LPI - Tutorial 2 : Classe e Objetos

1 - Classe e Objetos : Mapeando Entidade do Mundo Real em novo Tipo da Aplicação

Nesta seção vamos demonstrar a versatilidade de mapear uma entidade do mundo real em um novo tipo da linguagem. Vamos desenvolver o projeto LPI-E2 para implementar uma aplicação para cadastrar alunos baseada em uma única entidade do mundo real : Aluno. Nesta aplicação vamos :

- cadastrar vários alunos;
- imprimir os dados do conjunto de alunos cadastrados;
- definir alguns filtros e selecionar os alunos cadastrados que atendam esses filtros;
- imprimir o subconjunto de alunos selecionados.

Se olharmos para o mundo real, veremos que um aluno pode ter um grande número de atributos. Para exercitarmos a definição de atributos com diferentes tipos, vamos escolher os seguintes atributos: nome, ano de nascimento, sexo e estrangeiro (sim ou não). No Tutorial 3, vamos substituir o atributo ano de nascimento pelo atributo data; assim poderemos calcular a idade do aluno. No entanto, o atributo data será baseado no tipo Data para o qual definiremos três atributos: dia, mês e ano. Mas como vamos trabalhar os conceitos de forma incremental, por enquanto, vamos utilizar o atributo ano de nascimento representado simplesmente por um tipo inteiro.

Para organizar os módulos desta aplicação são utilizados diretórios que podem conter subdiretórios ou módulos (arquivos py). Temos duas opções para organizar os módulos: diretórios e pacotes. Ao criarmos um pacote é gerado automaticamente um módulo __init__. Como vamos utilizar o bloco inicial da aplicação da mesma forma que utilizamos no Tutorial 1, optamos por utilizar diretórios. Lembre-se que o bloco principal da aplicação está vinculado ao comando condicional:

```
if __name__ == '__main__':
    # comandos do bloco principal da aplicação
```

No paradigma orientado a objetos temos o conceito de classe. Neste aplicação, vamos definir um diretório entidades contendo o módulo aluno, no qual vamos definir a classe Aluno. No contexto de uma classe, definimos funções utilizadas para manipular os dados da classe, denominadas métodos. Inicialmente definimos a classe Aluno com os métodos padronizados __init__ e __str__. Observe que a palavra reservada class é utilizada para definir a classe e a palavra reservada def é utilizada para definir os métodos, que nada mais são que funções que manipulam os dados da classe.

```
class Aluno:

    def __init__(self, nome, ano_nascimento, sexo, estado_civil, estrangeiro):
        self.nome = nome
        self.ano_nascimento = ano_nascimento if ano_nascimento > 1950 else -1
        self.sexo = sexo if sexo in ('M', 'F') else 'indefinido'
        self.estado_civil = estado_civil if estado_civil\
            in ('solteiro', 'casado', 'separado', 'divorciado', 'viúvo') else 'indefinido'
        self.estrangeiro = estrangeiro

    def __str__(self):
        return self.nome + ' -- ano de nascimento: ' + str(self.ano_nascimento)
               + ' - sexo: ' + self.__to_str_sexo__()\
               + ' - estado civil: ' + self.__to_str_estado_civil__()
               + self.__to_str_estrangeiro__()
```

O método __init__ é utilizado para definir e inicializar os dados da classe, que são sempre referenciados pela palavra reservada self (próprio), indicando que os dados pertencem à própria classe. Observe também que por padrão todo método deve ter como primeiro (ou único) parâmetro o self. Os dados de uma classe são comumente chamados de atributos porque especificam uma propriedade que uma entidade tem no mundo real. Um atributo relevante para um aluno é o seu nome. Observe que o atribuito self.nome está sendo inicializado com o valor do parâmetro nome, que deverá posteriormente receber um valor do tipo string (uma sequencia de caracteres), como por exemplo: 'Ana Julia Silveira'. Em geral, strings em Python são delimitados, por haspas simples: ' '.

