GRADUAÇÃO

UniCesumar

Programação I

Prof. Msc. Aparecido Vilela Junior

aparecido.vilela@unicesumar.edu.br

Python

Uma introdução

Introdução

- *Criada por Guido van Rossum no final de 1989, no Instituto Nacional de Matemática e Ciência da Computação da Holanda (CWI);*

- *Tinha como principal foco auxiliar físicos e engenheiros;*

- *O nome é uma homenagem ao grupo humorístico britânico Monty Python, criador do programa Monty Python's Flying Circus;*

- *Linguagem de programação livre, com desenvolvimento comunitário;*

Introdução

- Além de sua expressividade, Python gera programas inteligíveis, fáceis de compreender e de manter.

- Muitos módulos científicos já existem internet afora que desobrigam o cientista do trabalho de reinventar a roda.

Qualidade de Software

- O código do Python é projetado para ser legível, reutilizável e de fácil manutenção, mostrando que ela é muito mais que uma tradicional linguagem de scripting.

- A uniformidade de Python a torna fácil de entender mesmo que você não programe com ela.

- Python tem suporte aos mais avançados mecanismos de reuso de software, com Programação Orientada a Objetos (POO).

Produtividade no Desenvolvimento

- Python apresenta maior produtividade quanto a escrita de programas em linguagens compiladas estaticamente como C/C++ e Java.
- O código Python é equivalente a aproximadamente 1/3 (um-terço) ou 1/5 (um-quinto) do tamanho do código escrito em C/C++ ou em Java para uma mesma aplicação.
- Isso significa menos tempo perdido com tipos, com debuger e com manutenção.
- Os programas Python são executados em tempo real, sem necessidade de compilação ou de ligação com outras ferramentas, fazendo com que o programador impulsione mais velocidade.

Portabilidade

- Portabilidade é uma característica de programas que podem ser executados sobre mais de uma plataforma de computadores, isto é, programas que podem ser executados sobre diferentes sistemas operacionais, como Linux e Windows, sem a necessidade de ser compilado para um ou outro sistema operacional.

- Além disso Python oferece vários codificações portáveis de interface gráfica com usuário, acesso a banco de dados, sistemas baseados na web, sistemas para dispositivos móveis, entre outras. Mesmo interfaces de Sistema Operacional pode ser portável em Python, além de navegadores de diretórios que podem ser

Bibliotecas

- Python trás uma pré-coleção de funções, módulos e classe, e várias funcionalidades portáveis conhecidas como **biblioteca padrão.**

- Mas pode também acrescentar programas de terceiros.

Características

- É uma linguagem interpretada (com Perl, Shell script, etc) – Não é necessário compilar o programa;
- Os arquivos fonte podem ser executados diretamente pelo interpretador, que os converte em byte codes (que são multiplataforma);
- O Python pode ser executado diretamente no terminal:

  python (executa o python no modo interativo)

  python teste.py (executa o programa teste.py)

- É ideal para ser usado como linguagem de scripts, automatizando tarefas;
- É uma linguagem de aprendizado fácil, com sintaxe clara e concisa.

Características

- É uma linguagem com tipagem forte, porém dinâmica;
- Estrutura simples (não é necessário digitar ponto-e-vírgula ao final de cada linha de instruções – caso seja preciso continuar na linha seguinte pode-se fazer uso da barra invertida);
- Também não é necessário abrir e fechar blocos de código com chaves, como o C, por exemplo;
- A identificação de blocos é feita através de identação, Ex:

```
(1)while x<100:
(2)s=s+x
(3)x=x+1
(4)print s
```

Características

- *É uma linguagem orientada a objetos;*

  Tudo em python é objeto, até mesmo os inteiros. Desta forma, até os tipos mais básicos possuem métodos específicos;

- *Oferece ferramentas para:*

  - Programação funcional;

  - Processamento de imagens;

  - Interface gráfica;

  - Processamento distribuído;

  - Integração com C e C#.

Características

- *É uma linguagem orientada a objetos;*

  Tudo em python é objeto, até mesmo os inteiros. Desta forma, até os tipos mais básicos possuem métodos específicos;

- *Oferece ferramentas para:*

  - Programação funcional;

  - Processamento de imagens;

  - Interface gráfica;

  - Processamento distribuído;

  - Integração com C e C#.

