

3 Aula 03: 08/AGO/2019

3.1 Aulas passadas

3.1.1 Primeira aula

Motivação básica para a criação de uma classe `Fracao`. Escrevemos as funções que estão no arquivo `hn_aula.py`.

Representamos uma fração através de dois inteiros `num` e `den`, numerador e denominador. As funções escritas foram `harmonico()`, `soma_fracao()`, `simplifique()` e `mdc()`.

Assim, o problema de determinar o número harmônico de ordem `n` nos levou a:

- uma abstração para manipularmos fração irredutíveis.
- questões envolvendo eficiência já que para determinarmos o `mdc()` foi possível ver como o algoritmo de Euclides é absurdamente mais rápido que os métodos tradicionais.

3.1.2 Segunda aula

Começamos a falar de classes e objetos. Usamos a série harmônica como motivação. Implementação parte da classe `Fracao`: `__init__()`, `__str__()` e `__add__()`. Hitoshi fez `simplifique()` e Coelho não.

3.2 Hoje

Refinar a abstração enquanto eficiência ficará para uma próxima oportunidade.

Continuamos a implementar a classe `Fracao` para mostrar:

- acesso a atributos de método: `hn.num`, `hn.den`
- chamada de métodos: `hn.get()` além de `self.simplifique()`
- chamada de métodos com parâmetros: `hn.put()`
- estender `Fracao + Fracao` para `Fracao + int`: `__add__()` usa `type(other)`
- estender para `int + Fracao`: `__radd__()` que se apoia em `__add__()`

Mensagem é *precisamos ensinar para o Python como realizar as operações sobre **Fracao***.

O doce sintático de `+` pode ser estendido para `-`, `*`, `/`, `%`,...: `__sub__()`, `__mul__()`, `__truediv__()`,...

3.3 Programação orientada a objetos

3.3.1 Cliente

```
from fracao import Fracao

def main():
    n = int(input("Digite n: "))

    # calcule o número harmônico de ordem n
    hn = Fracao()
    for i in range(1, n+1):
        f = Fracao(1,i)
        hn = hn + f
    num, den = hn.get() # MOTIVAÇÃO para get() e put() é forçado
    print("1 + ... + 1/%d + 1/%d = %s = %f" %(n-1, n, hn, num/den))

#-----
# início da execução do programa
if __name__ == "__main__":
    main()
```

3.3.2 Tópicos

- Objetos: referências
- Classes nativas e classes definidas pelo usuário
- atributos
- método especial `__init__()` (aula passada)
- método especial `__str__()` (aula passada)
- método especial `__add__()` (aula passada)
- método `get()` (hoje)
- método `put()` (hoje)
- método especial `__add__()` modificado (hoje)
- método especial `__radd__()` (hoje)
- outros métodos

3.3.3 Objetos e classe nativas

Em Python, todo valor é um objeto.

Uma lista, ou mesmo um inteiro, todos são objetos

```
6          é um objeto da classe int
3.14       é um objeto da classe float
[1,2,3]    é um objeto da classe list
```

Para saber a classe de um objeto:

```
type(objeto)
```

```

>>> type(6)
<class 'int'>
>>> id(6)
4297370848
>>> i = 6
>>> j = 6
>>> id(i)
4297370848
>>> id(j)
4297370848
>>>

```

Linguagens orientadas a objetos permitem ao programadores criarem novas classes.

A função `print()` requer que o objeto se converta para um string que possa ser exibido.

`__str__` é o método padrão que diz como deve se comportar

3.3.4 Classes

Nós usamos muitas classes nativos do Python.

Agora iremos definir novas classes. Em particular uma classe `Fracao`.

Classes são formadas por atributos que podem ser variáveis ou funções que são chamadas de métodos.

A primeira letra em um nome de uma classe deve ser maiúscula.

3.3.5 Objetos

Um objeto contém as informações/valores de um tipo definido pelo programador.

3.3.6 Esquema geral da classe `Fracao`

Visão geral que será explicada pouco a pouco.

```
r = Fracao(4,3)
```

```

      +-----+
self--->| METODOS: __init__() |
      |           __str__() |
r ---->| +-----+ |
      | | ESTADO:  | |
      | | num  -----> 4 |
      | | deb  -----> 3 |
      | +-----+ |
      |           |
      +-----+

```

3.3.7 Métodos

Métodos são funções associadas com uma determinada classe.

```
r1 = Fracao(1,2)
```

Métodos são como funções, mas há duas diferenças:

- métodos são definidos dentro de uma classe
- a sintaxe para executar um método é diferente

O primeiro parâmetro de um método é chamados **self**.

```
imprima(r1)
```

sugere

- função imprima, aqui está um objeto para você imprimir
- `r1.imprima()` sugere `r1`, imprima a si mesmo
- essa mudança de perspectiva pode ser polida, mas não é óbvio que seja útil.

