python

Fábio Rodrigues Jorge

E-mail: fabinhojorgenet@gmail.com

Objetivo do Minicurso

Este minicurso tem por objetivo apresentar uma noção introdutória á linguagem Python.

Temas abordados no minicurso:

-variáveis

-comandos de decisão

-laços

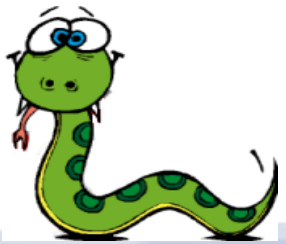
-Try e Except

-métodos (funções)

História do PYTHON

Python é uma linguagem de computação de altíssimo nível criado pelo holandês Guido Van Rossum sob o ideal de “Programação de Computadores para todos”.

Esse ideal fez com que o desenvolvimento em Python tivesse sempre em mente o código aberto, disponibilidade em multiplataformas e principalmente a clareza da sintaxe.



Python é uma linguagem orientada a Objetos, um paradigma que facilita entre outras coisas o controle sobre a estabilidade dos projetos quando estes começam a tomar grandes proporções.

Porém Python também possui total versatilidade permitindo que o usuário programe na forma procedural se desejar.

Vantagens

-Permite tanto programação Orientada a Objeto quanto programação Procedural.

-Fácil e rápido aprendizado

-Linguagem Simples e Elegante

-Python incentiva você a escrever seus programas de maneira correta

-Linguagem altamente modular, ou seja, muito do que você precisa para o seu programa já foi escrito nos módulos de Python aumentando ainda mais a velocidade no desenvolvimento.

Estas e outras vantagens fizeram que grandes empresas e universidades começassem a usar Python com sucesso. Entre elas temos a Philips, Industrial Light and Magic (Empresa de George Lucas), que usou Python para controlar efeitos especiais de Star Wars, a Nasa, a Aliança Espacial Universal (USA), a Nokia que usa o Python em seus celulares, a Disney, os sites do Google e Yahoo, entre outros.

-Blender
-Gimp
-Ubuntu

Mais empresas que usam Python:
http://www.python.org.br/wiki/EmpresasPython

Instalando o PYTHON

Windows

É necessário o download: **http://www.python.org/getit/**

O próprio site sugere a versão 2.7.2 do Python. Isso porque vários tutoriais e módulos pela internet utilizam essa versão, onde pode se encontrar alguma incompatibilidade com a versão 3.x do Python.

Após a instalação, coloque o caminho de onde o Python está instalado na variável de Sistema “PATH”.

Instalando o PYTHON

Linux

A maioria dos linux possui o Interpretador Python ja instalado.

Para utilizar o Python na plataforma Linux basta abrir o terminal e digitar “python”

A versão default do Python muda para cada plataforma.

Bom, então podemos começar . . .

Python é uma linguagem dinamicamente tipada. Isso quer dizer que não é necessário tipar as variáveis para usa-las. Seu tipo é modificado dependendo do tipo de seu valor.

Para se atribuir o valor 2 para a variável "var" apenas precisamos fazer:

```
>>> var = 2
```

O valor 2 é inteiro, então a variável “var” é uma variável inteira.

Por via das duvidas, para se saber de que tipo é a variável utilizamos a função "type ([variável])“.

```
>>> var = 2
>>> type(var)
<type 'int'>
>>>
```

Tipos de variáveis em Python

Os principais tipos de variáveis em Python são:

-inteiros (int)

-floats (float)

-strings (str)

-listas (list)

-tuplas (tuple)

-dicionários (dic)

Veremos mais detalhadamente a seguir.

