LPI - Tutorial 1 : Conceitos Básicos

1 - A popularidade do Paradigma Orientado a Objetos

Duas características principais são responsáveis pela enorme popularidade do paradigma orientado a objetos: (a) o mapeamento de entidades do mundo real em unidades de programação, que são utilizadas como novos tipos da linguagem; e (b) e capacidade de definir unidades de programação com dados e métodos (funções que acessam esses dados) comuns em unidades mais genéricas e herdá-los em outras unidades mais especilizadas.

Python é uma linguagem orientada a objetos cuja utilização tem crescido muito atualmente. Suas principais características são: independência de plataforma, gerenciamento automático da utilização de mémoria e transparência no endereçamento de memória. Python provê uma plataforma de desenvolvimento com uma extensa API (Application Program Interface), que suporta diversas funcionalidades. Várias outras linguagens suportam o paradigma orientado a objetos, como por exemplo: Java, C++, PHP, JavaScript, Python, Pearl, Delphi, C#, Visual Basic. Observe que dominar o paradigma orientado a objetos lhe será muito útil no aprendizado de outras linguagens.

A manipulação de conjuntos de dados em Python é muito poderosa, favorecendo a criação de um código compacto e versátil, o que tem contribuido para grande popularidade atual desta linguagem. Além disso, esta linguagem está sendo largamente utilizada para o desenvolvimento de software científico, como por exemplo, aprendizado de máquinas (*machine learning*) e aprendizado profundo (*deep learning*) que estão potencializando aplicações muito bem sucedidas na área de inteligência artificial.

Na disciplina Linguagem de Programação I (LPI), você aprenderá os conceitos básicos da linguagem Python (Tutorial 1) e posteriormente desenvolverá uma aplicação orientada a objetos de forma incremental (Tutoriais 2 a 5).

No Tutorial 1 vamos abordar: (a) a instalação do ambiente que irá utilizar no desenvolvimento de suas aplicações; e (b) conceitos básicos da linguagem Python. Na seção 2, você aprenderá a instalar o ambiente necessário para executar aplicações na linguagem Python, e executará um programa extremamente simples para validar o funcionamento do ambiente instalado. Na seção 3, você aprenderá a utilizar: variáveis, tipos, funções e comandos condicionais. Na seção 4, você aprenderá a utilizar estruturas de dados e comandos de iteração. Na seção 5, você aprenderá a ler e salvar dados de arquivos CSV e JSON que são utilizados em muitas aplicações.

No Tutorial 2 vamos exercitar o paradigma orientado a objetos, definindo seus principais conceitos e ilustrando suas aplicações na linguagem Python. Vamos exercitar o mapeamento de uma entidade do mundo real na definição de um novo tipo para sua No jargão da orientação a objetos esse novo tipo é denominado classe de objetos, que por simplicidade passaremos a chamar de classe. Na classe podemos agrupar dados e métodos. Poderemos criar variáveis, denominadas objetos, a partir do novo tipo representado por uma classe. Na aplicação do Tutorial 2, vamos cadastrar, imprimir e filtrar várias instâncias de objetos criados a partir da classe, mapeada a partir de uma entidade do mundo real.

No Tutorial 3, ficará mais evidente a praticidade do mapeamento de entidades em novos tipos da linguagem. Vamos, então desenvolver uma aplicação mapeando várias entidades do mundo real em novos tipos para compor a nossa aplicação.

No Tutorial 4 vamos conceituar e exercitar o agrupamento de dados e métodos comuns em uma superclasse (mais genérica) e a herança dos mesmos em subclasses (mais específicas), nas quais poderemos acrescentar outros dados e métodos.

No Tutorial 5, os valores dos dados dos objetos, que nos tutoriais anteriores foram criados diretamente por programa, passarão a ser lidos por uma interface textual. Adicionalmente os objetos criados serão persistidos em arquivos ao final da execução, e recuperados no ínicio de uma nova execução, de forma que os objetos criados em execuções anteriores não sejam perdidos.

Os cinco tutoriais foram sendo escritos em uma linguagem mais informal, para tornar a leitura mais amigável. Leia atentamente todas as seções de cada tutorial, na sequência proposta sem pular nenhuma seção, e execute o código que será apresentado, sempre que a execução do mesmo for sugerida ao longo do texto. Os exercícios são opcionais, mas ajudam a fixar os conceitos aprendidos.

Para facilitar a leitura quando um parágrafo ou trecho de código não couber no final da página, será transferido para a próxima página, portanto, não estranhe espaços em branco ao final de uma página.

2 - Instalação do Python e do IDE PyCharm

Para fixar os conceitos que serão apresentados neste tutorial, é fundamental que cada trecho de código apresentado seja implementado e testado. Desta forma, será necessário instalar:

- a versão estável mais recente do Python;
- e a versão mais recente do IDE (*Integrated Development Environment* - Ambiente de Desenvolvimento Integrado) PyCharm da Jet Brains.

No site https://www.python.org/, selecione a opção Downloads e a opção da versão mais recente do Python para Windows (atualmente: Python 3.10.2), baixe e instale no seu computador o arquivo executável da versão mais recente do Python (por exemplo: python-3.10.2-amd64).

No site https://www.jetbrains.com/pt-br/pycharm/ selecione a opção Baixar e novamente esta opção para a baixar a versão gratuita Community do PyCharm, baixe e instale em seu computador o arquivo executável da versão mais recente do PyCharm Community (por exemplo: pycharm-community-2021.3.2).

Para implementar e testar os trechos de código das seções 3 e 4 deste tutorial, que ilustram conceitos básicos da linguagem Python, crie um projeto no IDE PyCharm com o nome LPI-E0:

- selecione File -- New Project
- na janela Create Project
  - mantenha a seleção de New environment using para criar um ambiente específico para o seu projeto
    - em Location
      - navegue para escolher um diretório para armazenar o seu projeto
      - complete o diretório com o nome do projeto : LPI-E1
    - em Basic Interpreter
      - selecione a versão Python 3.10
  - mantenha a seleção de Create a main.py welcome script
  - finalize a criação ativando o botão Create

Crie um diretório padrão para armazenar os diretórios e arquivos fontes do seu projeto:

- selecione a aba Project no canto esquerdo da tela para visualizar a janela lateral Project
- clique na linha com o nome do seu projeto e selecione, com o botão direito do mouse
  - New -- Directory
    - preencha com o nome src (abreviação para source)
      - diretório onde tradicionalmente são armazenados os arquivos fontes do projeto
        - em geral organizados em subdiretórios de src
- assinale o diretório src como raiz dos arquivos fontes do projeto
  - selecione o diretório src com o botão direito do mouse
  - escolha a opção: Mark Directory as -- Mark as Source Root
- selecione o arquivo main.py e mova-o para o diretório src

Ajuste o arquivo main.py:

- na janela de projeto (lateral esquerda)
  - clique no arquivo main.py para visualizar o seu código na janela à direita da janela de projeto
- remova a seleção de ponto de parada para teste passo a passo (debug)
  - clicando no círculo vermelho no lado esquerdo de uma das linhas do código
- remova todos os comentários : strings iniciados com o caracter #
- altere o código do arquivo main.py para ficar da seguinte forma:

```
def imprimir(linguagem_programação):
    print(f'Vamos aprender a programar na linguagem {linguagem_programação}.')

if __name__ == '__main__':
    imprimir('Python')
```

Execute o programa, clicando na seta verde no canto superior direito da sua janela:

Você visualizará o resultado do processo na janela Run (execução)

```
Vamos aprender a programar na linguagem Python.

Process finished with exit code 0
```

O código em Python é organizado em blocos de comandos. Diferentemente de outras linguagens (como: Java, C++ e C), bloco em Python não são delimitados por { }, e sim por identação (utilizando Tab). Portanto, você deve estar muito atento para que a identação de seu código esteja correta.

Na execução, o Python atribui à variável pré-definida __name__ o string '__main__', e executa o corpo principal do programa, que neste caso tem somente o comando condicional **if**. Variáveis pré_definidas são cercadas por duplo underline, como é o caso da variável __name__.

O comando condicional if testa com sucesso que a variável __name__ é igual ao (contém o mesmo valor que) string '__main__'. Pelo fato do teste do comando **if** ser bem sucedido, o seu corpo interno de comandos é executado: neste exemplo, somente a chamada da função imprimir, passando como argumento o string 'Python'.

