Python Básico

Prof. Dr. Dilermando Piva Jr.

Por onde começo ?

… Criando nosso primeiro Hello World !

Hello World

... ‘hello world’ - Python X {Java, C, PHP, Pascal}

```
program helloworld;
begin
    writeln<'Hello World!'>;
end
```

```
class HelloWorld
{
    public static void main(String[] args)
    {
        System.out.println("Hello World!");
    }
}
```

```
#include <stdio.h>
int main (void)
{
    printf("Hello World");
    return 0;
}
```

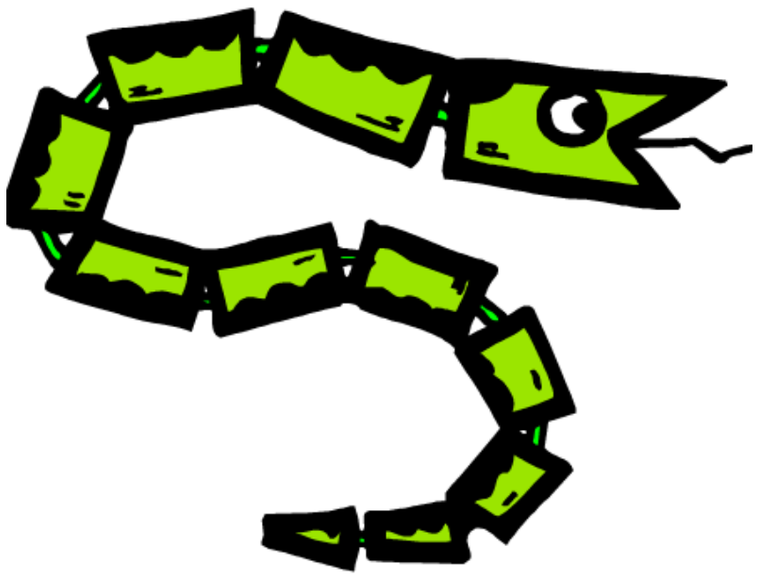
```
<?php
    echo "Hello World";
?>
```

... em Python ...

```
print (“Hello World”)
```

Tipos e operações

Vamos ver um trecho de código em Python!



Código Base

```
x = 34 - 23                    #Um comentário
y = "Hello"                    #Outro comentário
z = 3.45
if z == 3.45 or y == "Hello":
    x = x + 1
    y = y + "World"      #Concatenação de String
print (x)
print (y)
```

... entendendo o código...

- Atribuição utiliza = e comparação utiliza ==

... entendendo o código...

- Atribuição utiliza = e comparação utiliza ==

```
x = 34 - 23                    #Um comentário
y = "Hello"                    #Outro comentário
z = 3.45
if z == 3.45 or y == "Hello":
    x = x + 1
    y = y + "World"        #Concatenação de String
print (x)
print (y)
```

... entendendo o código...

- Números: + - * / % tem suas funções características
- + pode ser usado como concatenação de Strings;
- % pode ser usado para formatar Strings (assim como em C).

... entendendo o código...

- Números: + - * / % tem suas funções características
- + pode ser usado como concatenação de Strings;
- % pode ser usado para formatar Strings (assim como em C).

```
x = 34 - 23                 #Um comentário
y = "Hello"                 #Outro comentário
z = 3.45
if z == 3.45 or y == "Hello":
    x = x + 1
    y = y + "World"         #Concatenação de String
print (x)
print (y)
```

... entendendo o código...

- Operadores lógicos são palavras e não símbolos (||, &&)

- and, or, not

... entendendo o código...

- Operadores lógicos são palavras e não símbolos (||, &&)

- and, or, not

```
x = 34 - 23                    #Um comentário
y = "Hello"                    #Outro comentário
z = 3.45
if z == 3.45 or y == "Hello":
    x = x + 1
    y = y + "World"            #Concatenação de String
print (x)
print (y)
```

... entendendo o código...

- print é o comando básico para “impressão” na tela

... entendendo o código...

- print é o comando básico para “impressão” na tela

```
x = 34 - 23                    #Um comentário
y = "Hello"                    #Outro comentário
z = 3.45
if z == 3.45 or y == "Hello":
    x = x + 1
    y = y + "World"        #Concatenação de String
print (x)
print (y)
```

... entendo o código...

- E se você quiser receber uma entrada diretamente do usuário ?
  - *input() - retorna uma string !*

>>> input('Digite um valor')

... entendendo o código...

- A primeira atribuição em uma variável também é responsável por cria-lá.
  - Os tipos das variáveis não precisam ser informados;
  - Python descobre o tipo da variável por conta própria!

... entendendo o código...

- A **primeira atribuição** em uma variável também é responsável por cria-lá.
  - Os tipos das variáveis não precisam ser informados;
  - Python descobre o tipo da variável por conta própria!

```
x = 34 - 23                 #Um comentário
y = "Hello"                 #Outro comentário
z = 3.45
if z == 3.45 or y == "Hello":
    x = x + 1
    y = y + "World"         #Concatenação de String
print (x)
print (y)
```

... Usando o Shell

- Para iniciar o shell basta digitar o comando
  - #> python
- Quando o shell é iniciado aparecerão três '>' (">>>") indicando que ele está ativo e pode receber comandos
  - Exemplo
    - #> python
    - >>> print "HelloWorld!!!"
    - HelloWorld!!!
    - >>>

... Usando o Shell

- Para obter informações como métodos e atributos de um objeto basta executar o comando “dir”. Obs.: Tudo em Python é objeto!
  - >>>dir(“string de teste”)
  - <tudo sobre strings!>
  - >>>
- Para visualizar a documentação de um Objeto basta executar o comando “help”
  - >>>help(1000)
  - <Documentação do objeto Inteiro>

… Usando o Shell

- Para repetir o comando anterior pode-se
  - Usar a seta para cima
  - Digitar ‘_’
- Para navegar entre os comando já executados basta usar as setas para cima e para baixo
- Para obter ajuda geral executa-se o comando “help()”
  - Para sair do help “quit”
  - Para obter a lista dos módulos “modules”

Whitespace

- Importante para identação e novas linhas
  - Use \ para quando for para uma próxima linha prematuramente.
- Em Python não há { } !! Isso é para definição de dicionários (*dict*)- nas versões atuais… já existe!!
- Blocos de código definidos por identação!

