INE5231 Computação Científica I

Prof. A. G. Silva

27 de junho de 2017

Conteúdo programático

- **O computador - [3 horas-aula]**
- **Representação de algoritmos - [3 horas-aula]:**
- **Linguagens de programação estruturadas [3 horas-aula]**
- **Introdução à programação em C [6 horas-aula]**
- **Programas envolvendo processos de repetição e seleção [6 horas-aula]**
- **Variáveis estruturadas unidimensionais homogêneas [9 horas-aula]**
- **Variáveis estruturadas multidimensionais homogêneas [6 horas-aula]**
- **Subdivisão de problemas e subprogramação [6 horas-aula]**
- **Variáveis estruturadas heterogêneas [6 horas-aula]**
- **Programação utilizando uma linguagem de computação técnica numérica [6 horas-aula]**

Python para Computação Científica

Versão Nov/2007 (Python 2.5, NumPy 1.0.4, SciPy 0.6.0, matplotlib 0.90.1, PIL 1.1.6)

C3

Alexandre Gonçalves Silva

DCC – CCT – UDESC

Tópicos

- A linguagem Python
  - Introdução
  - Controles de fluxo
  - Estruturas de dados
  - Classes
  - Módulos
  - Exemplos
  - Exercícios
- Computação científica
  - Introdução
  - Módulos
  - Aplicações
  - Exemplos
  - Exercícios
- Considerações Finais
- Referências

A linguagem

- Concebida pelo Holandês Guido van Rossum
- Amsterdã, 1990
- Definições de Python
  - Nome popular a uma grande cobra
  - “Monty Python’s Flying Circus”
- Entidades
  - Python Software Foundation
  - OSI Certified Open Source
- Página oficial
  - www.python.org
- Comunidade brasileira
  - www.pythonbrasil.com.br

Usuários e Eventos

- Indústria
  - Red Hat Linux
  - Industrial Light & Magic
  - Google, Yahoo, Infoseek
  - YouTube
  - IBM e Philips
  - Disney Feature Length Animation
  - RealNetworks
- Ciência
  - NASA
  - National Institutes of Health (USA)
  - National Weather Service (USA)
  - Lawrence Livermore National Laboratories
  - Theoretical Physics Division at Los Alamos National Laboratory
  - UNICAMP, USP, FURG, INPE, UFES, UDESC, SOCIESC
- Governo
  - The US Navy
  - The US Dept. of Agriculture
- Indústria
  - Conectiva
  - Embratel
  - Blender
  - Gimp
  - Inkscape
  - MayaVi
  - BitTorrent
  - WinCVS
  - Jext
- Bindings/wrapper
  - PyOpenGL
  - PyGTK
  - wxPython
  - PySDL
  - PythonMagick
  - ...
- Eventos
  - PyCon
  - PyCon UK
  - PyCon Brasil
  - EuroPython
  - pyCologne
  - SciPy
  - ...

Características

- Interpretada, de altíssimo nível (*very-high-level*)
- Orientada a objetos
- Tratamento de erros e exceções
- Coleta de lixo automática
- Estruturas de dados avançadas
- Com “baterias incluídas” (alguns módulos nativos: os, cgi, ftplib, gzip, math, re, xmllib, sockets, entre inúmeros outros)

Qualidades

- Código aberto e gratuito
- Extremamente portável, bytecode/máquina virtual
- Genérica, flexível e extensível
- Bem projetada
- Em crescimento, comunidade ativa (última versão/documentação de Abril/2007)
- Pacote interessantes (alguns módulos instaláveis: administração de sistemas, interfaces gráficas, internet, banco de dados, programação científica, inteligência artificial, entre inúmeros outros)

Sintaxe

- Elegante e simples
- Tipagem dinâmica (não-declarativa)
- Flexibilidade no tratamento de argumentos
- Interface simples para estruturas complexas
- Rápida prototipação (RAD), ambiente de testes
- Blocos marcados pela indentação
- Legibilidade

Instalação

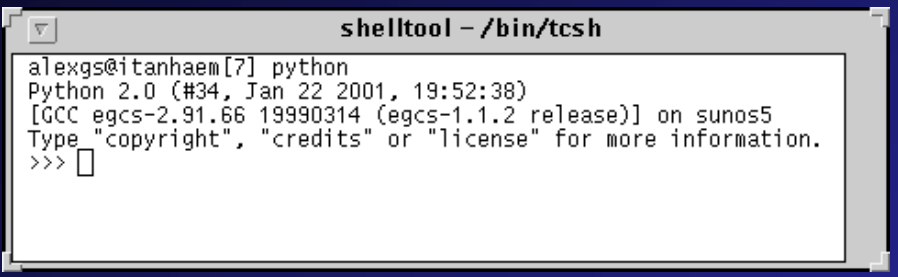
- No Windows
  - *Download* do executável em http://www.python.org/download
  - Versão atual (23/10/2007): python-2.5.1.msi
- No Linux
  - Usar `apt-get install python` ou `urpmi python` ou ...
- Outros (Unix-based)
  - *Download* do fonte (Ex.: `Python-2.5.1.tgz`)
    1. Descompactar (Ex.: `tar zxvf Python-2.5.1.tgz`)
    2. Entrar no diretório (Ex.: `cd Python-2.5.1`)
    3. Configurar (Ex.: `./configure --prefix=nome_do_diretório`)
    4. Editar Modules/Setup para configurar a interface gráfica
       - Na seção _tkinter, indicar os diretórios das bibliotecas do X11, X11R6 e dos programas Tcl e Tk.
    5. Compilar (Ex.: `make`)
    6. Instalar (Ex.: `make install`)
    7. Digitar `python`

