





Apresentação

Aluno do 4º semestre da graduação em Ciência da Computação da UECE.

Presidente da empresa jr. da Computação da UECE – Acens

Fábio Cerqueira (fabio@acens.com.br)



Objetivo do Curso

- Mostrar uma “nova” e ótima opções de linguagens para desenvolvimento;
- Apresentar a linguagem Python, suas facilidades e sua produtividade;
- Realizar práticas para desenvolver a maneira de pensar com Python.



O que você já conhece?





Porque Python?

Versão do primeiro programa em C:

```
#include <stdio.h>

int main() {
    printf("Hello World!");
    return 0;
}
```

Versão do primeiro programa em Python:

```
print "Hello World!"
```

"Python é simples e correta."



Quem usa Python?



Eu \o/



Para que se usa Python?

- Educação;
- Desenvolvimento Web;
- Desktop GUIs;
- Acesso a Bases de Dados;
- Computação Numérica e Científica;
- Programação em Rede;
- Jogos e Gráficos 3D;



História do Python



- Criada no natal de 1989;
- Guido Van Rossum;
- Nome não veio da cobra. (Monty Python)
- Influências de ABC, Haskell, C, Perl, SmallTalk, Modula 3;
- Hoje Guido Van Rossum trabalha na Google.



Características do Python

- Interpretada;
- Portável(Multi-plataforma);
- Extensível (C, Java, .NET);
- Livre;
- Tudo é objeto em Python;
- Multiparadigma: Procedural, Orientada a objetos, Funcional;
- Case-sensitive;



Características do Python II

- Simples e Legível;
- Suporte nativo a estrutura de dados de alto nível;
- Sem declaração de variáveis;
- Tipagem Forte e Dinâmica;
- Controle de escopo por indentação.



Download e instalação do Python

- Download no site oficial: <http://www.python.org/download/>
- Versão usada no curso: 2.5 (python-2.5.msi)
- Versão em desenvolvimento: beta 2.6 e 3.0