Comentários

- Comentários começam com #
  - Convenção: Você pode definir uma “documentação” em string como primeira linha de qualquer nova função que você definir.
  - Muito importante para o desenvolvedor, crítico para o usuário!

```
def my_function(x, y):
    """This is the docstring. This
    function does blah blah blah."""
    # The code would go here...
```

Conhecendo a linguagem...

- Dinamicamente tipada
  - Exemplo
    - >>>a = 10
    - >>>a = “teste”
    - >>>
- Fortemente tipada, não existe cast.
  - Se quiser mudar o tipo, use uma função
    - Exemplo
      - >>>a = (int) 1.0 # ERRO!!!
      - >>>a = int(1.0)

Conhecendo a linguagem...

- Não possui declaração de tipos
  - Java
    - int a = 0;
  - Python
    - a = 0
- Não possui comandos declarativos (“óbvios”)
  - Java
    - Algo n = new Algo();
  - Python
    - n = Algo()

Conhecendo a linguagem...

- Python é dinamicamente tipado, você não precisa declarar o tipo das variáveis
- A declaração acontece automaticamente no momento em que um valor é atribuído à variável
- Variáveis podem mudar de tipo, simplesmente com uma nova atribuição de um tipo diferente
- Python permite atribuir um único valor à várias variáveis ao mesmo tempo
- Assim como permite atribuir multiplos objetos à múltiplas variáveis

```
max = 10                # integer
value = 150.0           # float point
name = "Python"         # string
nothing = None          # null value
```

```
x = 1
x = "String value"
```

```
a = b = c = 1
```

```
a, b, c = 1, 2, "Python"
```

Tipos Básicos

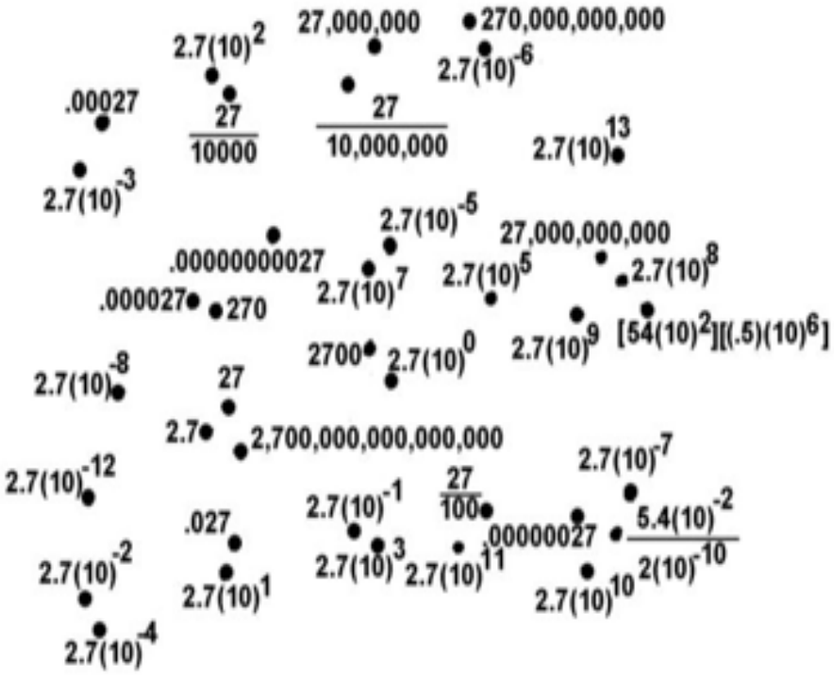
- Inteiros (padrão para números)
  - Divisão entre inteiros, resposta um inteiro!
- Inteiros Longos
  - L ou l no final. (Convertido automaticamente com precisão de inteiros > 32 bits)
- Floats (ponto flutuante)
  - 1.23, 3.4e-10
- Complexas
  - >> 2 + 3j
- Operações válidas: +, *, >> , ** , pow, abs, etc.

Tipos Básicos

- Representação numérica
  - Representação de dígitos com/sem formatação de string
- Divisão clássica / base
  - Uso dos operadores // e /
- Operações em nível de bit
  - 1 << 2 , 1| 2 , 1&2
- Notações hexadecimal / octal
  - 2 , 0x10 , 0100 , oct(64), hex(255) , int('200'), int('0100',8) , int('0x40',16)
- Operações válidas: +, *, >> , ** , pow, abs, round, etc.

Tipos Básicos

- Números são objetos imutáveis em Python, isso significa que seus valores não podem ser alterados
- Existem 3 tipos de dados para números em Python
  - int (signed integer)
    - Também chamado apenas de integer, comporta números positivos e negativos sem casas decimais
  - float (floating point)
    - Representa números reais e são escritos com ponto decimal, dividindo um inteiro em partes fracionárias, também podendo ser escrito em notação científica com “e” indicando potencia de 10 (ex: 2.5e2)
  - complex
    - São escritos por dois valores reais, a parte real e a parte imaginária na forma (real + IMAG J). Neste caso o número i ($\sqrt{-1}$) é designado pela letra j. (Não são muito utilizados)