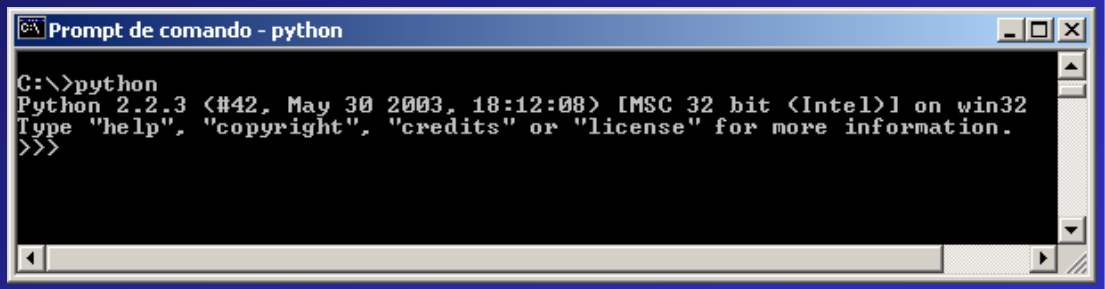
Uso do interpretador

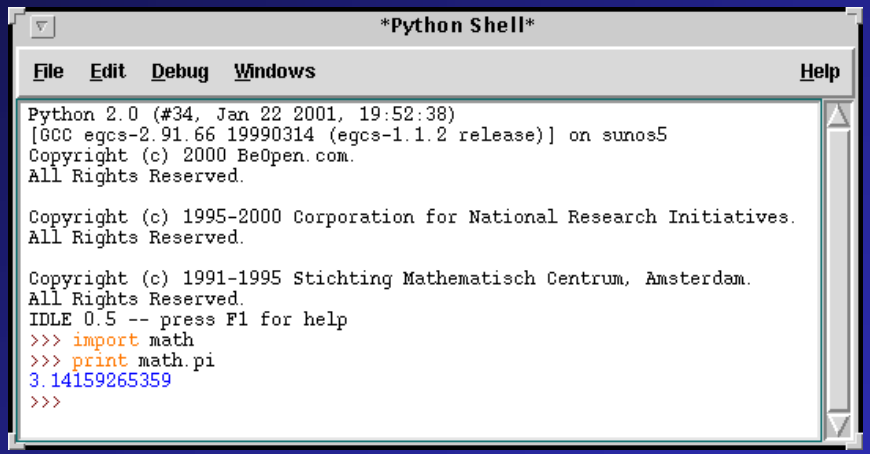
- Acrescentar o caminho do diretório de instalação do Python (se for necessário)
  - No Windows: alterar via variáveis de ambiente em propriedades do sistema ou pelo DOS digitando: `set path=%path%;C:\python25`
  - No Unix/Linux: incluir no .bashrc (p. ex.) de inicialização:
    ```
    export PATH=/diretorio_do_python/bin:$PATH
    ```
  - Digitar no prompt do DOS ou shell do Unix/Linux: `python`
  - Executar um script em particular: `python nomeDoScript.py`
- Script auto-executável (Unix/Linux)
  - Colocar, na primeira linha do script: `#! /diretorio_do_python python`
  - Mudar permissão do arquivo para executável: `chmod +x meuScript.py`
- Codificação de caracteres
  - Acrescentar esta linha para uso de caracteres Unicode (com acentuação):
    ```
    # -*- coding: iso-8859-15 -*-
    ```

Plataformas e Ambientes

- Algumas plataformas







- Entre várias outras ...
- Ambientes de desenvolvimento
  - IDLE, PyCrust, ipython, Anjuta, BoaConstructor, KDevelop, PythonWin, emacs, ...

C3

Olá, mundo!

- Saída padrão - `print`

1
```
>>> print "Olá, mundo!"
Olá, mundo!
```

2
```
>>> nome = "mundo"
>>> print "Olá, %s!" %nome
Olá, mundo!
```

3
```
>>> nome = "mundo"
>>> print "Olá,", nome
Olá, mundo!
```

4
```
>>> nome = "mundo"
>>> print "Olá, "+nome
Olá, mundo!
```

Entrada de dados

- Entrada de string – `raw_input`

```
>>> nome = raw_input("Entre com seu nome: ")
Entre com seu nome: Alexandre
>>> print nome
Alexandre
>>> type(nome)
<type 'str'>
```

- Entrada de valor – `input`

```
>>> idade = input("Entre com sua idade: ")
Entre com sua idade: 20
>>> print idade
20
>>> type(idade)
<type 'int'>
```

Documentação

- Comentários (#)

```
>>> # Esta linha é um comentário
```

- Strings e documentação (""" ou ''')

1
```
>>> variavel = """Tudo daqui
pra frente é
documentação"""
>>> print variavel
Tudo daqui
pra frente é
Documentação
```

2
```
>>> def teste(): #definição de função
        """Tudo daqui
           pra frente é
           documentação"""
        return
>>> teste()
```

Números

- Inteiros

```
>>> 1234
1234
>>> 99999999L
99999999L
>>> (16-3*2)/2
5
>>> 7 / 2
3
>>> 7 % 2
1
>>> 077 #octal
63
>>> 0xFF #hexa
255
```

- Reais

```
>>> 1.234
1.234
>>> 123.4e-2
1.234
>>> 7.0 / 2
3.5
>>> 2.0**3
8.0
>>> round(9.6)
10
```

- Complexos

```
>>> complex(2, -1)
(2-1j)
>>> 1j * 1J
(-1+0j)
>>> a = 3.0 + 4.0j
>>> a.real
3.0
>>> a.imag
4.0
>>> abs(a)
5.0
```

Atribuição e Igualdade

- Atribuição (=)

```
>>> valor = 9.5
>>> a, b, c = 10, 20, 30
>>> x = y = 100
```

- Igualdade (`is` ou ==) e
  Diferença (`is not` ou !=)

1
```
>>> x = 5
>>> TESTE = x is 5
>>> print TESTE
True
```

2
```
>>> TESTE = x is not 5
>>> print TESTE
False
```

Lógicos e Relacionais

- Operadores lógicos
  - `or`
  - `and`
- Operadores relacionais
  - `<`
  - `>`
  - `<=`
  - `>=`
- Operadores lógicos bit-a-bit
  - `|`
  - `&`
- Deslocamento de bits
  - `<<`
  - `>>`
- Exemplos

1
```
>>> x = 15
>>> TESTE = x < 5 or (x > 10 and x <= 20)
>>> print TESTE
True
```

2
```
>>> TESTE = x < 5 or 10 < x <= 20
>>> print TESTE
True
```

Built-in (1)

- Alguns métodos:

abs, apply, bool, buffer, callable, chr, classmethod, cmp, coerce, compile, **complex**, copyright, credits, delattr, dict, dir, divmod, eval, execfile, exit, file, filter, float, getattr, globals, hasattr, hash, help, hex, id, **input**, int, intern, isinstance, issubclass, iter, len, license, list, locals, long, map, max, min, object, oct, open, ord, pow, property, quit, range, **raw_input**, reduce, reload, repr, **round**, setattr, slice, staticmethod, str, super, tuple, type, unichr, unicode, vars, xrange, zip

- Exibição de atributos - `dir()`

1
```
>>> dir() # escopo global
['__builtins__', '__doc__', '__name__']
```

2
```
>>> dir(__builtins__) # mostra os métodos builtins acima
```

3
```
>>> x = 2 + 1j
>>> dir(x) # mostra os métodos de número complexo
```

Built-in (2)

- Ajuda - `help()`

```
>>> help(max)
Help on built-in function max in module __builtin__:

max(...)
    max(iterable[, key=func]) -> value
    max(a, b, c, ...[, key=func]) -> value

    With a single iterable argument, return its largest item.
    With two or more arguments, return the largest argument.
```

- Identificação de tipo - `type()`

```
>>> nota = 9.5
>>> type(nota)
<type 'float'>
```

Strings (1)

- Definição (' ou " ou ''' ou """)

1
```
>>> 'teste...'
'teste...'
```

2
```
>>> 'teste \'2\'...'
"teste '2'..."
```

3
```
>>> "teste '3'..."
"teste '3'..."
```

4
```
>>> '''"teste '4'..." foi escrito no quadro'''
'"teste \'4\'..." foi escrito no quadro'
```

5
```
>>> """Esta string
apresenta mais
de uma linha."""
'Esta string\napresenta mais\nde uma linha.'
```

Strings (2)

