Python

Prof.: Fábio Junior Alves

faguanil@gmail.com

python

Bibliografias Utilizadas

python

Sites Consultados

- http://www.async.com.br/projects/python/pnp/
- http://under-linux.org/curso-de-python-gratuito-no-under-linux-1266/
- http://www.python.org.br/wiki/DocumentacaoPython?action=AttachFile&do=get&target=python24.pdf
- http://docs.python.org/tutorial/

python

Comunidades

- http://www.python.org
- http://www.pythonbrasil.com.br

python

A Linguagem Python

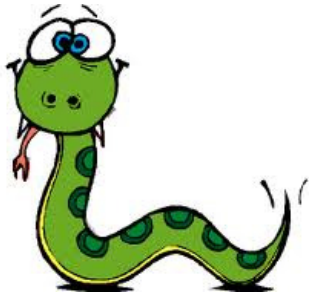
- Foi lançada por **Guido Van Rossum** (Holandês) em 1991;

python

A Linguagem Python

- O nome da linguagem origina-se da série humorística britânica Monty Python's Flying Circus, do grupo humorístico britânico Monty Python.

- Embora muitas pessoas façam associação com o réptil do mesmo nome (em português, píton ou pitão).



python

A Linguagem Python

- Python é uma linguagem de programação poderosa e fácil de aprender;
- Possui estruturas de dados de alto nível;
- Eficiente abordagem da programação orientada a objetos;
- Sua sintaxe e tipagem dinâmica juntamente com seu interpretador nativo fazem dela a linguagem ideal para scripting e desenvolvimento rápido de aplicações em diversas áreas sob várias plataformas.

python

A Linguagem Python

- Python vem crescendo em várias áreas da computação, como:
  - Inteligência Artificial;
  - Banco de Dados;
  - Biotecnologia;
  - Animação 3D;
  - Aplicativos Móveis;
  - Plataformas Web;
  - Etc.

python

Quem usa Python

python

Python é Utilizada lá Fora

Google

"Python tem sido uma parte importante do Google desde o início, e permanece assim conforme o sistema cresce e evolui. Hoje, dezenas de engenheiros do Google usam Python, e estamos procurando por mais pessoas com conhecimento nessa linguagem." Peter Norvig, diretor de qualidade de busca do Google Inc.

`http://www.pythonbrasil.com.br/moin.cgi/PythonNoGoogle`

"[...] nossa filosofia geral é 'Python onde podemos, C++ onde somos obrigados.' Python não é definitivamente apenas um pequeno pedaço, nem é usado apenas para tarefas 'de scripting'; se ficássemos sem todo nosso código Python de uma hora para outra, nossa poderosa infra-estrutura que tem sido descrita como 'a arma secreta do Google' ficaria abalada." Alex Martelli, Líder Técnico, Sistemas de Produção, Google Inc.

python

Python é Utilizada lá Fora

Youtube

"Python é rápido o suficiente para o nosso site e permite que nós produzamos características que pode ser mantidas em tempo recorde, com um mínimo de desenvolvedores." Cuong Do, Arquiteto de Software, YouTube.com

`http://www.python.org/about/quotes/`

"O YouTube (uma das propriedades mais valiosas do Google) é essencialmente todo Python [...]." Alex Martelli, Líder Técnico, Sistemas de Produção, Google Inc.

python

Python é Utilizada lá Fora

Industrial Light and Magic

"Python desempenha um papel-chave no nosso esquema de produção. Sem ela um projeto do tamanho de 'Guerras nas Estrelas: Episódio II' teria sido muito difícil de sair. Da renderização da multidão ao processamento em lote e a composição, Python mantém todas as partes unidas." Tommy Burnette, Diretor Técnico Sênior, ILM

"Quando me transferi [...], todos os membros da equipe [...] me acompanharam — menos Tommy. Ele [...] fora contratado pela Industrial Light & Magic[...]. Tommy se tornou um destacado programador de linguagem **Python, por sorte dele a linguagem escolhida pela empresa de Lucas**. E sorte mesmo é quando o preparo se une à oportunidade. " Randy Pausch, A Lição Final

