



Nome:

Número:

Data:

Curso:

Capítulo 9 - Abstração de dados

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:

- *Construtor:*

$\text{vetor} : \text{real} \times \text{real} \times \text{real} \mapsto \text{vetor}$

$\text{vetor}(x, y, z)$ tem como valor o vetor cuja extremidade é o ponto (x, y, z) .

- *Seletores:*

$\text{valor_x} : \text{vetor} \mapsto \text{real}$

$\text{valor_x}(v)$ tem como valor a componente x da extremidade do vetor v .

$\text{valor_y} : \text{vetor} \mapsto \text{real}$

$\text{valor_y}(v)$ tem como valor a componente y da extremidade do vetor v .

$\text{valor_z} : \text{vetor} \mapsto \text{real}$

$\text{valor_z}(v)$ tem como valor a componente z da extremidade do vetor v .

- *Reconhecedores:*

$\text{eh_vetor} : \text{universal} \mapsto \text{lógico}$

$\text{eh_vetor}(\text{arg})$ tem valor verdadeiro apenas se arg é um vetor.

$\text{eh_vetor_nulo} : \text{vetor} \mapsto \text{lógico}$

$\text{eh_vetor_nulo}(v)$ tem valor verdadeiro apenas se v é o vetor $(0, 0, 0)$.

- *Teste:*

$\text{vetores_iguais} : \text{vetor} \times \text{vetor} \mapsto \text{lógico}$

$\text{vetores_iguais}(v1, v2)$ tem valor verdadeiro apenas se os vetores $v1$ e $v2$ são iguais.

(a) Defina uma representação para vetores utilizando tuplos.

(b) Escreva em Python as operações básicas, de acordo com a representação escolhida.

(c) Com base nas operações básicas, escreva uma função para calcular o produto escalar de dois vetores. O produto escalar dos vetores representados pelos pontos (a, b, c) e (d, e, f) é dado pelo real $a*d + b*e + c*f$.

a)

Representação $[(x,y,z)] = (x,y,z)$

b)

```
def vetor(x,y,z):
    if not (isinstance(x,int) and \
            isinstance(y,int) and isinstance(z,int)):
        raise ValueError('vetor: argumento(s) invalido(s)')
    return (x,y,z)

def valor_x(vetor):
    return vetor[0]

def valor_y(vetor):
    return vetor[1]

def valor_z(vetor):
    return vetor[2]

def e_vetor(univ):
    return isinstance(univ,tuple) and len(univ)==3 and \
           isinstance(univ[0],int) and \
           isinstance(univ[1],int) and \
           isinstance(univ[2],int)

def eh_vetor_nulo(vetor):
    return valor_x(vetor) == 0 and \
           valor_y(vetor) == 0 and \
           valor_z(vetor) == 0

def vetores_iguais(vetor1,vetor2):
    return valor_x(vetor1) == valor_x(vetor2) and \
           valor_y(vetor1) == valor_y(vetor2) and \
           valor_z(vetor1) == valor_z(vetor2)
```

c)

```
def prod_escalar_vetores(vetor1,vetor2):
    return valor_x(vetor1) * valor_x(vetor2) + \
           valor_y(vetor1) * valor_y(vetor2) + \
           valor_z(vetor1) * valor_z(vetor2)
```



Nome:

Número:

Data:

Curso:

Capítulo 9 - Abstração de dados

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:

- *Construtor:*

$\text{vetor} : \text{real} \times \text{real} \times \text{real} \mapsto \text{vetor}$

$\text{vetor}(x, y, z)$ tem como valor o vetor cuja extremidade é o ponto (x, y, z) .

- *Seletores:*

$\text{valor_x} : \text{vetor} \mapsto \text{real}$

$\text{valor_x}(v)$ tem como valor a componente x da extremidade do vetor v .

$\text{valor_y} : \text{vetor} \mapsto \text{real}$

$\text{valor_y}(v)$ tem como valor a componente y da extremidade do vetor v .

$\text{valor_z} : \text{vetor} \mapsto \text{real}$

$\text{valor_z}(v)$ tem como valor a componente z da extremidade do vetor v .

- *Reconhecedores:*

$\text{eh_vetor} : \text{universal} \mapsto \text{lógico}$

$\text{eh_vetor}(\text{arg})$ tem valor verdadeiro apenas se arg é um vetor.

$\text{eh_vetor_nulo} : \text{vetor} \mapsto \text{lógico}$

$\text{eh_vetor_nulo}(v)$ tem valor verdadeiro apenas se v é o vetor $(0, 0, 0)$.

- *Teste:*

$\text{vetores_iguais} : \text{vetor} \times \text{vetor} \mapsto \text{lógico}$

$\text{vetores_iguais}(v1, v2)$ tem valor verdadeiro apenas se os vetores $v1$ e $v2$ são iguais.

