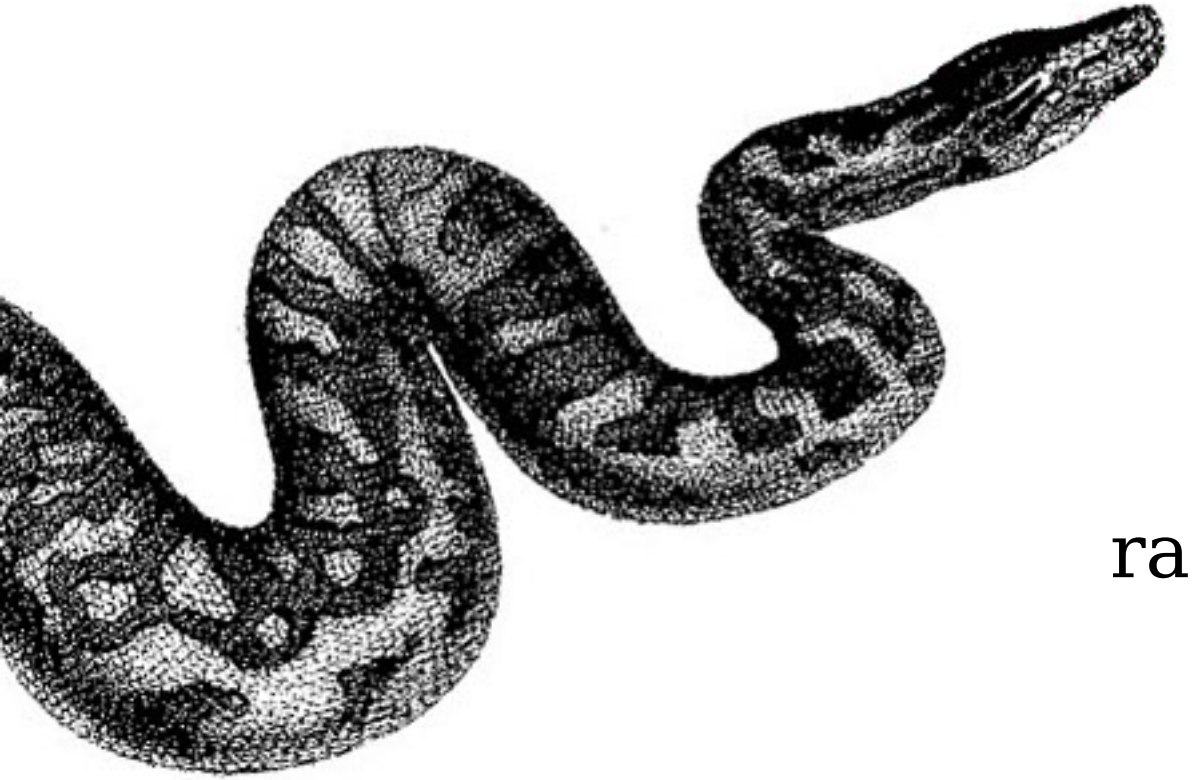
Python:
tipos básicos



Luciano Ramalho
ramalho@python.pro.br

PythonPro

Tipos de dados básicos

- Números: int, long, float, complex
- Strings: str e unicode
- Listas e tuplas: list, tuple
- Dicionários: dict
- Arquivos: file
- Booleanos: bool (True, False)
- Conjuntos: set, frozenset
- None

Números inteiros

`int`: usualmente inteiros de 32 bits

`long`: alcance limitado apenas pela memória

ATENÇÃO: a divisão entre inteiros em Python < 3.0 sempre retorna outro inteiro

```
>>> 1 / 2
0
>>> 1. / 2
0.5
```

Python promove de **`int`** para **`long`** automaticamente

```
>>> 2**30 + (2**30-1)
2147483647
>>> 2**31
2147483648L
```

PythonPro

Outros números

float: ponto-flutuante de 32 bits

complex: número complexo

Construtores ou
funções de conversão:

- **int(a)**
- **long(b)**
- **float(c)**
- **complex(d)**
- **abs(e)**

```
>>> c = 4 + 3j
>>> abs(c)
5.0
>>> c.real
4.0
>>> c.imag
3.0
```

PythonPro

Operadores numéricos

Aritméticos

básicos: **`+ - * / **`** (o último: potenciação)

aritmética de inteiros: **`% //`** (resto e divisão)

Bit a bit:

`& | ^ ~ >> <<` (and, or, xor, not, shr, shl)