O atributo self.estrangeiro, é inicializado de forma semelhante ao atributo self.nome. Posteriormente esse atributo irá receber um valor do tipo boolean, cujos valores em Python são: **True** (para indicar que o aluno é estrangeiro) e **False** (para indicar o contrário, ou seja, que o aluno não é estrangeiro).

Uma opção interessante para delimitar os valores que podem ser atribuídos a uma atributo é condicionar a atribuição à uma condição de viabilidade. No caso do atributo self.ano_nascimento, estamos condicionando a atribuição à condição if ano_nascimento > 1950 else -1, indicando que a atribuição do valor do parâmetro ano_nascimento somente será realizada se o valor for maior que 1950, caso contrário será atribuído o valor -1, para indicar que o valor recebido não é adequado como ano de nascimento do aluno.

No caso do atributo self.sexo, a atribuição foi condicionada a que o valor pertença à tupla que caracteriza os possíveis valores que o sexo do aluno pode assumir ('M': masculino ou , 'F': feminino): if sexo in ('M', 'F') else 'indefinido'. No caso do atributo self.estado_civil, a tupla utilizada define, de forma semelhante, os valores possíveis para o estado civil do aluno. Se os atributos não atenderem à condição de viabilidade, serão inicializados com o string 'indefinido'. Em outras linguagens de programação esse conjunto de valores factíveis para um determinado atributo é chamado de enumerado, porque enumera os valores que um dado atributo pode assumir.

O outro método com nome padronizado, é o método __str__, que retorna um string representando o conteúdo dos atributos do objeto. Dois esclarecimentos são necessários para entender a implementação desse método. O primeiro é que, conforme já vimos na seção 3.2, quanto o operador + é utilizado para concatenar strings, todas as suas partes devem ser strings. Assim sendo, o método pré-definido str está sendo utilizado para converter para string o atributo self.ano_nascimento, que durante a execução receberá um valor inteiro.

O segundo esclarecimento, é que algumas vezes é desejável utilizar um método para converter determinados valores de atributos em strings mais amigáveis para serem visualizados pelo usuário da aplicação. Com esse objetivo, foram definidos três métodos de conversão para string.

```
class Aluno:

    def __to_str_sexo__(self):
        if self.sexo == 'M': return 'masculino'
        elif self.sexo == 'F': return 'feminino'
        else: return 'indefinido'

    def __to_str_estado_civil__(self):
        if self.sexo == 'M': return self.estado_civil
        elif self.sexo == 'F':
            if self.estado_civil != 'indefinido': return self.estado_civil[:-1] + 'a'
            else: return 'indefinido'
        else: return self.estado_civil + '/a'
```

```
    def __to_str_estrangeiro__(self):
        if not self.estrangeiro: return ''
        else:
            if self.sexo == 'M': return ' - estrangeiro'
            elif self.sexo == 'F': return ' - estrangeira'
            else: return ' - estrangeiro/a'
```

O método __to_str_sexo__ foi implementado para que os strings 'M' e 'F' sejam substituídos por strings mais amigáveis para a interface: 'masculino' e 'feminino'.

O método __to_str_estado_civil__ foi implementado para harmonizar o gênero do estado civil com o sexo do aluno, de forma a torna mais amigável a visualização na interface. Ou seja, embora o atributo self.estado_civil seja inicializado e utilizado no gênero masculino, será visualizado com o gênero feminino, quando for relacionado com uma aluna. Observe a expressão self.estado_civil[:-1] utilizada neste método, que gera um substring que se inicia no primeiro caracter do string e finaliza no penúltimo caracter do string.

O método __to_str_estrangeiro__ foi implementado para retornar string vazio (' ') quando o aluno não for estrangeiro e retornar o string 'estrangeiro' ou 'estrangeira' em concordância com o sexo do aluno.

