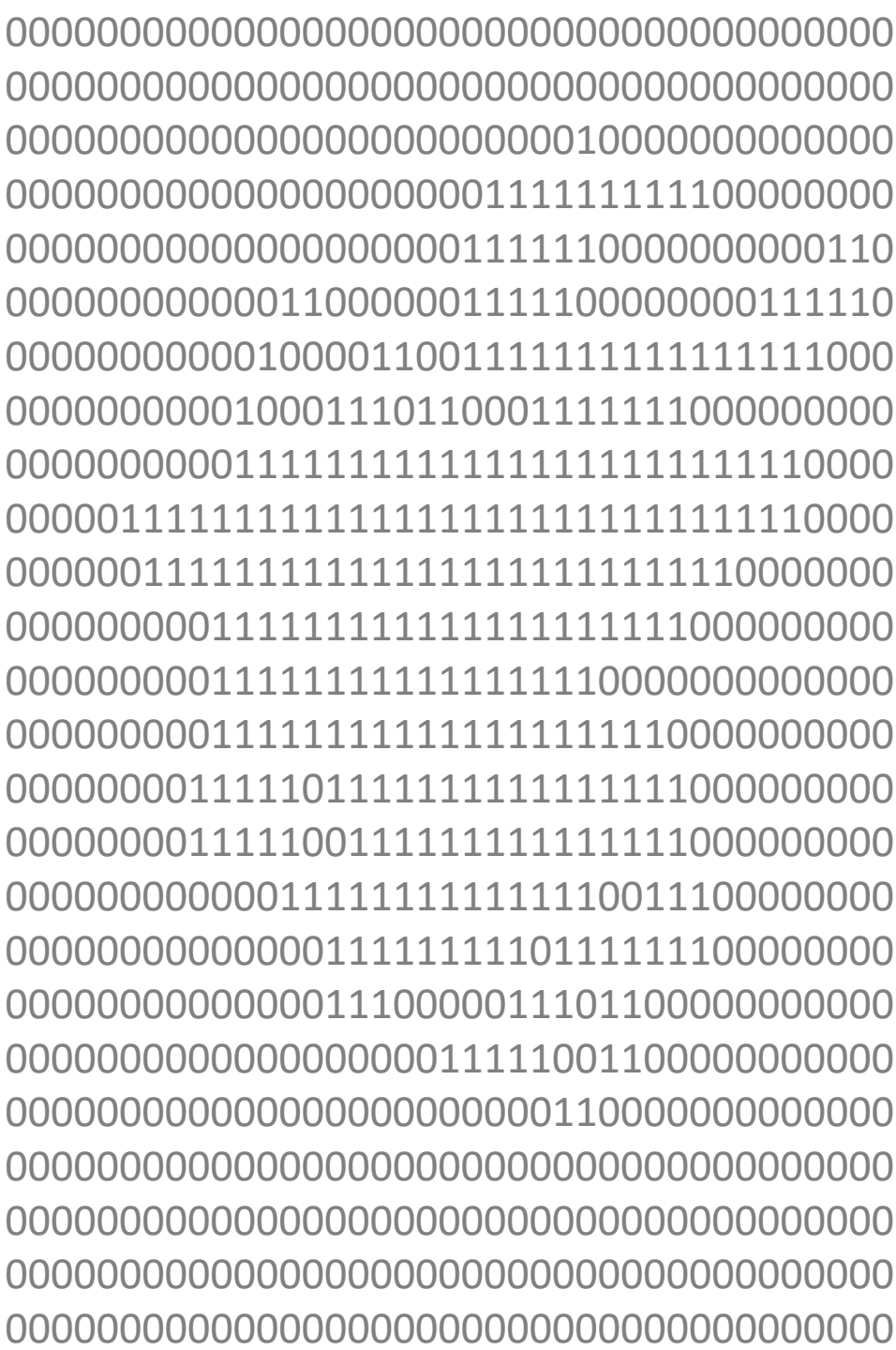


Introdução à Física Computacional

Aula 03

Um programa de computador nada mais é que uma sequência de instruções, que pode ser lida pelo computador. A sequência de instruções é o que o leva o computador a realizar uma dada tarefa. Em Python podemos ter um programa bem simples, que peça para que o computador mostre na tela uma sequência de DNA, como segue:

```
print("GATTACA")
```

A função *print()* gera na tela do computador a sequência de DNA, indicada abaixo.

```
python show_seq.py
GATTACA
```

Todo programa tem que estar num arquivo, o programa acima está no arquivo *show_seq.py.* Para executá-lo, precisamos chamar o aplicativo Python, com o comando *python show_seq.py* e depois <*Enter*>. No Windows, o comando é digitado na tela do Prompt de Comando, para o Mac OS X e Linux no Terminal.

Uma **linguagem de programação** é um conjunto cuidadosamente definido de regras de como escrever um programa de computador. As linguagens apresentam um conjunto de instruções e de regras de como essas instruções se relacionam. Pensando-se especificamente em linguagens de alto nível, as instruções (ou comandos) tendem a ser em inglês (como as instruções em Python mostradas ao lado). O programa escrito é chamado **código fonte**, ou simplesmente **código** ou **fonte**. O computador não entende tal linguagem diretamente. Assim, as instruções contidas num código fonte devem ser traduzidas para a linguagem de máquina, que é dependente de cada tipo de computador. Nos programas o símbolo "#" é usado para comentários. Tudo que estiver depois do símbolo #, e na mesma linha, é ignorada pelo Python.

Computador HAL 9000

Disponível em: <http://s374.photobucket.com/albums/oo184/jeanamann/?action=viewandcurrent=hal9000.jpgandnewest=1 >

```
# Program hal.py
print ("Dave, I'm loosing my mind...")
```

Normalmente, o Python não depende do computador. Posso escrever um programa num computador rodando Windows e executá-lo num computador rodando Mac OSX.

O Python é uma linguagem de alto nível, o que significa que suas instruções são próximas da linguagem humana, que não pode ser entendida pelo computador. A linguagem de alto nível necessita ser traduzida. Na tradução, as instruções em Python são convertidas para uma linguagem de máquina. A linguagem de máquina é binária, ou seja, formada por “zero” e “um” e de difícil programação, por isso temos as linguagens de alto nível. Na linguagem Python temos **um interpretador de Python**, que está instalado no computador. Como vimos, para acioná-lo basta digitarmos *python* (ou *python3*) e o nome do programa, por exemplo, se quisermos executar o programa *hal.py*, devemos digitar o seguinte comando e pressionar <*Enter*>:

```
python hal.py
Dave, I’m losing my mind...
```

Teremos o mesmo resultado se digitarmos diretamente o nome do programa. O interpretador de Python é capaz de executar as linhas de instruções de um programa em Python. O resultado da execução do programa *hal.py* é a mensagem “*Dave, I’m losing my mind...”.*

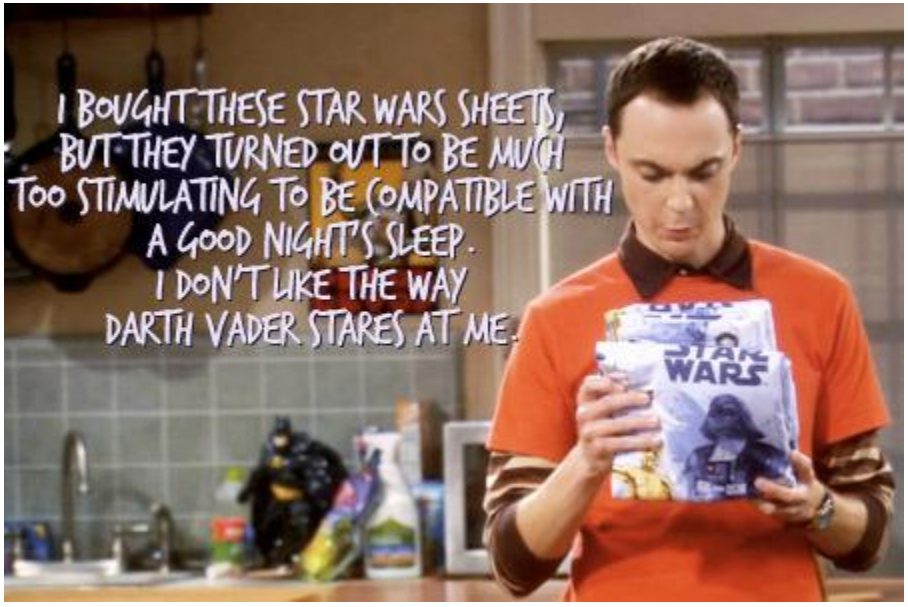
Uma das áreas do conhecimento humano onde a **lei de Murphy** aplica-se com constância é a programação. **"Se algo pode dar errado dará !".** Adaptando para o mundo da programação. **"Se algo pode ser digitado errado, será !".** Assim, temos que ter cuidado especial ao digitarmos as linhas de código de um programa. Vamos usar um programa simples, que mostra uma mensagem na tela e introduzir um erro de digitação e veremos o que acontece. O programa chama-se *dark.py,* o seu código fonte está listado abaixo.

Disponível em:
< http://favim.com/image/9492/>

```
# Program to show a message on screen
print ("You don't know the power of the dark side!\n")
```

Não digitamos o *n* de *print*. Salvamos o programa com o mesmo nome e acionamos o interpretador Python. O resultado está mostrado na tela azul abaixo.

```
python dark.py
Traceback (most recent call last):
  File "dark.py", line 2, in <module>
    prit ("You don't know the power of the dark side!\n")
NameError: name 'prit' is not defined
```



Disponível em:
< http://favim.com/image/44067/ >

Caos! Um monte de mensagens. E isto só porque esquecemos um inocente “n”. Se a interpretação do código fosse feita por um humano, tal erro de digitação seria claro e corrigido. Não é o caso do interpretador Python. O *prit* não é uma instrução reconhecida pelo interpretador Python, e ele faz questão de deixar isto bem claro. Os erros de digitação e outros serão indicados, criticados e o programa não será executado. Não temos nossa mensagem limpa na tela. A última mensagem “*NameError: name 'prit' is not defined.*”, resume o relatório de erros. A execução do programa *dark.py* foi abortada, pois “prit” não é uma instrução reconhecida do Python. Erros que impedem a execução do programa são chamados de **erro de sintaxe** ou **erro de execução**.

Disponível em:
< http://favim.com/image/40691/ >

Bem, você pode fazer todo curso assim, usando um editor de texto e executando os programas em Python via linha de comandos. Para isso, digite as linhas de código no editor de texto, salve o arquivo com a extensão *.py* e depois abra o Prompt de Comando e digite *python* e o nome do programa para executá-lo.