- Vazia

```
>>> s1 = ''
```

- Tamanho – `len()`

```
>>> s2 = 'UDESC'
>>> len(s2)
5
```

- Concatenação (+)

```
>>> 'DCC/' + s2
'DCC/UDESC'
```

- Replicação (*)

```
>>> 3*s2
'UDESCUDESCUDESC'
```

Strings (3)

- *Slicing* - string[inicio:fim:passo]

```
>>> palavra = 'UDESC'
1 >>> palavra[2]
'E'
2 >>> palavra[0:3]
'UDE'
3 >>> palavra[3:]
'SC'
4 >>> palavra[-1]
'C'
5 >>> palavra[-2:]
'SC'
6 >>> palavra[::2]
'UEC'
7 >>> palavra[::-1]
'CSEDU'
```

Strings (4)

- Transformação em string – str(objeto) ou `objeto`

```
>>> complexo = 2.5-1.0j

>>> type(complexo)
<type 'complex'>

>>> texto = `complexo`
>>> type(texto)
<type 'str'>

>>> complexo
(2.5-1j)

>>> texto
'(2.5-1j)'
```

Strings (5)

- Exemplo

```
>>> opcao = 'Interpolacao_Linear'
>>> opcao.upper()
'INTERPOLACAO_LINEAR'
>>> opcao.upper().find('LIN')
13
>>> opcao.split('_')
['Interpolacao', 'Linear']
```

1
2
3

- Alguns métodos:

capitalize, center, count, decode, encode, endswith, expandtabs, **find**, index, isalnum, isalpha, isdigit, islower, isspace, istitle, isupper, join, ljust, lower, lstrip, replace, rfind, rindex, rjust, rstrip, **split**, splitlines, startswith, strip, swapcase, title, translate, **upper**, zfill

Listas (1)

- Definição, slicing, concatenação e replicação

```
1  >>> lista = []                              # lista vazia

2  >>> lista = [10, 20, 'aa', 'bb', 8.5] # definição

3  >>> print lista
   [10, 20, 'aa', 'bb', 8.5]

4  >>> lista[2:4]                              # slicing
   ['aa', 'bb']

5  >>> lista[:3] + [30, 40, 10*5]          # concatenação
   [10, 20, 'aa', 30, 40, 50]

6  >>> 3*lista[-2:] + ['cc']                   # replicação
   ['bb', 8.5, 'bb', 8.5, 'bb', 8.5, 'cc']
```

Listas (2)

- Alteração, remoção e inserção

```
1 >>> lista = [10, 20, 'aa', 'bb', 8.5]
2 >>> lista[1] = lista[1] + 5 # alteração individual
  >>> lista
  [10, 25, 'aa', 'bb', 8.5]
3 >>> lista[2:4] = [30, 40]   # alteração de itens
  >>> lista
  [10, 25, 30, 40, 8.5]
4 >>> lista[0:2] = []         # remoção
  >>> lista
  [30, 40, 8.5]
5 >>> lista[1:1] = [100, 200] # inserção
  >>> lista
  [30, 100, 200, 40, 8.5]
6 >>> lista.append(2.5)       # médodo append
  >>> lista
  [30, 100, 200, 40, 8.5, 2.5]
```

Listas (3)

- Sub-listas aninhadas

```
>>> lista = [30, 40, 8.5]
>>> lista[1] = [100, 200]
>>> lista
[30, [100, 200], 8.5]
>>> lista[1][0]
100
>>> len(lista)
3
```

- Alguns métodos

```
append, count, extend, index, insert, pop, remove,
reverse, sort
```

Exercícios A

1. Faça um *script* que entre com um número entre 0 e 99999 pelo teclado e produza um string com 5 dígitos (com zeros a esquerda se for o caso).

2. Considere a definição da seguinte lista de inteiros:

```
>>> lista = [50, 100, 80, 5, 90, 70, 40, 30, 10, 1, 20]
```

   Faça um *script* que, a partir desta, retorne:
   - O mínimo e máximo
   - O valor médio
   - Nova lista com ordenação descrescente

   Observação: utilize somente os métodos builtins e de lista

Primeiro programa

- Série de Fibonacci

```
>>> a, b = 0, 1
>>> while b < 100:
        print b,
        a, b = b, a+b

1 1 2 3 5 8 13 21 34 55 89
```

- Considerações
  - Blocos de comando marcados pela indentação
  - Após "**:**" é feita indentação automaticamente no *IDLE*
  - Criaremos *scripts* no *NewWindow* do *IDLE* para programas maiores

Controle condicional

- Comando `if-elif-else`

```
x = input('Entre com um inteiro: ')
if x == 0:
   print 'Zero'
elif x == 1:
   print 'Um'
elif x < 0:
   print 'Negativo'
else:
   print 'Positivo'
```

Controle iterativo (1)

- Comandos `for` e `while`

```
for i in [1,2,3,4,5]:
   print 'Esta é a iteração número', i
```

```
x = 10
while x >= 0:
   print 'x ainda não é negativo.'
   x = x - 1
```

- Intervalos – `range(inicio,fim,passo)`

```
range(2,10,3)
```

```
for valor in range(100):
   print valor
```

```
nomes = ['Fulano', 'Ciclano', 'Beltrano']
for i in range(len(nomes)):
   print i+1, nomes[i]
```

Controle iterativo (2)

- Comandos `else`, `break` e `continue`

```
for n in range(2, 10):
    for x in range(2, n):
        if n % x == 0:
            print n, '=', x, '*', n/x
            break
    else:
        print n, 'é primo!'
```

- Comando `pass`

```
while True:
    pass
```

- Outro exemplo

```
nomes = ['Fulano', 'Ciclano', 'Beltrano']
for i in nomes:
    print i
```

Funções (1)

- Comandos `def` e `return`

```
def soma(k1, k2=10):
    '''Soma de inteiros no intervalo [k1,k2]'''
    soma = 0
    valores = []
    for i in range(k1, k2+1):
        soma = soma + i
        valores.append(i)
    return soma, valores
```

```
>>> help(soma)
>>> s1, s2 = soma(3,5)
>>> s1
12
>>> s2
[3, 4, 5]
>>> soma(8)
(27, [8, 9, 10])
```

Funções (2)

- Palavra-chave em parâmetros

1
```
def figura(nome, cor_borda='preto', cor_fundo='branco',
           altura=10, largura=10):
        print 'Nome da figura:', nome
        print 'Cores: borda=%s, fundo=%s' %(cor_borda,cor_fundo)
        print 'Dimensoes: altura=%d, largura=%d' %(altura,largura)
```

```
>>> figura('elipse',altura=50)
Nome da figura: elipse
Cores: borda=preto, fundo=branco
Dimensoes: altura=50, largura=10
```

2
```
def teste(a, *parametros, **palavras_chave):
        for p in parametros: print p,
        print a
        chaves = palavras_chave.keys()
        for c in chaves: print c, ':', palavras_chave[c]
```

```
>>> teste('abc',10,20,30,x='aaa',y='bbb',z='ccc')
10 20 30 abc
y : bbb
x : aaa
z : ccc
```

Funções (3)

- Função `lambda`:

1
```
def incrementa(n):
    return lambda x: x + n
```

```
>>> f = incrementa(10)
>>> f(0)
10
>>> f(1)
11
```