2 - Construindo e Imprimindo um Conjunto de Objetos

Ivar Jacobson, um sueco com larga experiência no desenvolvimento de software de grande porte para o setor de telecomunicações, definiu em seu famoso livro Object Oriented Software Engineering, que as classes de um sistema bem projetado no paradigma orientado a objeto devem ser organizadas em três grandes grupos: (1) as classes de entidade mapeadas a partir das entidades do mundo real, que são de interesse para um dado sistema em desenvolvimento; (2) as classe de interface responsáveis por toda a entrada e saída do sistema; e as (3) as classes de controle, responsáveis por coordenar a execução do sistema. Utilizar esse direcionamento de projeto, resulta no desenvolvimento de projetos mais robustos, no qual as mudanças não se propagam indesejadamente para qualquer tipo de classe do sistema. Mudanças no layout da interface são absorvidas pelas classes de interface, e mudanças no controle do sistema são absorvidas pelas classes de controle. Em sistemas mais complexos é comum termos várias classes de controle, mas nas aplicações que vamos construir nesta disciplina, uma única classe de controle será suficiente.

Vamos definir o módulo cadastro_alunos, no pacote controle, para criar inicialmente um conjunto de objetos de alunos e imprimi-los.

A criação do objeto de uma classe é realizada a partir da chamada de um método com o mesmo nome da classe, denominado construtor, porque é utilizado para construir objetos (variáveis) do tipo da classe. Observe a criação de 8 objetos da classe Aluno, inseridos sequencialmente (apendados) na lista alunos. O implementador tem a opção de passar os argumentos de uma função atribuindo-os aos parâmetros. Utilizamos essa opção para tornar mais legível a primeira chamada do construtor da classe Aluno. Nas demais chamadas esse artifício é desnecessário porque todas se referem à chamada do mesmo método e, portanto, tem os mesmo nomes de parâmetros.

Para armazenar os objetos da classe Aluno no módulo cadastro_alunos, vamos definir a lista alunos no módulo aluno, do diretório entidades. Adicionalmente, vamos definir duas funções associadas à lista alunos, para retornar a lista (get_alunos) e para inserir um novo aluno na lista (inserir_aluno).

```
alunos = []

def get_alunos(): return alunos

def inserir_aluno(aluno): alunos.append(aluno)
```

No módulo cadastro_alunos no diretório controle, vamos implementar inicialmente o cadastro de alunos, e impressão desses alunos na tela.

```
from entidades.aluno import inserir_aluno, Aluno, get_alunos, selecionar_alunos

def cadastrar_alunos():
    inserir_aluno(Aluno(nome='Ana Julia', ano_nascimento=1995, sexo='F', estado_civil='solteiro',
                  estrangeiro=False))
    inserir_aluno(Aluno('Joaquim', 1990, 'M', 'casado', True))
    inserir_aluno(Aluno('Ana Ligia', 1998, 'F', 'solteiro', False))
    inserir_aluno(Aluno('Mateus', 1991, 'M', 'solteiro', False))
    inserir_aluno(Aluno('Livia', 1985, 'F', 'casado', True))
    inserir_aluno(Aluno('Roberto', 1990, 'M', 'casado', False))
    inserir_aluno(Aluno('Ana Maria', 1991, 'F', 'solteiro', False))
    inserir_aluno(Aluno('Sandro', 1992, 'X', 'solteiro', True))

def imprimir_objetos(cabeçalho, objetos, filtros = None):
    if filtros == None: print('\n' + cabeçalho)
    else: print ('\n' + cabeçalho + filtros)
    for índice, objeto in enumerate(objetos): print(str(índice + 1) + ' - ' + str(objeto))


if __name__ == '__main__':
    cadastrar_alunos()
    imprimir_objetos('Alunos Cadastrados', get_alunos())
```

