Programação I
Aula 19 — Aritmética com racionais

Pedro Vasconcelos
DCC/FCUP

Nesta aula

Aritmética com racionais

A linguagem Python inclui tipos numéricos de *inteiros* e *vírgula-flutuante*; exemplos:

$$-7, \quad 42, \quad 1.333 \times 10^6, \qquad 0.5 \times 10^{-3}$$

Vamos definir um tipo de dados novo para *números racionais*; exemplos:

$$\frac{1}{2}, \quad \frac{2}{3}, \quad -\frac{5}{7}, \quad \frac{1237}{100}$$

Aritmética com racionais (cont.)

Plano:

1. Definir uma classe `Fraction` para representar frações
2. Implementar métodos para simplificar, somar e multiplicar frações

Vamos usar apenas *operações sobre inteiros*, logo não há erros de arredondamento.

(Uma versão simplificada da biblioteca `fractions` distribuida com o Python.)

Números racionais

- Um número racional é sempre composto por:
  um numerador: um inteiro;
  um denominador: um inteiro diferente de 0.
- Qualquer par (p, q) nestas condições define um número racional que escrevemos como

$$\frac{p}{q}$$

- A representação não é única; por exemplo:

$$\frac{2}{3}, \quad \frac{4}{6}, \quad \frac{-8}{-12}$$

  representam o *mesmo* número racional.

Em Python

Vamos definir uma classe `Fraction` com dois atributos: `num` e `denom`.

Em Python (cont.)

```
class Fraction:
    '''Classe para representar frações.'''
    def __init__(self, num, denom):
        if denom==0:
            raise ZeroDivisionError("denominador zero")
        self.num   = num
        self.denom = denom

    def __str__(self):
         return '%d/%d' % (self.num,self.denom)
```


Exemplos

```
>>> print(Fraction(2,3))
2/3
>>> print(Fraction(5,6))
5/6
>>> print(Fraction(4,6))
4/6
>>> print(Fraction(5,10))
5/10
```

Problema: as frações não são simplificadas!

Simplificar frações

Podemos simplificar uma fração encontrando um *divisor comum* ao numerador e denominador:

$$\frac{8}{12} = \frac{4 \times 2}{6 \times 2} \quad \text{é equivalente a} \quad \frac{4}{6}$$

Para simplificar completamente, basta dividir ambos numerador e denominador pelo seu *maior divisor comum*.

Seja $d = \text{mdc}(p, q)$; então

$$\frac{p}{q} \quad \text{é equivalente a} \quad \frac{p \div d}{q \div d}$$

Esta transformação funciona *mesmo quando a fração já está simplificada* (porque nesse caso $d = 1$).

Algoritmo de Euclides

Para calcular o m.d.c. podemos usar o algoritmo de Euclides.

Uma implementação em Python:

```
def mdc(a: int, b: int) -> int:
    "Calcular o máximo divisor comum de a,b."
    while b != 0:
        r = a%b
        a = b
        b = r
    return a
```

Construir frações simplificadas

A melhor forma de garantir que as frações são *sempre* simplificadas é no inicializador.

Desta forma, de cada vez que criarmos uma nova fração temos a garantia que está simplificada.

Construir frações simplificadas (cont.)

```
class Fraction:
    ⋮
    def __init__(self, num, denom):
        if denom==0:
            raise ZeroDivisionError("denominador zero")
        d = mdc(num, denom)
        self.num = num//d
        self.denom = denom//d
    ⋮
```

Note que `mdc` é uma função fora da da classe `Fraction`.

Exemplos

```
>>> print(Fraction(4,6))
2/3
>>> print(Fraction(8,12))
2/3
>>> print(Fraction(2,3))
2/3
>>> print(Fraction(5,10))
1/2
```


Somar frações

Para somar frações temos de encontrar um denominador comum:

$$\frac{p}{q} + \frac{r}{s} = \frac{p \times s}{q \times s} + \frac{q \times r}{q \times s} \qquad \text{(denominador comum)}$$

$$= \frac{p \times s + q \times r}{q \times s}$$

Não temos de simplificar: este trabalho é feito pelo inicializador de `Fraction`.

Em Python

Vamos definir o método `__add__` para definir soma de frações.

A expressão

```
Fraction(1,2) + Fraction(2,3)
```

corresponde a chamar o método `__add__(self,other)` com argumentos

```
self = Fraction(1,2)
other = Fraction(2,3)
```


Em Python (cont.)

```
class Fraction:
  ⋮
  def __add__(self, other):
    # somar a fração self com a fração other
    # numerador do resultado
    n = self.num*other.denom + self.denom*other.num
    # denominador do resultado
    d = self.denom*other.denom
    # construir uma fração e retornar
    return Fraction(n,d)
```

Exemplos

```
>>> print(Fraction(1,2) + Fraction(2,3))
7/6
>>> print(Fraction(1,2) + Fraction(1,2))
1/1
>>> print(Fraction(1,2) + Fraction(2,1)
5/2
```


Subtração de frações

Pegunta: como fazer uma subtração?

$$\frac{1}{2} - \frac{1}{3} = \cdots$$

Resposta: Podemos somar com a *fração simétrica*

$$\frac{1}{2} + (\frac{-1}{3}) = \frac{1 \times 3 + (-1) \times 2}{2 \times 3} = \frac{1}{6}$$

Frações negativas

Como representar frações negativas?

$$\frac{-1}{3} \quad \text{vs.} \quad \frac{1}{-3}$$

- Qualquer das duas opções funciona serve
- Vamos fixar que *o denominador deve ser sempre positivo* (ou seja: o sinal está no numerador)
- É fácil modificar o inicializador para garantir esta propriedade

Frações negativas (cont.)

```
class Fraction:
  def __init__(self, num, denom):
      if denom==0:
          raise ZeroDivisionError("denominador zero")
      if denom<0:
          denom = -denom
          num = -num
      d = mdc(num, denom)
      self.num = num//d
      self.denom = denom//d
  ⋮
```

Exemplos

```
>>> print(Fraction(2,-3))
-2/3

>>> print(Fraction(-4,-3))
4/3

>>> print(Fraction(1,2) + Fraction(1,-2))
0/1
```


Multiplicação de frações

Basta multiplicar numeradores e denominadores:

$$\frac{p}{q} \times \frac{r}{s} = \frac{p \times r}{q \times s}$$

Exercício: implementar um método `__mul__` para implementar esta operação.

Igualdade

```
>>> Fraction(1,2) == Fraction(1,2)
False
>>> meio = Fraction(1,2)
>>> meio == meio
True
```

- Por omissão: a igualdade compara se *dois objetos são o mesmo*
- Mas para frações queremos comparar se *são equivalentes*
- Para isso temos de definir o método `__eq__`

Igualdade (cont.)

```
class Fraction:
    ⋮
    def __eq__(self, other):
        return (self.num == other.num and
                self.denom == other.denom)
```

As frações são *sempre* simplificadas no inicializador: basta testar se têm o mesmo numerador e denominador.

Igualdade (cont.)

Agora os operadores `==` e `!=` dão as respostas esperadas:

```
>>> Fraction(1,2) == Fraction(4,8)
True
>>> Fraction(1,2) + Fraction(1,2) == Fraction(1,1)
True
>>> Fraction(2,3) != Fraction(6,8)
True
```


Sumário

Definimos uma versão simplificada da biblioteca `fractions`:

- im novo tipo de dados para números racionais (frações)
- operações para simplificar, somar, multiplicar e comparar