



Exercícios de ordenação (bolha, seleção e inserção)

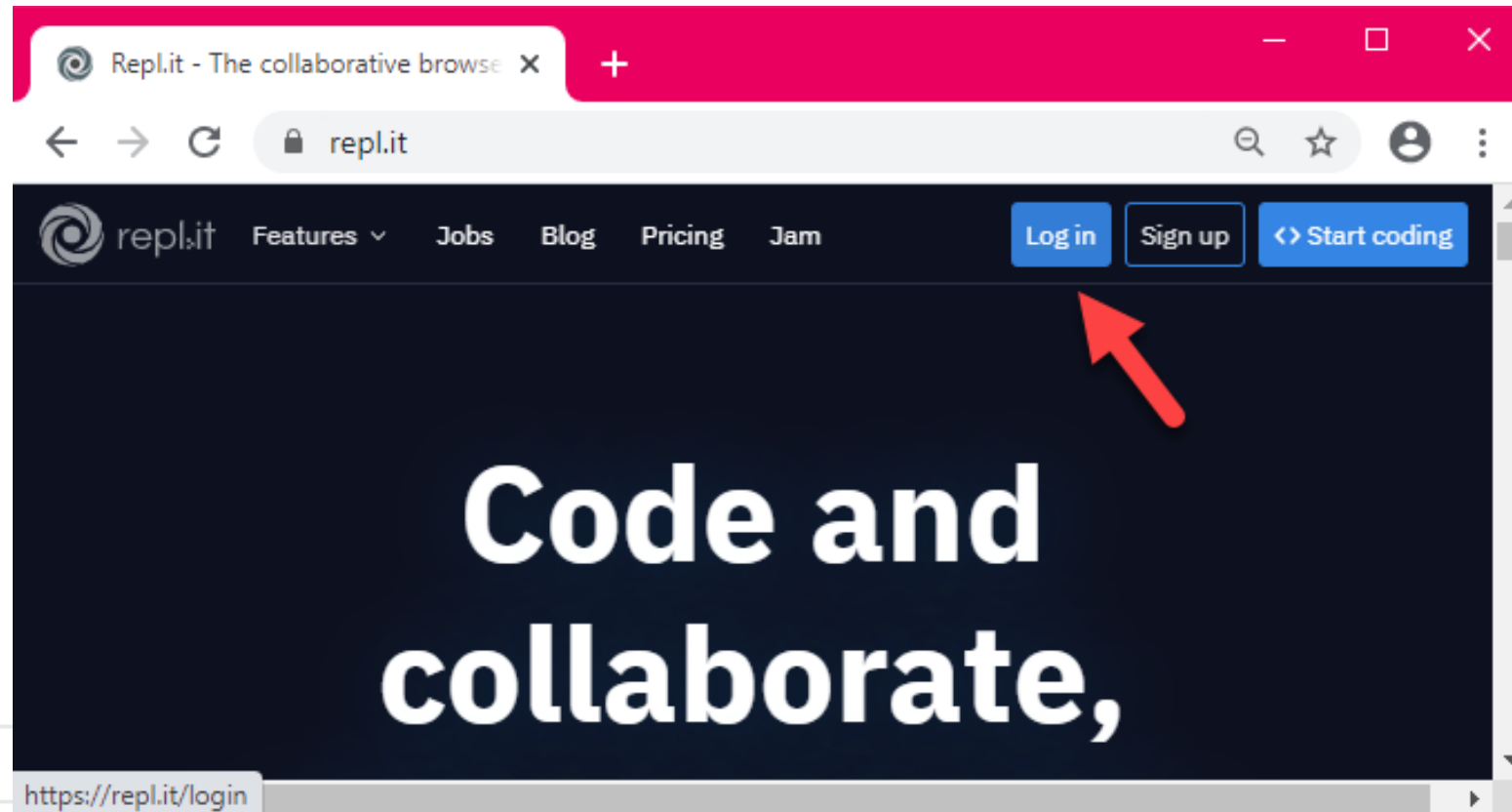
Algoritmos e Programação II - Turma 02N - 2º semestre de 2020

