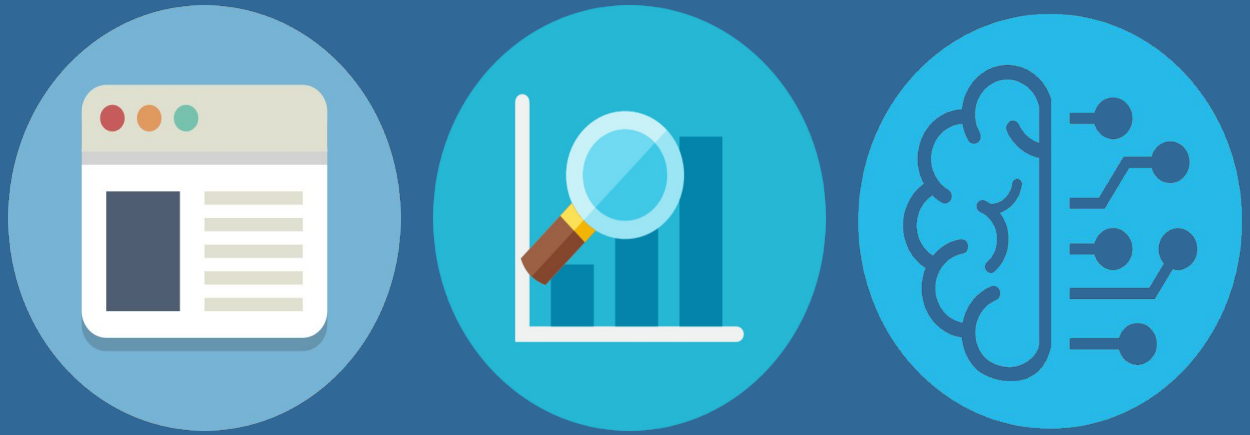
Python
Conhecendo a linguagem

DEVMEDIA

Linguagem de programação

- propósito geral
- multiplataforma
- multiparadigma



Sintaxe da linguagem

```
class Csv:
    def imprimirUsuarios(self, lista):
        if len(lista):
            usuarios = ""
            for usuario in lista:
                usuarios += usuario + ", "
            print(usuarios)


ser = Csv()
ser.imprimirUsuarios(["Fulano", "Ciclano", "Beltrano"])
```

```
Fulano, Ciclano, Beltrano,
```

Uma pequena comparação

```java
package br.com.devmedia.aprendendojava.aplicacao;

public class Program {

    public static void main(String[] args) {
        User padeiro = new User("Manuel", 1500);
        padeiro.setSalario(2000);
        System.out.print(padeiro.getSalario());
    }
}
```

```python
from user import User

if __name__ == "__main__":
    padeiro = User("Manuel", 1500)
    padeiro.salario = 2000
    print(padeiro.salario)
```

Múltiplos Paradigmas

DEVMEDIA

```
def processar(dados):
```

```
class Programador:
```

```
lambda item: item.valor
```

DEVMEDIA

Procedural

```
def realizar_operacao(operacao):
    return "realizado: " + operacao

if __name__ == "__main__":
    tarefa = realizar_operacao("assistir")
    print(tarefa)
```

```
realizado: assistir
```

Orientado a objeto

```
class Operacao:
    def realizar(self, args):
        return "realizado: " + args

operacao = Operacao()
print(operacao.realizar("comer"))
```

```
realizado: comer
```

DEVMEDIA

Funcional

```
def estudar(callback):
    return callback("estudar")

tarefa = estudar(lambda args: "realizado: " + args)
print(tarefa)
```

```
realizado: estudar
```

Python permite truques poderosos

```
palavra = "programadores"
lista = []
for letra in palavra:
    lista.append(letra)
```

```
['p', 'r', 'o', 'g', 'r', 'a', 'm', 'a', 'd', 'o', 'r', 'e', 's']
```

Por exemplo compreensão de listas

```
palavra = "programadores"
lista = [letra for letra in palavra]
```

```
['p', 'r', 'o', 'g', 'r', 'a', 'm', 'a', 'd', 'o', 'r', 'e', 's']
```

