Curso introdutório de Python

FURB - 2017

Quem sou eu?

Backend Developer na M2Agro;

Pythonista e pai a 3 anos, mas na área a quase 10;

Nerd padrão;

De Recife, morando em Blumenau;

csantos.machado@gmail.com

https://about.me/andresmachado

twitter: @andresmachado_

Disclaimer!

Ementa

- Aula 01 - Introdução a linguagem Python
  - Um pouco de história
  - O Zen do Python
  - O que podemos fazer com Python
  - Cases de sucesso no mundo
  - Características da linguagem
  - Tipagem e dados primitivos no Python

- Aula 02 - Indo mais a fundo
  - Python data model e os "__magicmethods__"
  - Estruturas de dados no Python e suas diferenças
  - Compreensão de listas e expressões geradoras
  - Dicionários e Manipulação de arquivos (context managers com "with")
  - Mini-projeto: Desenvolvendo uma lista de convidados Pythonica

- Aula 03 - Conclusão
  - DefaultDict e namedtuple - Estruturas de dados avançada
  - Orientação a objetos com Python
  - Uma introdução a @decorators (decoradores)
  - Introdução ao paradigma funcional com Python e o módulo functools
  - Mini-projeto: Concluindo a lista de convidados Pythonica
  - Conclusão: o que fazer agora?

Aula 01

Um pouco de história

- Seu criador foi Guido Van Rossum em 1991
- Linguagem dinâmica, interpretada e de tipagem forte
- Sim, o nome veio do grupo de humor Monty Python
- Foi baseada na linguagem ABC, uma das primeiras a utilizar o conceito de orientação a objetos
- Sua implementação principal é em C (CPython), embora existam outras implementações como o PyPy, Jython(Java), IronPython(.NET) e etc...

A evolução do Python

Zen do Python (PEP 20)

Bonito é melhor que feio.
Explícito é melhor que implícito.
Simples é melhor que complexo.
Complexo é melhor que complicado.
Linear é melhor do que aninhado.
Esparso é melhor que denso.
Legibilidade conta.
Casos especiais não são especiais o bastante para quebrar as regras.
Ainda que praticidade vença a pureza.
Erros nunca devem passar silenciosamente.
A menos que sejam explicitamente silenciados.
Diante da ambigüidade, recuse a tentação de adivinhar.
Deveria haver um — e preferencialmente só um — modo óbvio para fazer algo.
Embora esse modo possa não ser óbvio a princípio a menos que você seja holandês.
Agora é melhor que nunca.
Embora nunca freqüentemente seja melhor que *já*.
Se a implementação é difícil de explicar, é uma má idéia.
Se a implementação é fácil de explicar, talvez seja uma boa idéia.
Namespaces são uma grande idéia — vamos ter mais dessas!