A função imprimir recebe o parâmetro (entre parênteses) linguagem_programação, para o qual foi atribuído como argumento o string 'Python'. Ao ser chamada a função executa o seu corpo interno de comandos, que neste exemplo, é composto somente pela chamada da função pré-definida (*builtin*) print, utilizada para imprimir strings. Funções pré-definidas são disponíveis em Python sem que seja necessário defini-las.

O argumento passado para a função print é um string formatado (f'...') para compor trechos fixos ('Vamos aprender a programar na linguagem ' e '.') com trechos variáveis (entre chaves: { }). O trecho variável recebe o valor passado como argumento ('Python') para o parâmetro linguagem_programação. Como resultado é impresso na tela do seu computador, o string: Vamos aprender a programar na linguagem Python.

O resultado da execução é mostrado na janela de execução (*Run*). Adicionalmente, como mensagem de final de execução é impressa a mensagem: Process finished with exit code 0, cuja tradução é: Processo concluído com código de saída 0. O código 0 indica que a execução foi concluída com sucesso. Caso seja reportado o código 1, a mensagem indica execução mal sucedida. Esta mensagem se repete em todas as execuções e será omitida nas ilustrações da seção 2.

Este exemplo muito simples, com uma breve explicação, é o seu primeiro contato com a execução de um programa em Python. Na próxima seção, vamos conceituar e ilustrar variáveis, tipos, funções e comandos condicionais.

3 - Variáveis, Tipos, Funções e Comandos Condicionais

Nesta seção vamos ilustrar vários conceitos de forma incremental. Para facilitar o entendimento dos conceitos, em cada subseção serão ilustrados somente a função e o trecho de código do programa principal, tratados na respectiva subseção.

O código completo do programa é formado pelo código de todas as funções e pelo corpo do programa definido com base na agregação de todos os trechos ilustrados nas subseções. Portanto, ao estudar cada subseção acrescente ao código do seu programa, os códigos parciais apresentados na seção que você está estudando. Ao concluir o estudo das quatro subseções, você terá o programa completo da seção 3.

Variáveis são utilizadas para armazenar valores que são utilizados para serem processados por funções. Comando condicionais são utilizados para executar blocos internos de comandos quando uma determinada condição de execução é verdadeira.

3.1 - Definição de variáveis de vários tipos e impressão por chamada de função

Um módulo é um arquivo, com extensão py, que contém código Python: funções, classes (veremos na seção 5) e variáveis.

Funções utilizadas nesta seção serão definidas em um módulo Python separado. Para criar esse módulo:

- selecione o diretório src dos fontes do seu projeto
- com o botão direito do mouse selecione as operações: New -- Python File
- no campo Name informe o nome do arquivo (módulo):
  - variáveis__tipos__funções__comandos_condicionais

No módulo variáveis__tipos__funções__comandos_condicionais defina uma função com o seguinte código:

```
def imprimir_variável(nome_variável, variável):
    print(f'variável {nome_variável} do tipo {type(variável)} com o valor : {variável}')
```

No módulo main defina o seguinte código:

```
from variáveis__tipos__funções__comandos_condicionais import imprimir_variável

def ilustrar__variáveis__tipos__funções__comandos_condicionais():
    print('\n3.1 - definição de variáveis de vários tipos e impressão por chamada de função')
    disciplina = 'Linguagem de Programação I'
    carga_horária = 72
    nota_mínima_aprovação = 6.0
    é_disciplina_obrigatória = True
    imprimir_variável('disciplina', disciplina)
    imprimir_variável('carga_horária', carga_horária)
    imprimir_variável('nota_mínima_aprovação', nota_mínima_aprovação)
    imprimir_variável('é_disciplina_obrigatória', é_disciplina_obrigatória)


if __name__ == '__main__':
    ilustrar__variáveis__tipos__funções__comandos_condicionais()
```

Para utilizar uma função definida em outro módulo você precisa importar essa função do seu módulo de origem. As palavras reservadas from e import são utilizadas para importar a função imprimir_variável do módulo variáveis__tipos__funções__comandos_condicionais.

O corpo do programa principal (bloco interno ao comando: if __name__ == '__main__':) é composto pela chamada da função ilustrar__variáveis__tipos__funções__comandos_condicionais.

Execute o programa. Você verá o seguinte resultado:

```
3.1 - definição de variáveis de vários tipos e impressão por chamada de função
variável disciplina do tipo <class 'str'> com o valor : Linguagem de Programação I
variável carga_horária do tipo <class 'int'> com o valor : 72
variável nota_mínima_aprovação do tipo <class 'float'> com o valor : 6.0
variável é_disciplina_obrigatória do tipo <class 'bool'> com o valor : True
```

Em todas as ilustrações que serão apresentadas neste tutorial, a primeira linha será utilizada para imprimir (utilizando a função print) o número da subseção sendo ilustrada (neste caso: 3.1) e o seu conteúdo. Os caracteres \n, utilizados no string a ser impresso, indicam que deve ser pulada uma linha antes de realizar a impressão do string.

Vamos descrever o código da subseção 3.1. Como já vimos no final da seção 1, quando o programa é executado, ocorre a execução do corpo do programa principal, que neste exemplo, é somente a chamada da função ilustrar__variáveis__tipos__funções__comandos_condicionais.

Na função ilustrar__variáveis__tipos__funções__comandos_condicionais são ilustrados:

- comandos de atribuição, no quais quatro variáveis são criadas e inicializadas com valores de diferentes tipos;
- chamadas da função imprimir_variável, para as quatro variáveis criadas anteriormente.

Variáveis são utilizadas para armazenar dados (ex: nota_mínima_aprovação = 6.0) que serão utilizados na execução do seu programa. O nome de uma variável pode ser composto por uma única palavra (ex: disciplina) com letras minúsculas ou por várias palavras interligadas por *underline* (ex: carga_horária).

Python é uma linguagem com tipagem dinâmica, o que significa que você não especifica o tipo de dado sendo definido. Esse tipo será inferido a partir dos valores atribuídos a esse objeto. Uma variável pode ser inicializada com um valor de um determinado tipo. Esse valor poderá ser alterado durante a execução do programa. Por enquanto, vamos considerar que os valores atribuídos a uma variável podem ser dos seguintes tipos:

- str : string --- ex: disciplina = 'Linguagem de Programação I';
- int : inteiro positivo ou negativo --- ex: carga_horária = 72;
- float : ponto flutuante positivo ou negativo --- ex: nota_mínima_aprovação = 6.0;
- bool : boolean com valores True ou False --- ex: é_disciplina_obrigatória = True.

Função tem um nome, a partir do qual é chamada, e uma lista de parâmetros, para os quais são atribuídos uma lista de argumentos na chamada da função. Apesar do nome de uma função ser formado por palavras separadas por *underline* (ou por uma única palavra), da mesma forma que uma variável, é recomendável que a primeira palavra seja um verbo no infinitivo para designar a ação que será realizada na execução da função (ex: imprimir_variável) ou um substantivo designando uma função sendo realizada (ex: fatorial).

Quando o nome da função por formado por nomes compostos, fica mais legível separar o nome composto (ou os nomes compostos, se for o caso) por dois caracteres *underline*, como por exemplo na função ilustrar__variáveis__tipos__funções__comandos_condicionais. Neste caso, as partes do nome da função (ilustrar, variáveis, tipos, funções, comandos_condicionais) foram separadas por dois caracteres *underlines*, para diferenciar da separação das palavras do nome composto comandos_condicionais, separadas por um único caracter *underline*.

Parâmetros são variáveis cujo escopo é o bloco interno de uma função. O escopo é o intervalo de linhas do programa na qual a variável é válida e acessível. No caso de uma função os parâmetros são válidos somente para utilização nos comandos do corpo interno da função. O parâmetro é nomeado da mesma forma que uma variável (ex: nome_variável) . Na chamada da função é passado um argumento para cada um dos parâmetros da função.

O argumento é um valor, uma variável, ou uma chamada de função. No código atual são passados: um valor e uma variável. Tomando como exemplo a chamada imprimir_variável('carga_horária', carga_horária):

- o primeiro argumento passado é o valor 'carga_horária' do tipo str, que como consequência da chamada de função é atribuído ao parâmetro nome_variável da função imprimir_variável;
- e o segundo argumento passado é a variável carga_horária, cujo valor do tipo int atribuido à variável previamente (carga_horária = 72), é atribuído ao parâmetro variável da função imprimir_variável.

A definição de uma função inicia com a palavra reservada def e é separada de seu corpo interno de comandos (indentados com Tab) pelo caracter **:**. Uma palavra reservada faz parte da linguagem e não pode ser utilizada como nome de variável. Neste tutorial, para chamar atenção, as palavras reservadas da linguagem Python são ilustradas em negrito.