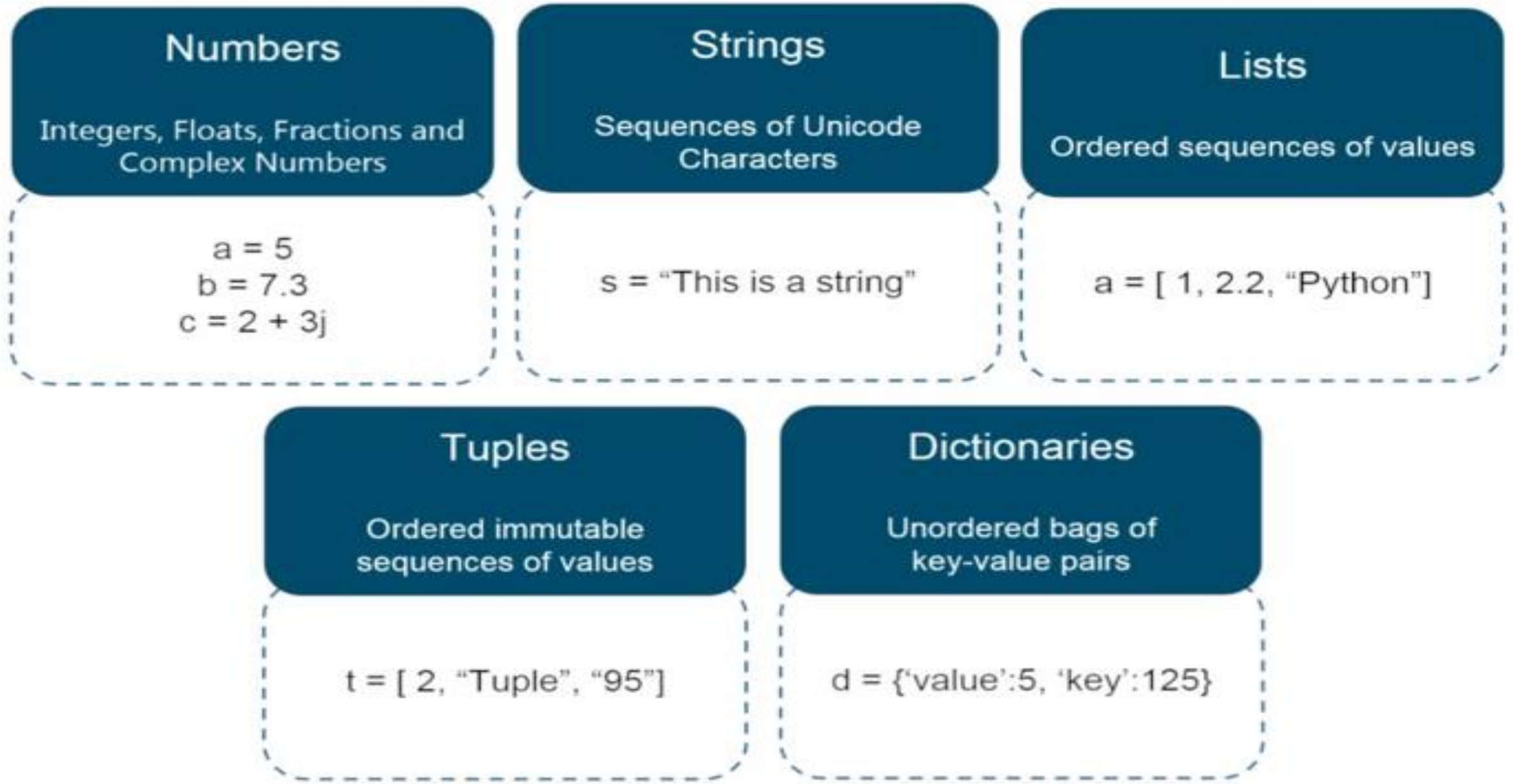
Tipos Básicos

Operação	Resultado
x + y	Soma dos valores x e y
x - y	Subtração de x por y
x * y	Multiplicação de x por y
x / y	Divisão de x por y
x // y	Divisão de x por y, obs.: Pegando o piso.
x % y	Resto da divisão de x por y
+x	Não altera nada
-x	Inverte o sinal de x
abs(x)	Valor absoluto de x
int(x)	x convertido em inteiro
long(x)	x convertido em long
float(x)	x convertido em float
complex(re, im)	Um número complexo com parte real re e imaginária im

Operação	Resultado
x | *y*	Bit a bit **ou** de x e y
x ^ *y*	Bit a bit **ou exclusivo** de x e y
x & *y*	Bit a bit **e** de x e y
x << *n*	*x deslocado à esquerda n bits*
x >> *n*	*x deslocado à direito n bits*
~x	os bits de x invertidos

exp (x)	Retorna o exponencial de x (e^x)
log (x)	Retorna o logaritmo natural de x (inverso da função exponencial)
pow (x, y)	Retorna o valor de x elevado à potencia y
sqrt (x)	Retorna a raiz quadrada de x
round (x, y)	Retorna x arredondado em y casas decimais

Tipos Básicos



Tipos Básicos

- Strings
  - “abc” ou ‘abc’
- Operadores de expressão de Python e sua precedência
- http://docs.python.org/reference/expressions.html#summary

Símbolo	ação comparativa
“<”	Menor que
“<=”	menor ou igual
“>”	maior que
“>=”	maior ou igual
“==”	igual (objeto ->referência)
“!=”	diferente
“<>”	diferente
“is”	igualdade de objetos
“is not”	diferença de objetos

```
>>>True > False
<Qual seria o resultado???>
```

Tipos Básicos

- Strings

- String, assim como numbers, são objetos imutáveis em Python. Dessa forma um update só pode ser possível com a alocação de um novo objeto com o novo conteúdo
- Python não suporta tipos “char”. Um char em Python é considerado uma String de tamanho 1
- Uma String pode ser atribuída de várias formas diferentes
  - ‘Uma String’
  - “Outro exemplo de String”
  - ”’String com
    multiplas linhas”’
- Strings iniciam sempre com indice 0
- Pode se usar operações como slicing ([], [:]), concatenação (+), repetição (*) e *membership* (in)