Características

- *Integração com outros programas como linguagem de script.*
- *Blender:*

  O Python permite acessar todas as estruturas do Blender (operações 3D, manipulação de materiais, texturas e ambientes de cenas);

- *BrOffice.org:*

  Dá suporte ao Python como linguagem de Macro (automatiza determinadas tarefas);

Variáveis

- São nomes dados a áreas de memória
  - Nomes podem ser compostos de algarismos,letras ou _
  - O primeiro caractere não pode ser um algarismo
  - Palavras reservadas (if, while, etc) são proibidas
- Servem para:
  - Guardar valores intermediários
  - Construir estruturas de dados
- Uma variável é modificada usando o comando de atribuição:

  Var = expressão
- É possível também atribuir a várias variáveis simultaneamente:

  var1,var2,...,varN = expr1,expr2,...,exprN

Variáveis


```
>>> a=1
>>> a
1
>>> a=2*a
>>> a
2
>>> a,b=3*a,a
>>> a,b
(6,2)
>>> a,b=b,a
>>> a,b
(2,6)
```

Variáveis

- Variáveis são criadas dinamicamente e destruídas quando não mais necessárias, por exemplo, quando saem fora de escopo (veremos isso mais tarde)
- O *tipo* de uma variável muda conforme o valor atribuído, i.e., int, float, string, etc.
  - Não confundir com linguagens *sem tipo*
  - Ex.:

```
>>> a ="1"
>>> b = 1
>>> a+b
Traceback (most recent call last):
  File "<stdin>", line 1, in ?
TypeError: cannot concatenate 'str' and 'int' objects
```

]

Números

- Há vários tipos numéricos que se pode usar em python
  - **Int**: números inteiros de *precisão fixa*
    - 1 , 2 , 15 , -19
  - **Long**: números inteiros de *precisão arbitrária*
    - 1L , 10000L , -9999999L
  - **Floats**: números racionais de precisão *variável*
    - 1.0 , 10.5 , -19000.00005 , 15e-5
  - **Complex**: números complexos
    - 1+1j , 20j , 1000+100J

Operadores

+ (Soma)

- (Subtração)

** (Multiplicação)*

/ (Divisão)

// (Divisão inteira)

*** (Exponenciação)*

% (Resto de divisão inteira)

Strings – Fatias (slices)

- Notação para separar trechos de uma string
  - Notação: *string*[*índice1*:*índice2*]
  - Retorna os caracteres desde o de índice1 (inclusive) até o de índice2 (exclusive)
  - Se o primeiro índice é omitido, é assumido 0
  - Se o último índice é omitido, é assumido o fim da string

Strings – Fatias (slices)


```
>>> a
'abcde'
>>> a[0:2]
'ab'
>>> a [2:]
'cde'
>>> a[:]
'abcde'
>>> a[-1:]
'e'
>>> a[:-1]
'abcd'
```

Strings

- *Concatenação:*

```
>>> st='estou'
>>> s=' na aula de programação II'
>>> print (st+s)
estou na aula de programação II
>>> s='estou' '  na aula de programação II'
>>> print(s)
estou na aula de programação II
```

Strings

- *Interpolação:*

```
>>> st='vida boa'
>>> print “o comprimento de %s é %d” % (st,len(st))
o comprimento de vida boa é 8
```

- *Uso da string como sequencia:*

```
>>> c=‘Aula’
>>> for c in s:
...     print (c )
...
A
u
l
a
```

Strings

- *Repetição de strings:*

```
>>>3*'string'
'stringstringstrig‘
```

- *Métodos:*

```
>>> s=‘Engenharia'
>>> print(s.lower()) #retorna a string com caracteres minúsculos
redes
>>> if s.startswith(‘R'): #verifica se a string começa com ‘R'
...     print('começa' )
...
começa
>>> if s.endswith(‘S'): #verifica se a string termina com ‘S'
...     print ('termina' )
...
termina
```

Strings

- São cadeias de caracteres
- Constituem outro tipo fundamental do python
- Constantes ***string*** são escritas usando aspas simples ou duplas
  - Ex.: "a" ou 'a'
- O operador “+” pode ser usado para concatenar strings
  - Ex.: "a"+"b" é o mesmo que "ab"
- O operador “*” pode ser usado para repetir strings
  - Ex.: "a"*10 é o mesmo que "aaaaaaaaaa"

