**Creating a Endpoint with Flask**

Now that our application is structured, we can start coding some relevant endpoints. As mentioned before, the goal of our application is to help users to manage incomes and expenses. We will begin by defining two endpoints to handle incomes. Let's replace the contents of the file with the following:

from flask import Flask, jsonify, request

app = Flask(\_\_name\_\_)

incomes = [

{ 'description': 'salary', 'amount': 5000 }

]

@app.route('/incomes')

def get\_incomes():

return jsonify(incomes)

@app.route('/incomes', methods=['POST'])

def add\_income():

incomes.append(request.get\_json())

return '', 204

Since improving our application, we have removed the endpoint that returned "Hello, world!" to users. In its place, we defined an endpoint to handle HTTP GET requests to return incomes and another endpoint to handle HTTP POST requests to add new ones. These endpoints are annotated with @app.route to define routes listening to requests on the /incomes endpoint. [Flask provides great documentation on what exactly this does](http://flask.pocoo.org/docs/0.12/api/#flask.Flask.route).

To facilitate the process, we currently manipulate incomes as [dictionaries](https://docs.python.org/3/tutorial/datastructures.html#dictionaries). However, we will soon create classes to represent incomes and expenses.

To interact with both endpoints that we have created, we can start our application and issue some HTTP requests:

# start the cashman application

./bootstrap.sh &

# get incomes

curl http://localhost:5000/incomes

# add new income

curl -X POST -H "Content-Type: application/json" -d '{

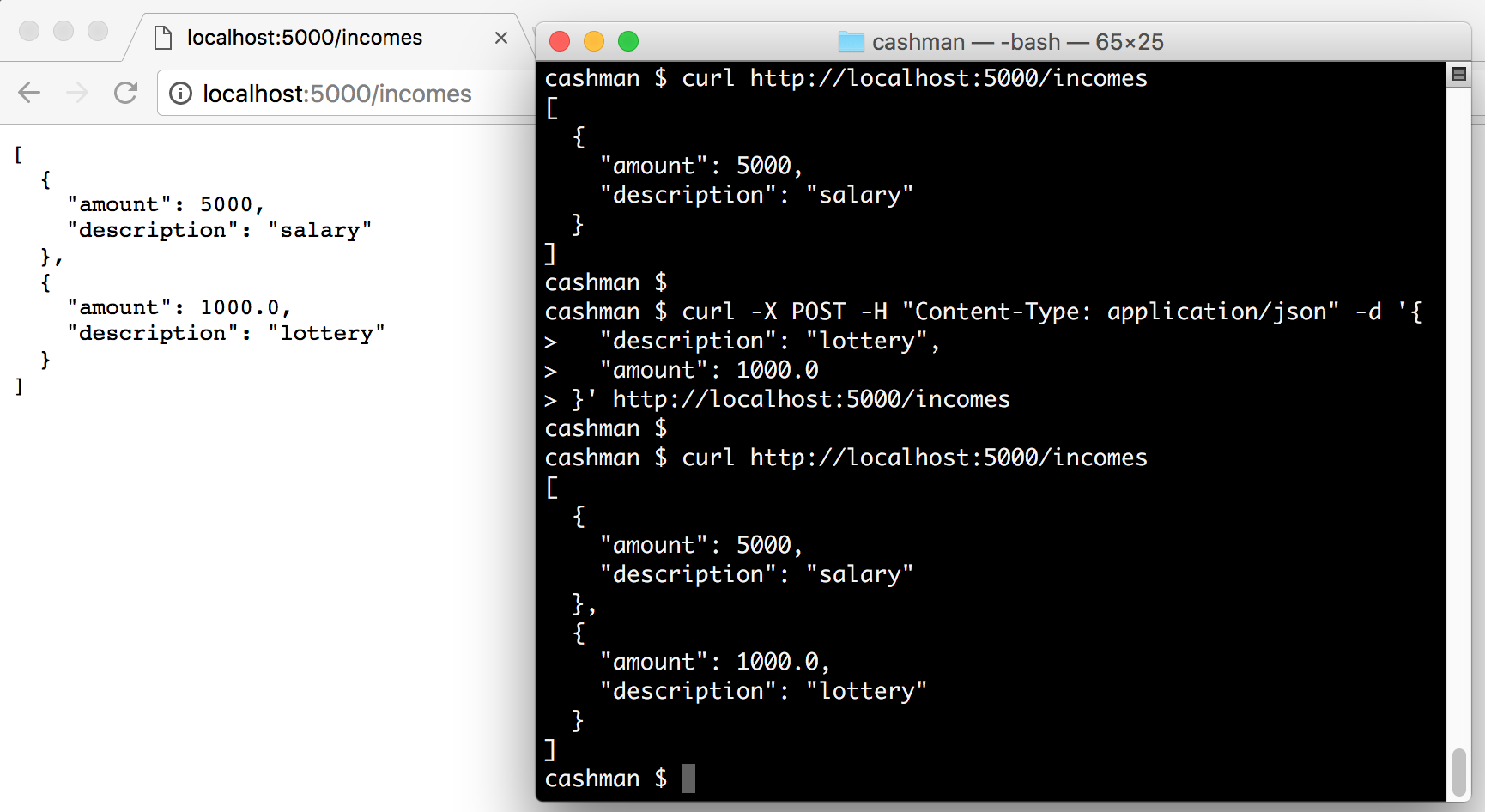
"description": "lottery",

"amount": 1000.0

}' http://localhost:5000/incomes

# check if lottery was added

curl localhost:5000/incomes



**Mapping Models with Python Classes**

Using dictionaries in a simple use case like the one above is enough. However, for more complex applications that deal with different entities and have multiple business rules and validations, we might need to encapsulate our data into [Python classes](https://docs.python.org/3/tutorial/classes.html).

We will refactor our application to learn the process of mapping entities (like incomes) as classes. The first thing that we will do is create a submodule to hold all our entities. Let's create a model directory inside the cashman module and add an empty file called \_\_init\_\_.py on it.

# create model directory inside the cashman module

mkdir -p cashman/model

# initialize it as a module

touch cashman/model/\_\_init\_\_.py

Mapping a Python superclass

We will create three classes in this new module/directory: Transaction, Income, and Expense. The first class will be the base for the two others, and we will call it Transaction. Let's create a file called transaction.py in the model directory with the following code:

import datetime as dt

from marshmallow import Schema, fields

class Transaction(object):

def \_\_init\_\_(self, description, amount, type):

self.description = description

self.amount = amount

self.created\_at = dt.datetime.now()

self.type = type

def \_\_repr\_\_(self):

return '<Transaction(name={self.description!r})>'.format(self=self)

class TransactionSchema(Schema):

description = fields.Str()

amount = fields.Number()

created\_at = fields.Date()

type = fields.Str()

Besides the Transaction class, we also defined a TransactionSchema. We will use the latter to deserialize and serialize instances of Transaction from and to JSON objects. This class inherits from another superclass called Schema that belongs on a package not yet installed.

# installing marshmallow as a project dependency

pipenv install marshmallow

[Marshmallow is a popular Python package](https://marshmallow.readthedocs.io/en/latest/) for converting complex datatypes, such as objects, to and from built-in Python datatypes. We can use this package to validate, serialize, and deserialize data. We won't dive into validation in this article, as it will be the subject of another one. Though, as mentioned, we will use marshmallow to serialize and deserialize entities through our endpoints.