TÉCNICO LISBOA

Fundamentos de Programação - 2018/2019 Aula Prática 12 (30 minutos) Turno 2ª feira 9:00-10:30
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 um valor acima 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 um valor abaixo do limite inferior este não é alterado.

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
```

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
```

TÉCNICO LISBOA

Fundamentos de Programação - 2018/2019 Aula Prática 10 (30 minutos) Turno 2ª feira 10:30-12:00
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`, devolve os custos decorrentes das chamadas efetuadas;
- `chamada`, efetua uma chamada, atualizando o valor dos custos. Recebe a tarifa e a duração da chamada em minutos.

Mostra-se a seguir um exemplo de interação:

```
>>> tarifario = {'local':1, 'movel':20, 'internacional':41}
>>> c1 = cartao_telefonico(tarifario)
>>> c1.consulta()
0
>>> c1.chamada('local', 5)
>>> c1.consulta()
5
```

Solução:

```
class cartao_telefonico:

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

    def consulta(self):
        return self.gasto

    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
        else:
           raise ValueError('chamada: argumento invalido')
```

TÉCNICO LISBOA

Fundamentos de Programação - 2018/2019 Aula Prática 12 (30 minutos) Turno 4ª feira 08:00-9:30
Nome:
Número:
Data:
Curso:

Capítulo 11 - Programação orientada a objectos

Defina uma classe `votacao` que corresponde a uma urna de uma votação. A sua classe deve receber a lista dos possíveis candidatos (representados como strings) e manter como estado interno o número de votos em cada candidato (iniciados a 0). Os outros métodos suportados pela classe são:

- `vota`, recebe um dos possíveis candidatos, aumentando o número de votos nesse candidato em um, se o candidato recebido não é válido o voto é contabilizado como nulo;
- `resultados`, deve apresentar os resultados da votação para os todos os candidatos e o número de votos nulos,
- `vencedor`, retorna o candidato ganhador da votação.

Mostra-se a seguir um exemplo de interação:

```
>> eleicoes = votacao(['ERC', 'JpC', 'PSC'])
>> eleicoes.vota('ERC')
>> eleicoes.vota('JpC')
>> eleicoes.vota('PSC')
>> eleicoes.vota('PSC')
>> eleicoes.vota('CUP')
>> eleicoes.resultados()
ERC 1
PSC 2
JpC 1
nulos 1
>> eleicoes.vencedor()
PSC
```

Solução:

```
class votacao:

    def __init__(self, candidatos):
        if isinstance(candidatos, (list,tuple)) and \
           len(candidatos) >= 1 and \
           all([type(c)==str for c in candidatos]):
               self.cand = {}
               self.nulos = 0
               for c in candidatos:
                   self.cand[c] = 0
        else:
           raise ValueError('votacao: argumento invalido')

    def vota(self, c):
        if c in self.cand:
            self.cand[c] = self.cand[c] + 1
        else:
            self.nulos = self.nulos + 1

    def resultados(self):
        for c in self.cand:
            print(c, self.cand[c])
        print('nulos', self.nulos)

    def vencedor(self):
        maior = 0
        for c in self.cand:
           if self.cand[c] > maior:
                maior = self.cand[c]
        vencedor = []
        for c in self.cand:
            if self.cand[c] == maior:
                vencedor += [c]
        if len(vencedor) == 1:
            return vencedor[0]
        else:
            return 'empate'
```

TÉCNICO LISBOA

Fundamentos de Programação - 2018/2019 Aula Prática 12 (30 minutos) Turno 5ª feira 10:30-12:00
Nome:
Número:
Data:
Curso:

Capítulo 11 - Programação orientada a objectos

Suponha que desejava criar a classe `data` em Python. 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.
A classe `data` tem as operações `dia`, `mes` e `ano` que devolvem, respectivamente o dia, o mês e o ano da data. A representação externa de uma data é `dd/mm/aaaa` (em que `dd` representa o dia, `mm` o mês e `aaaa` o ano).
a) Defina a classe `data`.
b) Defina a função `idade` que recebe como argumentos a data de nascimento de uma pessoa e outra data posterior e devolve a idade da pessoa na segunda data. Por exemplo:

```
>>> idade(data(2, 1, 2003), data(2, 1, 2005))
2
>>> idade(data(2, 1, 2003), data(2, 3, 2006))
3
```

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())