Strings

- Python usa a tabela de caracteres default do S.O.
  - Ex.: ASCII, UTF-8
- Caracteres não imprimíveis podem ser expressos usando notação "barra-invertida" (\)
  - \n é o mesmo que *new line*
  - \r é o mesmo que *carriage return*
  - \t é o mesmo que *tab*
  - \b é o mesmo que *backspace*
  - \\ é o mesmo que \
  - \x41 é o mesmo que o caractere cujo código hexadecimal é 41 ("A" maiúsculo)

Strings


```
>>> "ab\rd"
'ab\rd'
>>> print("ab\rd“) # print exibe chars não imprimíveis
adb
>>> print("abc\td“)
abc     d
>>> print("abc\nd“)
abc
d
>>> print("abc\\nd“)
abc\nd
>>> print("ab\bc“)
ac
>>> print("\x41\xA1“)
Aí
```

Strings

- Constantes string podem ser escritas com várias linhas desde que as aspas não sejam fechadas e que cada linha termine com uma barra invertida
- Ex.:

```
>>> print ("abcd\n\
... efgh\n\
... ijk“)
abcd
efgh
ijk
>>> print ("abcd\
... efgh\
... ijk“)
abcdefghijk
>>>
```

Strings

- Também é possível escrever constantes string em várias linhas incluindo as quebras de linha usando três aspas como delimitadores
- Ex.:

```
>>> print( """
Um tigre
dois tigres
três tigres""")

Um tigre
dois tigres
três tigres
>>> print ('''abcd
efgh''')
abcd
efgh
```

Strings – Índices

- Endereçam caracteres individuais de uma string
  - Notação: *string*[*índice*]
  - O primeiro caractere tem índice 0
  - O último caractere tem índice -1
  - Ex.:

```
>>> a = "abcde"
>>> a[0]
'a'
>>> a[-1]
'e'
```

Expressões booleanas

- Também chamadas expressões lógicas
- Resultam em verdadeiro (True) ou falso (False)
- São usadas em comandos condicionais e de repetição
- Servem para analisar o estado de uma computação e permitir escolher o próximo passo
- Operadores mais usados
  - Relacionais: > , < , ==, !=, >=, <=
  - Booleanos: and, or, not
- Avaliação feita em “Curto-circuito”
  - Expressão avaliada da esquerda para a direita
  - Se o resultado (verdadeiro ou falso) puder ser determinado sem avaliar o restante, este é retornado imediatamente

Expressões booleanas


```
>>> 1==1
True
>>> 1==2
False
>>> 1==1 or 1==2
True
>>> 1==1 and 1==2
False
>>> 1<2 and 2<3
True
>>> not 1<2
False
>>> not 1<2 or 2<3
True
>>> not (1<2 or 2<3)
False
>>> "alo" and 1
1
>>> "alo" or 1
'alo'
```

Expressões booleanas

- As constantes **True** e **False** são apenas símbolos convenientes
- Qualquer valor não nulo é visto como verdadeiro enquanto que **0** (ou **False**) é visto como falso
- O operador **or** retorna o primeiro operando se for vista como *verdadeiro*, caso contrário retorna o Segundo
- O operador **and** retorna o primeiro operando se for vista como *falso*, caso contrário retorna o Segundo
- Operadores relacionais são avaliados antes de **not**, que é avaliado antes de **and**, que é avaliado antes de **or**

Expressões booleanas


```
>>> 0 or 100
100
>>> False or 100
100
>>> "abc" or 1
'abc'
>>> 1 and 2
2
>>> 0 and 3
0
>>> False and 3
False
>>> 1 and 2 or 3
2
>>> 0 or 2 and 3
3
>>> 1 and not 0
True
```

Funções Embutidas

- Além dos operadores, é possível usar funções para computar valores
- As funções podem ser definidas:
  - Pelo programador (veremos + tarde)
  - Em módulos da biblioteca padrão
  - Por *default:* são as funções *embutidas* (*built-in*)
    - Na verdade, fazem parte do módulo __builtins__, que é sempre importado em toda aplicação
- Ex.:
  - abs(*x*) retorna o valor absoluto do número *x*
  - chr(*x*) retorna uma string com um único caractere cujo código ASCII é *x*
  - ord(*s*) retorna o código ASCII do caractere *s*