Operadores

Operadores Algebricos:

+, -, *,/ e %
** (expoente 2**3 == 8)

Operadores Booleanos:

== (igualdade)
!= ou <> (diferença)
> e < (maior e menor)
>= e <= (maior/menor e igual)

Operadores

Operadores Lógicos:

and (E)
or (OU)
not([expressão]) -> inverte o resultado de um and ou or.
Ex:
 not(1 and 1)
 saida -> False

Operador de Atribuição:

=

Comentário:

(simples linha)
‘’’ (multiplas linhas)
‘’’ (multiplas linhas)

Algumas Funções Usuais

raw_input() -> entrada crua de informação (por default é String)
Ex: var1 = raw_input("Informe o valor: ")
caso queira transformar em outro tipo use as seguintes funções:

float(), int(), str(), list() -> cast de tipo
Ex:
var1 = int(raw_input("Informe o valor: "))
aux = str(raw_input("Informe o valor: "))

print "" -> Imprime uma String

```
>>> var = "e ae!"
>>> print "oi"
oi
>>> print var
e ae!
```

Usando o Interpretador

Faça alguns testes com esses valores:

```
a = 5          (Atribua o valor a variável a)
b = 4          (Atribua o valor a variável b)

#A seguir teste o resultado para essas operações:

a == b
a != b
a > b

3*a > 2*b
4*a != 5*b

3*a+2*b
b*b
b**2
a+b
```

Print com Formatação

Ele pode ser usado imprimindo uma string diretamente:

```
>>> print "Ola Mundo!"
Ola Mundo!
```

Podemos utilizar o print com várias concatenações também:

```
>>> var = "Mundo"
>>> print "Ola "+var+"!"
Ola Mundo!
>>>
```

Formatar a saída da string (igual fazemos em C):

```
>>> var = 1.1
>>> print "valor: %f" % (var)
valor: 1.100000
>>> print "valor: %2.2f" % (var)
valor: 1.10
```

Exemplo de Script em Python

Esse código lê um valor, converte ele para float e em seguida faz o calculo dele em metrôs.

```
>>>metros = float(raw_input("Informe o valor em metros: "))
>>>print "\n\t"+str(metros)+"m = "+str(metros*100)+"cm"
```

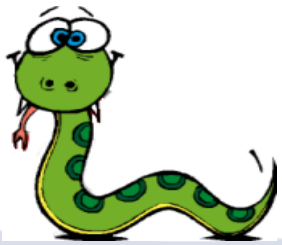
Alguns Enunciados de problemas

1. Faça um Programa que mostre a mensagem "Alo mundo" na tela.

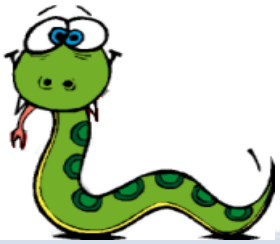
2.Faça um Programa que peça um número e então mostre a mensagem *O número informado foi [número]*.

3. Faça um Programa que peça dois números e imprima a soma.

4. Faça um Programa que peça a temperatura em graus Farenheit, transforme e mostre a temperatura em graus Celsius.
C = (5 * (F-32) / 9).



Variáveis



Strings

Strings são objetos em Python utilizados para se trabalhar com textos. São sequências de caracteres onde o usuário pode ter total acesso à qualquer posição dessa sequência.
Exemplo:

```
>>> texto = "paralelepípedo"
>>> print texto
paralelepípedo
>>> print texto[0]
p
>>> print 2*texto[2]          #2 vezes uma letra
rr
>>>
```

Strings

Em Strings você também pode trabalhar com um intervalo dentro de uma sequência:

```
>>> texto = "paralelepípedo"
>>> print texto
paralelepípedo
>>> print texto[0:5]
paral
>>>
```

Também usando um intervalo porém informando só o ponto inicial ou final:

```
>>> print texto[5:]       #da posição 5 em diante
elepípedo
>>> print texto[:5]       #da posição 0 até antes da posição 5
paral
>>>
```

Strings

Pegaremos agora outra palavra:

```
>>> texto = "arara"
>>> print texto [0:4]
arar
>>> print texto[0:4:2]
aa
>>>
```

A variável texto foi impressa utilizando o comando "print texto [0:4]". Quando inserimos o valor ":2" logo depois da sequência de 0:4 estamos dizendo que queremos os valores des da posição 0 até antes da posição 4 com incremento 2.

