

1 Aula 01: 01/AGO/2019

1.1 resumo de MAC0110

Até agora, mais ou menos, conversamos sobre:

- classes nativas: `int`, `float`, `bool`, `list`, `str`, `Nonetype`, `dict`
- funções
- listas, listas de listas, matrizes
- strings
- dicionários
- percorrer listas, strings e dicionários
- apelidos versus clones
- **sobretudo**: raciocínio aplicado a resoluções de problemas compu... blá-blá-blá

1.2 Página de MAC0122

<https://edisiplinas.usp.br>:

1.3 Critérios de avaliação

Provas, EPs e provinhas: primeira provinha é hoje.

1.4 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/mágicos: `__add__()`, `__mul__()`, ...
- doce sintático (= *syntactic sugr*)

Hmm. Chegaremos nisso só, **talvez**, na próxima aula.

1.5 Problema de motivação (versão aproximada)

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

$$\begin{aligned} &1 + \frac{1}{2} + \frac{1}{3} + \frac{1}{4} + \dots + \frac{1}{n} \\ &\frac{1}{n} + \frac{1}{(n-1)} + \frac{1}{(n-2)} + \dots + \frac{1}{2} + 1 \end{aligned}$$

programa	solução
hn0.py	aproximada
hn1.py	exata usando fração não irredutível
hn2.py	exata usando fração irredutível e Euclides
hnX.py	exata usando fração irredutível e <code>mdc()</code> força-bruta

Observação 1: no semestre passado (mais ou menos na aula 11), discutimos o cálculo dos números harmônicos de ordem n da direita para a esquerda e da esquerda para a direita.

Na ocasião o objetivo era falarmos sobre `floats` (com sua precisão limitada) versus `ints` (potencialmente ilimitados).

Hoje, essa discussão nos levará a implementação da classe `Fracao`.

Observação 2 (`hn0.py`): pode ser usado para lembrar funções e os cálculos aproximados envolvendo `floats`.

Observação 3 (`hn0.py`): pode ser usado para dar uma muito breve revisão dos comandos `for` x `while` e mostrar a tabulação e definição de função para quem não usou Python em MAC0110.

Observação 4 (`hnX.py` x `hn2.py`): depois de fazermos algo com fração e legal rodarmos o programa com o uma função `mdc()` simples e outra com o `mdc()` usando o algoritmo de Euclides. Isso é só para dar o sabor que “resolver” o problema não basta, a solução deve ser efetiva.

Tempos para `hnX.py`:

n	tempo
15	0m1.238s
16	0m2.048s
17	0m15.662s
18	0m30.148s
19	5m33.619s

Tempos para `hn2.py`:

n	tempo
19	0m0.060s
20	0m0.057s
40	0m0.070s
80	0m0.057s
160	0m0.084s

1.5.1 Solução aproximada

Solução usando float, ou seja, aproximada.

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

    hn1 = harmonico_ED(n)
    print("1 + ... + 1/%d + 1/%d = " %(n-1,n), hn1)

    hn2 = harmonico_DE(n)
    print("1/%d + 1/%d + ... + 1 = " %(n,n-1), hn2)

#-----

def harmonico_ED(n):
    '''(int) -> float

    Recebe um inteiro n e retorna o número harmônico
    de ordem n que foi calculado adicionando-se os termos
    da esquerda para a direita.
    '''
    soma = 0
    i = 1
    while i < n+1:
        soma += 1/i
        i += 1
    return soma

#-----

def harmonico_DE(n):
    '''(int) -> float

    Recebe um inteiro n e retorna o número harmônico
    de ordem n que foi calculado adicionando-se os termos
    da direita para a esquerda.
    '''
    soma = 0
    i = n
    while i > 0:
        soma += 1/i
        i -= 1
    return soma

#-----

# início da execução do programa
main()

#-----
```

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

    soma = 0
    i = 1
    while i < n+1:
        soma += 1/i
        i += 1
    print("1 + 1/2 + ... + 1/n = ", soma)

    soma = 0
    i = n
    while i > 0:
        soma += 1/i
        i -= 1
    print("1/n + 1/(n-1) + ... + 1 = ", soma)
```

1.5.2 Exemplos de execução

```
Digite n: 1000000
1 + 1/2 + ... + 1/n = 14.392726722864781
1/n + 1/(n-1) + ... + 1 = 14.392726722865754
```

```
Digite n: 6
1 + 1/2 + ... + 1/n = 2.4499999999999997
1/n + 1/(n-1) + ... + 1 = 2.45
```

1.6 Representação de frações

Podemos representar uma fração de maneira exata através do seu *numerador* e o seu *denominador*:

```
1/3 em vez de 0.3333333
2/4 em vez de 0.5
```

1.7 Problema (versão exata)

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

$$1 + \frac{1}{2} + \frac{1}{3} + \frac{1}{4} + \dots + \frac{1}{n}$$

de **maneira exata** usando frações.