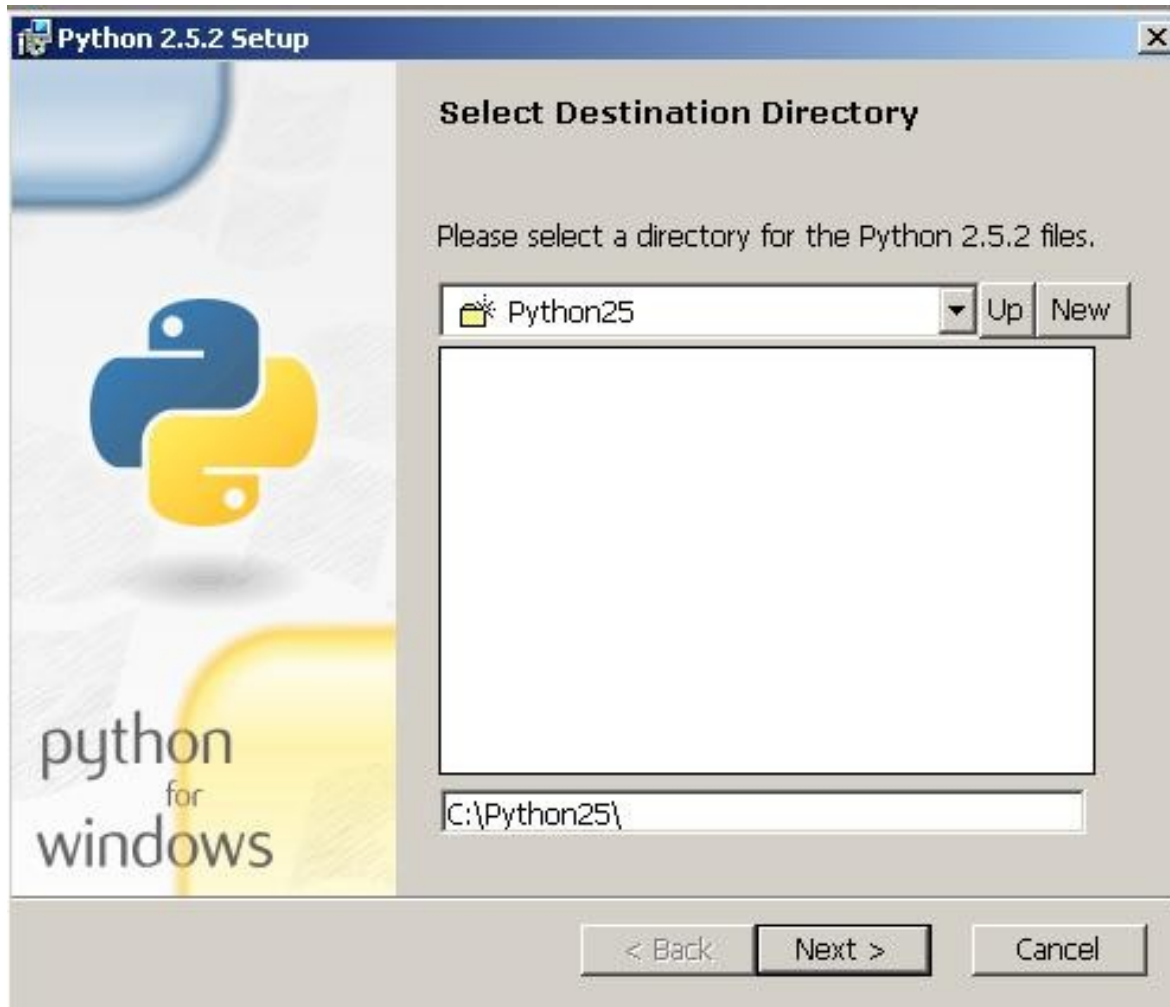


Instalação do Python



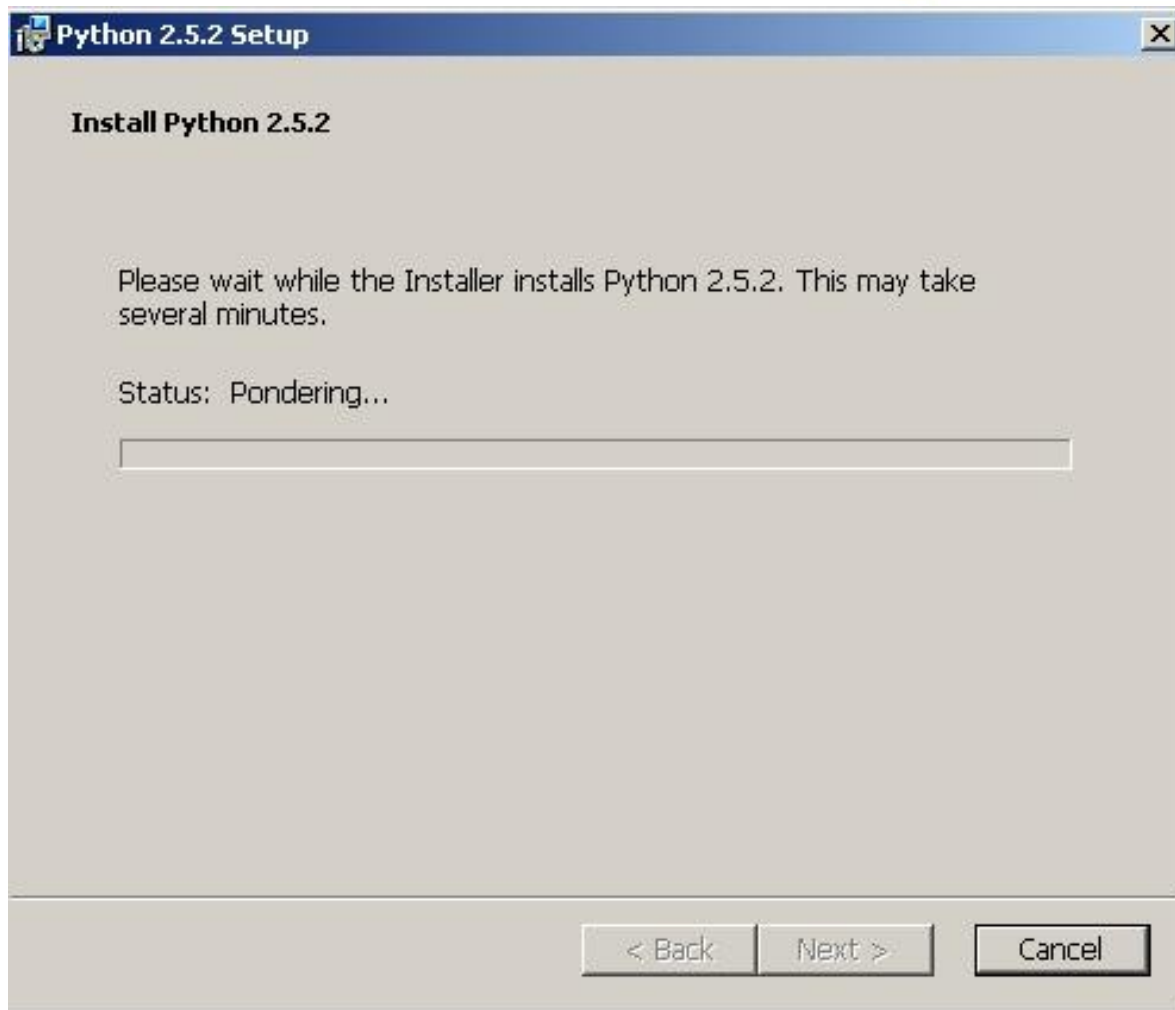


Instalação do Python





Instalação do Python





O Shell interativo

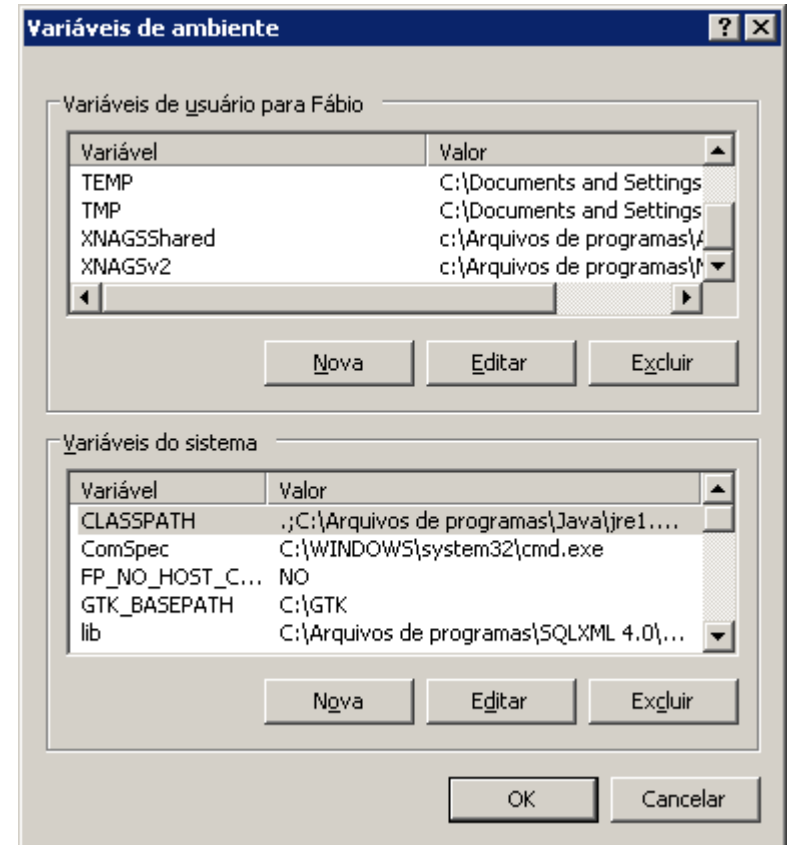
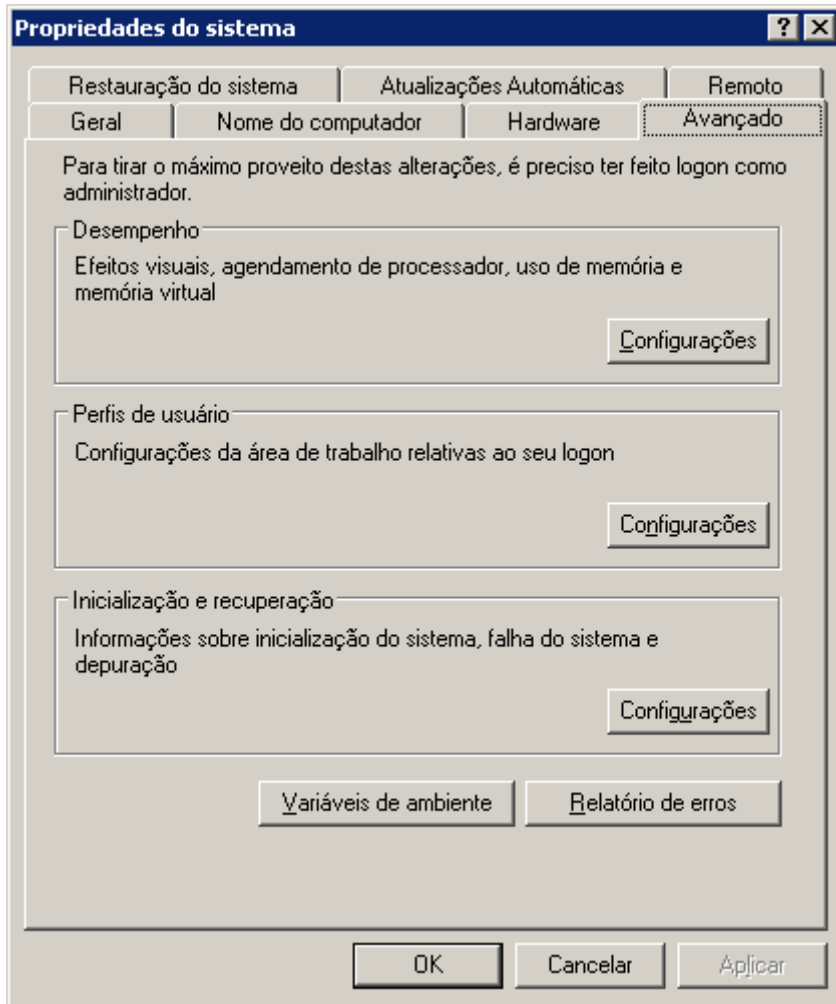
A screenshot of a Windows command prompt window. The title bar reads "C:\WINDOWS\system32\cmd.exe - python". The window content shows the command "python" being executed, followed by the Python 2.5.2 startup banner: "Python 2.5.2 (r252:60911, Feb 21 2008, 13:11:45) [MSC v.1310 32 bit (Intel)] on win32". Below the banner, it says "Type 'help', 'copyright', 'credits' or 'license' for more information." and the interactive prompt ">>> _" is displayed.

```
C:\WINDOWS\system32\cmd.exe - python

C:\Documents and Settings\Fábio>python
Python 2.5.2 (r252:60911, Feb 21 2008, 13:11:45) [MSC v.1310 32 bit (Intel)] on win32
Type "help", "copyright", "credits" or "license" for more information.
>>> _
```



Configurando Windows





Hello World!

```
C:\WINDOWS\system32\cmd.exe - python

C:\Documents and Settings\Fábio>python
Python 2.5.2 (r252:60911, Feb 21 2008, 13:11:45) [MSC v.1310 32 bit (Intel)] on
win32
Type "help", "copyright", "credits" or "license" for more information.
>>> print "Hello World!"
Hello World!
>>> _
```