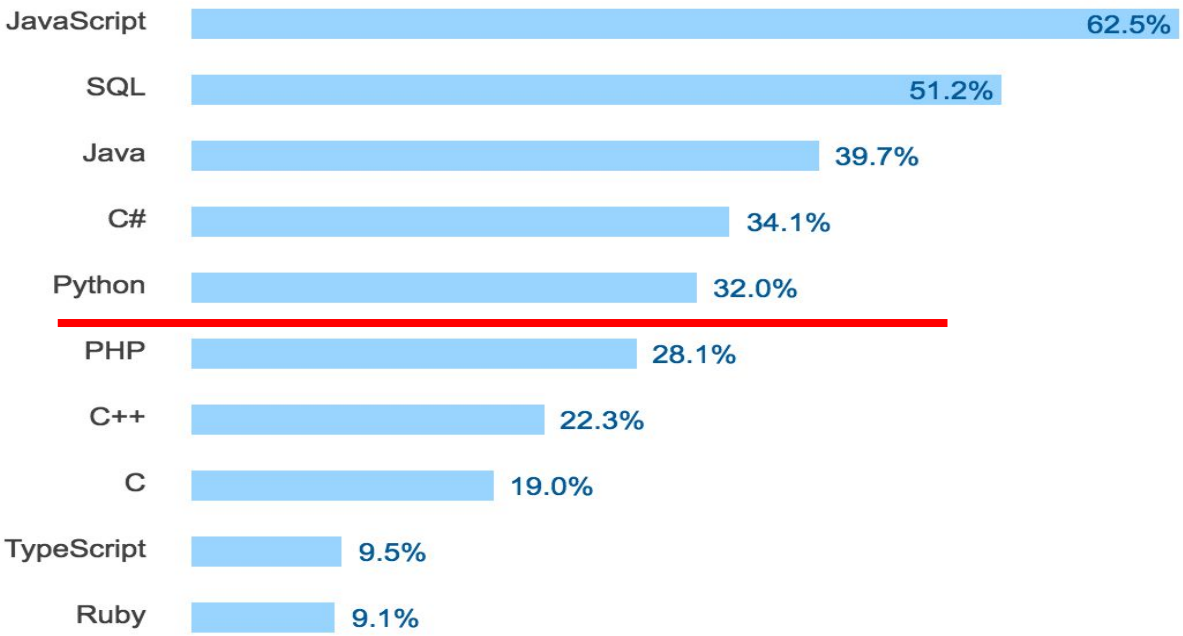
alguns números do python

https://stackoverflow.com/insights/survey/2017/

Most Popular Technologies

Programming Languages

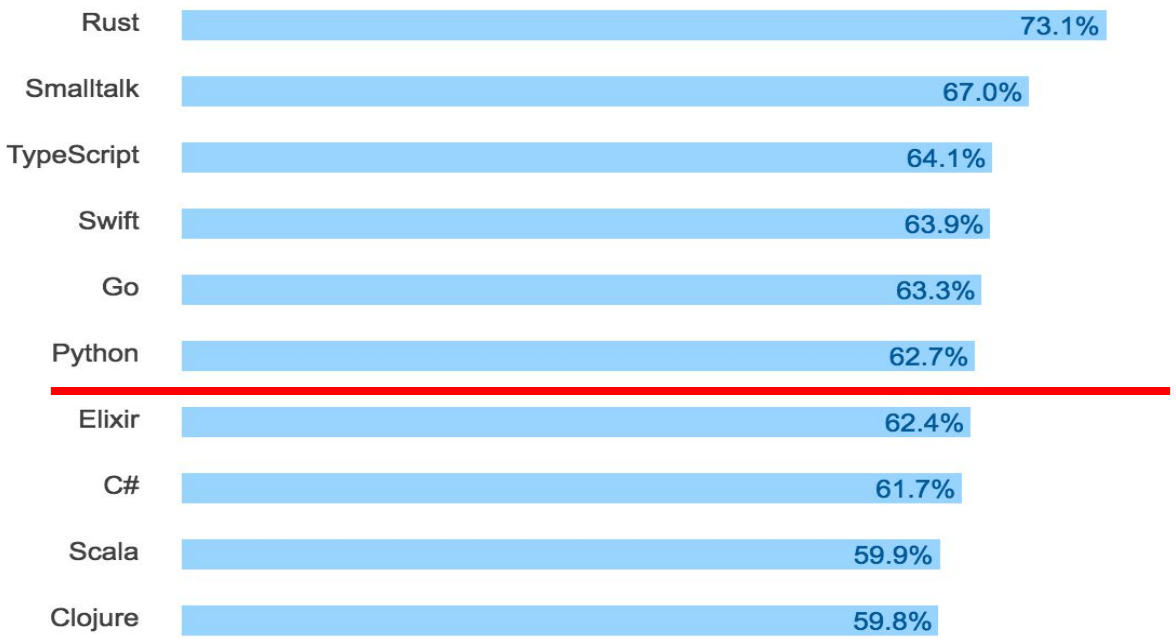
% of This Category | % of All Respondents