A sua lista de parâmetros é cercada por parênteses. Esta lista pode ter vários, apenas um ou nenhum parâmetro. Mesmo que a lista seja vazia, os parênteses precisam ser representados na definição e na chamada da função.

O corpo de comandos da função imprimir_variável é composto somente pela chamada da função print, que imprime um string formatado com f'...', concatenando strings fixos e strings gerados a partir da utilização de variáveis (ex: nome_variável) ou de funções (ex: type(variável)), encapsulados por chaves ({ }). A função type é utilizada para retornar o tipo de uma variável.

Um variável definida no módulo é uma variável que poder ser utilizada por qualquer função do módulo. No entanto, para alterar o valor dessa variável, no contexto de uma função, precisamos declará-la como global na função; caso contrário, a atribuição será feita com uma variável local, com o mesmo nome da variável do módulo e, neste caso, o valor da variável do módulo não será alterado. O uso de atribuição à variáveis de módulo em funções, será ilustrado no Tutorial 5; por ora veremos apenas um pequeno exemplo.

```
quantidade_acessos = 0

def realizar_novo_acesso():
    global quantidade_acessos
    quantidade_acessos += 1
```

3.2 - Utilizando comando condicional e operador + para concatenar strings

No módulo variáveis__tipos__funções__comandos_condicionais, acrescente a definição da função calcular_status_aprovação_aluno. Observe que as condições utilizadas nos comandos condicionais não são delimitadas por **( )**, como ocorre nas linguagens de programação linguagens C, C++ e Java; no entanto, são finalizadas pelo separador **:** .

```
def calcular_status_aprovação_aluno(média):
    if média >= 6.0: return 'aprovado'
    elif 4.0 <= média < 6.0: return 'de exame'
    else: return 'reprovado'
```

O comando condicional teste sempre a primeira condição (com **if**) e pode opcionalmente: (a) testar várias outras condições (com elif, que é uma abreviação para else if) e (b) executar instruções associadas à situação senão ou do contrário (else). Tanto as condições testadas, quanto a palavra reservada else, são finalizadas pelo caracter **:** .

Se bloco interno de comandos é composto por um comando que pode ser inserido após o delimitador **:** e ser representado na mesma linha, o usuário tem o opção de indentar esse comando em outra linha ou utilizar a mesma linha do separador **:** . Nos tutoriais desta disciplina, optei porque representar o único comando interno na mesma linha, quando sua representação completa couber nesta linha, como foi o caso, por exemplo, dos comando internos do método calcular_status_aprovação_aluno. No entanto, embora aumente desnecessariamente a quantidade de linhas para representar o mesmo código, o aluno tem a opção de representar os comando internos únicos através de indentação. Neste caso, o método calcular_status_aprovação_aluno seria representado da seguinte forma:

```
def calcular_status_aprovação_aluno(média):
    if média >= 6.0:
        return 'aprovado'
    elif 4.0 <= média < 6.0:
        return 'de exame'
    else:
        return 'reprovado'
```

Para ilustrar a chamada de uma função que utiliza um comando condicional, e a utilização do operador + para concatenar strings, comente o código do trecho de ilustração da seção 3.1 e acrescente o código de ilustração da seção 3.2 na definição da função de ilustração da seção 3:

```
def ilustrar_variáveis_tipos_funções_comandos_condicionais():
    # print('\n3.1 - definição de variáveis de vários tipos e impressão por chamada de função')
    # disciplina = 'Programação Aplicada à Engenharia'
    # carga_horária = 72
    # nota_mínima_aprovação = 6.0
    # é_disciplina_obrigatória = True
    # imprimir_variável('disciplina', disciplina)
    # imprimir_variável('carga_horária', carga_horária)
    # imprimir_variável('nota_mínima_aprovação', nota_mínima_aprovação)
    # imprimir_variável('é_disciplina_obrigatória', é_disciplina_obrigatória)
    print('\n3.2 - função com comando condicional e operador + para concatenar strings')
    nota_aluno = 5.5
    print('status do aluno : ' + calcular_status_aprovação_aluno(nota_aluno))
    print('nota do aluno : ' + str(nota_aluno))
```

Para comentar um trecho de código, no IDE PyCharm, basta selecionar as linhas do trecho de código a ser comentado e utilizar o comando Ctrl /. Quando você quiser remover os comentários, basta selecionar o trecho comentado e utilizar novamento o comando Ctrl /.

Para que a função calcular_status_aprovação_aluno possa ser utilizada no módulo main, será necessário incluir a importação dessa função:

```
from variáveis_tipos_funções_comandos_condicionais import imprimir_variável,\
calcular_status_aprovação_aluno
```

O caracter \ é utilizado para informar que o código da linha continua na linha seguinte. Você só precisará utilizá-lo quando o código de uma linha tiver que ser representado em mais de uma linha.

Lembre-se nas próximas subseções, de acrescentar as importações das funções originárias de outros módulos.

A saída da execução do programa é:

```
3.2 - função com comando condicional e operador + para concatenar strings
status do aluno : de exame
nota do aluno : 5.5
```

As duas chamadas da função print utilizam o operador + para concatenar strings. Na primeira chamada é passado como um dos argumentos a chamada da função obter_status_aprovação_aluno. O corpo desta função é composto por um comando condicional composto de três partes:

- teste da condição do **if** (se) : **if** média >= 6.0:
  - condição testada : se média maior ou igual a 6.0
  - caso este teste seja bem sucedido
    - execução de bloco interno: **return** 'aprovado'
- caso o teste da condição do **if** seja mal sucedido
  - teste da condição do **elif** (senão se) : **elif** 4.0 <= média < 6.0:
    - condição testada : se média maior ou igual a 4.0 e menor que 6.0
    - caso este teste seja bem sucedido
      - execução de bloco interno : **return** 'de exame'
- caso teste da condição do **elif** seja mal sucedido
  - execução de bloco interno ao **else** (senão) : **return** 'reprovado'

Devido ao fato, de que neste código ilustrativo, os blocos internos do **if**, **elif** e **else** tem apenas um único comando, o implementador tem a opção de alinhar esse comando após o caracter **:**, em vez de indentar o comando na próxima linha, desde que o código alinhado caiba na mesma linha.

Assim sendo, a forma indentada (em geral, encontrada nos códigos Python)

```
def calcular_status_aprovação_aluno(média):
    if média >= 6.0:
        return 'aprovado'
    elif 4.0 <= média < 6.0:
        return 'de exame'
    else:
        return 'reprovado'
```

poderá ser substituída pela forma alinhada (mais compacta).

```
def calcular_status_aprovação_aluno(média):
    if média >= 6.0: return 'aprovado'
    elif 4.0 <= média < 6.0: return 'de exame'
    else: return 'reprovado'
```

A palavra reservada return é utilizada para retornar um valor, ou uma lista de valores (entre vírgulas). Observe que uma função pode não retornar nenhum valor, tal como a função imprimir_variável ilustrada na subseção 3.1.

Na segunda chamada da função print em um dos argumentos passados é utilizada a chamada da função str, para converter o valor obtido da variável nota_aluno do tipo float para o tipo str. Essa conversão é necessária, porque para utilizar o operador + para concatenar strings é necessário que as partes concatenadas sejam strings. Essa conversão não foi necessária na primeira chamada, porque a função obter_status_aprovação_aluno retorna um valor do tipo str. É importante pontuar que somente o valor lido é convertido, mas que o tipo da variável continua associado ao valor atribuído à variável, ou seja, a variável nota_aluno continua com o valor 5.5 do tipo float.

3.3 - Utilizando comando condicional aninhado (interno)

Um comando condicional pode ter no bloco de comandos de qualquer uma das duas partes, um outro comando condicional. Esse comando condicional interno (contido por outro comando condicional externo) é chamado de comando condicional aninhado, porque sua utilização ocorre em um bloco de comandos de um comando condicional externo.

