TÉCNICO LISBOA

Fundamentos de Programação - 2019/2020 Aula Prática 12 (30 minutos) Turno 5ª feira 10:30-12:00
Nome:
Número:
Data:
Curso:

Capítulo 11 - Programação orientada a objectos

Suponha que quer representar o conceito de tempo de um *relógio*, dividindo-o em horas, minutos e segundos. No tipo *relógio*, o número de minutos e de segundos está compreendido entre `0` e `59` e o número de horas está compreendido entre `0` e `24`. Por exemplo `12:00:00` é uma representação válida do tipo *relógio*. Podemos considerar as seguintes operações básicas para o tipo *relógio*:

- `obter_horas`, que retorna a componente as horas do relógio;
- `obter_minutos`, que retorna a componente os minutos do relógio;
- `obter_segundos`, que retorna a componente os segundos do relógio;
- `cria_copia_relogio`, que retorna uma cópia do relógio;
- Defina também a representação do relógio, seguindo os exemplos abaixo.

Mostra-se a seguir um exemplo de interação:

```
>>> r = relogio(20, 15, 50)
>>> r
20:15:50
>>> r.obter_horas()
20
>>> r.obter_minutos()
15
>>> r.obter_segundos()
50
>>> r2 = r.cria_copia_relogio()
>>> r2
20:15:50
```

Solução:

```
class relogio:
    def __init__(self, horas, minutos, segundos):
        self.horas = horas
        self.minutos = minutos
        self.segundos = segundos

    def obter_horas(self):
        return self.horas

    def obter_minutos(self):
        return self.minutos

    def obter_segundos(self):
        return self.segundos

    def cria_copia_relogio(self):
        return relogio(self.horas, \
                       self.minutos, \
                       self.segundos)

    def __repr__(self):
        return "{:02d}".format(self.horas) + \
         ":" + "{:02d}".format(self.minutos) + \
         ":" + "{:02d}".format(self.segundos)
```

TÉCNICO LISBOA

Fundamentos de Programação - 2019/2020 Aula Prática 10 (30 minutos) Turno 4ª feira 08:00-09:30
Nome:
Número:
Data:
Curso:

Capítulo 11 - Programação orientada a objectos

Suponha que deseja criar o tipo *vetor* de um espaço 3D em Python. Um vetor num referencial cartesiano pode ser representado pelas coordenadas da sua extremidade (x,y,z), estando a sua origem no ponto (0, 0, 0). Podemos considerar as seguintes operações básicas para vetores:

- `obter_x`, que retorna a componente x da extremidade do vetor;
- `obter_y`, que retorna a componente y da extremidade do vetor;
- `obter_z`, que retorna a componente z da extremidade do vetor;
- `vetores_iguais` que recebe dois vetores e retorna `True` caso os vetores sejam iguais e `False`, caso contrário.
- Defina também a representação do vetor, como nos exemplos abaixo.

Mostra-se a seguir um exemplo de interação:

```
>>> v1 = vetor(1,2,3)
>>> v1
(1, 2, 3)
>>> v1.obter_x()
1
>>> v1.obter_y()
1
>>> v1.obter_z()
1
>>> v1 = vetor(1,2,3)
>>> v1.obter_x()
1
>>> v1.obter_y()
2
>>> v1.obter_z()
3
>>> v2 = vetor(1,2,4)
>>> vetores_iguais(v1,v2)
False
```

Solução:

```
class vetor:
    def __init__(self, x, y, z):
        self.x = x
        self.y = y
        self.z = z

    def obter_x(self):
        return self.x

    def obter_y(self):
        return self.y

    def obter_z(self):
        return self.z

    def __repr__(self):
        return "(" + str(self.x) + ", " + \
                     str(self.y) + ", " + \
                     str(self.z) + ")"


#Funcao externa
def vetores_iguais(v1, v2):
    return v1.obter_x() == v2.obter_x() \
       and v1.obter_y() == v2.obter_y() \
       and v1.obter_z() == v2.obter_z()
```

TÉCNICO LISBOA

Fundamentos de Programação - 2019/2020 Aula Prática 12 (30 minutos) Turno 2ª feira 09:00-10:30	
Nome:	
Número:	
Data:	
Curso:	

Capítulo 11 - Programação orientada a objectos

Crie a classe *hotel* cujo construtor recebe o preço de um quarto individual e o preço de um quarto duplo. No hotel é permitido fazer a reserva de quartos, numa reserva é dado o número de quartos individuais e duplos que se pretendem reservar e é devolvido o custo dessa reserva.
Defina os seguintes métodos da classe *hotel*:

- `preco_individual`, que retorna o preço de um quarto individual;
- `preco_duplo`, que retorna o preço de um quarto duplo;
- `reserva` que retorna o custo de uma reserva efetuada no hotel;
- Defina também a representação do hotel, como nos exemplos abaixo.