(a) Defina uma representação para vetores utilizando tuplos.

(b) Escreva em Python as operações básicas, de acordo com a representação escolhida.

(c) Com base nas operações básicas, escreva uma função para calcular a norma euclidiana de um vetor. A norma de um vetor representado pelo ponto (a, b, c) é dado pelo real $\sqrt{a * a + b * b + c * c}$.

a)

Representação $[(x, y, z)] = (x, y, z)$

b)

```
def vetor(x,y,z):
    if not (isinstance(x,int) and \
            isinstance(y,int) and isinstance(z,int)):
        raise ValueError('vetor: argumento(s) invalido(s)')
    return (x,y,z)

def valor_x(vetor):
    return vetor[0]

def valor_y(vetor):
    return vetor[1]

def valor_z(vetor):
    return vetor[2]

def e_vetor(univ):
    return isinstance(univ,tuple) and len(univ)==3 and \
           isinstance(univ[0],int) and \
           isinstance(univ[1],int) and \
           isinstance(univ[2],int)

def eh_vetor_nulo(vetor):
    return valor_x(vetor) == 0 and \
           valor_y(vetor) == 0 and \
           valor_z(vetor) == 0

def vetores_iguais(vetor1,vetor2):
    return valor_x(vetor1) == valor_x(vetor2) and \
           valor_y(vetor1) == valor_y(vetor2) and \
           valor_z(vetor1) == valor_z(vetor2)
```

c)

```
def norma_euclideana(vetor1):
    return sqrt(valor_x(vetor1) * valor_x(vetor1) + \
                valor_y(vetor1) * valor_y(vetor1) + \
                valor_z(vetor1) * valor_z(vetor1))
```



Nome:

Número:

Data:

Curso:

Capítulo 9 - Abstração de dados

Um número racional é qualquer número que possa ser expresso como o quociente de dois inteiros: o numerador (um inteiro positivo, negativo ou nulo) e o denominador (um inteiro positivo). Os racionais a/b e c/d são iguais se e só se $a*d = b*c$. Diz-se que um racional está na forma canónica se o numerador e o denominador forem primos entre si.

(a) Defina uma representação para número racional utilizando tuplos.

(b) Escreva as operações básicas `cria_rac`, `numerador`, `denominador`, `e_rac` e `racs_iguais` (cujo significado é óbvio), com base na representação de forma canónica. Isto é, supondo que o racional n/d é representado pelo par (n', d') , tal que $n*d' = n'*d$ e n' e d' são primos entre si.

Note que se dividirmos dois inteiros pelo seu máximo divisor comum obtemos dois inteiros que são primos entre si. Pode usar a função:

```
def mdc(m, n):  
    while n != 0:  
        m, n = n, m % n  
    return m
```

(c) Com base nas operações básicas do tipo racional, escreva a função `negativo_rac` que muda o sinal de um racional.

(d) Que alterações seriam precisas na função `negativo_rac` se a representação do racional não estivesse na forma canónica (isto é, se n e d não fossem primos entre si)?

a)

Representação [num/den] : (num, den)

b)

```
def cria_rac(n, d):  
    if not (isinstance(n, int) and isinstance(d, int) and d > 0):  
        raise ValueError('cria_rac: argumentos invalidos.')
```

```
    else:
```

```
        return (n / mdc(n, d), d / mdc(n, d))
```

```
def numerador(r):
```

```
    return r[0]
```

```
def denominador(r):
```

```
    return r[1]
```

```
def e_racional(arg):
```

```
    return isinstance(arg, tuple) and \  
           isinstance(numerador(arg), int) and \  
           isinstance(denominador(arg), int) and \  
           denominador(arg) > 0
```

```
def rac_iguais(r1, r2):
```

```
    return numerador(r1) * denominador(r2) == \  
           numerador(r2) * denominador(r1)
```

c)

```
def negativo_rac(r):
```

```
    return cria_rac(numerador(r) * -1, denominador(r))
```

d)

Não precisa de fazer nenhuma alteração, dado que o `cria_rac()` já cria um racional na forma canónica.



Nome:

Número:

Data:

Curso:

Capítulo 9 - Abstração de dados

Um número racional é qualquer número que possa ser expresso como o quociente de dois inteiros: o numerador (um inteiro positivo, negativo ou nulo) e o denominador (um inteiro positivo). Os racionais a/b e c/d são iguais se e só se $a*d = b*c$.

Suponha que o racional n/d é representado do seguinte modo:

$$\mathfrak{R}[n/d] = \begin{cases} 2^n \times 3^d & n \geq 0 \\ 2^{|n|} \times 3^d \times 5 & n < 0 \end{cases}$$

(a) Escreva as operações básicas `cria_rac`, `numerador`, `denominador`, `e_rac` e `racs_iguais` (cujo significado é óbvio), de acordo com esta representação.