No módulo variáveis__tipos__funções__comandos_condicionais, acrescente a definição da função calcular_expectativa_aprovação_aluno:

```
def calcular_expectativa_aprovação_aluno(estudo_antecipado_tutoriais,
    teste_implementações_tutoriais, percentual_realização_exercícios_propostos):
    if percentual_realização_exercícios_propostos == 100:
        if estudo_antecipado_tutoriais and teste_implementações_tutoriais == 'completo':
            return 'expectativa muito alta de aprovação',\
                   '100% dos exercícios e estudo completo dos tutoriais'
        elif estudo_antecipado_tutoriais and teste_implementações_tutoriais == 'parcial':
            return 'expectativa alta de aprovação',\
                   ' 100% dos exercícios e estudo parcial dos tutoriais'
        else: return 'expectativa média de aprovação',\
                     ' 100% dos exercícios e nenhum estudo dos tutoriais'
    elif 70 <= percentual_realização_exercícios_propostos < 100:
        if estudo_antecipado_tutoriais and teste_implementações_tutoriais == 'completo':
            return 'expectativa alta de aprovação',\
                   'pelo menos 70% dos exercícios e estudo completo dos tutoriais'
        else: return 'expectativa média de aprovação',\
                     'pelo menos 70% dos exercícios e estudo parcial dos tutoriais'
    elif 50 <= percentual_realização_exercícios_propostos < 70:
        if estudo_antecipado_tutoriais and teste_implementações_tutoriais == 'completo':
            return 'expectativa média de aprovação',\
                   'pelo menos 50% dos exercícios e estudo completo dos tutoriais'
        else: return 'expectativa baixa de aprovação',\
                     'pelo menos 50% dos exercícios e estudo parcial dos tutoriais'
    else:
        if estudo_antecipado_tutoriais and teste_implementações_tutoriais == 'completo':
            return 'expectativa baixa de aprovação',\
                   'menos de 50% dos exercícios e estudo completo dos tutoriais'
        else: return 'expectativa muito baixa de aprovação',\
                     'menos de 50% dos exercícios e estudo parcial dos tutoriais'
```

Na lista de valores retornados, o caracter \ é utilizado para representar quebra de linha, pois a lista de strings retornada não cabe em uma única linha.

Para ilustrar chamadas de uma função que utiliza comandos condicionais aninhados, comente os trechos de ilustração das subseções 3.1 e 3.2 e acrescente o trecho de ilustração da seção 3.3. Por simplicidade, os trechos comentados foram omitidas no código ilustrado a seguir.

```
def ilustrar_variáveis_tipos_funções_comandos_condicionais():
    print('\n3.3 - chamadas de função que utiliza comandos condicionais aninhados')
    expectiva_aprovação, justificativa = calcular_expectativa_aprovação_aluno(
        estudo_antecipado_tutoriais = True, teste_implementações_tutoriais = 'completo',
        percentual_realização_exercícios_propostos = 100)
    print("para : estudo_antecipado_tutoriais = True, teste_implementações_tutoriais = 'completo',
          percentual_realização_exercícios_propostos = 100")
    print('- ' + expectiva_aprovação + '\n- justificativa: ' + justificativa)
    expectiva_aprovação, justificativa = calcular_expectativa_aprovação_aluno(
        estudo_antecipado_tutoriais = True, teste_implementações_tutoriais = 'parcial',
        percentual_realização_exercícios_propostos = 80)
    print("para : estudo_antecipado_tutoriais = True, teste_implementações_tutoriais = 'parcial',
          percentual_realização_exercícios_propostos = 80")
    print('- ' + expectiva_aprovação + '\n- justificativa: ' + justificativa)
```

O caracter **(** ao final da linha, ou o caracter **,** separando uma lista de argumentos, dispensam o uso do caracter \ para representar quebra de linha.

A saída da execução do programa é:

```
3.3 - chamadas de função que utiliza comandos condicionais aninhados
para : estudo_antecipado_tutoriais = True, teste_implementações_tutoriais = 'completo',
       percentual_realização_exercícios_propostos = 100
- expectativa muito alta de aprovação
- justificativa: 100% dos exercícios e estudo completo dos tutoriais
para : estudo_antecipado_tutoriais = True, teste_implementações_tutoriais = 'parcial',
       percentual_realização_exercícios_propostos = 80
- expectativa média de aprovação
- justificativa: pelo menos 70% dos exercícios e estudo parcial dos tutoriais
```

Na lista de argumentos passados para uma função, os caracteres **(** ou **,** são utilizados para representar quebra de linha em substituição ao caracter \.

Comando condicionais aninhados são comandos condicionais que fazem parte do bloco de comandos interno de qualquer uma das três partes de outro comando condicional.

Para que um bloco interno a uma parte de um comando condicional seja executado a condição testada deve ser verdadeira (True). Se a variável não for booleana é necessário compará-la com algum valor. Por exemplo: **if** percentual_realização_exercícios_propostos < 100. No entanto, se a variável da condição for booleana não é necessário compará-la com True ou False, porque o valor de uma variável booleana já é True ou False.

A comparação utilizando uma variável booleana

- **if** estudo_antecipado_tutoriais:

é uma simplificação da comparação

- **if** estudo_antecipado_tutoriais == True:

Pelo mesmo motivo, a comparação:

- **if not** estudo_antecipado_tutoriais:

é uma simplificação da comparação

- **if** estudo_antecipado_tutoriais == False:

Conforme ilustrado na função calcular_expectativa_aprovação_aluno, as partes **if** e **else** podem ser utilizadas uma única vez no comando condicional; no entanto, a parte **elif** pode ser utilizada várias vezes, testando condições distintas.

3.4 - Função recursiva

Nesta seção, serão ilustradas duas formas de calcular o fatorial de um número inteiro: (a) utilizando iteração; ou (b) utilizando recursividade. Uma função que utiliza iteração executa um bloco de comandos repetidas vezes (iteração). Uma função recursiva chama a si própria.

No módulo variáveis__tipos__funções__comandos_condicionais, acrescente a definição das seguintes funções:

```
def fatorial2(n):
    if n == 1: return 1
    else: return n * fatorial2(n - 1)

def fatorial1(n):
    fatorial = n
    while n > 1:
        fatorial *= n - 1
        n -= 1
    return fatorial
```

Para ilustrar a implementação de fatorial por uma função utilizando iteração e por outra utilizando recursão, comente os trechos de ilustração das subseções anteriores e acrescente o trecho de ilustração da seção 3.4.

```
def ilustrar__variáveis__tipos__funções__comandos_condicionais():
    print('\n3.4 - chamada de função com implementação recursiva ou não recursiva com iteração')
    n = 5
    print(f'função não recursiva (utilizando iteração) : fatorial({n}) = {fatorial1(n)}')
    print(f'função recursiva : fatorial2({n}) = {fatorial2(n)}')
```

A saída da execução do programa é:

```
3.4 - chamada de função com implementação recursiva ou não recursiva com iteração
função não recursiva (utilizando iteração) : fatorial(5) = 120
função recursiva : fatorial2(5) = 120
```

É possível representar fatorial(n) = n x (n - 1) ... 1. Na implementação da função fatorial1 é utilizado o comando de iteração **while**. Este comando testa uma condição de continuidade da iteração, neste exemplo: n > 1. Enquanto a condição for verdadeira, o bloco interno ao comando será executado iterativamente (repetidas vezes). Em Python o operador * representa uma multiplicação.

Também é possível representar fatorial(n) = n x fatorial (n - 1). Na implementação da função fatorial2, a função chama a si própria (recursividade), tornando o código mais compacto.

Uma soma, multiplicação, divisão e subtração podem ser representadas de uma forma mais compacta. Por exemplo, a soma x = x + 8 pode ser representada por x += 8. Esse tipo de simplificação foi utilizado para realizar uma multiplicação e uma subtração na função fatorial1.

Ao concluir a seção 3, você pode remover os comentários dos trechos de código que haviam sido comentados.

4 - Estruturas de Dados e Comandos de Iteração

Um conjunto de dados é ordenado (ou indexado), quando seus elementos podem ser lidos ou alterados a partir de um índice ou de um intervalo de índices (subconjunto de elementos do conjunto). Portanto, um conjunto não ordenado não pode ser indexado.

O índice inicial de um conjunto de elementos é 0 (zero). Para indexar um conjunto de elementos você pode utilizar uma notação que indica o intervalo de valores a ser acessado: [índice_inicial_inclusivo : índice_final_exclusivo]. Índice inicial inclusivo significa que intervalo de valores vai iniciar com esse índice. Índice final exclusivo significa que intervalo de valores vai finalizar com esse índice menos 1. Por exemplo, se conjunto for indexado no intervalo [3:9], serão acessados os elementos do conjunto do índice 3 até o índice 8.

Um conjuntos de dados é imutável quando após a atribuição dos seus elementos, um novo elemento não pode ser inserido, e seus elementos não podem ser alterados ou removidos.

Você não necessita utilizar uma biblioteca externa para criar e utilizar os conjuntos de dados incorporados na linguagem Python de uma forma bastante compacta.