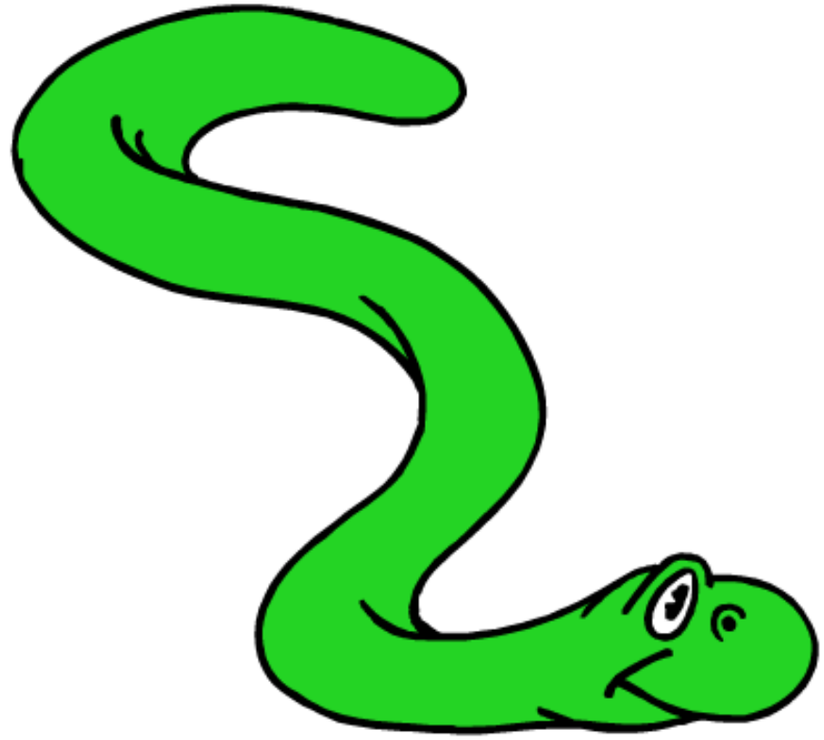
Comandos básicos

- Alguns comandos básicos que podem ajudar no ínicio!
  - dir(element) - todos os atributos e métodos que estão associados a elemento.
  - type(element) - Descobrir o tipo do objeto!
  - import   - importe módulos para uso no seu código!

```
import modulo
import modulo.algo
import modulo.algo as malg
from modulo import *
from modulo import algo
from modulo import algo.item as ait
```

Atribuição

... Vamos entender como funciona atribuição!



Atribuição

- Atribuição de uma variável em Python significa criar um rótulo para armazenar uma referência para algum objeto.
  - Atribuição cria referências e não cópias!
  - Inferência do tipo da referência baseado no tipo de dado atribuído
- A referência é deletada por meio de Garbage Collection
  - Quando o objeto deixa de ser referenciado por nenhum outro rótulo(variável).

Atribuição

- Lembre-se que Python a tipagem é dinâmica!
  - Declarar variáveis sem atribuí-las irá levantar um erro!

```
>>> y
Traceback (most recent call last):
  File "<pyshell#16>", line 1, in -toplevel-
    y
NameError: name 'y' is not defined
>>> y = 3
>>> y
3
```

Atribuição

- Você pode inicializar várias variáveis de uma só vez!
  - x = y = z = 2.0

- Rótulos de variáveis são Case Sensitive e não podem iniciar com número. Números, letras e underscores são permitidos!
  - bob bob_2 _bob _2_bob bob_2 BoB

- Não esquecer das palavras reservadas!

```
and, assert, break, class, continue, def, del,
elif, else, except, exec, finally, for, from,
global, if, import, in, is, lambda, not, or, pass,
print, raise, return, try, while
```

Atribuição

- Entendendo manipulação de atribuição de referências
  - x = y não significa que você fez uma cópia de y!
  - x = y o que realmente faz é x referencia ao objeto que y referencia!

- O que realmente acontece por trás dessa simples atribuição:

```
>>> x = 3
>>> x = x + 1
>>> print x
4
```

Atribuição

- Mas e se fizermos isso ?! Qual será o valor de x ?

```
>>>  x = “casa”
>>>  y = x
>>> x = “fazenda”
>>> print x
```

Atribuição

- Mas e se fizermos isso ?! Qual será o valor de x ?

```
>>>  x = “casa”
>>>  y = x
>>> y = “fazenda”
>>> print x
```

- Do mesmo jeito que nós esperávamos! Dados nativos são imutáveis! (String, Inteiros, float, complexos).

```
>>>  x = “casa”  #cria 3, x referencia ao objeto string “casa”
>>>  y = x        # Cria variavel y,  referencia ao objeto string “casa”
>>> y = “fazenda”   #Cria referencia ao objeto string “fazenda”
>>> print x         # Nenhum efeito em x, ainda referencia “casa”
>>>  casa
```

Listas, Strings e Tuplas

... O poder de python agora!

Listas, Strings e Tuplas

- Todos os três são Sequências!
  - Podem ser indexados por algum valor ordinal posicional
  - Todas as operações apresentadas aqui nesta seção podem ser aplicadas em todos os tipos de sequência

- Listas
  - li = [1,2,3, ‘abc’]

- Tuplas
  - li = (23, ‘abc’, 4.56, (2,3), ‘def’)

- Strings
  - st = “Hello World” st = ‘Hello World’

Listas, Strings e Tuplas

- Manipulando sequências!
  - Pelo índice a partir de 0 Ex: ti [0]
  - Índices podem ser positivos ou negativos! Ex: ti[1] (esq.) ti[-4] (dir.)

- Fracionamento e matrizes!
  - li[1:3] L[1:] matrix = [[1,3,4] , [3,5,6] , [7,8,9]]

- Operador *in*
  - retorna um booleano. Checa se um valor está em uma sequência!
    - 4 in li

Listas, Strings e Tuplas

```
>>>a = [ 1,2,3,4,5]  #criação da lista
>>>a[ 0]
1
>>>a[ 2]
3
>>>a[ -1]
5
>>>a[ -3]
3
>>>a[ 1:]
[ 2,3,4,5]
>>>a[ :3]
[ 1,2,3]
>>>a[ 1:4:2]  #acrescido o passo, coleta-se pulando de 2 em 2
[ 2,4]
>>>a[ ::-1]
[ 5,4,3,2,1]  #passo negativo inverte a sequência
```

Exemplos2

Operacões em Listas

- Operador + , *
  - a = “Hello” + “ World” (concatenação)
  - *[3]* * 4 (repetição)
- Operador len() e append()
  - len() - retorna um inteiro com o tamanho da sequência!
  - pop() - retira o último elemento da lista (conceito de pilhas!)
  - append() - adiciona um elemento ao final da lista!
- Atribuição
  - *list[0] = ‘3’*
  - Fazendo cópias de sequência , Cuidado!!!