Prof. Dr. Bruno da Silva Rodrigues

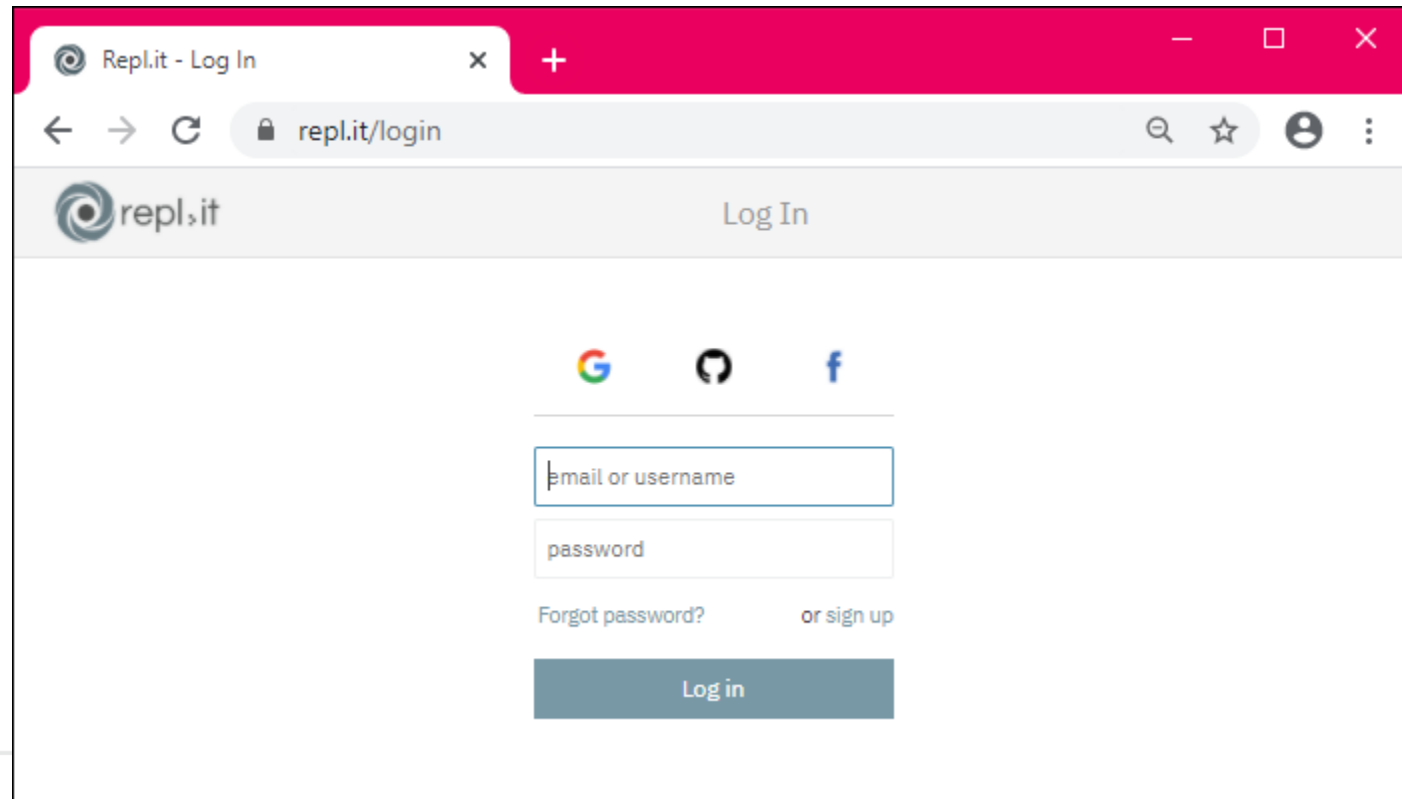
Prof. Tomaz Mikio Sasaki



Acesse <https://repl.it> e efetue o login na aplicação.



Efetue o login utilizando uma das contas sugeridas (Google, GitHub, Facebook), ou crie uma nova **conta gratuita** na aplicação.



The image shows a web browser window with the title "Repl.it - Log In". The address bar displays "repl.it/login". The page header includes the Repl.it logo and the text "Log In". Below the header, there are three social media login options: Google, GitHub, and Facebook. Underneath these, there are two input fields: "email or username" and "password". Below the password field, there are two links: "Forgot password?" and "or sign up". At the bottom, there is a blue "Log in" button.