Para ilustrar essa seção crie o módulo Python estruturas_dados__comandos_iteração.

4.1 - String : conjunto ordenado de caracteres

String é utilizado especificamente para manipular um conjunto de caracteres.

No módulo estruturas_dados__comandos_iteração defina as seguintes funções:

```
def converter_texto(texto, tipo):
    if tipo == 'maiúscula': return texto.upper()
    elif tipo == 'minúscula': return texto.lower()
    elif tipo == 'capital' : return texto.captalize()
    else: return 'tipo de conversão desconhecido'

def contar_caracteres_tipo(texto, tipo):
    total_caracteres_tipo = 0
    for caracter in texto:
        if tipo == 'letra_maiúscula' and caracter.isupper(): total_caracteres_tipo += 1
        elif tipo == 'letra_minúscula' and caracter.islower(): total_caracteres_tipo += 1
        elif tipo == 'número' and caracter.isnumeric(): total_caracteres_tipo += 1
    return total_caracteres_tipo

def mostrar_mensagem(objetivo, texto, tipo, resultado):
    print("texto '%s' - %s %s : %s" % (texto, objetivo, tipo, resultado))
```

No módulo main importe essas funções.

```
from estruturas_dados__comandos_iteração import mostrar_mensagem, contar_caracteres_tipo,
     converter_texto
```

No módulo main defina a função ilustrar__estruturas_dados__comandos_iteração com o trecho de código de ilustração da subseção 4.1.

```
print('\n4.1 - manipular strings')
    #        01234567890123456789012345678901
    texto = '2022 - Linguagem de Programação I'
    print("texto '%s' limitado no intervalo [7:16] : %s" % (texto, texto[7:16]))
    print("texto '%s' limitado no intervalo [-13:] : %s" % (texto, texto[-13:]))
    ação = 'contar caracteres com'
    tipo = 'letra_maiúscula'
    mostrar_mensagem(ação, texto, tipo, contar_caracteres_tipo(texto, tipo))
    tipo = 'número'
    mostrar_mensagem(ação, texto, tipo, contar_caracteres_tipo(texto, tipo))
    ação = 'converter para'
    tipo = 'minúscula'
    mostrar_mensagem(ação, texto, tipo, converter_texto(texto, tipo))
    tipo = 'maiúscula'
    mostrar_mensagem(ação, texto, tipo, converter_texto(texto, tipo))
```

No corpo principal do programa, comente a chamada da função de ilustração da seção 2 e acrescente a chamada da função de ilustração da seção 4.

```
if __name__ == '__main__':
    # ilustrar__variáveis__tipos__funções__comandos_condicionais()
    ilustrar__estruturas_dados__comandos_iteração()
```

A saída da execução do programa é:

```
4.1 - manipular strings
texto '2022 - Linguagem de Programação I' limitado no intervalo [7:16] : Linguagem
texto '2022 - Linguagem de Programação I' limitado no intervalo [-13:] : Programação I
texto '2022 - Linguagem de Programação I' - contar caracteres com letra_maiúscula : 3
texto '2022 - Linguagem de Programação I' - contar caracteres com número : 4
texto '2022 - Linguagem de Programação I' - converter para minúscula :
   2022 - linguagem de programação i
texto '2022 - Linguagem de Programação I' - converter para maiúscula :
   2022 - LINGUAGEM DE PROGRAMAÇÃO I
```

Na função mostrar_mensagem são utilizados demarcadores %s para compor o string passado como argumento na chamada da função print:

- print("texto '%s' - %s %s : %s" % (texto, objetivo, tipo, resultado))

Os demarcadores %s indicam que posições no string que serão substituídas pelas variáveis entre parênteses após o caracter %. Ou seja, o primeiro %s será substituído pelo valor do parâmetro texto, o segundo %s pelo valor do parâmetro objetivo, o terceiro %s pelo valor do parâmetro tipo e o quarto %s pelo valor do parâmetro resultado.

No variável texto = '2022 - Linguagem de Programação I':

- o primeiro índice (0) corresponde ao caracter '2' que inicia o string;
- e o último índice (32) corresponde ao caracter 'I' que finaliza o string.

'Linguagem' é o substring obtido de texto[7:16], que corresponde à indexação do texto no intervalo 7 até 15, dado que o índice final do intervalo é exclusivo, ou seja, não incluído no intervalo.

'Programação I' é o substring obtido de texto[-13:]. O índice inicial como -10 significa 10 índices anteriores ao final do string. O índice final exclusivo é 33, dado que o último índice incluído no intervalo é 39. Então para o final exclusivo -10 corresponde a 33 - 13 = 30. Portanto texto[-10:] é equivalente a texto[20:33].

Na função contar_caracteres_tipo é utilizado o comando de iteração: **for** caracter **in** texto:. Esse comando copia cada caracter da variável texto, do tipo str, e atribui à variável caracter a cada repetição de execução do bloco de comandos interno ao comando **for**. Então, a cada repetição de execução (loop) do bloco interno do comando **for**, a variável caracter é disponibilizada com o valor do próximo caracter do string atribuído anteriormente à variável texto, para ser utilizada com uma variável nos comandos do bloco interno do comando for. Por exemplo, para texto = 'Linguagem', na primeira execução do comando a variável caracter recebe o valor 'L' (primeiro caracter do string 'Linguagem'). Na próxima execução recebe o valor 'i' (segundo caracter), e assim por diante, até assumir o valor 'm' (último caracter do string 'Linguagem'). Note que as variáveis caracter e texto, poderiam ter qualquer outro nome.

A função contar_caracteres_tipo utiliza três funções aplicáveis a strings:
isupper : retorna True se todas os caracteres do string (ex: string.isupper()) são letras maísculas;
islower : idem para letras minúsculas;
isnumeric : idem para caracteres numéricos.

Note que neste caso, a função está sendo aplicada a um string que contém um único caracter, com o objetivo de totalizar quantos caracteres de um dado tipo (letra maíscula, letra minúscula ou caracter numérico) existem no string do parâmetro texto.

Na função converter_texto, são utilizadas as seguintes funções aplicáveis a strings::
upper : converte todos as letras de um string para letras maísculas;
lower : idem para letras minúsculas;
captalize : converte todas as palavras de um string para palavras iniciando com a primeira letra maiúscula e os demais letras da palavra com letras minúsculas.

O objetivo da função converter_texto é o de converter as letras de um string para o tipo passado como argumento.

4.2 - Lista (list) : conjunto ordenado

Lista suporta o armazenamento de qualquer tipo de elemento, e aceita elementos duplicados. No módulo estruturas_dados__comandos_iteração defina a seguinte função:

```
def converter_valores_negativos_matrizes(matriz):
    matriz_convertida = []
    for índice_linha, linha in enumerate(matriz):
        linha_convertida = []
        for índice_coluna, valor in enumerate(linha):
            if valor >= 0: linha_convertida.append(valor)
            else: linha_convertida.append(-valor)
        matriz_convertida.append(linha_convertida)
    return matriz_convertida
```

Para ilustrar o uso de listas, na função ilustrar__estruturas_dados__comandos_iteração comente o trecho que ilustra a subseção 4.1 (será omitido no código mostrado a seguir) e inclua o trecho que ilustra a seção 4.2.

```
def ilustrar__estruturas_dados__comandos_iteração():
    print('\n4.2 - manipular listas')
    cursos_facet = ['Física', 'Matemática', 'Química']
    print('conjunto inicial : ' + str(cursos_facet))
    cursos_facet.insert(0, 'Engenharia da Computação')
    cursos_facet.append('Sistemas de Informação')
    print('inserindo elemento na posição 0 e apendando elemento : ' + str(cursos_facet))
    print('intervalo [1:4] de elementos do conjunto : ' + str(cursos_facet[1:4]))
    del cursos_facet[1:-1]
    print('removendo do segundo elemento até o penúltimo : ' + str(cursos_facet))
    matriz = [[2, -7, 2], [8, 5, -4]]
    print('matriz com números negativos : ' + str(matriz))
    print('matriz com números negativos convertidos para positivos : '
        + str(converter_valores_negativos_matrizes(matriz)))
```

A saída da execução do programa é:

```
4.2 - manipular listas
conjunto inicial : ['Física', 'Matemática', 'Química']
inserindo elemento na posição 0 e apendando elemento : ['Engenharia da Computação', 'Física',
   'Matemática', 'Química', 'Sistemas de Informação']
intervalo [1:4] de elementos do conjunto : ['Física', 'Matemática', 'Química']
removendo do segundo elemento até o penúltimo : ['Engenharia da Computação',
   'Sistemas de Informação']
matriz com números negativos : [[2, -7, 2], [8, 5, -4]]
matriz com números negativos convertidos para positivos : [[2, 7, 2], [8, 5, 4]]
```

Inicialmente é atribuída à variável cursos_facet uma lista com 3 elementos:

- ['Física', 'Matemática', 'Química']

Como a lista é um conjunto ordenado, você poderá referenciar os índices dos elementos da lista para realizar operações. Vamos ilustrar: (a) inserção de elementos na lista, a partir de um dado índice; (b) cópia de uma sublista baseada na definição de um intervalo de índices; (c) a inserção de um elemento no final da lista (apendar um elemento); e (d) a remoção de sublistas definidas por um único elemento (único índice) ou por vários elementos em sequência (intervalo de índices).