Most Loved, Dreaded, and Wanted

Most Loved, Dreaded, and Wanted Languages

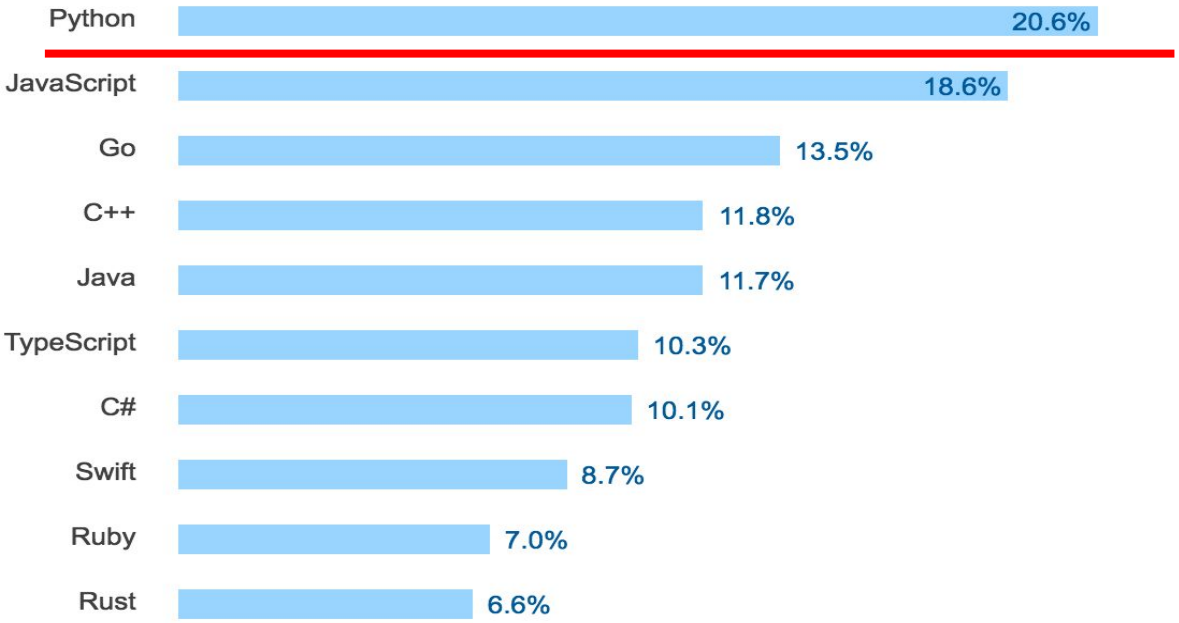
Loved | Dreaded | Wanted



Most Loved, Dreaded, and Wanted

Most Loved, Dreaded, and Wanted Languages

Loved | Dreaded | Wanted



desenvolvimento de aplicações web...

processamento e manipulação de dados, Data Mining e deep learning...

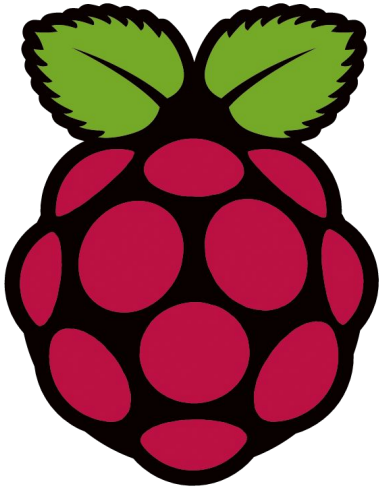
desenvolvimento de games...

aplicações para desktop e mobile...

tkinter

pygtk

sistemas embarcados...



Cases de sucesso do Python

Características da Linguagem

Indentação

- A indentação não é apenas visual, se o código estiver indentado errado o programa não roda (e o interpretador alerta);
- É muito parecido com escrever pseudocódigo;
- Esta obrigação mantém a legibilidade e facilidade de escrita do código;

```
for i in range(100):
    if i % 5 == 0 and i % 3 == 0:
        print("FizzBuzz")
    elif i % 3 == 0:
        print("Fizz")
    elif i % 5 == 0:
        print("Buzz")
    else:
        print(i)
```

Portabilidade

- Windows
- UNIX
- Android
- JVM
- .NET

- CLI
- Desktop App
- WebApp
- Microservices

Facilidade de escrita

```
from datetime import datetime

name = "Andre"
now = datetime.now()
birthday = datetime(1987, 05, 14)
diff = now - birthday
age = diff.days // 365

print("Meu nome é %s e eu tenho %d anos" % (name, age))
```

Multiparadigma

- Orientado a objetos
- Funcional
- Procedural
- Herança Múltipla (MRO)

Batteries Included

- Strings
- Data Types
- Math
- File and Directory Access
- Persistência de dados
- Compressão de dados
- File Formats
- Logging
- Testing (unittest, doctest)
- Raspagem de dados
- Protocolos de Internet
- Multimedia