A saída da execução inicial da aplicação é a seguinte:

```
Alunos Cadastrados
  Ana Julia -- ano de nascimento: 1995 - sexo: feminino - estado civil: solteira
  Joaquim -- ano de nascimento: 1990 - sexo: masculino - estado civil: casado - estrangeiro
  Ana Ligia -- ano de nascimento: 1998 - sexo: feminino - estado civil: solteira
  Mateus -- ano de nascimento: 1991 - sexo: masculino - estado civil: solteiro
  Livia -- ano de nascimento: 1985 - sexo: feminino - estado civil: casada - estrangeira
  Roberto -- ano de nascimento: 1990 - sexo: masculino - estado civil: casado
  Ana Maria -- ano de nascimento: 1991 - sexo: feminino - estado civil: solteira
  Sandro -- ano de nascimento: 1992 - sexo: indefinido - estado civil: solteiro/a - estrangeiro/a
```

Este é o momento oportuno para diferenciarmos dados de classes de dados de módulos, bem como funções de métodos.

Dados de classe são instanciados com valores, potencialmente distintos, toda vez que um novo objeto da classe é criado (repare nos atributos dos 8 alunos criados pela função cadastrar_alunos). As funções de uma classe que manipulam os dados específicos de um dado objeto, como por exemplo __init__ e __str__, são chamadas de métodos, no contexto do paradigma orientado a objetos. Observe que quando um objeto é criado a partir do seu construtor (que tem o mesmo nome da classe do objeto sendo criado), o método __init__ é executado implicitamente e, por esse motivo, o construtor deve receber argumentos para todos os parâmetros declarados no método __init__. Por sua vez, o método __str__ gera um string a partir dos dados de um objeto específico. Ou seja, métodos manipulam dados referentes a um dado objeto, por exemplo, os dados da aluna 'Ana Julia', cadastrada na função cadastrar_alunos, do módulo cadastro_alunos.

No entanto, para definir conjunto de objetos, precisamos definir variáveis de módulo, e não de classes; como ocorre, por exemplo, com a lista alunos definida no módulo aluno. Da mesma forma, para manipular um conjunto de objetos, precisamos funções no módulo e não métodos nas classes; como ocorre com as funções get_alunos e inserir_aluno, que manipulam a lista alunos.

De maneira geral, variáveis e funções definidas em um módulo, por ser acessadas por qualquer outro módulo, desde que importadas; enquanto dados e métodos de um objeto, precisam ser acessados a partir de objeto específico. Por esse motivo, se comportam como variáveis e funções globais, acessíveis fora do escopo de objetos.

A partir do Tutorial 3, vamos mapear várias entidades do mundo real em classes da aplicação. Por esse motivo, foi implementada a função imprimir_objetos, que servirá para imprimir objetos criados a partir de várias classes. No parâmetro cabeçalho é passado como argumento o string da mensagem que será mostrada, informando os objetos que estão sendo impressos e, adicionalmente, se esses são objetos foram cadastrados (ilustrados nesta seção) ou selecionados a partir de filtros (ilustrados na próxima seção). No parâmetro objetos, será passada como argumento uma lista de objetos. O parâmetro filtros é inicializado como None, de forma que não será preciso passar nenhum argumento para esse parâmetro, para imprimir objetos cadastrados. No entanto, para imprimir objetos selecionados a partir de filtros (próxima seção), será impresso o string que representa os filtros utilizados na seleção. Lembre-se, como vimos na subseção 4.2 do Tutorial 1, que a função enumerate, utilizada em um comando for retorna o índice e o valor dos elementos de uma lista. O índice está sendo somado com 1 antes de ser impresso, porque o inicial de uma lista é 0 e não 1.

Observe que em função dos métodos de conversão para strings que foram implementados como métodos auxiliares do método __str__ da classe Aluno, a impressão dos atributos é amigável para a interface: (a) o sexo está utilizando um string completo, e não apenas o caracter F ou M; (b) os gêneros do estado civil e do estrangeiro concordam com o sexo do aluno; e (c) a informação de estrangeiro é omitida para alunos que não são estrangeiros. Para o último aluno cadastrado ('Sandro') foi atribuído o sexo 'indefinido' somente com o objetivo de mostrar como fica a visualização de estado civil e estrangeiro neste caso.

