

Para facilitar o entendimento das instruções desta lista de exercícios, vou supor que estará sendo implementada pela aluna hipotética [Ana Julia de Paiva Lopes](#), realizando a entrega postada na cidade de [Dourados](#) no dia [05/12/22](#).

Essa data é hipotética somente para sinalizar que o aluno não precisa esperar a data final ([12/12/22](#)) para enviar a sua entrega. Quanto mais cedo enviar, mais chances terá de realizar correções até a data final de entrega, pois será avaliada somente a última entrega realizada até a data final.

No PyCharm crie o projeto : [LPI-LE](#). No diretório do projeto:

- crie os diretórios [src](#) e [dados](#)
- mova o arquivo [main.py](#) para dentro do diretório [src](#)

No módulo [main](#) defina as seguinte funções:

```
def exercicio4():  
    pass  
  
def exercicio3():  
    pass  
  
def exercicio2():  
    pass  
  
def exercicio1():  
    pass
```

A palavra reserva [pass](#) indica que o corpo interno da função será substituído posteriormente por um código, evitando que o PyCharm acuse erro enquanto você não implementa o código definitivo.

No corpo do projeto do módulo [main](#) defina:

```
if __name__ == '__main__':  
    print('Lista de Exercícios: Ana Julia de Paiva Lopes') # informar nome completo do aluno  
    exercicio1()  
    exercicio2()  
    exercicio3()  
    exercicio4()
```

Os enunciados dos quatro exercícios serão apresentados após as orientações de como empacotar a entrega para envio.

Ao concluir a implementação da lista de exercícios você deverá empacotar a sua entrega. Crie um diretório de acordo com o seguinte exemplo: [LPI-LE - Ana Julia de Paiva Lopes](#). O nome do aluno/a [não](#) deve utilizar somente letras maiúsculas; siga o exemplo acima.

No diretório criado, copie os subdiretórios [src](#) e [dados](#) do seu projeto, contendo os arquivos internos a esses diretórios, e acrescente os arquivos: [fontes.pdf](#) e [saída.pdf](#).

A primeira linha (cabeçalho) do arquivo [fontes.pdf](#) deverá conter a seguinte informação:

Linguagem de Programação I – Lista de Exercícios – Ana Julia de Paiva Lopes

O corpo do arquivo [fontes.pdf](#) deverá conter os códigos de todos os módulos implementados na sua lista de exercícios, a começar pelo módulo [main](#). Escreva **Módulo: main** e copie o código deste módulo, pule uma linha e proceda de forma semelhante com os demais módulos.

A última linha do arquivo [fontes.pdf](#) deverá conter a seguinte informação:

<Cidade>, <data> <assinatura>

A assinatura deverá ser gerada pelo Adobe Acrobat Reader. Esse software disponibiliza um ícone em formato de pena de caneta tinteiro para você utilizar a funcionalidade de gerar a sua assinatura utilizando o mouse. Para as entregas da disciplina você deverá gerar uma assinatura com o seu **nome completo em letra cursiva**. Após gerar a assinatura, ela permanecerá armazenada no Adobe Acrobat Reader, de forma que você utilize sempre a mesma assinatura em todos os arquivos [pdf](#) enviados nas avaliações desta disciplina.

O arquivo [saída.pdf](#) deverá conter o mesmo cabeçalho e linha final, utilizados no arquivo [fontes.pdf](#), com o seguinte corpo: print (tela completa do PyCharm, minimizando as demais telas, para restar somente a tela de saída) de quantas telas forem necessárias para mostrar a saída gerada pelo execução da projeto da sua lista de exercícios.

Após concluir todo o conteúdo do arquivo [LPI-LE – Ana Julia de Paiva Lopes](#), gere a versão [zip](#) deste arquivo e envie para o email joinvile@ufgd.edu.br.

A entrega será considerada nula, se não cumprir **qualquer uma** das seguintes condições:

- entrega até **12/12/22**, data final determinada no Plano de Ensino da disciplina;
- o arquivo enviado
 - não estiver com o nome correto
 - não contiver os diretórios [src](#) e [dados](#) do projeto
 - não contiver os arquivos [fontes.pdf](#) e [saída.pdf](#)
 - houver divergência entre os conteúdos dos fontes do diretório [src](#) e do arquivo [fontes.pdf](#)
- se os arquivos pdf
 - não tiverem o cabeçalho
 - [Linguagem de Programação I – Lista de Exercícios – <nome completo do aluno/a>](#)
 - não tiverem a última linha, como por exemplo:
 - [Dourados, dia 05 de dezembro de 2022 <assinatura>](#)
- se a assinatura não contiver: **nome completo do aluno em letra cursiva**
- se a assinatura não for a mesma nos dois arquivos pdf
 - deverá ser a mesma em todas as entregas desta disciplina

Observe a pontuação de cada exercícios da lista. Serão pontuados **somente** os exercícios que executarem corretamente.

