



- Introdução
- Implementação
- Testes
- Cliente com Python
- Hospedagem no Heroku
- Mais

# Python

## Criando WebServices com Python

Usando Flask

### 01 Introdução

- Arthur Abella é um programador que gostaria de criar uma aplicação em Python para conversão de reais a dólares.
- Inicialmente, ele fez o código a seguir.

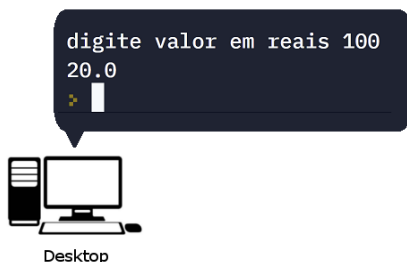


```
main.py ×
1 ▼ def converterParaDolares():
2     reais = float(input('digite valor em reais'))
3     dolares = reais / 5
4     print(dolares)
5
6 converterParaDolares()
```

- Inicialmente, Arthur ficou muito feliz e mostrou o resultado ao seu pai, que fez as seguintes considerações.

#### Consideração 1

- O aplicativo funciona em computadores (*Desktop*). Ou seja, se amanhã você quiser criar um aplicativo para celular (*mobile*) ou web (via *browser*) que chame este método não vai funcionar.



#### Consideração 2

- Tão breve Daniel Abella fez a primeira consideração, seu primogênito o questionou o motivo.
  - O motivo é que, dentro do método não é interessante colocar leitura de dados (com input, por exemplo), nem tampouco apresentação de dados (com print, por exemplo).
  - Oportunamente, Arthur apresentou uma nova versão do seu código.

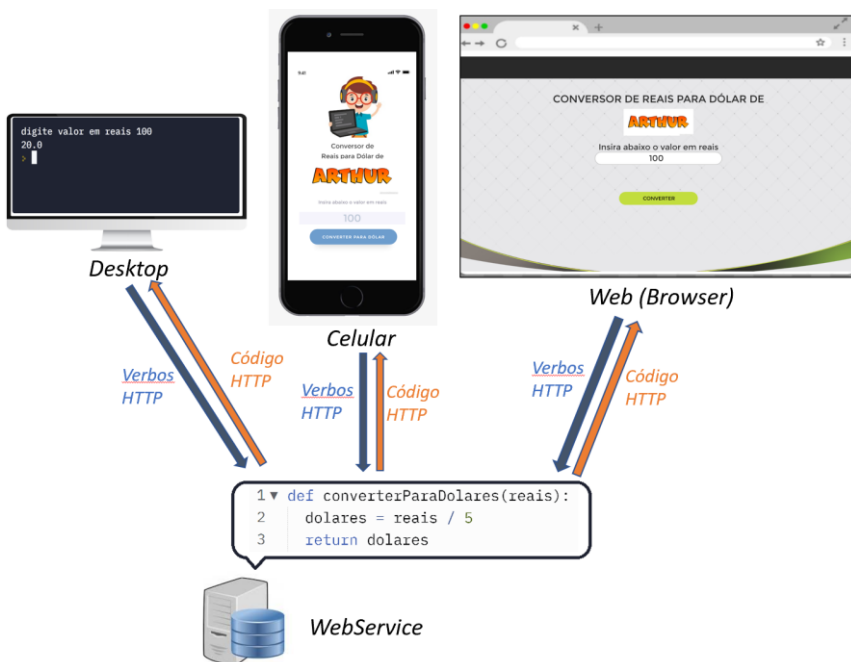
main.py ×

```
1 ▼ def converterParaDolares(reais):
2     dolares = reais / 5
3     return dolares
4
5 valor = float(input('digite valor em reais '))
6 valorEmDolares = converterParaDolares(valor)
7 print(valorEmDolares)
```

- Agora o código está elegível para ser chamado por celulares e web, pois a leitura de dados é fora do método (linha 5) e a apresentação da conversão é realizada também fora do método (na linha 7).