Funções Embutidas


```
>>> abs (10)
10
>>> abs (-19)
19
>>> chr (95)
'_'
>>> chr (99)
'c'
>>> ord ('a’)
97
```

Conversões

- *Existem funções que são utilizadas para retornar uma variável convertida para um tipo específico. Exemplos:*

```
>>> x=25
>>> str(x)  # retorna x convertido para string
'25'
>>> float(x) # retorna x convertido para float
25.0
>>> x='25'
>>> int(x)  # retorna x convertido para inteiro
25
>>> float(x) # retorna x convertido para float
25.0
>>>
```

Construções if

- *A construção if é utilizada para controle condicional e tem a seguinte sintaxe:*

```
if <condição>:
    <expressão 1>
elif <condição 2>:
    <expressão 2>
else:
    <expressão 3>
```

Operadores lógicos

- *Operadores lógicos:*

and - *retorna verdadeiro caso todas as entradas forem verdadeiro;*

or - *retorna verdadeiro caso uma das entradas for verdadeiro;*

not - *se a entrada for verdadeira passará a falsa e vice-versa;*

is - *retorna verdadeiro caso receba duas referências ao mesmo objeto (útil para comparar strings);*

in - *retorna verdadeiro caso receba uma entrada que é encontrada uma ou mais vezes em uma lista.*

- *Expressões condicionais:*

<variavel>=<valor1> if <condição> else <valor2>

FOR

- *Utilizado para percorrer listas ,sequencias e processar iteradores.*

- *Sintaxe:*

```
for <referência> in <sequencia>:
    <bolco de expressões>
    continue
    break
else:
    <bolco de expressões>
```

FOR

- *O laço não percorre somente sequencias estáticas, mas também sequencias geradas por iteradores. Exemplo:*

```
>>> lista=[1,2,3]
>>> for i in lista:
...         lista.append(len(lista)+1)
...         print (i )
...         if i>5:break
...
1
2
3
4
5
6
```

FOR

- *A função range(m, n, p) define uma sequencia de inteiros iniciada em m, menor do que n (ou maior, depende da direção da sequencia), a um passo p (que pode ser negativo). Exemplo:*

```
>>> for i in range(1,10,2):
...     Print(i )
...
1
3
5
7
9
```

WHILE

- *É utilizada para realizar iterações condicionais, onde não se sabe o momento em que as iterações terminam.*
- *Sintaxe:*

```
while<condição>:
    <bloco de expressões>
    continue
    break
else:
    <bloco de expressões>
```

- *Exemplo:*

```
>>> a=0
>>> while a<5:
...     print( a )
...     a+=2
...
0
2
4
```

Entrada de Dados


```
>>> idade = int(input('Digite sua idade:'))
Digite sua idade:10
>>> type(idade)
builtins.int
```

Exemplos

- x = int(raw_input("Digite um numero: ")) – versão 2.7
- x = int(input("Digite um numero: "))
- if x < 0:
- print(“Menor que Zero”)
- elif x == 0:
- print(“Zero”
- elif x == 1:
- print(“Um”)
- else:
- print(“Outros”)

Exemplos

- **Laços de Repetição**
  - for i in range(10):
- print(i)

- for i, v in enumerate(['tic', 'tac', 'toe']):
- print(i,v)

Exemplos

- # Uma lista de instrumentos musicais
- instrumentos =[“Baixo”,”Bateria”,”Guitarra”]
- # Para cada nome na lista de instrumentos
- for i in instrumentos:
  - # mostre o nome do instrumento musical
- print(i)

Exemplos


```
# Para i na lista 234, 654, 378, 798:
for i in [234,654,378,798]:
# Se o resto dividindo por 3 for igual a zero:
    if i%3 == 0:
    # Imprime...
        print(i,'/ 3 =',i/3)
```

IF


```
temp = int(input('Entre com a temperatura: '))
if temp < 0:
    print ('Congelando...')
elif 0 <= temp <= 20:
    print ('Frio')
elif 21 <= temp <= 25:
    print('Normal')
elif 26 <= temp <= 35:
    print('Quente')
else:
    print('Muito quente!')
```

