Linguagens de Programação para ***Machine Learning***

Prof. Dr. Diego Bruno

Education Tech Lead na DIO
Doutor em Robótica e *Machine Learning* pelo ICMC-USP

Vamos começar a programar...

Prof. Dr. Diego Bruno

Machine Learning

Linguagens de Programação

Vamos trabalhar inicialmente com as linguagens:

Python

R

Scilab

Paradigmas de Programação

```
from tensorflow.keras.models import Model
from tensorflow.keras.layers import Dense


class MyModel(Model):

    def __init__(self, hidden_units, outputs, **kwargs):
        super(MyModel, self).__init__(**kwargs)
        self.dense = Dense(hidden_units, activation='sigmoid')
        self.linear = LinearMap(hidden_units, outputs)

    def call(self, inputs):
        h = self.dense(inputs)
        return self.linear(h)


my_model = MyModel(64, 12, name='my_custom_model')
```

O que são paradigmas de programação?

Um paradigma de programação determina a visão que o programador possui sobre a estruturação e a execução do programa.

Por exemplo, em programação orientada a objetos, os programadores podem abstrair um programa como uma coleção de objetos que interagem entre si.

Quais os paradigmas?

Os paradigmas destas linguagens são importantes para entendermos melhor nossa forma de pensar sobre nossos problemas de computação:

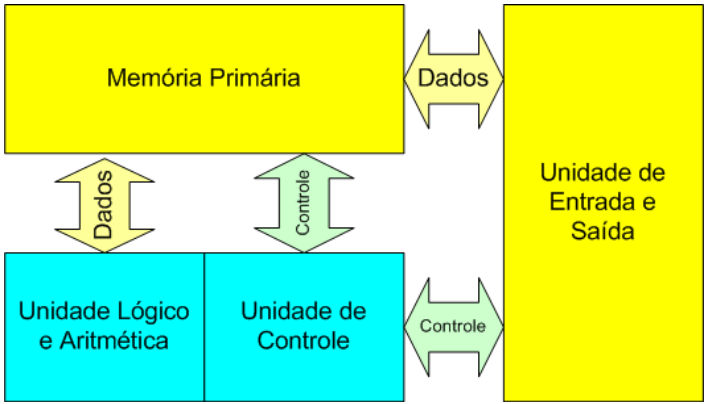
Lógica;
Funcional;
Imperativa;
Orientada a Objetos.

Paradigma de Programação Imperativa

```
from tensorflow.keras.models import Model
from tensorflow.keras.layers import Dense


class MyModel(Model):

    def __init__(self, hidden_units, outputs, **kwargs):
        super(MyModel, self).__init__(**kwargs)
        self.dense = Dense(hidden_units, activation='sigmoid')
        self.linear = LinearMap(hidden_units, outputs)

    def call(self, inputs):
        h = self.dense(inputs)
        return self.linear(h)


my_model = MyModel(64, 12, name='my_custom_model')
```

Programação Imperativa

O paradigma de programação que descreve a computação como ações, enunciados ou comandos que mudam o estado (variáveis) de um programa.



Arquitetura de Von Neumann

Este paradigma foi projetado para a arquitetura de computadores prevalecente

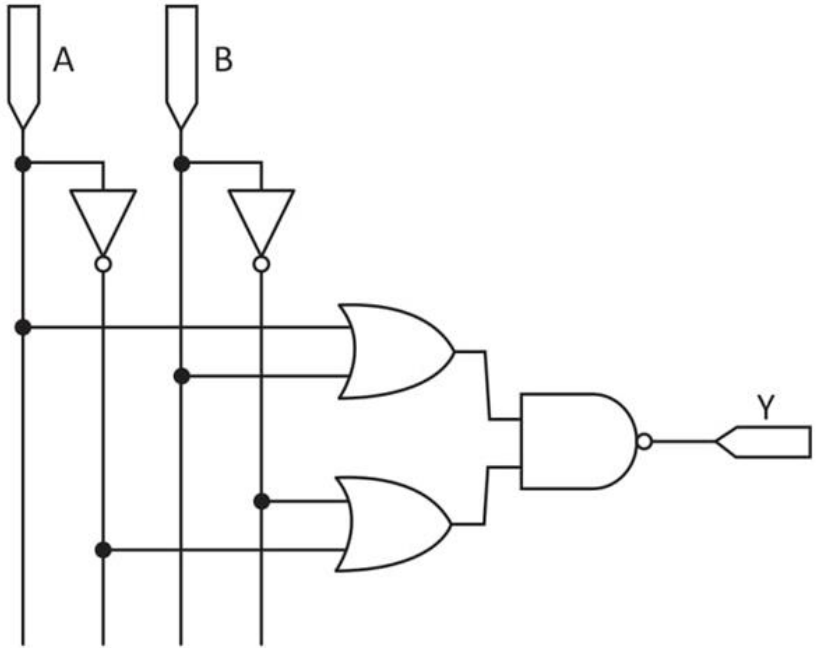
Assembly Language

Paradigma de Programação Lógica

```
from tensorflow.keras.models import Model
from tensorflow.keras.layers import Dense


class MyModel(Model):

    def __init__(self, hidden_units, outputs, **kwargs):
        super(MyModel, self).__init__(**kwargs)
        self.dense = Dense(hidden_units, activation='sigmoid')
        self.linear = LinearMap(hidden_units, outputs)

    def call(self, inputs):
        h = self.dense(inputs)
        return self.linear(h)


my_model = MyModel(64, 12, name='my_custom_model')
```

Programação Lógica

O sentido da **programação lógica** é trazer o estilo da lógica matemática à programação de computadores.



Programação Lógica

Considere o seguinte banco de dados:

gosta(maria, flores).
gosta(maria, pedro).
gosta(paulo, maria).

Se fizermos a pergunta:

?- gosta(maria, X).

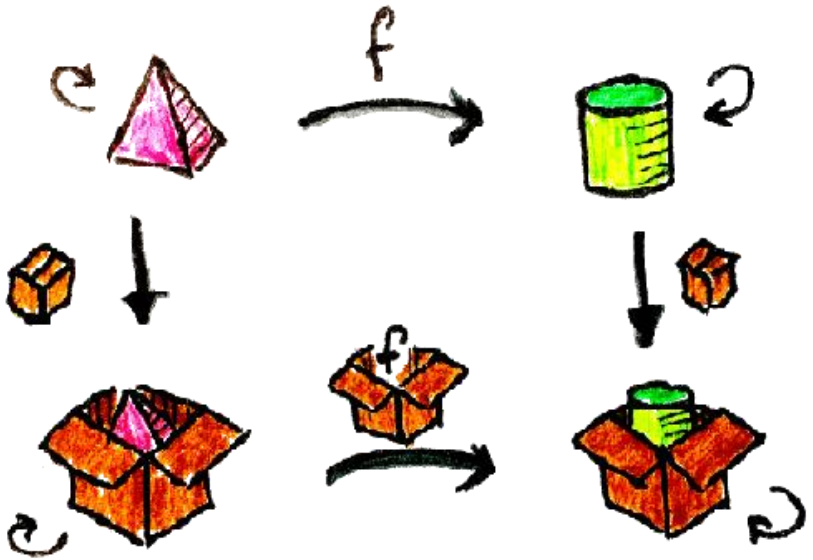
estaremos perguntando “Do que Maria gosta?”.
Prolog responde: **X = flores**

Paradigma de Programação Funcional

```
from tensorflow.keras.models import Model
from tensorflow.keras.layers import Dense


class MyModel(Model):

    def __init__(self, hidden_units, outputs, **kwargs):
        super(MyModel, self).__init__(**kwargs)
        self.dense = Dense(hidden_units, activation='sigmoid')
        self.linear = LinearMap(hidden_units, outputs)

    def call(self, inputs):
        h = self.dense(inputs)
        return self.linear(h)


my_model = MyModel(64, 12, name='my_custom_model')
```

Programação Funcional

Programação funcional é um paradigma de programação que trata a computação como uma avaliação de funções matemáticas.



2+2x3=?

Programação Funcional

Programação funcional é um paradigma de programação que trata a computação como uma avaliação de funções matemáticas.

Scheme

```
1  ((lambda (x) (+ x x)) (* 3 4))
```

Nesse caso, seria isso que aconteceria:

3*4 = 12;

x = 12;

x + x = 12 + 12 = 24;.

Linguagem funcional

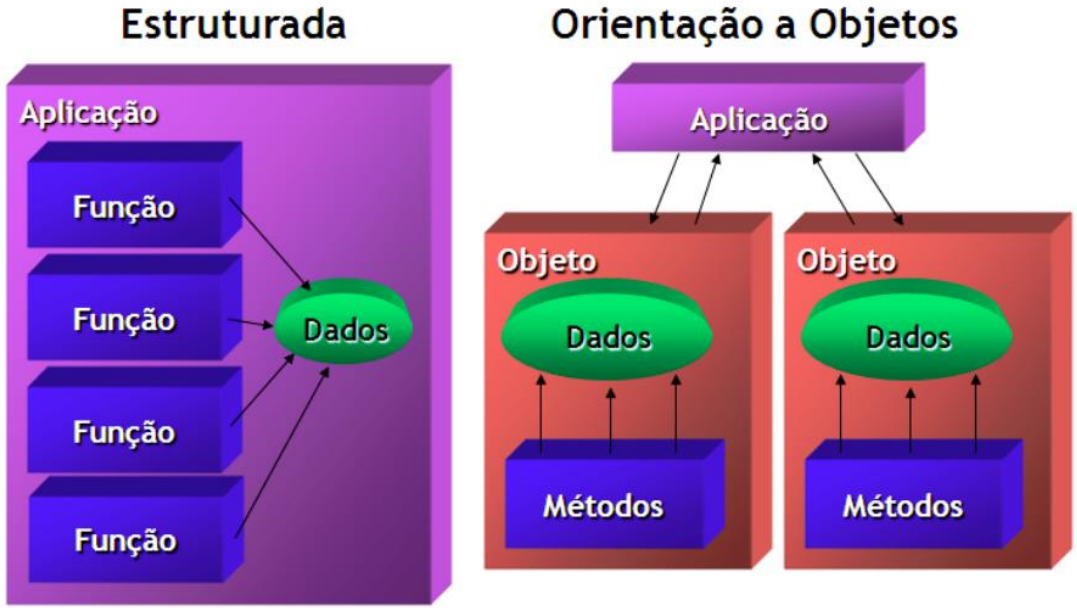
Suporte para funcional

Paradigma de Programação Orientada a Objetos

```
from tensorflow.keras.models import Model
from tensorflow.keras.layers import Dense


class MyModel(Model):

    def __init__(self, hidden_units, outputs, **kwargs):
        super(MyModel, self).__init__(**kwargs)
        self.dense = Dense(hidden_units, activation='sigmoid')
        self.linear = LinearMap(hidden_units, outputs)

    def call(self, inputs):
        h = self.dense(inputs)
        return self.linear(h)


my_model = MyModel(64, 12, name='my_custom_model')
```

Programação Orientada a Objetos

Na programação Orientada a Objetos temos como objetivo transformar nosso problema do mundo real em partes para o computador.