#### Consideração 3

- Arthur ficou empolgado com a ideia de fazer o aplicativo dele funcionar em celulares, web e desktops. E, questionou como isto poderia funcionar. Sendo assim, Daniel definiu alguns passos para fazer.
- **Passo 1:** A primeira coisa é transformar o seu método converterParaDolares em um webservice, que é o que vamos explicar neste resumo.
  - De maneira simplória, **imagine webservice como um método que pode ser chamado a partir de qualquer aplicação**, seja ela para celular, computador ou browser.
- **Passo 2:** Desenvolver 3 aplicações (um para celular, um para web e uma para desktops) que fazem a conversão de reais para dólares.
  - E, fazer com que, esses 3 aplicativos chamem o webservice desenvolvido no passo 1.
- Arthur, ligado no 220 como o pai, desenhou a ideia proposta por seu pai, apresentado na folha seguinte.
- Arthur detalhou a sua proposta, que é de:
  - Desenvolver uma aplicação para celulares usando Java (ou outras linguagens), que chama o webservice que faz a conversão
  - Desenvolver uma aplicação para computadores usando Python (ou outras linguagens), que chama o webservice que faz a conversão
  - Desenvolver uma aplicação para browsers usando Javascript (ou outras linguagens), que chama o webservice que faz a conversão



- E, quando o WebService recebe algum dos pedidos anterior (verbos HTTP), ele pode responder da seguinte forma:

Código	Significado
200	WebService para Cliente: "OK!" ou WebService para Cliente: "OK, toma aí o que você me pediu (lista de usuários)"
201	WebService para Cliente: "Created" pois, inseri um novo usuário no sistema
204	WebService para Cliente: "No content" pois, não achei o que você me pediu
400	Bad Request
404	Not Found
500	WebService para Cliente: "Internal Server Error" pois deve ter dado um pau da porra no webservice

- Para conhecer todos os códigos, eu sugiro conhecer o site <https://httpstatusdogs.com/>

## 02 Implementação

- Vamos tomar o sonho de Arthur uma realidade?
  - Vamos implementar um WebService que realiza a conversão de reais para dólar.

### Detalhes Técnicos

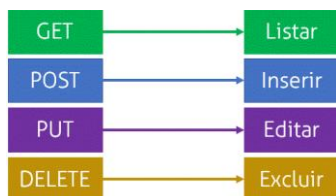
- Como vimos, os 3 aplicativos podem ser feitos em qualquer linguagem e podem chamar o nosso webservice `converterParaDolares`, que foi implementado em Python.
  - Como isso é possível? Vamos utilizar o protocolo HTTP, que possivelmente vocês não conhecem a fundo, mas é o que permite você acessar a Web.
- No HTTP, temos verbos que são a maneira com que o cliente (neste caso, um dos 3 aplicativos) chama o webservice.



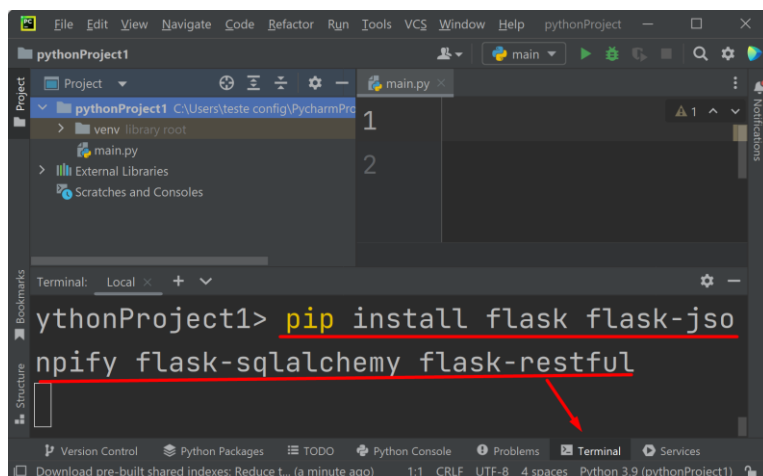
- A partir de agora, quando eu usar a palavra **cliente** (ou *client*, em inglês), estou me referindo "ao cara" que chama o webservice.
- Em nosso exemplo, temos 3 cliente para o webservice `converterParaDolares`: Um aplicativo para celular, um aplicativo para web (browser) e um aplicativo para desktop.