Repl.it - Log In

repl.it/login

repl.it Log In

Google GitHub Facebook

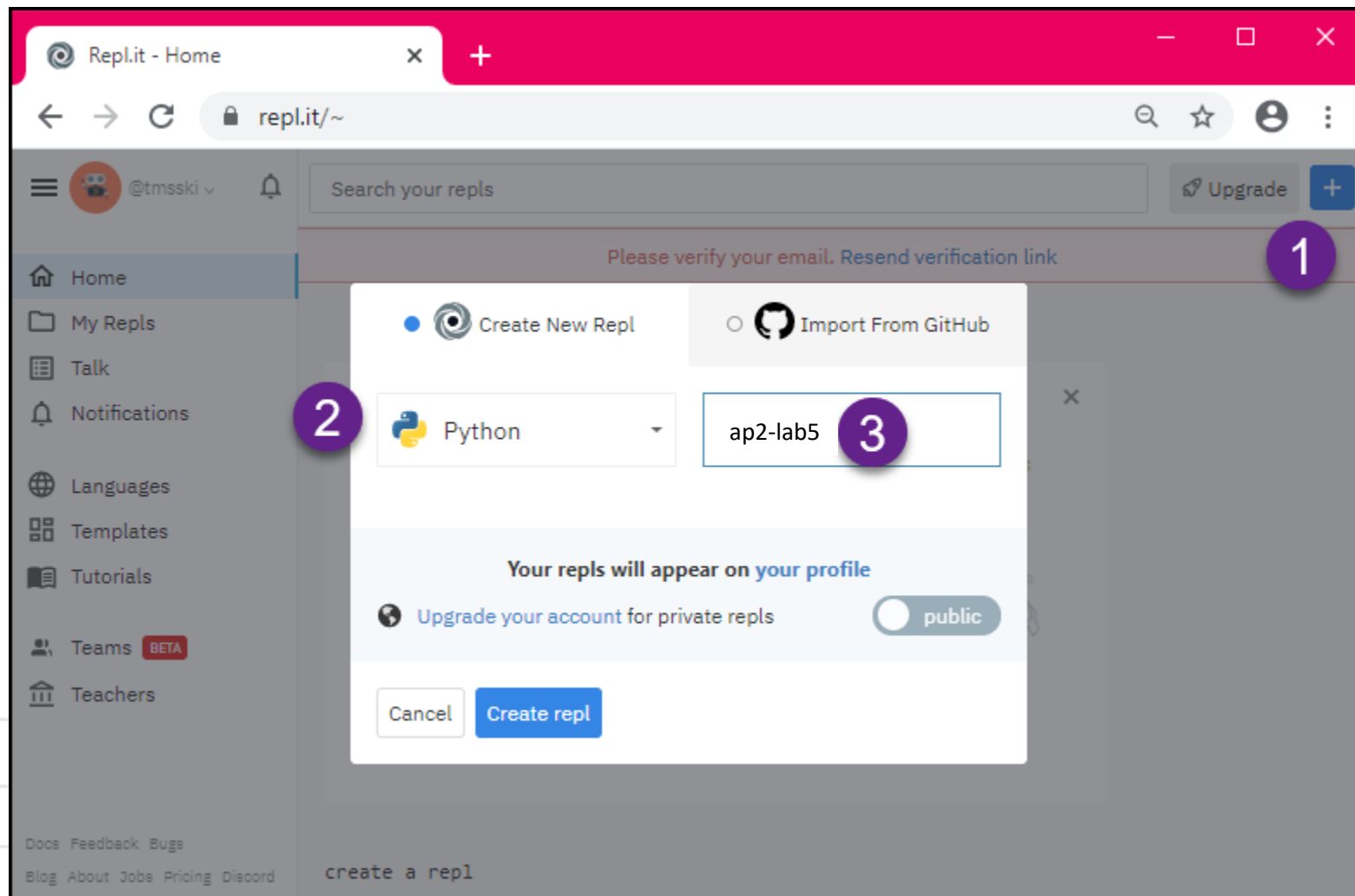
email or username

password

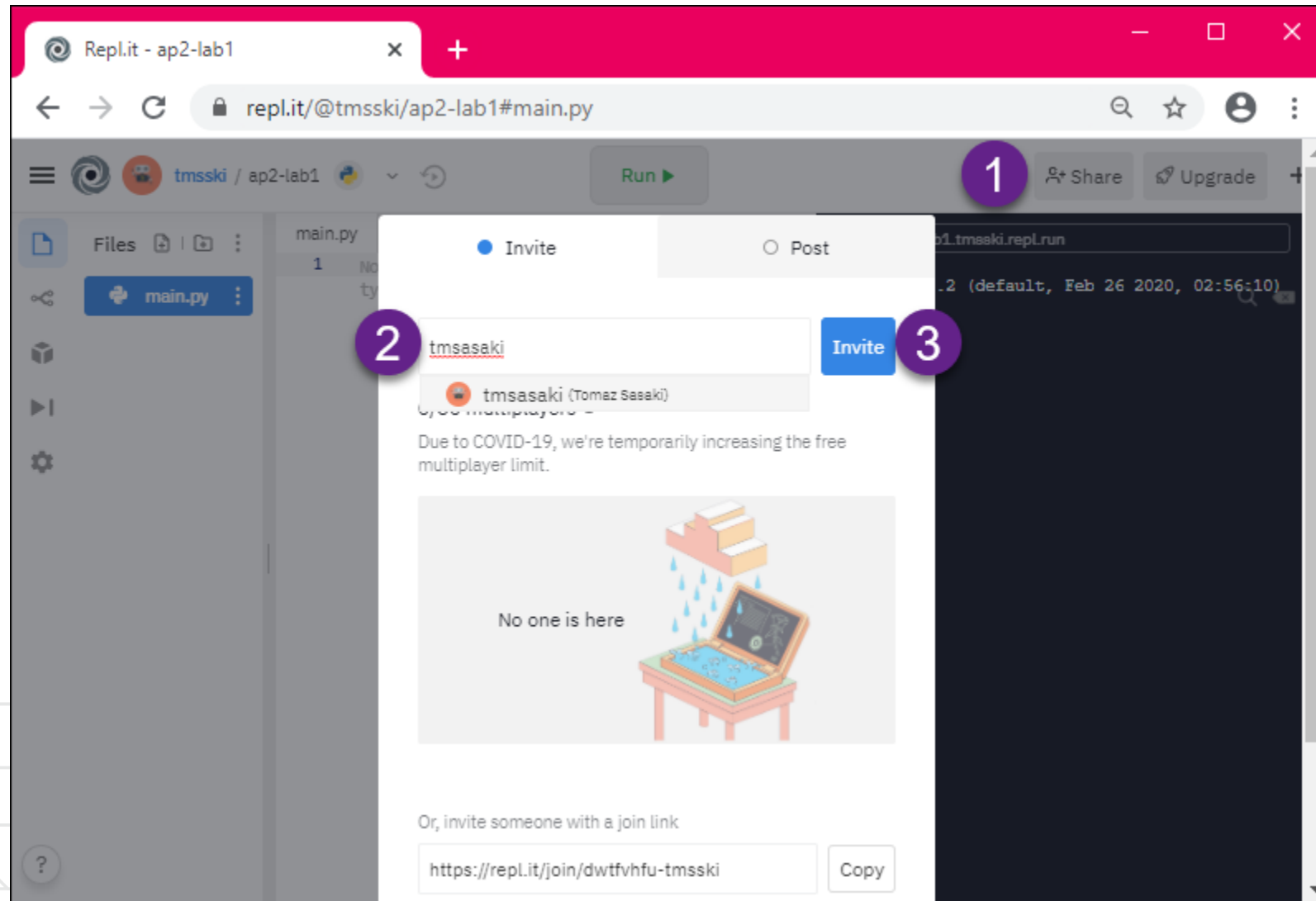
Forgot password? or sign up

Log in

Selecione + para criar um novo REPL, selecione a linguagem Python e dê o nome **ap2-lab5**.



Convide o professor para o seu REPL. Assim ele poderá acompanhar a execução dos exercícios.



"Combinado" da nossa disciplina

Como combinamos em aulas anteriores, vamos realizar todos os exercícios desta disciplina obedecendo às seguintes limitações (para simular em Python um *array* simples):

- Criaremos as listas com um tamanho inicial e não realizaremos nenhuma operação que altere este tamanho inicial.
- Em cada lista armazenaremos elementos de um único tipo.
- Não utilizaremos as diversas funções e métodos já prontos em Python que realizam operações nas listas, com exceção da função **len**.

Crie um arquivo com o nome **lab5.py**, e escreva nele o seguinte código:

```
1  import random
2
3  def contem(v, qtd, e):
4      """ verifica se o vetor 'v', que tem 'qtd' elementos, contém o elemento 'e' """
5      for i in range(qtd):
6          if v[i] == e:
7              return True
8      return False
9
10 def inverter(v):
11     """ inverte a ordem dos elementos do vetor 'v' """
12     n = len(v)
13     for i in range(n//2):
14         aux = v[i]
15         v[i] = v[n-1-i]
16         v[n-1-i] = aux
17
18 def gerar_aleatorio(tamanho):
19     """ gera um vetor com o tamanho solicitado e elementos pseudo-aleatórios """
20     v = [0] * tamanho
21     qtd = 0
22     n = len(v) * 10
23     while (qtd < len(v)):
24         e = random.randint(1, n)
25         if not contem(v, qtd, e):
26             v[qtd] = e
27             qtd += 1
28     return v
29
```

Preparação

Continuação do conteúdo do arquivo lab5.py:

```
30 def gerar_crescente(tamanho):
31     """ gera um vetor com o tamanho solicitado e elementos pseudo-aleatórios em ordem crescente """
32     v = [0] * tamanho
33     qtd = 0
34     inicial = 1