E slicing de listas

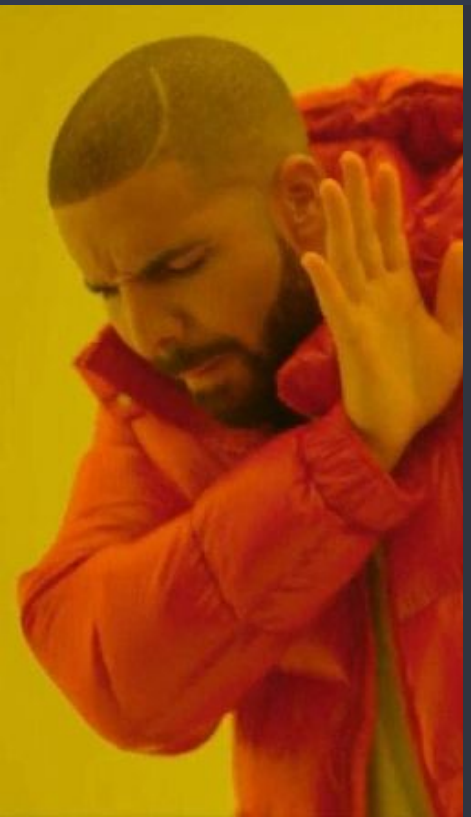
```
texto.substring(0, 13);
```


```
substr($texto, 0, 13);
```

```
texto[:13]
```

"programadores não vivem sem café"

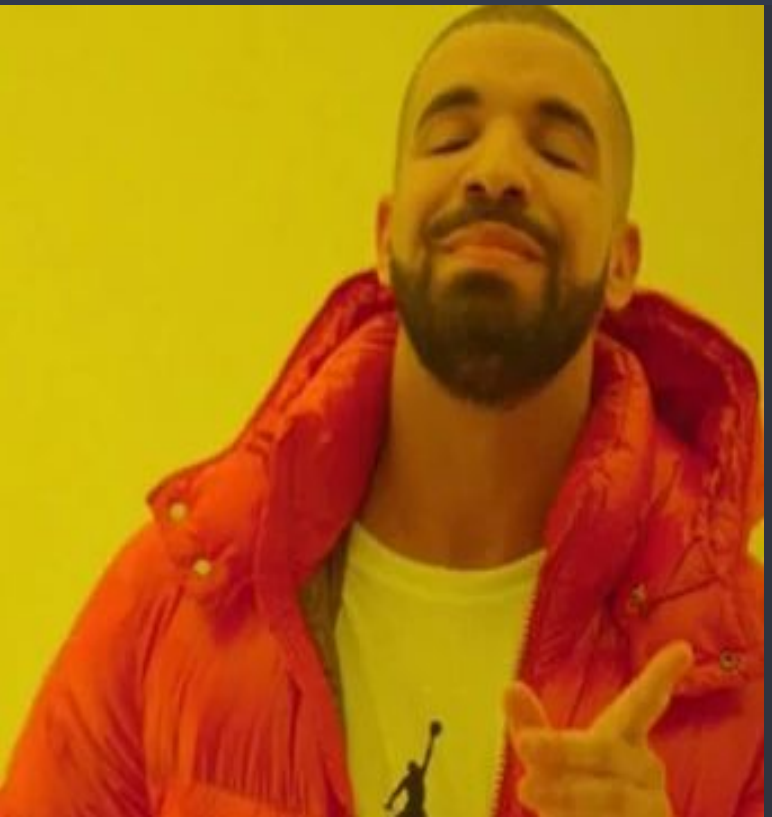
programadores
13 letras



Mas que podem ser confusos

```python
def calcular(args):
    return sum([x["valor"] * x["quantidade"] for x in args if isinstance(x, dict)])
```

Jeito Pythônico



```python
def calcular_carrinho_compras(carrinho):
    lista_compras = []
    for produto in carrinho:
        if isinstance(produto, dict):
            lista_compras.append(produto["valor"] * produto["quantidade"])

    return sum(lista_compras)
```

Filosofia e o jeito Pythonico de programar

Legibilidade é fundamental, já que código é quase sempre mais lido que escrito.

Código legível == código reutilizável

“Zen of Python”

DEVMEDIA

Bonito é melhor que feio.

Explícito é melhor que implícito.

Simples é melhor que complexo.

Complexo é melhor que complicado.

Reto 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, pode ser uma boa idéia.

Namespaces são uma grande idéia — vamos ter mais dessas!

Exemplo em Python
Em execução