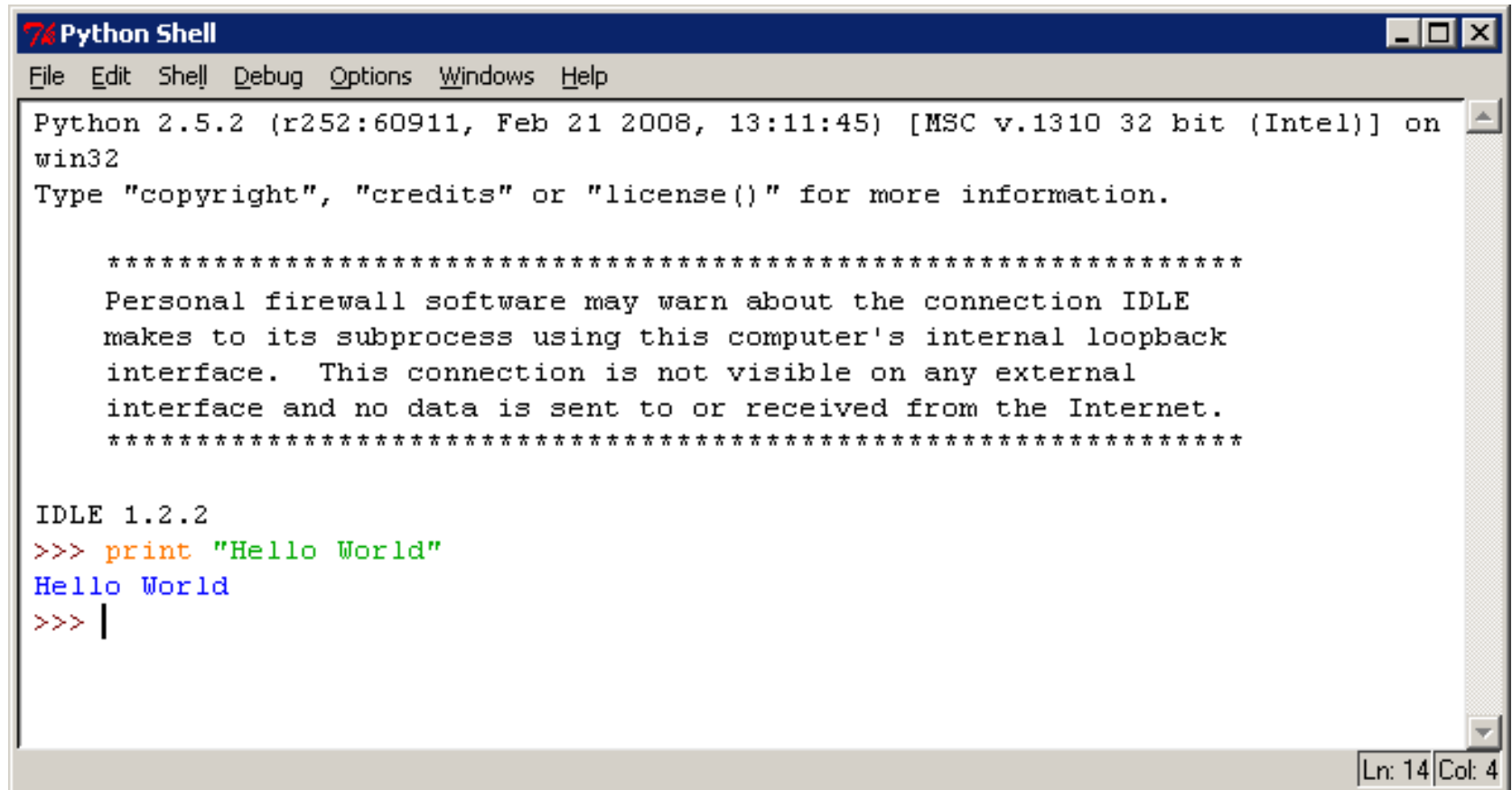


Usando o IDLE

- Opção para auto completar o código;
- Opção de debug;
- Sintaxe colorida;
- Editor além do Shell;
- Outras muitas opções;
- E ainda é mais bonito. =D



O IDLE



```
Python Shell
File Edit Shell Debug Options Windows Help

Python 2.5.2 (r252:60911, Feb 21 2008, 13:11:45) [MSC v.1310 32 bit (Intel)] on
win32
Type "copyright", "credits" or "license()" for more information.

*****
Personal firewall software may warn about the connection IDLE
makes to its subprocess using this computer's internal loopback
interface.  This connection is not visible on any external
interface and no data is sent to or received from the Internet.
*****

IDLE 1.2.2
>>> print "Hello World"
Hello World
>>> |

Ln: 14 Col: 4
```



“Tipos primitivos”

```
>>> #A linguagem possui vários tipos primitivos
>>> # int, float, bool, str, unicode, complex, tuple,
list, tuple, dict , set, frozenset
>>> # Quando uma variável é criada não muda mais de t
ipo(Fortemente tipada)
>>> x = 1
>>> print x
1
>>> type(x)
<type 'int'>
>>> y = 2.5
>>> type(y)
<type 'float'>
>>> z = (int)y
SyntaxError: invalid syntax
>>> #Não é possível realizar cast
>>> #Deve ser feita uma conversão
>>> z = int(y)
>>> print z
2
```



Variáveis

- Fortemente tipada;
- Tipagem dinâmica;
- Tudo é objeto;
- Não é necessário declarar.

```
>>> x = 5.7
>>> y = int(x)
>>> x
5.7000000000000002
>>> y
5
>>> type(x)
<type 'float'>
>>> type(y)
<type 'int'>
>>> z = x + y
>>> z
10.699999999999999
>>> print z
10.7
>>> type(z)
<type 'float'>
>>>
```



