TÉCNICO LISBOA

Fundamentos de Programação - 2020/2021 Aula Prática 11 (30 minutos) Turno 2ª feira 9:00-10:30
Nome:
Número:
Data:
Curso:

Capítulo 11 - Programação orientada a objectos

Crie a classe *estacionamento* cujo construtor recebe a lotação máxima do estacionamento criado. No estacionamento é possível entrar e sair de um carro. Não é possível ter mais carros que a lotação máxima nem o número de carros ser inferior a zero. Inicialmente é criado sem carros estacionados. Defina os seguintes métodos da classe *estacionamento*:

- `entrar`, que adiciona um carro ao estacionamento;
- `sair`, que remove um carro do estacionamento;
- `lotacao_maxima`, que retorna a capacidade máxima do estacionamento;
- `lotacao`, que retorna o número de carros dentro do estacionamento;

Defina também a representação do estacionamento, como nos exemplos abaixo. Mostra-se a seguir um exemplo de interação:

```
>>> e = estacionamento(50)
>>> e.lotacao_maxima()
50
>>> e.entrar()
>>> e.entrar()
>>> e.entrar()
>>> e.lotacao()
3
>>> e.sair()
>>> e
Lotacao: 2 de 50 lugares
```

Solução:

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

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

    def sair(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()) + " lugares."
```

TÉCNICO LISBOA

Fundamentos de Programação - 2020/2021 Aula Prática 11 (30 minutos) Turno 2ª feira 10:30-12:00
Nome:
Número:
Data:
Curso:

Capítulo 11 - Programação orientada a objectos

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

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

Defina também a representação do autocarro, como nos exemplos abaixo. Mostra-se a seguir um exemplo de interação:

```
>>> a = autocarro(30)
>>> a.passageiros()
0
>>> a.capacidade()
30
>>> a.sai(35)
>>> a.passageiros()
0
>>> a.entra(40)
>>> a.passageiros()
30
>>> a.sai(5)
>>> a.passageiros()
25
>>> a
Autocarro de 30 lugares com 25 passageiros.
>>> a.sai(26)
>>> a.passageiros()
0
```

Solução:

```
class autocarro:
    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

    def __repr__(self):
         return "Autocarro de " + str(self.capacidade()) + " lugares
com " + str(self.passageiros()) + " passageiros."
```

TÉCNICO LISBOA

Fundamentos de Programação - 2020/2021 Aula Prática 11 (30 minutos) Turno 5ª feira 10:30-12:00
Nome:
Número:
Data:
Curso:

Capítulo 11 - Programação orientada a objectos

Suponha que desejava representar o conceito `data`, em que uma data é caracterizada por um `dia` (um inteiro entre 1 e 31), um `mês` (um inteiro entre 1 e 12) e um `ano` (um inteiro que pode ser positivo, nulo ou negativo). Ignore os anos bissextos e os dias de cada mês. O tipo `data` suporta os seguintes métodos:

- `dia`, `mes` e `ano` que devolvem, respectivamente o dia, o mês e o ano da data.
- `anterior` que recebe como argumento uma data e tem o valor verdadeiro se a data da própria instância é anterior ao argumento e falso caso contrário.

Defina também a representação do data como `dd/mm/aaaa` (em que `dd` representa o dia, `mm` o mês e `aaaa` o ano). Mostra-se a seguir um exemplo de interação:

```
>>> d = data(5, 1, 2021)
>>> d
05/01/2021
>>> d.dia()
5
>>> d.mes()
1
>>> d.ano()
2021
>>> d.anterior(data(6, 1, 2021))
True
```

Solução:

```
class data:
    def __init__(self, d, m, a):
        if isinstance(d, int) and 1 <= d <= 31 and \
           isinstance(m, int) and 1 <= m <= 12 and \
           isinstance(a, int):
            self.d = d
            self.m = m
            self.a = a
        else:
            raise ValueError('data: argumentos errados')

    def dia(self):
        return self.d

    def mes(self):
        return self.m

    def ano(self):
        return self.a

    def __repr__(self):
        return '{:02d}/{:02d}/{:04d}'.format(self.dia(),
                                             self.mes(), self.ano())

    def anterior(self, outro):
        if self.a < outro.ano():
            return True
        elif self.a == outro.ano() and self.m < outro.mes():
            return True
        elif self.a == outro.ano() and \
             self.m == outro.mes() and self.d < outro.dia():
            return True
        return False
```

TÉCNICO LISBOA

Fundamentos de Programação - 2020/2021 Aula Prática 11 (30 minutos) Turno 5ª feira 14:00-15:30
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*. O tipo *relógio* deve suportar os seguintes métodos:

- `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;
- `mais_cedo`, que recebe outra instância de relógio e tem o valor verdadeiro se o próprio relógio marxar uma hora anterior que o argumento e falso caso contrário;

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
>>> r.mais_cedo(relogio(21, 15, 50))
True
```

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 mais_cedo(self, other):
        return (self.horas*3600 + 60*self.minutos +
self.segundos) < (other.horas*3600 + 60*other.minutos +
other.segundos)

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

TÉCNICO LISBOA

Fundamentos de Programação - 2020/2021 Aula Prática 11 (30 minutos) Turno 6ª feira 10:00-11:30
Nome:
Número:
Data:
Curso:

Capítulo 11 - Programação orientada a objectos

Defina a classe `contador_limitado` cujo construtor recebe dois números inteiros, correspondendo ao limite inferior e superior do contador. O contador quando é criado tem como valor inicial o limite inferior. Os outros métodos suportados pela classe são:

- `consulta`, que devolve o valor do contador;
- `inc`, que permite incrementar de uma unidade o valor do contador e devolve o valor do contador no final. Se se tentar incrementar o valor do contador para cima do limite superior este não é alterado;
- `dec`, que permite decrementar de uma unidade o valor do contador e devolve o valor do contador no final. Se se tentar decrementar o valor do contador para baixo do limite inferior este não é alterado.