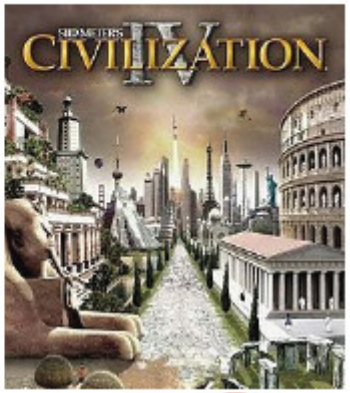
python

Python é Utilizada lá Fora

redhat

Inkscape

NOKIA

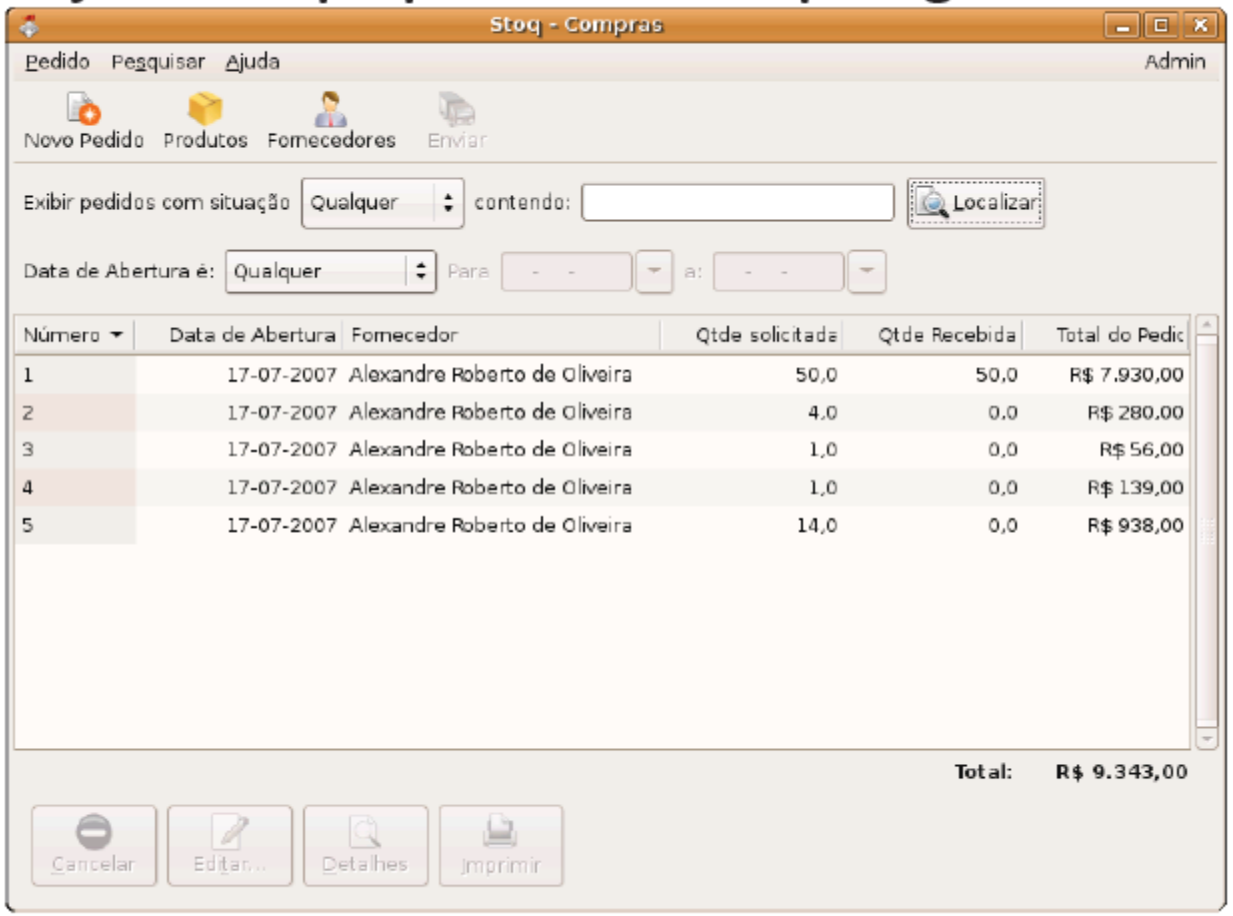


OpenOffice.org

python

Python é Utilizado aqui Dentro

Async - Stoq: aplicativos livres para gestão comercial

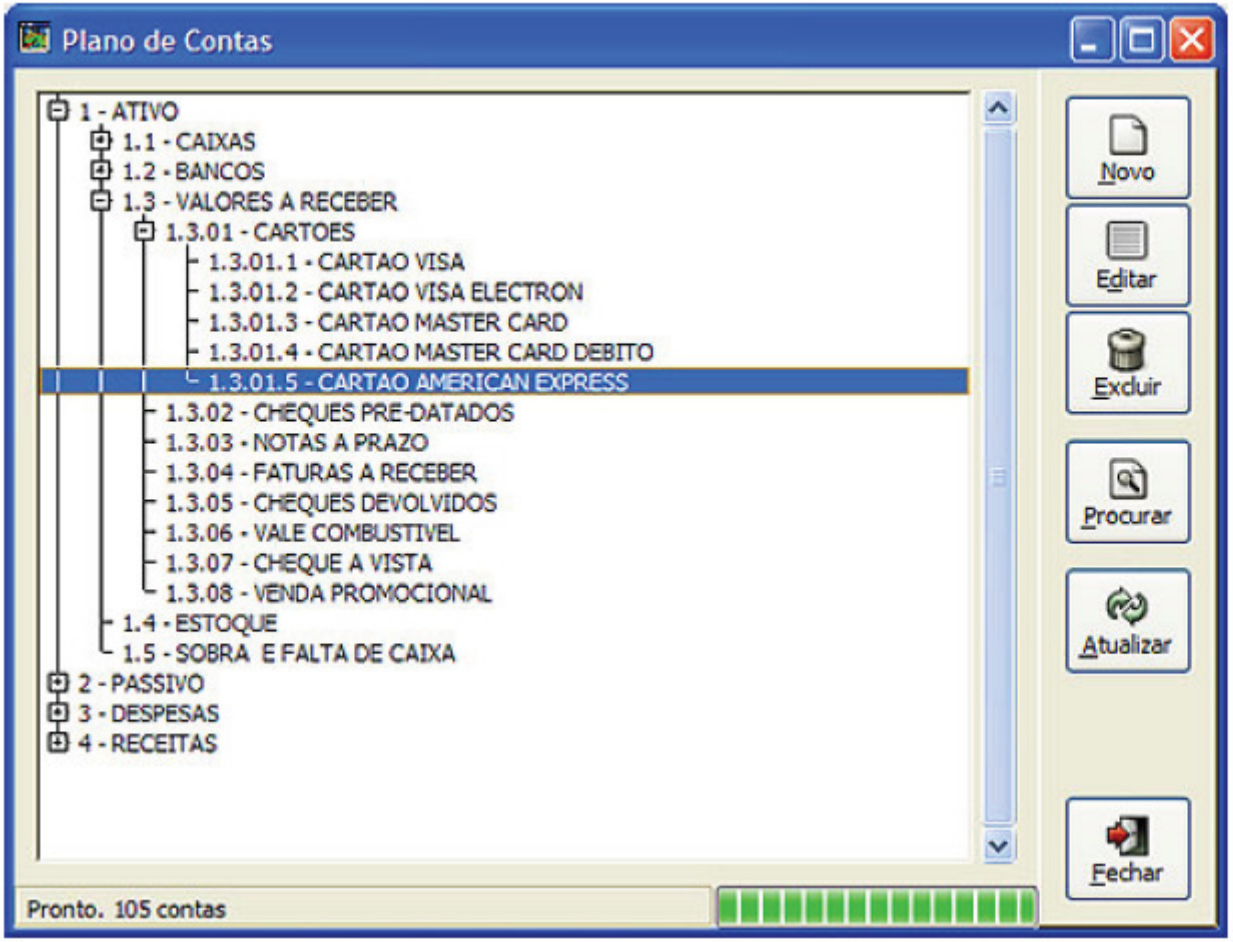


http://www.async.com.br

python

Python é Utilizado aqui Dentro

LZT - AutoSystem: automação de postos de combustível



python

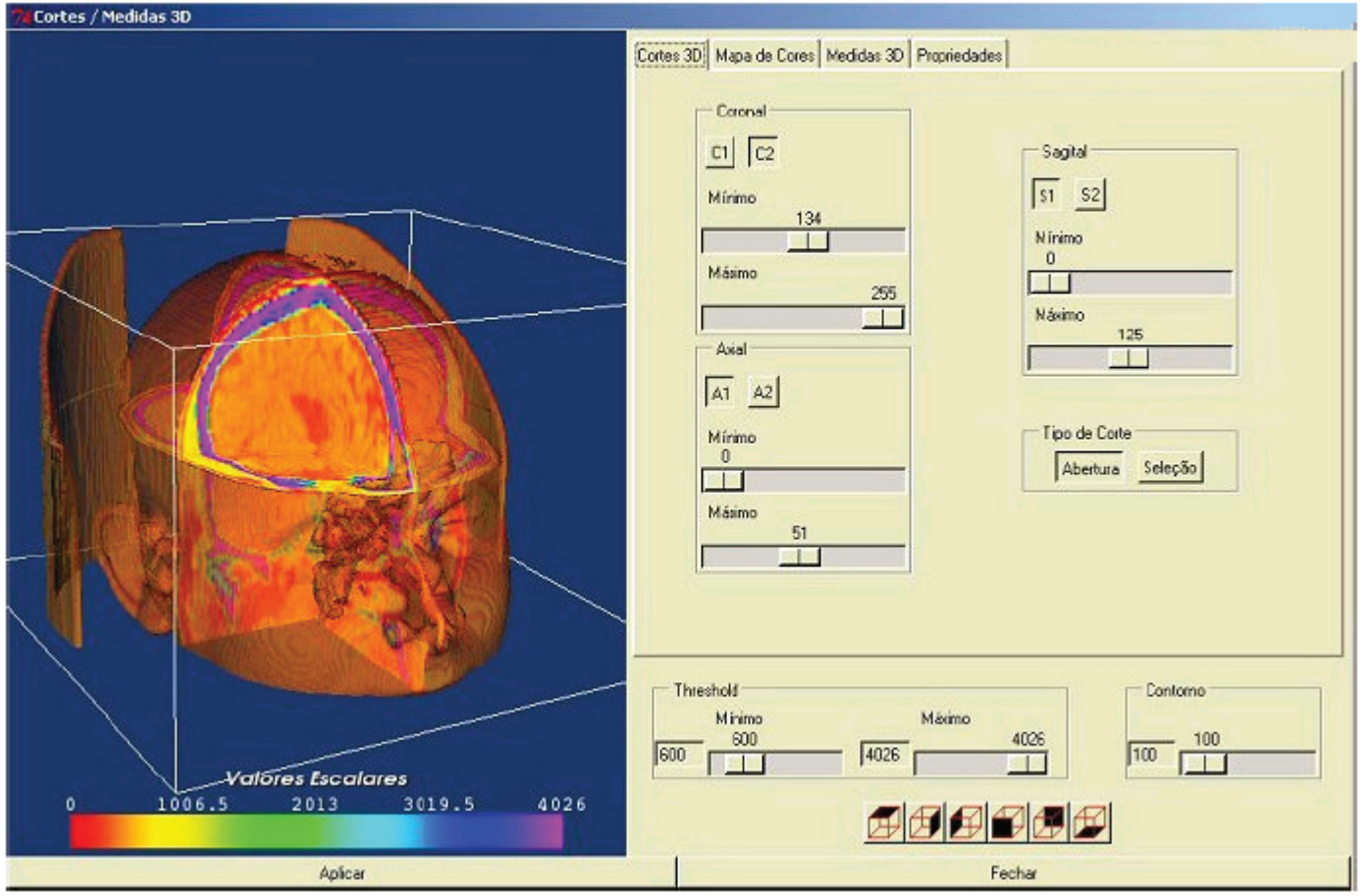
Python é Utilizado aqui Dentro

INdT - Instituto Nokia de Tecnologia

python

Python é Utilizado aqui Dentro

InVesalius - Reconstrução 3D de imagens médicas



python

Python é Utilizado na Web

python

Python é Utilizado na Web

CIA

python

Python é Utilizado na Web

IDG Brasil

python

Python é Utilizado na Web

python

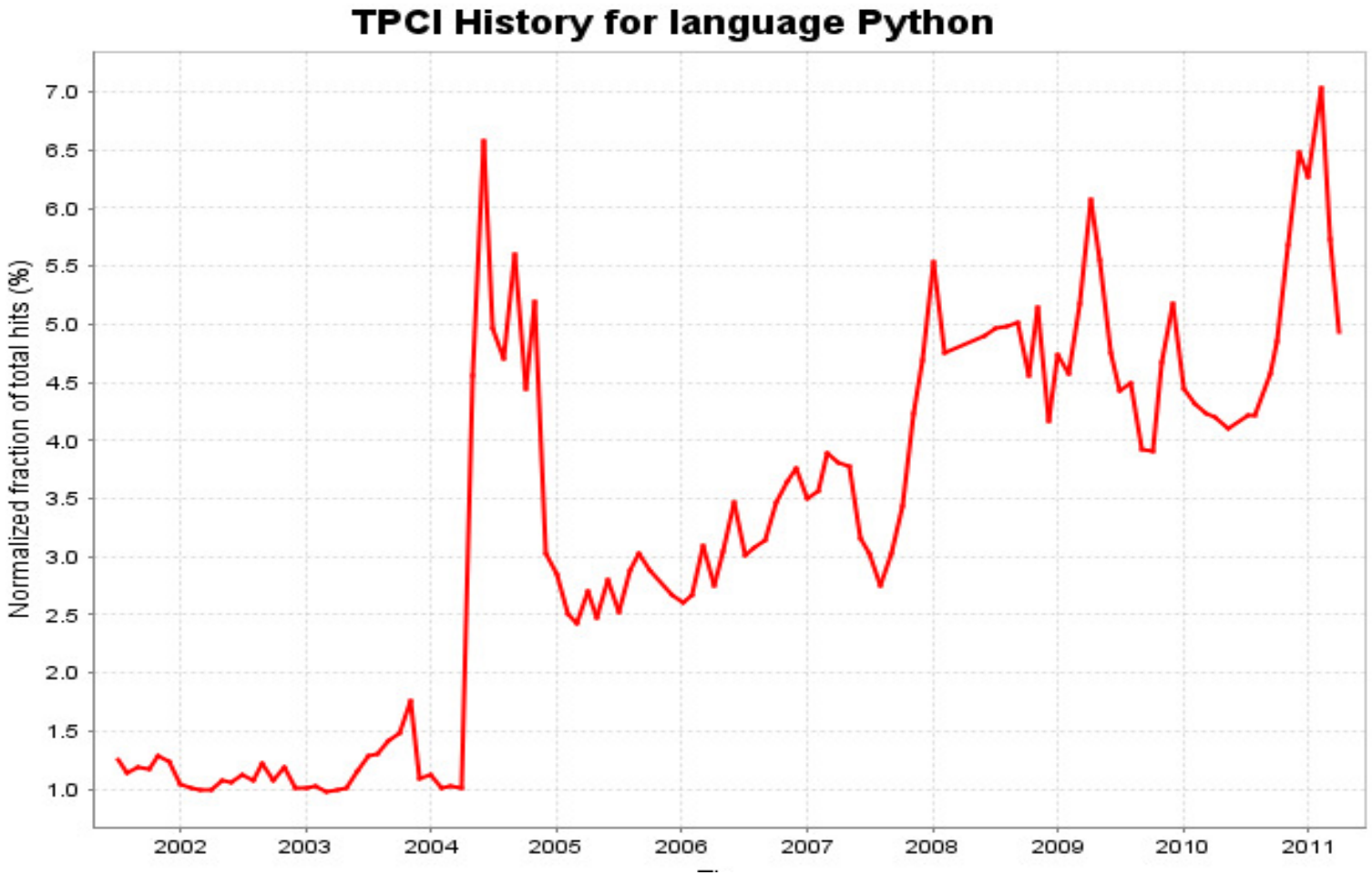
Popularidade das Linguagens de Programação

Position Apr 2011	Position Apr 2010	Delta in Position	Programming Language	Ratings Apr 2011	Delta Apr 2010	Status
1	2	↑	Java	19.043%	+0.99%	A
2	1	↓	C	16.162%	-1.90%	A
3	3	=	C++	9.225%	-0.48%	A
4	6	↑↑	C#	7.185%	+2.75%	A
5	4	↓	PHP	6.584%	-3.08%	A
6	7	↑	Python	4.931%	+0.73%	A
7	5	↓↓	(Visual) Basic	4.682%	-1.71%	A
8	11	↑↑↑	Objective-C	4.386%	+2.10%	A
9	8	↓	Perl	1.991%	-1.56%	A
10	10	=	JavaScript	1.513%	-0.96%	A
11	12	↑	Ruby	1.482%	-0.74%	A
12	20	↑↑↑↑↑↑↑	Lua	1.035%	+0.51%	A
13	9	↓↓↓↓	Delphi	1.034%	-1.68%	A
14	-	=	Assembly	0.967%	0.00%	A
15	23	↑↑↑↑↑↑↑	Lisp	0.934%	+0.45%	A
16	32	↑↑↑↑↑↑↑↑↑	Ada	0.768%	+0.41%	A
17	16	↓	Pascal	0.713%	+0.06%	A
18	21	↑↑↑	Transact-SQL	0.583%	+0.08%	B
19	-	=	Scheme	0.581%	0.00%	B
20	15	↓↓↓↓↓	Go	0.557%	-0.15%	A--

Fonte: http://www.tiobe.com/index.php/content/paperinfo/tpci/index.html

python

Popularidade da Linguagens Python



Fonte: http://www.tiobe.com/index.php/content/paperinfo/tpci/index.html

python

Características

- **Simplicidade:** Python é uma linguagem muito simples.

- **Facilidade de aprender:** Como você verá aprender a programar com Python é extremamente fácil. Como já mencionado, sua sintaxe é muito simples.

- **Software livre:** Python é software livre. Isto significa que você pode distribuir cópias, ter acesso ao código fonte, alterar, modificar, redistribuir e utilizar pedaços dele em outros programas.

python

Características

- **Portabilidade:** Python é portável para diversas arquiteturas, isto significa que seu programa escrito para Linux pode facilmente rodar em Linux, Windows, FreeBSD, Macintosh, Solaris, OS/2, etc.

- **Linguagem interpretada :** O código de um programa Python é lido e executado pelo interpretador Python sem necessitar de compilação.

python

Características

- **Orientação a objetos:** Python suporta tanto a programação orientada a função quanto a programação orientada a objetos.

- Etc.

python

Tipos de Alto Nível

- Além dos tipos básicos (inteiros, números de ponto flutuante, booleanos), alguns tipos pré-determinados em Python merecem atenção especial:

  - Listas;
  - Tuplas;
  - Strings ;
  - Dicionários;
  - Arquivos;
  - Classes e Instâncias .

python

Classificação das Linguagem

- Linguagens de programação são freqüentemente classificadas como compiladas ou interpretadas.

python

Linguagens Compiladas

- Nas compiladas, o texto (ou código-fonte) do programa é lido por um programa chamado compilador, que cria um arquivo binário, executável diretamente pelo hardware da plataforma-alvo.

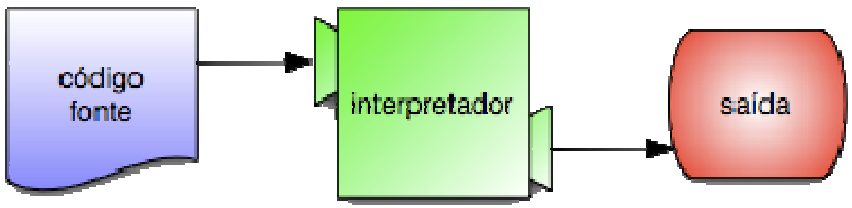
- Exemplos deste tipo de linguagem são C ou Fortran.

python

Linguagens Interpretadas

- Programas escritos em linguagens interpretadas não são convertidos em um arquivo executável. Eles são executados utilizando outro programa, o interpretador, que lê o código-fonte e o interpreta diretamente, durante a sua execução.

- Exemplos de linguagem interpretada incluem o BASIC tradicional, Perl e **Python**.



python

Comparando com outras Linguagens

- **Problema:**

  – Faça um algoritmo que leia uma seqüência de números inteiros e imprima o maior valor. Último Número = 0.

python

Comparando com outras Linguagens

```
uses winCrt;
var numero, maior : integer;
begin
    write ('Digite um número inteiro:');
    readln (numero);
    maior:= 0;
    while (numero <> 0) do begin
        if (numero > maior)then begin
            maior:= numero;
        end;
        write ('Digite um número inteiro:');
        readln (numero);
    end;
    write ('O maior valor:', maior);
end.
```

python

Comparando com outras Linguagens

```
import java.util.Scanner;
public class While1 {
	public static void main(String[] args) {
	 int numero = 0;
	 int maior = 0 ;
	 do {
		Scanner in = new Scanner(System.in);
		System.out.print("Digite um numero inteiro ou digite 0 (zero) para encerrar: ");
		numero = in.nextInt();
		if (numero > maior) {
			maior = numero;
		}


	 } while (numero != 0);
	 System.out.print("O Maior Valor é: ");
	 System.out.println(maior);
	}

}
```

python

Comparando com outras Linguagens

```
num = input('Digite um número inteiro:')
maior = 0
while num != 0:
    if num > maior:
        maior = num

    num = input('Digite um número inteiro:')

print 'O maior valor é:',maior
```

python

Diferença com as outras Linguagens

- Esqueça declarações de tipos de variáveis;
- Esqueça begin e end;
- Esqueça { e };
- Se você já era organizado, não sofrerá!
- A identação é obrigatória!

python

Controle de Bloco por Identação

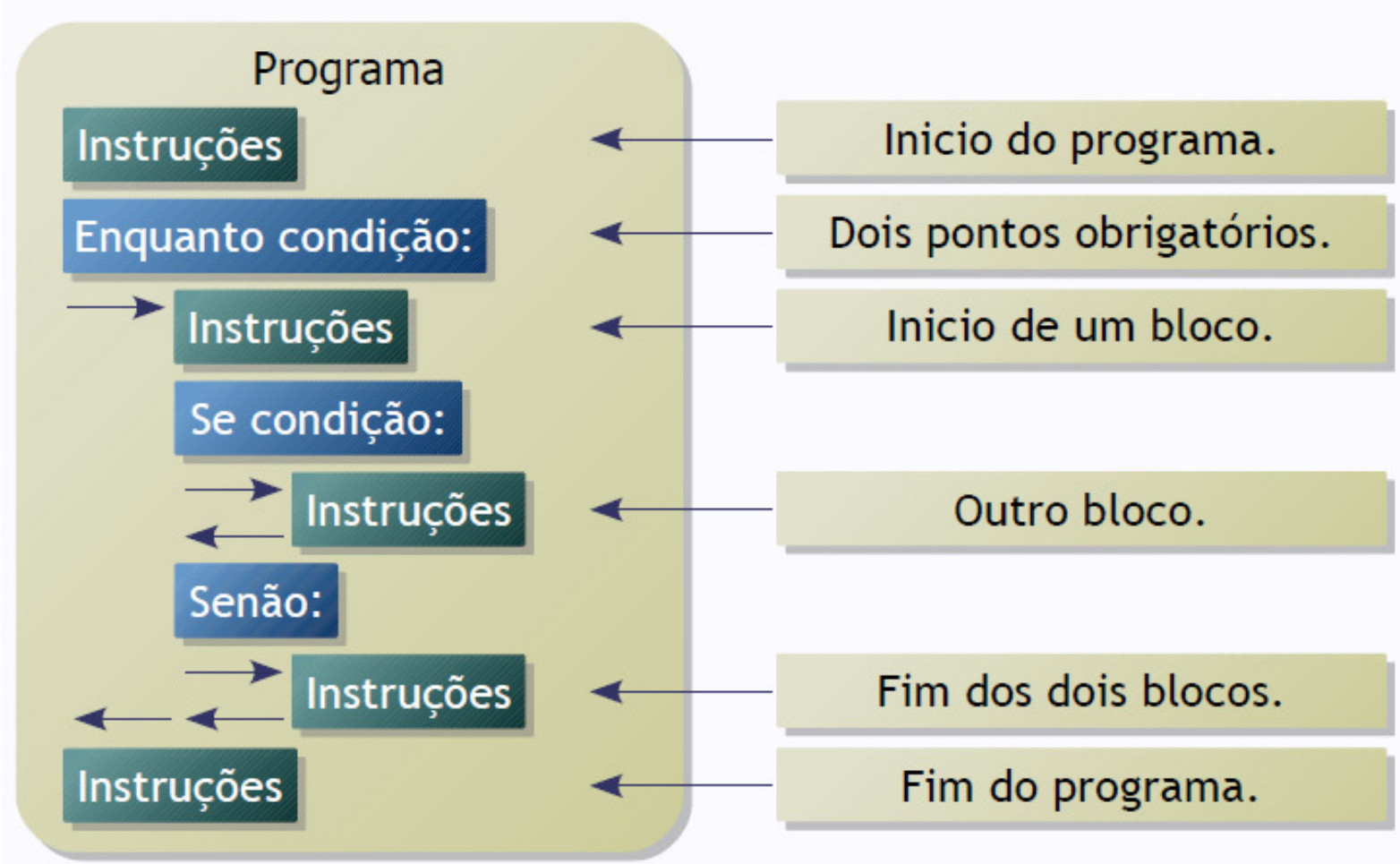
- Na maior parte das linguagens, há instruções ou símbolos específicos que delimitam blocos de código - os blocos que compõem o conteúdo de um laço ou expressão condicional, por exemplo. Em C:

```
if (a < 0) {
    /* bloco de código */
}
```