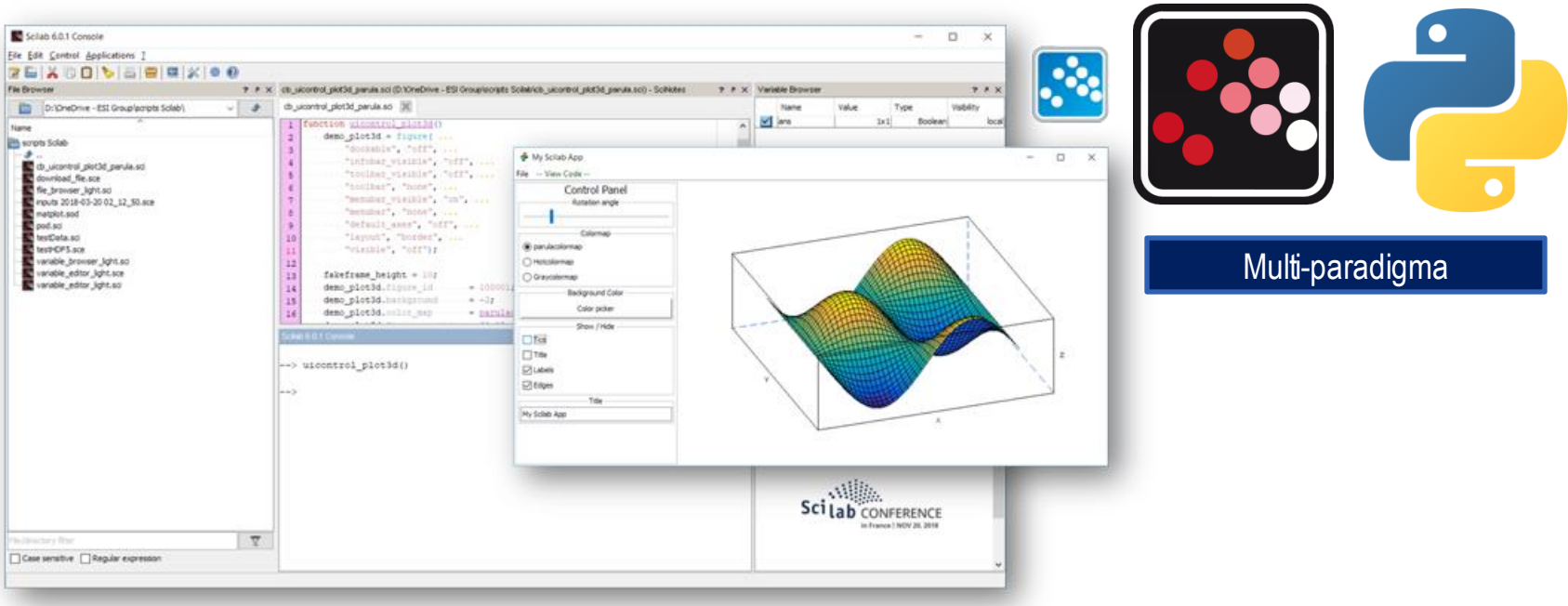
- Poliformismo
- Herança
- Encapsulamento
- Abstração

Paradigma de Programação Multi-paradigma

```
from tensorflow.keras.models import Model
from tensorflow.keras.layers import Dense


class MyModel(Model):

    def __init__(self, hidden_units, outputs, **kwargs):
        super(MyModel, self).__init__(**kwargs)
        self.dense = Dense(hidden_units, activation='sigmoid')
        self.linear = LinearMap(hidden_units, outputs)

    def call(self, inputs):
        h = self.dense(inputs)
        return self.linear(h)


my_model = MyModel(64, 12, name='my_custom_model')
```

Programação Orientada a Objetos

Scilab (laboratório de matriz) é um ambiente de computação numérica multi-paradigma.



Obrigado!

Prof. Dr. Diego Bruno

Machine Learning

```
from tensorflow.keras.models import Model
from tensorflow.keras.layers import Dense


class MyModel(Model):

    def __init__(self, hidden_units, outputs, **kwargs):
        super(MyModel, self).__init__(**kwargs)
        self.dense = Dense(hidden_units, activation='sigmoid')
        self.linear = LinearMap(hidden_units, outputs)

    def call(self, inputs):
        h = self.dense(inputs)
        return self.linear(h)


my_model = MyModel(64, 12, name='my_custom_model')
```

Python para ***Machine Learning***

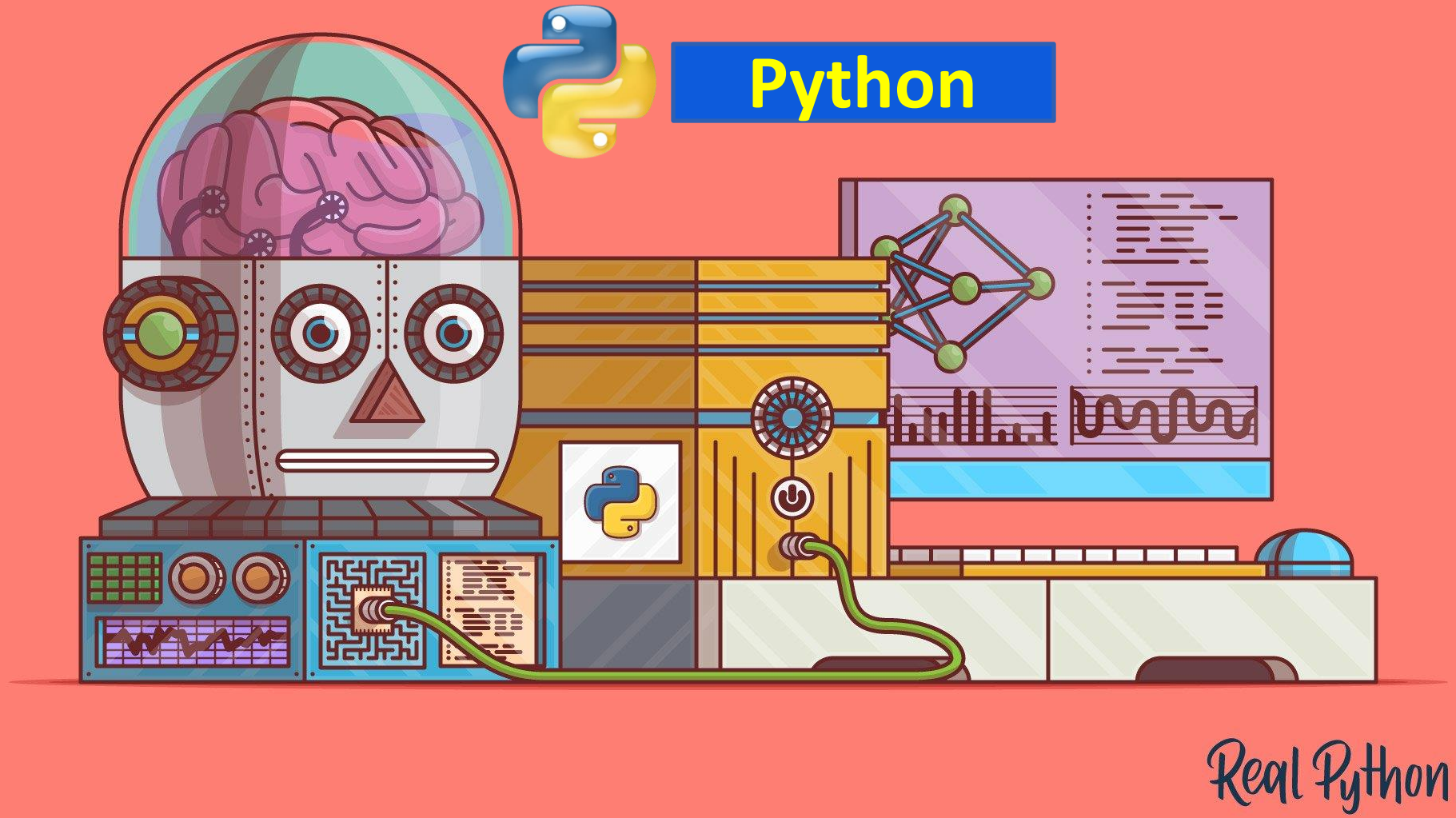
Prof. Dr. Diego Bruno

Education Tech Lead na DIO
Doutor em Robótica e *Machine Learning* pelo ICMC-USP

Vamos começar a programar...