Funções numéricas podem ser encontradas em diversos módulos

principalmente o módulo `math`

`sqrt(), log(), sin(), pi, radians() etc.`

PythonPro

Booleanos

Valores: **`True`, `False`**

outros valores: conversão automática

Conversão explícita: **`bool(x)`**

```
>>> bool(0)
False

>>> bool('')
False

>>> bool([])
False
```

```
>>> bool(3)
True

>>> bool('0')
True

>>> bool([[]])
True
```

PythonPro

Operadores booleanos

Operadores relacionais

`==  !=  >  >=  <  <=  is  is not`

Sempre retornam um **bool**

Operadores lógicos

`and  or`

Retornam o primeiro ou o segundo valor

Exemplo: `print nome or '(sem nome)'`

Avaliação com curto-circuito

`not`

sempre retorna um **bool**

PythonPro

None

O valor nulo e único (só existe uma instância de **None**)

Equivale a **False** em um contexto booleano

Usos comuns:

- valor default em parâmetros de funções
- valor de retorno de funções que não têm o que retornar

Para testar, utilize o operador **is**:

```
if x is None: return y
```

PythonPro

Aprendendo a aprender

Use o interpretador interativo!

Determinar o tipo de um objeto:

```
type(obj)
```

Ver docs de uma classe ou comando

```
help(list)
```

Obter uma lista de (quase) todos os atributos de um objeto

```
dir(list)
```

Listar símbolos do escopo corrente

```
dir()
```

Listas

Listas são coleções de itens heterogêneos que podem ser acessados sequencialmente ou diretamente através de um índice numérico.

Constantes do tipo lista são delimitadas por colchetes [ ]

```
a = []
b = [1,10,7,5]
c = ['casa',43,b,[9,8,7],u'coisa']
```

PythonPro

Listas

O método **`lista.append(i)`** é usado para colocar um novo item **`i`** na lista.

O método **`lista.extend(l)`** inclui todos os itens de **`l`** no final da lista. O resultado é o mesmo da expressão abaixo, só que mais eficiente pois evita copiar todos os itens da lista:

```
lista += l2
```

Função embutida **`len()`** retorna o número de itens da lista:

```
len(a), len(b), len(c) # 0, 4, ?
```

Listas

O método **`lista.sort()`** ordena os itens de forma ascendente e **`lista.reverse()`** inverte a ordem dos itens dentro da própria lista, e devolvem **`None`**.

A função embutida **`sorted(l)`** devolve uma lista com os itens de uma lista ou sequência qualquer ordenados, e **`reversed(l)`** devolve um iterador para percorrer a sequência em ordem inversa (do último para o primeiro item).

Operações com itens de listas

Atribuição

```
lista[5] = 123
```

Outros métodos da classe **list**

```
lista.insert(posicao, elemento)
lista.pop()  # +params: ver doc
lista.index(elemento) # +params: ver doc
lista.remove(elemento)
```

Remoção do item

```
del lista[3]
```

PythonPro

Uma função para gerar listas

range([inicio,] fim[, passo]**)**

Retorna uma progressão aritmética de acordo com os argumentos fornecidos

Exemplos:

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

PythonPro

Expressões para gerar listas

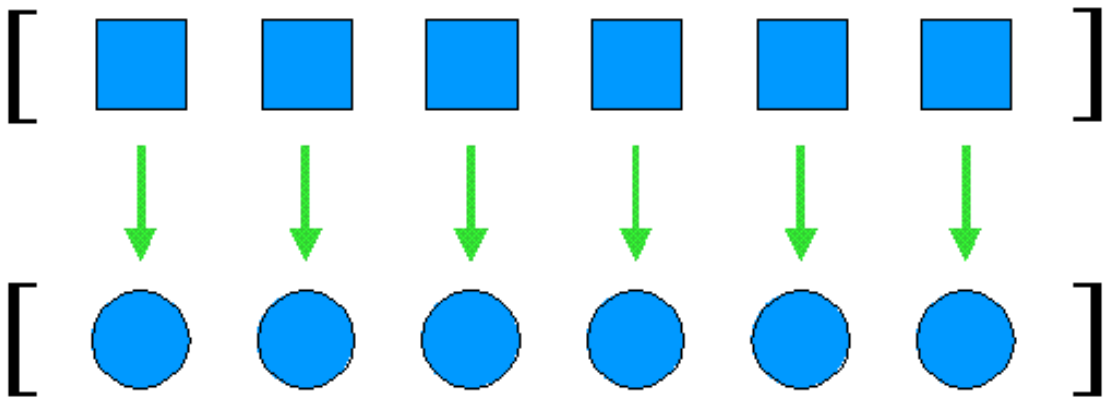
- “List comprehensions” ou “abrangências de listas”
- Produz uma lista a partir de qualquer objeto iterável
- Economizam loops explícitos