Símbolos da Interpolação

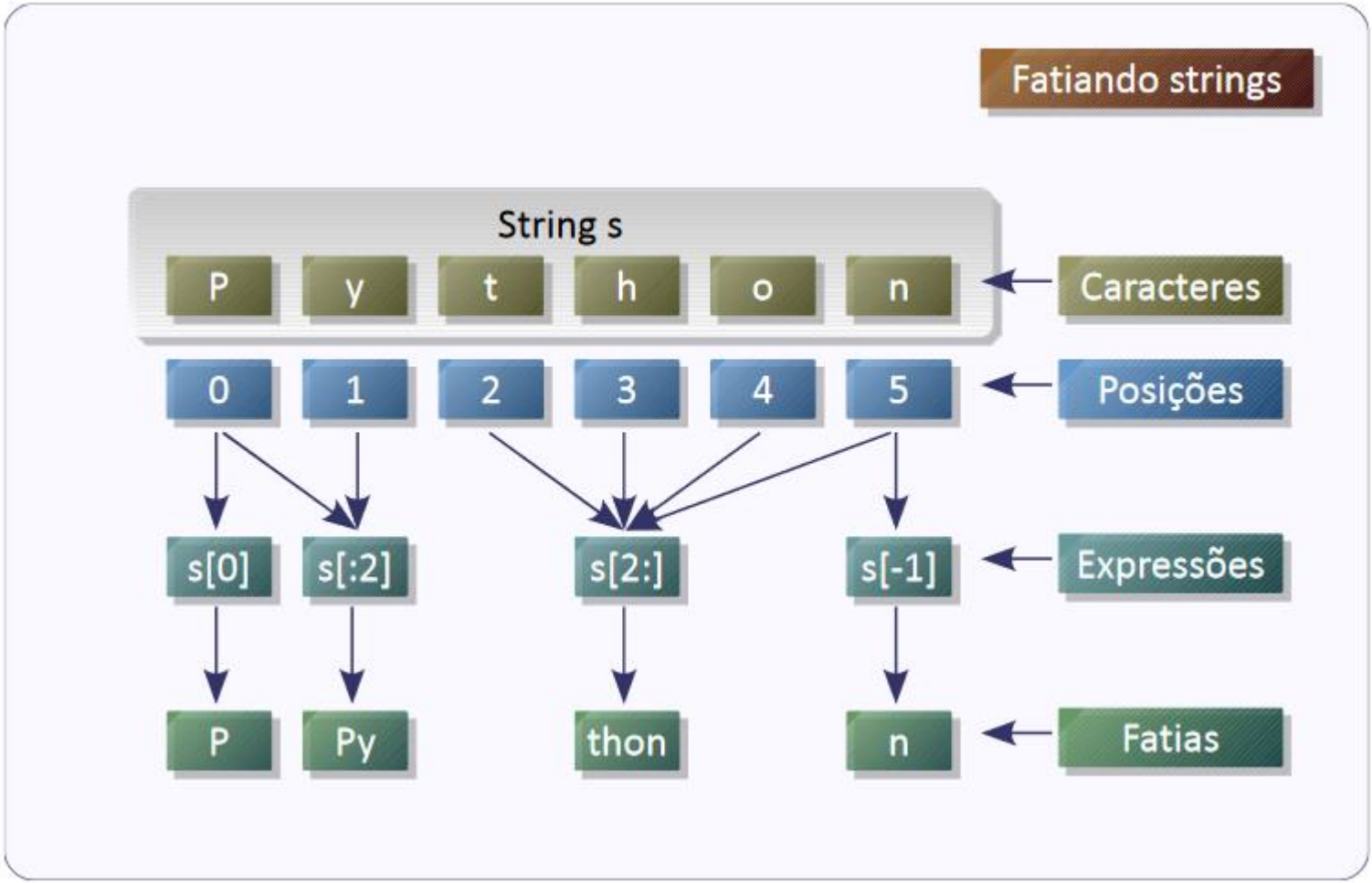
- Operador “%” é usado para fazer interpolação de strings
- A interpolação émais eficiente no uso de memória do que a concatenação convencional.
- Símbolos usados na interpolação:
- ▪
- %s: string.
- ▪%d: inteiro.
- ▪%o: octal.
- ▪%x: hexacimal.
- ▪%f: real.
- ▪%e: real exponencial.
- ▪%%: sinal de percentagem.