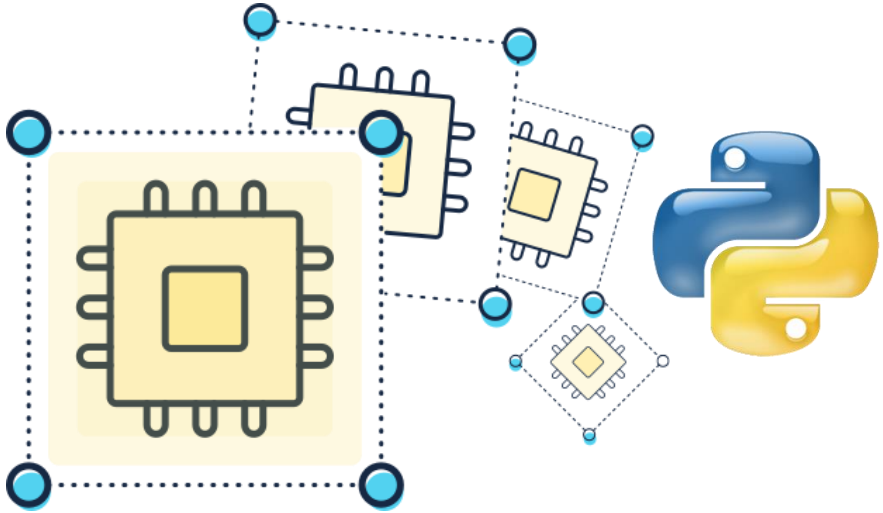
Prof. Dr. Diego Bruno

Machine Learning


Python
Real Python

O básico para Python em ML

1. Programação básica;
2. Instalação de dependências;
3. Bibliotecas de ML;
4. Ambientes de colaboração de código;
5. Configuração de GPU em CUDA.



Programação Básica com Python

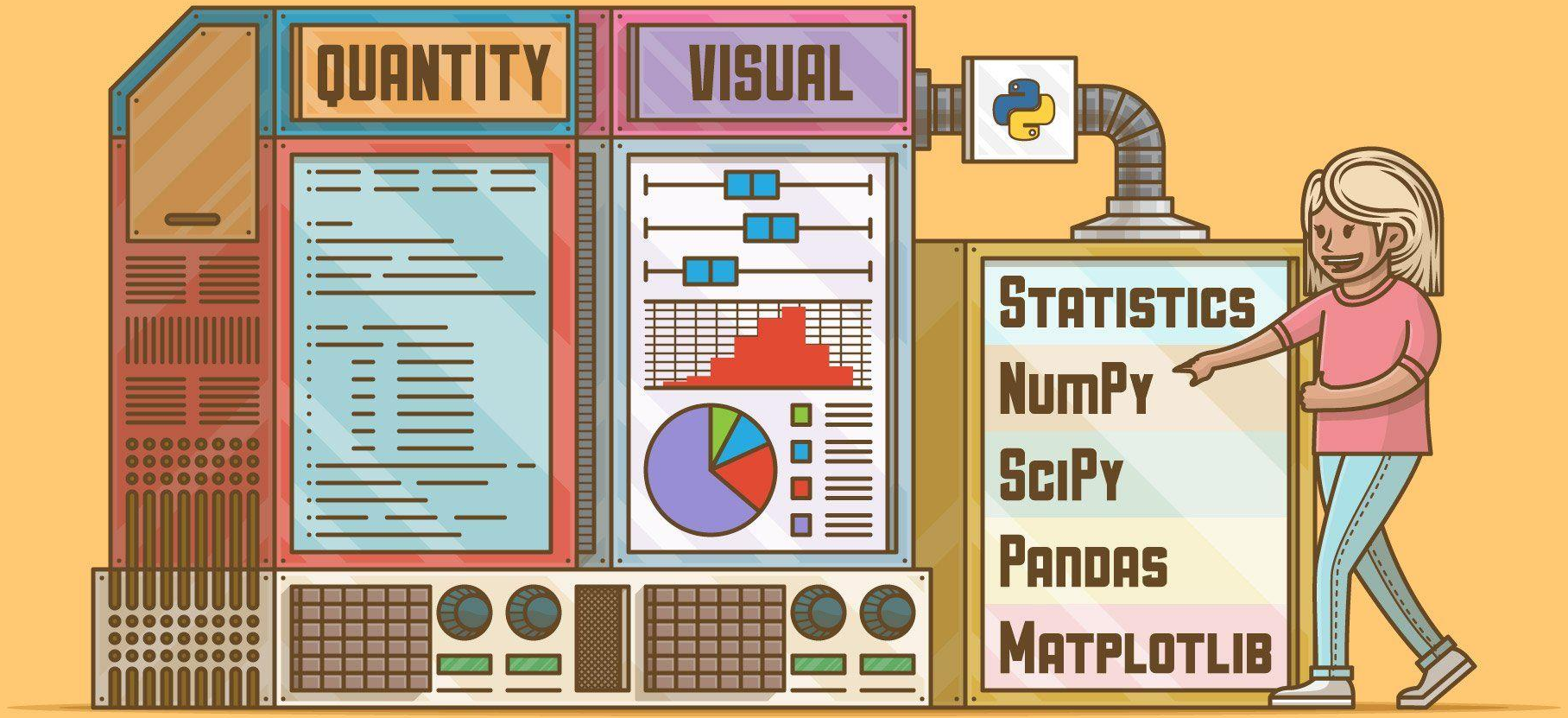
Real Python

Nosso primeiro programa...

```
>>> print("Hello World!")
Hello World!
>>>
```

Vamos treinar um pouco mais...

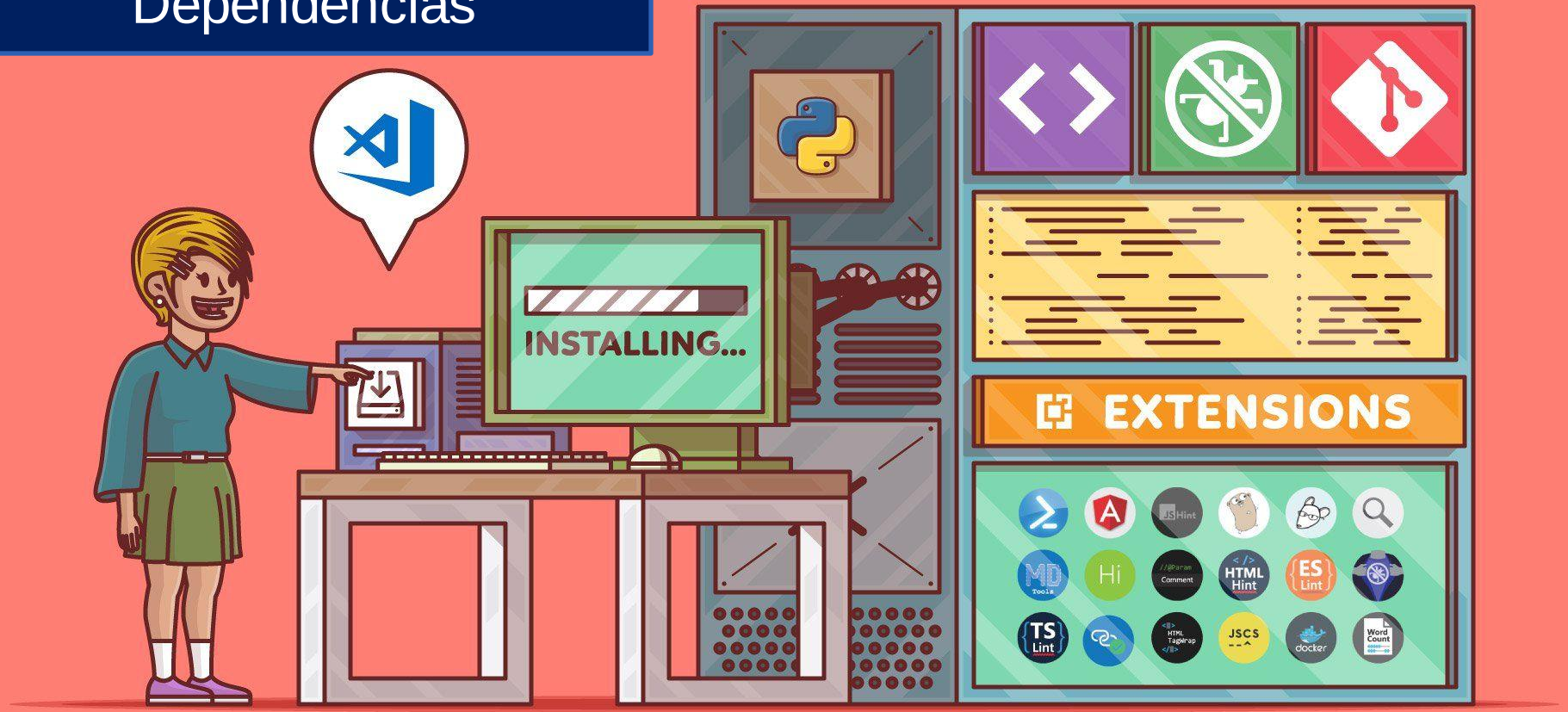
Real Python



Mas nosso objetivo é muito mais forte...

Real Python

Dependências

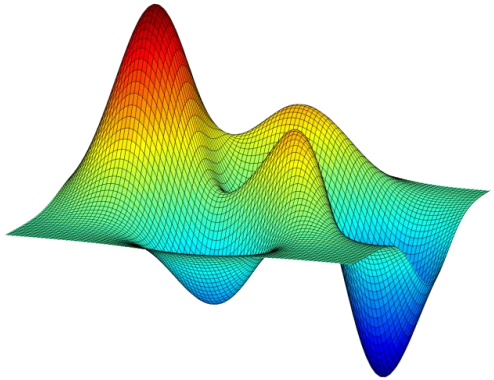


Obrigado!

Machine Learning

Prof. Dr. Diego Bruno

Scilab para ***Machine Learning***



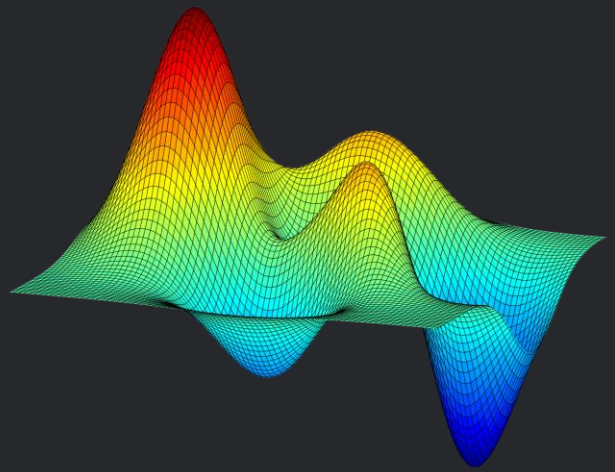
Prof. Dr. Diego Bruno

Education Tech Lead na DIO
Doutor em Robótica e *Machine Learning* pelo ICMC-USP

Vamos começar a programar...