- Em Python, blocos de código são demarcados apenas por espaços formando uma identação visual:

python

Controle de Bloco por Identação



python

Controle de Bloco por Identação

- **Exemplo:**

```
print "O valor de a é"
if a == 0:
    print "zero"
else:
    print a
```

python

Preparando o Ambiente

- Antes de começarmos a desenvolver qualquer tipo de algoritmo utilizando a linguagem python, é necessário que seja instalado um interpretador que é um programa que permite que você execute todos os comandos da linguagem.

python

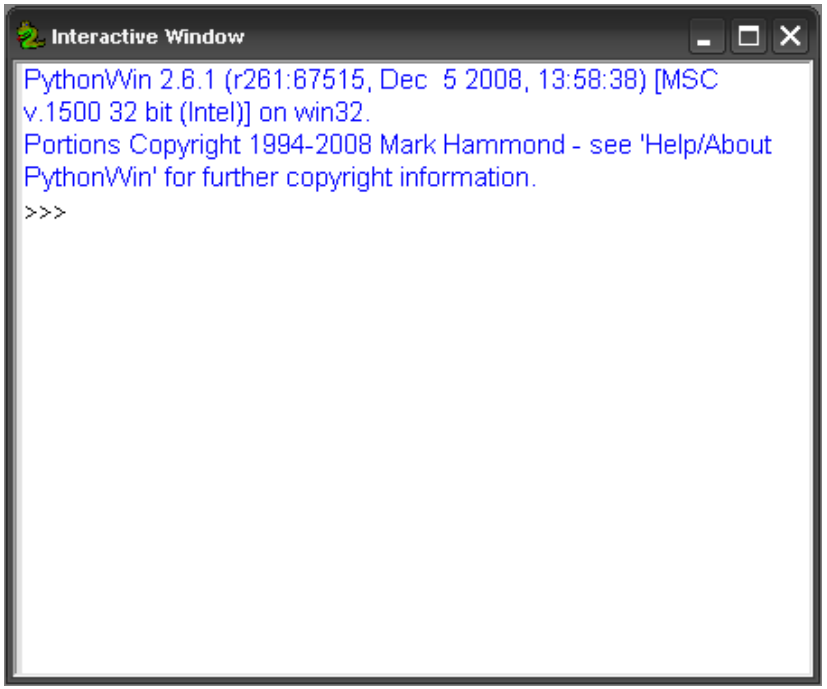
Preparando o Ambiente

- Para fazer o download do python basta entrar no site http://www.python.org/download/.

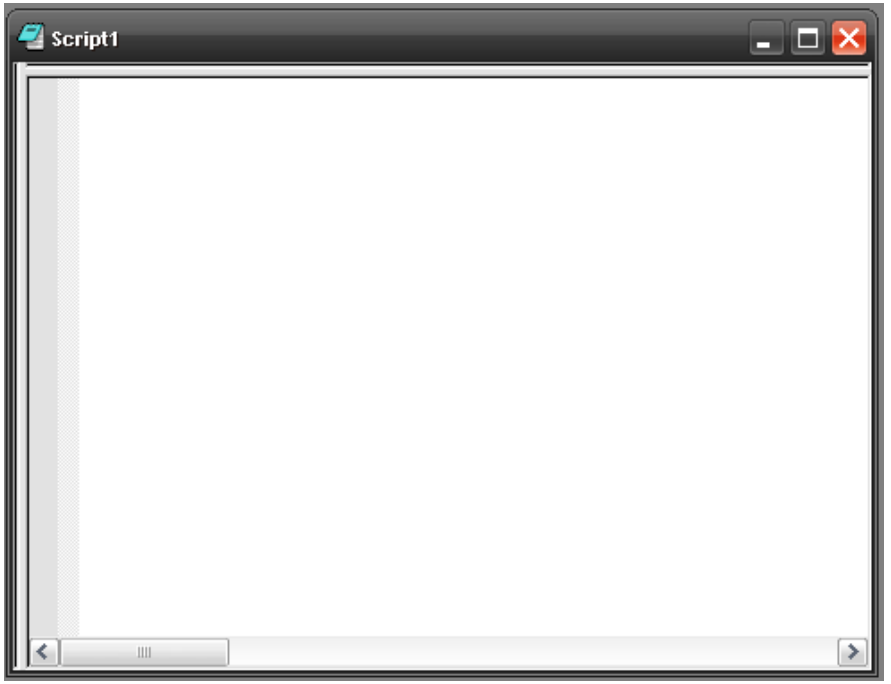
python

Preparando o Ambiente

- Existem duas maneiras de utilizar o Python:



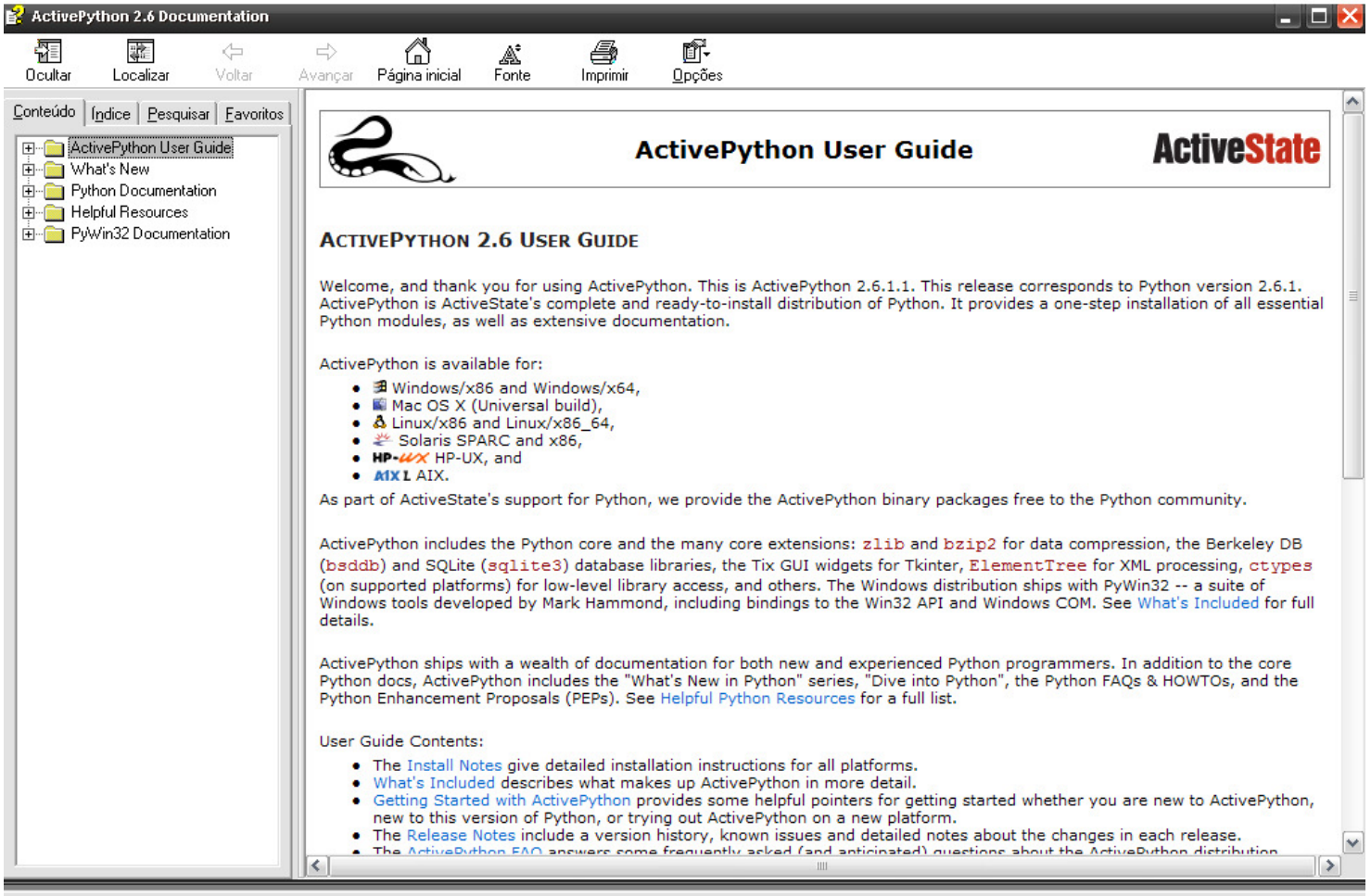
O Interpretador



O Editor de Texto

python

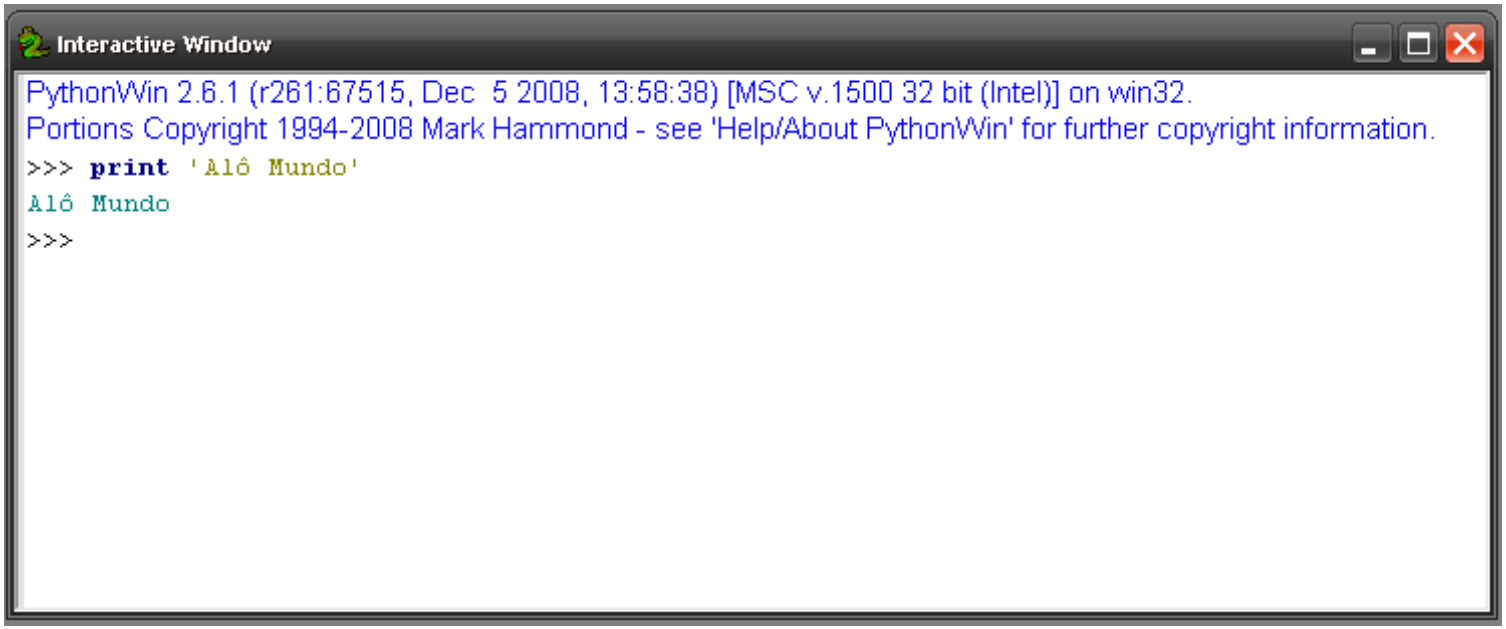
Ajuda do Python



python

Utilizando o Interpretador

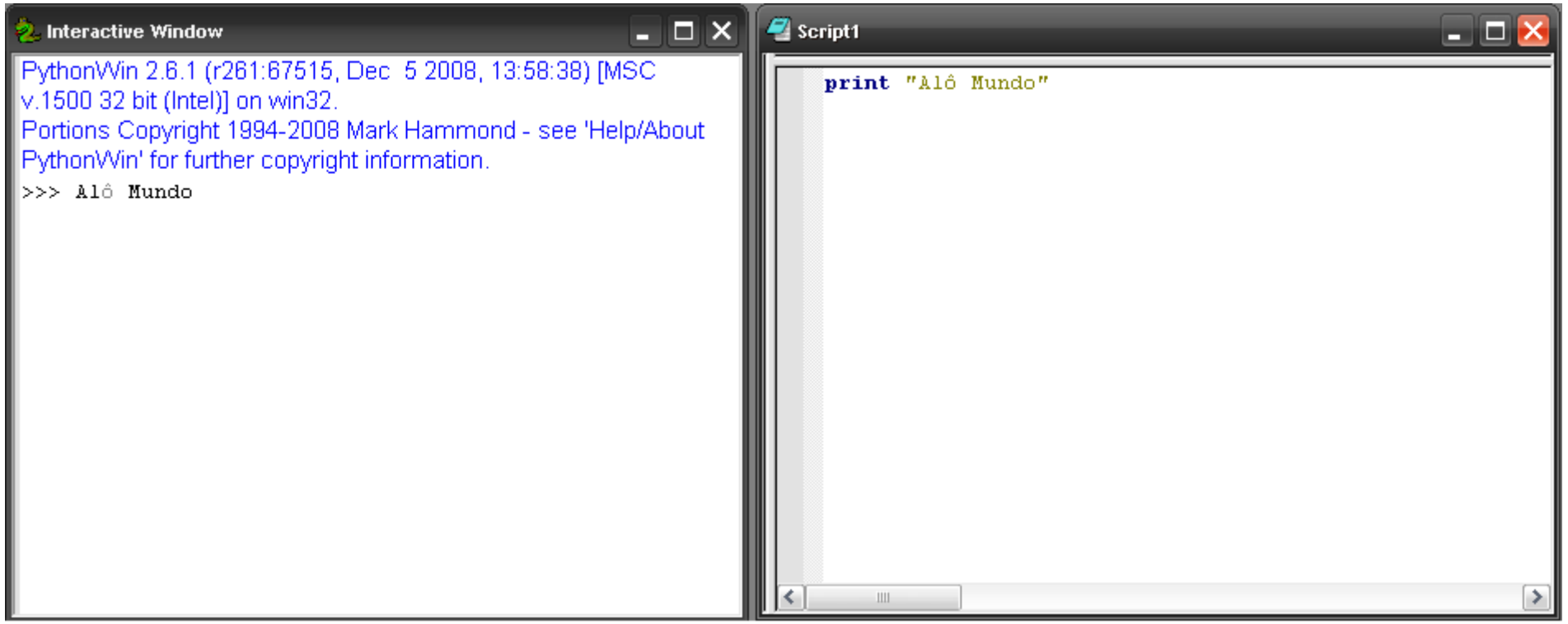
- Digite o comando **print ' Alô Mundo!'** e pressione **ENTER**:



python

Utilizando o Editor de Texto

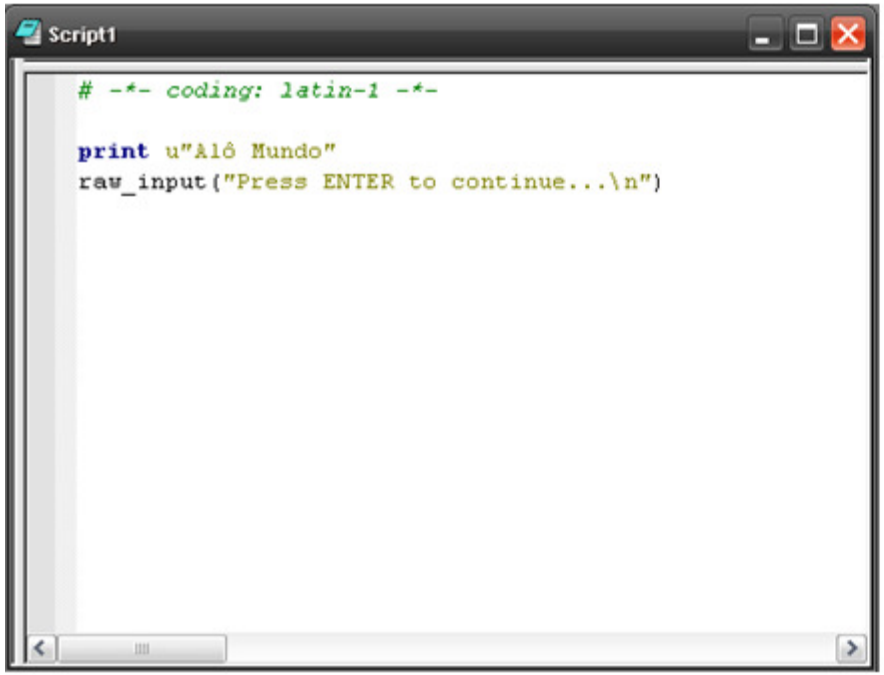
- Digite o comando **print ' Alô Mundo!'** salve o script e clique em F5 para executar:

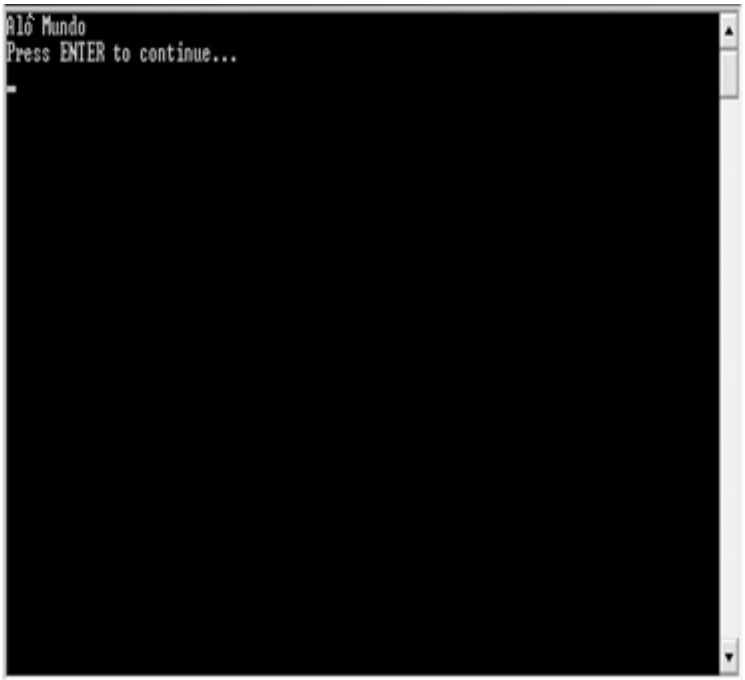


python

Execução "Direta" (Windows)

- Codificação do Arquivo:





A letra “U” é utilizada para informar ao interpretador do python que ele deve utilizar unicode para codificar/decodificar uma string.

python

Algumas Codificações Aceitáveis:

- # -*- coding: latin-1 -*-
- # -*- coding: iso-8859-1 -*-
- # This Python file uses the following encoding: utf-8
- # coding: latin-1.

python

Usando Python como uma Calculadora

```
>>> 2 + 5
7
>>> 2.5 + 3
5.5
>>> 4 * 5
20
>>> 3 * -4
-12
>>> 2 * 1.5
3.0
>>> 5 / 2
2
>>> 5.0 / 2
2.5
```

```
>>> 5 % 2
1
>>> 5.0 % 2
1.0
>>> 3 / 5
0
>>> 3.0 / 5
0.5999999999999998
>>> 3 % 5
3
>>> 2 ** 2
4
>>> pow(2,3)
8
```

python

Comentário

- O símbolo para comentários em Python é o caráter sustenido (#).

```
>>> a = 1 # atribuindo um valor a uma variavel
```

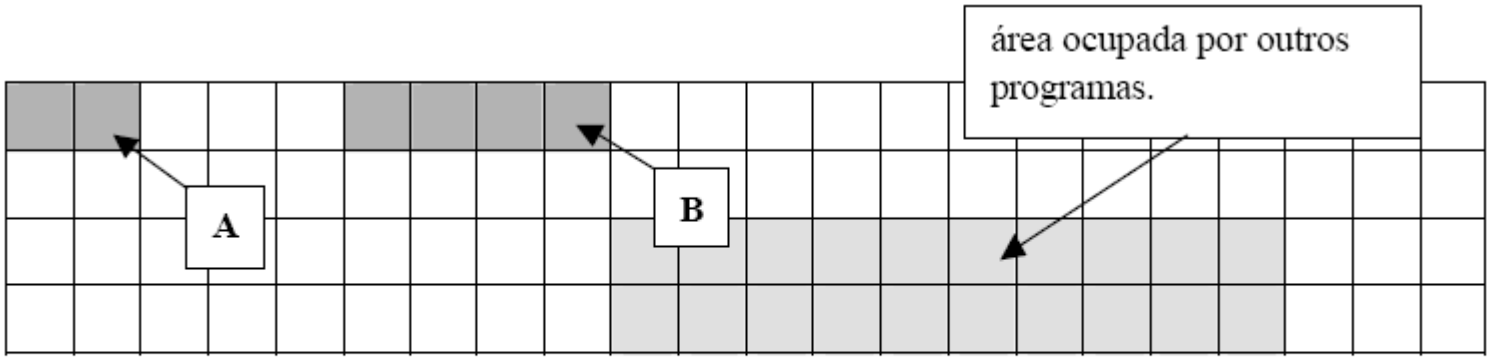
python

Variáveis

- Ao utilizar variáveis em Python não é necessário informar o tipo desta variável como fazemos em outras linguagens. Por exemplo, em C antes de utilizar um numero inteiro devemos primeiro criar a variável.

- Os dados são armazenados nas variáveis. Elas são apenas uma parte da memória do seu computador onde você armazena informações temporariamente.

Variáveis



Para armazenar o valor inteiro A= 1, necessita-se de 2 bytes (1 inteiro = 2 bytes na memória*);

Para armazenar o valor real B= 1, necessita-se de 4 bytes (1 real = 4 bytes na memória*);

python

Variáveis

- Regras para criar variáveis em Python:
- O primeiro caracter deve ser uma letra (maiúscula ou minúscula) ou o caracter de sublinhado (_);
- O resto do nome pode conter letras (maiúsculas ou minúsculas), sublinhado (_) ou dígitos (0-9);
- Exemplos de nomes de variáveis corretos são: nome, _idade, nome23, d3b1;
- Exemplos de nomes de variáveis inválidos são: 2things, media aritmetica.

python

Exemplo - Variáveis

```
>>> valor1=2
>>> print valor1
2
>>> valor1='Boa tarde'
>>> print valor1
Boa tarde
>>>
```

python

Variáveis

- O Python é case sensitive, isto é, ele percebe a diferença entre letras maiúsculas e minúsculas. Por isso tome muito cuidado ao criar uma variável com letras maiúsculas.

- Exemplo: “numero” e “Numero” não são a mesma coisa. Observe o primeiro **n**, no primeiro nome é minúsculo e no segundo é maiúsculo.

python

Exemplo – Variáveis são Case-Sensitive

```
numero = 5
Numero = 6

print 'O valor da variável numero:', numero

print 'Já o valor da variável Numero:', Numero
```

python

Tipagem Dinâmica

- Diferentemente de outras linguagens, com Python não precisamos declarar variáveis, nem seus tipos.

- Python possui o que é conhecido como tipagem dinâmica, ou seja, o tipo ao qual a variável está associada pode variar durante a execução do programa.

python

Exemplo - Tipagem Dinâmica

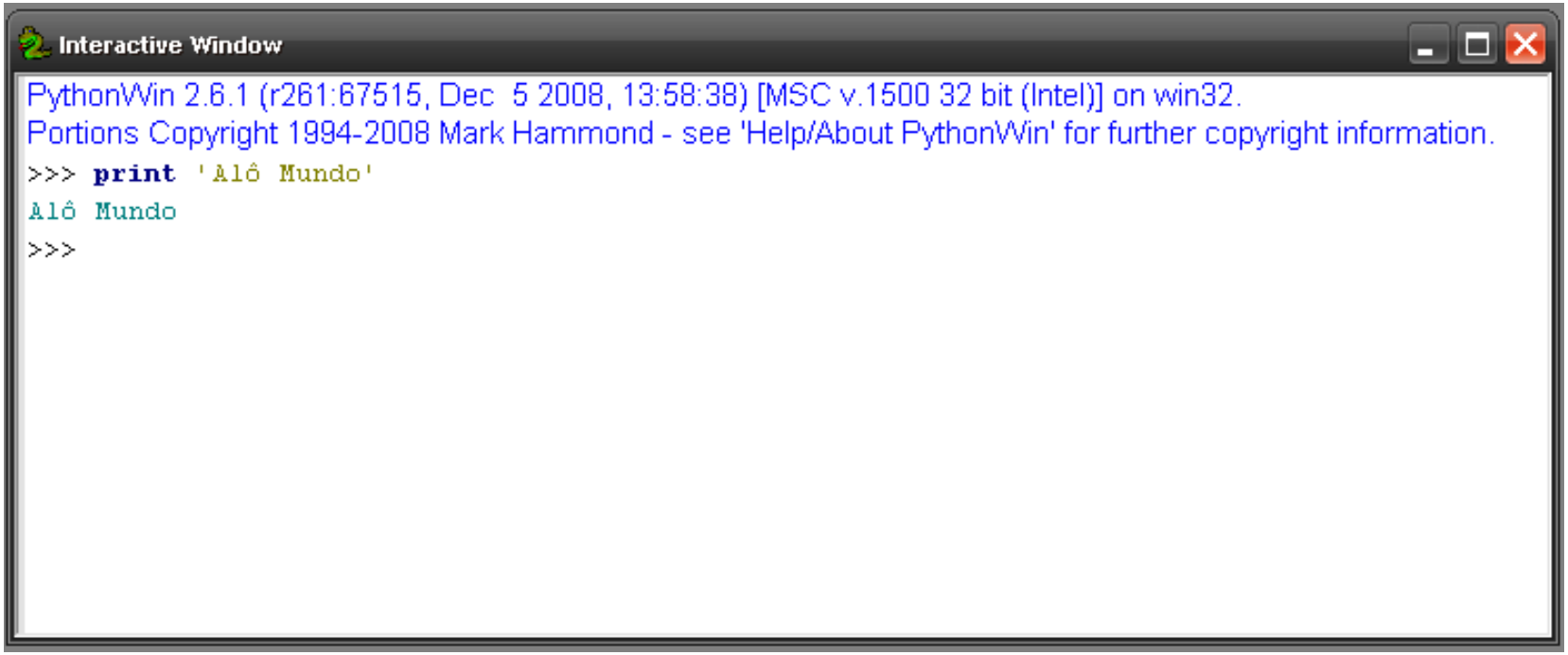
```
>>> a = 1
>>> print type(a)
<type 'int'>
>>> a = 'hum'
>>> print type(a)
<type 'str'>
```

Tipagem dinâmica, além de reduzir a quantidade de planejamento prévio e digitação para escrever um programa, é um mecanismo importante para garantir a simplicidade e flexibilidade das funções Python.