Após imprimir a lista inicial, a função insert é utilizada para incluir o elemento 'Engenharia da Computação' na lista inicial, na posição 0, deslocando o elemento que estava na posição 0 para a próxima posição (1) e os elementos seguintes idem. A seguir, é apendado o elemento 'Sistemas de Informação' ao final da lista, e impressa a lista resultante, que agora já conta com cinco elementos.

A sublista gerada a partir do intervalo [1:4] da lista é impressa. Lembre-se que o último elemento é exclusivo, portanto, serão impressos os elementos indexados pelos índices 1, 2 e 3 da lista.

A função del é utilizada para remover do segundo até o penúltimo [1:-1] elementos da lista, com a impressão das listas resultantes em ambos os casos.

Uma matriz pode ser representada como uma lista de listas. Foi utilizado um exemplo de utilização de matrizes, a conversão dos valores negativos de uma matriz para valores positivos, com a função converter_valores_negativos_matrizes. Para evitar a alteração dos valores da matriz original é criada um nova matriz (matriz_convertida) e montada um nova linha, a partir de cada linha da matriz original, com os valores originais (se forem positivos ou zero) e com os valores convertidos (se forem negativos). Cada nova linha montada é apendada na nova matriz, inicializada como uma lista vazia.

A função converter_valores_negativos_matrizes ilustra como acessar as linhas da matriz (listas internas), e os valores de cada linha utilizando dois loops de iteração e a função enumerate, que retorna o índice e os elementos de um conjunto. A matriz é uma lista cujos elementos são as suas linhas, representadas como listas internas. Cada linha é uma lista cujos elementos são os seus valores, correspondentes aos valores da matriz. O loop externo itera em cada linha da matriz obtendo : índice_linha e linha. O loop interno itera em cada coluna da linha obtendo : índice_coluna e valor. É possível utilizar os dois índices para indexar qualquer valor da matriz, ou seja, o valor da matriz, com índice de linha i e índice de coluna j é representado como: matriz[i][j].

Nesta seção foram ilustradas operações com uma lista de strings, mais você pode utilizar qualquer tipo como elemento de uma lista, inclusive uma outra lista. Indo mais além, você pode definir uma lista com elementos de vários tipos, como por exemplo:

- ['Dourados', 'UFGD', 2021, True, ['azul', 'branco']].

4.3 - Dicionário (dict) : conjunto de elementos não ordenado acessado a partir de chaves

O dicionário é composto de pares chave-valor. A chave é utilizada para inserção e posterior recuperação, alteração ou remoção do valor associado a ela. Chaves podem ser dos seguintes tipos: str, int, float, bool. Os valores podem ser de qualquer tipo: str, int, float, bool, list (lista), dict (dicionário), e outros tipos que serão ilustrados no restante deste tutorial.

No módulo estruturas_dados__comandos_iteração defina as seguintes funções:

```
def imprimir_lista_valores_indexados_por_mesma_chave(dicionário, chave_interna):
    valores = ''
    for chave in dicionário: valores +=  ' - ' + str(dicionário[chave][chave_interna])
    print("disciplinas indexadas pela chave '%s' :\n%s" % (chave_interna, valores))

def imprimir_chaves_dicionário(nome, dicionário):
    chaves = ''
    for chave in dicionário: chaves +=  ' - ' + str(chave)
    print("chaves do dicionário '%s' :\n%s" % (nome, chaves))
```

Para ilustrar o uso de dicionários, na função ilustrar__estruturas_dados__comandos_iteração comente os trechos que ilustram as subseções 4.1 e 4.2 e inclua o trecho que ilustra a seção 4.3.

```
def ilustrar__estruturas_dados__comandos_iteração():
    print('\n4.3 - manipular dicionários')
    país = {}
    país['nome'] = 'Brasil'
    país['continente'] = 'América do Sul'
    país['regiões'] = ['sul', 'sudeste', 'centro oeste', 'nordeste', 'norte']
    país['Índice de Desenvolvimento Humano'] = 0,765
    país['Índice de Percepção da Corrupção'] = 38
    país['regime democrático'] = True
    print('dicionário com vários tipos de elementos :\n  %s' % (país))
    chave = 'nome'
    print("indexando valor do dicionário '%s' pela chave '%s' : %s" % ('país', chave, país[chave]))
    chave = 'regiões'
    print("indexando valor do dicionário '%s' pela chave '%s' : %s" % ('país', chave, país[chave]))
    imprimir_chaves_dicionário('país', país)
    print("lista de chaves do dicionário '%s' :\n %s" % ('país', list(país)))
    disciplina = {'título':'Linguagem de Programação I', 'categoria':'optativa', 'carga horária':72,
        'modalidade':'prática'}
    print("dicionário '%s' com os pares chave-valor iniciais : \n  %s" % ('disciplina', disciplina))
    disciplina['categoria'] = 'obrigatória'
    del disciplina['modalidade']
    print("dicionário '%s' após alteração da categoria e remoção da modalidade :\n  %s"
        % ('disciplina', disciplina))
    disciplinas_sistemas_informação = {}
    disciplinas_sistemas_informação['Linguagem de Programação I']\
        = {'título':'Linguagem de Programação I', 'categoria':'obrigatória', 'carga horária':72}
    disciplinas_sistemas_informação['Tópicos em Deep Learning']\
        = {'título':'Tópicos em Deep Learning', 'categoria':'optativa', 'carga horária':72}
    disciplinas_sistemas_informação['Representação do Conhecimento']\
        = {'título':'Representação do Conhecimento', 'categoria':'eletiva', 'carga horária':72}
    chave_externa = 'Tópicos em Deep Learning'
    print("dicionário '%s' indexado com a chave '%s' :\n %s" % ('disciplinas_sistemas_informação',
          chave_externa, disciplinas_sistemas_informação[chave_externa]))
    chave_interna = 'categoria'
    print("dicionário '%s' indexado com chave externa '%s' e chave interna '%s' : %s"
          % ('disciplinas_sistemas_informação', chave_externa, chave_interna,
          disciplinas_sistemas_informação[chave_externa][chave_interna]))
    imprimir_lista_valores_indexados_por_mesma_chave(disciplinas_sistemas_informação, 'categoria')
    print('\n4.4 - manipular tuplas')
    categorias_disciplinas = ('obrigatória', 'optativa', 'eletiva')
    print('tupla : ' + str(categorias_disciplinas))
    reta = lambda x: 5*x + 2 # f(x) = 5*x + 2
    print('coordenadas da reta : %s' % (criar__lista_coordenadas__geradas__por__função(reta, -5, 6,
        2)))
    parábola = lambda x: x ** 2
    print('coordenadas da parábola : %s' % (criar__lista_coordenadas__geradas__por__função(parábola,
        -5, 6, 2)))
    print('\n4.5 - manipular conjuntos (sets)')
    faculdades1 = {'FACET', 'FAEN', 'FCS'}
    faculdades2 = {'FACET', 'FAEN', 'FADIR'}
    print('união de %s com %s : %s' %(faculdades1, faculdades2, faculdades1.union(faculdades2)))
    print('intersecção de %s com %s : %s' % (faculdades1, faculdades2,
        faculdades1.intersection(faculdades2)))
```