3 – Restrições de Acesso a Dados e Métodos de Objetos

Na linguagem orientada a objetos Java, são utilizadas palavras reservadas como modificadores para restringir o acesso a dados e métodos de uma classe, a partir de um objeto da classe. Se você definir um dado em uma classe, e informar com a palavra reservada private que esse dado é privado, não será possível ler ou alterar esse dado diretamente a partir de um objeto da classe. Da mesma forma, se você definir um método como privado, não será possível chamá-lo a partir de um objeto da classe.

Em Python, em vez de utilizar palavras reservadas para restringir o acesso a dados e métodos de uma classe, caso você queira restringir o acesso a um dado ou método, você deverá prefixar o nome do dado ou do método pelo string '__'. Na definição ilustrada para a classe Aluno, você pode alterar o nome do aluno diretamente. Suponha a primeira aluna cadastrada, com o nome de Ana Julia, você poderá alterar seu nome diretamente da seguinte forma:

```
aluna = get_alunos()[0]; # obtendo a primeira aluna cadastrada na lista alunos
aluna.nome = 'Ana Julia Andrade' # alterando diretamente o dado nome do objeto da classe Aluno
print(aluna.nome) # a impressão será : Ana Julia Andrade
```

No entanto, se você redefinir o dado nome, da classe Aluno, para __nome, você não conseguirá mais alterar diretamente esse dado. Ou seja, o comando aluna.__nome = 'Ana Julia Andrade', não terá nenhum efeito, mantendo o nome original 'Ana Julia'.

De forma equivalente, se você redefinir o nome do método de uma classe com o prefixo '__', esse método não poderá ser acessado por um objeto da classe. Suponha por exemplo que você definiu na classe Aluno o método: def get_aluno(self): return self.aluno. Neste caso, você poderá utilizar esse método com qualquer objeto da classe Aluno, por exemplo, com o comando: nome = aluna.get_nome(). No entanto, se você redefinir o nome do método para __get_nome e tentar executá-lo a partir de um objeto da classe, por exemplo, utilizando o comando nome = aluna.__get_nome(), você terá como resultado uma mensagem de erro informando que a execução falhou.

Não vamos utilizar restrição de acesso a dados e métodos nos tutorias dessa disciplina, no entanto, é importante que você saiba como fazê-lo.

4 – Estruturas de Dados e Classes Genéricas no Contexto de Tipagem Dinâmica

Linguagens orientadas a objeto, como Java e C++, tem tipagem estática. Ou seja, para declarar uma variável ou um parâmetro você precisa informar previamente o seu tipo, e para todo comando que altere o valor da variável ou que passe um argumento para o parâmetro, o compilador da linguagem irá checar se o valor atribuído pertence ao tipo previamente especificado.

Python é uma linguagem orientada a objetos com tipagem dinâmica. Ou seja, você não declara o tipo de uma variável ou parâmetro ao definí-lo. O tipo da variável ou parâmetro será inferido quando um valor for atribuído; e mais, você pode atribuir um valor de um dado tipo e posteriormente atribuir um valor de outro tipo para a mesma variável.

Com a tipagem dinâmica, você perde a capacidade do compilador de verificar se os valores atribuídos a variáveis ou parâmetros pertencem a um dado tipo, especificado previamente; então, você deve estar mais atento à implementação do seu código. Em contrapartida, você ganha muito em flexibilidade. Por exemplo, uma mesma função que soma dois números poderá ser executada para somar valores do tipo int ou float. Indo mais além, quando você define conjuntos como uma lista ou um dicionário, você pode inserir objetos de classes distintas nesses conjuntos, ou até valores que não sejam objetos.