python

Comandos de Entrada e Saída de Dados

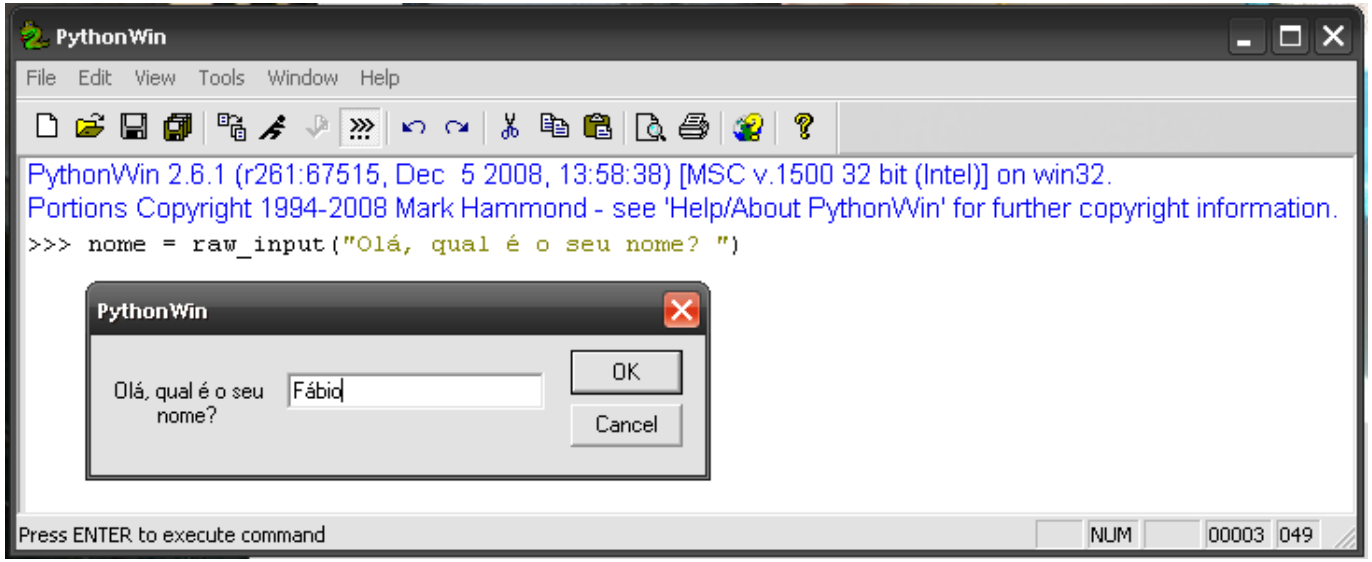
- **Print**: comando utilizado para escrever na tela. Este comando é simples de usar, basta escrever a frase entre aspas: **print 'Bom Dia!'**



python

Comandos de Entrada e Saída de Dados

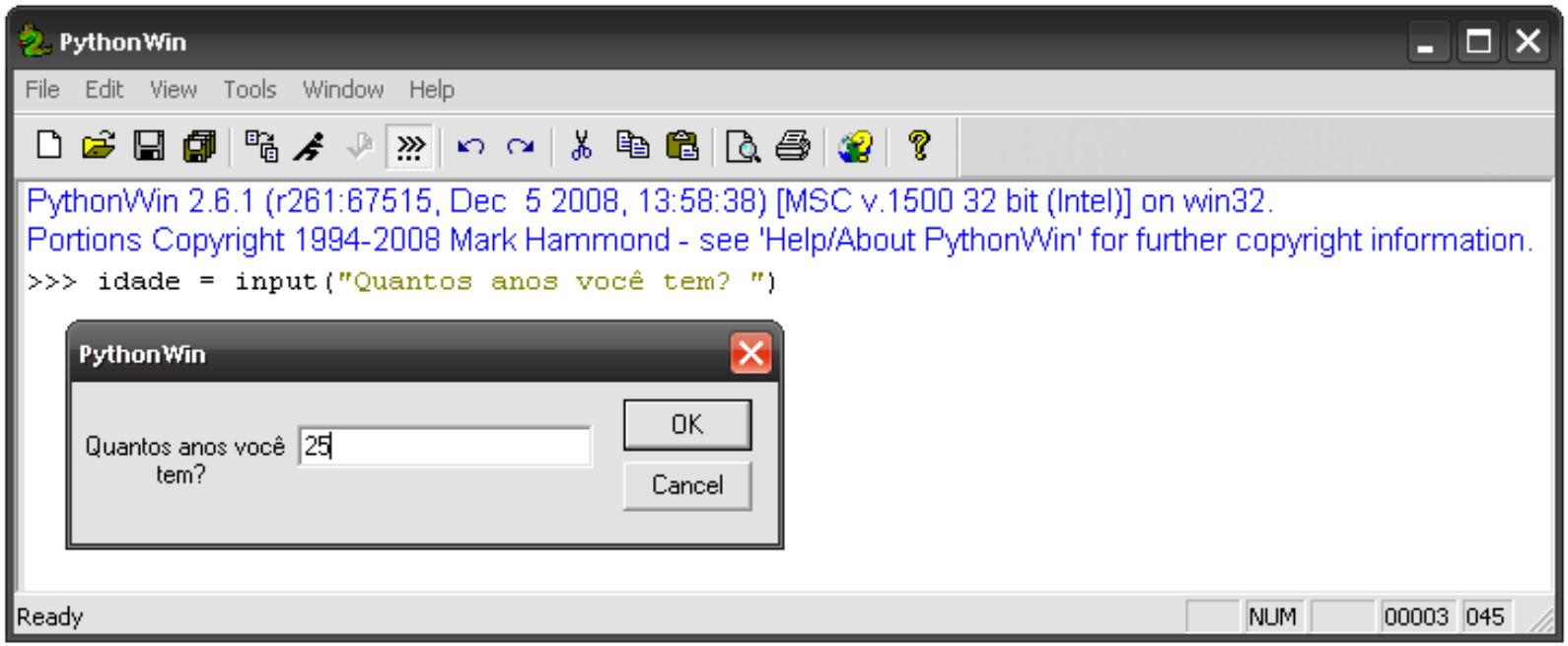
- **Raw_input**: comando que lê texto digitado no teclado. Este texto precisa ser guardado em algum lugar, para armazená-lo podemos usar uma **variável**, que é um pequeno espaço de memória reservado e que pode ser usado mais tarde.



python

Comandos de Entrada e Saída de Dados

- Enquanto o comando **raw_input** pode ser qualquer tipo de dados (letras, números, binários), o comando **input**, serve apenas para armazenar números.

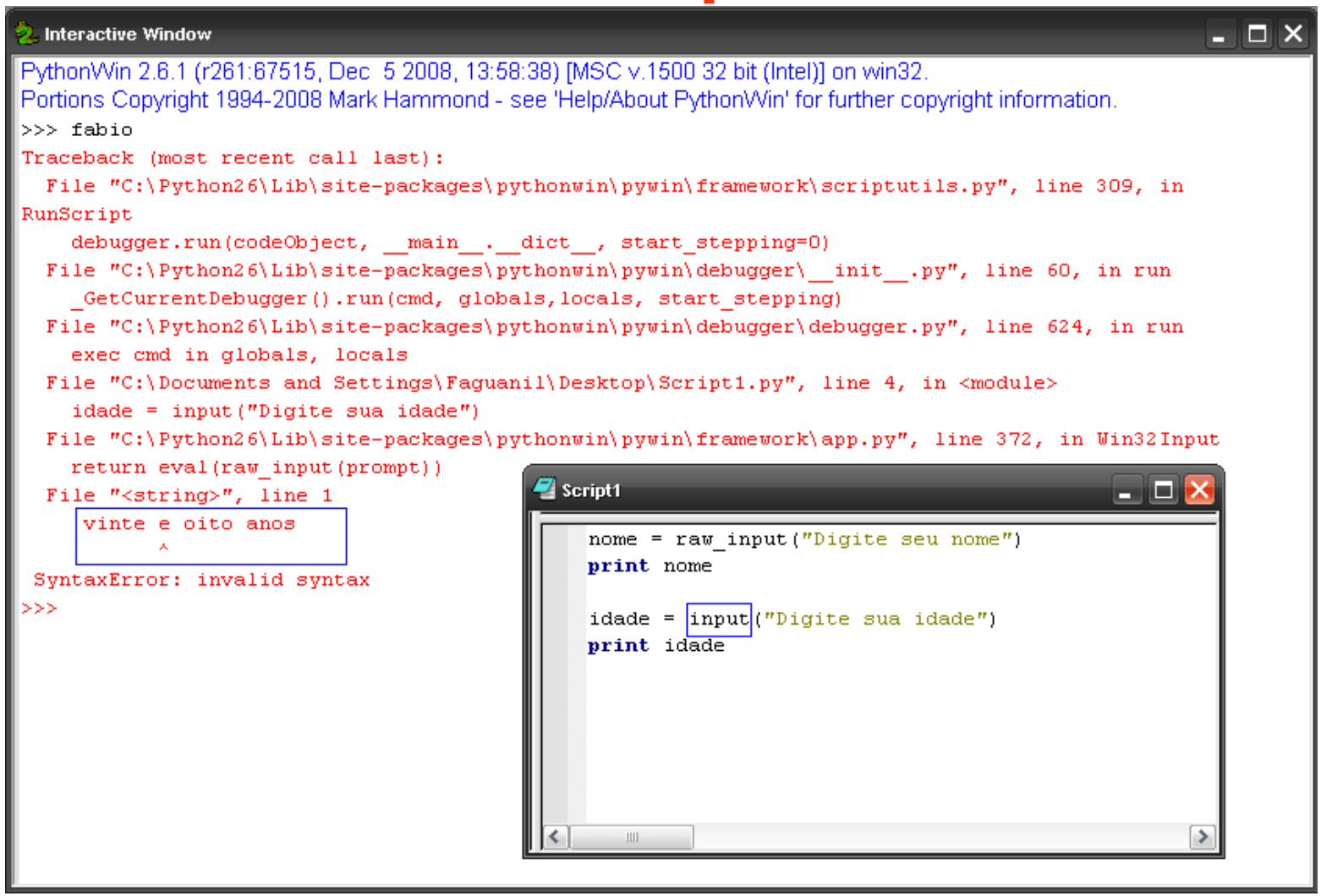


python

Tratando Erros na Entrada de Dados

- Erros são humanos e sempre ocorrem ao escrevermos programas, sejam simples ou complexos, o modo de como as linguagens de programação nos retornam estes erros é que há mudança.

Tratando Erros na Entrada de Dados
Exemplo



python

Tratando Erros na Entrada de Dados

- Em Python, os erros podem serem tratados muito elegantemente através das **Exceções**.

- A exceção é um recurso de linguagens de programação modernas que serve para informar que uma condição incomum ocorreu.

- Embora existam outras aplicações, em geral comunicam-se através de exceções erros ou problemas que ocorrem durante a execução de um programa.

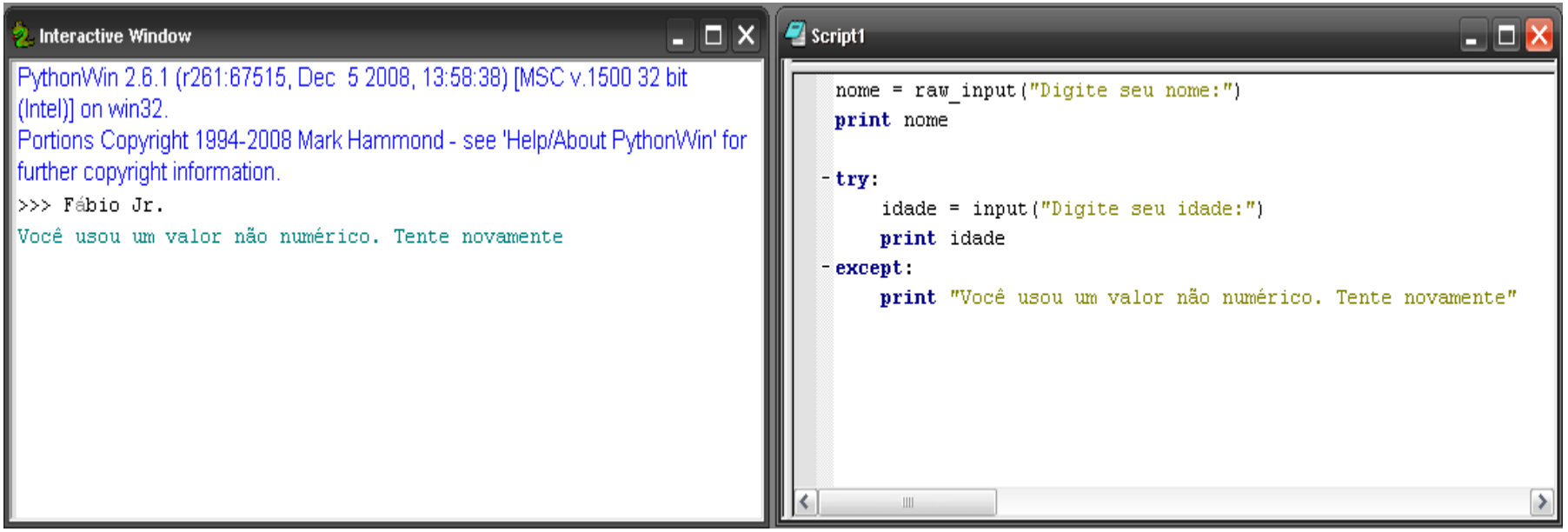
python

Tratando Erros na Entrada de Dados

- **Sintaxe:**

```
try:
        primeiro tenta fazer isso
        e tudo que esteja neste alinhamento (indentação, lembra?)
        isso também
        e isso
except:
        se qualquer linha dentro do bloco try der erro,
        então serão executadas estas linhas.
```

Tratando Erros na Entrada de Dados
Exemplo



python

Exercício

- Desenvolva um algoritmo que peça ao usuário as seguintes informações: nome, idade, CPF, telefone e e-mail. Não esqueça de tratar os erros.

python

Tipos Numéricos

- Tipos numéricos representam valores numéricos.
- Em Python há alguns tipos numéricos pré-definidos: inteiros (int), números de ponto flutuante (float), booleanos (bool) e complexos (complex).
- Estes tipos suportam as operações matemáticas comuns como adição, subtração, multiplicação e divisão, e podem ser convertidos entre si.