Nos exemplos visto até agora talvez não seja.

Algumas vezes mover a responsabilidade de uma função para um objeto faz com que seja possível escrever um código mais versátil que é mais fácil de ser reutilizado e mantido.

No momento (ou em MAC0122) o que está escrito acima é longe de óbvio...

3.3.8 Construtores

O método especial `__init__()` é responsável por construir e retornar um objeto.

Chamado quando um objeto é criado (= *instanciado* é um nome mais bonito).

3.3.9 Imprimindo um objeto

O método especial `__str__()` cria e retorna um string que diz como o objeto deve ser impresso por `print()`.

3.3.10 Exemplo: Classe Fracao

Um objeto contém as informações/valores de um tipo definido pelo programador.

```
+-----+
|  __init__  __add__  |
|  __str__   |
|      +-----+      |
|      | ESTADO      | |
|      |              | |
|      | num          | |
|      | den          | |
|      |              | |
|      +-----+      |
|                      |
```

3.4 Roteiro para construir a classe

3.4.1 `__init__()`

```
>>> r = Fracao(2,3)
>>> print(r.num)
2
>>> print(r.den)
3
>>> print(r)
<__main__.Fracao object at 0xb5360eac>
>>> print("%d/%d" %(r.num,r.den))
2/3
```

3.4.2 `__str__()`

```
>>> r = Fracao(3,1)
>>> print(r)
3/1
```

3.4.3 `__str__()` para inteiros

```
>>> r = Fracao(2,1)
>>> print(r)
2
```

3.4.4 parâmetros opcionais

```
>>> r = Fracao(2)
>>> r = Fracao()
>>> r1 = Fracao(12,6)
>>> print(r1)
12/3
```

3.4.5 `simplifique()`: irredutível

3.4.6 `simplifique()`: negativo

3.4.7 `get()`:

3.4.8 `put()`: objetos são mutáveis

3.4.9 `__add__()`

3.4.10 `__sub__()`

3.4.11 `get()`

3.4.12 `put()`

3.4.13 `__radd__()`

3.5 Implementação

```
#-----
class Fracao:
    #-----
    def __init__(self, num = 0, den = 1):
        """ (Fracao, int, int) -> Fracao

        Construtor: cria um objeto Fracao.
        Mágica: funcao retorna mas nao tem return.
        """

        self.num = num
        self.den = den
        self.simplifique()

    #-----
    def __repr__(self):
        """ (Fracao) -> str

        Retorna a string representa self oficialmente.
        """

        return self.__str__()

    #-----
    def __str__(self):
        """ (Fracao) -> str

        Retorna o string que print() usa para imprimir um
        Fracao.
        """

        if self.den == 1:
            texto = "%d" %self.num
        else:
            texto = "%d/%d" %(self.num,self.den)
        return texto

    #-----
    def simplifique(self):
        """ (Fracao) -> None

        Recebe um racional e altera a sua representacao
        para a forma irredutivel.
        """

        comum = mdc(self.num, self.den)
        self.num //= comum
        self.den //= comum
        if self.den < 0:
            self.den = -self.den
            self.num = -self.num
```

```

#-----
def get(self):
    """ (Fracao) -> int, int

    Recebe um racional e retorna o seu numerador e o
    seu denominador.
    """
    return self.num, self.den

#-----
def put(self, novo_num, novo_den):
    """ (Fracao) -> None

    Recebe um racional e dois inteiros novo_num e
    novo_den e modifica a fracao para representar
    novo_num/novo_den.
    """
    self.num = novo_num
    self.den = novo_den
    self.simplifique()

#-----
def __add__(self, other):
    """ (Fracao, Fracao ou int) -> Fracao

    Retorna a soma da fracao `self` e da fracao ou int `other`.
    Usado pelo Python quando escrevemos Fracao + Fracao ou
    Fracao + int
    """
    if type(other) == int:
        novo_num = self.num + self.den*other
        novo_den = self.den
    else:
        novo_num = self.num*other.den + self.den*other.num
        novo_den = self.den*other.den
    f = Fracao(novo_num,novo_den)
    return f

#-----
def __radd__(self, other):
    """ (Fracao, int) -> Fracao

    Retorna a soma da fracao `self` e int `other`.
    Usado pelo Python quando escrevemos int + Fracao
    """
    return self + other # self.__add__(other)

#-----
def __sub__(self, other):