2
```
def componha(f1, f2):
    return lambda x, f1=f1, f2=f2: f1(f2(x))
```

```
>>> from math import sin, cos
>>> sincos = componha(sin, cos)
>>> print sincos(3)
-0.836021861538
>>> print sin(cos(3))
-0.836021861538
```

Funções (4)

- Ferramentas de programação funcional
  - filter
  - map
  - reduce

1
```
>>> def pares(x): return (x % 2) == 0

>>> filter(pares, range(10))
[0, 2, 4, 6, 8]
```

2
```
>>> def quadrado(x): return x*x

>>> map(quadrado, range(10))
[0, 1, 4, 9, 16, 25, 36, 49, 64, 81]
```

3
```
>>> def soma(x,y): return x+y

>>> reduce(soma, range(10))
45
```

Funções (5)

- Recursividade

```
def fatorial(n):
        if n == 0:
                return 1
        else:
                return n * fatorial(n-1)
```

Erros e Exceções

- Erros
  - ZeroDivisionError
  - NameError
  - TypeError
  - ...

```
>>> 1/0
Traceback (most recent call last):
  File "<stdin>", line 1, in ?
ZeroDivisionError: integer division or modulo by zero
```

- Exceções

```
def divisao_segura(a,b):
    try:
        return a/b
    except ZeroDivisionError:
        print "Erro: divisão por zero!"
    except:
        print "Erro inesperado..."
    return None
```

Estruturas de Dados (1)

- Listas como pilhas

```
>>> pilha = [1, 2, 3]
>>> pilha.append(4)
>>> pilha
[1, 2, 3, 4]
>>> pilha.pop()
4
>>> pilha
[1, 2, 3]
```

- Listas como filas

```
>>> fila = ["Ciclano","Beltrano"]
>>> fila.append("Fulano")
>>> fila
['Ciclano', 'Beltrano', 'Fulano']
>>> fila.pop(0)
'Ciclano'
>>> fila
['Beltrano', 'Fulano']
```

Estruturas de Dados (2)

- Tuplas

1
```
>>> tupla = 12345, 54321, 'oi!'
```

2
```
>>> tupla
(12345, 54321, 'oi!')
```

3
```
>>> tupla[0]
12345
```

4
```
>>> x, y, z = tupla
>>> x
12345
>>> y
54321
>>> z
'oi!'
```

Estruturas de Dados (3)

- Conjuntos

1
```
>>> cesta = ['maca', 'laranja', 'maca', 'pera', 'laranja',
             'banana']
>>> frutas = set(cesta)
>>> frutas
set(['pera', 'laranja', 'banana', 'maca'])
>>> 'laranja' in frutas
True
>>> 'melao' in frutas
False
```

2
```
>>> a = set('abracadabra')
>>> b = set('alacazam')
>>> a - b
set(['r', 'b', 'd'])
>>> a | b
set(['a', 'c', 'b', 'd', 'm', 'l', 'r', 'z'])
>>> a & b
set(['a', 'c'])
>>> a ^ b
set(['b', 'd', 'm', 'l', 'r', 'z'])
```

Estruturas de Dados (4)

- Dicionários

```
1  >>> fone = {"aa" : 3232, "bb" : 7575, "cc" : 7777}

2  >>> fone
   {'aa': 3232, 'cc': 7777, 'bb': 7575}

3  >>> fone["bb"]
   7575

4  >>> fone.keys()
   ['aa', 'cc', 'bb']

5  >>> fone.has_key("cc")
   True
```

Classes (1)

- Exemplo

```
class agenda:
   def __init__(self, cont=None):  # construtor
      self.conteudo = cont or []
   def adicione(self, nome):
      self.conteudo.append(nome)
```

```
>>> a = agenda()
>>> a.adicione('Fulano')
>>> a.adicione('Ciclano')
>>> a.conteudo
['Fulano', 'Ciclano']
```

Classes (2)

- Sobrecarga de operadores

```
class racional:
   def __init__(self, num, den):
      self.num = num
      self.den = den
   def __str__(self):
      return str(self.num) + '/' + str(self.den)
   def __mul__(a, b):
      c = racional(a.num*b.num, a.den*b.den)
      return c
```

```
>>> a = racional(1,3)
>>> b = racional(5,2)
>>> c = a * b
>>> c.num
5
>>> c.den
6
>>> print c
5/6
```

Classes (3)

- Herança

```
class NomeClasseDerivada(NomeClasseBase):
    …
```

- Herança múltipla

```
class NomeClasseDerivada(Base1, Base2, Base3):
    …
```

Exercícios B

```
class racional:
   def __init__(self, num, den):
      self.num = num
      self.den = den
   def __str__(self):
      return str(self.num) + '/' + str(self.den)
   def __mul__(a, b):
      c = racional(a.num*b.num, a.den*b.den)
      return c
```

1. Alterar o tipo de dado abstrato números racionais para que não permita denominador igual a zero.

2. Implementar a redução da razão.

   Exemplo: `6/27 = 2/9`

   Algoritmo MDC:

```
se b>a faça: troque a e b entre si
enquanto b>0 faça: r=a%b; a=b; b=r
          "a" será o MDC ao final do laço
```

Arquivos

- Leitura e escrita

```
>>> f = open('/tmp/texto.txt', 'w')
```

- Métodos

```
>>> f.read()
'Isto é o que está dentro do arquivo.\n'
```

```
>>> f.readline()
'Ista é a primeira linha do arquivo.\n‘
>>> f.readline()
'Ista é a segunda linha do arquivo.\n'
```

```
>>> f.write('Escrevendo este texto\n')
```

```
>>> f.seek(5)
>>> f.read(1)
>>> f.close()
```

Módulos

- Módulos
  - Arquivos texto com código Python
  - Arquivo compilado em C/C++
  - Exemplos:

1
```
>>> import math
>>> dir(math)
['__doc__', '__name__', 'acos', 'asin', 'atan', 'atan2',
'ceil', 'cos', 'cosh', 'e', 'exp', 'fabs', 'floor',
'fmod', 'frexp', 'hypot', 'ldexp', 'log', 'log10', 'modf',
'pi', 'pow', 'sin', 'sinh', 'sqrt', 'tan', 'tanh']
>>> a = math.cos(math.pi)
```
2
```
>>> from math import sin, cos, tan
>>> a = cos(math.pi)
```
3
```
>>> from math import *
>>> a = cos(pi)
```

Execução de módulos

- Modos de execução de um *script* em arquivo:
  - $ python nome.py
  - *run script* (*F5*) no IDLE
  - *execute buffer* (*Ctrl+c Ctrl+c*) no emacs
  - Script executável no Unix:

```
#! /usr/bin/env python
```

- Organização em módulos:
  - `>>> import meu_módulo`
  - `>>> from meu_módulo import minhas_funções`
  - `>>> from meu_módulo import *`
  - Módulo executável:

```
if __name__ == "__main__": executar_função()
```

Alguns Módulos

- Alguns módulos inclusos:
  - math
  - time
  - string
  - sys
  - os
  - cgi
  - Tkinter
- Alguns módulos separados:
  - pygame
  - PyOpenGL
  - PIL
  - NumPy