Strings

Se podemos incrementar também podemos decrementar! Veja:

```
>>> texto = "livro"
>>> print texto
livro
>>> print texto[4:0:-1]        #da sequência 4 ate 0
decrementando 1 porém não encontrando o primeiro zero
orvi
>>> print texto[4::-1]         #da sequencia 4 até o começo
decrementando 1
orvil
>>> print texto[::-1]          #do fim até o começo
decrementando 1
orvil
>>>
```

Listas

Um dos objetos mais utilizados em Python é a Lista(list).Sintaxe:

```
>>> lista = [1,2,3]
>>> print lista
[1,2,3]
>>>
```

A lista anterior é uma lista de inteiros porém ela pode ser de qualquer tipo ex:

```
>>> lista = [1.2 , 5.2]                #uma lista de floats
>>> lista = ["ola", "mundo", "!"]       # uma lista de strings
>>> lista = [[1,2,3], [1,2,3], [1,2,3]]       #uma lista de lista, mais conhecido como matriz 3x3
>>> lista = [1.1, "uhuuu", [5,5,5]]       #uma lista de vários tipos
```

Listas

Uma função que podemos executar sobre qualquer sequência em Python é a len([variável]).
Ela nos mostra o tamanho da sequência.

```
>>> lista = [1,2,3,4,5]
>>> print len(lista)
5
>>>
```

Listas possuem posições iguais as das Strings, porém diferentes delas, essas posições podem alteradas a qualquer momento:

```
>>> lista = [1,2,3,4]
>>> lista[0] = 5
>>> print lista
[5,2,3,4]
>>>
```

Listas

Trabalhando com matrizes

```
>>> lista1 = [1,2,3]
>>> lista2 = [6,5,5]
>>> lista3 = [8,8,8]
>>> matriz = [lista1, lista2, lista3]
>>> print matriz
[[1,2,3], [6,5,5],[8,8,8]]
>>> print matriz[0]          # matriz[0] é igual a lista1, como se
fossem a mesma váriavel
[1,2,3]
>>> print matriz[0][1]
2
>>> print lista1[1]
2
>>> print [1,2,3][1]
2
```

Manipulação de Listas

Append()

A função append adiciona um valor ao final da lista:

```
>>> l = [1,2,3]
>>> print l
[1,2,3]
>>> l.append(9)
>>> print l
[1,2,3,9]
>>> l.append("oi")
>>> print l
[1,2,3,9,"oi"]
>>>
```

Manipulação de Listas

insert()

Diferente do append que insere no final da lista apenas, o insert você consegue inserir em qualquer lugar passando como parâmetro o índice e o valor.

```
>>> l = [1,2,3]
>>> l.insert(0,9)
>>> print l
[9,1,2,3]
>>> l.insert(0,"ola Mundo!")
>>> print l
["ola Mundo!", 9,1,2,3]
>>>
```

Manipulação de Listas

remove()

Passando como parâmetro o valor que deseja remover.

```
>>> l = [1,2,3]
>>> print l
[1,2,3]
>>> l.remove(2)
>>>print l
[1,3]
>>> l.remove(9)
< erro >
>>>
```

Manipulação de Listas

pop()

Ela remove e retorna o ultimo valor da lista. Para quem sabe pilha e teve dificuldade de implementar os métodos push e pop, saibam que em Python já estão implementados! *push=append, pop=pop*

```
>>> l = [1,2,3]
>>> print l.pop()
3
>>> print l.pop()
2
>>> print l
[1]
>>>
```

Tuplas

As tuplas são objetos iguais as listas com a diferença de que tuplas são imutáveis. Uma vez criadas não podem ser modificadas.

Sintaxe:

```
>>> t = (1,2,3)
>>> print t
(1,2,3)
>>> t[0] = 5
< erro >
>>> print t[0]
1
>>>
```

Tuplas

Com as tuplas podemos fazer uma manipulação diferente das variáveis. Essa técnica é chamada de "packing-unpacking".

```
>>> (a,b) = ("ola", 5.734)
>>> print a
ola
>>>print b
5.734
>>> a,b = "ola", 5.734          #pode se usar sem parênteses
```