Podem usar a função:

```
def expoente(num, base):  
    exp = 0  
    while num % base == 0:  
        exp = exp + 1  
        num = num // base  
    return exp
```

(b) Com base nas operações básicas do tipo racional, escreva a função `negativo_rac` que muda o sinal de um racional.

(c) Que alterações seriam precisas na função `negativo_rac` se a representação do racional fosse um tuplo (isto é, $\mathfrak{R}[n/d] = (n, d)$)?

Solução:

a)

```
def cria_rac(n, d):
    if not (isinstance(n, int) and isinstance(d, int) and d > 0):
        raise ValueError("cria_rac: argumentos invalidos")
    if n >= 0:
        return (2**n) * (3**d)
    else:
        return 2 ** abs(n) * 3 ** d * 5

def numerador(r):
    if not e_rac(r):
        raise ValueError("numerador: argumentos invalidos")
    if r % 5 == 0:
        return -expoente(r, 2)
    else:
        return expoente(r, 2)

def denominador(r):
    if not e_rac(r):
        raise ValueError("denominador: argumentos invalidos")
    return expoente(r, 3)

def e_rac(r):
    return isinstance(r, int) and r >= 1 and \
        r // (2**expoente(r, 2) * 3**expoente(r, 3)) in (1, 5)

def rac_iguais(r1, r2):
    return e_rac(r1) and e_rac(r2) and r1 == r2
```

b)

```
def negativo_rac(r):
    if not e_rac(r):
        raise ValueError("denominador: argumentos invalidos")
    return cria_rac(-numerador(r), denominador(r))
```

c)

Nenhuma



Nome:

Número:

Data:

Curso:

Capítulo 9 - Abstração de dados

Suponha que quer representar o tempo, dividindo-o em horas e minutos. No tipo tempo o número de minutos está compreendido entre 0 e 59, e o número de horas apenas está limitado inferiormente a zero. Por exemplo 546 : 37 é um tempo válido.

Considere as operações básicas do tipo tempo:

- *Construtor:*
 $cria_tempo : \mathbb{N}_0 \times \mathbb{N}_0 \mapsto tempo$
 $cria_tempo(h,m)$, em que $h \geq 0$ e $0 \leq m \leq 59$ tem como valor o tempo $h : m$.
- *Seletores:*
 $horas : tempo \mapsto \mathbb{N}_0$
 $horas(t)$ tem como valor as horas do tempo t .
 $minutos : tempo \mapsto \mathbb{N}_0$
 $minutos(t)$ tem como valor os minutos do tempo t .
- *Reconhecedores:*
 $eh_tempo : universal \mapsto lógico$
 $eh_tempo(arg)$ tem o valor verdadeiro apenas se arg é um tempo.
- *Teste:*
 $tempos_iguais : tempo : tempo \mapsto lógico$
 $tempos_iguais(t1, t2)$ tem o valor verdadeiro apenas se os tempos $t1$ e $t2$ são iguais.

(a) Escolha uma representação para o tipo tempo usando dicionários.

(b) Escreva em Python as operações básicas, de acordo com a representação escolhida.

(c) Com base nas operações básicas do tipo tempo, escreva a função

$num_minutos : tempo \mapsto inteiro$

$num_minutos(t)$ tem como valor o número de minutos entre o tempo 0 : 0 e o tempo t .

Solução:**a)**

Representação [h:m] : {'h':h, 'm':m}

b)

```
def cria_tempo(h, m):
    if type(h) != int or h <= 0 or type(m) != int or 59 < m < 0:
        raise ValueError("cria_tempo: argumentos invalidos")
    return {'h': h, 'm': m}

def horas(t):
    return t['h']

def minutos(t):
    return t['m']

def eh_tempo(arg):
    return type(arg) == dict and len(arg) == 2 and \
           'h' in arg and 'm' in arg

def tempos_iguais(t1, t2):
    return eh_tempo(t1) and eh_tempo(t2) and \
           horas(t1) == horas(t2) and minutos(t1) == minutos(t2)

c)
def numero_minutos(t):
    if not eh_tempo(t):
        raise ValueError(numero_minutos: argumentos invalidos')
    return horas(t)*60 + minutos(t)
```



Nome:

Número:

Data:

Curso:

Capítulo 9 -Abstração de dados

Suponha que quer representar o tempo, dividindo-o em horas e minutos. No tipo `tempo` o número de minutos está compreendido entre 0 e 59, e o número de horas apenas está limitado inferiormente a zero. Por exemplo `546 : 37` é um tempo válido.