A alternativa que usaremos no curso é desenvolvermos os programas em um ambiente integrado de desenvolvimento, ou IDE (*Integrated Development Environment*, em inglês). A minha sugestão é que você usem o Pyzo.

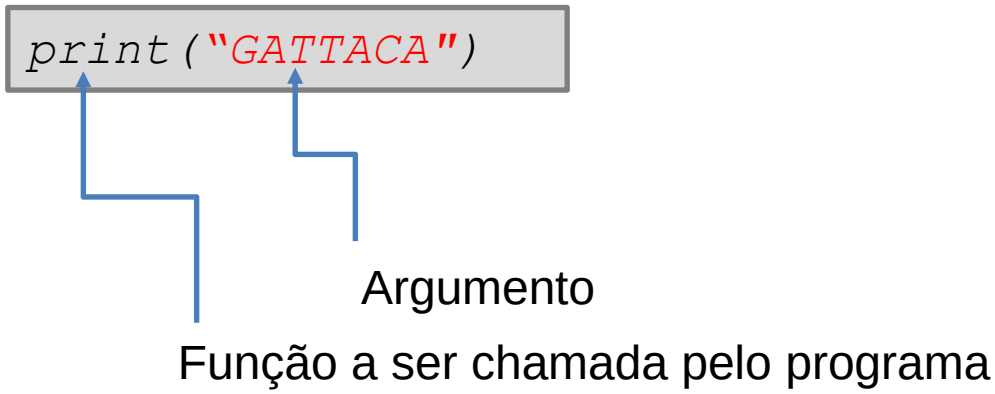
Página de entrada https://pyzo.org/.

Considerando que vocês já estão com o interpretador Python 3 instalado e o Pyzo, vamos digitar um programa bem simples. É comum iniciarmos com o programa “Hello World”, eu irei poupá-los disso. Se quiserem, podem escrever o programa “Hello World”. Seu primeiro programa chama-se *my_message.py,* e tem a duas linhas de código mostradas abaixo, coloque como argumento da função *print()* a mensagem que você quiser, não use acentos.

```
# Type your message in Python
print (“Your message \n")
```

Bem, já sabemos o que digitar. A questão é: como preparar o arquivo com o código fonte? Uma forma é usarmos um editor de texto, digitarmos as linhas acima e salvarmos o arquivo texto com extensão *.py*. Não é aconselhável usar o Word para a edição. Podemos usar o *WordPad* do Windows para isso, desde que tenhamos o cuidado de salvar como texto e com a extensão *.py*. Lembre-se, estamos considerando o ambiente Windows. Para o Linux ou Mac OS X podemos usar o editor *vi* ou outro disponível. Podemos ainda usar o ambiente integrado de desenvolvimento integrado *IDLE*, que tem um editor de texto e roda os scripts em Python de forma integrada. O *IDLE* não será usado no presente curso. Para seu código use o Pyzo, é fácil e totalmente integrado. Depois de digitar seu código, salve o arquivo e rode o programa no Pyzo, se tudo funcionar, parabéns, você já é um programador, com muito para aprender, mas quem não tem?

Vamos olhar mais cuidadosamente a função *print()*. Podemos pensar que a função *print()* é um miniprograma que executa uma tarefa específica, neste caso mostrar uma mensagem na tela. A mensagem pode está entre aspas, como em *print("GATTACA")*. A função é chamada pelo programa e executa sua tarefa, assim no jargão dizemos que a função *print()* foi chamada. O termo entre as aspas é chamado **argumento**, dizemos que passamos o argumento "GATTACA" para a função *print().*



As funções em Python podem retornar ou fornecer informações de volta para a parte do programa que chamou a função. Esses valores são chamados de valores de retorno (*return values*).

Ao chamarmos a função *print()*, esta mostra o argumento na tela e depois pula para a próxima linha (*newline*), se chamarmos outra função *print()* em seguida, a mensagem será mostrada na próxima linha, como no código *mult_lines.py* abaixo.

```
# Program to demonstrate print() function
print("First line.")
print("Second line.")
print("Third line with end parameter,",end=" ")
print("we still in the third line",end=", ")
print("finally at the end of third line.")
print("Forth line and end!")
```

Na primeira linha em vermelho, vemos o parâmetro *end*=“ “, que indica que a função *print()* terá um espaço em branco “ “ no final da linha e não uma nova linha (*newline*). A função *print()* mostrará no final o que estiver indicado no parâmetro *end*=“ “, podemos colocar uma vírgula no final, como mostrado na quinta linha do código acima. O uso do parâmetro *end*=“ “ força que ao chamarmos uma nova função *print()*, esta começará na linha anterior.

O resultado da execução do programa *mult_lines.py* está mostrado abaixo.

```
First line.
Second line.
Third line with end parameter, we still in the third line, finally at the end of third line.
Forth line and end!
```

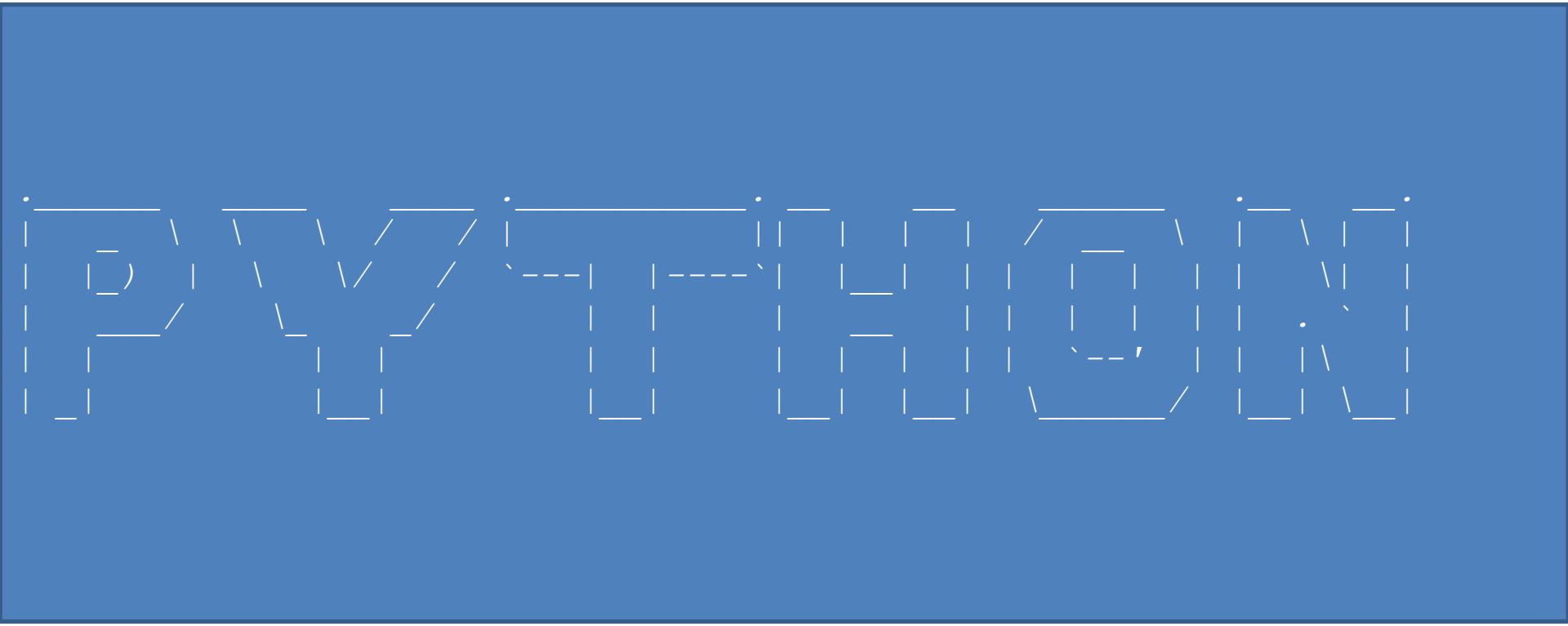
Podemos chamar a função *print()* para mostrar mensagens de texto mais complexas, usando-se o recurso das aspas duplas repetidas três vezes. Esse recurso permite que diversas linhas sejam mostradas entre a primeira aparição das """ e o fechamento das aspas """. No programa *ascii_art.py*, usamos o recurso de arte com ASCII (*American Standard Code for Information Interchange*), para colocarmos a palavra PYTHON com o grafismo do Star Wars. Quem tiver interesse em arte com ASCII, visite o site: *http://www.network-science.de/ascii/.*

```
# Program to demonstrate the use of """ in Python.
# ASCCII art available at: http://www.network-science.de/ascii/
# Font: Star Wars, size 80.

print("""


.______    ____    ____  ._____________. __    __    _______     .__   __.
|   _  \   \   \  /   / |             ||  |  |  |  /  __   \    |  \ |  |
|  |_)  |   \   \/   /  `---|  |----`|  |__|  | |  |  |  |  |    \|  |
|   ___/     \_    _/        |  |      |   __   | |  |  |  |  | . `  |
|  |            |  |          |  |      |  |  |  | |  `--'  |  |  |\   |
| _|            |__|          |__|      |__|  |__|  \______/   |__| \__|
"""
)
```

O código fonte está no arquivo *ascii_art.py*, ao executarmos o código temos o resultado abaixo.



Um recurso importante das linguagens de programação como Python e Perl são as **sequências de escape**. Uma sequência de escape indica uma ação específica, relacionada como a exibição de mensagens. Por exemplo, ao mostrarmos um argumento com a função *print()*, podemos ter a necessidade de pular mais de uma linha, para isto adicionamos a sequência de escape *\n* no argumento a ser passado para função *print()*, como indicado abaixo.

```
print("First line.\n")
print("Second line.")
print("Third line.")
```

A execução da segunda linha acima leva ao pulo de duas linhas, antes de ser mostrada a mensagem da segunda função *print().* A terceira linha é mostrada sem uma nova linha. O resultado da execução do código acima está mostrado abaixo.

```
First line.

Second line.
Third line.
```

A tabela abaixo traz as principais sequências de escape usadas em Python.