Conhecendo outros tipos

Conversão automática de int para long:

```
>>> x = 123561
>>> type(x)
<type 'int'>
>>> grande = x ** 12
>>> print grande
12664177051358979477135584432535550802333736729296303
230260321
>>> grande
12664177051358979477135584432535550802333736729296303
230260321L
>>> type(grande)
<type 'long'>
>>>
```



Conhecendo outros tipos

Tipo bool representa valores lógicos 0 e 1

```
>>> completo = False
>>> print completo
False
>>> completo = not completo
>>> print completo
True
>>> type(completo)
<type 'bool'>
```

O que é falso para Python?

```
>>> falso = bool('')
>>> falso
False
>>> falso = bool(0)
>>> falso
False
>>> falso = bool(())
>>> falso
False
>>> falso = bool([])
>>> falso
False
>>> falso = bool({})
>>> falso
False
```



Complex? 0.o

Python vem com o tipo para representar números complexos.

```
>>> c = 2+3j
>>> type(c)
<type 'complex'>
>>> k = 5-7j
>>> z = c + k
>>> print z
(7-4j)
>>> print z.conjugate
<built-in method conjugate of complex object at 0x009
6A2D8>
>>> #ops
>>> print z.conjugate()
(7+4j)
>>> print z.imag
-4.0
>>> print z.real
7.0
```




Tudo é objeto

As variáveis podem ser iniciadas da forma já mostrada ou com uso de construtores.

```
>>> a = int('1111',2)
>>> a
15
>>> type(a)
<type 'int'>
```

O construtor do tipo int possui opção de escolha da base.



Atribuição mutua

Troca-troca de valores.

```
>>> nome = 'Cerqueira'
>>> snome = 'Fábio'
>>> #ops.. digitei errado!
>>> #normalmente para fazer a troca entre variáveis é preciso usar uma auxiliar.
>>> aux = nome
>>> nome = snome
>>> snome = aux
>>> print nome, snome
Fábio Cerqueira
>>> #em Python você pode fazer isso:
>>> nome, snome = snome, nome
>>> print nome, snome
Cerqueira Fábio
```

A atribuição mutua é consequência da atribuição múltipla.



Strings. E o tipo char?

- É, Python não tem tipo char =/
- Isso é um problema?

```
>>> char = 'F'
>>> print char
F
>>> type(char)
<type 'str'>
>>> ord('F')
70
>>> ord("Acens")
```

- Aspa simples ou aspa dupla podem ser usadas nas strings.

```
>>> minhaString = "Acens"
>>> print minhaString
Acens
>>> minhaString = 'Acens'
>>> print minhaString
Acens
```



Operadores

Operadores aritméticos

```
>>> 1 + 1
2
>>> 5 - 2
3
>>> 7 * 8
56
>>> 5 / 2
2
>>> # 5 / 2 = 2 ? o.O
>>> 5 / 2.
2.5
>>> #Aaaahhh!!!
>>> 10 % 3
1
>>> 2 ** 4
16
>>> #Uia! Exponenciação nativa =D
```



Mais Operadores

Operadores de comparação

```
>>> 2 > 1
True
>>> 3 == 2
False
>>> 5 != 10
True
>>> 12 >= 12
True
>>> 17 <= 15
False
>>> #olha que interessante
>>> x = 10
>>> 5 < x < 20
True
>>> x = 2
>>> 5 < x < 20
False
```

Existem outros operadores usados para comparação, futuramente citarei mais alguns e outros você conhecerá com a prática.



Mais Operadores

Operadores lógicos

```
>>> a = 2
>>> b = 5
>>> c = 10
>>> a > b and c > a
False
>>> a / b == c or c == 10
True
>>> not a
False
>>> not(a == b or a+b > c)
True
```

Operadores lógicos são usados principalmente junto com estruturas de controle que veremos neste curso.



Mais Operadores

Operadores de bitwise

```
>>> print a,b
7 2
>>> #and
>>> a & b
2
>>> #2 binário é: 111 & 010 = 010
>>> #or
>>> a | b
7
>>> #xor
>>> a ^ b
5
>>> #complemento de 2
>>> ~a
-8
>>> #shift direita
>>> a >> 1
3
>>> #shift esquerda
>>> b << 1
4
```



Voltando às strings

Strings são seqüências(Tira ou não o trema?) especiais do python.

- Operações mágicas sobre string.(Slices)

```
>>> texto = "Teste de escrita!"
>>> texto[0]
'T'
>>> texto[12]
'r'
>>> texto[4:]
'e de escrita!'
>>> texto[:3]
'Tes'
>>> texto[4:10]
'e de e'
>>> texto[1::2]
'et eecia'
>>> texto[1::3]
'eesi!'
```

```
>>> texto[-1]
'!'
>>> texto[-6]
'c'
>>> texto[:-6]
'Teste de es'
>>> texto[-4:]
'ita!'
>>> texto[::-1]
'!atircse ed etseT'
```




Operadores para string

Python oferece alguns operadores para trabalhar com string.

```
>>> #Concatenação.. O operador +
>>> nome = pnome + ' de ' + snome + ' ' + tnome
>>> print nome
Fábio de Sousa Cerqueira
>>> #Substituição.. O operador %
>>> idade = 20
>>> info = 'Nome: %s \nIdade: %d' % (nome, idade)
>>> print info
Nome: Fábio de Sousa Cerqueira
Idade: 20
```



