gregos como símbolo da beleza e da perfeição da natureza.

$$\varphi = \lim_{n \rightarrow \infty} \frac{F_{n+1}}{F_n} = \frac{1 + \sqrt{5}}{2} \approx 1.618$$

Sabido isso, sua tarefa é gerar a figura abaixo, usando o módulo que preferir, mas de forma recursiva. Para que os gregos realmente fiquem contentes com a majestade da sua figura é preciso que ela esteja conforme a proporção dada por φ .

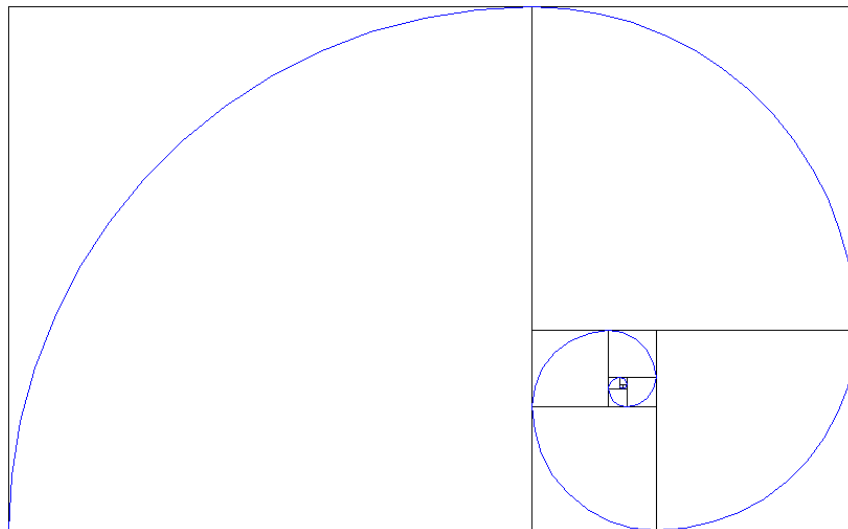


Figure 3: A proporção áurea

2.7 Música IV - O Piano ★★★

Desafio: Use os resultados das etapas anteriores (**Musica I, II e III**) e construa um Piano Virtual com o `tkinter`. Você pode desenhar as teclas em um `Canvas` e usar o método `Canvas.bind` para associar letras como `a, s, d, f, g, h, j` e `k` às teclas brancas e `w, e, t, y` e `u` às pretas, por exemplo. Outra forma de fazer é usar o `widget Button` para fazer cada tecla.

- A mantissa é composta pela soma de potências de $\frac{1}{2}$, portanto números que não são resultado de somas finitas dessas potências vão apresentar erros de representação.

3.2 Computação Quântica **!Incompleto!** ★★★★★

$$|\uparrow\rangle + |\downarrow\rangle \quad |A\rangle + |\Omega\rangle \quad |0\rangle + |1\rangle$$

Figure 5: Superposição

4 Plotagem de Gráficos (Matplotlib)

Interlúdio: `nan` e `inf`

Um outro tópico interessante de se falar, ainda no escopo da **aritmética de ponto-flutuante** (`float`), é a presença de três números especiais no padrão **IEE 754**: `nan`, `inf` e `-inf`.

Como o nome já indica, `inf` e `-inf` são usados pra representar o infinito positivo ∞ e o negativo $-\infty$, respectivamente. É comum que surjam durante operações de divisão por zero, ou alguma outra operação que exceda a precisão do expoente.

O `nan` (***n**ot **a** **n**umber*), por sua vez, é fruto de operações indeterminadas, como $\infty - \infty$ e $\frac{0}{0}$. É particularmente útil para representar valores ausentes. Quando presente em um `array` que será usado para plotar um gráfico, o `Matplotlib` simplesmente ignora as entradas, criando lacunas na curva desenhada.

É possível obter esses números usando a função `float`, passando como parâmetro o nome do número desejado:

```
1 >>> x = float("inf")
2 >>> y = float("nan")
3 >>> z = float("-inf")
```

Listing 2: 'nan e inf'

4.1 Ying-Yang ★★

Parte positiva de uma cor e parte negativa de outra.

5 Conexão e Redes (**socket**)

Interlúdio: **ipconfig/ifconfig**

```
1 $ ifconfig
```

```
1 > ipconfig
```

References

- [1] Prof. Pedro Asad (2016)
- [2] Prof. Brunno Goldstein (2016)
- [3] Prof. Claudio Esperança (2018)
- [4] Prof. Jonas Knopman (2018)