python

Tipos Numéricos

- **Exemplo:**

```
>>> a = 1 # valor inteiro
>>> preco = 10.999 # valor ponto flutuante
>>> t = True # valor booleano
>>> i = 4+3j # valor complexo
```

python

Operadores

- **Aritméticos:**

```
>>> print 7 + 3 #adição
10
>>> print 10 - 3 #subtração
7
>>> print 10 / 2 #divisão inteira
5
>>> print 5.0 / 2 #divisão em ponto flutuante
2.5
>>> print 7 % 3 #resto da divisão inteira
1
>>> print 2 * 2 #multiplicação
4
>>> print 2 ** 3 #exponenciação
8
```

python

Operadores

- **Atribuição**

- Este operador é representado por um único símbolo de igualdade, **=**, definindo uma variável e automaticamente atribuindo a ela um valor.

```
>>> a = 5
>>> idade = input("Digite seu idade")
```

python

Operadores

- **Condicionais de Igualdade**

```
>>> print 2 == 4 # igualdade
False
>>> print 2 <> 4 # diferente de
True
>>> print 2 != 4 # diferente de
True
```

python

Operadores

- **Condicionais de Comparação**

```
>>> print 1 < 2 # menor que
True
>>> print 5 > 3 # maior que
True
>>> 3 >= 4 # maior ou iagual que
False
>>> 5 <= 6 # menor ou igual que
True
```

python

Operadores

- **Operadores Lógicos**

- Os operadores lógicos **not**, **and** e **or** permitem modificar e agrupar o resultado de testes condicionais:

```
>>> curso = "SI"
>>> disciplina = "Programação"
>>> curso == "SI" and disciplina == "Programação"
True
>>> nome = "Ana"
>>> idade = 25
>>> nome == "Ana" or idade == 26
True
>>> a = 10
>>> b = 5
>>> not(a == b)
True
```

python

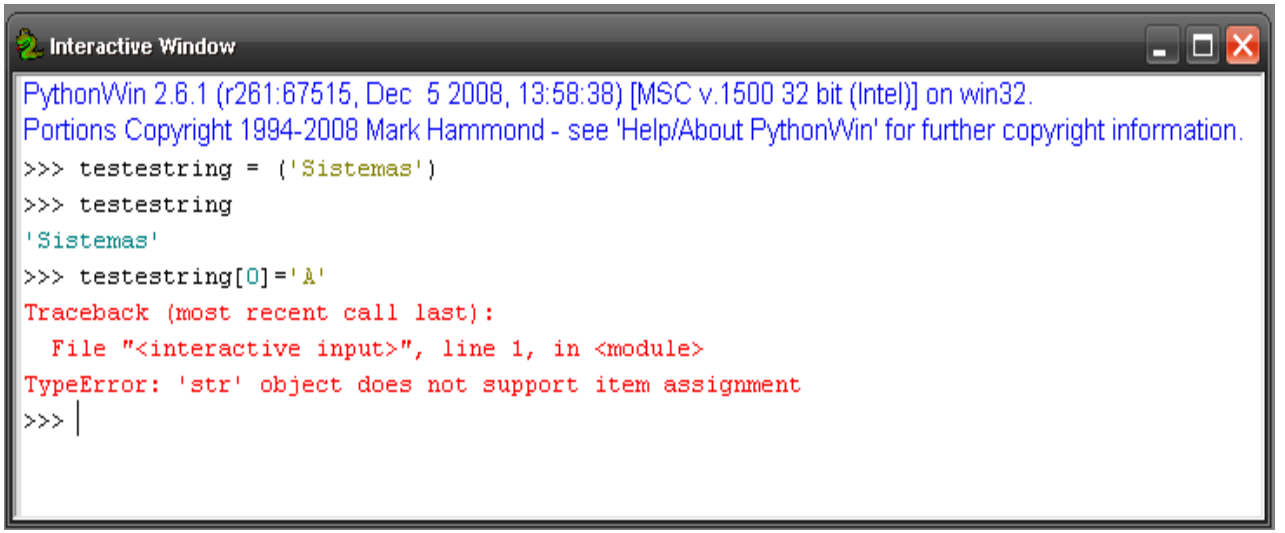
Strings

- A string, é uma seqüência imutável com um propósito especial: armazenar cadeias de caracteres.

```
>>> a = "Sistemas de Informacao"
>>> print a
Sistemas de Informacao
```

Strings

Imutável Teste



python

Strings

- Strings podem ser delimitadas tanto com aspas simples quanto duplas; se delimitamos com aspas duplas, podemos usar as aspas simples como parte literal da string, e vice-versa.

```
>>> a = "Sistema de Informacao."
>>> b = 'Sistema de Informacao.'
>>> a == b
True
```

python

Strings

- São usados caracteres especiais para denotar quebra de linha (\n), tabulação (\t) e outros.

```
>>> a = "Hoje\n\t é o primeiro dia."
>>> print a
Hoje
        é o primeiro dia.
```

python

Strings

- Para criar uma string com múltiplas linhas, é útil o delimitador aspas triplas: as linhas podem ser quebradas diretamente, e a string pode ser finalizada com outras três aspas consecutivas:

```
>>> a = """Laboratorio de
... Linguagem de
... Programacao"""
>>> print a
Laboratorio de
Linguagem de
Programacao
```

python

Strings

- Como toda seqüência, a string pode ser indexada ou dividida em *slices*, usando o operador colchetes:

S	I	S	T	E	M	A	S
0	1	2	3	4	5	6	7

Conteúdo

Índice

python

Strings

- **Exemplo de divisão:**

```
>>> palavra = 'Sistemas'
>>> palavra [3]
't'
>>> palavra [:5]
'Siste'
>>> palavra [3:]
'temas'
>>> palavra [2:6]
'stem'
>>> palavra [0:7:2]
'Ssea'
>>> palavra [-2]
'a'
```

python

Strings

- **Operações:**
  - **Concatenação:**

```
>>> frase = ' de informacao'
>>> print palavra + frase
Sistemas de informacao
```

  - **Composição:**

```
>>> print "Maria possui %d anos de idade" %idade
Maria possui 25 anos de idade
>>> print "Maria possui", idade, "anos de idade."
Maria possui 25 anos de idade.
```

python

Strings

- Para verificar o tamanho da string utilizamos a função **len.**

- **Exemplo:**

```
>>> print len(palavra)
8
```

python

Strings

- Para substituir uma palavra em uma string, basta utilizar o método **repalce**.

- **Exemplo:**

```
>>> texto = 'Sistemas de Informacao'
>>> print texto
Sistemas de Informacao
>>> texto = texto.replace('Informacao','Informação')
>>> print texto
Sistemas de Informação
```

python

Strings

- Podemos brincar com a caixa das letras (maiúsculo ou minúsculo).

- Vamos deixar tudo maiúsculo. **Exemplo**:

```
>>> texto = 'Sistemas de Informação'
>>> print texto.upper()
SISTEMAS DE INFORMAçãO
```

- 'Agora vamos deixar tudo minúsculo. **Exemplo:**

```
>>> texto = 'Sistemas de Informação'
>>> print texto.lower()
sistemas de informação
```

python

Strings

- Para contar o número de vezes que a palavra especificada aparece na string, basta utilizar a função **count**.

- **Exemplo:**

```
>>> texto = 'A comunidade Python Brasil reune grupos de usuários em
todo o Brasil interessados em difundir e divulgar a linguagem de
programação Python, abrigando em seu domínio todo o material editado
sobre esta linguagem em nosso idioma (português do Brasil).'
>>> texto.count('o')
24
```

Nesse caso estou apenas procurando pela vogal ‘o’.

python

String

- Para saber mais algumas informações sobre formatação de string acesse o site:

- http://python.org.br/wiki/ManipulandoStringsComPython

python

python

Estrutura de Decisão

python

Estrutura de Decisão

- Em python, a estrutura de decisão é o **if**. O **if** nada mais é que nosso "**se**".

- Essa estrutura também é conhecida por estrutura condicional.

- As condições servem para selecionar quando uma parte do programa deve ser ativada e quando deve ser simplesmente ignorada.

python

Estrutura de Decisão

- O **if** tem a seguinte sintaxe:

```
if condicao:
    faz alguma coisa
```

A Identação é esse deslocamento do código para a direita.

- O Python é um pouco exigente quanto a identação. Diferente de outras linguagens o Python exige que o código seja identado caso contrário o programa não roda.

python

Estrutura de Decisão

- **Exemplo 1:**

- Faça um algoritmo que leia dois valores e imprima o maior deles.

```
a = input ("Digite o primeiro valor:")
b= input ("Digite o segundo valor:")

if a >  b:
    print "O maior valor é:", a

if b >  a:
    print "O maior valor é:", b
```

python

Estrutura de Decisão

- Podemos simplificar o algoritmo que estamos desenvolvendo através da cláusula **else**. O **else** nada mais é que nosso "**senão**".

- A estrutura **if/else** é uma adaptação da lógica **se/senão.**

- **Sintaxe:**

```
if condicao:
    faz alguma coisa
else:
    faz outra coisa
```

python

Estrutura de Decisão

- **Exemplo 2:**

- Faça um algoritmo que leia dois valores e imprima o maior deles.

```
a = input("Digite o primeiro valor:")
b = input("Digite o segundo valor:")

if a > b:
    print "O maior valor é:",a
else:
    print "O maior valor é:",b
```

python

Estrutura de Decisão

- Em alguns casos precisamos de mais de um estrutura de **ifs** e acabamos criando um **if** dentro do outro, o que muitas vezes pode complicar o código.

- Esse tipo de "junção" de **ifs** é chamado de **if** aninhados.

- **Exemplo :**

```
a = input("Digite o um valor:")
if a > 0:
    print "O número é positivo"
else:
    if a < 0:
        print "O número é negativo"
    else:
        print "O número é digitado é zero"
```

python

Estrutura de Decisão

- Python apresenta uma solução muito interessante ao problema de múltiplos **ifs** aninhados.

- A cláusula **elif** substitui um par **else if**.

- **Sintaxe:**

```
if condicao:
    faz alguma coisa
elif condicao:
    faz alguma coisa
else:
    faz outra coisa
```

python

Estrutura de Decisão

- **Exemplo 3:**

- Faça um algoritmo que leia um valor e imprima se esta valor é positivo, negativo ou igual a zero.

```
num = input('Digite um número:')

if num == 0:
  print 'O número lido é igual a zero'
elif num > 0:
  print 'O número é positivo'
else:
  print 'O número é negativo'
```

python

Estrutura de Decisão

- **Cuidados !!!**
- A estrutura de **if/elif/else** só tem 2 restrições:
  - Deve começar com um if, isto é, não podemos começar com um elif e muito menos com um else;
  - Só pode ter um ou nenhum else na estrutura, isto é, o else não é obrigatório, mas caso ele seja usado, só pode constar uma única vez na estrutura;
- Em suma, depois do if primordial podemos colocar quantos elifs quisermos e acabar a estrutura assim. Caso seja necessário podemos utilizar um, e apenas um, else.

python

python

Estruturas de Repetição

python

Estruturas de Repetição

- São utilizadas para executar a mesma parte de um programa várias vezes, normalmente dependendo de uma condição.

- A linguagem python possui duas estruturas de repetição:
  - **While**
  - **For**

python

Estruturas de Repetição While

- A estrutura **while** (enquanto) ou loop **while**, como também é chamado, tem como objetivo repetir a execução de um bloco de código enquanto uma certa condição é valida.

- A sua estrutura é a seguinte:

```
while condicao:
    bloco de codigo
```

python

Estruturas de Repetição While

- **Exemplo 1:**

- Faça um algoritmo que mostre os 20 primeiros números pares.

```
contador = 0
par = 2
while contador < 20:
    print par
    par = par + 2
    contador = contador + 1
```

Variável Contadora

Variável Acumuladora

python

Estruturas de Repetição While

- **Interrompendo a Repetição**

- Embora muito útil, a estrutura while só verifica sua condição de parada no início de cada repetição.

- Já a instrução break é utilizada para interromper a execução de while independentemente do valor atual de sua condição.

python

Estruturas de Repetição While

- **Interrompendo a Repetição**

- **Exemplo 2:**

```
x = 0
while True:
    print 'x =',x
    x = x + 1
    if x == 10:
        print 'Saindo...'
        break
```

python

python

Estruturas de Repetição For

- O laço **for**, diferente do laço **while**, pois não utiliza uma condição.
- Ele precisa somente de um argumento iterável e de uma variável para repassar o item de cada iteração.
- O **for** sempre trabalha em conjunto com a instrução **in**.
- A sua estrutura básica é a seguinte:

```
for variavel in item_iteravel:
    faz algo
```

python

Estruturas de Repetição For

- Com o for, podemos “**varrer**” qualquer sequência:
  - Strings;
  - Listas;
  - Tuplas ou;
  - Dicionários.

python

Estruturas de Repetição For

- Se você precisar fazer uma **iteração** em strings, basta utilizar o comando for, conforme o exemplo abaixo:

- **Exemplo1:**

```
faculdade = "Atenas"
for letra in faculdade:
        print letra
```

O **for** em python "varre" toda a sequência(faculdade)e guarda o valor na variável(letra) em cada interação.

python

Estruturas de Repetição For

- **Função Range**

- Agora se você precisar **iterar** sobre seqüências numéricas, a função interna range() é a resposta. Ela gera listas contendo progressões aritméticas.

- **Exemplo:**

```
>>> range(10)
[0, 1, 2, 3, 4, 5, 6, 7, 8, 9]
>>> range(5,10)
[5, 6, 7, 8, 9]
>>> range(0,10,3)
[0, 3, 6, 9]
```

python

Estruturas de Repetição For

- **Função Range**

- Ao especificar somente um valor, a função range retorna uma lista de número.

- Diferente do que imaginamos, ela não retorna uma lista até o número especificado (nesse caso, de 0 a 10) mas sim um intervalo que se inicia no zero e termina no número inteiro que antecede o número especificado.

python

Estruturas de Repetição For

- **Exemplo 2:**

- Faça um algoritmo que imprima 10 vezes a frase "Sistemas de Informação".

```
for i in range(10):
    print "Sistemas de Informacao"
```

python

Estruturas de Repetição For

- **Exemplo 3:**

- Faça um algoritmo que mostre os 20 primeiros números pares.

```
for i in range(2,42,2):
    print i
```

python

Estruturas de Repetição For

- **Interrompendo a Repetição**

- Vale lembrar que a instrução **break** também interrompe o **For**.

- **Exemplo 4:**

```
for i in range(10):
    num = input("Digite um numero:")
    if num % 2 == 0:
        print 'O número digitado é par'
        break
```

python

While e For

- Quando usar **while** e quando usar **for**?

- Para casos em que o fim é indeterminado:
  - Usar **while**
    - while x <> 0:
    - while x > 0:

- Para casos em que o fim é determinado:
  - Usar **for**
    - for i in range (100):
    - for i in range (1,n):

python

Listas

python

Listas

- Uma lista é um conjunto ordenado de valores (como um vetor em outras linguagens) , acessados por um índice.

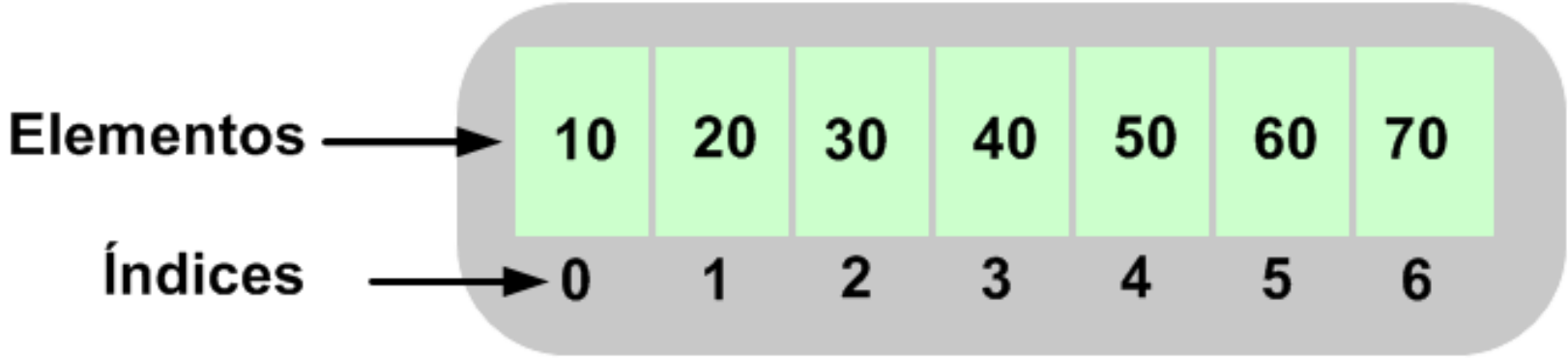
- Os índices são iniciados em zero e atribuídos sequencialmente a partir deste.