Operadores para string

```
>>> #Repetição.. O operador *
>>> print 'a' * 5
aaaaa
>>> print 'legal ' * 3
legal legal legal
>>> msg = '-' * 30 + '\n' + info + '\n' + '-'*30
>>> print msg
-----
Nome: Fábio de Sousa Cerqueira
Idade: 20
-----
```



Tipos especiais de string

Fugindo dos caracteres especiais.

```
>>> print 'c:\acens\testes'
c:\acens estes
>>> #repare que não foi impresso o esperado
>>> #Há dois modos de resolver isso.
>>> #Usando outra \
>>> print 'c:\\acens\\teste'
c:\acens\teste
>>> #A outra maneira elimina todos os especiais
>>> print r'c:\acens\teste'
c:\acens\teste
>>> type(r'ainda assim é string')
<type 'str'>
```

O **r** no início da string indica que os **\...** não serão interpretados como especiais.



Tipos especiais de string

São as strings de múltiplas linhas.

```
>>> hinodotimao = """Salve o Corinthians,
O campeão dos campeões,
Eternamente
Dentro dos nossos corações..."""
>>> print hinodotimao
Salve o Corinthians,
O campeão dos campeões,
Eternamente
Dentro dos nossos corações...
>>> hinodotimao
'Salve o Corinthians,\nO campe\xe3o dos campe\xf5es,\nEternamente\nDentro dos nossos cora\xe7\xf5es...'
>>> type(hinodotimao)
<type 'str'>
```

Inicia a string com `"""` e finaliza com `"""`



Tipos especiais de string

Tipo especial Unicode.

```
>>> #Unicode é um padrão para representar textos
de qualquer sistema de escrita
>>> #Em python usa-se u antes da string para repr
esentar
>>> texto = u'Meu texto unicode!'
>>> #aparentemente não muda muito, mas para manip
ulação de arquivos isso pode se tornar chato e até
mesmo complicado
>>> print texto
Meu texto unicode!
>>> texto
u'Meu texto unicode!'
>>> type(texto)
<type 'unicode'>
>>> #Repare que não é mais do tipo str, mas é uma
string.
```



Métodos do objeto str

```
>>> #para saber os métodos de um objeto usa-se o
comando dir()
>>> dir(str)
['__add__', '__class__', '__contains__', '__delat
tr__', '__doc__', '__eq__', '__ge__', '__getattri
bute__', '__getitem__', '__getnewargs__', '__gets
lice__', '__gt__', '__hash__', '__init__', '__le
__', '__len__', '__lt__', '__mod__', '__mul__', '__
ne__', '__new__', '__reduce__', '__reduce_ex__',
 '__repr__', '__rmod__', '__rmul__', '__setattr__',
 '__str__', 'capitalize', 'center', 'count', 'de
code', 'encode', 'endswith', 'expandtabs', 'find'
, 'index', 'isalnum', 'isalpha', 'isdigit', 'islo
wer', 'isspace', 'istitle', 'isupper', 'join', 'l
just', 'lower', 'lstrip', 'partition', 'replace',
'rfind', 'rindex', 'rjust', 'rpartition', 'rsplit
', 'rstrip', 'split', 'splitlines', 'startswith',
'strip', 'swapcase', 'title', 'translate', 'upper
', 'zfill']
```



Métodos do objeto str

Existem vários métodos para string. Abaixo alguns:

```
>>> texto = 'curso de python - fábio cerqueira'
>>> #O método capitalize coloca em maiúsculo a primeira letra.
>>> print texto
curso de python - fábio cerqueira
>>> #Método center centraliza uma string com espaços
>>> print 'teste'.center(20)
        teste
>>> print '*' + 'teste'.center(20) + '*'
*         teste          *
>>> nomedocurso, instrutor = texto.split('-')
>>> nomedocurso
'curso de python '
>>> instrutor
' f\xelbio cerqueira'
```



Documentação interativa

Python possui uma documentação interativa que ajuda bastante o programador. Como não mostrarei todos os métodos do tipo str veja uma maneira simples de aprender.

```
>>> help(str)
```

```
Help on class str in module __builtin__:
```

```
class str(basestring)
| str(object) -> string
|
| Return a nice string representation of the object.
|
| If the argument is a string, the return value is
the same object.
|
| Method resolution order:
|     str
|     basestring
|     object
|
| Methods defined here:
```

O comando help() mostra a documentação para o objeto passado como parâmetro.



Documentação interativa

Não é preciso acessar a documentação de toda a classe para estudar só um método.

```
>>> help(''.find)
```

```
Help on built-in function find:
```

```
find(...)
```

```
    S.find(sub [,start [,end]]) -> int
```

```
    Return the lowest index in S where substring sub
    is found,
```

```
    such that sub is contained within s[start:end].
```

```
Optional
```

```
    arguments start and end are interpreted as in slice
    notation.
```

```
    Return -1 on failure.
```

Documentação para o método find() da classe str



Comandos

Em Python existem comandos nativos bastante úteis, alguns já foram usados nesta aula.