Módulo time

- Exemplo:

```
from time import time

t0 = time()
#...
#Colocar o trecho de código aqui
#...
t = time() - t0
print 't = %.4f \n' %t
```

Módulo string

- Exemplo:

```
1  >>> import string

2  >>> modulos = ["math", "string", "Numeric"]

3  >>> comando = "import "+string.join(modulos, ", ")

4  >>> print comando
   import math, string, Numeric

5  >>> exec(comando)
```

Módulo sys

- Exemplo:

```
1 >>> import sys
2 >>> sys.version
  2.0 (#34, Jan 22 2001, 19:52:38)
  [GCC egcs-2.91.66 19990314 (egcs-1.1.2 release)]
3 >>> sys.path.append('/tmp')
  >>> print sys.path
  ['/home/spec/alexgs/python/idle-0.5',
  '/home/spec/alexgs/python2.0/lib/python2.0',
  '/home/spec/alexgs/python2.0/lib/python2.0/plat-sunos5',
  '/home/spec/alexgs/python2.0/lib/python2.0/lib-tk',
  '/home/spec/alexgs/python2.0/lib/python2.0/lib-dynload',
  '/home/spec/alexgs/python2.0/lib/python2.0/site-packages',
  '/home/spec/alexgs/python2.0/lib/python2.0/site-packages/
  Numeric', '/tmp']
```

```
setenv PYTHONPATH .:/dir_1:/dir_2:...:/dir_n
```

Módulo cgi

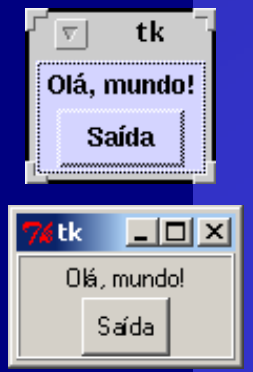
```
#!/bin/python
# Calendário dinâmico

print 'Content-type: text/html\n'
print '<HTML><TITLE>Calendário Dinâmico</TITLE>'
print '<BODY>'
try:
   import cgi
   from time import time, localtime   # calendar
   ano, mes, dia = localtime(time())[:3]
   print '<H1>Joinville, %02d/%02d/%04d</H1>' %(dia, mes, ano)
   print '...'
   opcao = cgi.FieldStorage()
   ...
except:
   print '<HR><H3>Erro no CGI!</H3><PRE>'
print '</BODY></HTML>'
```

Módulo Tkinter (1)

- Exemplos:

```
from Tkinter import *
root = Tk()
w = Label(root, text="Olá, mundo!")
w.pack()
root.mainloop()
```

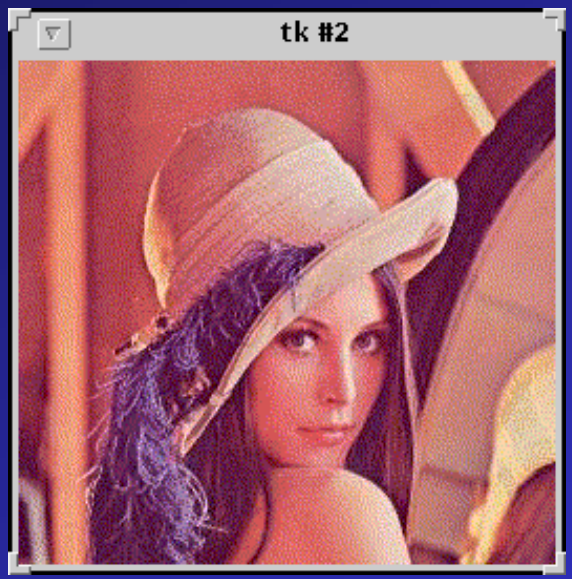

```
import Tkinter
from Tkconstants import *
tk = Tkinter.Tk()
frame = Tkinter.Frame(tk, relief=RIDGE, borderwidth=2)
frame.pack(fill=BOTH, expand=1)
label = Tkinter.Label(frame, text="Olá, mundo!")
label.pack(fill=X, expand=1)
button = Tkinter.Button(frame, text="Saída",
                        command=tk.destroy)
button.pack(side=BOTTOM)
tk.mainloop()
```



Módulo Tkinter (2)

- Exemplo:

```
>>> from ia636 import *  # em www.dca.fee.unicamp.br/ia636
>>> f = iaread('nome_do_arquivo.ppm')
>>> f_= iashow(f)
```



Exercícios C

- Crie dois arquivos texto – A e B – com *N* números aleatórios inteiros. Ex. de número entre [0,1000):

```
>>> import random
>>> a = random.randint(0,1000)
```

- Faça uma função que leia dois arquivos texto com números inteiros e:
  1. Crie uma lista de inteiros para cada arquivo
  2. Crie uma terceira lista, resultado uma operação qualquer (soma, máximo, ...) elemento a elemento das duas primeiras
  3. Calcule o erro quadrático médio (MSE) entre as duas primeiras listas:

$$\widehat{\mathrm{MSE}}(\hat{\theta}) = \frac{1}{n}\sum_{j=1}^{n}(\theta_j - \theta)^2$$

Computação científica

Introdução (1)

"Computação científica (ou ciência computacional) é o campo de estudo interessado na construção de modelos matemáticos e técnicas de soluções numéricas utilizando computadores para analisar e resolver problemas científicos e de engenharia. De forma prática, é a aplicação de simulação computacional e outras formas de computação para problemas em diversas disciplinas científicas."
(*Wikipédia*, visitado em 27nov07)

"… área de atividade/conhecimento que envolve a utilização de ferramentas computacionais (software) para a solução de problemas científicos em áreas da ciência não necessariamente ligadas à disciplina da ciência da computação, ou seja, a computação para o restante da comunidade científica."
(*Flávio Coelho*, 2007)

Introdução (2)

- Alguns domínios
  - Simulação numérica
  - Predições
  - Análise de dados
  - Visualização científica
  - Reconhecimento de padrões
  - Processamento gráfico
  - Inteligência artificial
  - Redes
  - Pesquisa operacional
  - Contabilidade, economia
  - Ensino de matemática, física, biologia, ...

Introdução (3)

- Linguagens usuais
  - FORTRAN
  - C
- Linguagens facilitadoras
  - MATLAB
  - GNU Octave
  - SciLab
  - Mathematica

Python
+
Módulos auxiliares

C3

Pacote numérico

- www.numpy.org
- Computação científica
- Array multidimensional
- Próximo à sintaxe do MATLAB
- Histórico
  - Numeric
  - numarray
  - numpy
- Módulos
  - Principal
  - Análise de Fourier
  - Álgebra Linear
  - ...