35     final = len(v) * 10
36     while (qtd < len(v)):
37         e = random.randint(inicial, final)
38         v[qtd] = e
39         qtd += 1
40         inicial = e + 1
41         final = e + len(v)*10
42     return v
43
44 def gerar_decrescente(tamanho):
45     """ gera um vetor com o tamanho solicitado e elementos pseudo-aleatórios em ordem decrescente """
46     v = gerar_crescente(tamanho)
47     inverter(v)
48     return v
49
50 def bubbleSort(alist):
51     """ ordena os elementos do vetor utilizando o algoritmo de ordenação da bolha;
52     retorna o número de execuções do laço de repetição """
53     cont = 0
54     for passnum in range(len(alist)-1,0,-1):
55         for i in range(passnum):
56             cont += 1
57             if alist[i]>alist[i+1]:
58                 temp = alist[i]
59                 alist[i] = alist[i+1]
60                 alist[i+1] = temp
61     return cont
62
```


Continuação do conteúdo do arquivo lab5.py:

```
30 def gerar_crescente(tamanho):
31     """ gera um vetor com o tamanho solicitado e elementos pseudo-aleatórios em ordem crescente """
32     v = [0] * tamanho
33     qtd = 0
34     inicial = 1
35     final = len(v) * 10
36     while (qtd < len(v)):
37         e = random.randint(inicial, final)
38         v[qtd] = e
39         qtd += 1
40         inicial = e + 1
41         final = e + len(v)*10
42     return v
43
44 def gerar_decrescente(tamanho):
45     """ gera um vetor com o tamanho solicitado e elementos pseudo-aleatórios em ordem decrescente """
46     v = gerar_crescente(tamanho)
47     inverter(v)
48     return v
49
50 def bubbleSort(alist):
51     """ ordena os elementos do vetor utilizando o algoritmo de ordenação da bolha;
52     retorna o número de execuções do laço de repetição """
53     cont = 0
54     for passnum in range(len(alist)-1,0,-1):
55         for i in range(passnum):
56             if alist[i]>alist[i+1]:
57                 temp = alist[i]
58                 alist[i] = alist[i+1]
59                 alist[i+1] = temp
60             cont += 1
61     return cont
62
```