- O cliente chama o webservice de uma das maneiras a seguir:

Verbo	Significado
GET	Cliente para WebService: "Ei, me dá a lista de todos os usuários que tem no sistema"
POST	Cliente para WebService: "Ei, toma os dados de um novo usuário, podes inserir?"
PUT	Cliente para WebService: "Ei, tem como alterar os dados do usuário 123? Toma!"
DELETE	Cliente para WebService: "Ei, tem como excluir todos os dados do usuário 123?"



- Passo 1:** Cria um projeto no Pycharm
- Após criado o projeto, na parte inferior do *Pycharm*, conforme indicado na seta da imagem a seguir, clique em Terminal. Posteriormente, insira o seguinte comando: `pip install flask flask-jsonify flask-sqlalchemy flask-restful`



- Por meio deste comando, você instalará as seguintes dependências
  - Flask:** Flask é um micro framework que utiliza a linguagem Python para criar aplicativos Web.
  - Flask-Restful:** Extensão para Flask que adiciona suporte para a construção rápida de APIs REST. É uma abstração leve que funciona com seu ORM/bibliotecas existentes. Flask-RESTful incentiva as melhores práticas com configuração mínima. Se você estiver familiarizado com o Flask, o Flask-RESTful deve ser fácil de entender.

- **Passo 2:** Importar as dependências necessárias para criar um webservice e relacionar as variáveis necessárias na aplicação

```
1 from flask import Flask, request, jsonify
2 from flask_restful import Resource, Api
3
4 app = Flask(__name__)
5 api = Api(app)
```

- **Passo 3:** Incluir o nosso método sem nenhuma alteração

```
8 def converterParaDolares(reais):
9     dolares = reais / 5
10    return dolares
```

- **Passo 4:** Criamos o nosso webservice que chama o método do passo 3

```
13 class Cotacao(Resource):
14     def post(self):
15         valorEmReal = float(request.json['valor'])
16         result = converterParaDolares(valorEmReal)
17         return jsonify(result)
```

- Como vimos anteriormente, temos diversos métodos HTTP que mostram como o cliente chama o webservice. Ou seja, sabemos que o cliente vai nos chamar. Neste sentido, precisamos definir qual (ou quais) dos verbos HTTP que vamos atender.
- Especificamente para a conversão, creio que devemos implementar um webservice que atenda o verbo POST, uma vez que ele vai mandar um dado (valor em reais).
- Para isto, o Flask exige a criação de uma classe, conforme podemos ver a seguir a classe `Cotacao`. E, criamos um método cujo nome é o verbo HTTP, neste caso `post`. E, como todo método dentro de classe, tem que ter como primeiro parâmetro o `self`.
- No corpo do método, na linha 15, o trecho `request.json['valor']` resgata o valor informado no campo valor pelo cliente (uma das 3 aplicações). Ainda na linha 15, convertemos este valor para float e na linha seguinte (16), chamamos o nosso método `converterParaDolares`, que faz propriamente a conversão e devolve ao cliente.
- Mas, pera aí, o que tem na linha 17? `jsonify(result)`? Basicamente converte o resultado para um JSON. Mas, pera aí, o que é JSON?



Como vimos anteriormente, os clientes vão chamar o nosso WebService. E, os clientes não precisam ser implementados em Python, confere? Podem ser em qualquer linguagem. Desta maneira, precisamos que: Sempre que enviar e receber dados do (e para) os webservices, temos que ter uma linguagem que todo mundo entenda, que é a JSON (O inglês da programação).

