

Python

Aplicado na Web

Frederico Minuzzi¹, Vinicius Gasparini¹

¹Centro de Ciências Tecnológicas – Universidade Estadual de Santa Catarina (UDESC)
Joinville – SC – Brasil

freddyminu@gmail.com, v.gasparini@edu.udesc.br

Resumo. *O presente trabalho tem como objetivo apresentar a linguagem de programação Python, seus princípios e diferenciais, e também comentar sobre o uso de ferramentas para desenvolvimento de aplicações web.*

1. Historia

Python é uma linguagem de programação criada por Guido van Rossum, um holandês que trabalhava no Instituto de Pesquisa Nacional para Matemática e Ciência da Computação dos países baixos em meados de 1991. Na época, Rossum trabalhava com ABC, uma linguagem altamente não modular e de difícil importação e exportação de código.



Figura 1. Guido van Rossum

Rossum ao modelar a linguagem, utilizou alguns princípios de *design* que até então eram pouco aplicados a computação. Essas práticas foram descritas na publicação *The Zen of Python* [3] e podemos destacar entre elas:

- *Beautiful is better than ugly*
- *Explicit is better than implicit*
- *Simple is better than complex*
- *Complex is better than complicated*
- *Readability counts*

2. Vantagens e desvantagens

2.1. Vantagens

O Python por ser uma linguagem de alto nível contém uma sintaxe extremamente intuitiva e de fácil entendimento [4]. A modularidade do Python faz com que seja fácil de importar e exportar código, além disso, Python é a linguagem mais popular do mundo atual fazendo com que exista inúmeras bibliotecas.[2]

2.2. Desvantagens

Decorrente do fato de ser uma linguagem interpretada, muita memória é consumida no processo de traduzir o código escrito para linguagem de máquina. O fato que o Python é apenas single core (causado pelo interpretador) faz com que não há maneira de dividir a carga de trabalho e com isso o tempo de execução é muito superior ao seus competidores. Vale ressaltar porém que em alguns casos o tempo da implementação do código faz essa relação ser vantajosa. [5]

3. Especialização

Python foi criado com o objetivo de ser altamente extensional/modular, ou seja, a linguagem em si não havia muita funcionalidade mas ela facilitava o uso de bibliotecas e *frameworks*. O fato que o Python é a linguagem muito popular faz com que esta modularização seja muito bem aproveitada com a comunidade aquecida, existindo milhares de módulos permitindo inúmeras funcionalidades, desde para aplicações web até de renderização de vídeos.

4. Multi-paradigma

O fato que o Python é uma linguagem multi-paradigma traz bastante versatilidade para seu uso. Uma vez que não somente pode-se explorar cada uma das vantagens de cada paradigma, podemos mesclar em um mesmo código diversos conceitos dos paradigmas funcional, imperativo, orientado a objetos, estruturado e reflexivo.

5. Frameworks Web

5.1. Django

Django é um *framework* de desenvolvimento web *server-side*, isto é, preocupa-se principalmente com o fluxo de requisições para processamento de dados. Facilmente acoplável com outras tecnologias de *front-end* por meio de API, o Django é uma das principais ferramentas dentro do ecossistema Python.

Mantido desde 2005 por um grupo de voluntários, a Django Software Foundation [1], o Django é *open source*, conta com uma documentação vasta tanto para os iniciantes quanto para usuários avançados.

5.2. Versátil e escalável

Django é usado para muitos propósitos, desde prototipagem e criação de MVPs até o desenvolvimento de aplicativos personalizados e construção de grandes plataformas. Django permite um desenvolvimento rápido para a web, portanto grandes empresas como Dropbox, YouTube, Instagram, PayPal, eBay e Reddit são todos desenvolvidos em Django.

Django pode escalar facilmente. Seus componentes são desacoplados, o que significa que podem ser adicionados ou removidos quando necessário, assim como peças de Lego. Dependendo dos requisitos específicos do produto, o desenvolvimento pode ser ampliado ou reduzido. Você pode alterar facilmente o número e a complexidade dos componentes do Django.

5.3. Arquitetura

Baseado na arquitetura MVT (*model-view-template*), bastante semelhante ao MVC, o Django se destaca pela boa definição de responsabilidade para cada camada desse sistema. Os *models* são responsáveis por manipular e estruturar os dados da aplicação. As *views* encapsulam a lógica responsável pelo processamento da requisição do usuário e se encarregam de posteriormente responder ao cliente. Por fim, a camada de *template* fornece suporte a uma sintaxe amigável para renderizar informações que serão apresentadas ao usuário.

Ainda dentro do contexto de arquitetura, o Django opera sob o princípio de *request phase* e *response phase*.

Durante a *request phase*, ocorre a admissão da requisição pelo servidor, resolução de URL e processamento da *View*. Durante cada etapa dessas transições, a informação pode sofrer alteração por meio dos *middlewares*. *Middlewares* são funções que tem como responsabilidade preparar o dado para processamento da próxima etapa ou até mesmo operar sob esse recebido e dar uma saída alternativa.