- # -*- coding: latin1 -*-
- # Zeros a esquerda
- print ('Agora são %02d:%02d.' %(6,30))

- # Real (número após o ponto controla as casas decimais)
- print('Percentagem: %.1f%%, Exponencial:%.2e' %(5.333,0.00314))

- # Octal e hexadecimal
- print('Decimal: %d, Octal: %o, Hexadecimal: %x' % (10,10,10))

Slices - Strings



Índices

- Os índices no Python:
- ▪
  - Começam em zero.
- ▪
  - Contam a partir do fim se forem negativos.
- ▪
  - Podem ser definidos como trechos, na forma [inicio:fim + 1:intervalo].

Se não for definido o inicio, será considerado como zero. Se não for definido o fim + 1, será considerado o tamanho do objeto. O intervalo (entre os caracteres), se não for definido, será 1.

Exercícios

1. Escreva um script que realize a leitura de 2 números e apresenta as 4 operações básicas (Soma, Subtração, Multiplicação, Divisão).
2. Faça um programa que leia as idades de 4 pessoas e calcule informe a média das idades.
3. Faça um programa para informar se o número lido é um número perfeito. Um número perfeito é igual a soma dos seus divisores. Por exemplo 28 = 14+7+4+2+1
4. Construa um algoritmo que leia cinco números inteiros e identifique o maior e o menor.
5. Utilizando o **FOR**, faça um algoritmo que leia a idade de 5 pessoas e escreva quantas delas possuem idade igual ou superior a 18 anos.
6. Faça um algoritmo que lê 4 valores e apresente a média dos pares.
7. Faça um algoritmo para converter um número decimal para binário.

Exercícios

8) Tendo como dados de entrada a altura de uma pessoa, construa um algoritmo que calcule seu peso ideal, usando a seguinte fórmula: (72.7*altura) - 58

Tendo como dados de entrada a altura e o sexo de uma pessoa, construa um algoritmo que calcule seu peso ideal, utilizando as seguintes fórmulas:

Para homens: (72.7*h) - 58

Para mulheres: (62.1*h) - 44.7 (h = altura)

Peça o peso da pessoa e informe se ela está dentro, acima ou abaixo do peso

9) João Papo-de-Pescador, homem de bem, comprou um microcomputador para controlar o rendimento diário de seu trabalho. Toda vez que ele traz um peso de peixes maior que o estabelecido pelo regulamento de pesca do estado de São Paulo (50 quilos) deve pagar uma multa de R$ 4,00 por quilo excedente. João precisa que você faça um programa que leia a variável peso (peso de peixes) e verifique se há excesso. Se houver, gravar na variável excesso e na variável multa o valor da multa que João deverá pagar. Caso contrário mostrar tais variáveis com o conteúdo ZERO.

Exercícios

10) Faça um Programa que pergunte quanto você ganha por hora e o número de horas trabalhadas no mês. Calcule e mostre o total do seu salário no referido mês, sabendo-se que são descontados 11% para o Imposto de Renda, 8% para o INSS e 5% para o sindicato, faça um programa que nos dê:

salário bruto.

quanto pagou ao INSS.

quanto pagou ao sindicato.

o salário líquido.

calcule os descontos e o salário líquido, conforme a tabela abaixo:

+ Salário Bruto : R$

- IR (11%) : R$

- INSS (8%) : R$

- Sindicato (5%) : R$

= Salário Liquido : R$

Obs.: Salário Bruto - Descontos = Salário Líquido.

Instalação

- www.python.org
- No Ubuntu
- sudo apt-get install python-tk

Referências:

[1] Curso de Python – Gustavo Noronha Silva, Fórum Mineiro de Software Livre

[2] Python para desenvolvedores – Luis Eduardo Borges

[3] Computação Gráfica em Python – Luis Eduardo Borges

[4] Tutorial Python, Release 2.4.2 – Guido Van Rossum , Fred L. Drake, Jr.

[5] Por que as pessoas usam... Python? - Emerson Henrique, Thiago Paiva, I Jornada de Iniciação Científica da ASPER - Ciências da Computação e Processamento de Dados