JSON significa *Javascript Object Notation* (Notação de Objetos em Javascript) e é amplamente utilizado. JSON é bem parecido com dicionários, pois guarda uma série de chaves e valor.

Dict



JSON

```
{'name': 'Apple'
 'color': 'Red'
 'quantity': 10 }
```

```
{"name": "Apple", "color":
 "Red", "quantity": 10 }
```

Fonte: <https://favtutor.com/blogs/dict-to-json-python>

```
20 api.add_resource(Cotacao, '/cotacao')
21
22 if __name__ == '__main__':
23     app.run()
```

- A linha 20, nos diz que, se você chamar o webservice no endereço `http://127.0.0.1:5000/cotacao`, ele deve chamar a classe `Cotacao`. Agora é só executar, pessoal! Ao executar, espera-se a seguinte saída no console:

```
* Running on http://127.0.0.1:5000 (Press CTRL+C to quit)
* Serving Flask app 'main' (Lazy loading)
* Environment: production
WARNING: This is a development server. Do not use it in a production deployment.
Use a production WSGI server instead.
* Debug mode: off
```

- As linhas 22 e 23, pode inserir lá, antes de rodar né 😊

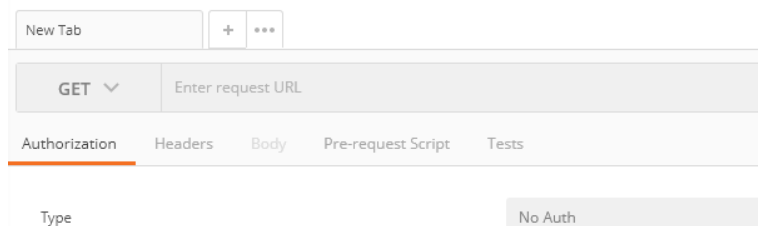
## 03 Testes

- Agora que implementamos um *webservice*, precisamos testar. E, quando me refiro a testar, me refiro a simular um cliente chamando o WebService para verificar se a resposta é adequada.
- Para isto, vamos utilizar o **POSTman**, que é uma ferramenta que simula clientes.

- **Passo 1:** Instalar o POSTman

- Acesse o endereço <https://www.postman.com/downloads/>

- **Passo 2:** Criar uma *request* (requisição) que simule um cliente
- Clicamos no botão **NEW**
- Depois clique na opção **GET** **Request** para criar a nossa primeira request.
- Na janela que abre, clique em **X**, pois estes passos são desnecessários para o momento. A janela a seguir é apresentada.

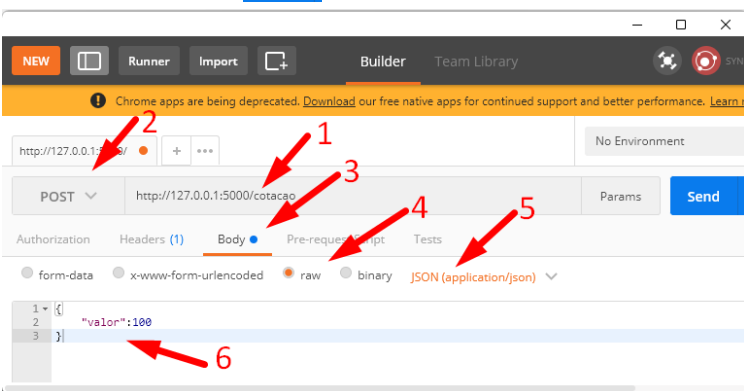


- Primeiro, você deve lembrar qual a URL do seu webservice, cuja formação é apresentada a seguir.