Exemplos2

Operacões em Listas

- Qual será o valor de b ?

```
>>> a = [1,2,3]
>>>  b = a
>>> a.append(4)
>>> print b
```

Operacões em Listas

- Qual será o valor de b ?

```
>>> a = [1,2,3]
>>>  b = a
>>> a.append(4)
>>> print b
```

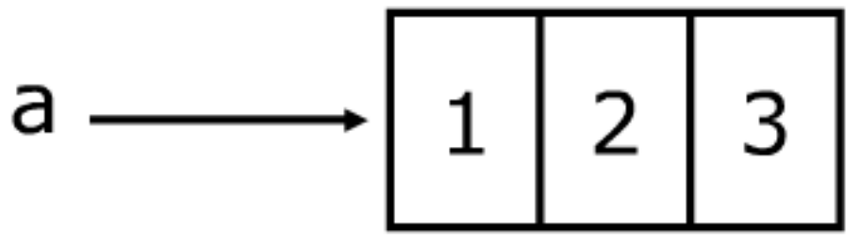
- Surpresa!

```
>>>  b = [1,2,3,4]
```

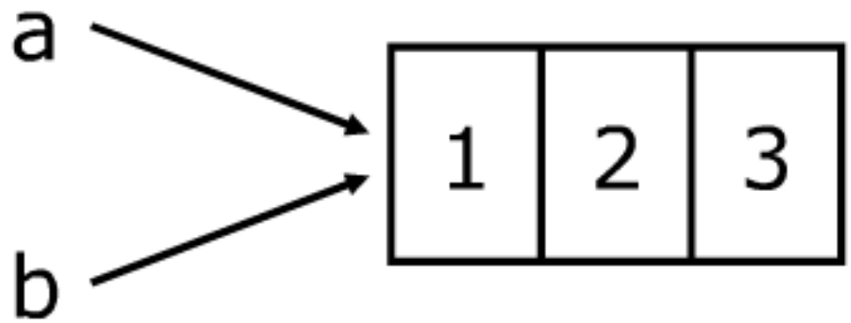
- Dados do tipo listas, dicionarios e pré-definidos pelo usuário são **mutáveis**!

Operacões em Listas

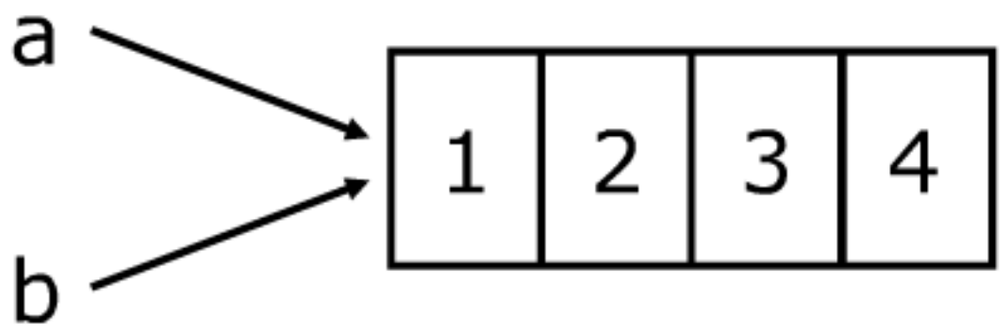
a = [1, 2, 3]



b = a



a.append(4)



Operacões em Listas

- Para fazer cópias de listas
  - a = b[:] (2 cópias independentes)
  - a = b (os 2 referenciam o mesmo objeto)

- Qual a diferença entre listas e tuplas ?
  - Listas são mutáveis e Tuplas imutáveis!
  - l = [1,‘abc’,4]   t = (1,‘abc’,4,5)

- Atribuição em listas e tuplas
  - *list[0] = ‘3‘ ok!*
  - *t[0] = 3   NOK!!! (Deve-se criar uma nova tupla! - t = (3, ‘abc’,4,5)*

Listas de Listas....

- Matriz... (lista de listas...)

```
listas = [[1, 2, 3], [4, 5, 6], [7, 8, 9]]  # Matriz 3 x 3


# Iterando com loops em uma lista aninhada
for lista in listas:
    for num in lista:
        print(num)
```

```
/home/geek/.virtualenvs/ppe/bin/p
1
2
3
4
5
6
7
8
9

Process finished with exit code 0
```

```
# List Comprehension

[[print(valor) for valor in lista] for lista in listas]
```

Tuplas x Listas

- Listas são mais lentas porém mais poderosas que tuplas
  - Listas podem ser modificadas e tem diversos operadores que podem ser utilizados
  - Tuplas são imutáveis e tem menos funcionalidades!