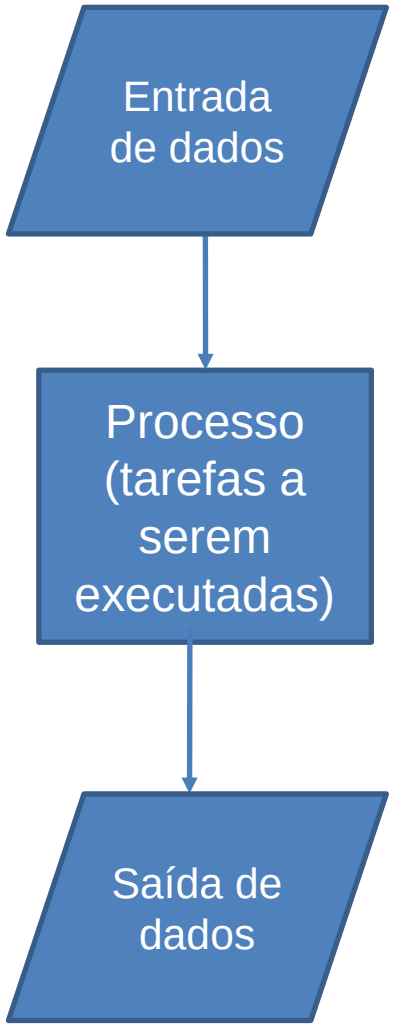
Sequência de escape	Descrição da sequência de escape
\n	Nova linha (*newline*)
\t	Tabulação
\\	Inserção de uma barra \
\’	Inserção de aspa simples
\”	Inserção de aspas duplas
\a	Faz beep ao ser executado
\b	Volta um espaço

O programa *escape_seq.py* ilustra as principais sequências de escape em Python.

```
# Program to demonstrate the main escape sequences in Python
print("Escape sequence for new line (\\n) => \n New line.")
print("Escape sequence for tab (\\t) =>\tWord1\tWord2\tWord3")
print('Escape sequence for double quotes (\\") => \" ')
print("Escape sequence for single quote (\\') => \' ")
print("Escape sequence for slash (\\\) => \\")
print("Escape sequence for backspace (\\b) => Word\bA")
print("Escape beep (\\a) \a")
```

O resultado da execução do código acima está mostrado abaixo.

```
Escape sequence for new line (\n) =>
 New line.
Escape sequence for tab (\t) => Word1   Word2   Word3
Escape sequence for double quotes (\") => "
Escape sequence for single quote (\') => '
Escape sequence for slash (\\) => \
Escape sequence for backspace (\b) => WorA
Escape beep (\a)
```



O que é um algoritmo?

Algoritmos são descrições passo a passo de uma tarefa, passível de implementação computacional. Por exemplo, uma descrição passo a passo de como resolver uma equação de segundo grau é um algoritmo. Cada programa de computador é a implementação de um algoritmo ou vários algoritmos. Quando você usa um programa, nada mais está fazendo que usando a implementação de algoritmos. O presente curso está focado no aprendizado de algoritmos e sua implementação na linguagem de programação Python. Os algoritmos serão estudados com aplicações na área de física computacional, com exceção aos primeiros algoritmos, que serão de aplicações gerais.

Ingredientes do nosso delicioso bolo de leite condensado sem farinha de trigo

Colocando a definição de uma forma mais simples ainda, podemos pensar no algoritmo como uma receita de bolo, por exemplo, a receita para prepararmos um bolo de leite condensado sem farinha de trigo.
Para o bolo temos os seguintes ingredientes:

-1 lata de leite condensado;
-4 ovos pequenos (ou 3 grandes);
-1 medida de leite igual ao volume da lata de leite condensado;
-100 g de coco ralado e
-uma colher de fermento.

Os ingredientes são as entradas do algoritmo.

Foto 1

Foto 2

Foto 3

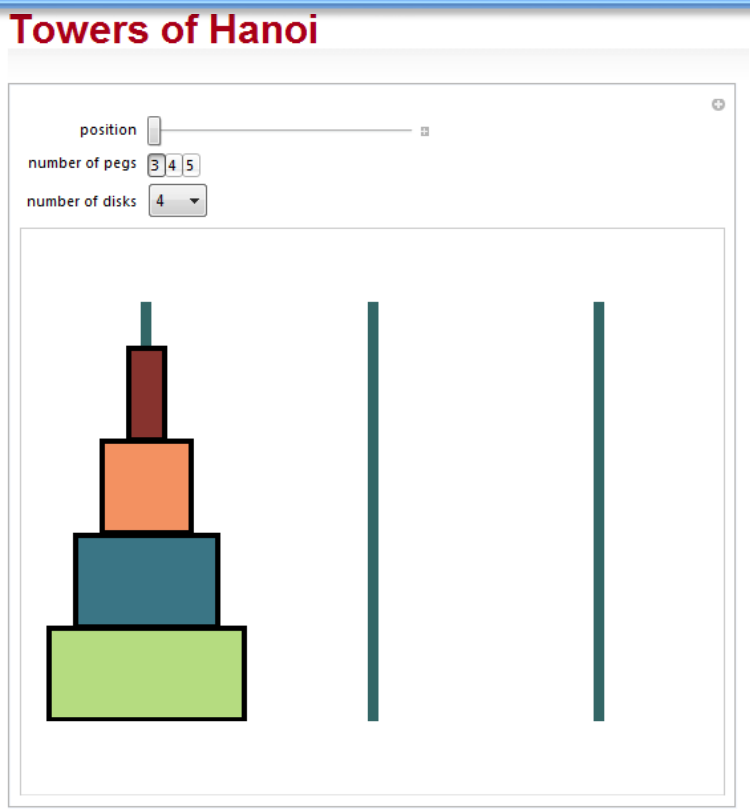
Foto 4

O processo do algoritmo é a receita, no nosso bolo é a seguinte: coloque todos os ingredientes num liquidificador e mexa tudo (Foto 1). Unte uma forma de bolo com manteiga (Foto 2). Coloque o conteúdo do liquidificador na forma untada. Asse no forno convencional a 260º por aproximadamente 40 minutos (Foto 3).

A saída é nosso bolo de leite condensado (Foto 4). Assim, na solução de muitos problemas científicos e tecnológicos, temos algoritmos.

Acredito que o conceito de algoritmo tenha ficado claro. Podemos então partir para uma definição computacional, extraída de um livro de algoritmos. *"Informalmente é qualquer procedimento computacional bem definido que toma algum valor ou conjunto de valores como entrada e produz algum valor ou conjunto de valores como saída. Também podemos visualizar um algoritmo como uma ferramenta para resolver um problema computacional bem especificado. O enunciado do problema especifica em termos gerais o relacionamento entre a entrada e a saída desejada. O algoritmo descreve um procedimento computacional específico para se alcançar esse relacionamento da entrada com a saída."* [1]

[1]Cormen, T.H, Leiserson, C.E., Rivest, R. L., Stein, C. Algoritmos. Teoria e Prática. 2ª edição. Editora Campus Ltda. Rio de Janeiro. p3.



O problema da Torre de Hanoi busca transferir todos os discos da pilha da esquerda para alguma das pilhas da direita. No processo de transferência, um disco maior não pode ficar sobre um menor.
A imagem acima é uma implementação de algoritmo para solução desse problema com o programa *Mathematica.*

TowersofHanoi-source.nb. Arquivo nb do *Mathematica* Disponível em:
<http://demonstrations.wolfram.com/TowersOfHanoi/>

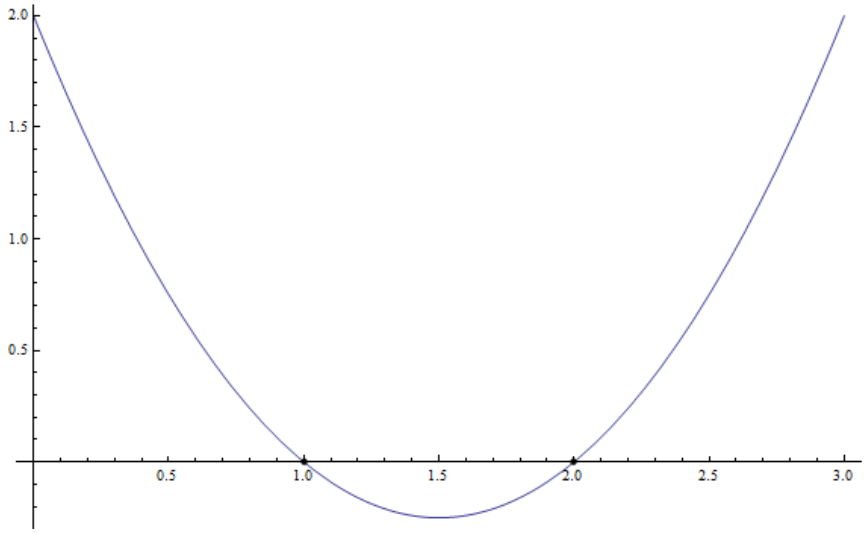
A melhor forma de fixarmos um conceito é aplicando-o a um problema prático já conhecido. Vamos considerar um algoritmo para a resolução da equação de segundo grau, mostrada abaixo.

$$ax^2 + bx + c = 0$$

Para resolver a equação vamos usar a fórmula de Bhaskara, que determina as duas raízes possíveis da equação de segundo grau (x1 e x2), como segue:

$$x1 = \frac{-b + \sqrt{\Delta}}{2a} \qquad x2 = \frac{-b - \sqrt{\Delta}}{2a}$$

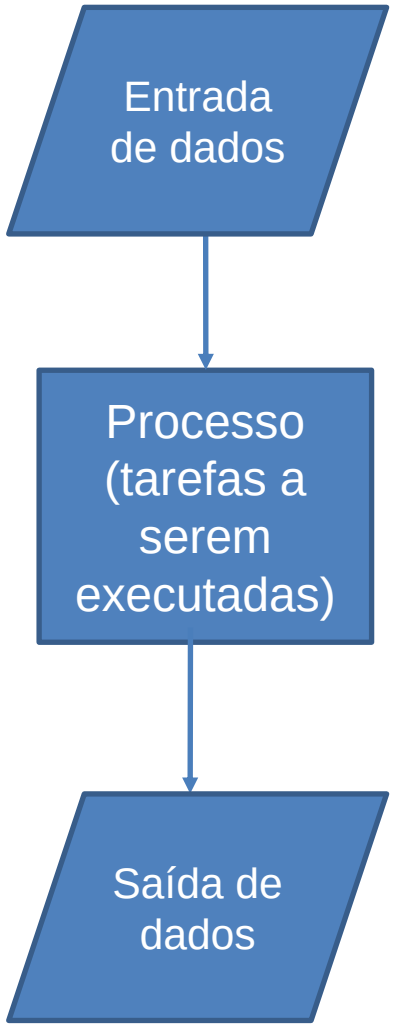
Onde $\Delta = b^2 - 4ac$.



Representação gráfica da função $y = x^2 - 3x + 2$. As raízes são os pontos onde a curva corta o eixo dos x, ou seja, 1 e 2.