# função externa
def idade(d1, d2):
    a = d2.ano() - d1.ano()
    if d2.mes() < d1.mes():
        a = a - 1
    elif d2.mes() == d1.mes() & d2.dia() < d1.dia():
        a = a - 1
    return a
```

TÉCNICO LISBOA

Fundamentos de Programação - 2018/2019 Aula Prática 12 (30 minutos) Turno 6ª feira 12:30-14:00
Nome:
Número:
Data:
Curso:

Capítulo 11 - Programação orientada a objectos

Suponha que desejava criar a classe `relogio` em Python. Um relógio é caracterizado por um `hora` (um inteiro entre 0 e 11), um `minuto` (um inteiro entre 0 e 59) e um `segundo` (um inteiro entre 0 e 59).

A classe `relogio` tem as operações *getHora*, *getMinuto* e *getSegundo* que devolvem, respectivamente os `horas`, os `minutos` e os `segundos` do relógio. A representação externa de um relógio é `hh:mm:ss` (em que `hh` representa a hora, `mm` os minutos, `ss` os segundos).

a) Defina a classe `relogio`.

b) Defina a função `diferenca` que recebe como argumentos dois relógios devolve o número de horas completas de diferença. Por exemplo:

```
>>> diferenca(relogio(12, 21, 20), relogio(2, 11, 20))
10
```

Solução:

```
class relogio:
    def __init__(self, h, m, s):
        if isinstance(h, int) and 0 <= h <= 11 and \
           isinstance(m, int) and 0 <= m <= 59 and \
           isinstance(s, int) and 0 <= s <= 59:
            self.h = h
            self.m = m
            self.s = s
        else:
            raise ValueError('relogio: argumentos errados')

    def getHora(self):
        return self.h
    def getMinuto(self):
        return self.m
    def getSegundo(self):
        return self.s
    def __repr__(self):
        return '{:02d}:{:02d}:{:02d}'.format(self.h, self.m,
                                             self.s)

# Função externa
def diferenca(r1, r2):
    def diff(rmin, rmax):
        h = rmax.getHora() - rmin.getHora()
        if rmax.getMinuto() < rmin.getMinuto() or \
           rmax.getMinuto() == rmin.getMinuto() and \
           rmax.getSegundo() < rmin.getSegundo():
            h = h - 1
        return h
    if r1.relogio_anterior(r2):
        return diff(r1, r2)
    else:
        return diff(r2, r1)
```

TÉCNICO LISBOA

Fundamentos de Programação - 2018/2019 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 `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;
- `enche`, 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.

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.sai(26)
>>> a.passageiros()
0
```

Solução:

```
class autocarro:
    def __init__(self,capacidade):
        if isinstance(capacidade, int) and capacidade>=0:
            self.cap = capacidade
            self.npass = 0
        else:
            raise ValueError('autocarro: capacidade invalida')

    def capacidade(self):
        return self.cap

    def passageiros(self):
        return self.npass

    def sai(self, n):
        if self.npass >= n:
            self.npass = self.npass - n
        else:
            self.npass = 0

    def entra(self,n):
        if self.npass + n <= self.cap:
            self.npass = self.npass + n
        else:
            self.npass = self.cap
```

TÉCNICO LISBOA

Fundamentos de Programação - 2018/2019 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 `garrafa` cujo construtor recebe a capacidade de uma garrafa em litros. A garrafa inicialmente será criada vazia. Os outros métodos suportados pela classe são:

- `capacidade`, que devolve a capacidade total da garrafa;
- `nivel`, que devolve o volume de líquido presente na garrafa;
- `despeja`, que recebe a quantidade de líquido a remover da garrafa, em litros. Se a quantidade exceder o volume presente na garrafa, o volume presente na garrafa passa a ser 0;
- `enche`, que recebe a quantidade de líquido a colocar na garrafa. Se a quantidade de líquido a colocar na garrafa fizer com que esta ultrapasse a sua capacidade, o volume presente na garrafa passa a ser igual à sua capacidade.

Mostra-se a seguir um exemplo de interação:

```
>>> g1 = garrafa(1.5)
>>> g1.nivel()
0
>>> g1.capacidade()
1.5
>>> g1.despeja(2)
>>> g1.nivel()
0
>>> g1.enche(2)
>>> g1.nivel()
1.5
>>> g1.despeja(1)
>>> g1.nivel()
0.5
>>> g1.despeja(1)
>>> g1.nivel()
0
```

Solução:

```
class garrafa:
    def __init__(self, capacidade):
        if isinstance(capacidade, (int,float)) and capacidade>=0:
            self.max_vol = capacidade
            self.vol = 0
        else:
            raise ValueError('garrafa: capacidade invalida')

    def capacidade(self):
        return self.max_vol

    def nivel(self):
        return self.vol

    def despeja(self, n):
        if self.vol >= n:
            self.vol = self.vol - n
        else:
            self.vol = 0

    def enche(self,n):
        if self.vol + n <= self.max_vol:
            self.vol = self.vol + n
        else:
            self.vol = self.max_vol
```