Lista de funções (1)

```
>>> import numpy
>>> dir(numpy)
['ALLOW_THREADS', 'BUFSIZE', 'CLIP', 'ERR_CALL', 'ERR_DEFAULT', 'ERR_DEFAULT2', 'ERR_IGNORE',
'ERR_LOG', 'ERR_PRINT', 'ERR_RAISE', 'ERR_WARN', 'FLOATING_POINT_SUPPORT', 'FPE_DIVIDEBYZERO',
'FPE_INVALID', 'FPE_OVERFLOW', 'FPE_UNDERFLOW', 'False_', 'Inf', 'Infinity', 'MAXDIMS', 'MachAr',
'NAN', 'NINF', 'NZERO', 'NaN', 'NumpyTest', 'PINF', 'PZERO', 'PackageLoader', 'RAISE',
'RankWarning', 'SHIFT_DIVIDEBYZERO', 'SHIFT_INVALID', 'SHIFT_OVERFLOW', 'SHIFT_UNDERFLOW',
'ScalarType', 'ScipyTest', 'True_', 'UFUNC_BUFSIZE_DEFAULT', 'UFUNC_PYVALS_NAME', 'WRAP',
'__all__', '__builtins__', '__config__', '__doc__', '__file__', '__name__', '__path__',
'__version__', '_import_tools', 'abs', 'absolute', 'add', 'add_docstring', 'add_newdoc',
'add_newdocs', 'alen', 'all', 'allclose', 'alltrue', 'alterdot', 'amax', 'amin', 'angle', 'any',
'append', 'apply_along_axis', 'apply_over_axes', 'arange', 'arccos', 'arccosh', 'arcsin',
'arcsinh', 'arctan', 'arctan2', 'arctanh', 'argmax', 'argmin', 'argsort', 'argwhere', 'around',
'array', 'array2string', 'array_equal', 'array_equiv', 'array_repr', 'array_split', 'array_str',
'asanyarray', 'asarray', 'asarray_chkfinite', 'ascontiguousarray', 'asfarray', 'asfortranarray',
'asmatrix', 'asscalar', 'atleast_1d', 'atleast_2d', 'atleast_3d', 'average', 'bartlett',
'base_repr', 'binary_repr', 'bincount', 'bitwise_and', 'bitwise_not', 'bitwise_or',
'bitwise_xor', 'blackman', 'bmat', 'bool', 'bool8', 'bool_', 'broadcast', 'byte', 'byte_bounds',
'c_', 'can_cast', 'cast', 'cdouble', 'ceil', 'cfloat', 'char', 'character', 'chararray',
'choose', 'clip', 'clongdouble', 'clongfloat', 'column_stack', 'common_type',
'compare_chararrays', 'complex', 'complex128', 'complex64', 'complex_', 'complexfloating',
'compress', 'concatenate', 'conj', 'conjugate', 'convolve', 'copy', 'core', 'corrcoef',
'correlate', 'cos', 'cosh', 'cov', 'cross', 'csingle', 'ctypeslib', 'cumprod', 'cumproduct',
'cumsum', 'delete', 'deprecate', 'diag', 'diagflat', 'diagonal', 'diff', 'digitize', 'disp',
'distutils', 'divide', 'dot', 'double', 'dsplit', 'dstack', 'dtype', 'e', 'ediff1d', 'emath',
'empty', 'empty_like', 'equal', 'errstate', 'exp', 'expand_dims', 'expm1', 'extract', 'eye',
'fabs', 'fastCopyAndTranspose', 'fft', 'finfo', 'fix', 'flatiter', 'flatnonzero', 'flexible',
'fliplr', 'flipud', 'float', 'float32', 'float64', 'float_', 'floating', 'floor', 'floor_divide',
```

Lista de funções (2)

```
'fmod', 'format_parser', 'frexp', 'frombuffer', 'fromfile', 'fromfunction', 'fromiter',
'frompyfunc', 'fromstring', 'generic', 'get_array_wrap', 'get_include', 'get_numarray_include',
'get_numpy_include', 'get_printoptions', 'getbuffer', 'getbufsize', 'geterr', 'geterrcall',
'geterrobj', 'gradient', 'greater', 'greater_equal', 'hamming', 'hanning', 'histogram',
'histogram2d', 'histogramdd', 'hsplit', 'hstack', 'hypot', 'i0', 'identity', 'iinfo', 'imag',
'index_exp', 'indices', 'inexact', 'inf', 'info', 'infty', 'inner', 'insert', 'int', 'int0',
'int16', 'int32', 'int64', 'int8', 'int_', 'int_asbuffer', 'intc', 'integer', 'interp',
'intersect1d', 'intersect1d_nu', 'intp', 'invert', 'iscomplex', 'iscomplexobj', 'isfinite',
'isfortran', 'isinf', 'isnan', 'isneginf', 'isposinf', 'isreal', 'isrealobj', 'isscalar',
'issctype', 'issubclass_', 'issubdtype', 'issubsctype', 'iterable', 'ix_', 'kaiser', 'kron',
'ldexp', 'left_shift', 'less', 'less_equal', 'lexsort', 'lib', 'linalg', 'linspace',
'little_endian', 'load', 'loads', 'loadtxt', 'log', 'log10', 'log1p', 'log2', 'logical_and',
'logical_not', 'logical_or', 'logical_xor', 'logspace', 'long', 'longcomplex', 'longdouble',
'longfloat', 'longlong', 'ma', 'mat', 'math', 'matrix', 'max', 'maximum', 'maximum_sctype',
'may_share_memory', 'mean', 'median', 'memmap', 'meshgrid', 'mgrid', 'min', 'minimum',
'mintypecode', 'mod', 'modf', 'msort', 'multiply', 'nan', 'nan_to_num', 'nanargmax', 'nanargmin',
'nanmax', 'nanmin', 'nansum', 'nbytes', 'ndarray', 'ndenumerate', 'ndim', 'ndindex', 'negative',
'newaxis', 'newbuffer', 'nonzero', 'not_equal', 'number', 'obj2sctype', 'object', 'object0',
'object_', 'ogrid', 'ones', 'ones_like', 'outer', 'pi', 'piecewise', 'pkgload', 'place', 'poly',
'poly1d', 'polyadd', 'polyder', 'polydiv', 'polyfit', 'polyint', 'polymul', 'polysub', 'polyval',
'power', 'prod', 'product', 'ptp', 'put', 'putmask', 'r_', 'random', 'rank', 'ravel', 'real',
'real_if_close', 'rec', 'recarray', 'reciprocal', 'record', 'remainder', 'repeat', 'require',
'reshape', 'resize', 'restoredot', 'right_shift', 'rint', 'roll', 'rollaxis', 'roots', 'rot90',
'round', 'round_', 'row_stack', 's_', 'savetxt', 'sctype2char', 'sctypeDict', 'sctypeNA',
'sctypes', 'searchsorted', 'select', 'set_numeric_ops', 'set_printoptions',
'set_string_function', 'setbufsize', 'setdiff1d', 'seterr', 'seterrcall', 'seterrobj',
'setmember1d', 'setxor1d', 'shape', 'short', 'show_config', 'sign', 'signbit', 'signedinteger',
'sin', 'sinc', 'single', 'singlecomplex', 'sinh', 'size', 'sometrue', 'sort', 'sort_complex',
'source', 'split', 'sqrt', 'square', 'squeeze', 'std', 'str', 'str_', 'string0', 'string_',
'subtract', 'sum', 'swapaxes', 'take', 'tan', 'tanh', 'tensordot', 'test', 'testing', 'tile',
'trace', 'transpose', 'trapz', 'tri', 'tril', 'trim_zeros', 'triu', 'true_divide', 'typeDict',
'typeNA', 'typecodes', 'typename', 'ubyte', 'ufunc', 'uint', 'uint0', 'uint16', 'uint32',
'uint64', 'uint8', 'uintc', 'uintp', 'ulonglong', 'unicode', 'unicode0', 'unicode_', 'union1d',
'unique', 'unique1d', 'unravel_index', 'unsignedinteger', 'unwrap', 'ushort', 'vander', 'var',
'vdot', 'vectorize', 'version', 'void', 'void0', 'vsplit', 'vstack', 'where', 'who', 'zeros',
'zeros_like']
```