- Para converter entre listas e tuplas ?

```
>>>a = [ 1,2,3,4,5]
>>>tuple(a)
 (1,2,3,4,5)
>>>list(tuple(a))
[ 1,2,3,4,5]
>>>help(tuple) #ler o help..
```

Métodos e Funções (Listas)

Função	Descrição
len(lista)	Retorna o tamanho total da lista
max(lista)	Retorna o maior elemento da lista
min(lista)	Retorna o menor elemento da lista
list(tupla)	Converte uma tupla em uma lista
list.append(obj)	Adiciona um objeto à lista
list.insert(index, obj)	Insere um objeto à lista em determinada posição
list.count(obj)	Retorna a quantidade de vezes que um determinado objeto ocorre na lista
list.index(obj)	Retorna o primeiro indice de ocorrencia de um determinado objeto
list.remove(obj)	Remove um objeto da lista
list.reverse()	Reverta o ordenamento da lista
list.sort()	Ordena a lista

Strings

- Formatação e conversão de Strings
- Usam os mesmos operadores básicos de lista
- Multi-Strings, Strings com aspas simples e duplas
- Caracteres Speciais e str() e unicode()

Exemplos2

Strings

- Também uma sequência e é Imutável!
  - “42” + 1 (erro!) Use “42” + str(1)
  - float(), int() -> string para numéro
- Atribuição
  - S = ‘spam’ S[0] = ‘x’ ERRO!!!
  - Strings são imutáveis!
  - String -> Lista -> String (.join)
- Formatação de string

Exemplos2

Métodos mais usados

- find(), replace(), join(), split()
- isdigit(), islower(), strip(),
- startswith(), upper(), lower()
- etc.

Exemplos2

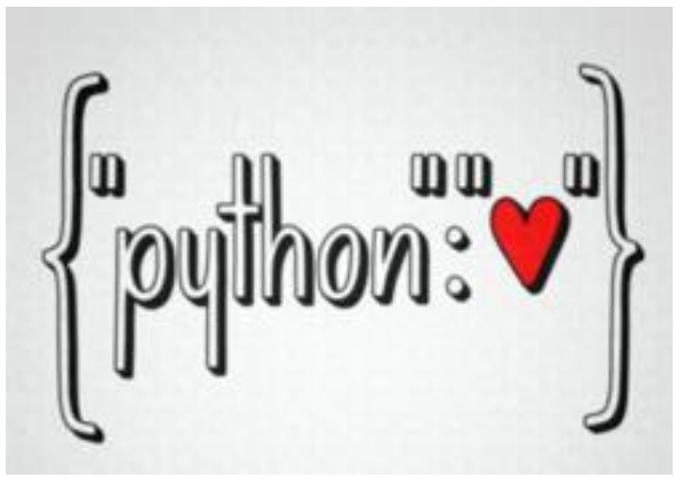
Dicionários

Um “hash map” pythonico!

Dicionários

- Estrutura de dados em forma de coleções onde os items são armazenados e buscados pela *chave* em vez do deslocamento posicional.
  - Chaves podem ser quaisquer objetos do tipo imutável
  - Valores podem ser de qualquer tipo
  - Um dicionário pode armazenar diferentes tipos de valores e é **mutável**!

- Criando e modificando dicionários!
  - d = {“user” : “Marcel” , “password”: 2342}



Dicionários

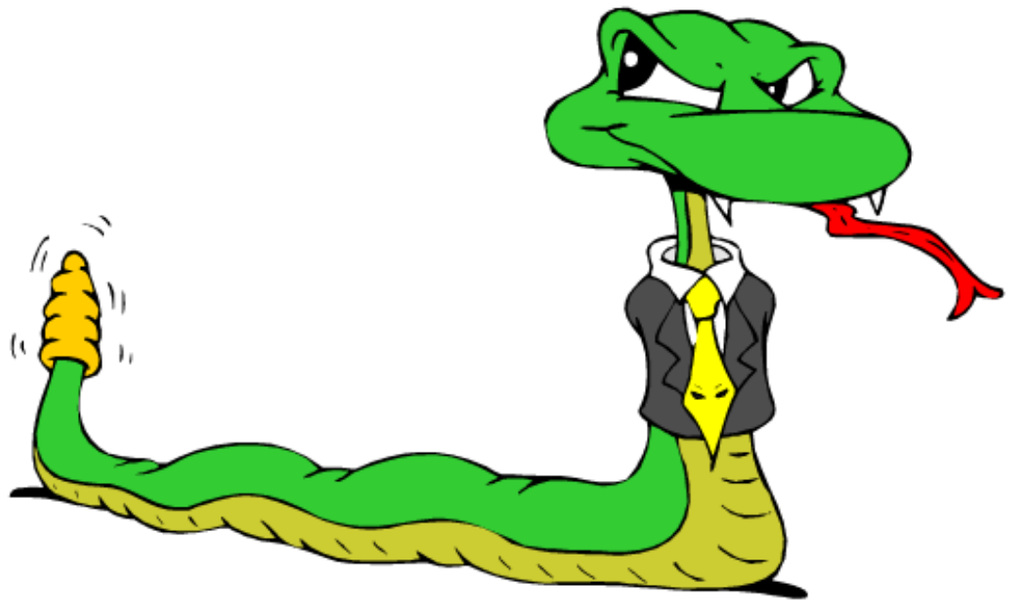
- Dicionários não são ordenados!
  - Uma nova chave pode aparecer em qualquerr lugar
  - Funciona como “hashing”

- Algums métodos:

Função	Descrição
len(dict)	Retorna o número de elementos do dicionário
str(dict)	Retorna a representação em formato string do dicionário
dict.keys()	Retorna a lista de chaves do dicionário
dict.values()	Retorna a lista de valores do dicionário
dict.items()	Retorna a lista de chave, valor do dicionário
dict.get(key, default=None)	Retorna o valor de uma chave ou um valor default caso não seja encontrada
dict.update(dict2)	Insere um elemento (chave, valor) no dicionário
dict.clear()	Remove todos os elementos do diciónário

Arquivos

Como é fácil manipular um arquivo!