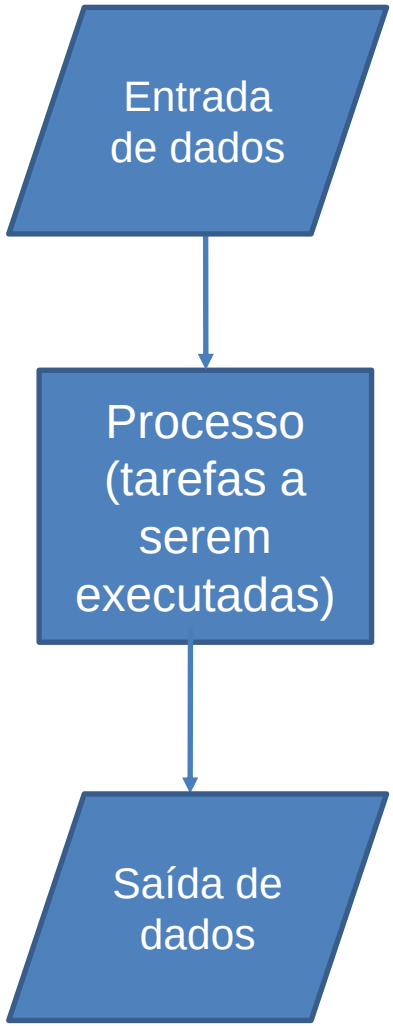
Disponível em:
<http://educacao.uol.com.br/matematica/bhaskara.jhtm>

Para facilitar, nosso algoritmo considera que a equação quadrática só terá raízes reais, ou seja, que o $\Delta \geq 0$.
Podemos usar o fluxograma geral de um algoritmo, para nos ajudar na implementação. Olhe o fluxograma ao lado, imagine onde encaixar a resolução da equação de segundo grau em cada etapa ao lado.



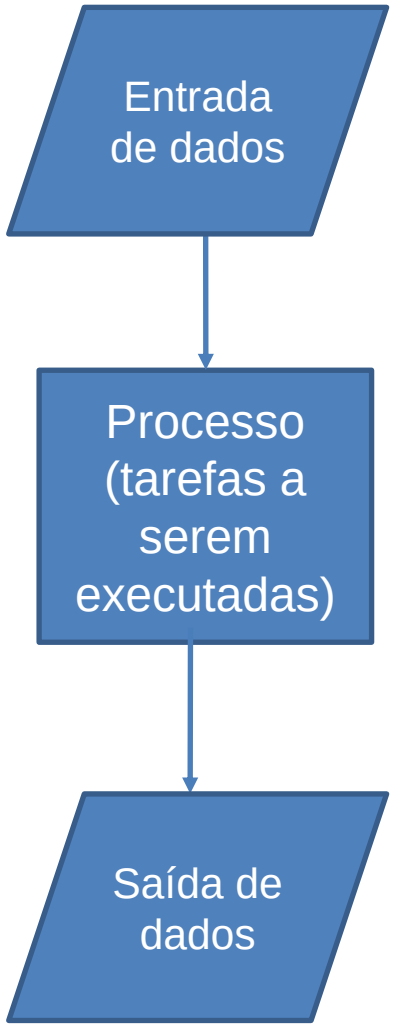
Disponível em:
http://educacao.uol.com.br/matematica/bhaskara.jhtm

Entrada de dados. Começando com a entrada de dados. Quais são as entradas? Na resolução de uma equação de segundo grau, precisamos dos coeficientes a, b e c. Então nossos dados de entrada são os coeficientes a, b e c. A entrada dos coeficientes poderia ocorrer por meio da leitura de um arquivo, onde tais números estariam armazenados. Outra possibilidade a ser considerada, o usuário do algoritmo fornecerá os coeficientes via teclado. Assim, o algoritmo tem que "perguntar" pelos coeficientes. Ou seja, o algoritmo tem que interagir com o usuário, de forma a obter a informação (coeficientes a, b e c). Vamos considerar que nosso algoritmo lerá as informações do teclado de computador.

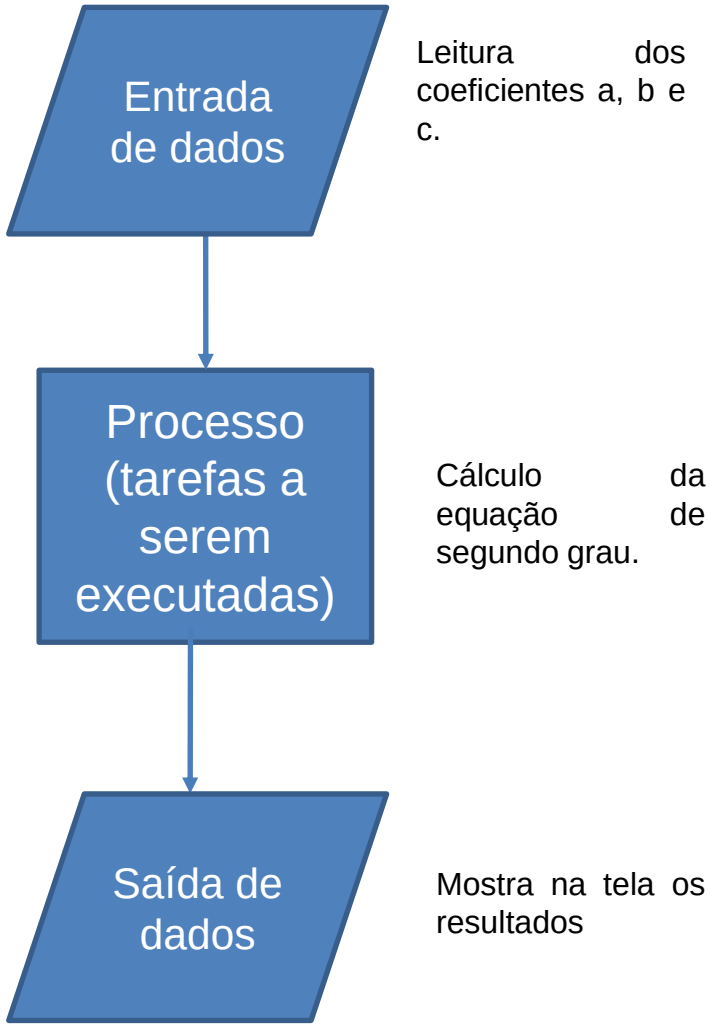


Leitura dos coeficientes a, b e c.

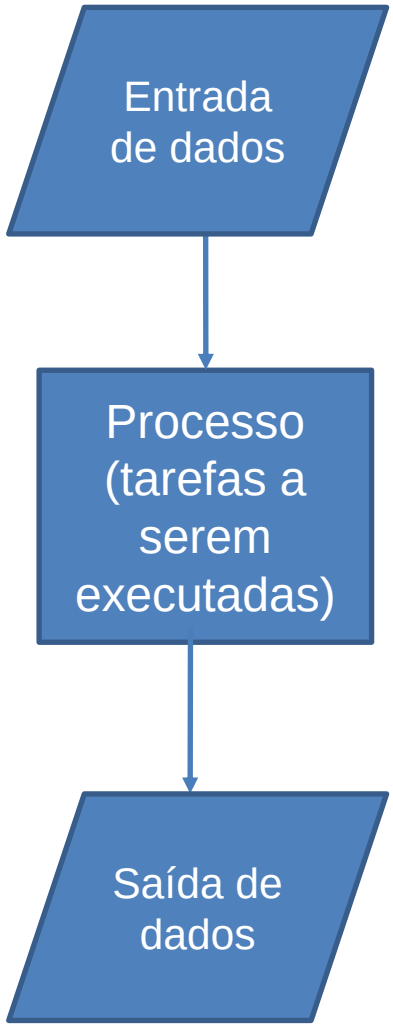
Processo. Nosso algoritmo leu os coeficientes (a, b e c). Assim, a informação fica disponível para uso no processo. Pensando em termos do computador, tal evento seria o armazenamento da informação na memória. O processo é o cálculo da equação de Bhaskara. Temos a equação para duas raízes, que podemos chamar x1 e x2. Antes de resolver a equação, temos que realizar um teste. Verificar se o $\Delta \geq 0$, caso satisfaça a condição, podemos calcular as raízes. Não satisfazendo a condição, temos que mostrar uma informação ao usuário, que no nosso algoritmo só trata com raízes reais. Essa situação ilustra a ideia de aplicabilidade, nosso modesto algoritmo pode ser usado para equações de segundo grau, mas somente aquelas com raízes reais.



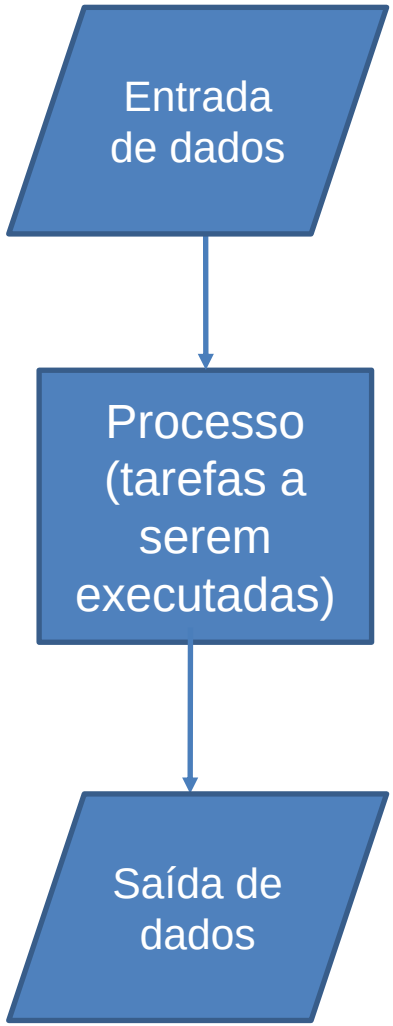
Saída de dados. Bem, calculamos as raízes da equação, agora vamos mostrar os resultados. As raízes da equação poderiam ser gravadas num arquivo de saída, para posterior análise, ou ainda poderiam ser mostrados na tela, ou ambos. Vamos optar pela última. Nosso algoritmo mostra os resultados (raízes da equação de segundo grau) na tela do computador. Na verdade o resultado pode ser uma mensagem dizendo que não há raízes reais para a equação, quando o $\Delta < 0$.



Agora já temos uma visão geral do funcionamento do nosso algoritmo para resolução da equação de segundo grau. Avançaremos para uma descrição mais detalhada do algoritmo. Usando a analogia com a receita de bolo, vamos descrever com detalhes os ingredientes da nossa receita (entrada de dados), bem como o processo (como calcular a equação). Há diversas formas de detalharmos um algoritmo. Uma já vimos, com o fluxograma, que é o diagrama esquemático desenhado ao lado. Poderíamos dividir o processo em outras caixas, com detalhamento do cálculo e eventuais decisões que temos que tomar no desenvolvimento do cálculo.