Em linguagens fortemente tipadas, as estruturas de dados são genéricas, porque podem ser definidas com base na classe, cujo objetos serão utilizados para povoar a estrutura de dados. De forma semelhante, existe o conceito de classe genérica que pode ser definida baseada em tipos em aberto (ex: T1 e T2) para utilização nas suas definições de dados e métodos. Antes de utilizar a classe genérica você precisará definir os tipos que haviam ficado em aberto; no entanto, a partir da mesma classe genérica, você poderá facilmente gerar novas classes simplesmente por atribuir várias combinações de diferentes tipos.

Neste ponto, podemos constatar a versatilidade de linguagens com tipagem dinâmica, como Python. Você simplesmente não necessita definir previamente uma lista a partir de um dado tipo e nem uma classe genérica baseada em tipos em aberto. Simplesmente, os tipos já são naturalmente genéricos e serão inferidos, quando os valores forem atribuídos.

5 - Utilizando Filtros para Selecionar um Subconjunto de Objetos

Para finalizar esse tutorial, vamos aprender como selecionar um subconjunto dos alunos cadastrados que atendam determinados filtros. No módulo aluno vamos acrescentar a função selecionar_alunos.

```
def selecionar_alunos(estado_civil = None, estrangeiro = None, ano_nascimento_mínimo = None):
    filtros = 'Filtros: '
    if estado_civil != None: filtros += 'estado civil: ' + estado_civil
    if estrangeiro: filtros += ' - estrangeiro'
    elif estrangeiro == False: filtros += ' - não estrangeiro'
    if ano_nascimento_mínimo != None: filtros += ' - ano de nascimento mínimo: '
        + str(ano_nascimento_mínimo)
    alunos_selecionados = []
    for aluno in alunos:
        if estado_civil != None and aluno.estado_civil != estado_civil: continue
        if ano_nascimento_mínimo != None and aluno.ano_nascimento < ano_nascimento_mínimo: continue
        if estrangeiro in (True, False) and aluno.estrangeiro != estrangeiro: continue
        alunos_selecionados.append(aluno)
    return filtros, alunos_selecionados
```

O parâmetro alunos, receberá como argumento uma lista de objetos cadastrados. Os três últimos parâmetros definem os filtros que serão utilizados para seleção. Os filtros são opcionais. Caso o valor do filtro não seja passado como argumento, já está previamente atribuído a None, para indicar que o implementador não deseja utilizar esse filtro na seleção.

Neste ponto, precisamos observar a versatilidade da tipagem dinâmica de Python: o tipo do filtro ano_nascimento_mínimo poderá ser inicializado com valor None e posteriormente ser atribuído a um valor inteiro, sem nenhum problema. No entanto, esse facilidade só será utilizada para inicialização prévia com None para padronizar filtros não obrigatórios, mas você não deverá alterar os tipos dos valores atribuídos a uma mesma variável para não tornar o seu código mais difícil de ser entendido e testado.

Na parte inicial da função é construído um string para concatenar os valores dos filtros que forem passados como argumentos. Na parte final da função, é definida a variável alunos_selecionados, inicializada com uma lista vazia. A seguir, é definido o loop de seleção dos alunos cadastrados que serão testados para verificar se atendem os filtros passados como argumento. O objeto aluno que atenter a todos filtros será inserido na lista alunos_selecionados. Ao final, a função irá retornar o string com os filtros solicitados (com valores diferentes de None) e a lista de alunos selecionados.

Agora, vamos entender como funciona o loop de seleção dos alunos. Para cada aluno da lista de alunos cadastrados, são checados os filtros, cujos valores foram passados pelo implementador. Para filtros com valor do tipo str, int, e float, os filtros são checados como obrigatórios verificando se são diferentes de None. Filtros do tipo bool, são checados como obrigatórios verificando se o valor passado pertence à tupla (True, False). Caso o filtro seja obrigatório, é checado se o filtro não é atendido e neste caso é utilizado o comando continue que faz com que a execução retorne ao início do loop para testar o próximo objeto aluno: o teste do aluno é interrompido (por que basta não atender um dos filtros, para ser descartado como selecionado), sem incluir o aluno testado na lista de selecionados.