O comando em python tem o seguinte formato:

```
nome ([parm1, parm3, parm4, parm5...])
```

```
>>> #len() - retorna o tamanho de uma sequência
```

```
>>> texto = 'UECE - Pública e de Qualidade.'
```

```
>>> len(texto)
```

```
30
```

```
>>> #help() - retorna o texto da docstring do parâmetro
```

```
>>> #dir() - retorna o métodos e atributos do parâmetro
```

```
>>> #abs() - retorna o valor absoluto de um número
```

```
>>> print abs(-13)
```

```
13
```



Mais comandos

```
>>> #len() - retorna o tamanho de uma sequência
>>> texto = 'UECE - Pública e de Qualidade.'
>>> len(texto)
30
>>> #help() - retorna o texto da docstring do parâmetro
>>> #dir() - retorna o métodos e atributos do parâmetro
>>> #abs() - retorna o valor absoluto de um número
>>> print abs(-13)
13
>>> all.__doc__
'all(iterable) -> bool\n\nReturn True if bool(x) is
True for all values x in the iterable.'
>>> #chr() - retorna o valor da tabela ASCII para o
número passado no parâmetro
>>> print chr(44)
','
>>> #ord() - faz o contrário do chr(). retorna o valor
do caractere passado como parâmetro
>>> ord(',')
44
```



Mais comandos

```
>>> #max() - retorna o maior item de uma sequência
>>> max('cbafed')
'f'
>>> #min() - retorna o menor item de uma sequência
>>> min('cbafed')
'a'
>>> #pow(x,y[,r]) - equivalente ao x ** y com dois
parâmetros, e (x ** y) % r com uso de três.
>>> pow(2,4)
16
>>> pow(2,4,5)
1
>>> #round() - retorna o número arredondado de acor
do com os parâmetros
>>> round(3.1415)
3.0
>>> round(3.1415,2)
3.1400000000000001
```

Existem outros comandos importantes que serão vistos durante o curso.



Entrada de dados

Em modo console para receber dados do usuário de forma interativa o Python aceita basicamente dois comandos. `input()` e `raw_input()`

```
>>> nome = raw_input("Qual o seu nome: ")
Qual o seu nome: Fábio Cerqueira
>>> idade = input("Qual a sua idade %s: " % nome.split(" ")[0])
Qual a sua idade Fábio: 20
```

Observe e diga porque em um deles foi usado `raw_input()` e no outro `input()`



raw_input ou input

Observe outro exemplo:

```
>>> cor = input("Qual a sua cor preferida: ")
Qual a sua cor preferida: verde
```

```
Traceback (most recent call last):
  File "<pyshell#16>", line 1, in <module>
    cor = input("Qual a sua cor preferida: ")
  File "<string>", line 1, in <module>
NameError: name 'verde' is not defined
```

Ao tentarmos receber uma string com input não foi possível salvar o dado na variável cor.

```
>>> verde = (0,255,0)
>>> cor = input("Qual a sua cor preferida: ")
Qual a sua cor preferida: verde
>>> cor
(0, 255, 0)
```

Note que o valor atribuído a cor foi o valor que havia sido salvo na variável verde.



raw_input ou input

O mesmo exemplo da cor com o comando `raw_input()`:

```
>>> cor = raw_input("Qual a sua cor preferida: ")
Qual a sua cor preferida: verde
>>> cor
'verde'
```

Note que o valor atribuído a `cor` foi exatamente a string digitada.

E se eu quiser que seja o valor da variável `verde`? O que fazer?

```
>>> verde = (0,255,0)
>>> cor = raw_input("Qual a sua cor preferida: ")
Qual a sua cor preferida: verde
>>> cor = eval(cor)
>>> cor
(0, 255, 0)
```

O comando `eval` faz a “mágica” \o



O comando eval()

Entendendo melhor o comando eval()

```
>>> eval('1+(3*2)')
7
>>> dobro = lambda x:2*x #Cria uma função lambda(nem ligue!)
>>> resul = raw_input("Digite a função: ")
Digite a função: dobro(3)
>>> resul
'dobro(3) '
>>> eval(resul)
6
```

O eval retorna o resultado da interpretação da string passada que representa uma expressão Python.

Para ver mais use a documentação interativa. `help(eval)`



“Sim... E aí?”

- Podemos chegar a conclusão que usar `input()` é o mesmo que usar `eval(raw_input());`
- Que `raw_input()` retorna uma string;
- E que o uso vai depender muito em que situação vai ser usada.



Controle de fluxo

Observe o exemplo:

```
#!/usr/bin/env python
#-*- coding: utf-8 -*-
nomecompleto = raw_input("Qual o seu nome: ")
nome,sobrenome = nomecompleto.split(" ")
idade = int(raw_input("Qual sua idade %s: " % nome))
localfamilia = raw_input("De onde é a família %s: " % sobrenome)
if idade < 18:
    maior = False
    print "Você não é de maior."
else:
    maior = True
    print "Você é de maior."
```

SAÍDA:

```
>pythonw -u "exemplo.py"
Qual o seu nome: Fábio Cerqueira
Qual sua idade Fábio: 20
De onde é a família Cerqueira: Piauí
Você é de maior.
>Exit code: 0
```



if e else em C

```
#include <stdio.h>

int main() {
    int maior = 0, idade = 0;
    printf("Qual a sua idade: ");
    scanf("%d", &idade);
    if (idade < 18) {
        printf("Voce nao eh de maior.\n");
        maior = 0;
    }
    else {
        printf("Voce eh de maior.\n");
        maior = 1;
    }
    return 0;
}
```



Controle de escopo

Observe que o código em python não usa { } para controle de escopo. E muito menos begin e end como em outra linguagens.

```
if condicao:
    pass #<bloco de comandos para o if>
else:
    pass #<bloco de comandos para o else>
```

- O único indicador de início de escopo são os dois pontos ":"
- O bloco será identificados pelas indentação.