Outra forma de detalharmos o nosso algoritmo, é com o uso de **pseudocódigos**. A ideia de pseudocódigo é simples. Usamos uma descrição escrita das etapas a serem realizadas pelo algoritmo. Uma abordagem possível do pseudocódigo é deixá-lo o mais universal possível, sem levar em consideração características de implementação de uma linguagem de programação. Essa abordagem facilita a implementação do algoritmo em qualquer linguagem de programação de alto nível, como o Python. **Entenda que o pseudocódigo é um recurso para facilitar a programação, não é para ser interpretado e executado como o código fonte. O interpretador Python não entende pseudocódigo, o programador sim.**



Colocaremos o pseudocódigo da equação de segundo grau e detalharemos cada passo. A primeira linha é um formalismo para indicar o início e a última indica o fim.

Início

Leia (a, b, c)
Calcule o $\Delta = b^2 - 4ac$
Se ($\Delta \geq 0$) *então*

$$x1 = \frac{-b + \sqrt{\Delta}}{2a}$$

$$x2 = \frac{-b - \sqrt{\Delta}}{2a}$$

Escreva: "As raízes da equação são": x1, x2

Senão

Escreva: "Não há raízes reais"

Fim

Na segunda linha os valores dos coeficientes da equação de segundo grau são lidos. Na terceira linha temos o cálculo do valor de Δ, que será posteriormente avaliado e usado no cálculo das raízes da equação.

Início

Leia (a, b, c)
Calcule o $\Delta = b^2 - 4ac$
Se $(\Delta \geq 0)$ *então*

$$x1 = \frac{-b + \sqrt{\Delta}}{2a}$$

$$x2 = \frac{-b - \sqrt{\Delta}}{2a}$$

Escreva: "As raízes da equação são": x1, x2

Senão

Escreva: "Não há raízes reais"

Fim

Na quarta linha temos o teste do Δ. Se o valor calculado for maior ou igual a zero, o algoritmo realiza a tarefa indicada no quadro verde, caso contrário (Δ < 0), o algoritmo faz o que está indicado do quadro vermelho. O quadro verde é o cálculo das duas raízes reais (x1 e x2) da equação do segundo grau. O quadro vermelho exibe a mensagem que não há raízes reais.

Início

Leia (a, b, c)
Calcule o $\Delta = b^2 - 4ac$
Se $(\Delta \geq 0)$ *então*

$$x1 = \frac{-b + \sqrt{\Delta}}{2a}$$

$$x2 = \frac{-b - \sqrt{\Delta}}{2a}$$

Escreva: "As raízes da equação são": x1, x2

Senão

Escreva: "Não há raízes reais"

Fim

Fluxograma. Vamos indicar ao lado os quatro dos principais símbolos usados em fluxogramas. O terminal é usado para indicação do início ou do fim do algoritmo. Tais indicadores facilitam a leitura do algoritmo, deixando claro onde devemos começar a execução e onde o algoritmo acaba.

Outro símbolo é a caixa de processo, indica algum procedimento do algoritmo, como cálculos por exemplo.

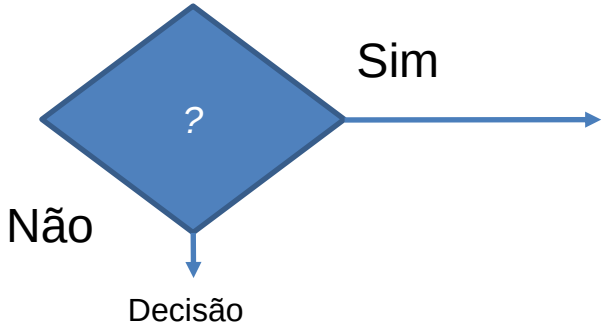
O paralelogramo indica entrada e saída, usado para leitura de dados e exibição dos resultados na tela.

O losango é a caixa de decisão, se a condição indicada no losango é satisfeita, o algoritmo segue o caminho do “Sim”, caso contrário o caminho do “Não”.

Terminal (Início ou Fim)

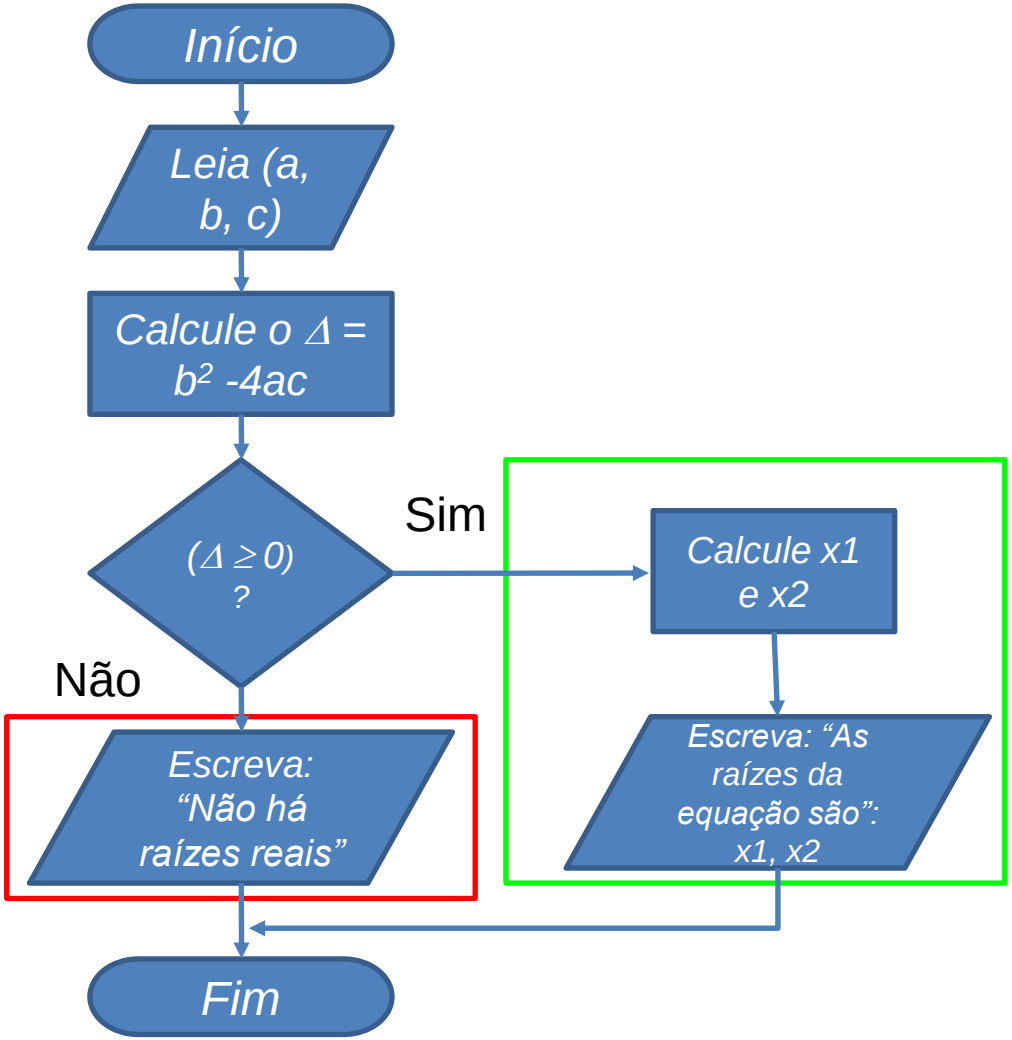
Processo

Entrada/saída



O fluxograma para o algoritmo de resolução da equação de segundo grau está mostrado ao lado. Os quadros vermelho e verde não fazem parte do fluxograma, estão colocados somente para indicar o bloco que é executado quando o $\Delta \geq 0$ (verde) e quando o Δ não satisfaz a condição (vermelho), ou seja, quando é menor que zero.

O fluxograma é normalmente lido de cima para baixo, com setas indicando a sequência de execução do fluxograma.



Uma parte importante do uso de uma linguagem de programação está centrada na manipulação de números. Os números em Python usam até 17 dígitos, como o número 1.3333333333333333. Esses números são chamados de ponto flutuante de precisão dupla, podemos pensar na representação dos números reais, sem obviamente a abrangência do conjunto dos números reais. As limitações são o número de dígitos (17), e os limites variam de entre os sistemas operacionais e hardwares. Por exemplo, no Windows 8.1, o Python 3.4.0 tem os seguintes limites de números de ponto flutuante: 1e-323 e 1e308. O número 1.10^{308} é representado em Python pela notação 1e308. Abaixo temos uma lista de valores numéricos possíveis em Python.

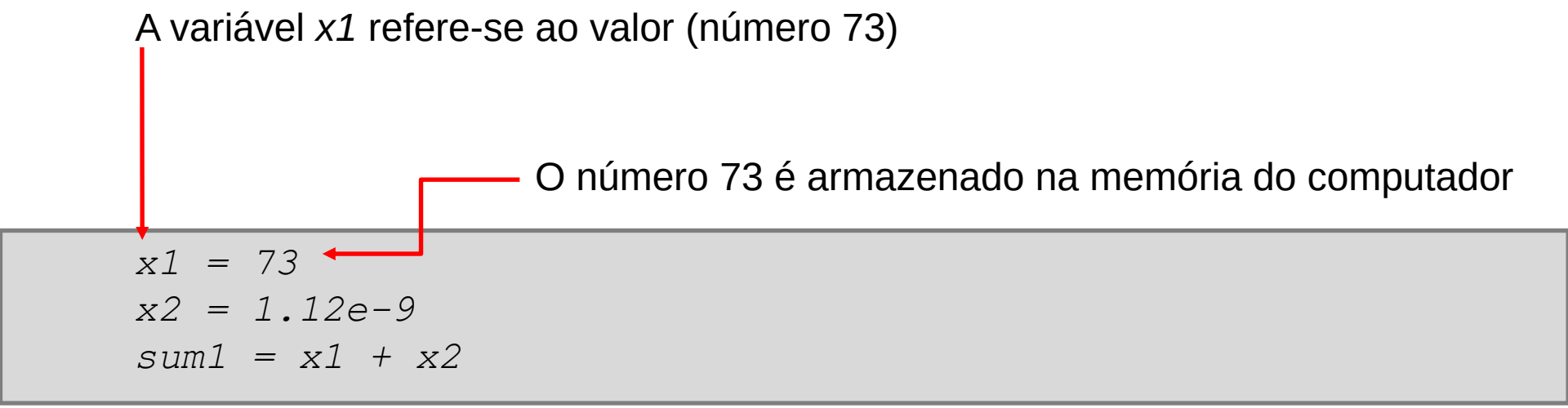
```
1
1.12e-9
3.141592653589793
-3.141592653589793
1.11E-9
1234567890
```

Veremos vários trechos simples de programas em Python, que executam operações aritméticas básicas. Depois implementaremos em Python a solução da equação de segundo grau, usando o algoritmo que desenvolvemos nos slides anteriores. Em Python temos um conjuntos de operadores para tais funções matemáticas. Usando nossa abordagem para o aprendizado, veremos através de exemplos escritos em Python. Inicialmente trabalharemos com variáveis numéricas. Abaixo temos exemplos de variáveis.

```
x1 = 73
x2 = 1.12e-9
sum1 = x1 + x2
```

Especificamente nas linhas de código acima, as variáveis *x1* e *x2* referem-se aos valores “73” e 1.12e-9”. Tais valores serão somados e atribuídos à variável *sum1*. Uma variável é uma forma de rotularmos e acessarmos informações, por exemplo a informação da variável *x1*. A primeira linha de código acima cria a variável *x1*, esta linha é chamada comando de atribuição (*assignment statement*).