Abrangência de listas

Sintaxe emprestada da linguagem funcional Haskell

Processar todos os elementos:

```
L2 = [n*10 for n in L]
```

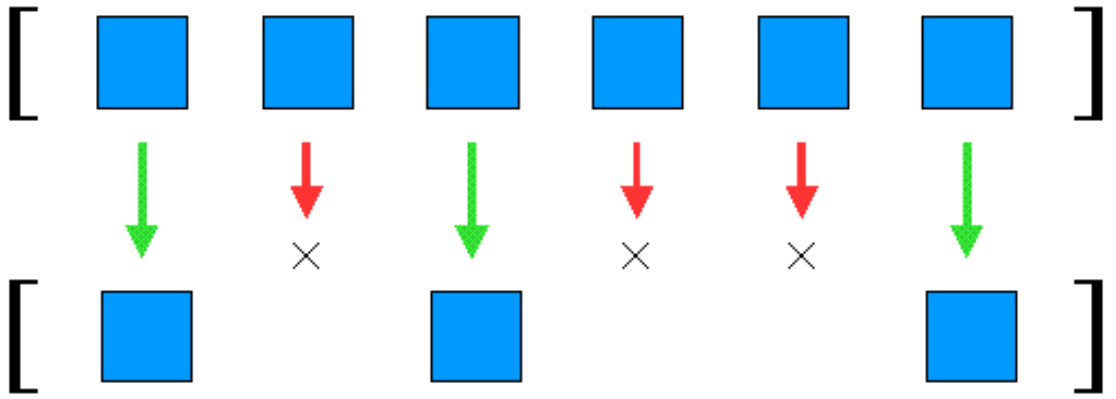


PythonPro

Abrangência de listas

Filtrar alguns elementos:

```
L2 = [n for n in L if n > 0]
```



Processar e filtrar

```
L2 = [n*10 for n in L if n > 0]
```

PythonPro

Produto cartesiano

Usando dois ou mais comandos **for** dentro de uma list comprehension

```
>>> qtds = [2,6,12,24]
>>> frutas = ['abacaxis', 'bananas', 'caquis']
>>> [(q,f) for q in qtds for f in frutas]
[(2, 'abacaxis'), (2, 'bananas'), (2, 'caquis'),
 (6, 'abacaxis'), (6, 'bananas'), (6, 'caquis'),
 (12,'abacaxis'), (12,'bananas'), (12,'caquis'),
 (24,'abacaxis'), (24,'bananas'), (24,'caquis')]
```

PythonPro

Produto cartesiano (2)

```
>>> naipes = 'copas ouros espadas paus'.split()
>>> cartas = 'A 2 3 4 5 6 7 8 9 10 J Q K'.split()
>>> baralho = [ (c, n) for n in naipes for c in cartas]
>>> baralho
[('A', 'copas'), ('2', 'copas'), ('3', 'copas'), ('4', 'copas'),
 ('5', 'copas'), ('6', 'copas'), ('7', 'copas'), ('8', 'copas'),
 ('9', 'copas'), ('10', 'copas'), ('J', 'copas'), ('Q', 'copas'),
 ('K', 'copas'), ('A', 'ouros'), ('2', 'ouros'), ('3', 'ouros'),
 ('4', 'ouros'), ('5', 'ouros'), ('6', 'ouros'), ('7', 'ouros'),
 ('8', 'ouros'), ('9', 'ouros'), ('10', 'ouros'), ('J', 'ouros'),
 ('Q', 'ouros'), ('K', 'ouros'), ('A', 'espadas'), ('2', 'espadas'),
 ('3', 'espadas'), ('4', 'espadas'), ('5', 'espadas'),
 ('6', 'espadas'), ('7', 'espadas'), ('8', 'espadas'),
 ('9', 'espadas'), ('10', 'espadas'), ('J', 'espadas'),
 ('Q', 'espadas'), ('K', 'espadas'), ('A', 'paus'), ('2', 'paus'),
 ('3', 'paus'), ('4', 'paus'), ('5', 'paus'), ('6', 'paus'),
 ('7', 'paus'), ('8', 'paus'), ('9', 'paus'), ('10', 'paus'),
 ('J', 'paus'), ('Q', 'paus'), ('K', 'paus')]
>>> len(baralho)
52
```