Note as modificações feitas na função de **bubbleSort** para contar as execuções do laço

Crie um arquivo chamado **exemplo1.py** com o seguinte conteúdo:

```
exemplo1.py
1  from lab5 import *
2
3  v1 = gerar_aleatorio(5)
4  print('\n## Vetor aleatório com 5 elementos:\n', v1)
5
6  v2 = gerar_crescente(7)
7  print('\n## Vetor ordenado com 7 elementos:\n', v2)
8
9  v3 = gerar_decrescente(11)
10 print('\n## Vetor decrescente com 11 elementos:\n', v3)
```

Exemplo 1

Siga os passos ao lado para executar o **exemplo1**.

1 faça a importação do exemplo1 no arquivo main.py

2 execute

3 verifique o resultado

```
main.py
1  import exemplo1
2
3

https://ap2-lab5.tmsski.repl.run

## Vetor aleatório com 5 elementos:
[24, 29, 16, 10, 23]

## Vetor ordenado com 7 elementos:
[5, 7, 69, 138, 181, 211, 249]

## Vetor decrescente com 11 elementos:
[532, 454, 435, 354, 295, 262, 173, 160, 119, 75, 46]
```

Crie um arquivo chamado **exemplo2.py** com o seguinte conteúdo:

```
exemplo2.py
1  from lab5 import *
2
3  vetor = gerar_aleatorio(10)
4
5  print('\n## Vetor gerado:', vetor)
6
7  contagem = bubbleSort(vetor)
8
9  print('\n## Vetor ordenado:', vetor)
10
11 print('\n## Contador:', contagem)
12
```

Exemplo 2

Altere o arquivo **main.py** e execute o **exemplo2**.



The screenshot shows a web-based code editor interface. At the top, there is a 'Run' button with a green play icon and a 'Share' button with a user icon. Below the buttons, the editor displays two files: 'main.py' and 'exemplo2.py'. The 'main.py' file is open and shows the following code:

```
1  import exemplo2
2
3
```

To the right of the code editor, there is a terminal window showing the output of the program. The terminal URL is <https://ap2-lab5.tmsski.repl.run>. The output is as follows:

```
## Vetor gerado: [98, 23, 24, 63, 92, 60, 62, 61, 36, 34]
## Vetor ordenado: [23, 24, 34, 36, 60, 61, 62, 63, 92, 98]
## Contador: 45
> 
```

Exercício 1

Altere o código do arquivo **exemplo2.py** de forma a preencher a tabela com a contagem de execuções do laço da função **bubbleSort** para cada um dos casos (veja que já preenchi o valor correspondente à execução do slide anterior).

número de elementos	vetor inicial com elementos aleatórios	vetor inicial com elementos ordenados de forma crescente	vetor inicial com elementos ordenados de forma decrescente
10	45	45	45
20	190	190	190
100	4950	4950	4950
200	19900	19900	19900
1000	499500	499500	499500
2000	1999000	1999000	1999000
10000	49995000	49995000	49995000

Exercício 2

- a. Acrescente no arquivo **lab5.py** uma função que implemente o algoritmo de **ordenação por seleção**, de acordo com o pseudo-código estudado na aula de teoria:
- b. Seguindo o exemplo das alterações feitas na função **bubbleSort** para incluir o contador (slide 9), altere sua função de **ordenação por seleção** para que também retorne a contagem do número de execuções do laço mais interno.

```
para i crescendo de 0 até n-2 faça
    pivo <- i
    para j crescendo de i+1 até n-1 faça
        menor = i
        se (array[j] < array[menor]) entao
            menor = j
    temp = array[pivo]
    array[pivo] = array[menor]
    array[menor] = temp
```

Exercício 2 (continuação)

- c. Desenvolva um programa para conseguir preencher a tabela com a contagem de execuções do laço da função **de ordenação por seleção** para cada um dos casos.

número de elementos	vetor inicial com elementos aleatórios	vetor inicial com elementos ordenados de forma crescente	vetor inicial com elementos ordenados de forma decrescente
10	45	45	45
20	190	190	190
100	4950	4950	4950
200	19900	19900	19900
1000	499500	499500	499500
2000	1999000	1999000	1999000
10000	49995000	49995000	49995000

Exercício 3

- a. Acrescente no arquivo **lab5.py** uma função que implemente o algoritmo de **ordenação por inserção**, de acordo com o pseudo-código estudado na aula de teoria:
- b. Seguindo o exemplo das alterações feitas na função **bubbleSort** para incluir o contador (slide 9), altere sua função de **ordenação por inserção** para que também retorne a contagem do número de execuções do laço mais interno.