Exercício 1 (2,0 pontos)

No diretório `src` crie o módulo `email_acadêmico`. No módulo `email_acadêmico`, implemente a função descrita a seguir.

`criar_emails_acadêmicos`

- parâmetros
 - `alunos` : lista com nomes de alunos
- funcionalidade
 - para cada `nome` na lista `alunos`
 - utilize a função `lower` para converter o `nome` para letras minúsculas
 - utilize a função `split` para obter uma lista com as palavras do nome do aluno
 - gere o `email_acadêmico`
 - concatenando todas as palavras ligando-as pela caracter ponto
 - ignorando as palavras de ligação: `de`, `da`, `das`, `do`, `dos`
 - apendando o string: `@academico.ufgd.edu.br`
 - armazenando a tupla (`nome`, `email_acadêmico`) na lista `emails_acadêmicos`
- retorno
 - `emails_acadêmicos`

Na função `exercício1`, do módulo `main`, implemente:

- pule uma linha e imprima: `Exercício 1`
- crie uma lista com nomes de cinco alunos
 - Silvia Lemos da Silva, Fernando Tavares de Almeida, Rafael Souza Junior
Sandra Maria dos Santos, Pedro Valente Neto
- utilize a função `criar_emails_acadêmicos` para criar uma lista de tuplas com nomes e emails acadêmicos dos alunos
- itere na lista e imprima nome e email acadêmico para cada aluno

Exercício 2 (2,0 pontos)

No diretório `src` crie o módulo `data`. No módulo `data`, implemente a função `comparar_datas(data1, data2)`.

A função `comparar_datas` recebe duas tuplas como argumento (cada tuplas com 3 valores inteiros para representar: dia, mês e ano) e deve retornar:

- 1: se `data1` é maior que a `data2`
- 0: se `data1` é igual à `data2`
- -1: se `data1` é menor que a `data2`

No módulo `main`, implemente a função `informar_comparação(data1, data2)`. Esta função deve executar a função `comparar_datas` com as datas recebidas como argumentos e imprimir:

- data 1
- data 2
- imprimir
 - resultado da comparação: a data `<data1>` é maior/igual/menor que a data `<data2>`

Na função `exercício2`, do módulo `main`, implemente:

- pule uma linha e imprima: `Exercício 2`
- execute
 - `informar_comparação((31,7,2021), (31,5,2021))`
 - `informar_comparação((31,5,2021), (31,5,2021))`
 - `informar_comparação((31,5,2021), (31,5,2022))`

Exercício 3 (3,0 pontos)

diagonal da matriz (em azul) -- matriz triangular superior -- matriz triangular inferior

1 7 3 7 2	1 7 3 7 2	1 0 0 0 0
9 2 4 1 6	0 2 4 1 6	9 2 0 0 0
4 8 2 3 5	0 0 2 3 5	4 8 2 0 0
8 3 5 1 7	0 0 0 1 7	8 3 5 1 0
4 6 1 5 9	0 0 0 0 9	4 6 1 5 9

No diretório `src` crie o módulo `matriz`. No módulo `matriz`, implemente as funções descritas a seguir.

`gerar_diagonal_matriz`

- parâmetros
 - `matriz_quadrada`
- funcionalidade
 - obter a `dimensão` da matriz
 - matriz quadrada tem o mesmo número de linhas e de colunas
 - preencha a lista `diagonal` com os valores da matriz correspondentes à diagonal da matriz (valores da matriz com os mesmos índices na linha e na coluna)
 - varie os índices da matriz de 0 até `dimensão - 1` para obter os índices de linha ou coluna correspondentes aos índices da diagonal das matriz
- retorno
 - `diagonal`

`gerar_matriz_triangular`

- parâmetros
 - `matriz_quadrada`
 - `tipo_matriz_triangular` : superior ou inferior
- funcionalidade
 - itere nas linhas `matriz_quadrada`
 - itere nas colunas da `matriz_quadrada`
 - copie para `matriz_triangular` os valores da linha da matriz
 - se `tipo_matriz_triangular` : superior
 - substitua por 0 os valores com índices das colunas inferiores ao índice da coluna da diagonal
 - se `tipo_matriz_triangular` : inferior
 - substitua por 0 os valores com índices das colunas superiores ao índice da coluna da diagonal
- retorno
 - `matriz_triangular`

Na função `exercício3`, do módulo `main`, implemente:

- pule uma linha e imprima: `Exercício 3`
- inicialize `matriz_quadrada` com os dados da matriz ilustrada acima
- utilize a função `gerar_diagonal_matriz` para imprimir a diagonal da `matriz_quadrada`
- utilize a função `gerar_matriz_triangular` para imprimir a matriz triangular superior e a matriz triangular inferior da `matriz_quadrada`

Exercício 4 (3,0 pontos)

No diretório `src` crie o módulo `seguro`. No módulo `seguro`, implemente as funções descritas a seguir.