A saída da execução do programa é:

```
4.3 - manipular dicionários
dicionário com vários tipos de elementos :
  {'nome': 'Brasil', 'continente': 'América do Sul',
   'regiões': ['sul', 'sudeste', 'centro oeste', 'nordeste', 'norte'],
 'Índice de Desenvolvimento Humano': (0, 765),
  'Índice de Percepção da Corrupção': 38, 'regime democrático': True}
indexando valor do dicionário 'país' pela chave 'nome' : Brasil
indexando valor do dicionário 'país' pela chave 'regiões' : ['sul', 'sudeste', 'centro oeste',
   'nordeste', 'norte']
chaves do dicionário 'país' :
 - nome - continente - regiões - Índice de Desenvolvimento Humano - Índice de Percepção da Corrupção
 - regime democrático
lista de chaves do dicionário 'país' :
 ['nome', 'continente', 'regiões', 'Índice de Desenvolvimento Humano',
  'Índice de Percepção da Corrupção', 'regime democrático']
dicionário 'disciplina' com os pares chave-valor iniciais :
  {'título': 'Linguagem de Programação I', 'categoria': 'optativa', 'carga horária': 72,
   'modalidade': 'prática'}
dicionário 'disciplina' após alteração da categoria e remoção da modalidade :
  {'título': 'Linguagem de Programação I', 'categoria': 'obrigatória', 'carga horária': 72}
dicionário 'disciplinas_sistemas_informação' indexado com a chave 'Tópicos em Deep Learning' :
 {'título': 'Tópicos em Deep Learning', 'categoria': 'optativa', 'carga horária': 72}
dicionário 'disciplinas_sistemas_informação' indexado com chave externa 'Tópicos em Deep Learning'
   e chave interna 'categoria' : optativa
disciplinas indexadas pela chave 'categoria' :
 - obrigatória - optativa - eletiva
```

Esse trecho de código ilustra várias situações:

- dicionário país sendo acrescido de pares chave-valor com valores de vários tipos
- dicionário país sendo indexado pela chave 'nome'
- idem para a chave 'região'
- iteração para imprimir chaves do dicionário país
  - função imprimir_chaves_dicionário itera nas chaves do dicionário
    `for chave in dicionário:`
- impressão de lista de chaves do dicionário país
- impressão do dicionário disciplina
- impressão do dicionário disciplina após
  - após alteração do valor da chave 'modalidade'
  - e remoção do par chave-valor indexado pela chave 'carga horária'
- dicionário disciplinas_sistemas_informação sendo acrescido de pares chave-valor com:
  - chave : título de uma disciplina
  - valor : dicionário representando pares chave-valor de uma disciplina
- dicionário disciplinas_sistemas_informação indexado pela chave 'Tópicos em Deep Learning'
- disciplina interna ao dicionário disciplinas_sistemas_informação
  - indexado pelas chave externa 'Tópicos em Deep Learning'
  - e pela chave interna 'modalidade'
- disciplinas internas ao dicionário disciplinas_sistemas_informação indexadas pela chave 'categoria'
  - função imprimir_lista_valores_indexados_por_mesma_chave
    - itera nas chaves do dicionário para montar string com os valores nos dicionários internos indexados pela chave passada como parâmetro

4.4 - Tupla (tuple) : conjunto imutável e ordenado de elementos

Tupla pode ser acessada através de índices e aceita elementos de qualquer tipo. Seus elementos não podem ser alterados individualmente, nem inseridos ou removidos.

No módulo estruturas_dados__comandos_iteração defina as seguinte função:

```
def criar__lista_coordenadas__geradas__por__função(função, horizontal_inicial,
        horizontal_final_exclusiva, passo):
    coordenadas_percurso = []
    for coordenada_horizontal in range(horizontal_inicial, horizontal_final_exclusiva, passo):
        coordenada_vertical = função(coordenada_horizontal)
        coordenadas_percurso.append((coordenada_horizontal, coordenada_vertical))
    return coordenadas_percurso
```

Para ilustrar o uso de tuplas, na função ilustrar__estruturas_dados__comandos_iteração comente os trechos que ilustram as subseções 4.1, 4.2 e 4.3, e inclua o trecho que ilustra a seção 4.4.

```
print('\n4.4 - manipular tuplas')
categorias_disciplinas = ('obrigatória', 'optativa', 'eletiva')
print('tupla : ' + str(categorias_disciplinas))
reta = lambda x: 5*x + 2
print('coordenadas da reta : %s' % (criar__lista_coordenadas__geradas__por__função(reta, -5, 6, 2)))
parábola = lambda x: x ** 2
print('coordenadas da parábola : %s' % (criar__lista_coordenadas__geradas__por__função(parábola, -5,
      6, 2)))
```

A saída da execução do programa é:

```
4.4 - manipular tuplas
tupla : ('obrigatória', 'optativa', 'eletiva')
coordenadas da reta : [(-5, -23), (-3, -13), (-1, -3), (1, 7), (3, 17), (5, 27)]
coordenadas da parábola : [(-5, 25), (-3, 9), (-1, 1), (1, 1), (3, 9), (5, 25)]
```

Inicialmente é criada e impressa a tupla categorias_disciplinas, com strings correspondentes às categorias das disciplinas do curso de Sistemas de Informação.

Vamos aproveitar para aprender dois conceitos muito úteis: (a) definição de função anônima; e (b) passagem de função como parâmetro.

A palavra reservada lambda é utilizada para criar uma função anônima (sem nome), como por exemplo lambda x: 5*x + 2, que é equivalente a f(x) = x * 5 + 2. A seguir a função anônima é atribuída à variável reta. Neste caso, a variável reta passa a ser uma função com parâmetro x e corpo 5*x + 2. Posteriormente, no código acima, é defina a função anônima lambda x: x ** 2, equivalente a f(x) = x**2, que é atribuída à variável parábola. Os caracteres x**n indicam: x elevado a n; ou seja, x**2 é x elevado ao quadrado.

A função criar__lista_coordenadas__geradas__por__função recebe uma função como argumento do seu primeiro parâmetro função. Seus outros parâmetros são: horizontal_inicial, horizontal_final_exclusiva e passo. Para iterar nas coordenadas horizontais é utilizado o comando de iteração:

```
for coordenada_horizontal in range(horizontal_inicial, horizontal_final_exclusiva, passo):
```

A função range é utilizada para gerar um intervalo de valores, partindo de um valor inicial, incrementando esse valor com um passo (valor do incremento), e parando antes que o valor final seja atingido (exclusivo) ou ultrapassado.

A passagem de uma função como parâmetro, é um mecanismo muito poderoso, pois permite que uma única função, neste caso a função criar__lista_coordenadas__geradas__por__função, utilize uma função distinta para realizar seu processamento, cada vez que é chamada com uma outra função como parâmetro.

4.5 - Set : conjunto não ordenado sem elementos duplicadas

Set não aceita elementos de mesmo valor duplicados, e seus elementos não pode ser acessados através de índices.

Para ilustrar o uso de sets (conjuntos não ordenados sem elementos duplicados), na função ilustrar__estruturas_dados__comandos_iteração comente os trechos que ilustram as subseções 4.1, 4.2, 4.3 e 4.4, e inclua o trecho que ilustra a seção 4.5.

```
def ilustrar__estruturas_dados__comandos_iteração():
    print('\n4.5 - manipular conjuntos (sets)')
    faculdades1 = {'FACET', 'FAEN', 'FCS'}
    faculdades2 = {'FACET', 'FAEN', 'FADIR'}
    print('união de %s com %s : %s' %(faculdades1, faculdades2, faculdades1.union(faculdades2)))
    print('intersecção de %s com %s : %s' % (faculdades1, faculdades2,
        faculdades1.intersection(faculdades2)))
```

A saída da execução do programa é:

```
4.5 - manipular conjuntos (sets)
união de {'FAEN', 'FACET', 'FCS'} com {'FAEN', 'FADIR', 'FACET'} : {'FADIR', 'FACET', 'FAEN', 'FCS'}
intersecção de {'FAEN', 'FACET', 'FCS'} com {'FAEN', 'FADIR', 'FACET'} : {'FAEN', 'FACET'}
```

Nesta ilustração são definidos dois conjuntos (sets), ambos com três faculdades da UFGD. A função union é utilizada para gerar um conjunto a partir da união dos conjuntos iniciais. Note que as faculdades FAEN e FACET, que fazem parte dos dois conjuntos iniciais não são íncluídas duas vezes no conjunto gerado, porque esse tipo de conjunto não aceita elementos duplicados.

A função intersection é utilizada para gerar um conjunto a partir da interseção dos conjuntos iniciais. Note que somente as faculdades que fazem parte dos dois conjuntos iniciais, FAEN e FACET, são incluídas no conjunto gerado.

Com esta seção encerramos a aplicação LPI-E0, com a qual ilustramos a utilização de conceitos básicos de Python.

5 – Leitura e Escrita em Arquivos

Nesta seção, vamos ilustrar a leitura e a escrita de dados em dois formatos de arquivos muito utilizados: CSV e JSON.