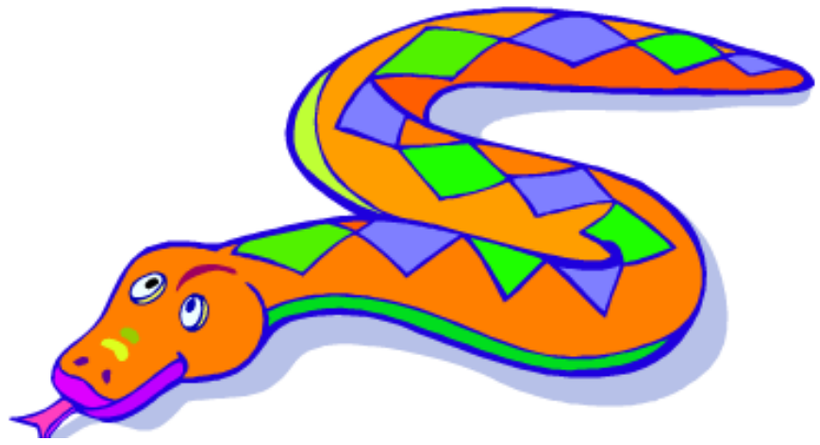
Arquivos

- Apenas uma linha para abrir um arquivo!
  - file = open(“data”, ‘r’) tipos: r, a, w
- Alguns métodos para operações em arquivos:
  - file.read(), readline(), readlines(),
  - file.write(), writelines(),
  - file.close()

Exemplos3.py

Booleanos

Expressões lógicas



Expressões lógicas

- *True* e *False* são constantes em Python
  - False : 0, None, [] , {} , 0.0
  - True: Valores Numéricos exceto 0, objeto não vazios
  - Um dicionário pode armazenar diferentes tipos de valores e é **mutável**!
- Operadores de comparação: ==, != , < , <=, etc.
  - X == Y (efetua teste de equivalência de valor)
  - X is Y (Testa a identidade do objeto)

Exemplos3.py

Expressões lógicas

- *None* é similar ao NULL em linguagem C
  - L = [None] * 100 (declara uma lista de 100 items None)
- Operações com or e and
  - not -> inversão lógica (true -> false , false -> true)
  - and e or (&& e II)
    - **Casos especiais: Ele retorna o valor de uma das sub-expressões!
- isinstance(element,type)
  - Verifica se um elemento é do tipo type

Exemplos3.py

Instruções compostas

```
If python == “cool”:
        print “Oh yeah!”
```

Fluxo de Controle

- Várias expressões Python para controlar o fluxo do programa. Todos eles fazem uso de testes condicionais booleanos.
  - ifs, else
  - loops while, for
  - assert

Instruções if

- Não esqueçam da identação em blocos!
- E do (:) após a expressão booleana!

- C

```
if (a == 1) {
   printf("op1\n");
} else if (a == 2) {
   printf("op2\n");
} else {
   printf("outra\n");
}
```

- Python

```
if a == 1:
   print "op1"
elif a == 2:
   print "op2"
else:
   print "outra"
```

Instruções if

Selection.py

```
var1 = 100
if var1:
    print(var1)
```

Selection

```
C:\dev\workspace_git\DataScienceFac
100

Process finished with exit code 0
```

Selection.py

```
var1 = 100
if var1:
    print('OK')
else:
    print('NOK')
```

Selection

```
C:\dev\workspace_git\DataScienceFa
OK

Process finished with exit code 0
```

Selection.py

```
var = 100
if var < 150:
    if var == 150:
        print("150")
    elif var == 100:
        print("100")
    elif var <= 50:
        print("<= 50")
else:
    print(">= 150")
```

Selection

```
C:\dev\workspace_git\DataScienceFac
100

Process finished with exit code 0
```

Instruções if

```
x = 5
if 0 < x and x < 12 or x == 5:
    print u"x é menor que uma dúzia!"

if 0 < x < 12:
    print u"x é menor que uma dúzia!"

if x in [1,2,3,4,5,6,7]:
    print u"x pertence da lista!"
elif x == 5 <= 10 > 0 < 1000:
    print u"x não está neste intervalo!"
else:
    print u"nenhuma das condições acima são verdadeiras!"

st = x < 7 and "reprovado" or "aprovado"
```

Instrução assert

- O uso de assert permite verificar se algo é verdadeiro durante a execução do programa.
- Se a condição for falsa, o programa é interrompido.

```
assert(number_of_players < 5)
```

Instruções while

- Você pode usar o comando break para sair do loop mais próximo que a envolve.

- Você pode usar o comando continue para pular para o ínicio do loop mais próximo que a envolve e pular para a próxima iteração.

- Você pode usar o comando pass quando você não quer que se faça nada (instrução vazia)

- Você pode o o bloco else do loop para quando se quer executar um código quando se sai normalmente do loop (sem ser por comando break)

Instruções while

```
while x < 10:
    print x
    x+=1
    if x == 10:
        break

while not fim:
    fim = fim - 1

while True:
    pass
```

Loops.py

```
1   count = 0
2   while count < 5:
3       print(count)
4       count = count + 1
5
6
```

```
t = []
for x in xrange(10):
    if x % 2:
        print u"é par"
        continue
    else: t.append(x)
    print "%d é par" % x
```

Instruções for

- Loops for iteram sobre uma sequência de items (listas, tuplas, string ou quaisquer outros objetos cuja a linguagem considere como um “iterator”)

- Várias maneiras de iterar sobre um conjunto de items!

- Também possui o bloco else quando se sai normalmente do loop (similar ao while)

- Função muito usada nos loops for: range()
  - range() - Retorna uma lista de números que varia de 0 a ao número passado como parâmetro.
  - xrange() - Retorna uma lista como range() só que libera o item quando for requisitado! Mais eficiente, porém apenas com items do mesmo tipo e sem suporte à slicing, repetição e concatenação.