- Uma lista pode conter zero ou mais elementos de um mesmo tipo ou de tipos diferentes, podendo inclusive conter outras listas.

python

Listas

- **Exemplo:**



python

Listas

- Listas são similares a strings, porém elas são mutáveis.
- As listas são **estruturas flexíveis** quanto ao seu tamanho.

python

Criando uma Lista

- Para criar uma lista, usamos colchetes e vírgulas para separar os elementos.

- **Exemplo:**

Lista Vazia

```
>>> A = []
```

Lista de elementos inteiros

```
>>> B = [10,20,30]
```

Lista contendo elementos de vários tipos

```
>>> C = [30,4.5,"SI"]
```

Lista contendo outras listas

```
>>> D = [B,C]
```

python

Acessando Elementos

- Para acessar um elemento específico de uma lista, usamos o nome da lista seguido do índice entre colchetes.

- **Exemplo:**

```
>>> B = [10,20,30]
>>> print B[0]
10
>>> print B[1]
20
>>> print B[2]
30
```

python

Acessando Elementos

- Usando índices negativos, as posições são acessadas a partir do final da lista, -1 indicando o último elemento.

- **Exemplo:**

```
>>> C = [30,4.5,"SI"]
>>> print C[-1]
SI
>>> print C[-2]
4.5
>>> print C[-3]
30
```

python

Modificando Listas

- Para modificar o valor de uma lista, basta atribuir ao índice o novo valor.

- **Exemplo:**

```
>>> B = [10,20,30]
>>> B[0]=40
>>> print B
[40, 20, 30]
```

python

Copiando Listas

- Embora listas em python sejam um recurso muito poderoso, todo poder traz responsabilidades.

- Um dos efeitos colaterais de listas aparece quando tentamos fazer cópias.

python

Copiando Listas

- **Exemplo:**

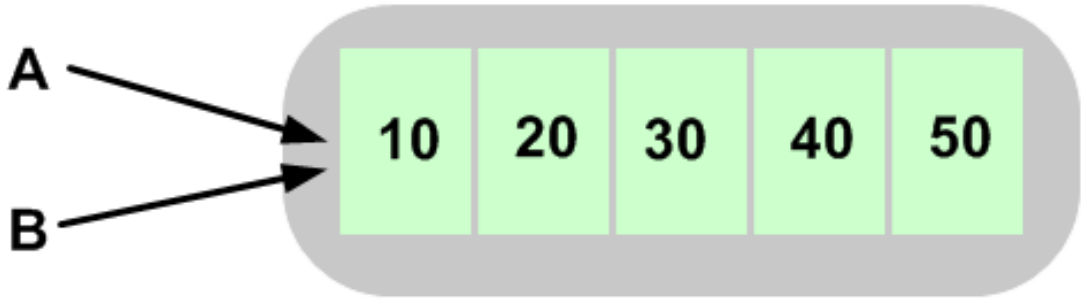
```
>>> A = [10,20,30,40,50]
>>> B = A
>>> print A
[10, 20, 30, 40, 50]
>>> print B
[10, 20, 30, 40, 50]
>>> B[0] = 60
>>> print A
[60, 20, 30, 40, 50]
>>> print B
[60, 20, 30, 40, 50]
```

python

Copiando Listas

- Ao modificarmos a lista **B**, modificamos também o conteúdo de **A**, pois ambos são referências para a mesma lista na memória.

- Isso acontece porque uma lista em python é um objeto e, quando atribuímos um objeto a outro, estamos apenas copiando a mesma referência da lista e não seus dados em si.



python

Copiando Listas

- Para criar uma cópia independente de uma lista, utilizamos outra sintaxe.

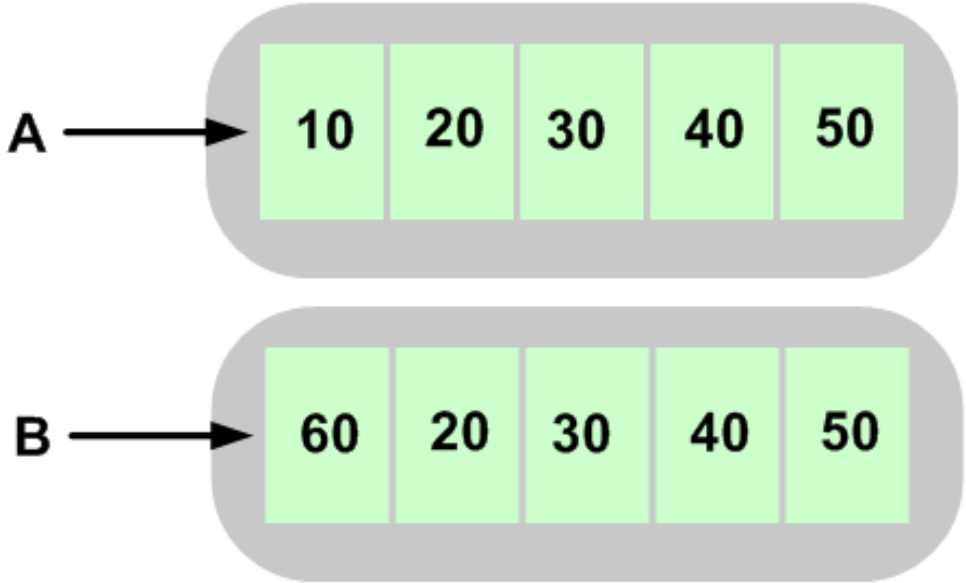
- **Exemplo:**

```
>>> A = [10,20,30,40,50]
>>> B = A[:]
>>> print A
[10, 20, 30, 40, 50]
>>> print B
[10, 20, 30, 40, 50]
>>> B[0]=60
>>> print A
[10, 20, 30, 40, 50]
>>> print B
[60, 20, 30, 40, 50]
```

python

Copiando Listas

- Ao escrevermos **B = A[:]** estamos nos referindo a uma cópia de **A**. Assim **A** e **B** se referem a áreas diferentes na memória, permitindo alterá-las de forma independente.



python

Fatiando Listas

- Podemos também fatiar uma lista, da mesma forma que fizemos com strings.

- **Exemplo:**

```
>>> A = [10,20,30,40,50]
>>> print A[1:3]
[20, 30]
>>> print A[0:]
[10, 20, 30, 40, 50]
>>> print A[:4]
[10, 20, 30, 40]
>>> print A[:-2]
[10, 20, 30]
```

python

Tamanho da Lista

- Para definirmos o tamanho de uma lista, basta utilizarmos a função **len**.

- O valor retornado é igual ao número de elementos da lista.

- **Exemplo:**

```
>>> A = [10,20,30,40,50]
>>> print len(A)
5
```

python

Operações em Listas

- O operador (**+**) é utilizado para concatenar listas.

- **Exemplo:**

```
>>> A = [1,2,3,4,5]
>>> B = [6,7,8,9,10]
>>> C = A + B
>>> print C
[1, 2, 3, 4, 5, 6, 7, 8, 9, 10]
```

python

Operações em Listas

- Similarmente, o operador (*) repete uma lista um número dado de vezes.

- **Exemplo:**

```
>>> A = [1,2,3]
>>> print A
[1, 2, 3]
>>> print A * 3
[1, 2, 3, 1, 2, 3, 1, 2, 3]
```

python

Operações em Listas

- Para saber se um elemento pertence a uma lista utilizamos o operador **in**.

- **Exemplo:**

```
>>> A = [10,20,30]
>>> 20 in A
True
>>> 40 in A
False
```

python

Adicionando Elementos na Lista

- Uma das principais vantagens de trabalharmos com listas é poder adicionar novos elementos durante a execução do programa.

- Para adicionarmos um elemento ao fim da lista, utilizamos o método **append**.

- **Exemplo:**

```
>>> A = [10,20,30,40,50]
>>> print A
[10, 20, 30, 40, 50]
>>> A.append(60)
>>> print A
[10, 20, 30, 40, 50, 60]
```

python

Listas Aninhadas

- Uma lista aninhada é uma lista que aparece como um elemento de uma outra lista.

- **Exemplo:**

Lista aninhada

```
>>> A = [10,20,[30,40]]
>>> print A
[10, 20, [30, 40]]
```

python

Listas Aninhadas

- Para extrairmos um elemento de uma lista aninhada, podemos agir em duas etapas.

- **Exemplo:**

```
>>> A = [10,20,[30,40]]
>>> elem = A[2]
>>> print elem[0]
30
```

- Ou podemos combiná-las:

```
>>> A = [10,20,[30,40]]
>>> print A[2][0]
30
```

python

Exemplos de Utilização de Listas

- **Exemplo 1:**

- Faça um algoritmo que crie uma lista contendo dez posições e cada posição tenha um valor inteiro. Mostre os valores.

python

Solução Exemplo 1

```
rep = []
for cont in range(10): # entrada de dados
        valor = input("Digite um valor:")
        rep.append(valor)

print rep
```

python

Exemplos de Utilização de Listas

- **Exemplo 2:**

  Uma prova de química foi feita por um grupo de 10 alunos. Faça um algoritmo para ler as notas obtidas pelos alunos, e depois exibir um relatório das notas iguais ou superiores a 7,5:

python

Solução Exemplo 2

```
nota = []
for cont in range(10): # entrada de dados
        valor = input("Digite um valor:")
        nota.append(valor)

for cont in range(10): # compara notas
        if nota[cont]>=7.5:
                print nota[cont]
```

python

Exemplos de Utilização de Listas

- **Exemplo 3:**

- Num concurso público, um candidato respondeu a uma avaliação com 80 questões de múltipla escolha, onde cada questão tinha resposta A até D. Fazer um algoritmo para ler o gabarito da prova e as respostas do aluno, informando quantas questões ele acertou.

python

Solução Exemplo 3

```
gabarito = []
resposta = []
certo = 0
for cont in range(10): # leitura do gabarito
        valor = input("Digite a resposta da questao:")
        gabarito.append(valor)

for cont in range(10): # leitura das respostas do candidato
        valor = input("Digite a resposta do candidato:")
        resposta.append(valor)

for cont in range(10): # compara notas
        if gabarito[cont] == resposta[cont]:
                acerto = acerto + 1

print "O canditado acertou", acerto, "questões"
```

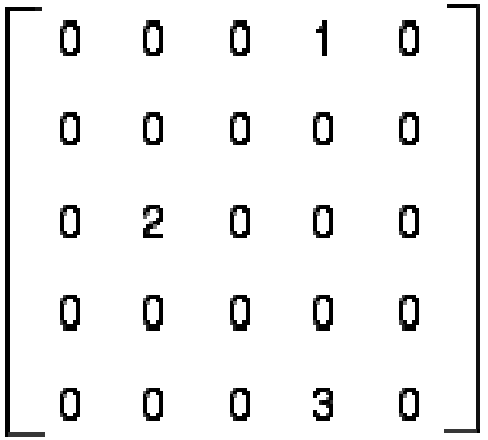
python

python

Matriz

- Em python, uma matriz é uma lista que possui mais de uma dimensão, geralmente duas.

- Sendo bidimensional, a variável atua como uma grade de linhas e colunas, onde a interseção entre uma linha e uma coluna armazena um dado.



python

Matriz

- Cabe observar que uma lista unidimensional nada mais é do que uma matriz com uma única linha – ou uma única coluna, dependendo do ponto de vista.

- Para trabalhar com matriz em python utilizamos cadeias de For e listas (listas aninhadas).

- **Exemplo:**

```
>>> matriz = [[1,2,3],[4,5,6]]
>>> print matriz
[[1, 2, 3], [4, 5, 6]]
>>> print matriz[1]
[4, 5, 6]
```

python

Exemplos de Utilização de Listas

- **Exemplo 1:**

- Faça um algoritmo que preencha uma matriz 3x5 com o número 1 em todas as posições e imprima o resultado.

python

Solução Exemplo 1

```
matriz=[]
for i in range(3):
    matriz.append([])
    for j in range(5):
        matriz[i].append(1)

for i in range(3):
    print'\n'
    for j in range(5):
        print matriz[i][j],
```

python

Exemplos de Utilização de Listas

- **Exemplo 2:**

- Faça um algoritmo para gerar e exibir a matriz abaixo:

	1	2	3	4
1	11	12	13	14
2	15	16	17	18
3	19	20	21	22

python

Solução Exemplo 2

```
matriz=[]
cont=11
for i in range(3):
    matriz.append([])
    for j in range(4):
        matriz[i].append(cont)
        cont=cont+1

for i in range(3):
    print'\n'
    for j in range(4):
        print matriz[i][j],
```

python

python

Funções

- São pequenos trechos de código reutilizáveis.
- Podemos definir nossas próprias funções.
- Nós já usamos algumas funções internas do Python como **len**, **range, print e input**.
- Podemos utilizar uma função em qualquer lugar do programa, quantas vezes quiser, ou seja, basta "**chamar**" a função para que ela possa ser executada.

python

Funções

- Funções são definidas usando a palavra-chave **def**. Esta é seguida pelo nome da função, um par de parênteses que pode envolver algumas variáveis, dois pontos e o bloco de comandos:

- **Exemplo:**

```
def <nome da funcao>():
        # bloco de comando
```

python

Funções

- **Exemplo 1:**

- Faça uma algoritmo que possua uma função que imprima a frase “Alo Mundo”.

```
# Definindo a função
def alo():
    print 'Alo Mundo'

#Corpo principal do algoritmo

alo() # Chamada da função
```

python

Funções

- **Exemplo 2:**

- Faça uma algoritmo que possua uma função que imprima a soma de dois números.

```
def soma():
        a = input("Digite um valor:")
        b = input("Digite outro valor:")
        print a + b

# Corpo principal do algoritmo

soma() # chamada da função
```

python

Passagem de Parâmetros e Argumentos

- **Parâmetros** são as variáveis que podem ser incluídas nos parênteses da definição da função.

- O bloco de comandos da função pode utilizar estas variáveis para processar a informação e executar algum trabalho útil.