Tuplas

- Tuplas são sequências imutáveis
  - não é possível modificar as referências contidas na tupla
- Tuplas constantes são representadas como sequências de itens entre parenteses
  - em certos contextos os parenteses em redor das tuplas podem ser omitidos

```
a, b = b, a
```

```
>>> t1 = 1, 3, 5, 7
>>> t1
(1, 3, 5, 7)
```

PythonPro

Conversões entre listas e strings

s.split([sep[,max]])

retorna uma lista de strings, quebrando **s** nos brancos ou no separador fornecido

max limita o número de quebras

s.join(l)

retorna todas as strings contidas na lista **l** "coladas" com a string **s** (é comum que s seja uma string vazia)

```
''.join(l)
```

list(s)

retorna s como uma lista de caracteres

Tuplas

Atribuições múltiplas utilizam tuplas

```
#uma lista de duplas
posicoes = [(1,2),(2,2),(5,2),(0,3)]

#um jeito de percorrer
for pos in posicoes:
    i, j = pos
    print i, j

#outro jeito de percorrer
for i, j in posicoes:
    print i, j
```

PythonPro

Operações com sequências

Sequências são coleções ordenadas

nativamente: strings, listas, tuplas, buffers

Operadores:

`s[i]`	acesso a um item
`s[-i]`	acesso a um item pelo final
`s+z`	concatenação
`s*n`	**`n`** cópias de **`s`** concatenadas
`i in s`	teste de inclusão
`i not in s`	teste de inclusão negativo

PythonPro

Fatiamento de sequências

`s[a:b]` cópia de **`a`** (inclusive) até **`b`** (exclusive)

`s[a:]` cópia a partir de **`a`** (inclusive)

`s[:b]` cópia até **`b`** (exclusive)

`s[:]` cópia total de **`s`**

`s[a:b:n]` cópia de **`n`** em **`n`** itens

Atribuição em fatias:

```
s[2:5] = [4,3,2,1]
```

válida apenas em sequências mutáveis

Funções nativas p/ sequências

`len(s)`

número de elementos

`min(s), max(s)`

valores mínimo e máximo contido em **`s`**

`sorted(s)`

retorna um iterador para percorrer os elementos em ordem ascendente

`reversed(s)`

retorna um iterador para percorrer os elementos do último ao primeiro

Dicionários

- Dicionários são coleções de valores identificados por chaves únicas
  - Outra definição: dicionários são coleções de pares chave:valor que podem ser recuperados pela chave
- Dicionários constantes são representados assim:

```
uf={ 'PA':'Pará', 'AM':'Amazonas',
     'PR':'Paraná','PE':'Pernambuco'}
```

PythonPro

Dicionários: características

- As chaves são sempre únicas
- As chaves têm que ser objeto imutáveis
  - números, strings e tuplas são alguns tipos de objetos imutáveis
- Qualquer objeto pode ser um valor
- A ordem de armazenagem das chaves é indefinida
- Dicionários são otimizados para acesso direto a um item pela chave, e não para acesso sequencial em determinada ordem

Dicionários: operações básicas

Criar um dicionário vazio

```
d = {}
d = dict()
```

Acessar um item do dicionário

```
print d[chave]
```

Adicionar ou sobrescrever um item

```
d[chave] = valor
```

Remover um item

```
del d[chave]
```

Alguns métodos de dicionários

Verificar a existência de uma chave

```
d.has_key(c)
c in d
```

Obter listas de chaves, valores e pares

```
d.keys()
d.values()
d.items()
```

Acessar um item que talvez não exista

```
d.get(chave) #retorna None ou default
```

PythonPro

Conjuntos

Conjuntos são coleções de itens únicos e imutáveis

Existem duas classes de conjuntos:

- **set**: conjuntos mutáveis
  - suportam s.add(item) e s.remove(item)
- **frozenset**: conjuntos imutáveis
  - podem ser elementos de outros conjuntos e chaves de dicionários

PythonPro

Removendo repetições

Transformar uma lista num set e depois transformar o set em lista remove todos os itens duplicados da lista

```
l = [2, 6, 6, 4, 4, 6, 1, 4, 2, 2]
s = set(l)
l = list(s)
print l
# [1, 2, 4, 6]
```

PythonPro

Arquivos

Objetos da classe **file** representam arquivos em disco

Para abrir um arquivo, prefira **io.open()** (em vez de **open()**)

abrir arquivo binário para leitura

```
arq = io.open('/home/juca/grafico.png','rb')
```

abrir arquivo texto para leitura

```
arq = io.open('/home/juca/nomes.txt',
  encoding='utf8')
```

abrir arquivo para acrescentar (append)

```
arq = io.open('/home/juca/grafico.png','a')
```

PythonPro