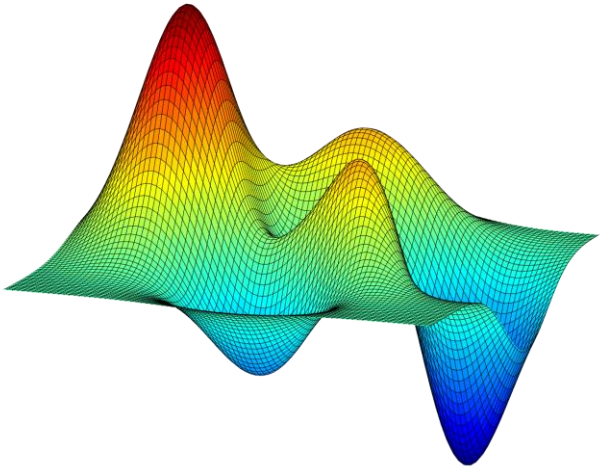
Prof. Dr. Diego Bruno

Machine Learning

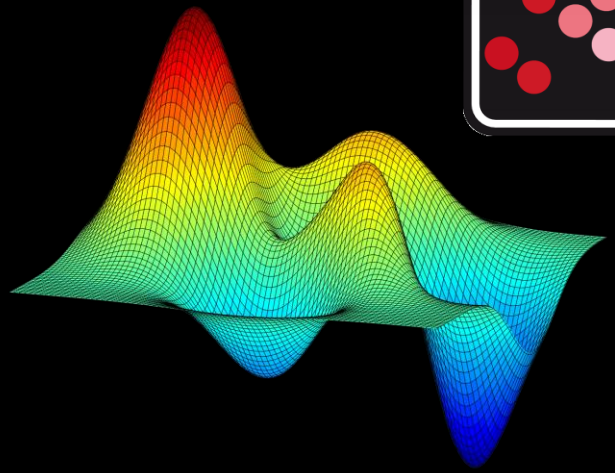


Aplicações do Scilab

O **Scilab** é um software científico para computação numérica semelhante ao MATLAB que fornece um poderoso ambiente computacional aberto para aplicações científicas e de engenharia. Disponível gratuitamente para várias plataformas: Windows, Linux e Mac OS X.

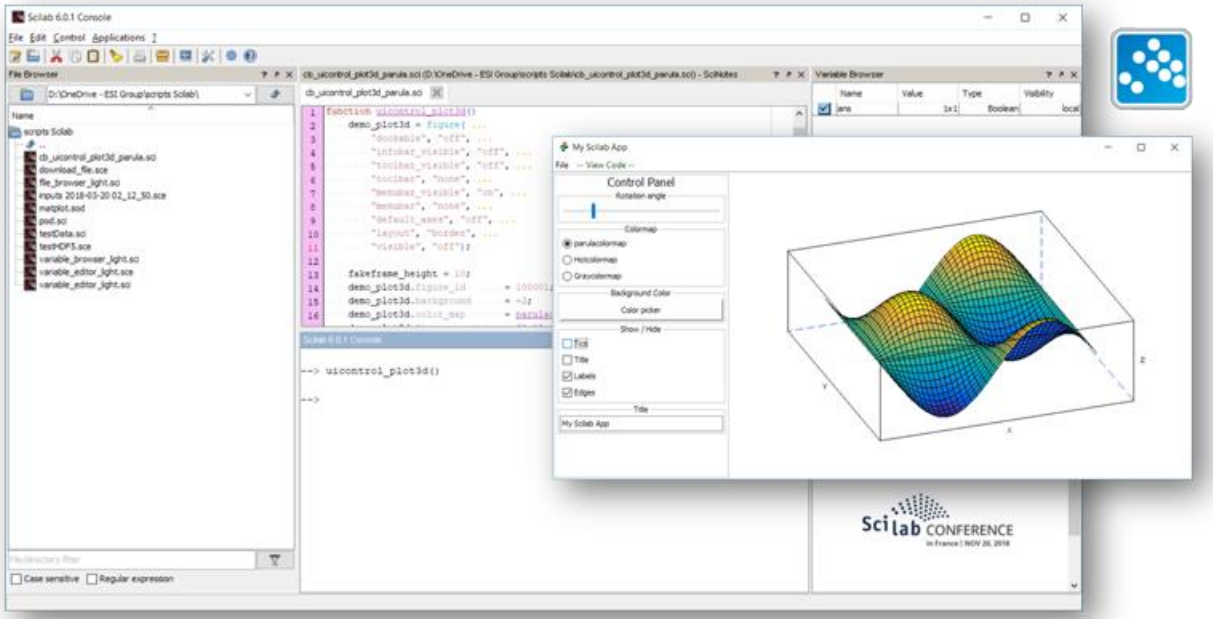


Matemática Básica para Scilab



Programação Orientada a Objetos

Scilab (laboratório de matriz) é um ambiente de computação numérica multi-paradigma.



Multi-paradigma

Funções básicas no Scilab

O sinal de prontidão “-->” indica que o Scilab aguarda a digitação de um comando ou expressão, que deve ser finalizado pela tecla ENTER. Exemplos:

```
-->5+10
 ans  =
    15.
-->10/4
 ans  =
    2.5
```

Operações Matemáticas

As operações básicas da matemática podem ser realizadas no Scilab por meio dos seguintes operadores:

+	Soma
-	Subtração
*	Multiplicação
/	Divisão
^	Potenciação (x^y)

Exemplo:

```
-->2*6
 ans  =
    12.
```

Precedência de Operadores

Quando uma expressão envolve diversos operadores, o Scilab considera a ordem de precedência dos mesmos para avaliar a expressão:

Prioridade	Operação
1ª	Potenciação: ^
2ª	Multiplicação e divisão: *, /
3ª	Soma e subtração: +, –

Exemplos:

`-->4*3^2`
`ans =`
`36.`
	Como o operador de potenciação tem maior prioridade em relação ao de multiplicação, a operação 3^2 é avaliada primeiro e o seu resultado (9) é multiplicado por 4.

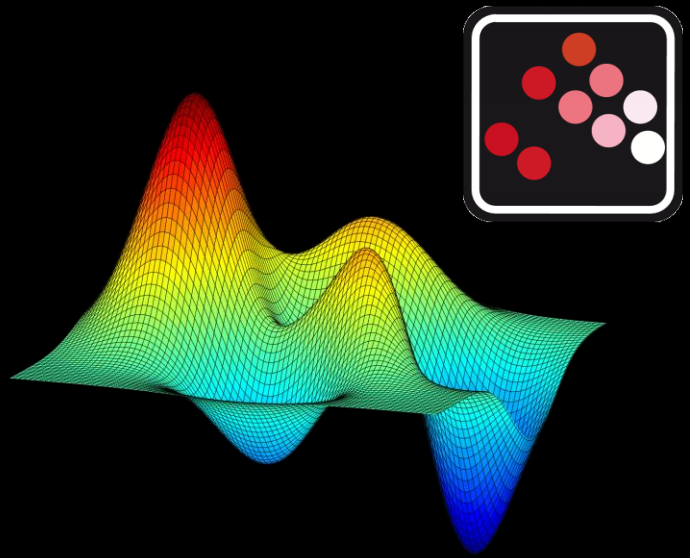
`-->10+4/2`
`ans =`
`12.`
	4/2 é avaliado primeiro, pois o operador de divisão tem prioridade sobre o de adição. O resultado é então somado com 10.

Expressões com frações

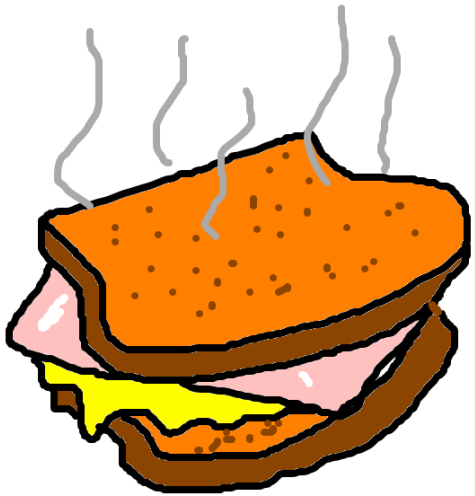
Repare que, se os parênteses não fossem utilizados, a expressão 10+4/2 no Scilab seria equivalente à expressão matemática, $10 + \frac{4}{2}$,

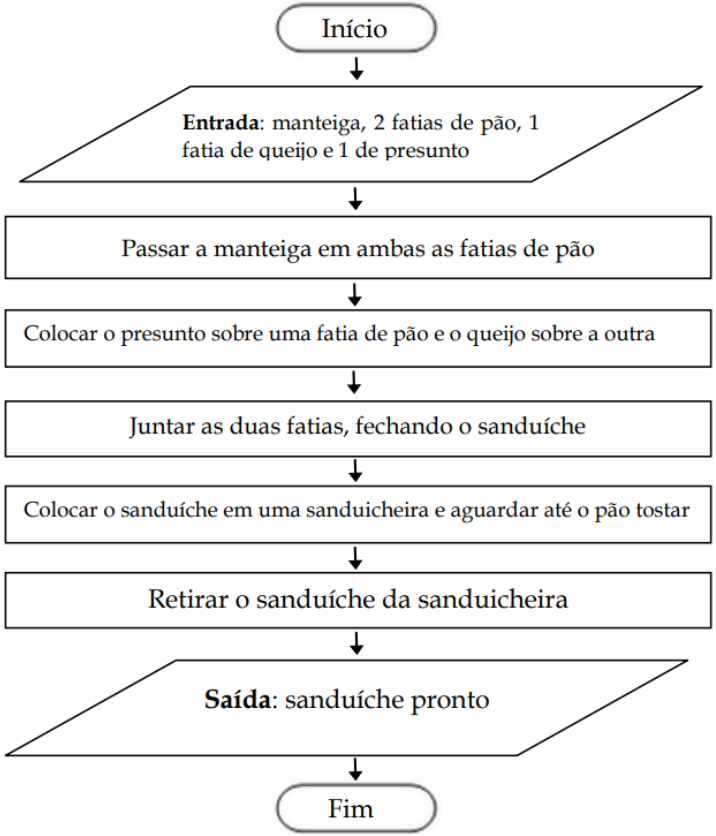
Expressão Matemática	Expressão correspondente no Scilab
$\frac{1}{2}+\frac{3}{5}-\frac{5}{8}$	`-->1/2 + 3/5 - 5/8`
$\frac{3+2^5}{5}$	`-->(3+2^5)/5`
$\frac{5\,(4+2)\,-\,1}{10+3^2}$	`-->(5*(4+2)-1)/(10+3^2)`
$\frac{2^4+2^6}{2^5-1}+20$	`-->(2^4+2^6)/(2^5-1)+20`