Para ilustrar essa seção crie o módulo leitura_escrita_arquivos. Para armazenar os arquivos que serão gerados, crie o diretório dados no projeto, no mesmo nível hierárquico do diretório src.

5.1 - Salvar/recuperar matriz (lista de listas) em/de arquivo CSV

O formato CSV (Comma Separated Values), Valores Separados por Vírgulas, é muito utilizado para representar dados de planilhas ou de base de dados.

No módulo leitura_escrita_arquivos defina as seguintes funções :

```
def carregar_arquivo_csv(nome_arquivo):
    arquivo = open('dados/'+ nome_arquivo + '.csv', 'r')
    matriz_str = arquivo.read().strip('\n')
    arquivo.close()
    matriz = matriz_str.split('\n')
    for índice_linha, linha_str in enumerate(matriz):
        linha = list(linha_str.split(', '))
        matriz[índice_linha] = linha
        for índice_coluna, valor in enumerate(linha):
            valor = valor.strip()
            linha[índice_coluna] = valor
    return matriz

def salvar_arquivo_csv(nome_arquivo, matriz):
    arquivo = open('dados/' + nome_arquivo + '.csv', 'w')
    for linha in matriz:
        valores_str = ', '.join(map(str, linha))
        arquivo.write(f"{valores_str}\n")
    arquivo.close()
```

No módulo main, defina a função ilustrar_leitura_escrita_arquivos com o trecho de código de ilustração da subseção 4.1.

```
def ilustrar_leitura_escrita_arquivos():
    print('\n5.1 - salvar/recuperar matriz (lista de listas) em/de arquivo CSV')
    disciplinas_matriz1 = [
        ['título', 'categoria', 'carga horária'],
        ['Linguagem de Programação I', 'obrigatória', 72],
        ['Tópicos em Deep Learning', 'optativa', 72],
        ['Representação do Conhecimento', 'eletiva', 72]
    ]
    salvar_arquivo_csv('disciplinas1', disciplinas_matriz1)
    disciplinas_matriz2 = carregar_arquivo_csv('disciplinas1')
    for disciplina in disciplinas_matriz2: print(disciplina)
```

No corpo principal do programa, comente as chamadas das funções que ilustram as seções 2 e 3, e acrescente a chamada da função que ilustra a seção 4.

```
if __name__ == '__main__':
    # ilustrar_variáveis_tipos_funções_comandos_condicionais()
    # ilustrar_estruturas_dados_comandos_iteração()
    ilustrar_leitura_escrita_arquivos()
```

A saída da execução do programa é:

```
5.1 - salvar/recuperar matriz (lista de listas) em/de arquivo CSV
['título', 'categoria', 'carga horária']
['Linguagem de Programação I', 'obrigatória', '72']
['Tópicos em Deep Learning', 'optativa', '72']
['Representação do Conhecimento', 'eletiva', '72']
```

Cria uma matriz (lista de listas) de disciplinas. A função salvar_arquivo_csv é chamada para salvar a matriz de disciplinas em um arquivo com extensão csv. Utiliza a função carregar_arquivo_csv para carregar o conteúdo do arquivo gerado previamente e itera na linhas do arquivo, obtendo e imprimindo cada linha (lista com o valores de uma disciplina).

A função salvar_arquivo_csv :

- utiliza a função open para criar a variável arquivo para escrita de um arquivo no sub diretório dados do projeto
- itera na linhas da matriz
  - utiliza a função map para aplicar a função str a todos os valores da linha da matriz
  - utiliza a função join, aplicável a um string, para gerar um string agrupando os valores separados por ', '
  - utiliza a função write para escrever o string de cada linha no arquivo finalizando com o caracter '\n'
- utiliza a função close para fechar o arquivo

A função carregar_arquivo_csv :

- utiliza a função open para criar a variável arquivo para leitura de arquivo no sub diretório dados do projeto
- utiliza a função read para ler o conteúdo do arquivo
- utiliza a função strip para remover o caracter \n (pula linha) do final do arquivo;
- itera no string da matriz gerada obtendo o índice e string de cada linha da matriz
  - utiliza a função split para remover os caracteres ', ' entre subtrings de cada linha
  - cria uma lista com os strings gerados a partir da linha;
  - substitui o string da linha na matriz, pela lista de strings gerados a partir da linha;
  - itera na lista de strings da linha obtendo o índice do string e o string
    - utiliza a função strip para remover caracteres em branco em torno do string
    - substitui o string na matriz pelo string sem caracteres em branco ao redor
- retorna a matriz lida e processada

5.2 - Salvar/recuperar dicionário de dicionários em/de arquivo JSON

O formato JSON (JavaScript Objetcs Notation), Notação de Objetos Java Script, é muito utilizado para representar objetos (ver Tutorial 2), que são suportados pelas linguagens de programação mais utilizadas atualmente. Esse formato é equivalente ao formato de um dicionário.

No módulo leitura_escrita_arquivos, importe o módulo json e defina as seguintes funções:

```
import json

def carregar_arquivo_json(nome_arquivo):
    arquivo = open('dados/' + nome_arquivo + '.json', 'r')
    dicionário = json.load(arquivo)
    return dicionário

def salvar_arquivo_json(nome_arquivo, dicionário):
    arquivo = open('dados/' + nome_arquivo + '.json', 'w')
    json.dump(dicionário, arquivo)
    arquivo.close()
```

Na função ilustrar_leitura_escrita_arquivos comente o trecho de código de ilustração da subseção 5.1, e acrescente o trecho de ilustração da subseção 5.2.

```
def ilustrar_leitura_escrita_arquivos():
    print('\n5.2 - salvar/recuperar dicionário de dicionários em/de arquivo JSON')
    disciplinas_dicionários1 = {}
    disciplinas_dicionários1['Linguagem de Programação I']\
        = {'título': 'Linguagem de Programação I', 'categoria': 'obrigatória', 'carga horária': 72}
    disciplinas_dicionários1['Tópicos em Deep Learning'] \
        = {'título': 'Tópicos em Deep Learning', 'categoria': 'optativa', 'carga horária': 72}
    disciplinas_dicionários1['Representação do Conhecimento'] \
        = {'título': 'Representação do Conhecimento', 'categoria': 'eletiva', 'carga horária': 72}
    salvar_arquivo_json('disciplinas2', disciplinas_dicionários1)
    disciplinas_dicionários2 = carregar_arquivo_json('disciplinas2')
    for chave in disciplinas_dicionários2: print(disciplinas_dicionários2[chave])
```

A saída da execução do programa é:

```
5.2 - salvar/recuperar dicionário de dicionários em/de arquivo JSON
salvar_arquivo_json
{'título': 'Linguagem de Programação I', 'categoria': 'obrigatória', 'carga horária': 72}
{'título': 'Tópicos em Deep Learning', 'categoria': 'optativa', 'carga horária': 72}
{'título': 'Representação do Conhecimento', 'categoria': 'eletiva', 'carga horária': 72}
```

Cria um dicionário de dicionários, sendo que cada dicionário interno representa uma disciplina. A função salvar_arquivo_json é chamada para salvar o dicionário de disciplinas em um arquivo com extensão json. Utiliza a função carregar_arquivo_json para carregar o conteúdo do arquivo gerado previamente e itera na linhas do arquivo, obtendo e imprimindo cada linha (dicionário representando uma disciplina).

A função salvar_arquivo_ json :

- utiliza a função open para criar a variável arquivo para escrita de um arquivo no sub diretório dados do projeto
- utiliza a função json.dump para salvar um dicionário em um arquivo com extensão json
- utiliza a função close para fechar o arquivo

A função carregar_arquivo_json :

- utiliza a função open para criar a variável arquivo para leitura de arquivo no sub diretório dados do projeto
- utiliza a função json.load para carregar um dicionário a partir de um arquivo com extensão json
- retorna o dicionário lido

Exercícios

O aprendizado incremental baseado na ilustração de uma aplicação, que vai evoluindo em paralelo com o detalhamento de conceitos, é muito efetivo se você tiver método para estudar. Você receberá uma Lista de Exercícios, para consolidar os conhecimentos adquiridos neste tutorial.

Estude detalhadamente todo o tutorial, realizando as execuções solicitadas. Revise os conceitos aprendidos, antes de iniciar a implementação da Lista de Exercícios. Se aparecer alguma dúvida na realização de algum exercício, deixe o exercício de lado e estude novamente as seções do material associadas a sua dúvida. Então deixe o material de lado e retome seu exercício.