Aula 02: 07/AGO/2018

Aula passada

Motivação básica para a criação de uma classe `Fracao`.

Hoje

Começamos a trilhar um caminho para uma *introdução rudimentar à programação orientada a objetos* e tratar de tópicos como:

- objetos: referências
- classes nativas *versus* classes definidas pelo usuário
- atributos de estado e métodos
- método especial/mágico construtor `__init__()`
- método especial/mágico `__str__()`
- objetos são mutáveis
- sobrecarga de operadores e os métodos especiais: `__add__()`, `__mul__()`, ...
- doce sintático (= *syntactic suger*)

Veremos esses tópicos no decorrer das próximas aulas.

Problema (versão exata)

Dado um inteiro positivo `n`, calcular o valor de H_n, o número *harmônico de ordem* `n`:

```
   1 +    1/2    +   1/3   + 1/4 + ... + 1/n
```

de **maneira exata** usando frações.

Exemplos

```
Digite n: 6
1   +   1/2   + ... + 1/n =  49/20

Digite n: 7
1   +   1/2   + ... + 1/n =  363/140
>>> 363/140
2.592857142857143

Digite n: 7
1   +   1/2   + ... + 1/n =  2.5928571428571425
1/n + 1/(n-1) + ... +  1  =  2.5928571428571425

Digite n: 15
1   +   1/2   + ... + 1/n =  3.3182289932289937
1/n + 1/(n-1) + ... +  1  =  3.318228993228993
```

```
Digite n: 15
1   +   1/2   + ... + 1/n =  1195757/360360
>>> 1195757/360360
3.3182289932289932
```

Solução

Basta calcular da esquerda para a direita ou da direita para a esquerda.

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

    num, den = harmonico(n)
    print("1 + ... + 1/%d + 1/%d = %d/%d = %f" %(n-1,n,num,den,num/den))

#---------------------------------------------------------------
def harmonico(n):
    ''' (int) -> int, int

    Recebe um numero inteiro positivo e retorna o numero harmonico
    de ordem n representado como uma racional.
    O numero harmonico e calculado somando os termos
    da esquerda para a direita.
    '''
    num, den = 0, 1 # numerador e denominador
    for i in range (1, n+1):
        num, den = soma_fracoes(num, den, 1, i)
    return num, den

#---------------------------------------------------------------
def soma_fracoes(n1, d1, n2, d2):
    ''' (int, int, int, int) -> int, int

    Recebe quatro numeros inteiros n1, d1, n2 e d2 representando
    duas fracoes n1/d1 e n2/d2 e retorna um par (num,den)
    que representa a soma desses números.

    Pre-condicao: a função supõe que os quadro números dados nao
    sao nulos.
    '''
    den = d1 * d2
    num = n1*d2 + n2*d1
    return simplifique(num, den)


#-----------------------------------------
def simplifique(n, d):
    '''(int, int) -> int, int
```

```
    Recebe uma racional n/d e retorna a correspondente
    racional irredutível.
    '''
    comum = mdc(n,d)
    n //= comum  # precisa ser //
    d //= comum  # precisa ser //
    if d < 0:
        d = -d
        n = -n
    return n, d

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

    Pré-condição: a função supõe que pelo menos um dos
        parâmetros é não nulo.
    """
    if n == 0: return m
    if n <  0: n = -n
    if m <  0: m = -m
    # calcule o mdc()
    r = m%n
    while r != 0:
        m = n
        n = r
        r = m % n
    return n

#--------------------------------------------------
# início da execução do programa
main()
```

Solução usando uma classe Fracao

No momento a classe `Fracao` é apenas imaginária.