Para testar o atendimento a um dado filtro, o atributo do aluno é verificado como diferente do valor do filtro, para executar o comando continue e interromper o teste do aluno, caso o atributo não corresponda ao valor do filtro. No caso de um filtro com valor do tipo int ou float, para o qual seja definido um valor de filtro como mínimo ou máximo será feita uma comparação para verificar que o filtro não atende esse valor mínimo (o atributo do objeto não pode ser inferior ao valor mínimo fo filtro) ou máximo (ou o atributo do objeto não pode ser superior ao valor máximo do filtro). A estratégia de utilizar o comando continue para testar se o filtro não é atendido, simplica a legibilidade do código gerado, definindo um único teste para cada filtro, sem a necessidade de utilizar encadeamentos de operadores booleanos (and e or) que tornariam a expressão mais complexa. Observe como ficaria o loop de seleção se não utilizassemos o comando continue:

```
for aluno in alunos:
    if (estado_civil == None or aluno.estado_civil == estado_civil)\
        and (ano_nascimento_mínimo == -1 or aluno.ano_nascimento >= ano_nascimento_mínimo)\
        and (estrangeiro not in (True, False) or aluno.estrangeiro == estrangeiro):
        alunos_selecionados.append(aluno)
```

Nesta alternativa de implementação, seria verificado para todos os filtros, se cada filtro não é obrigatório ou se a condição do filtro é atendida. Se todos os filtros forem atendidos, o objeto aluno é inserido na lista de selecionados.

Um boa estratégia utilizada para testar o funcionamento correto da seleção de alunos é começar não utilizando nenhum filtro como obrigatório (o que levará à seleção de toda a lista de alunos cadastrados) e ir acrescentando um filtro obrigatório de cada vez, mantendo os valores dos filtros utilizados anteriormente, de forma que a lista de alunos cadastrados diminua a medida que cada filtro adicional é tornado obrigatório. Vamos utilizar essa estratégia nos quatro tutoriais desta disciplina.

Finalmente, vamos acrescentar combinações de filtragem de objetos da classe Aluno ao módulo cadastro_alunos. Cada vez que um novo filtro é passado com argumento, esse argumento é atribuído ao nome do parâmetro correspondente na função selecionar_alunos. Como já vimos anteriormente, o nome do parâmetro é uma opção para tornar mais legível a passagem dos argumentos para uma dada função.

```
if __name__ == '__main__':
    cadastrar_alunos()
    imprimir_objetos('Alunos Cadastrados', get_alunos())
    filtros, alunos_selecionados = selecionar_alunos()
    imprimir_objetos('Alunos Selecionados com ', alunos_selecionados, filtros)
    filtros, alunos_selecionados = selecionar_alunos(estado_civil='solteiro')
    imprimir_objetos('Alunos Selecionados com ', alunos_selecionados, filtros)
    filtros, alunos_selecionados = selecionar_alunos('solteiro', estrangeiro=False)
    imprimir_objetos('Alunos Selecionados com ', alunos_selecionados, filtros)
    filtros, alunos_selecionados = selecionar_alunos('solteiro', False, ano_nascimento_mínimo=1992)
    imprimir_objetos('Alunos Selecionados com ', alunos_selecionados, filtros)
```