Tecnicamente, dizemos que um **comando de atribuição armazena o valor do lado direito do sinal de igual (=) na memória do computador**, enquanto a **variável do lado esquerdo somente refere-se ao valor**, e não o armazena diretamente. Assim podemos dizer que a variável *x1* obtém o valor 73, ao invés de dizermos que é atribuído o valor 73 à variável. Muitos autores usam ambos os termos com o mesmo sentido, por exemplo: DAWSON, Michael. **Python Programming, for the absolute beginner.** 3ed. Boston: Course Technology, 2010, página 31.



Sobre as variáveis, quando escolhemos um nome para um dada variável, por exemplo, *x3*, temos que seguir alguns critérios. Uma variável inicia com uma letra ou *underscore* "_", não usamos números para iniciar o nome da variável. Podemos usar letras maiúsculas ou minúsculas, mas tenha em mente que o Python diferencia entre elas, assim as variáveis *X3* e *x3* são duas variáveis distintas. **Uma variável pode ser alterada durante a execução do programa, conforme a necessidade.** Só não muda o seu endereço de memória. Segue uma lista de nomes de variáveis permitidas em Python, para as quais valores foram atribuídos usando o operador atribuição (=).

```
x3 = 3.14
dark_side = 0.0
_So_it_is_a_variable = 1.0e-99
doce_de_jaca = 0.00008
my_variable_1 = 1.00001
_  = 2
```

Segue um código em Python, onde os valores das variáveis são mostrados na tela. O nome do programa é *test_of_variables.py* .

```
x3 = -1.0
X3 = 3.14
dark_side = 0.0
_So_it_is_a_variable = 1.0e-99
doce_de_jaca = 0.00008
my_variable_1 = 1.00001
_  = 2
_123_  = 8.123e-5
_1_2_3_Variable = -1.4e-190
minimumFloatValue = 1e-323
maximumFloatValue = 1e308
bigInteger = 1234567890123456
print ("x3 = ",x3," and X3 = ", X3)
print ("\ndark_side = ", dark_side)
print ("\nSo_it_is_a_variable = ", _So_it_is_a_variable)
print ("\ndoce_de_jaca = ", doce_de_jaca)
print ("\nmy_variable_1 = ", my_variable_1)
print ("\n_  = ", _)
print ("\n_123_  = ", _123_)
print ("\n_1_2_3_Variable = ", _1_2_3_Variable)
print("\nminimumFloatValue = ",minimumFloatValue)
print("\nmaximumFloatValue = ",maximumFloatValue)
print("\nbigInteger = ", bigInteger)
```

Abaixo temos o resultado de rodarmos o código *test_of_variables.py* .

```
Examples of variables in Python.

x3 = -1.0 and X3 = 3.14

dark_side =  0.0

So_it_is_a_variable =  1e-99

doce_de_jaca =  8e-05

my_variable_1 =  1.00001

_ =  2

_123_ =  8.123e-05

_1_2_3_Variable =  -1.4e-190

minimumFloatValue =  1e-323

maximumFloatValue =  1e+308

bigInteger =  1234567890123456
```

Vimos até agora o uso de variáveis que já apresentavam um valor inicial. Destacamos que podemos mudar os valores da variável ao longo do programa, por exemplo veja o código abaixo, chamado *change_value.py.*

```
x1 = 73                     # Assigns value 73 to variable x1
print("x1 = ",x1)           # Shows results on screen
x1 = 3.14159                # Assigns value 3.14159 to variable x1
print("x1 = ",x1)           # Shows results on screen
```

Inicialmente atribuímos 73 à variável *x1*, mostramos o resultado na tela, com a função *print()*. Depois atribuímos o valor 3.14159 à mesma variável *x1* e mostramos o novo resultado na tela. Ao executarmos o código acima, temos os resultados abaixo.

```
x1 =  73
x1 =  3.14159
```

Veremos a seguir como inserir valores a uma variável durante a execução do programa, ou seja, como um usuário do programa pode atribuir valor a uma variável, sem ter que editar o código fonte.

A função *input()* do Python permite que seja atribuído um valor a uma variável pelo usuário, durante a execução do código. Essa função permite que o usuário interaja com o programa, e não simplesmente assista a sua execução. Abaixo temos uma linha de código com as funções *int()* e *input().*

```
x1 = int(input("Enter a number =>"))
```

Na linha de código acima, a função *input()* tem como argumento a string *"Enter a number =>* ". Esta mensagem será mostrada na tela, como no caso da função *print()*, a novidade aqui é que ao encontrar a função *input()*, o interpretador Python realiza uma pausa na execução do restante do programa. O programa fica esperando que o usuário digite algum valor via teclado e pressione <*Enter*>. Na linha de código acima, ao entrar um valor, este é convertido de string para um número inteiro, devido à função *int().* O valor lido e convertido para inteiro é atribuído à variável *x1*. Usamos a função *int()* para garantir que o número lido seja inteiro, se deixássemos sem a função *int()* a entrada seria considerada uma string. Outra possibilidade é usarmos a função *float()*, que converte a entrada para um número de ponto de flutuante, a linha de comando para a leitura de um número de ponto flutuante está mostrada abaixo.

```
x1 = float(input("Enter a number =>"))
```

A linguagem Python tem os operadores matemáticos básicos. Para ilustrar seu uso, discutiremos um programa que realiza a leitura de dois números e aplica o operador a eles. Assim temos que definir os nomes das variáveis, que serão *x1* e *x2*, o resultado do operador matemático será atribuído à variável *result*. Poderíamos escrever o algoritmo antes e depois implementar em Python, mas a aplicação direta dos operadores é tão simples, que não vale o esforço de escrevermos o algoritmo. **Vamos direto ao Python!** O quadro abaixo ilustra a linha de comando com a aplicação do operador soma (+) às variáveis *x1* e *x2*. O operador atribuição (=) atribui o resultado da operação realizada à direita (*x1 + x2*), à variável indicada à esquerda (*result*).

```
result = x1 + x2
```

Para a entrada dos valores das variáveis *x1* e *x2*, usaremos a função *input()* do Python. Por exemplo, para a leitura da variável *x1* do tipo de ponto flutuante, usamos:

```
x1 = float(input("Enter a number => "))
```

A função *input()* mostra a mensagem na tela (argumento da função), depois espera que seja digitado um valor via teclado. O valor digitado será convertido para ponto flutuante e atribuído à variável, indicada no lado esquerdo do operador atribuição.

Abaixo temos uma tabela com os principais operadores matemáticos disponíveis em Python.

Operador matemático	*Descrição*	*Exemplo*
+	*Soma*	*result = x1 + x2*
-	*Subtração*	*result = x1 - x2*
*	*Multiplicação*	*result = x1 * x2*
/	*Divisão*	*result = x1 / x2*
%	*Módulo (resto inteiro da divisão)*	*result = x1 % x2*
//	*Divisão inteira (parte inteira da divisão, sem arredondamento)*	*result = x1 // x2*
**	*Potenciação*	*result = x1 ** x2*

O operador = é chamado operado atribuição, ou seja, atribui o resultado da direita à variável da esquerda.
Temos os operadores acima implementados no código *math_op1.py*.

Código fonte do programa *math_op1.py*

```
# Program to demonstrate basic mathematical operators in Python.
x1 = float(input("Enter a number => "))      # Reads value for variable x1
x2 = float(input("Enter a number => "))      # Reads value for variable x2
result = x1 + x2                             # Applies sum (+) operator
print("Sum = ",result)                       # Shows result on screen
result = x1 - x2                             # Applies subtraction (-) operator
print("Subtraction = ",result)               # Shows result on screen
result = x1*x2                               # Applies multiplication (*) operator
print("Multiplication = ",result)            # Shows result on screen
result = x1/x2                               # Applies division (/) operator
print("Division = ",result)                  # Shows result on screen
result = x1%x2                               # Applies modulus (%) operator
print("Modulus = ",result)                   # Shows result on screen
result = x1//x2                              # Applies floor division (//) operator
print("Floor Division = ",result)            # Shows result on screen
result = x1**x2                              # Applies exponentiation (+) operator
print("Exponentiation = ",result)            # Shows result on screen
```

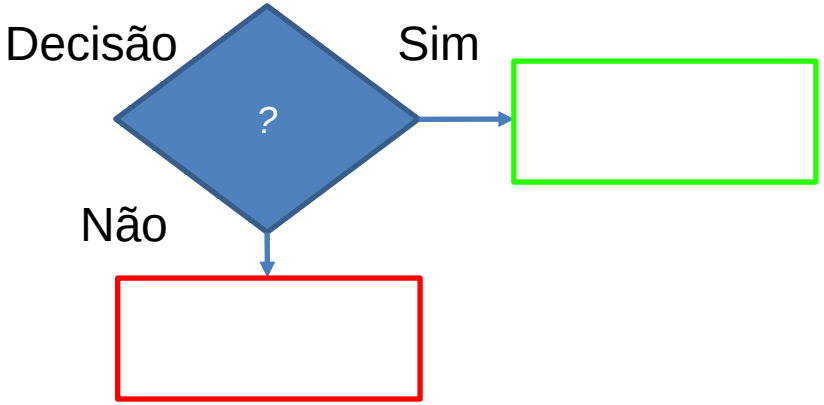
Se entrarmos 7 para a variável *x1* e 3 para a variável *x2*, temos os resultados abaixo:

```
Enter a number => 7
Enter a number => 3
Sum =  10.0
Subtraction =  4.0
Multiplication =  21.0
Division =  2.3333333333333335
Modulus =  1.0
Floor Division =  2.0
Exponentiation =  343.0
```