O comandos **pass** usado acima não faz nada =D



Switch

Quem já usou Switch para controle de tomada de decisão em um código?

```
switch(caso) {  
    case 1:  
        printf("Caso 1");  
        break;  
    case 2:  
        printf("Caso 2 e ");  
    case 3:  
        printf("Caso 3");  
        break;  
    default:  
        printf("Caso padrao");  
}
```

Legal... Python não tem... =/



elif

elif é usado em situações que existem mais de dois caminhos para uma avaliação.

```
>>> calc = input("Digite uma operação: ")
Digite uma operação: 4+5
>>>
if calc > 5:
    print "calc maior que 5"
elif calc < 5:
    print "calc menor que 5"
else:
    print "calc igual a 5"

calc maior que 5
>>> calc
9
```

Para quem sente falta do **switch** o **elif** pode ser usado para as mesmas coisas.



Condicional de uma linha

Sintaxe:

VALORVERDADE if condicao else VALORFALSO

Em outras linguagens isso é bem mais bonito.

Exemplo:

```
>>> nome = raw_input("Digite seu nome: ")
Digite seu nome: Fulano
>>> print "Seu nome é", "maior" if len(nome) > len("Fábio") else
"menor", "que o meu"
Seu nome é maior que o meu
>>> nome = raw_input("Digite seu nome: ")
Digite seu nome: Igor
>>> print "Seu nome é", "maior" if len(nome) > len("Fábio") else
"menor", "que o meu"
Seu nome é menor que o meu
```



Estrutura de repetição

As estruturas de repetição em Python são: for e while

while

Observe o exemplo:

```
>>> i = 1
>>> while i < 100:
    print i,
    i+=1
```

```
1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24
25 26 27 28 29 30 31 32 33 34 35 36 37 38 39 40 41 42 43 44 45
46 47 48 49 50 51 52 53 54 55 56 57 58 59 60 61 62 63 64 65 66
67 68 69 70 71 72 73 74 75 76 77 78 79 80 81 82 83 84 85 86 87
88 89 90 91 92 93 94 95 96 97 98 99
```

Assim como todo bloco de comandos em Python, aqui o controle de Escopo também é feito por indentação.



Continuando while

O while em Python é muito comum em relação a mesma estrutura em outras linguagens.

Sintaxe:

`while` condicao:

`<bloco de comandos>`

`else:`

`<bloco de comandos>`

Heim? O.o else no while? O.O Oooooooooooooh!

A grande diferença do while do python para outras linguagens.



break

Python possui a instrução break.

Veja o exemplo:

```
>>> i = 1
>>> while i < 100:
    if i == 50:
        break
    print i,
    i += 1
```

```
1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24
25 26 27 28 29 30 31 32 33 34 35 36 37 38 39 40 41 42 43 44 45
46 47 48 49
```

A instrução break interrompe o laço realizando uma saída abrupta dele e finalizando sua execução.



Python possui também a instrução continue

Veja o exemplo:

```
>>> login = ""
>>> while True:
    login = raw_input("Digite o login: ")
    if login == "": continue
    if login == "fabio":
        print "Login correto."
        break
```

```
Digite o login:
Digite o login: acens
Digite o login: pedro
Digite o login: fabio
Login correto.
```

A instrução continue termina a interação atual do laço voltando para checagem de condição.



E o else, homi?

O else em laços é executado caso não seja executada uma interrupção feita pela instrução break.

Exemplo:

```
>>> i = 1
>>> valor = int(raw_input("Digite o valor para parar: "))
Digite o valor para parar: 999
>>>
while i < 100:
    if i == valor: break
    i+=1
else:
    print "Não parei. Valor fora da faixa"
```

Não parei. Valor fora da faixa

Como o valor digitado no exemplo foi 999 não irá executar o break, logo o bloco do else foi executado.



for

O **for** encontrado na linguagem Python é diferente do for de Java, C, PHP, C#...

Ele é **parecido** com foreach do PHP e C#.

Sintaxe:

```
for var in iterable:
```

```
    <comandos do for>
```

```
else:
```

```
    <comandos do else>
```

Assim como no **while** o **for** também possui o bloco **else** e funciona da mesma maneira.



Entendendo o for

Nada melhor que um exemplo:

```
>>> empresa = "Acens"  
>>> for letra in empresa:  
    print letra
```

```
A  
c  
e  
n  
s
```

O for em Python “varre” toda a sequência(empresa) e guarda o valor na variável(letra) em cada interação. Outro exemplo:

```
>>> empresa = "Acens"  
>>> for i,letra in enumerate(empresa):  
    print "%d => %s" %(i,letra)
```

```
0 => A  
1 => c  
2 => e  
3 => n  
4 => s
```



E se eu quiser fazer como um for normal?

Comparação do for em C e Python

C

```
int i;  
for (i = 0; i < 100; i++)  
    printf("%d ", i);
```

Python

```
for i in range(100):  
    print i,
```

O comando range() gera uma sequência(lista) de inteiros. help(range)

C

```
int i;  
for (i = 50; i < 100; i += 2)  
    printf("%d ", i);
```

Python

```
for i in range(50, 100, 2):  
    print i,
```

Exibindo os pares de 50 até 99 com o a variável incrementando de 2 em 2.