Lista de métodos (1)

- Criação de array, tipos e métodos

```
>>> import numpy
>>> a = numpy.array([1,2,3])

>>> type(a)
<type 'numpy.ndarray'>

>>> a.dtype
dtype('int32')

>>> b = a.astype('float')
>>> b
array([ 1.,  2.,  3.])
```

1 2 3 4

Lista de métodos (2)

- Listagem de métodos de array

```
>>> a = numpy.array([1,2,3])

>>> dir(a)
['T', 'all', 'any', 'argmax', 'argmin', 'argsort',
'astype', 'base', 'byteswap', 'choose', 'clip', 'compress',
'conj', 'conjugate', 'copy', 'ctypes', 'cumprod', 'cumsum',
'data', 'diagonal', 'dtype', 'dump', 'dumps', 'fill',
'flags', 'flat', 'flatten', 'getfield', 'imag', 'item',
'itemset', 'itemsize', 'max', 'mean', 'min', 'nbytes',
'ndim', 'newbyteorder', 'nonzero', 'prod', 'ptp', 'put',
'ravel', 'real', 'repeat', 'reshape', 'resize', 'round',
'searchsorted', 'setfield', 'setflags', 'shape', 'size',
'sort', 'squeeze', 'std', 'strides', 'sum', 'swapaxes',
'take', 'tofile', 'tolist', 'tostring', 'trace',
'transpose', 'var', 'view']
```

Criação de arrays

- Criação de array

```
>>> import numpy
>>> numpy.array([1,2,3])                # array 1D
array([1, 2, 3])
>>> numpy.array([[1,2,3],[4,5,6]]) # array 2D
array([[1, 2, 3],
       [4, 5, 6]])
>>> numpy.arange(0,10,2)
array([0, 2, 4, 6, 8])
>>> numpy.zeros((2,5))
array([[ 0.,  0.,  0.,  0.,  0.],
       [ 0.,  0.,  0.,  0.,  0.]])
```

1 2 3 4

- Dimensão de um array

```
>>> lista = numpy.array([[1,2,3],[4,5,6]])
>>> lista.shape
(2, 3)
```

Arrays no numpy

- 1D x 2D

```
>>> a = numpy.arange(0.0,1.0,0.1)   # a.T nao se altera
>>> print a
[ 0.   0.1  0.2  0.3  0.4  0.5  0.6  0.7  0.8  0.9]
>>> a.shape
(10,)
>>> b = a[numpy.newaxis,:]
>>> b.shape
(1, 10)
>>> c = a[:,numpy.newaxis]
>>> c.shape
(10, 1)
```

- Funções matemáticas

```
>>> y = numpy.sin(a)
>>> print y
[ 0.          0.09983342  0.19866933  0.29552021  0.38941834  0.47942554
  0.56464247  0.64421769  0.71735609  0.78332691]
```

Algumas operações

- Transposição

1
```
>>> lista = numpy.array([[1,2,3]])
>>> lista.T   # numpy.transpose(lista) ou lista.transpose()
array([[1],
       [2],
       [3]])
```

- Operações numéricas e multiplicação de matrizes

2
```
>>> a = numpy.array([[1,3],[-2,0]])
>>> b = 2 * a
>>> c = a * b
```

3
```
>>> numpy.dot(a, b)
array([[-10,   6],
       [ -4, -12]])
```

Processamentos simples

- Criação, valores diferentes de zeros, limiarizações

1
```
>>> a = numpy.array([[1,0,0],[0,5,6],[7,0,0],[0,11,0]])
>>> print a
[[ 1  0  0]
 [ 0  5  6]
 [ 7  0  0]
 [ 0 11  0]]
```

2
```
>>> a.nonzero   # ou numpy.nozero(a)
(array([0, 1, 1, 2, 3]), array([0, 1, 2, 0, 1]))
```

3
```
>>> b = a > 5
>>> print b
[[False False False]
 [False False  True]
 [ True False False]
 [False  True False]]
```

Seleção de submatriz

- Nome_da_Matriz[início:fim:passo]

```
1  >>> a = numpy.array([0,1,2,3,4])

2  >>> a[1:3]   # da posição 1 até a posição 3 (exclusive)
   array([1, 2])

3  >>> a[0:4:2] # elementos pulando de 2 em 2
   array([0, 2])

4  >>> a[-1]    # último termo
   4

5  >>> a[3:]    # ou a[3::]; posição até o último termo
   array([3, 4])

6  >>> a[::-1]  # inverte vetor
   array([4, 3, 2, 1, 0])
```

Varredura implícita

- Grade de índices

```
>>> x,y = numpy.indices((2, 5))
>>> print x
[[0 0 0 0 0]
 [1 1 1 1 1]]
>>> print y
[[0 1 2 3 4]
 [0 1 2 3 4]]
>>> print x + y
[[0 1 2 3 4]
 [1 2 3 4 5]]
```

- Chamada de função

```
>>> def funcao(x,y):
        return x+y
>>> z = numpy.fromfunction(funcao, (2,5))
>>> print z.astype('int')
[[0 1 2 3 4]
 [1 2 3 4 5]]
```

Exercícios

- Matriz em forma de moldura

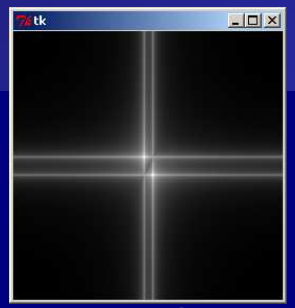
```
>>> import numpy
>>> a = numpy.ones((11, 11))
>>> b = numpy.zeros((5, 5))
>>> a[3:8,3:8] = b
```

- Matriz em forma de xadrez

```
from numpy import array, resize, concatenate
def xadrez(height, width):
    p1, p2 = array([0,1]), array([1,0])
    y1, y2 = resize(p1, (1, width)), resize(p2, (1, width))
    y = concatenate((y1, y2))
    y = resize(y, (height, width))
    return y
```

- Sintaxe Python/numpy

img



IMG

```
from numpy import *
# Criacao de uma senoide 2D
h, w = 256, 256
Th, Tw = 30, 60
thetah, thetaw = 0, 0
img = zeros((h,w), 'float')
for i in range(h):
    for j in range(w):
        img[i,j] = sin(2*pi*(i+1)/Th+thetah + 2*pi*(j+1)/Tw+thetaw)
# Aplicacao da FFT na imagem
IMG = fft.fft2(img)
```