Algoritmos

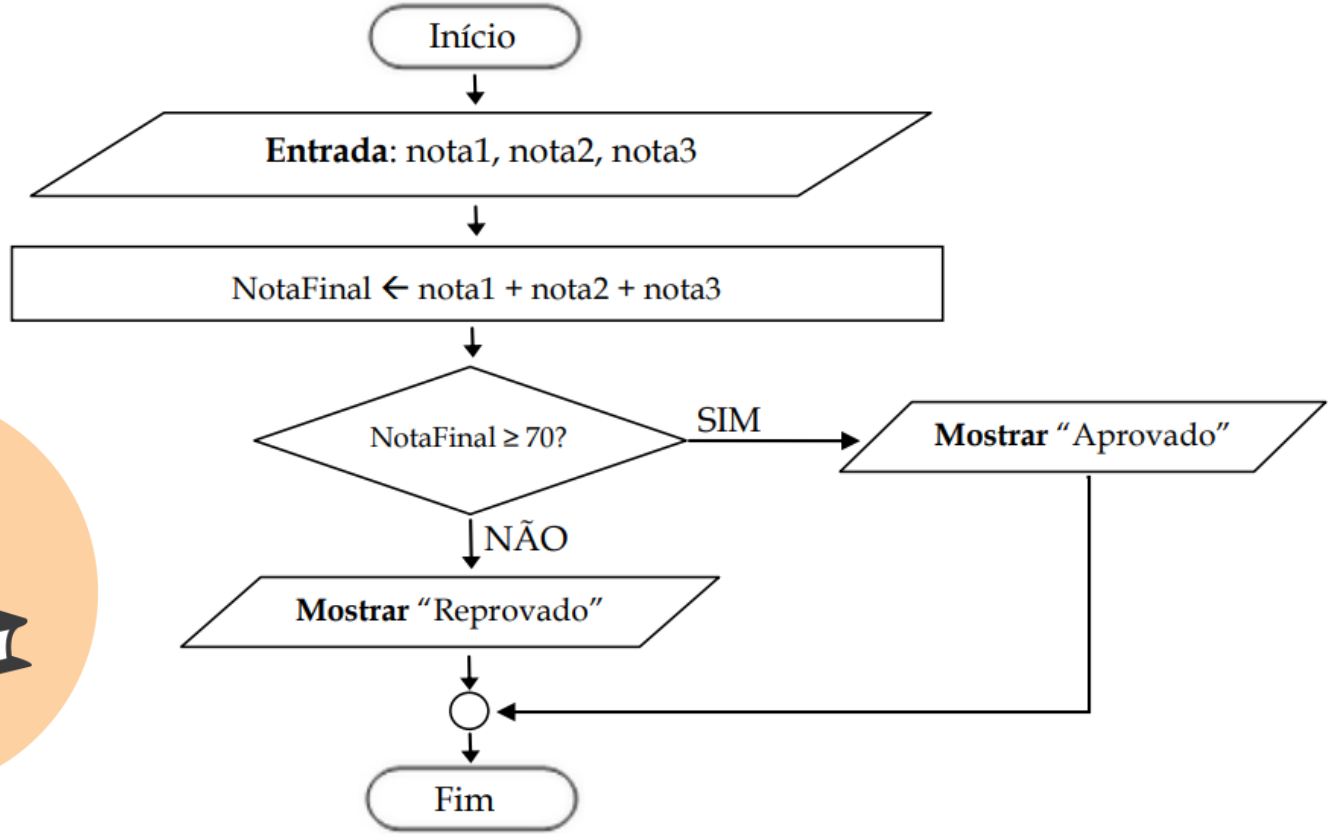


Introdução para Algoritmos

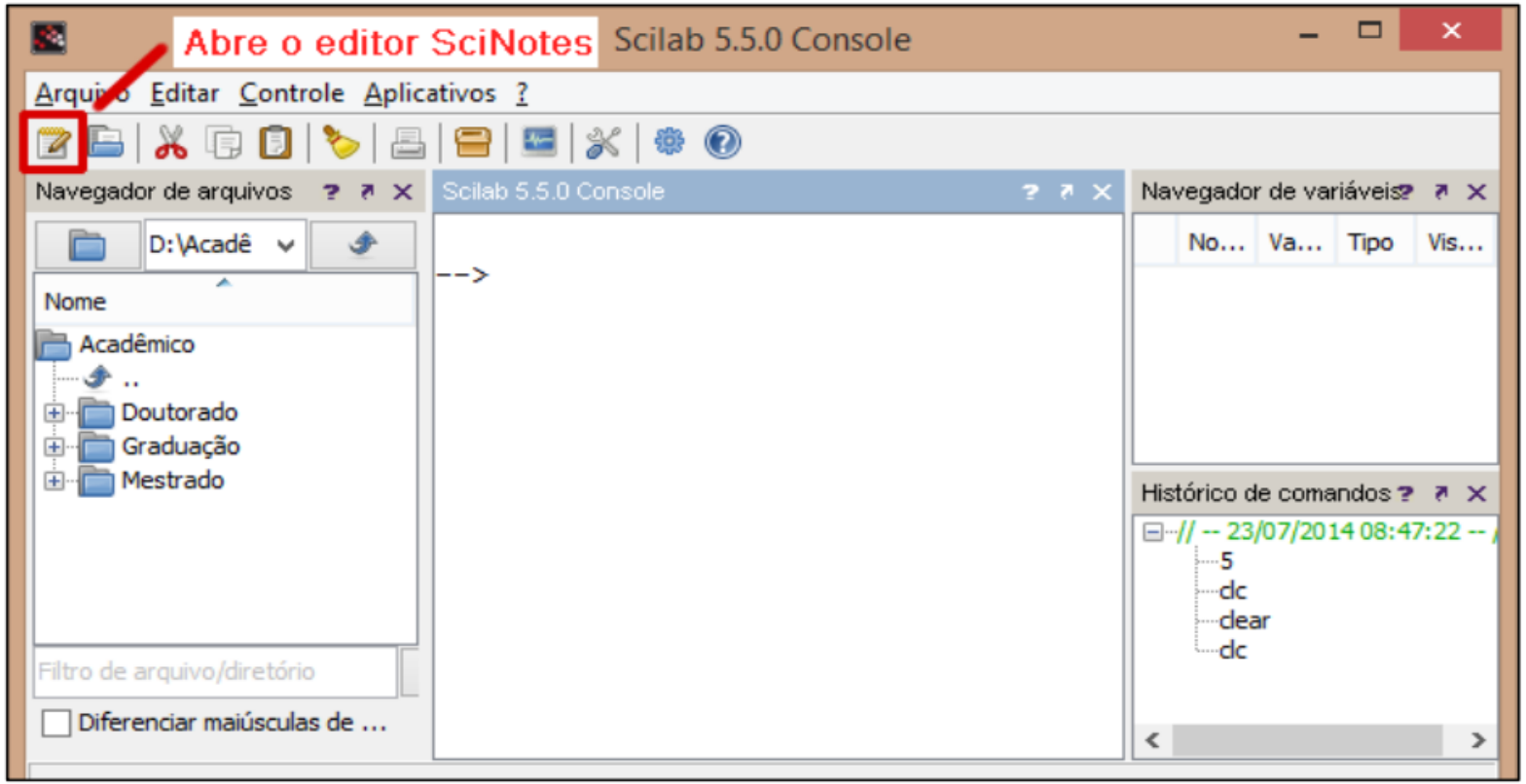




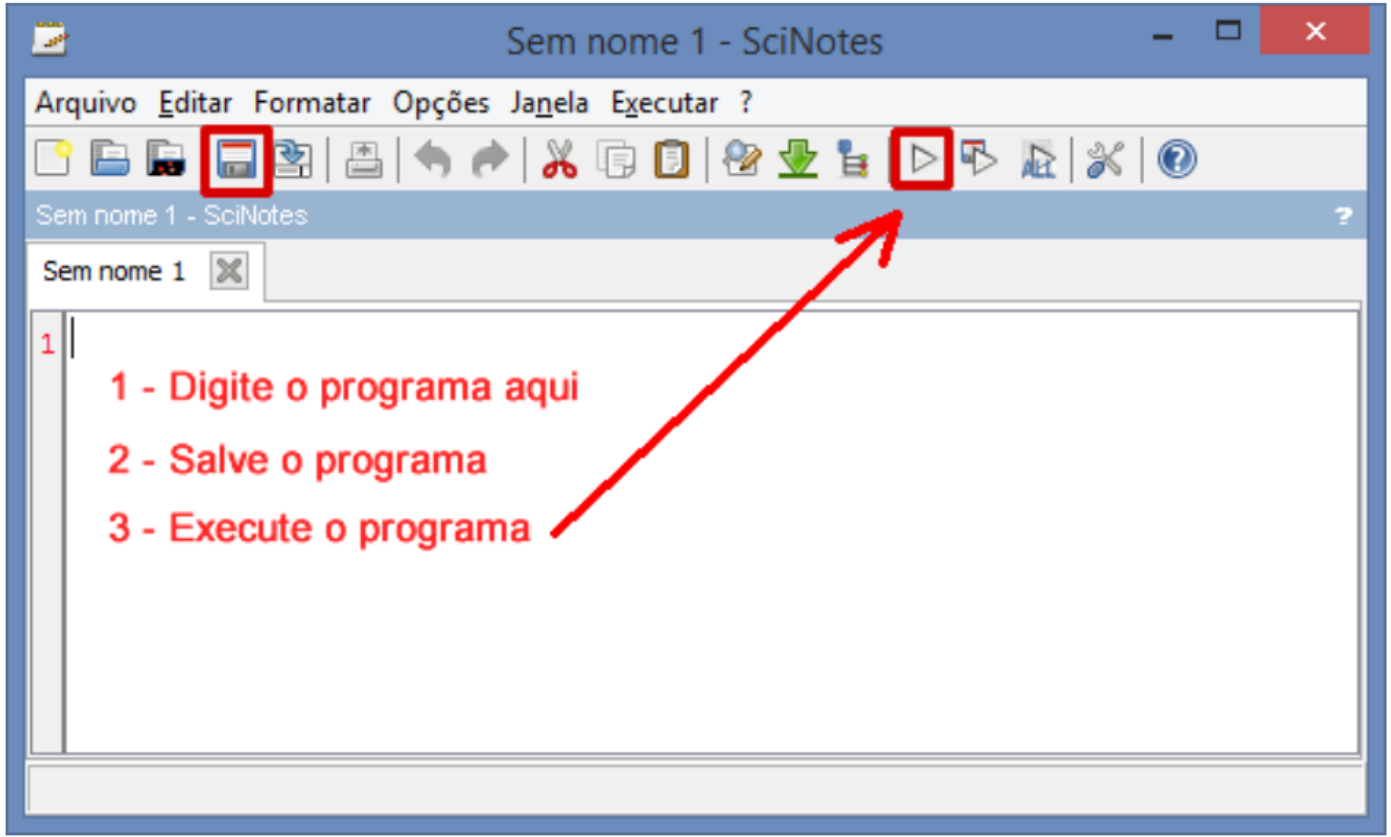
Introdução para Algoritmos



Scinotes


Abre o editor SciNotes
Scilab 5.5.0 Console
Arquivo Editar Controle Aplicativos ?
Navegador de arquivos
D:\Acadê
Nome
Acadêmico
..
Doutorado
Graduação
Mestrado
Filtro de arquivo/diretório
Diferenciar maiúsculas de ...
Scilab 5.5.0 Console
-->
Navegador de variáveis
No...
Va...
Tipo
Vis...
Histórico de comandos
// -- 23/07/2014 08:47:22 --
5
clc
clear
clc

Scinotes


Sem nome 1 - SciNotes
Arquivo Editar Formatar Opções Janela Executar ?
Sem nome 1 - SciNotes
Sem nome 1
1
1 - Digite o programa aqui
2 - Salve o programa
3 - Execute o programa

Funções *input* e *disp*

Entrada e saída de dados:

Exemplo para calcular a área de um quadrado.

```
lado = input("Informe a medida do lado de um quadrado: ");
area = lado * lado;
disp("A área do quadrado é: ");
disp(area);
```

Função Printf

A diferença é que, com a função ***printf***, podemos apresentar a mensagem de texto juntamente com o valor da variável.

```
lado = input("Informe a medida do lado de um quadrado: ");
area = lado * lado;
printf("A área do quadrado é: %f", area);
```

Estrutura Condicional **if-then-else**

A estrutura condicional **if-then-else (if = se; then = então; else = senão)** é utilizada quando se deseja executar um bloco de operações ou outro, dependendo de uma condição ser verdadeira ou falsa, respectivamente.

```
if (cond) then
        Comando C1;
        Comando C2;
        ⋮
        Comando Cn;
else
        Comando D1;
        Comando D2;
        ⋮
        Comando Dm;
end
```

Estrutura Condicional **if-then-else**

Calcular a área de um quadrado. Entretanto, o programa deve calcular a área da sala apenas quando o valor informado pelo usuário for **maior do que zero**, pois não há sala com lado negativo ou nulo.

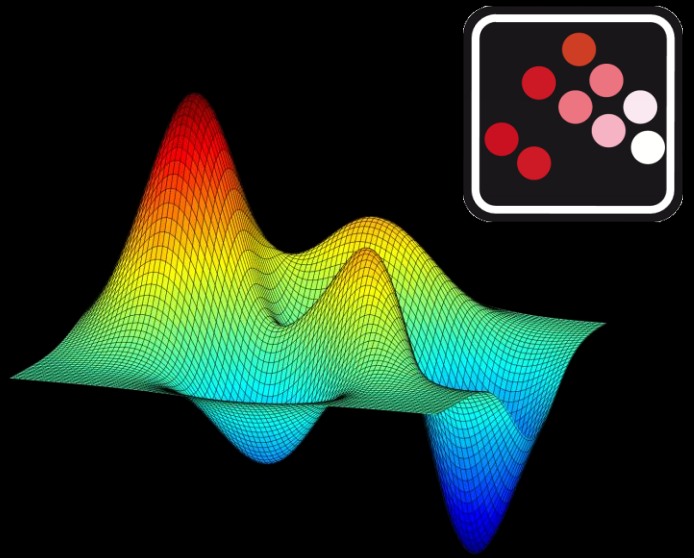
```
lado = input("Informe a medida do lado da sala: ");
if (lado > 0) then
        area = lado * lado;
        printf("A area da sala é %f", area);