Tipagem e dados primitivos

Tipagem

Python tem tipagem dinâmica, porém forte;

Você pode alterar o tipo de uma variável em tempo de execução e passar qualquer tipo de valor dentro de funções e métodos;

Por causa de sua tipagem forte, ele não realiza conversões automáticas de tipos

```
>>> "a" + 10
TypeError: Can't convert 'int' object to str implicitly
```

A menos que seja explicitamente convertido

```
>>> "a" + str(10)
'a10'
```

Tipos "built-in"

- Strings
- Integers
- Dicts
- Lists
- Tuples
- Boolean
- Float
- Long
- Sets
- Docstrings

Strings

Strings são indexadas como se fossem arrays (Index Zero)

É possível acessar cada item separadamente, porém, strings são imutáveis e não aceitam atribuição de valores;

Strings aceitam "slicing":

```
>>> a_string = 'Python is a beautiful programming language'
>>> len(a_string)
42
>>> a_string[0:6] # Retorne os valores do index 0 até o indice 2
'Python'
>>> a_string[6:22] # A partir do indice 3 (inclusivo) até o indice 5 (exclusivo)
' is a beautiful '
>>> a_string[6:] # Você pode omitir o indice final para considerar: 'Toda a string  a partir do indice 6'
' is a beautiful programming language'
>>> nova_lista[::-1] # Hack importante! Você pode inverter os valores da lista passando um indice negativo no parametro 'stride';
'egaugnal gnimmargorp lufituaeb a si nohtyP'
```

Formatação de Strings

O método format() da API de strings permite a formatação de diversas maneiras

SHUT UP AND SHOW ME THE CODE!

Formatação de Strings - Formatos

tipo		Um dos caracteres abaixo; **d** é o default para exibir **int**, **g** para **float** e **s** para todos os demais:
	s	**str/unicode**
	b	**int** como binário
	c	**int** como caractere Unicode correspondente
	d	**int** como decimal
	o	**int** como octal
	x X	**int** como hexadecimal: **x** caixa baixa, **X** alta
	e E	**float** em notação exponencial: **e** caixa baixa, **E** alta
	f F	**float** sem usar notação exponencial
	g G	**float** como **e E** ou **f F**, conforme a magnitute, mas sem zeros não significativos
	n	**float** como no tipo **g**, usando separadores decimal e de milhares conforme o locale ativo
	%	**float** como porcentagem, usando formato do tipo **f**, com o valor ×100, seguido do sinal **%**

Listas (Arrays)

Listas também são zeroth-index (indice 0);

Ao contrário das strings, elas são bem mais flexíveis quanto a atribuição de valores e por isso são mutáveis;

Listas possuem tamanho indefinido;

Também são heterogêneas, você pode ter diversos tipos de valores dentro da sua lista;

Listas (Slicing)

Assim como as strings, listas também aceitam slice - list(start:end:stride)

>>> nova_lista = 'Python is a beautiful programming language'.split(')
['Python', 'is', 'a', 'beautiful', 'programming', 'language']
>>> nova_lista[0:2] # Retorne os valores do index 0 até o índice 2
['Python', 'is'] # O índice 2 ('a') é exclusivo, por isso não está no retorno
>>> nova_lista[3:5] # A partir do índice 3 (inclusivo) até o índice 5 (exclusivo)
['beautiful', 'programming']
>>> nova_lista[3:] # Você pode omitir o índice final para considerar: 'Toda a lista a partir do índice 3'
['beautiful', 'programming', 'language']
>>> nova_lista[::-1] # Hack importante! Você pode inverter os valores da lista passando um índice negativo no parâmetro 'stride';
['language', 'programming', 'beautiful', 'a', 'is', 'Python']
>>> nova_lista[-2:] # Passar índices negativos no parâmetro 'start' fatia a lista de trás pra frente;
['programming', 'language']

Listas (Ordenando)

O jeito mais fácil de ordenar uma lista é utilizando o método sorted()

O sorted() aceita argumentos para modificar o seu comportamento;

O antigo método list.sort() também funciona, mas ele não é "puro", pois modifica a lista *in-place,* por isso é recomendado o sorted() por não alterar o valor original da lista;

Exercícios de Fixação - Aula 01