O código acima atribuiu "ola" para *a* e 5.734 para *b*.
Podemos facilmente trocar os valores entre duas variáveis:

```
>>> a = 5
>>> b = 1
>>> a,b = b,a
>>> print a
1
>>> print b
5
```

Dicionários

Dicionários não são sequências iguais strings e listas. Sua forma de indexação é diferente, usando chaves para acessar valores.

```
>>> d = {"um":"valor 1",2:"doisssss", "nome":"Fábio"}
>>> print d
{"um":"valor 1",2:"doisssss", "nome":"Fábio" }
>>>print d["um"]
valor1
>>> print d["nome"]
Fábio
>>> print d[2]
doisssss
>>> print d[“um"]+" tem "+str(d[2])+" "+d[“nome"]
valor1 tem doisssss Fábios
>>> d["nome"] = "Alguém"
>>> print d["nome"]
Alguém
>>>
```

Dicionários

Para saber quais são as chaves de um dicionário utilizamos o método keys() do objeto dicionário. Também podemos verificas se o dicionário possui alguma chave pelo método has_key([chave]):

```
>>> d = {"um":"valor 1",2:"doisssss", "nome":"Fábio",
"animal":"texugos" }
>>> print d.keys()
["um", 2, "nome", "animal"]
>>> print d.has_key("quarenta")
False
>>> print d.has_key(2)
True
>>>
```

If ... else ... elif

Antes de começarmos vamos falar

um pouco sobre blocos de códigos

Diferente de outras linguagens de programação que formam blocos de código usando begin / end ou { / }, o Python reconhece seus blocos através da tabulação (identação) do código. Essa identação forçada do código faz com que no final do seu programa você possua um código limpo, tornando-o mais fácil de entender.
Exemplo:

```
>>>if(condição):
>>>    código
>>>    código
>>>Continuação do programa
```

If ... else ... elif

A sintaxe da estrutura de decisão é a seguinte:

```
>>> if [condição] :
>>>     [código]
>>>     [código]
>>> elif [condição]:
>>>     [código]
>>>else:
>>>     [código]
>>>
```

If ... else ... elif

Exemplo:

```
>>> a = 10
>>> if a == 10:
>>>     print "a eh igual a 10"
>>> else:
>>>     print "a eh diferente de 10“
>>>
a eh igual a 10
>>>
```

Alguns Enunciados de problemas

1. Faça um Programa que peça dois números e imprima o maior deles

2. Faça um Programa que peça um valor e mostre na tela se o valor é positivo ou negativo

Estruturas de Repetição

(Laços)

While (Enquanto)

Estruturas de Repetição (Loop) permitem repetir um trecho do código até que a condição passada seja satisfeita.

A estrutura do "While" é a seguinte:

```
>>>
>>> While [condição] :
>>>         código
>>>         código
>>>         código
>>>
```

Obs: não esqueça de incrementar!! -> var += 1

While (Enquanto)

Enquanto a condição for verdadeira o código após os " : (dois pontos)" é repedido N vezes.

Exemplo:

```
>>> a = 0
>>> while a < 5:
>>>     print a
>>>     a = a + 1
>>>
0
1
2
3
4
>>>
```

For(para)

Diferente de outras linguagens de programação, o laço For em Python tem uma funcionamento diferente.

Em vez de repetir até que sua condição seja alcançada, como no While, o laço For executa percorrendo Estruturas de Sequencia (String, List e Tuplas) e sua condição de parada é o tamanho da sequencia.

Imagine a sequencia: [1,2,1,8]
Nesse caso o laço for repetiria o código que está em seu bloco 4 vezes.

For(para)

A estrutura do FOR:

```
>>>
>>> for [variável] in [variável sequencia] :
>>>     [código]
>>>     [código]
>>>     [código]
>>>
```

For(para)

A variável citada no exemplo acima percorre a variável sequencia recebendo o valor daquela posição. Para melhor entender veja o exemplo abaixo:

```
>>>
>>> l = [1,2,1,8]
>>> for i in l:
>>>     print "valor: "+str( i )
>>>
valor: 1
valor: 2
valor: 1
valor: 8
```

For(para)

```
>>> s = "Hello"
>>>for i in s:
>>>     print "valor: "+str( i )
>>>
valor: H
valor: e
valor: l
valor: l
valor: o
>>>
```

No exemplo acima, a variável " i " percorre a variável sequencial pegando o valor de cada posição e imprimindo-o.