A saída da execução final da aplicação é a seguinte:

```
Alunos Cadastrados
1 - Ana Julia -- ano de nascimento: 1995 - sexo: feminino - estado civil: solteira
2 - Joaquim -- ano de nascimento: 1990 - sexo: masculino - estado civil: casado - estrangeiro
3 - Ana Ligia -- ano de nascimento: 1998 - sexo: feminino - estado civil: solteira
4 - Mateus -- ano de nascimento: 1991 - sexo: masculino - estado civil: solteiro
5 - Livia -- ano de nascimento: 1985 - sexo: feminino - estado civil: casada - estrangeira
6 - Roberto -- ano de nascimento: 1990 - sexo: masculino - estado civil: casado
7 - Ana Maria -- ano de nascimento: 1991 - sexo: feminino - estado civil: solteira
8 - Sandro -- ano de nascimento: 1992 - sexo: indefinido - estado civil: solteiro/a - estrangeiro/a

Alunos Selecionados com Filtros:
```

```
1 - Ana Julia -- ano de nascimento: 1995 - sexo: feminino - estado civil: solteira
2 - Joaquim -- ano de nascimento: 1990 - sexo: masculino - estado civil: casado - estrangeiro
3 - Ana Ligia -- ano de nascimento: 1998 - sexo: feminino - estado civil: solteira
4 - Mateus -- ano de nascimento: 1991 - sexo: masculino - estado civil: solteiro
5 - Livia -- ano de nascimento: 1985 - sexo: feminino - estado civil: casada - estrangeira
6 - Roberto -- ano de nascimento: 1990 - sexo: masculino - estado civil: casado
7 - Ana Maria -- ano de nascimento: 1991 - sexo: feminino - estado civil: solteira
8 - Sandro -- ano de nascimento: 1992 - sexo: indefinido - estado civil: solteiro/a - estrangeiro/a

Alunos Selecionados com Filtros: estado civil: solteiro
1 - Ana Julia -- ano de nascimento: 1995 - sexo: feminino - estado civil: solteira
2 - Ana Ligia -- ano de nascimento: 1998 - sexo: feminino - estado civil: solteira
3 - Mateus -- ano de nascimento: 1991 - sexo: masculino - estado civil: solteiro
4 - Ana Maria -- ano de nascimento: 1991 - sexo: feminino - estado civil: solteira
5 - Sandro -- ano de nascimento: 1992 - sexo: indefinido - estado civil: solteiro/a - estrangeiro/a

Alunos Selecionados com Filtros: estado civil: solteiro - não estrangeiro
1 - Ana Julia -- ano de nascimento: 1995 - sexo: feminino - estado civil: solteira
2 - Ana Ligia -- ano de nascimento: 1998 - sexo: feminino - estado civil: solteira
3 - Mateus -- ano de nascimento: 1991 - sexo: masculino - estado civil: solteiro
4 - Ana Maria -- ano de nascimento: 1991 - sexo: feminino - estado civil: solteira

Alunos Selecionados com Filtros: estado civil: solteiro - não estrangeiro
    - ano de nascimento mínimo: 1992
1 - Ana Julia -- ano de nascimento: 1995 - sexo: feminino - estado civil: solteira
2 - Ana Ligia -- ano de nascimento: 1998 - sexo: feminino - estado civil: solteira
```

Observe que a estratégia escolhida para testar o funcionamento da seleção de alunos facilita bastante a verificação da utilização bem sucedida da adição de cada filtro como obrigatório. Inicialmente, sem nenhum filtro obrigatório a lista de selecionados é equivalente à lista de cadastrados e a medida que cada filtro adicional é tornado obrigatório, a lista de selecionados diminui pelo menos de um objeto da classe Aluno. Obviamente, para obter esse efeito, que facita o teste da seleção de alunos, você precisará escolher adequadamente os valores dos atributos dos objetos cadastrados e os valores dos filtros obrigatórios.

Exercícios

Exercício 1
Implemente a aplicação Cadastro de Disciplinas:
a) cadastrando objetos da classe entidade Disciplina, com os seguintes atributos: título, carga horária total, carga horária prática, categoria (obrigatória, optativa, eletiva), reuni (sim ou não);
b) e selecionando as disciplinas cadastradas com os seguintes filtros: carga teórica mínima, categoria, reuni.

Exercício 2
Implemente uma aplicação de cadastro de sua escolha atendendo os seguintes requisitos:
a) implementando uma classe baseada em uma entidade, utilizando pelo menos: um atributo do tipo str e bool;
b) e definindo os filtros que julgar necessário, incluido pelo menos um filtro baseado no atributo str e o outro baseado no atributo de tipo bool.