```



```

""" (Fraco, Fracao ou int) -> Fracao

Retorna a diferenca das fracoes `self` e `other`.
Usado pelo Python quando escrevemos Fracao - int
"""

if type(other) == int:
    novo_num = self.num - self.den*other
    novo_den = self.den
else:
    novo_num = self.num*other.den - self.den*other.num
    novo_den = self.den*other.den
f = Fracao(novo_num,novo_den)
return f

#-----
def __rsub__(self, other):
    """ (Fraco, int) -> Fracao

    Retorna a diferenca a fracao `self` e o int `other`.
    Usado pelo Python quando escrevemos int - Fracao
    """

    return self - other

#-----
def __mul__(self, other):
    """ (Fracao, Fracao ou int) -> Fracao

    Retorna o produto da fração `self` e a fração ou int `other`.
    Usado pelo Python quando escrevemos Fracao * Fracao
    ou int * Fracao
    """

    if type(other) == int:
        novo_num = self.num * other
        novo_den = self.den
    else:
        novo_num = self.num * other.num
        novo_den = self.den * other.den
    f = Fracao(novo_num, novo_den)
    return f

#-----
def __rmul__(self, other):
    """ (Fracao, int)
    Usado pelo Python quando escrevemos int * Fracao
    """

    return self * other

#-----
def __truediv__(self, other):

```

```

    if type(other) == int:
        novo_num = self.num
        novo_den = self.den * other
    else:
        novo_num = self.num * other.den
        novo_den = self.den * other.num
    f = Fracao(novo_num, novo_den)
    return f

#-----
def __rtruediv__(self, other):
    return self / other

#-----
def __eq__(self, other):
    if type(other) == int:
        prim_num = self.num
        seg_num = other * self.den
    else:
        prim_num = self.num * other.den
        seg_num = other.num * self.den
    return prim_num == seg_num

#-----
def __invert__(self):
    return Fracao(-self.num, self.den)

#-----
def mdc(m,n):
    """ (int, int) -> int
    Recebe dois inteiros m e n e retorna o
    mdc de m e n.

    Se m = 0 = n retorna 0 para indicando erro.
    """
    if n < 0: n = -n
    if m < 0: m = -m
    if n == 0: return m
    r = m%n
    while r != 0:
        m = n
        n = r
        r = m % n
    return n

```

3.6 Módulo fractions

Python 3.5.2 (default, Sep 10 2016, 08:21:44)

[GCC 5.4.0 20160609] on linux

Type "help", "copyright", "credits" or "license" for more information.

```
>>> import fractions
```

```
>>> dir(fractions)
```

```
['Decimal', 'Fraction', '_PyHASH_INF', '_PyHASH_MODULUS', '_RATIONAL_FORMAT', '__all__', '__bu
```

```
>>> help(fractions.gcd)
```

Help on function gcd in module fractions:

```
gcd(a, b)
```

Calculate the Greatest Common Divisor of a and b.

Unless b==0, the result will have the same sign as b (so that when b is divided by it, the result comes out positive).

```
>> r = fractions.Fraction()
```

```
>>> print(r)
```

```
0
```

```
>>> r1 = fractions.Fraction(1,2)
```

```
>>> r2 = fractions.Fraction(2,3)
```

```
>>> r1 + r2
```

```
Fraction(7, 6)
```

```
>>> print(r1 + r2)
```

```
7/6
```

```
>>>
```

3.7 Glossário

- **atributo:** Um dos itens nomeados de dados que compõem uma instância.
- **classe:** Um tipo de composto definido pelo usuário. Uma classe também pode ser pensada como um modelo para os objetos que são instâncias da mesma. (O iPhone é uma classe. Até dezembro de 2010, as estimativas são de que 50 milhões de instâncias tinham sido vendidas!)
- **construtor:** Cada classe tem uma “fábrica”, chamada pelo mesmo nome da classe, por fazer novas instâncias. Se a classe tem um método de inicialização, este método é usado para obter os atributos (ou seja, o estado) do novo objeto adequadamente configurado.
- **instância:** Um objeto cujo tipo é de alguma classe. Instância e objeto são usados como sinônimos.
- **instanciar:** Significa criar uma instância de uma classe e executar o seu método de inicialização.
- **linguagem orientada a objetos** Uma linguagem que fornece recursos, como as classes definidas pelo usuário e herança, que facilitam a programação orientada a objetos.
- **método:** Uma função que é definida dentro de uma definição de classe e é chamado em instâncias dessa classe.
- **método de inicialização:** Um método especial em Python, chamado `__init__()`, é chamado automaticamente para configurar um objeto recém-criado no seu estado inicial (padrão de fábrica).
- **objeto:** Um tipo de dados composto que é frequentemente usado para modelar uma coisa ou conceito do mundo real. Ele agrupa os dados e as operações que são relevantes para esse tipo de dados. Instância e objeto são usados como sinônimos.
- **programação orientada a objetos:** Um estilo poderoso de programação em que os dados e as operações que os manipulam são organizados em classes e métodos.