Exemplos4

Instruções for

```
for n in [1,2,3,4,5]:
    print n

for m in ["teste","de","for"]:
    print m, len(m)

for s in range(10): print s**s

d = {"gol": 30000,"fusca":2500,"hilux":115000}
for k, v in d.items():
    print u"O preço do %s é %d" % (k, v)
```

```
>>>range(10)
[0, 1, 2, 3, 4, 5, 6, 7, 8, 9]
>>>range(5,25,7)
[5, 12, 19]
>>>range(-10,-50,-15)
[-10, -25, -40]
>>>xrange(10)
xrange(10)
>>>for 'a in xrange(10): print    a,
...
0 1 2 3 4 5 6 7 8 9
```

Loops.py

```
for letter in 'Python':
    print('The letter is: ', letter)



frutas = ['banana', 'maça', 'laranja']
for fruta in frutas:
    print(fruta)



comidas = ('Pizza', 'Arroz', 'Feijão')
for index in range(len(comidas)):
    print(comidas[index])

```

Instruções while e for

- Estruturas de Repetição oferecem instruções de controle em determinadas situações:

- **break** – permite a saida do fluxo antes da condição de finalização do laço ser atingida

- **continue** – Instrui o interpretador a passar para a próxima iteração imediatamente, ignorando os comandos subsequentes

- **pass** – permite que um determinado trecho de código seja escrito sintaticamente correto, porém em sua execução não será executada nenhuma instrução naquele momento

Instrução zip

- zip() é bastante poderoso, pode unir sequências onde retorna uma lista de tuplas que se distribuem em pares os items paralelos extraídos dessas sequências.

- Permite também facilitar a construção de dícionarios!

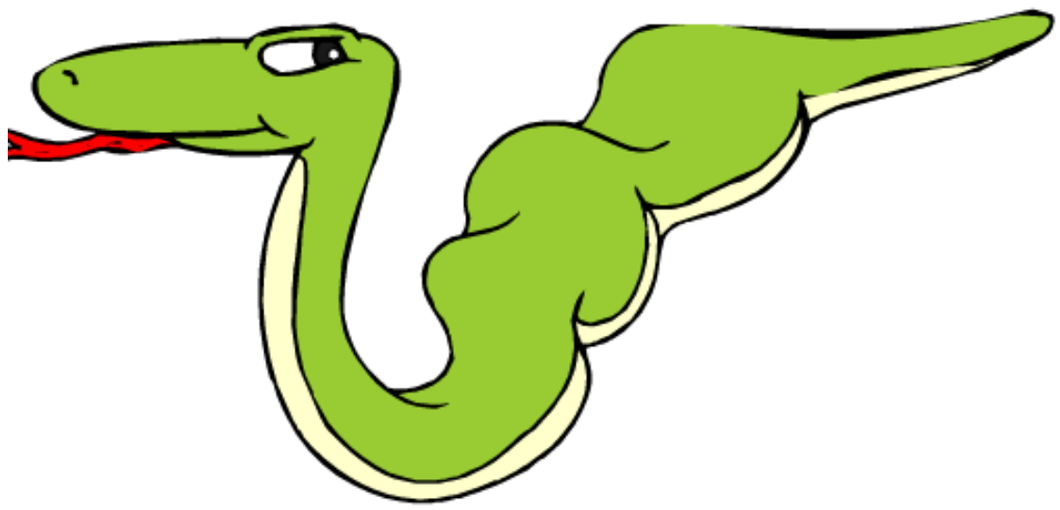
- x = dict(zip(kes,vals))

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

Exemplos4.py

Compreensão de listas

[i for i in “python é fácil demais”]



Compreensão de listas

- Funcionalidade muito poderosa da linguagem Python
  - Gera uma lista nova aplicando uma função para cada elemento da lista original.
  - Muito usado por programadores Python! (Economia de código!)

- A sintaxe da compreensão de lista usa-se de palavra-chaves:
  - [expression for name in list]

```
>>>s = [x**2 for x in range(10)]
>>>m = [x for x in s if x%2 == 0]
```

Compreensão de listas

- Permite também o uso de filtros (deteminam se uma determina expressão deve ser executada sobre um membro da lista)

- [expression for name in list if filter]

```
>>>d={"golf":40000,"celta":26000,"Hilux":95000,
"fusca":3000}
>>>possibilidades_rico = [car for car in d if d[car]
< 70000]
>>>possibilidades_pobre = [car for car in d if
d[car] < 300]
```

Exemplos4.py

Compreensão de listas

- Você também pode aninhar compreensão de listas!
- [expression for name in [expression for name in list]]

Exemplos4.py

QuickSort

- Algoritmo de ordenação bastante utilizado e muito eficiente
- Complexidade BigO(*nlogn)*

1 . Escolher um pivô inicial x;

2. Colocar todos itens com chave menor que a de x à esquerda de x, formando uma seqüência S1;

3. Colocar todos itens com chave maior que a de x à direita de x, formando uma seqüência S2;

4. Isto feito, o mesmo processo é aplicado às seqüências S1 e S2, que por sua vez produzirão novos segmentos;

5. O processo deve ser aplicado sucessivamente às seqüências enquanto elas tiverem tamanho ≥ 1;

QuickSort

- Você pensaria assim...

```
def partition(list, l, e, g):

    if list == []:
        return (l, e, g)
    else:
        head = list[0]
        if head < e[0]:
            return partition(list[1:], l + [head], e, g)
        elif head > e[0]:
            return partition(list[1:], l, e, g + [head])
        else:
            return partition(list[1:], l, e + [head], g)
```

QuickSort

- Agora que você sabe compreensão de listas, você pode fazer assim!

```
def qsort(L):
if len(L) <= 1: return L
return qsort( [ lt for lt in L[1:] if lt < L[0] ] )  +  [ L[0] ]  + \
       qsort( [ ge for ge in L[1:] if ge >= L[0] ] )
```

- E não é que lembra a linguagem funcional **Haskel** ?!

```
# qsort [] = []
# qsort (x:xs) = qsort elts_lt_x ++ [x] ++ qsort
elts_greq_x
#                  where
#                    elts_lt_x = [y | y <- xs, y < x]
#                    elts_greq_x = [y | y <- xs, y >= x]
```

Ordenação

- Mas um programador Pythonico, ainda faria mais eficiente!

```
list.sort()
```

- Utiliza-se de uma implementação nativa de Python para ordenação de sequências! Mais eficiente, híbrido com complexidade no pior caso de *nlogn.*

Python é muito poderoso!

- Não precisa reinventar a roda! Molde-a para adaptar ao seu problema!

- A documentação de Python é bastante vasta e há muitas funcionalidades prontas!