Ordenação-Por-Inserção (A, n)

```
1  para j crescendo de 1 até n-1 faça
2      x <- A[j]
3      i <- j-1
4      enquanto i >= 0 e A[i] > x faça
5          A[i+1] <- A[i]
6          i <- i-1
7      A[i+1] <- x
```

Exercício 3 (continuação)

- c. Desenvolva um programa para conseguir preencher a tabela com a contagem de execuções do laço da função **de ordenação por inserção** para cada um dos casos.

número de elementos	vetor inicial com elementos aleatórios	vetor inicial com elementos ordenados de forma crescente	vetor inicial com elementos ordenados de forma decrescente
10	9	9	9
20	19	19	19
100	99	99	99
200	199	199	199
1000	999	999	999
2000	1999	1999	1999
10000	9999	9999	9999

Exercício 4

Vamos criar gráficos com os dados obtidos. Suponha que, para um determinado algoritmo XYZ tenhamos obtidos os seguintes dados:

número de elementos	vetor inicial com elementos aleatórios	vetor inicial com elementos ordenados de forma crescente	vetor inicial com elementos ordenados de forma decrescente
10	20	30	1
20	40	45	4
100	200	203	10
200	400	190	40
1000	2000	900	1000
2000	4000	1500	3000
10000	20000	9000	100000

Exercício 4

Vamos criar gráficos com os dados obtidos. Suponha que, para um determinado algoritmo XYZ tenhamos obtidos os seguintes dados:

número de elementos	vetor inicial com elementos aleatórios	vetor inicial com elementos ordenados de forma crescente	vetor inicial com elementos ordenados de forma decrescente
10	20	30	1
20	40	45	4
100	200	203	10
200	400	190	40
1000	2000	900	1000
2000	4000	1500	3000
10000	20000	9000	100000

Crie um arquivo chamado **exemplo3.py** com o seguinte conteúdo:

```
exemplo3.py
1 import matplotlib.pyplot as plt
2 num_elementos = [10, 20, 100, 200, 1000, 2000, 10000]
3 aleatorio = [20, 40, 200, 400, 2000, 4000, 20000]
4 crescente = [30, 45, 203, 190, 900, 1500, 9000]
5 decrescente = [1, 4, 10, 40, 1000, 3000, 100000]
6 plt.plot(num_elementos, aleatorio, 'r')
7 plt.plot(num_elementos, crescente, 'b')
8 plt.plot(num_elementos, decrescente, 'g')
9 plt.xlabel("número de elementos")
10 plt.ylabel("contador")
11 plt.show()
12
```

Altere o arquivo **main.py** e execute o **exemplo3**.

Exercício 4

Vamos criar gráficos com os dados obtidos. Suponha que, para um determinado algoritmo XYZ tenhamos obtidos os seguintes dados:

número de elementos	vetor inicial com elementos aleatórios	vetor inicial com elementos ordenados de forma crescente	vetor inicial com elementos ordenados de forma decrescente
10	20	30	1
20	40	45	4
100	200	203	10
200	400	190	40
1000	2000	900	1000
2000	4000	1500	3000
10000	20000	9000	100000

Crie um arquivo chamado **exemplo3.py** com o seguinte conteúdo:

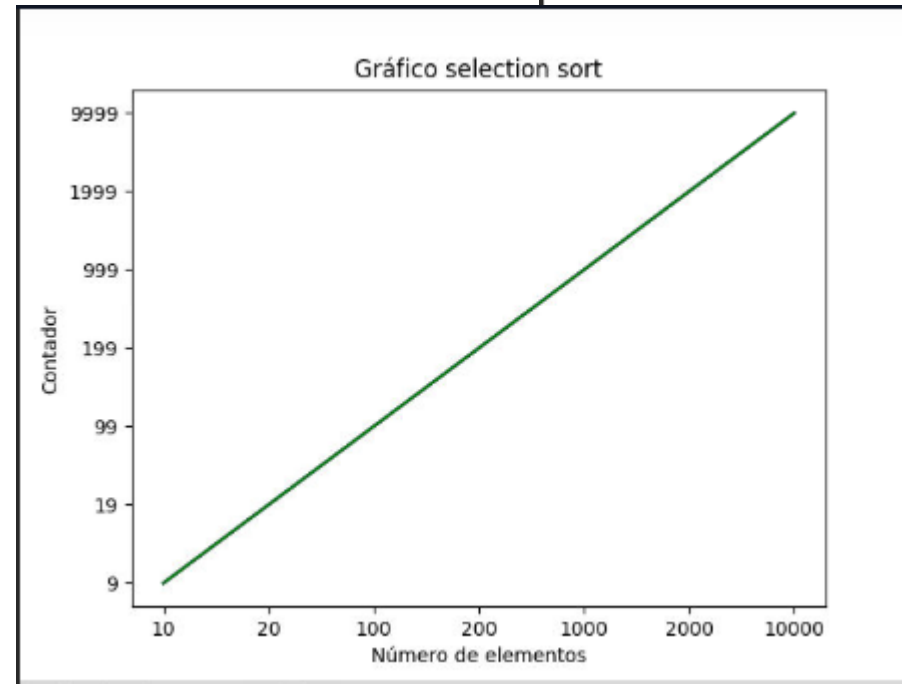
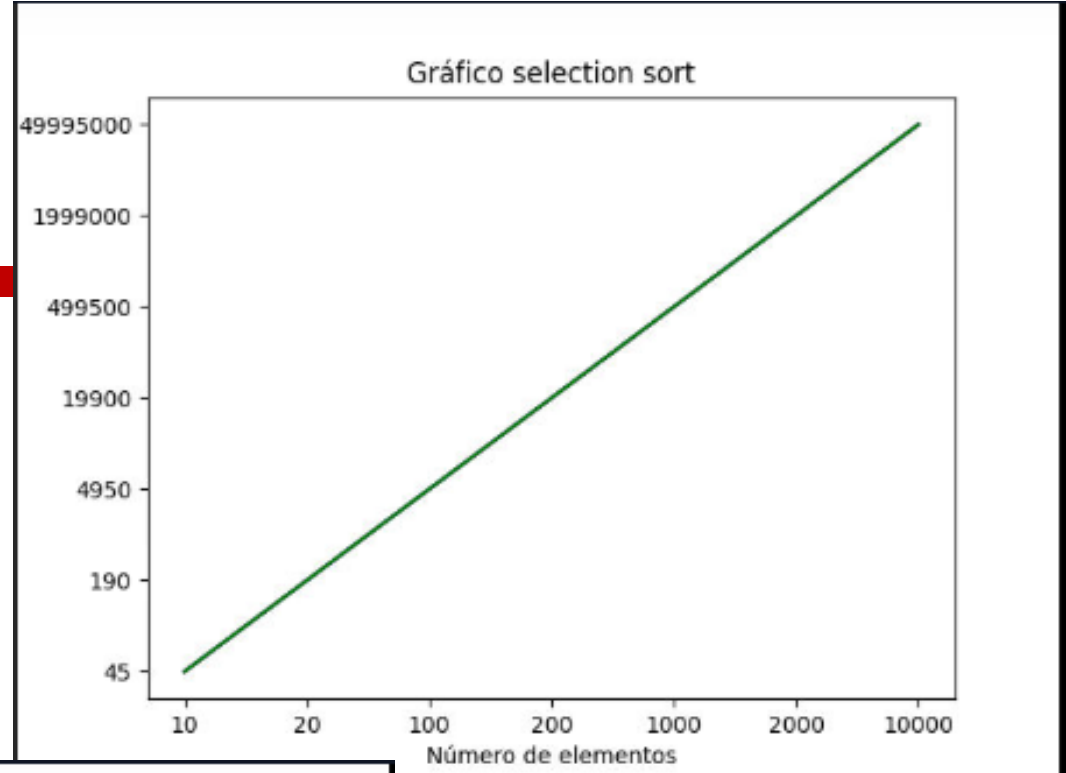
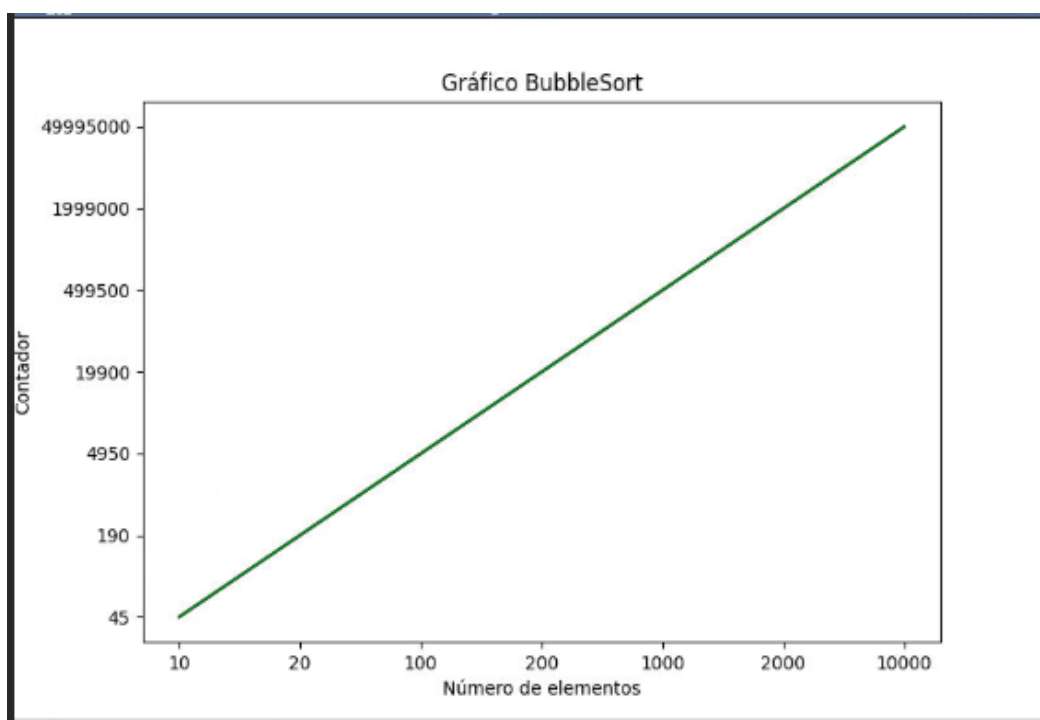
```
exemplo3.py
1 import matplotlib.pyplot as plt
2 num_elementos = [10, 20, 100, 200, 1000, 2000, 10000]
3 aleatorio = [20, 40, 200, 400, 2000, 4000, 20000]
4 crescente = [30, 45, 203, 190, 900, 1500, 9000]
5 decrescente = [1, 4, 10, 40, 1000, 3000, 100000]
6 plt.plot(num_elementos, aleatorio, 'r')
7 plt.plot(num_elementos, crescente, 'b')
8 plt.plot(num_elementos, decrescente, 'g')
9 plt.xlabel("número de elementos")
10 plt.ylabel("contador")
11 plt.show()
12
```