O programa funcionou sem erros, mas se introduzirmos o segundo número como zero, veja o resultado.

```
Enter a number => 7
Enter a number => 0
Sum =  7.0
Subtraction =  7.0
Multiplication =  0.0
Traceback (most recent call last):
  File "C:\Users\Walter\workspace\Bioinfo_Aula1\mathOp.py", line 10, in <module>
    result = x1/x2                                  # Applies division (/)
operator
ZeroDivisionError: float division by zero
```

O erro ocorreu pois entramos um valor não válido para divisão, e para os operadores relacionados (módulo e divisão inteira). Não podemos dividir por zero, assim seria interessante testarmos se o divisor (*x2*) é zero e emitir uma mensagem de erro. Quando apresentamos os símbolos mais usados para montagem de fluxogramas, vimos a caixa de decisão, o condicional "if", mostrado abaixo.




```
if(condição verdadeira):
        ......

else:
        ....
```

O condicional "if" testa uma situação, e se for verdadeira executa um trecho do código, caso contrário executa outra parte do código. É uma forma de rompermos a sequência natural de execução das linhas de comandos presentes num programa, que seguem de cima para baixo, como vimos nos programas anteriores. A presença da **estrutura de controle** "if", muda a sequência natural de execução. Em Python usamos a construção indicada no quadro cinza acima. Quando a condição for verdadeira, executamos o trecho em verde, caso não seja executamos o trecho em vermelho.

Quando usamos o condicional *if* para números, temos os seguintes operadores relacionais (de comparação).

Operador	Significado	Exemplo de teste condicional	Resultado do teste condicional
!=	Não é igual a	*7 != 3*	*True* (verdadeiro)
==	É igual a	*7 == 7*	*True* (verdadeiro)
>	Maior que	*7 > 3*	*True* (verdadeiro)
<	Menor que	*3 < 7*	*True* (verdadeiro)
>=	Maior ou igual a	*7 >= 8*	*False* (falso)
<=	Menor ou igual a	*8 <= 7*	*False* (falso)

O novo código chama-se *math_op2.py*.

Código fonte do programa *math_op2.py*

```
# Program to demonstrate basic mathematical operators in Python.
# In addition, we test if divisor is zero.
x1 = float(input("Enter a number => ")) # Reads value for variable x1
x2 = float(input("Enter a number => ")) # Reads value for variable x2
result = x1 + x2                        # Applies sum (+) operator
print("Sum = ",result)                  # Shows result on screen
result = x1 - x2                        # Applies subtraction (-) operator
print("Subtraction = ",result)          # Shows result on screen
result = x1*x2                          # Applies multiplication (*) operator
print("Multiplication = ",result)       # Shows result on screen
if x2 != 0:                             # Tests if x2 is different from zero
    result = x1/x2                      # Applies division (/) operator
    print("Division = ",result)         # Shows result on screen
    result = x1%x2                      # Applies modulus (%) operator
    print("Modulus = ",result)          # Shows result on screen
    result = x1//x2                     # Applies floor division (//) operator
    print("Floor Division = ",result)   # Shows result on screen
else:
    print("Error! Divisor should be different from zero!")  # Shows error message
result = x1**x2                         # Applies exponentiation (+) operator
print("Exponentiation = ",result)       # Shows result on screen
```

Se entrarmos 7 e 3, temos os resultados esperados:

```
Enter a number => 7
Enter a number => 3
Sum =  10.0
Subtraction =  4.0
Multiplication =  21.0
Division =  2.3333333333333335
Modulus =  1.0
Floor Division =  2.0
Exponentiation =  343.0
```

Entrando 7 e 0, temos os seguintes resultados.

```
Enter a number => 7
Enter a number => 0
Sum =  7.0
Subtraction =  7.0
Multiplication =  0.0
Error! Divisor should be different from zero!
Exponentiation =  1.0
```

Método raiz quadrada (*math.sqrt()*). Para raiz quadrada, só precisaremos de uma variável, *x1*. Aplicaremos o método *math.sqrt()* ao valor atribuído à *x1*, tal inserção pode causar problemas, visto que a raiz quadrada não gera resultados em números complexos. Para evitar números complexos, testamos se o valor atribuído à variável *x1* é maior ou igual a zero, satisfazendo à condição, calculamos a raiz, caso contrário, mostramos uma mensagem de erro. O método *math.sqrt()* faz parte do módulo *math*, que tem que ser importado antes da chamada do método. O módulo *math* é importado com o comando: *import math*, como mostrado no abaixo. Alternativamente, podemos usar o operador potenciação (**0.5) para a raiz quadrada, o que permite resultados complexos.

Código fonte do programa *square_root.py*

```
# Program to demonstrate the method math.sqrt() from math module in Python
import math                                  # Imports module math
x1 = float(input("Enter a number => ")) # Reads x1
if x1 >= 0:                                  # Tests if x1 is greater or equal to zero
    result = math.sqrt(x1)                   # Calculates square root
    print("Square root = ",result)           # Shows result on screen
else:
    print("Error! Entry should be >=0 .")    # Shows error message
```

Vamos testar para um número maior que zero (exemplo 1), para zero (exemplo 2) e para um número negativo (exemplo 3).

Exemplo 1

```
Enter a number => 73
Square root =  8.54400374531753
```

Exemplo 2

```
Enter a number => 0
Square root =  0.0
```

Exemplo 3

```
Enter a number => -1
Error! Entry should be >=0 .
```

O logaritmo na base e (e = 2,718281828459045...) é definido pelo método *math.log()*. O "e" é chamado de número de Euler. O quadro abaixo ilustra a programa para uso do método *math.log*. Este método e muitos outros estão definidos no módulo *math*. A função *log* é definida para números maiores que zero, tal condição será considerada com o teste da variável *x1*. Números maiores que zero tem seu *log* calculado, os outros geram uma mensagem de erro. Maiores informações sobre o método *math.log* disponíveis em: < http://www.tutorialspoint.com/python/number_log.htm >.

Código fonte do programa *log.py*

```
# Program to demonstrate how to calculate log using the method math.log() from
# math module.
import math                                          # Imports math module
x1 = float(input("Enter a number => "))              # Reads x1
if x1 > 0:                                           # Tests if x1 >= 0
    result = math.log(x1)                            # Calculates log(x)
    print("Log = ", result)                          # Shows result on screen
else:
    print("Error! Number should be >= 0")            # Shows error message
```

Função interna logaritmo natural (log()). Realizaremos 3 testes, no primeiro calculamos o logaritmo natural de e = 2.718281828459045..., no segundo teste calculamos o log de 1 e, por último, o log de 0.

Exemplo 1

```
Enter a number => 2.718281828459045
Log =  1.0
```

Exemplo 2

```
Enter a number => 1
Log =  0.0
```

Exemplo 3

```
Enter a number => 0
Error! Number should be >= 0
```

O módulo *math* tem um método para o logaritmo na base 10, *math.log10()*. O funcionamento é análogo ao método do logaritmo natural, a implementação segue a mesma lógica de programação, como mostrado no código abaixo.

Código fonte do programa *log10.py*

```
# Program to demonstrate how to calculate log10 using the method
# math.log10() from math module.
import math                                          # Imports math module
x1 = float(input("Enter a number => "))              # Reads x1
if x1 > 0:                                           # Tests if x1 >= 0
    result = math.log10(x1)                          # Calculates log(x)
    print("Log10 = ", result)                        # Shows result on screen
else:
    print("Error! Number should be >= 0")            # Shows error message
```

Logaritmo. Calcularemos o log na base 10 de 100 (exemplo 1), de 1 (exemplo 2) e de 0 (exemplo 3).

Exemplo 1

```
Enter a number => 100
Log10 =  2.0
```

Exemplo 2

```
Enter a number => 1
Log10 =  0.0
```

Exemplo 3

```
Enter a number => 0
Error! Number should be >= 0
```

Notação científica. Podemos entrar dados na forma de notação científica, usando o seguinte formato:

$$a,bc \cdot 10^{d} = a.bced$$

Vamos considerar a entrada de dados em notação científica do número $7,3.10^{-3}$ em Python fica da seguinte forma:

7.3e-3 (exemplo abaixo)

Aplicaremos a notação científica no programa de logaritmo na base 10.

```
Enter a number => 7e-3
Log10 =  -2.154901959985743
```

Outros métodos. Como outras linguagens de alto nível, o Python tem um arsenal de métodos matemáticos pré-definidos. O quadro abaixo traz uma lista desses métodos disponíveis no módulo *math*.

Método no módulo *math*	**Descrição**
math.atan2(y, x)	Calcula o arco-tangente em radianos do valor de y/x. O valor calculado está no intervalo de –Pi a +Pi.
math.cos(x)	Calcula o cosseno de x em radianos.
math.sin(x)	Calcula o seno de x em radianos.
math.tan(x)	Calcula a tangente de x em radianos.
math.exp(x)	Calcula e elevado à potência de x.
math.pow(x,y)	Calcula x elevado a y.
math.fabs(x)	Calcula o valor absoluto de x.
math.degrees(x)	Converte o ângulo x de radianos para graus.
math.radians(x)	Converte o ângulo x de graus para radianos.

Disponível em: < http://docs.python.org/2/library/math.html >

Podemos agora implementar em Python o algoritmo para solução da equação de segundo grau. À esquerda temos o pseudocódigo e à direita a implementação em Python. Usamos um código de cores, para mostrar a equivalência entre o pseudocódigo e o código em Python.

Início
Chama módulo math
Leia (a, b, c)

Calcule o $\Delta = b^2 - 4ac$
Se ($\Delta \geq 0$) então

$$x1 = \frac{-b + \sqrt{\Delta}}{2a}$$

$$x2 = \frac{-b - \sqrt{\Delta}}{2a}$$

Escreva: "As raízes da equação são": x1, x2
Senão
Escreva: "Não há raízes reais"
Fim

Código fonte do programa *bhaskara.py*

```
# Program to solve quadratic equation u
import math  # Imports math module

a = float(input("Enter coefficient a => ")) # Reads a
b = float(input("Enter coefficient b => ")) # Reads b
c = float(input("Enter coefficient c => ")) # Reads c
delta = b**2 - 4*a*c   # Calculates delta

if delta>=0 :  # Checks whether delta >= 0

    x1 = (-b + math.sqrt(delta))/(2*a)#Calculates x1


    x2 = (-b - math.sqrt(delta))/(2*a)#Calculates x2



    print("The roots are: ", x1," ",x2)#Shows results

else:
    print("There are no real roots!") # Shows message
```

O “Fim” do pseudocódigo não precisa ser implementado em Python.
Vamos testar nosso código com dois exemplos. No primeiro temos raízes reais (x^2 + 10 x + 21 = 0) (exemplo 1). No segundo exemplo calculamos as raízes da equação (x^2 + x + 1 = 0) (exemplo 2), que não tem raízes reais.