Mostra-se a seguir um exemplo de interação:

```
>>> h = hotel(20, 35)
>>> h
Individual: 20, duplo: 35
>>> h.preco_individual()
20
>>> h.preco_duplo()
35
>>> h.reserva(3,1)
95
```

Solução:

```
class hotel:
    def __init__(self, preco_individual, preco_duplo):
        self.individual = preco_individual
        self.duplo = preco_duplo

    def preco_individual(self):
        return self.individual

    def preco_duplo(self):
        return self.duplo

    def reserva(self, unidades_individual, unidades_duplo):
        return self.individual * unidades_individual + self.duplo *
unidades_duplo

    def __repr__(self):
        return "Individual: " + str(self.individual) + ", duplo: "
+ str(self.duplo)
```

TÉCNICO LISBOA

Fundamentos de Programação - 2019/2020 Aula Prática 12 (30 minutos) Turno 2ª feira 10:30-12:00
Nome:
Número:
Data:
Curso:

Capítulo 11 - Programação orientada a objectos

Crie a classe *hotel* cujo construtor recebe a lotação máxima do hotel criado. No hotel é possível fazer check-in quando um hóspede entra e check-out quando um hóspede sai. Não é possível ter mais hospedes que a lotação máxima nem um número de hóspedes inferior a zero. Inicialmente é criado sem hóspedes hospedados. Defina os seguintes métodos da classe *hotel*:

- `check_in`, que adiciona um hóspede ao hotel;
- `check_out`, que remove um hóspede ao hotel;
- `lotacao_maxima`, que retorna a capacidade máxima do hotel;
- `lotacao`, que retorna a lotação ocupado por hóspedes;
- Defina também a representação do hotel, como nos exemplos abaixo.

Mostra-se a seguir um exemplo de interação:

```
>>> h = hotel(10)
>>> h.lotacao_maxima()
10
>>> h.check_in()
>>> h.check_in()
>>> h.check_in()
>>> h.lotacao()
3
>>> h.check_out()
>>> h
Lotacao 2 de 10 hospedes
```

Solução:

```
class hotel:
    def __init__(self, capacidade):
        self.capacidade = capacidade
        self.ocupados = 0

    def check_in(self):
        if self.ocupados < self.capacidade:
            self.ocupados += 1

    def check_out(self):
        if self.ocupados > 0:
            self.ocupados -= 1

    def lotacao(self):
        return self.ocupados

    def lotacao_maxima(self):
        return self.capacidade

    def __repr__(self):
        return "Lotacao " + str(self.lotacao()) + " de " +
str(self.lotacao_maxima()) + " hospedes."
```

TÉCNICO LISBOA

Fundamentos de Programação - 2019/2020 Aula Prática 12 (30 minutos) Turno 6ª feira 12:30-14:00	
Nome:	
Número:	
Data:	
Curso:	

Capítulo 11 - Programação orientada a objectos

Crie a classe *produto* cujo construtor recebe o nome de um produto representado por uma cadeia de caracteres, um código representado por um inteiro e um preço representado por um float. Defina os seguintes métodos:

- `nome`, que devolve o nome do produto;
- `codigo`, que devolve o código do produto;
- `custo`, que devolve o custo do produto;
- Defina também a representação de um produto de acordo com os exemplos abaixo.