For(para)

O For também percorre a estrutura Dicionário, porém de um modo diferente das outras.
A variável recebe o valor da chave do dicionário.

Exemplo:

```
>>>
>>> a = {22:"uhu", 123:"xx", 57:5777}
>>> for t in a:
>>>     print t
>>>
22
```

For(para)

Existe uma função que cria uma lista do valor "0" até o valor que foi passado como parâmetro. Essa função é o " range() ".

Exemplo:

```
>>> l = range(5)
>>> print l
[0,1,2,3,4]
>>>
>>> for aux in range(5):
>>>     print aux
>>>
0
1
2
3
4
>>>
```

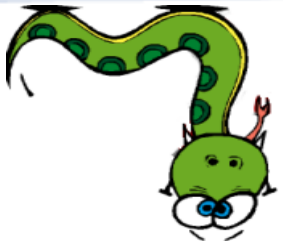
Os comandos Break e Exit

O comando Break força a interrupção do laço, não precisando esperar pela condição

Exemplo:

```
>>> a = 0
>>> while a < 5:
>>>     print "entrou no while"
>>>     break
>>>     print "passou pelo break"
>>>
entrou no while
>>>
```

A função “ exit() ” finaliza a aplicação.



Try e Except

Try e Except

O Try/Except é uma estrutura que permite em tempo de execução o código ser alterado.

Se o código contido dentro do bloco Try retornar algum erro ou “failure”, o código que esta no bloco do “Except” é executado.

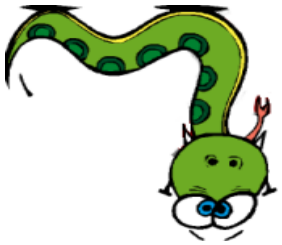
Sintaxe:

```
>>>try:
>>>    [código]
>>>except:
>>>    [código]
>>>
```

Try e Except

Exemplo:

```
>>> try:
>>>     x = 0
>>>     y = 5
>>>     var = y/x
>>>     print “try”
>>> except:
>>>     print “falhou”
>>>
```



Funções/Métodos

Funções

Funções normalmente são utilizadas para executar um bloco de código que vai ser repetido várias vezes, economizando linhas e deixando o código mais organizado.

Sintaxe:

```
>>> def [nome da função] (parametro1, parametro2, ...) :
>>>         [operações]
>>>         [operações]
>>>         [operações]
>>>
```

Funções

Vamos imaginar que seja necessário calcular o quadrado de um número várias vezes ao longo do programa.
Para facilitar criaremos a função “quad()”

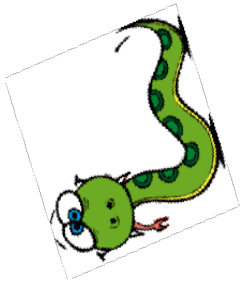
```
>>> def quad( var ):
>>>         return x**2
>>>
>>> print quad(2)
4
>>>
```

Funções

Existe um modo de trabalharmos com funções que aceitem N argumentos

Exemplo:

```
>>> def imprime(*var):
>>>       return var
>>>
>>> print imprime(1,2,3,”ola”)
(1,2,3,”ola”)
>>>
```



Avançando um pouco...

List comprehensions

É um recurso de tabalho com funções e sequências muito caracteristico do Python.

Esse recurso permite a geração de uma list a partir de uma função simples.

Ela se baseia em um for preenchendo o interior de uma “list”.

Exemplo:

```
>>> print [1 for i in range(5)]
[1,1,1,1,1]
>>> print [i for i in range(5)]
[0,1,2,3,4]
>>> print [x*x for x in range(5)]
[0,1,4,9,16]
```

Bom, e é isso!

Fábio Rodrigues Jorge
Graduando de Ciência da Computação – UNIVEM

Email: fabinhojorgenet@gmail.com

Lab: http://compsi.univem.edu.br/