1.7.1 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
```

1.7.2 Solução

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

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

    n1, d1 = harmonico_RacED(n)
    print("1 + ... + 1/%d = %d/%d = %f" %(n,n1,d1,n1/d1))

    n2, d2 = harmonico_RacDE(n)
    print("1/%d + ... + 1 = %d/%d = %f" %(n,n2,d2,n2/d2))

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

    Recebe um numero inteiro positivo e retorna o numero harmonico
    de ordem n representado como uma fracao.
    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 harmonico_RacDE(n):
    ''' (int) -> int, int
```

```

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

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

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

    Pre-condicao: a função supõe que os quatro 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 fracao n/d e retorna a correspondente
    fracao irredutível.
    '''
    comum = mdc(n,d) # comum = math.gcd(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.
    Pre-condição: a função supoe que n != 0
    """
    if m < 0: m = -m

```

```

if n < 0: n = -n
if n == 0: return m
r = m%n
while r != 0:
    m = n
    n = r
    r = m % n
return n

```

```

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

```

Nota: a diferença de consumo de tempo é gritante se usarmos

```

#-----
def mdc(m,n):
    """ (int, int) -> int
    Recebe dois inteiros m e n e retorna o
    mdc de m e n.
    Pre-condição: a função supoe que min(abs(n),abs(m)) != 0
    """
    if m < 0: m = -m # m = abs(m)
    if n < 0: n = -n # n = abs(n)
    d = min(n, m)
    if d == 0: return m
    while m % d != 0 or n % d != 0:
        d -= 1
    return d

```

1.8 Solução usando uma classe Fracao

No momento a classe Fracao é apenas imaginária.

```
from fracao import Fracao
```

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

    hn1 = harmonico_RacED(n)
    print("1 + ... + 1/%d + 1/%d = " %(n-1,n), hn1)

    hn2 = harmonico_RacDE(n)
    print("1/%d + 1/%d + ... + 1 = " %(n,n-1), hn2)

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

    Recebe um numero inteiro positivo e retorna o numero harmonico
    de ordem n representado como fracao.
    O numero harmonico e calculado somando os termos
    da esquerda para a direita.
    '''
    soma = Fracao(0)
    for i in range (1, n+1):
        soma += Fracao(1,i)
    return soma

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

    Recebe um numero inteiro positivo e retorna o numero harmonico
    de ordem n. O numero harmonico e calculado somando os termos
    da direita para a esquerda.
    '''
    soma = Fracao(0)
    for i in range (n, 0, -1):
        soma += Fracao(1, i)
    return soma
```

1.8.1 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)

```

1.9 Programação Orientada a Objetos

1.9.1 Tópicos:

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

1.9.2 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 aos 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

1.9.3 Classes

Nós usamos muitas classes nativas 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.

1.9.4 Objetos

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

1.9.5 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
| | den -----> 3
| +-----+
|
+-----+
```

1.9.6 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...

1.9.7 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).

1.9.8 Imprimindo um objeto

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

1.9.9 Classe Fracao

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

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

```
#-----
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 uma fracao e altera a sua representacao
    para a forma irredutivel.
    """
    comum = mdc(self.num,self.den)
    self.num //= comum
    self.den //= comum
    # trecho a seguir é supérfluo
    # devido ao sinal do mdc
    # if self.den < 0:
    #     self.den = -self.den
    #     self.num = -self.num

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

    Recebe uma fracao e retorna o seu numerador e o
    seu denominador.
    """
    return self.num, self.den

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

    Recebe uma fracao 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):
    """ (Fracao,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.

    Se n != 0, retorna um valor com o mesmo

```

senal de n. Isso é conveniente para simplificar frações.

```
"""  
if n == 0: return m  
r = m%n  
while r != 0:  
    m = n  
    n = r  
    r = m % n  
return n
```

1.10 Python gcd() e Fraction

gcd() é uma das funções do módulo math do Python.

```
>>> import math  
>>> math.gcd(10,8)  
2  
>>> math.gcd(-10,8)  
2  
>>> math.gcd(-10,-8)  
2  
>>> math.gcd(0,-8)  
8  
>>> math.gcd(0,0)  
0  
>>>
```

Python possui a classe fractions

```
>>> from fractions import Fraction  
>>> Fraction(16, -10)  
Fraction(-8, 5)  
>>> Fraction(123)  
Fraction(123, 1)  
>>> Fraction()  
Fraction(0, 1)  
>>> Fraction(' -3/7 ')  
Fraction(-3, 7)  
>>> Fraction('1.414213 \t\n')  
Fraction(1414213, 1000000)  
>>> Fraction('7e-6')  
Fraction(7, 1000000)  
>>> Fraction(2.25)  
Fraction(9, 4)  
>>> Fraction(1.1)  
Fraction(2476979795053773, 2251799813685248)  
>>> from decimal import Decimal  
>>> Fraction(Decimal('1.1'))  
Fraction(11, 10)
```

1.11 Métodos mágicos

Sobrecarga de operadores

Operator	Method
+	object.__add__(self, other)
-	object.__sub__(self, other)
*	object.__mul__(self, other)
//	object.__floordiv__(self, other)
/	object.__div__(self, other)
%	object.__mod__(self, other)
**	object.__pow__(self, other[, modulo])
<<	object.__lshift__(self, other)
>>	object.__rshift__(self, other)
&	object.__and__(self, other)
^	object.__xor__(self, other)
	object.__or__(self, other)

1.12 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.