else
        printf("O valor informado é inválido! ");
end
```

Estrutura Condicional **if-then-else**

Verificando se dois números são iguais ou não.

```
n1 = input("Informe o primeiro número: ");
n2 = input("Informe o segundo número: ");
if (n1 == n2) then
        printf("Os números fornecidos são iguais");
else
        printf("Os números fornecidos são diferentes");
end
```

Operadores Relacionais

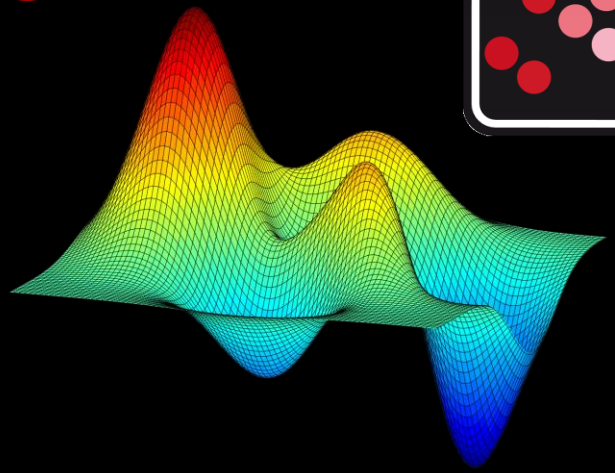


Operadores relacionais

O Scilab disponibiliza ao usuário um conjunto de operadores relacionais, que inclui os já utilizados == e >

Operador	Significado
==	Igual a
~=	Diferente de
>	Maior que
>=	Maior ou igual a
<	Menor que
<=	Menor ou igual a

Operadores Lógicos



Operadores lógicos

Os operadores lógicos do Scilab podem ser utilizados para a formação de expressões lógicas mais complexas (que envolvem, por exemplo, **duas ou mais condições**).

Operador Lógico	Significado	Exemplo	Significado
&	"e" lógico	**if** (a > 0 **&** a < 10)...	Verifica se o valor da variável **a** é maior que 0 e menor que 10.
|	"ou" lógico	**if** (a == 0 | b == 0) ...	Verifica se alguma das variáveis (**a** ou **b**) tem valor igual a zero.
~	Negação	**if** ~(a < 0) ...	Verifica se o valor da variável **a** NÃO é menor que zero. Isto é equivalente à condição **a >= 0**

Operadores lógicos

Os operadores lógicos do Scilab podem ser utilizados para a formação de expressões lógicas mais complexas (que envolvem, por exemplo, **duas ou mais condições**).

A	B	A & B	A | B	~A
V	V	V	V	F
V	F	F	V	F
F	V	F	V	V
F	F	F	F	V

Operadores lógicos

Exemplos:

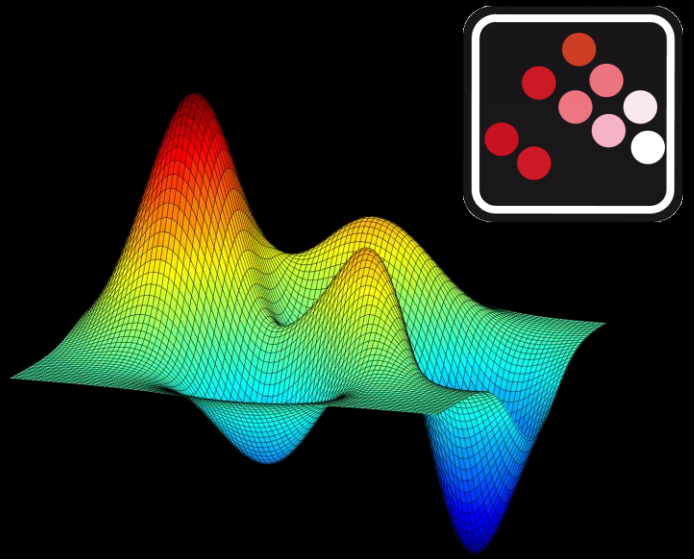
a)

```
a = 2; b = 3; c = 1;
if (a > b) then
   printf("%d",a);
else
   if (b > c) then
      printf("%d",b);
   else
      printf("%d",c);
   end
end
```

b)

```
a = 10; b = 20; c = 10;
if (a == b | a == c) then
   printf("%d",a);
else
   printf("%d",b);
end
```

Construção de Gráficos



Construção de gráficos

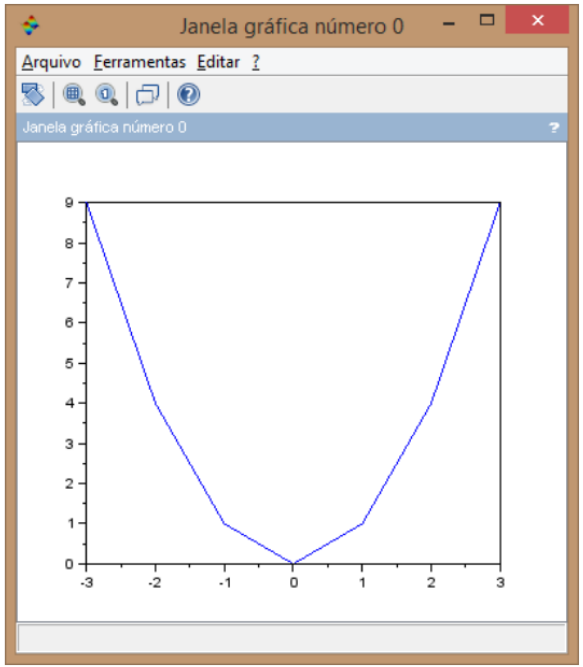
O Scilab disponibiliza uma série de recursos que possibilitam a construção de diversos tipos de gráficos.