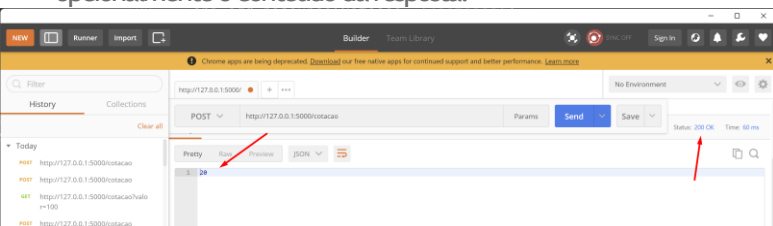
`http://127.0.0.1:5000/cotacao`

```
20 api.add_resource(Cotacao, '/cotacao')
21
22 if __name__ == '__main__':
23     app.run()
```

- **Passo 3:** Realizar a primeira requisição
- Conforme imagem a seguir, devemos preencher a requisição da seguinte maneira:
- **Seta 1:** Colocamos o endereço do Webservice (vimos como fazer no passo 2)
- **Seta 2:** Colocamos o verbo HTTP, lembra que usamos POST? Linha 14 do código
- **Seta 3:** Clicamos em Body (Corpo) para preencher o corpo requisição com alguma informação
- **Seta 4:** Clicamos em raw (crú) para informar que o dados que vamos informar é cru
- **Seta 5:** E o dado raw (crú) é em formato JSON
- **Seta 6:** Escrevemos em formato JSON os dados que o cliente quer enviar para o webservice
  - Na linha 15 do código, recuperamos o valor enviado pelo cliente na Seta 6
- Por fim, clique em **Send**



- Ao clicar no botão **Send**, a tela a seguir é apresentada na qual uma seta indica que a resposta da chamada ao webservice foi o código 200, que significa OK. E, na outra seta, verificamos o conteúdo da resposta. Ou seja, sempre vem o código e, opcionalmente o conteúdo da resposta.



**DON'T FORGET!** Baseado no código e no conteúdo de resposta (opcional), o cliente (celular, browser ou desktop) apresenta uma tela com o resultado.

## 04 Implementando um Cliente em Python

- Cliente, como informamos anteriormente, é quem chama o webservice, neste caso, pode ser um aplicativo para celular, desktop ou web.
- Agora, vamos criar um exemplo de um código em Python de uma aplicativo que chama o nosso webservice.

- **Passo 1:** Crie um projeto novo no Pycharm
- **Passo 2:** Instale as dependências necessárias por meio do comando a seguir: `python -m pip install requests`
- **Passo 3:** Chame o webservice da maneira apresentada a seguir.

```
1 import requests
2
3 enderecoWebservice = "http://127.0.0.1:5000/cotacao"
4 conteudo = {"valor": 100}
5 resposta = requests.post(enderecoWebservice, json=conteudo)
6 resposta.json()
```

- Linha 1: Importamos a dependência necessária
- Linha 3: Uma string com o endereço do webservice
- Linha 4: Um JSON com um campo valor com valor 100
- Linha 5: Chamamos um POST para o endereço da linha 3, passando o valor em JSON da linha 4
- Linha 6: Obteremos o resultado da chamada

## 05 Hospedar o Webservice

- Até o momento, rodamos o Webservice em nossa máquina, por isso o endereço continha 127.0.0.1, que é o endereço da máquina local. Que tal implantar em um servidor? Você consegue!
- <https://www.geeksforgeeks.org/deploy-python-flask-app-on-heroku/>
- <https://medium.com/data-hackers/deploy-flask-api-using-heroku-d5c7128481b7>

## 06º que tem mais?

- Apresentamos a você, em pouco tempo, como criar um webservice. Entretanto, é muito interessante que você tenha a base teórica necessária para aplicar o conhecimento aqui adquirido em ambientes mais complexos.
- Neste sentido, sugere-se:
  - O aprofundamento sobre o protocolo HTTP
  - Entendimento completo do padrão REST
  - Aplicar os outros verbos HTTP no Flask
  - Implantar o webservice em um servidor como Heroku
- Por fim, na próxima *cheat sheet*, vou apresentar um exemplo de webservice com as principais operações e, inclusive, salvando no banco de dados.
- Bons estudos!