- Quando a função é chamada são passados valores a serem atribuídos a estas variáveis, estes valores são chamados **argumentos**.

python

Passagem de Parâmetros e Argumentos

- **Exemplo 3:**

- Faça uma algoritmo que possua uma função que imprima a soma de dois números.

```
#Utilizando funções com parametros e argumentos

def soma(a,b):# a e b são os parametros da função
    print'A soma de',a,'e',b,'é',a+b

#chamando a funcao soma()
soma(3,4)#3 e 4 são os argumentos da função
```

python

Passagem de Parâmetros e Argumentos

- **Exemplo 4:**

- Escreva uma função que receba dois valores inteiros e apresente como saída uma mensagem informando se os números são iguais ou diferentes.

```
def compara(x,y):
        if x == y:
                print 'números iguais'
        else:
                print 'números diferentes'

# Corpo principal do algoritmo

a = input("Digite um valor:")
b = input("Digite outro valor:")
compara(a,b) # chamada da função
```

python

Variáveis Locais

- Quando você declara uma variável dentro de uma função, ela é totalmente independente de qualquer outra variável que tenha o mesmo nome fora da função, isto é, nomes de variáveis são **locais** a função.

- Isto é chamado **escopo** da variável. Todas as variáveis tem o escopo do bloco de comandos em que foram criadas.

python

Variáveis Locais

- **Exemplo 5:**

```
#verificando que variáveis tem escopo local
def verifica(x):
        print 'x é:',x
        x = 2
        print 'valor local de x agora é:', x

# Corpo principal do algoritmo

x = 50
verifica (x)# chamada da função
print 'x ainda é:', x
```

Variável Local da função.

Variável Local do corpo principal do algoritmo.

python

Variáveis Globais

- Se quisermos modificar uma variável global dentro de uma função, devemos informar que estamos usando uma variável global antes de inicializá-la, na primeira linha de nossa função.

- Essa definição é feita com a instrução **global**.

python

Variáveis Globais

- **Exemplo 6:**

```
#verificando que variáveis tem escopo global
def verifica():
        global x
        print 'x é:',x
        x = 2
        print 'valor local de x agora é:', x

# Corpo principal do algoritmo

x = 50
verifica()# chamada da função
print 'x ainda é:', x
```

Variável Global a todo algoritmo.

python

Retornando um Valor

- Nem sempre uma função irá por exemplo imprimir a soma de dois números,e sim apenas realizar uma operação.

- Sendo assim, para retornar um valor de uma função, basta usar a expressão **return** dentro da função.

python

Retornando um Valor

- **Exemplo 7:**

- Fazer um algoritmo que retorne a soma dos valores contidos em uma lista.

```
def soma(lista):
        total=0
        for i in range(4):
                total=total + lista[i]
        return(total)
corpo principal do algorimto
l=[10,20,25,35]
print(soma(l))
```

python

Retornando um Valor

- **Exemplo 8:**

- Fazer um algoritmo que retorne a média dos valores contidos em uma lista.

```
def soma(lista):
        total=0
        for i in range(4):
                total=total + lista[i]
        return(total)

def media(lista):
        return(soma(l)/len(l))

corpo principal do algorimto

l=[10,20,25,35]
print(media(l))
```

python

Documentando uma Função

- Para facilitar a vida de quem usará seus programas, e talvez a sua própria vida, quando tiver que fazer manutenção nos seus programas, pode-se inserir pequenas informações sobre as funções dentro delas que são as Chamadas “**doc strings**”.

python

Documentando uma Função

- **Exemplo 9:**

```
def f(x):
    """ Esta função retrna o quadrado de um número """
    return x **2

corpo principal do algorimto

print f(2)
print f.__doc__
```

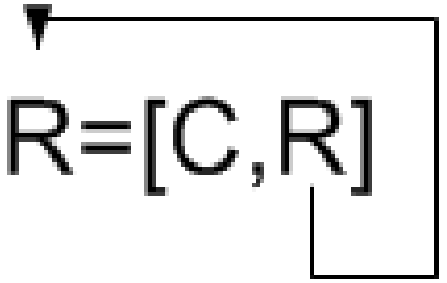
python

Recursividade

- A recursividade é o processo de definição de algo em termos de si mesmo.

- Uma função é recursiva se uma declaração no corpo da função chama a ela mesma.

- Exemplo: uma rotina recursiva R pode ser expressa como uma composição formada por um conjunto de comandos C e uma chamada (recursiva) à rotina R:



python

Recursividade

- Na recursividade, uma função deve estar apta a chamar a si própria.

- **Exemplo 10:**

```
def teste(n):
    if n == 0:
        print 'fogo'
    else:
        print n
        teste(n-1)

corpo principal do algorimto

teste(3)
```

python

Recursividade

- **Exemplo 11:**

- Faça um algoritmo utilizando recursividade para calcular o fatorial de um determinado número.

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

# corpo principal do algorimto
num = input('Digite um valor:')
print fatorial(num)
```

python

python

Módulos

- Imagine que pretendemos usar uma mesma função em vários programas sem ter de reescrever o código em cada um dos programas.

- Isso é possível com a utilização de **módulos**.

- **Módulos** são programas feitos para serem reaproveitados em outros programas.

- Eles tipicamente contêm funções, variáveis, classes e obje tos que provêm alguma funcionalidade comum.

python

Criando um Módulo

- Os módulos devem ser “Salvos” numa pasta(ou diretório como preferir) que esteja no path.

- Path, de uma forma simples, pode ser definida como o lugar onde o Python procura os módulos quando são invocados num programa.

python

Criando um Módulo

- Para saber quais as pastas que estão no path basta executar o comando abaixo:

```
>>> import sys
>>> for i in sys.path:
...     print i
...

C:\WINDOWS\system32\python26.zip
C:\Python26\DLLs
C:\Python26\lib
C:\Python26\lib\plat-win
C:\Python26\lib\lib-tk
C:\Python26\Lib\site-packages\pythonwin
C:\Python26
C:\Python26\lib\site-packages
C:\Python26\lib\site-packages\win32
C:\Python26\lib\site-packages\win32\lib
C:\Documents and Settings\Fabio
JR\Desktop
```

python

Criando um Módulo

- Vamos criar agora o nosso módulo, que será um conjunto de funções de manipulação de dois números(soma,diferença,produto,quociente,potência).

- Gravaremos no diretório principal do Python(C:\python26, se estiver no path), com o nome numeros.py.

python

Criando um Módulo

- **Exemplo:**

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

def diferenca(x,y):
        return x-y

def produto(x,y):
        return x*y

def quociente(x,y):
        return x/y

def potencia(x,y):
        return x**y
```

python

Importando um Módulo

- A instrução básica para manipular módulos é **import**.

- No exemplo a seguir, vamos importar o módulo numeros.py que acabamos de criar.

- **Exemplo:**

Nome do Módulo

Função do Módulo

```
>>> import numeros
>>> print numeros.soma(2,3)
5
>>> print numeros.potencia(3,4)
81
```

python

Importando um Módulo

- Para importar todas as funções do módulo numeros.py, basta utilizar os comandos abaixo:

- **Exemplo:**

```
>>> from numeros import *
>>> print soma(2,3)
5
>>> print diferenca(7,3)
4
```

python

Função Dir

- A função **dir()** retorna uma lista de Strings com o nome de todas as funções definidas em um determinado módulo.

- **Exemplo:**

```
>>> import numeros
>>> dir(numeros)
['__builtins__', '__doc__', '__file__',
'__name__', '__package__', 'diferenca',
'potencia', 'produto', 'quociente', 'soma']
```

O módulo math

- O módulo **math** é um módulo nativo do Python, isto é, ele é distribuído juntamente com o interpretador, e possui funções para realizar diversas operações matemáticas.

- **Exemplo:**

Calcula a raiz quadrada de um número.

```
>>> import math
>>> raiz = math.sqrt(81)
>>> print raiz
9.0
```

python

O módulo os

- O módulo os é utilizado para manipular o seu sistema operacional, qualquer que seja ele.

- **Exemplo:**

```
>>> import os
>>> print "Sistemas de Informação"
Sistemas de Informação
>>> os.system("cls")
```

Utilizado para apagar o conteúdo da tela.

python

O módulo random

- Uma forma de gerar valores para testar funções e popular listas é utilizar números aleatórios ou randômicos.

- O módulo random gera números aleatórios em python.

- **Exemplo:**

Utilizamos a função randint do módulo random para gerar um valor entre 1 e 10.

```
>>> import random
>>> valores = random.randint(1,10)
>>> print valores
10
```

python

Exemplos de Utilização de Módulo

- **Exemplo 1:**

- Faça um algoritmo que crie um módulo para reajustar o valor de um produto.

python

Solução Exemplo 1

- **1 Passo – Criar Módulo:**

```
# Criando Módulo para reajustar valor
def reajustavalor(valor,aumento):
    return (((valor * aumento) / 100.0) + valor)
```

- **2 Passo – Utilizar Módulo Desenvolvido:**

```
# importanto módulo reajusta
import reajusta

valor = input("Digite o valor do produto:")
aumento = input("Digite o valor do aumento:")

result = reajusta.reajustavalor(valor,aumento)
print 'O novo valor do produto é:', result
```

python

Arquivos

python

Introdução

- Durante a execução de um programa, seus dados ficam na memória.

- Para armazenar os dados permanentemente, você tem que colocá-los em um **arquivo**. Arquivos usualmente são guardados em um disco rígido (HD), num disquete, em um CD-ROM,etc.

python

Introdução

- Quando existe um número muito grande de arquivos, eles muitas vezes são organizados dentro de **diretórios** (também chamados de pastas).

- A principal vantagem na utilização de um arquivo está no fato que as informações podem ser utilizadas a qualquer momento e nele é possível armazenar um número maior de informações do que as de memória, estando apenas limitado ao tamanho do meio físico para sua gravação.

python

Manipulando Arquivos

- Ao Trabalharmos com arquivos, devemos sempre realizar o seguinte ciclo:
  - Abertura, leitura e/ou escrita, fechamento.
- A função **open** permite criar novos arquivos, abrir arquivos existentes, como leitura ou escrita.

python

Manipulando Arquivos

- Sintaxe:

```
variável = open("file", "modo")
```

python

Manipulando Arquivos

Onde:

- **variável** = Nome da variável que você que vai receber o conteúdo da função open (receber o arquivo propriamente dito).
- **file** = Nome do arquivo que você quer ler ou escrever.
- **modo** = Existem três tipos básicos de modos: 'w', 'r', 'a'.

python

Manipulando Arquivos

- Write (w) é usado para escrever algo no arquivo, apagando o que já havia nele, e caso o arquivo ainda não exista, ele será criado.

- Mas e se eu quiser apenas adicionar um texto sem apagar o que já tinha la? Ai estamos falando no modo 'a' (append, pra quem conhece os métodos de listas isso deve ser familiar), agora todo o texto escrito será adicionado no final do que tinha anteriormente , nesse caso, se o arquivo também não existir ele será criado sem problemas.

- E como era de se esperar, read (r) é usado para ler um arquivo já existente.

python

Manipulando Arquivos

- **Exemplo 1:**

Nome do Arquivo.

Modo de Abertura.

```
>>> f = open("teste.dat","w")
>>> print f
<open file 'teste.dat', mode 'w' at 0x013DD5C0>
```

- Se não existir nenhum arquivo de nome teste.dat, ele será criado. Se já existir um, ele será substituído pelo arquivo que estamos gravando (ou escrevendo).

- Quando executamos um comando print sobre o objeto arquivo, visualizamos o nome do arquivo, o modo e a localização do objeto na memória.

python

Manipulando Arquivos

- Para colocar dados dentro do arquivo, invocamos o método write do objeto arquivo.

- **Exemplo 2:**

```
>>> f = open("teste.dat","w")
>>> f.write("Sistemas de Informação")
>>> f.close()
```

python

Manipulando Arquivos

- Agora podemos abrir o arquivo de novo, desta vez para leitura, e ler o seu conteúdo para uma string. Desta vez, o argumento modo é “r”(o método read lê dados do arquivo) para leitura.

- **Exemplo 3:**

```
>>> f = open("teste.dat","r")
>>> texto = f.read()
>>> print texto
Sistemas de Informação
```

python

Manipulando Arquivos

- Caso queira inserir novas informações no arquivo, desta vez, o argumento modo é “a”(o método append).

- **Exemplo 4:**

```
>>> f = open("teste.dat","w")
>>> f.write("Sistemas")
>>> f.close()
>>> f = open("teste.dat","a")
>>> f.write(" de informação")
>>> f.close()
>>> f = open("teste.dat","r")
>>> cad = f.read()
>>> print cad
Sistemas de informaçao
```

python

Diretórios

- Quando você cria um novo arquivo abrindo-o e escrevendo nele, o novo arquivo fica no diretório corrente, ou seja, onde o python foi instalado.

- Se você quiser abrir um arquivo que esteja em algum outro lugar, você tem que especificar o **caminho** (*path*) para o arquivo, o qual é o nome do diretório onde o arquivo está localizado:

```
variável = open("caminho/file", "modo")
```

python

Diretórios

- **Exemplo 5:**

```
>>> f = open("c:/temp/teste.dat","w")
>>> f.write("Sistemas de Informação")
>>> f.close()
>>> f = open("c:/temp/teste.dat","r")
>>> texto = f.read()
>>> print texto
Sistemas de Informação
```

python

Exercício

- Crie um arquivo chamado aula.dat no diretório c:/temp e inserira algumas informações ao mesmo.

```
>>> f = open("c:/temp/aula.dat","w")
>>> f.write("Laboratório de Linguagem de Programação")
>>> f.close()
>>> f = open("c:/temp/aula.dat","r")
>>> texto= f.read()
>>> print texto
Laboratório de Linguagem de Programação
```

python

Criando um Diretório

- Para criar um diretório é necessário utilizar o módulo **os (permite interação com o sistema operacional)** que já vem incluído no Python.

- **Exemplo 6:**

```
# importando o modulo os
import os
# a função mkdir é utilizada para criar um diretório
os.mkdir('c:/teste')
```

python

Criando um Diretório e Inserindo Informação

- **Exemplo 7:**

```
# importando o modulo os
import os
# a função mkdir é utilizada para criar um diretório
os.mkdir('c:/teste')
# criando um arquivo
cad = open("c:/teste/aula.dat","w")
#inserindo informações no arquivo aula.dat
cad.write("Python")
cad.close()
#abrindo o arquivo para leitura
cad = open("c:/teste/aula.dat","r")
texto = cad.read()
print texto
cad.close()
```

python

Exemplos de Utilização de Listas

- **Exemplo 1:**

```
# script utilizado para compactar um arquivo
import os,zipfile
#define o caminho
caminho = "c:/temp.txt"
loczip = os.path.splitext(caminho)[0] + ".zip"
zip = zipfile.ZipFile (loczip,"w")
zip.write(caminho)
zip.close()
```

python