Defina também a representação do `contador_limitado`, seguindo os exemplos abaixo. Mostra-se a seguir um exemplo de interação:

```
>>> c1 = contador_limitado(3, 5)
>>> c1.inc()
4
>>> c1.consulta()
4
>>> c1.inc()
5
>>> c1.inc()
5
>>> c1.dec()
4
>>> c1.dec()
3
>>> c1.dec()
3
>>> c1
Contador de 3 a 5 (atual 3)
```

Solução:

```
class contador_limitado:
      def __init__(self, inf, sup):
            if isinstance(inf, int) and isinstance(sup, int) and \
               inf < sup:
                  self.inf = inf
                  self.sup = sup
                  self.contador = inf
            else:
                  raise ValueError('args invalidos')

      def consulta(self):
            return self.contador

      def inc(self):
            if self.contador < self.sup:
                  self.contador += 1
            return self.contador

      def dec(self):
            if self.contador > self.inf:
                  self.contador -= 1
            return self.contador

      def __repr__(self):
            return “Contador de ” + str(self.inf) + “ a ” +
str(self.sup) + “ (atual ” +  str(self.contador) + “)”
```

TÉCNICO LISBOA

Fundamentos de Programação - 2020/2021 Aula Prática 11 (30 minutos) Turno 6ª feira 14:00-15:30
Nome:
Número:
Data:
Curso:

Capítulo 11 - Programação orientada a objectos

Defina a classe `cartao_telefonico` cujo construtor recebe o tarifário em vigor. O tarifário é representado por um dicionário, em que cada tipo de chamada é representado por uma cadeia de caracteres a que está associado o custo por minuto de conversação, por exemplo:
`{'local':1, 'nacional':12, 'movel':20, 'internacional':41}`
Os outros métodos suportados pela classe são:

- `consulta_custo`, devolve os custos decorrentes das chamadas efetuadas;
- `consulta_chamadas`, devolve o número de chamadas efetuadas;
- `chamada`, efetua uma chamada, atualizando o valor dos custos. Recebe a tarifa e a duração da chamada em minutos.

Defina também a representação do `cartao_telefonico`, seguindo os exemplos abaixo.
Mostra-se a seguir um exemplo de interação:

```
>>> tarifario = {'local':1, 'movel':20, 'internacional':41}
>>> c1 = cartao_telefonico(tarifario)
>>> c1.consulta_custo()
0
>>> c1.chamada('local', 5)
>>> c1.consulta_custo()
5
>> c1.chamada('nacional', 10)
>>> c1
Despesa de 125 céntimos em 2 chamadas
```

Solução:

```
class cartao_telefonico:

    def __init__(self, tarifario):
        if isinstance(tarifario, dict):
           self.tar = tarifario
           self.gasto = 0
           self.chamadas = 0
        else:
           raise ValueError('cartao_telfonico: argumento invalido')

    def consulta_custo(self):
        return self.gasto

    def consulta_chamadas(self):
        return self.chamadas

    def chamada(self, tipo, durac):
        if tipo in self.tar and \
           isinstance(durac, int) and durac >= 0:
                custo = durac * self.tar[tipo]
                self.gasto = self.gasto + custo
                self.chamadas += 1
        else:
           raise ValueError('chamada: argumento invalido')

    def __repr__(self):
        return "Despesa de " + str(self.gasto) +  " céntimos em "
+  str(self.chamadas) + " chamadas"
```

TÉCNICO LISBOA

Fundamentos de Programação - 2020/2021 Aula Prática 11 (30 minutos) Turno 6ª feira 15:30-17:00
Nome:
Número:
Data:
Curso:

Capítulo 11 - Programação orientada a objectos

Crie a classe *ski_rental* cujo construtor recebe o preço diário de aluguer de um par de skis normais e de um snowboard. Na loja é permitido fazer reservas antecipadas. Numa reserva é dado o número de skis e snowboards que se pretendem reservar e é devolvido o custo dessa reserva.
Defina os seguintes métodos da classe *ski_rental*:

- `preco_ski`, que retorna o preço de uns skis;
- `preco_snowboard`, que retorna o preço de um snowboard;
- `reserva` que retorna o custo de uma reserva efetuada na loja de aluguer;

Defina também a representação do *ski_rental*, como nos exemplos abaixo. Mostra-se a seguir um exemplo de interação:

```
>>> s = ski_renal(20, 35)
>>> s
Ski: 20; Snowboard: 35
>>> s.preco_ski()
20
>>> s.preco_snowboarsd()
35
>>> s.reserva(3,1)
95
```

Solução:

```
class ski_rental:
    def __init__(self, preco_ski, preco_board):
        self.ski = preco_ski
        self.board = preco_board

    def preco_ski(self):
        return self.ski

    def preco_snowboard(self):
        return self.board

    def reserva(self, unidades1, unidades2):
        return self.ski * unidades1 + self.board * unidades2

    def __repr__(self):
        return "Ski: " + str(self.ski) + "; Snowboard: " +
str(self.board)
```