Exemplo 1

```
Enter coefficient a => 1
Enter coefficient b => 10
Enter coefficient c => 21
The roots are:  -3.0   -7.0
```

Exemplo 2

```
Enter coefficient a => 1
Enter coefficient b => 1
Enter coefficient c => 1
There are no real roots!
```

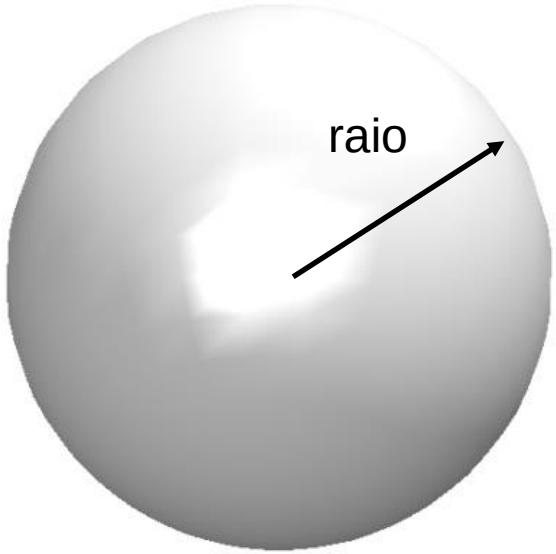
É muito comum na implementação de equações em Python, necessitarmos do valor de Pi. Em Python podemos chamar a constante Pi a partir do módulo *math*, como segue: *math.pi* .
Depois de importado o módulo *math*, podemos chamar o valor de Pi somente fazendo menção a ele, como mostrado abaixo para o cálculo da área.

```
area = math.pi*raio**2
```

Vamos usar um programa simples para aplicarmos a chamada do valor de Pi. Veremos um programa que calcula a área e o volume de uma esfera (*sphere1.py*), a partir da entrada do valor do raio. As equações da área e volume estão indicadas abaixo.

$$\text{area} = 4\pi(\text{raio})^2$$

$$\text{volume} = \frac{4}{3}\pi(\text{raio})^3$$



O pseudocódigo para o cálculo da área e do volume da esfera está mostrado abaixo. Testamos se o raio é maior ou igual a zero, para calcularmos a área e o volume, caso não seja, mostramos uma mensagem de erro. Recomenda-se que os pseudocódigos sejam apresentados de uma forma independente da linguagem de programação, no qual serão posteriormente implementados. Contudo, percebi que muitos alunos usam o pseudocódigo para implementar o código em Python. Assim, sempre que possível, deixarei as variáveis de agora em diante no padrão da linguagem Python, o que permite que os alunos copiem a linha com boa parte das equações diretamente do pseudocódigo, com exceção para as equações mais longas. No código abaixo, podemos copiar as linhas em vermelho, para implementar as equações em Python. **O "Fim" do pseudocódigo não precisa ser implementado em Python.**

```
Início
   Chama módulo math
   Leia (raio)
   Se (raio >0) então
         area = 4*math.pi*raio**2
         volume = (4/3)*math.pi*raio**3
         Escreva: "A area da esfera = ", area
         Escreva: "O volume da esfera = ", volume
   Senão
         Escreva: "Erro! Raio tem que maior que zero!"
Fim
```

O código *sphere1.py* está mostrado abaixo. As primeiras linhas são comentários, usados para informar o que o programa faz, bem como autor, data e outras informações relevantes. Não indicaremos mais este tipo de informação nos programas seguintes, por uma questão de espaço. Veja que usamos a sequência de escape *\n* no início da string, ou seja, será pulada uma linha, antes de mostrar a mensagem na tela. Analise com atenção cada linha de código, verificando se entendeu o que cada linha realiza. Lembre-se, a execução do programa segue de cima para baixo, com exceção do condicional *if*, que testa se o valor atribuído à variável *raio* é maior ou igual a zero.

```
# Program to calculate the area and the volume of a sphere using radius value.
# Author: Walter F. de Azevedo Jr.
# Date: March 15th 2019.
import math                                                    # Imports math module
raio = float(input("Enter radius => "))                        # Reads radius
if raio>=0:                                                    # Tests if radius >= 0
    area= 4*math.pi*raio**2                                    # Calculates area
    volume = (4/3)*math.pi*raio**3                             # Calculates volume
    print("\nArea of sphere = ", area)                         # Shows area
    print("\nVolume of sphere = ", volume)                      # Shows volume
else:
    print("\nError! Radius should be >= 0!")                     # Shows error message
```

Usaremos o programa *sphere1.py* para o cálculo da área e do volume para 3 situações. Inicialmente para raio igual 1 (exemplo 1), no segundo exemplo para raio igual 10,77 (exemplo 2) (não esqueça de usar ponto decimal para a entrada de dados), no último usamos um raio negativo, -1 (exemplo 3).

Exemplo 1

```
Enter radius => 1
Area of sphere =  12.566370614359172
Volume of sphere =  4.1887902047863905
```

Exemplo 2

```
Enter radius => 10.77
Area of sphere =  1457.6097700343018
Volume of sphere =  5232.819074423143
```

Exemplo 3

```
Enter radius => -1

Error! Radius should be >= 0!
```

Modificaremos o código visando à sua otimização, ou seja, melhorar o código para que este realize sua tarefa de forma mais precisa ou com um número menor de operações matemáticas. Foi sugerido por Steeb e o colaboradores (STEEB, Willi-Hans; HARDY, Yorick;HARDY, Alexandre; STOOP, Ruedi. **Problems and Solutions in Scientific Computing with C++ and Java Simulations**. Singapora: World Scientific Publishing Co.Pte. Ltd., 2004. p2.) que o cálculo do volume seja realizado com as seguintes equações:

$$\text{area} = 4\pi(\text{raio})^2 \qquad \text{volume} = \text{area} * \text{raio}/3$$

O cálculo do volume usa a informação já calculada da área, se considerarmos simplesmente o número de operações, a versão anterior tem quatro operações, a nova versão tem duas, o que reduz o tempo e execução. Isso se não levarmos em conta que a potenciação ao cubo tem uma complexidade maior que a multiplicação simples da versão otimizada, então a vantagem da segunda versão é ainda maior. Nem tente perceber a diferença de tempo de execução do programa atual. Com um código tão simples não será possível, mas imagine um programa onde o cálculo da esfera será repedido um milhão de vezes, o uso da segunda versão será mais eficiente. Como exercício, implemente o código *sphere2.py*, que usa as equações acima para o cálculo do volume. O resumo do programa está no próximo slide.

Cálculo da área e do volume da esfera (versão 2)

Programa: *sphere2.py*

Resumo
O programa *sphere2.py* calcula a área e o volume de uma esfera, a partir do raio. O cálculo da área é dado pela equação: *area = 4*math.pi*raio**2*. O volume é calculado a partir da área, com a seguinte equação: *volume = area*raio/3*. Tal equação do volume reduz o número de operações de cálculo, como destacado em: STEEB, Willi-Hans; HARDY, Yorick;HARDY, Alexandre; STOOP, Ruedi. **Problems and Solutions in Scientific Computing with C++ and Java Simulations**. Singapora: World Scientific Publishing Co.Pte. Ltd., 2004. p2. O programa usa o valor de Pi do módulo *math*.

Após a implementação do código em Python, teste o programa *sphere2.py* para os valores indicados na tabela abaixo, para confirmar se o seu código está funcionando de forma correta. Quando você consegue executar um código sem erros, mas o programa não gera os dados esperados, dizemos que temos um **erro de lógica**, que são os mais difíceis de consertar, pois podem indicar um erro do algoritmo usado para resolver o problema. Assim, sempre que possível, temos que testar os programas para valores conhecidos.

Raio	Área	Volume
1	12.566370614359172	4.188790204786905
0.001	1.256637061435917e-05	4.188790204786391e-09
10.77	1457.6097700343018	5232.819074423143
0	0.0	0.0
-1.0	Erro! Raio tem que maior que zero!	

Implemente em Python as seguintes equações da cinemática. O programa mostrará o resultado (variável dependente à esquerda) a partir de dados de entrada (variáveis independentes à esquerda). Para testar o seu código, use valores para os quais você calculou a resposta.

Equação
$v = v_0 + at$
$x = x_0 + v_0 t + \frac{1}{2} a t^2$
$v^2 = v_0^2 + 2a(x - x_0)$
$x = x_0 + \frac{1}{2}(v + v_0)t$
$x = x_0 + vt - \frac{1}{2} a t^2$

-BRESSERT, Eli. **SciPy and NumPy**. Sebastopol: O'Reilly Media, Inc., 2013. 56 p.
-DAWSON, Michael. **Python Programming, for the absolute beginner.** 3ed. Boston: Course Technology, 2010. 455 p.
-HAL, Tim, STACEY, J-P. **Python 3 for Absolute Beginners.** Springer-Verlag New York, 2009. 295 p.
-HETLAND, Magnus Lie. **Python Algorithms. Mastering Basic Algorithms in the Python Language.** Nova York: Springer Science+Business Media LLC, 2010. 316 p.
-IDRIS, Ivan. **NumPy 1.5. An action-packed guide dor the easy-to-use, high performance, Python based free open source NumPy mathematical library using real-world examples. Beginner's Guide.** Birmingham: Packt Publishing Ltd., 2011. 212 p.
-KIUSALAAS, Jaan. **Numerical Methods in Engineering with Python.** 2ed. Nova York: Cambridge University Press, 2010. 422 p.
-LANDAU, Rubin H. **A First Course in Scientific Computing: Symbolic, Graphic, and Numeric Modeling Using Maple, Java, Mathematica, and Fortran90**. Princeton: Princeton University Press, 2005. 481p.
-LANDAU, Rubin H., PÁEZ, Manuel José, BORDEIANU, Cristian C. **A Survey of Computational Physics. Introductory Computational Physics.** Princeton: Princeton University Press, 2008. 658 p.
-LUTZ, Mark. **Programming Python.** 4ed. Sebastopol: O'Reilly Media, Inc., 2010. 1584 p.
-TOSI, Sandro. **Matplotlib for Python Developers.** Birmingham: Packt Publishing Ltd., 2009. 293 p.