Defina uma função `produto_mais_caro` que recebe dois produtos e retorna o produto mais caro, em caso de custos iguais, retorna o primeiro produto.

```
>>> p = produto("banana", 11111, 1.35)
>>> p
Produto: banana, codigo: 11111, custo: 1.35
>>> p.nome()
'banana'
>>> p.custo()
1.35
>>> p.codigo()
11111
>>> p2 = produto("pera", 11222, 1.55)
>>> produto_mais_caro(p, p2)
Produto: pera, codigo: 11222, custo: 1.55
```

Solução:

```
class produto:
    def __init__(self, nome, codigo, custo):
        self.name = nome
        self.code = codigo
        self.price = custo

    def nome(self):
        return self.name

    def custo(self):
        return self.price

    def codigo(self):
        return self.code

    def __repr__(self):
        return "Produto: " + self.nome() + \
                ", codigo: " + str(self.codigo()) + \
                ", custo: " + str(self.custo())

    #Funcao externa
def produto_mais_caro(p1, p2):
    if p1.custo() >= p2.custo():
        return p1
    else:
        return p2
```

TÉCNICO LISBOA

Fundamentos de Programação - 2019/2020 Aula Prática 12 (30 minutos) Turno 6ª feira 14:00-15:30	
Nome:	
Número:	
Data:	
Curso:	

Capítulo 11 - Programação orientada a objectos

Crie a classe *produto* cujo construtor recebe o nome de um produto representado por uma cadeia de caracteres, um código representado por um inteiro e um preço representado por um float. Defina os seguintes métodos:

- `nome`, que devolve o nome do produto;
- `codigo`, que devolve o código do produto;
- `custo`, que devolve o custo do produto;
- Defina também a representação de um produto de acordo com os exemplos abaixo.

Define uma função `produtos_iguais` que recebe dois produtos e devolve `True` caso os produtos sejam iguais e `False`, caso contrário.

```
>>> p = produto("banana", 11111, 1.35)
>>> p
Produto: banana, codigo: 11111, custo: 1.35
>>> p.nome()
'banana'
>>> p.custo()
1.35
>>> p.codigo()
11111
>>> p2 = produto("pera", 11222, 1.55)
>>> produtos_iguais(p, p2)
False
```

Solução:

```
class produto:
    def __init__(self, nome, codigo, custo):
        self.name = nome
        self.code = codigo
        self.price = custo

    def nome(self):
        return self.name

    def custo(self):
        return self.price

    def codigo(self):
        return self.code

    def __repr__(self):
        return "Produto: " + self.nome() + ", codigo: " +
str(self.codigo()) + ", custo: " + str(self.custo())


    #Funcao externa
def produtos_iguais(p1, p2):
    return p1.custo()==p2.custo() and p1.nome() == p2.nome() and
p1.codigo()== p2.codigo()
```

TÉCNICO LISBOA

Fundamentos de Programação - 2019/2020 Aula Prática 12 (30 minutos) Turno 6ª feira 15:30-17:00
Nome:
Número:
Data:
Curso:

Capítulo 11 - Programação orientada a objectos

Crie a classe *comboio* cujo construtor recebe a capacidade de um comboio em número de passageiros. O comboio inicialmente é criado vazio. Os outros métodos suportados pela classe são:

- `capacidade`, que devolve a capacidade total do comboio;
- `passageiros`, que devolve o número de passageiros presentes no comboio;
- `sai`, que recebe o número de passageiros a sair do comboio. Se o número exceder o número de passageiros presentes no comboio, o número de passageiros presentes passa a ser 0;
- `entra`, que recebe o número de passageiros a entrar no comboio. Se o número de passageiros a entrar no comboio fizer com que a sua capacidade seja ultrapassada, o número de passageiros presentes no comboio passa a ser igual à sua capacidade.

Mostra-se a seguir um exemplo de interação:

```
>>> c = comboio(30)
>>> c.passageiros()
0
>>> c.capacidade()
30
>>> c.sai(35)
>>> c.passageiros()
0
>>> c.entra(40)
>>> c.passageiros()
30
>>> c.sai(5)
>>> c.passageiros()
25
>>> c.sai(26)
>>> c.passageiros()
0
```

Solução:

```
class comboio:
    def __init__(self, limite):
        self.limite = limite
        self.lotacao = 0

    def capacidade(self):
        return self.limite

    def passageiros(self):
        return self.lotacao

    def sai(self, passageiros):
        if self.lotacao >= passageiros:
            self.lotacao -= passageiros
        else:
            self.lotacao = 0

    def entra(self,passageiros):
        if self.lotacao + passageiros <= self.limite:
            self.lotacao += passageiros
        else:
            self.lotacao = self.limite
```