Construção de um gráfico pela especificação de uma série de pares ordenados.

```
x = [-3, -2, -1, 0, 1, 2, 3];
y = [9, 4, 1, 0, 1, 4, 9];
plot(x, y);
```

Construção de gráficos

Construção de um gráfico pela especificação de uma série de pares ordenados.



	p_1	p_2	p_3	p_4	p_5	p_6	p_7
x	-3	-2	-1	0	1	2	3
$f(x)$	9	4	1	0	1	4	9

Construção de gráficos

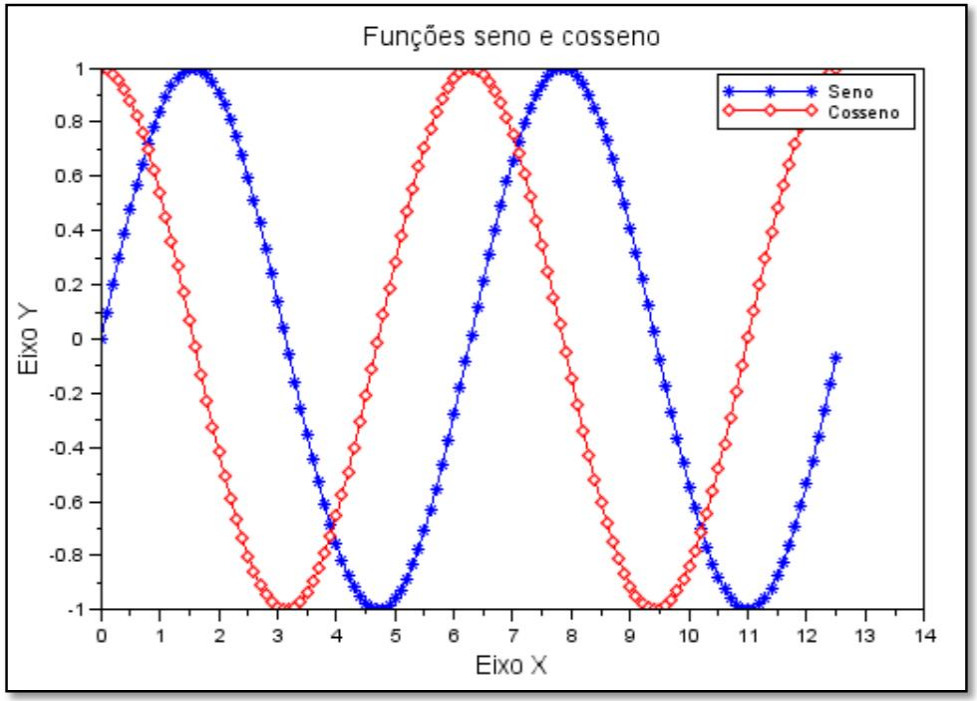
Representação de senos e cossenos:

```
clf
x = [0:0.1:4*%pi];
y1 = sin(x);
y2 = cos(x);
plot(x,y1,'-*b');
plot(x,y2,'-dr');
xtitle('Funções seno e cosseno');    // insere o título
xlabel('Eixo X');                    // insere o rótulo do eixo horizontal x
ylabel('Eixo Y');                    // insere o rótulo do eixo vertical y
legend('Seno', 'Cosseno');           // insere uma legenda para identificar,
                                     // na respectiva ordem, as duas curvas
```

Construção de gráficos

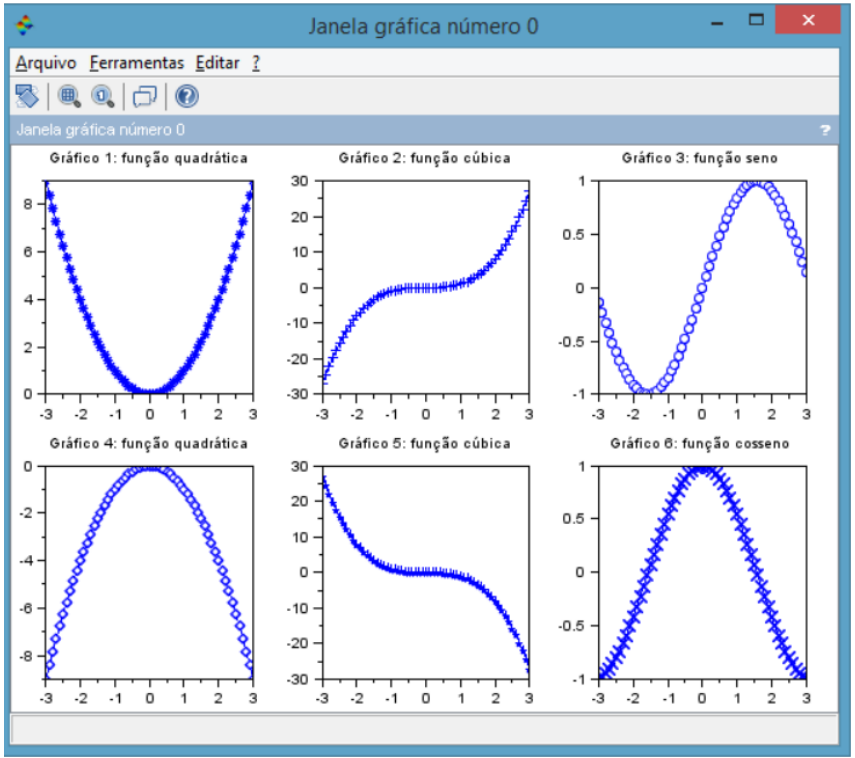
Representação de senos e cossenos:

Resultado:

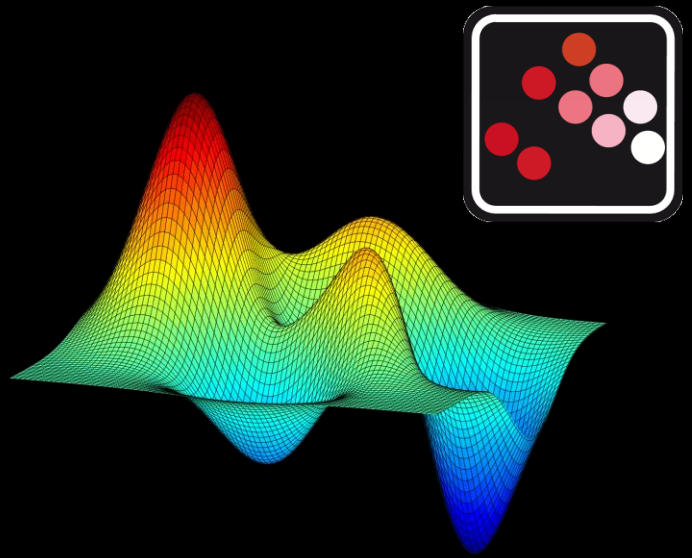


Inserindo diversos gráficos

Inserindo diversos gráficos na mesma janela com a função *subplot* :



Operações com Matrizes



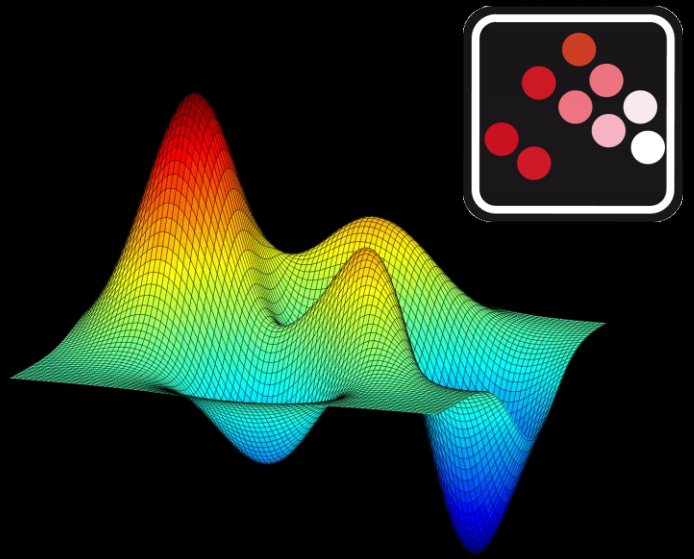
Operações com Matrizes

Matrizes podem ser definidas no Scilab de maneira semelhante aos vetores. Os elementos devem ser especificados entre colchetes

$$A = \begin{bmatrix} 1 & 2 & 3 \\ 2 & 5 & 1 \\ 7 & 4 & 9 \end{bmatrix} \qquad B = \begin{bmatrix} 2 & 1 \\ 3 & 2 \\ 4 & 3 \end{bmatrix} \qquad C = \begin{bmatrix} 9 & 8 & 7 & 6 \\ 1 & 2 & 3 & 4 \end{bmatrix}$$

```
-->A = [1,2,3; 2,5,1; 7,4,9]
 A  =
    1.    2.    3.
    2.    5.    1.
    7.    4.    9.
-->B = [2 1; 3 2; 4 3]
 B  =
    2.    1.
    3.    2.
    4.    3.

-->C = [9 8 7 6
-->1 2 3 4]
 C  =
    9.    8.    7.    6.
    1.    2.    3.    4.
```

Estruturas de Repetição



Estruturas de repetição

A estrutura de **repetição *while*** é utilizada quando se deseja repetir a execução de um bloco de comandos enquanto uma determinada condição for verdadeira.