Considere as operações básicas do tipo `tempo`:

- *Construtor:*
 $cria_tempo : \mathbb{N}_0 \times \mathbb{N}_0 \mapsto tempo$
 $cria_tempo(h,m)$, em que $h \geq 0$ e $0 \leq m \leq 59$ tem como valor o tempo $h : m$.
- *Seletores:*
 $horas : tempo \mapsto \mathbb{N}_0$
 $horas(t)$ tem como valor as horas do tempo t .
 $minutos : tempo \mapsto \mathbb{N}_0$
 $minutos(t)$ tem como valor os minutos do tempo t .
- *Reconhecedores:*
 $eh_tempo : universal \mapsto lógico$
 $eh_tempo(arg)$ tem o valor verdadeiro apenas se arg é um tempo.
- *Teste:*
 $tempos_iguais : tempo : tempo \mapsto lógico$
 $tempos_iguais(t_1, t_2)$ tem o valor verdadeiro apenas se os tempos t_1 e t_2 são iguais.

(a) Escolha uma representação para o tipo `tempo` usando tuplos.

(b) Escreva em Python as operações básicas, de acordo com a representação escolhida.

(c) Com base nas operações básicas do tipo `tempo`, escreva a função

$depois : tempo \times tempo \mapsto lógico$

$depois(t_1, t_2)$ tem o valor verdadeiro apenas se t_1 corresponder a um instante de tempo posterior a t_2 .

Solução:**a)**

Representação [h:m] : (h,m)

b)

```
def cria_tempo(h, m):
    if type(h) != int or h <= 0 or type(m) != int or 59 < m < 0:
        raise ValueError("cria_tempo: argumentos invalidos")
    return (h,m)
```

```
def horas(t):
    return t[0]
```

```
def minutos(t):
    return t[1]
```

```
def eh_tempo(arg):
    return type(arg) == tuple and len(arg) == 2
```

```
def tempos_iguais(t1, t2):
    return eh_tempo(t1) and eh_tempo(t2) and \
        horas(t1) == horas(t2) and minutos(t1) == minutos(t2)
```

c)

```
def depois(t1, t2):
    if not eh_tempo(t1) or not eh_tempo(t2):
        raise ValueError('depois: argumentos invalidos')
    return (horas(t1)*60 + minutos(t1)) > \
        (horas(t2)*60 + minutos(t2))
```



Nome:

Número:

Data:

Curso:

Capítulo 9 -Abstração de dados

Suponha que quer representar o tempo, dividindo-o em horas e minutos. No tipo tempo o número de minutos está compreendido entre 0 e 59, e o número de horas apenas está limitado inferiormente a zero. Por exemplo 546:37 é um tempo válido.

Considere as operações básicas do tipo tempo:

- *Construtor:*
 $cria_tempo : \mathbb{N}_0 \times \mathbb{N}_0 \mapsto tempo$
 $cria_tempo(h,m)$, em que $h \geq 0$ e $0 \leq m \leq 59$ tem como valor o tempo $h : m$.
- *Seletores:*
 $horas : tempo \mapsto \mathbb{N}_0$
 $horas(t)$ tem como valor as horas do tempo t .
 $minutos : tempo \mapsto \mathbb{N}_0$
 $minutos(t)$ tem como valor os minutos do tempo t .
- *Reconhecedores:*
 $eh_tempo : universal \mapsto lógico$
 $eh_tempo(arg)$ tem o valor verdadeiro apenas se arg é um tempo.
- *Teste:*
 $tempos_iguais : tempo : tempo \mapsto lógico$
 $tempos_iguais(t1, t2)$ tem o valor verdadeiro apenas se os tempos $t1$ e $t2$ são iguais.

(a) Escolha uma representação para o tipo tempo usando dicionários.

(b) Escreva em Python as operações básicas, de acordo com a representação escolhida.

(c) Com base nas operações básicas do tipo tempo, escreva a função

$diferenca_segundos : tempo \times tempo \mapsto inteiro$

$diferenca_segundos(t1, t2)$ tem como valor o número de segundos entre o tempo $t1$ e o tempo $t2$.

Solução:**a)**

Representação [h:m]: {'h':h, 'm':m}

b)

```
def cria_tempo(h, m):
    if type(h) != int or h <= 0 or type(m) != int or 59 < m < 0:
        raise ValueError("cria_tempo: argumentos invalidos")
    return {'h': h, 'm': m}

def horas(t):
    return t['h']

def minutos(t):
    return t['m']

def eh_tempo(arg):
    return type(arg) == dict and len(arg) == 2 and \
           'h' in arg and 'm' in arg

def tempos_iguais(t1, t2):
    return eh_tempo(t1) and eh_tempo(t2) and \
           horas(t1) == horas(t2) and minutos(t1) == minutos(t2)

c)
def diferenca_segundos(t1, t2):
    if not eh_tempo(t1) or not eh_tempo(t2):
        raise ValueError('diferenca_segundos: argumentos
invalidos')
    return (horas(t1)*3600 + minutos(t1)*60) - \
           (horas(t2)*3600 + minutos(t2)*60)
```