Altere o arquivo **main.py** e execute o **exemplo3**.

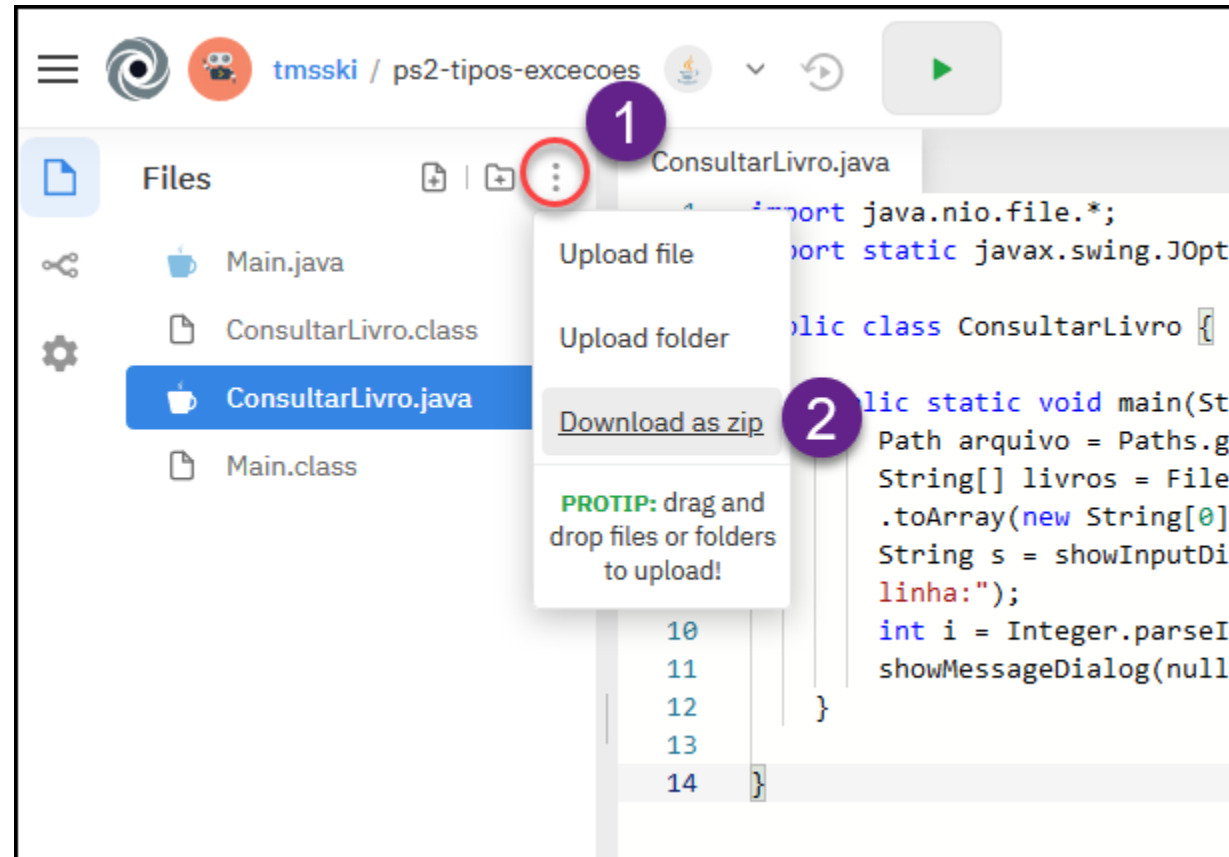
Exercício 4 (continuação)

Seguindo o exemplo dado no arquivo **exemplo3.py**, gere:

- o gráfico com as contagens para o algoritmo de *bubble sort*;
- o gráfico com as contagens para o algoritmo de *selection sort*;
- o gráfico com as contagens para o algoritmo de *insertion sort*.



Quando terminar todos os exercícios, siga os passos abaixo para baixar todo o conteúdo em um arquivo compactado.



Faça a entrega também das tabelas com os dados das suas execuções (pode ser em um arquivo Excel).