`obter_população_cidade`

- parâmetro
 - `cidade_segurado`
- funcionalidade
 - no diretório dados crie manualmente o arquivo `PopulaçõesCidadesBrasileiras.csv` com cidades e populações de algumas cidades brasileiras
 - cidade, população
 - Belo Horizonte, 2521564
 - Campinas, 1213792
 - Campo Grande, 906092
 - Curitiba, 1948626
 - Dourados, 225495
 - Florianópolis, 508826
 - Fortaleza, 2686612
 - Piracicaba, 407252
 - Ponta Porã, 93937
 - Porto Alegre, 1488252
 - Rio de Janeiro, 6747815
 - Salvador, 2886698
 - São Paulo, 12325232
 - utilize a função `carregar_arquivo_csv`, do módulo `util`, para carregar o arquivo `PopulaçõesCidadesBrasileiras` na lista `populações_cidades_brasileiras`
 - itere na lista `populações_cidades_brasileiras`
 - para obter a `população` da `cidade_segurado`
- retorno
 - `população`

calcular_risco_seguro

- parâmetros
 - idade, tempo_habilitação, tipo_residência, população_cidade
- funcionalidade
 - calcular **risco_seguro** acrescentando as seguintes pontuações
 - se **idade**
 - entre 18 e 21 : 3 pontos
 - entre 22 e 26 : 1 ponto
 - entre 80 e 90 : 2 pontos
 - acima de 90 : 4 pontos
 - se **tempo_habilitação**
 - menor que 1 ano : 2 pontos
 - se **população_cidade**
 - menor que 100 mil
 - se **tipo_residência**
 - casa : 1 ponto
 - de 100 mil a 400 mil
 - se **tipo_residência**
 - casa : 2 pontos
 - apto : 1 ponto
 - mais de 400 mil a 1 milhão
 - se **tipo_residência**
 - casa : 3 pontos
 - apartamento : 2 pontos
 - mais de 1 milhão a 3 milhões
 - se **tipo_residência**
 - casa : 5 pontos
 - apartamento : 2 pontos
 - condomínio fechado : 1 ponto
 - acima de 3 milhões
 - se **tipo_residência**
 - casa : 7 pontos
 - apartamento : 3 pontos
 - condomínio fechado : 2 ponto
 - calcular **categoria_risco** da seguinte forma
 - se **risco_seguro**
 - menor que 5 : **categoria_risco** é baixa
 - para risco entre 5 e 10 : **categoria_risco** é média
 - maior que 10 : **categoria_risco** é alta
- retornar : **risco_seguro, categoria_risco**

`imprimir_risco_segurados(segurados)`

- parâmetros
 - `nome`, `idade`, `tempo_habilitação`, `tipo_residência`, `cidade`
- funcionalidade
 - obter `população_cidade`
 - imprimir linha 1 e linha 2 da seguinte forma
 - linha 1
 - `<nome>` : com `<idade>` anos de idade, `<tempo_habilitação>` anos de habilitação, reside em `<tipo_residência>`, na cidade de `<cidade>` com cerca de `<população_cidade>` habitantes
 - linha 2:
 - -- risco de seguro : `<risco_seguro>` -- categoria do seguro : `<categoria_seguro>`
 - para calcular `risco_seguro` e `categoria_seguro` : utilizar a função `calcular_risco_seguro`

Na função `exercício4` do módulo `main`

- pule uma linha e imprima: `Exercício 4`
- crie o dicionário `segurados` com chave `nome` composto de 6 dicionários de segurados com as seguintes chaves e valores
 - `nome`, `idade`, `tempo_habilitação`, `tipo_residência`, `cidade`
 - Marina Tempira, 22, 4, casa, Ponta Porã
 - Leonardo Talure, 35, 17, apartamento, Belo Horizonte
 - Adriana Raski, 18, 0, condomínio, São Paulo
 - Fabrício Salvi, 85, 5, apartamento, Florianópolis
 - Alexia Caltaro, 87, 0, casa, Salvador
 - Tales Petrus, 91, 70, casa, São Paulo
- utilize as funções `calcular_risco_seguro` e `imprimir_risco_segurados` para imprimir os dados dos riscos de seguros do dicionário `segurados`