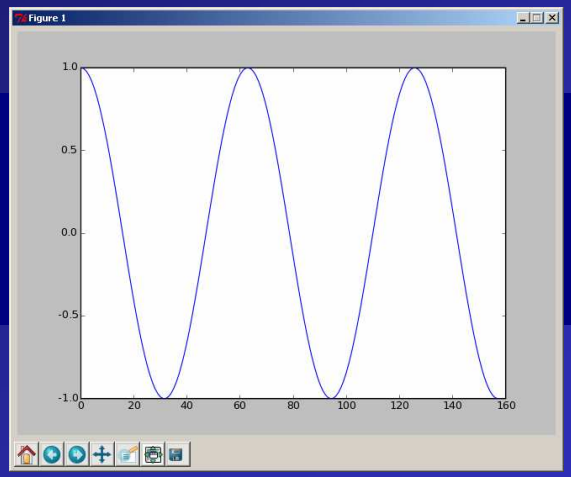
- Sintaxe MATLAB

```
% Criacao de uma senoide 2D
h = 256; w = 256;
Th = 30; Tw = 60;
thetah = 0; thetaw = 0;
img = zeros(h,w);
for i = 1:h
    for j = 1:w
        img(i,j) = sin(2*pi*i/Th+thetah + 2*pi*j/Tw+thetaw);
    end
end
% Aplicacao da FFT na imagem
IMG = fft2(img);
```

Outros módulos (1)

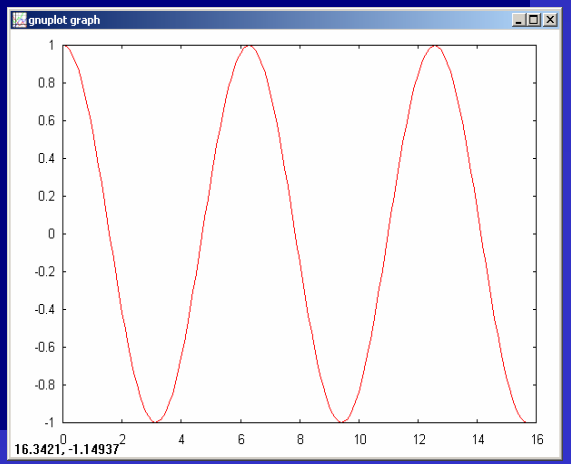
- pylab

```
from pylab import *
>>> x = arange(0,5*pi,0.1)
>>> y = cos(x)
>>> plot(y)
```



- Gnuplot

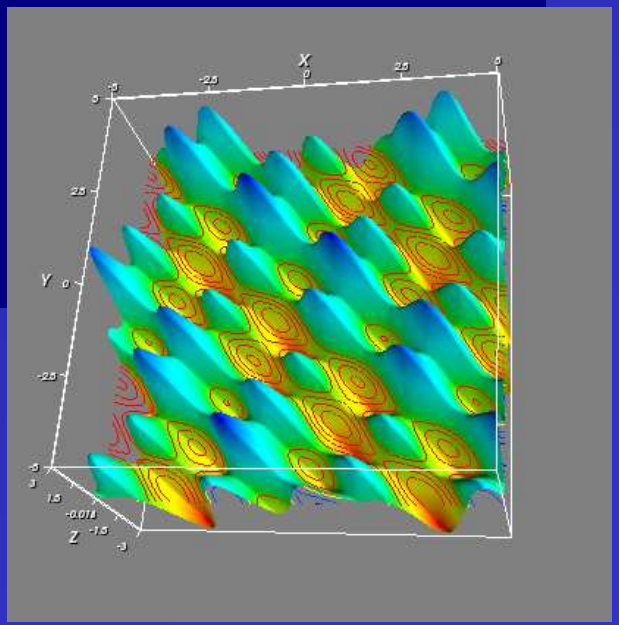
```
>>> from numpy import *
>>> import Gnuplot
>>> x = arange(0,5*pi,0.1)
>>> y = cos(x)
>>> g = Gnuplot.Gnuplot()
>>> d = Gnuplot.Data(x, y)
>>> g('set data style lines')
>>> g.plot(d)
```



Outros módulos (2)

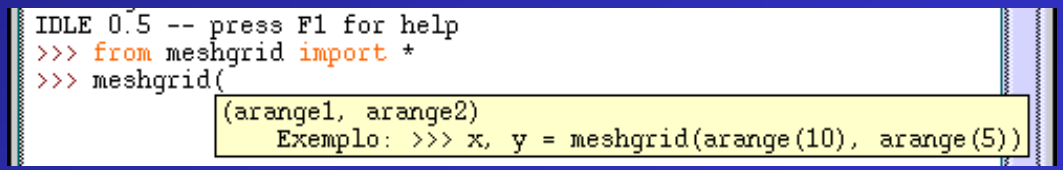
- ipython + TVTK

```
>>> from enthought.tvtk.tools import mlab
>>> from scipy import *

>>> def f(x,y):
  return sin(x+y) + sin(2*x-y) + cos(3*x+4*y)

>>> x = linspace(-5.0, 5.0, 200)
>>> y = linspace(-5.0, 5.0, 200)

>>> fig = mlab.figure()
>>> surf = mlab.SurfRegularC(x,y,f)
>>> fig.add(surf)
```



Considerações Finais (1)

- Uso
  - Ambiente de desenvolvimento rápido
  - Administração de sistemas (XML, regexp, sockets, …)
  - Interface gráfica (Tk, wxwindows, gtk, …)
  - *Scripts* para internet (CGI, HTTP, FTP, applets, …)
  - Programação de banco de dados (Oracle, Informix, Sybase)
  - Programação numérica (numpy)
  - Processamento de imagens (ia636, pymorph, PIL)
  - Jogos (pygame, PyOpenGL)
  - Inteligência artificial, CORBA, …
- Documentação
  - Recente, constante evolução
  - Ajuda dos objetos sem exemplos
  - Facilidades da linguagem e de seu *IDLE*



Considerações Finais (2)

- Soluções para desempenho crítico
  - Implementar em C/C++ e criar um *wrapper* em Python
  - SWIG
  - ADESSO
- Características
  - Não compilação ou ligação
  - Não declaração de tipos
  - Gerenciamento automático de memória
  - Tipos de dados e operações de alto nível
  - Programação orientada a objetos
  - *Extending* e *embedding* em C
  - Classe, módulos e exceções
  - Carregamento dinâmico de módulo em C
  - Interativo, de natureza dinâmica
  - Acesso a informações do interpretador
  - Grande portabilidade, compilação para byte-code

Referências

- Livros

Flávio Coelho. *Computação Científica com Python*. 1ª Edição do autor, 2007.

Mark Pilgrim. *Mergulhando no Python*. Alta Books, 1ª Edição, 2005.

- Documentação eletrônica

http://www.python.org/doc

http://www.pythonbrasil.com.br

- Pacotes

http://www.numpy.org

http://www.scipy.org

- Alguns trabalhos

http://www.dca.fee.unicamp.br/ia636

http://www.mmorph.com/pymorph

http://ortoprog.sourceforge.net