Após a *request phase*, entra em ação a *response phase*. Nela a requisição aciona a criação de um *Model* quer se comunica, na maior parte das vezes, com o banco de dados (ORM) e então o dado prossegue para a geração do *Template*. Aplicações do tipo API não possuem como saída uma página html, portanto essa etapa de retorno consiste basicamente de responder ao servidor o dado processado.

Esse fluxo pode ser melhor observado na Figura 2.

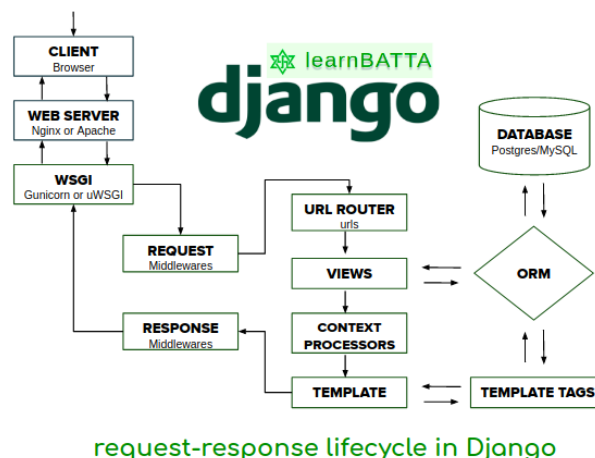


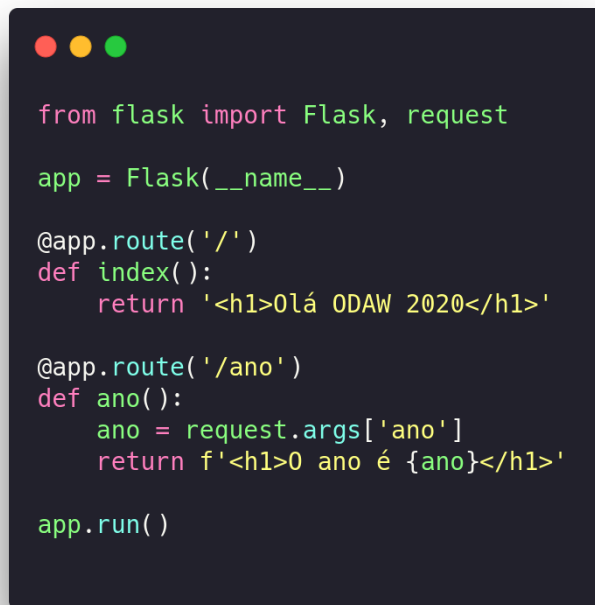
Figura 2. Ciclo Request-response do Django.

6. Flask

Flask é uma alternativa mais simples, e consequentemente mais leve, rápida e barata de se desenvolver *web apps* em Python. Tem como maior princípio a ser enxuto e altamente customizável.

Se você está focado na experiência e nas oportunidades de aprendizado ou se deseja ter mais controle sobre quais componentes usar (como quais bancos de dados deseja usar e como deseja interagir com eles), o Flask se mostra a escolha perfeita. [6]

Toda simplicidade pode ser vista no exemplo Figura 3, onde definimos uma rota para a base da aplicação em '/'. Em '/ano' ao enviarmos como parâmetro na requisição um ano, esse dado é embutido durante a renderização do documento html.



```
from flask import Flask, request

app = Flask(__name__)

@app.route('/')
def index():
    return '<h1>Olá ODAW 2020</h1>'

@app.route('/ano')
def ano():
    ano = request.args['ano']
    return f'<h1>0 ano é {ano}</h1>'

app.run()
```

Figura 3. Aplicação *single-page* simples em Flask.

Referências

- [1] Django Software Foundation. *About*. 2005. URL: <https://www.djangoproject.com/foundation/> (acesso em 24/02/2021).
- [2] Stack Overflow. *2020 Developer Survey*. 2020. URL: <https://insights.stackoverflow.com/survey/2020> (acesso em 24/02/2021).
- [3] Tim Peters. *PEP 20 – The Zen of Python*. 2004. URL: <https://www.python.org/dev/peps/pep-0020/> (acesso em 24/02/2021).
- [4] Guido van Rossum. *Python's Design Philosophy*. 2009. URL: <https://python-history.blogspot.com/2009/01/pythons-design-philosophy.html> (acesso em 24/02/2021).
- [5] Danilo Morais da Silva. “Python: História e Ascendência”. Em: *PROGRAMAR* 58 (2018), pp. 96–99.
- [6] Evandro Souza. *Flask RESTplus*. 2018. URL: <https://medium.com/trainingcenter/flask-restplus-ea942ec30555> (acesso em 24/02/2021).