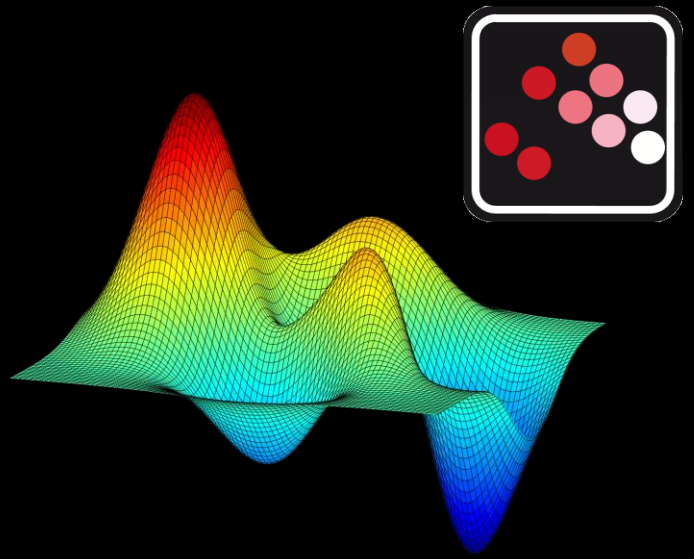
```
while (condição)
        comando 1
        comando 2
        ...
        comando n
end
```

Estruturas de repetição

Utilizando a estrutura ***while***, faça um programa para ler números inteiros do teclado até que o número 0 seja lido. Ao término da leitura, o programa deverá apresentar a soma de todos os números informados.

```
total = 0;
x = input('Digite o primeiro número: ');
while (x ~= 0)
    total = total + x;
    x = input('Digite o próximo número (ou 0 para encerrar): ');
end
printf('A soma dos números informados é: %d', total);
```

Processamento de Imagens



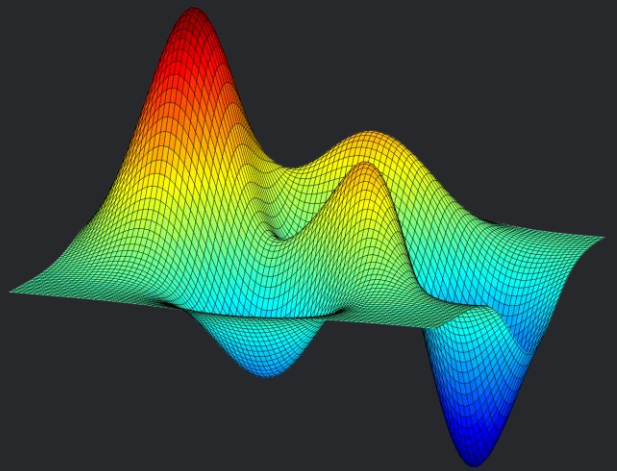
Processamento de imagens

```
RGB = imread('teaset.png');
imshow(RGB)f=gcf();
f.name='Color Image';
Image = rgb2gray(RGB);
//figure('name','Gray Level Image'); imshow(Image); f=gcf();f.name='Gray Level Image'; imshow(Image,jetcolormap(256))f=gcf();
f.name='Pseudo Color Image';
```

Obrigado!

Machine Learning

Prof. Dr. Diego Bruno



R para *Machine Learning*

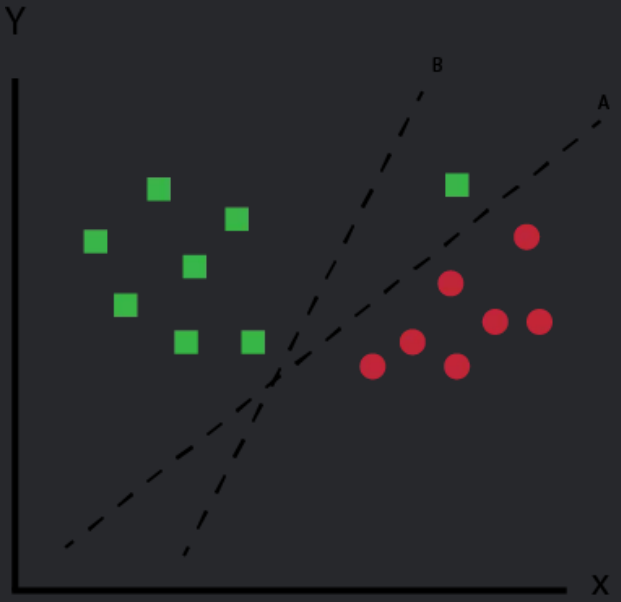
Prof. Dr. Diego Bruno

Education Tech Lead na DIO
Doutor em Robótica e *Machine Learning* pelo ICMC-USP

Vamos começar a programar...

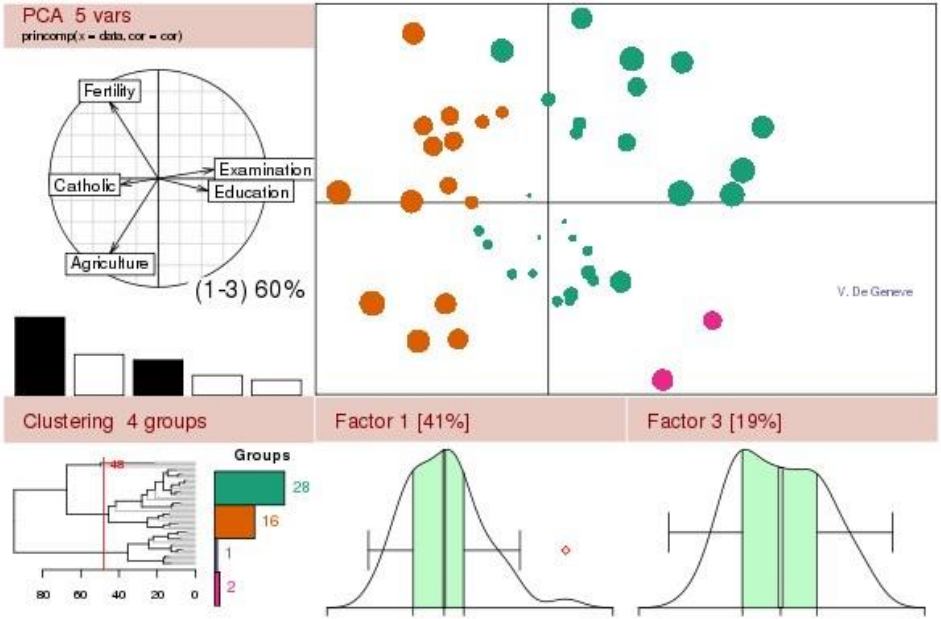
Prof. Dr. Diego Bruno

Machine Learning



Linguagens de Programação

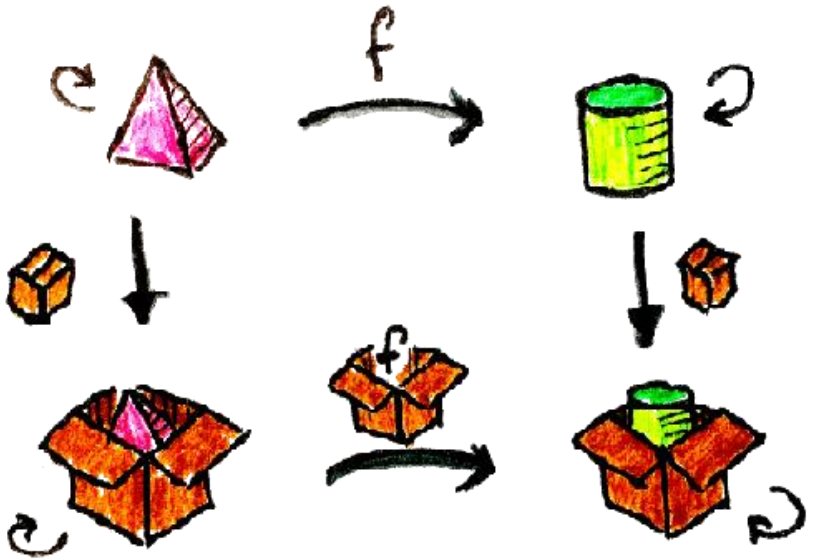
R é uma linguagem de programação multi-paradigma orientada a objetos, programação funcional, dinâmica, voltada à manipulação, análise e visualização de dados:



Paradigmas de Programação

Programação Funcional

Programação funcional é um paradigma de programação que trata a computação como uma avaliação de funções matemáticas.



2+2x3=?

Programação Funcional

Programação funcional é um paradigma de programação que trata a computação como uma avaliação de funções matemáticas.

Scheme

```
1  ((lambda (x) (+ x x)) (* 3 4))
```

Nesse caso, seria isso que aconteceria:

3*4 = 12;

x = 12;

x + x = 12 + 12 = 24;.

Linguagem funcional

Suporte para funcional

Programação em R

A versão base do R possui uma coleção enorme de funções:

- Modelos Estatísticos
- Algoritmos Computacionais
- Métodos Matemáticas
- Visualização de Dados

Programação em R

Pacotes:

Uma coleção de funções que podem ser escritas em R, C++, Fortran e C e que são chamadas diretamente de dentro do R.

Qualquer pessoa pode desenvolver seus pacotes e então submeter ao CRAN, disponibilizar através do *GitHub* ou *standalone*.

Programação em R

Mas as vezes não é suficiente:

Assim como alguns softwares estatísticos, o R também é extensível através de ”módulos”. Em R estes módulos são chamados de pacotes, bibliotecas ou packages.

Programação em R

As funcionalidades do R, podem ser ampliadas carregando estes pacotes, tornando um software ainda mais poderoso, capaz de realizar inúmeras tarefas:

- Análise multivariada;
- Análise Bayesiana;
- Manipulação de dados;
- Gráficos a nível de publicação;
- Big Data, Deep Learning;
- Processamento de imagens.

Programação em R

Alguns pacotes

- **maptools:** Funções para leitura, exportação e manipulação de estruturas espaciais.
- **cluster:** Funções para análise de clusters.
- **ggplot2:** Criação de gráficos elegantes.
- **rmarkdown:** criação de documentos (dinâmicos) em PDF, Word, HTML.
- **nlme:** Modelos lineares e não-lineares de efeitos mistos.

Obrigado!

Prof. Dr. Diego Bruno

Machine Learning

```
from tensorflow.keras.models import Model
from tensorflow.keras.layers import Dense


class MyModel(Model):

    def __init__(self, hidden_units, outputs, **kwargs):
        super(MyModel, self).__init__(**kwargs)
        self.dense = Dense(hidden_units, activation='sigmoid')
        self.linear = LinearMap(hidden_units, outputs)

    def call(self, inputs):
        h = self.dense(inputs)
        return self.linear(h)


my_model = MyModel(64, 12, name='my_custom_model')
```