```
from fracao import Fracao

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

    hn = harmonico(n)
    print("1 + ... + 1/%d + 1/%d = " %(n-1,n), hn)

#-------------------------------------------------------------------
def harmonico(n):
    ''' (int) -> Fracao

    Recebe um numero inteiro positivo e retorna o numero harmonico
    de ordem n representado como fração.
    O numero harmonico e calculado somando os termos
    da esquerda para a direita.
    '''
    soma = Fracao()
    for i in range(1,n+1):
        soma += Fracao(1,i)
    return soma


#-------------------------------------
if __name__ == "__main__":
    main()
```

Solução curta e grossa usando `Fracao`

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

    soma = Fracao()
    for i in range(1,n+1):
        soma += Fracao(1,i)

    print("1   +   1/2   + ... + 1/n = ", soma)
```

Programação orientada a objetos

Tópicos

- Objetos: referências
- Classes nativas e classes definidas pelo usuário
- atributos
- método especial `__init__()`
- método especial `__str__()`
- outros métodos

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

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.

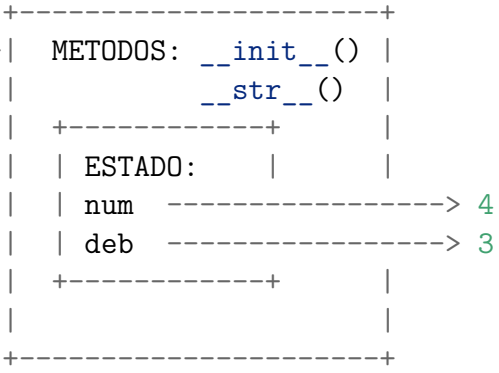
Objetos

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

Esquema geral da classe Fracao

Visão geral que será explicada pouco a pouco.

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



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...

Construtores

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

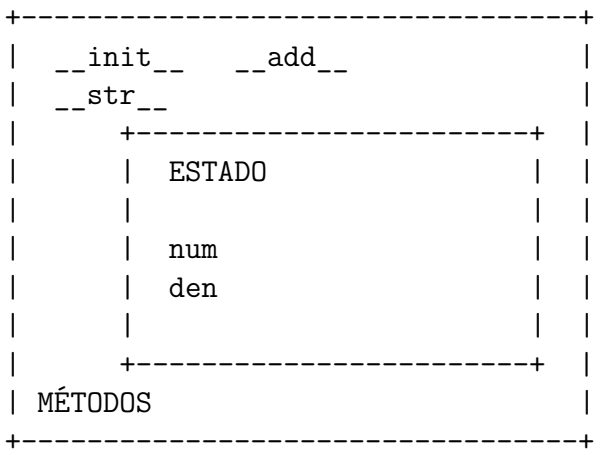
Chamado quando um objeto é criado (= *instanciado* é um nome mais bonito).

Imprimindo um objeto

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

Exemplo: Classe `Fracao`

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



Ordem para construir a classe

`__init__()`

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

`__str__()`

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

__str__() para inteiros

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

parâmetros opcionais

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

simplifique(): irredutível

simplifique(): negativo

get():

put(): objetos são mutáveis

__add__()

__sub__()

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 __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) -> Fracao

    Retorna a soma dos racionais `self` e `other`.
    Usado pelo Python quando escrevemos Fracao + Fracao
    """
    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 __sub__(self, other):
        """ (Fraco, Fracao) -> Fracao

        Retorna a diferenca das fracoes `self` e `other`.
        Usado pelo Python quando escrevemos Fracao - Fracao
        """
        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 __mul__(self, other):
        """ (Fracao, Fracao) -> Fracao

        Retorna o produto dos racionais `self` e `other`.
        Usado pelo Python quando escrevemos Fracao * Fracao
        """
        novo_num = self.num * other.num
        novo_den = self.den * other.den
        f = Fracao(novo_num, novo_den)
        return f

    #-------------------------------------
    def __truediv__(self, other):
        novo_num = self.num * other.den
        novo_den = self.den * other.num
        f = Fracao(novo_num, novo_den)
        return f

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



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

    Pré-condição: a função supõe que pelo menos um dos
        parâmetros é não nulo.
```

```
    """
    if n == 0: return m
    if n <  0: n = -n
    if m <  0: m = -m
    # calcule o mdc()
    r = m%n
    while r != 0:
        m = n
        n = r
        r = m % n
    return n
```

Grossário

- **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.
- **liguagem 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.