Table Of Contents

1. Model-View-Controller
2. Flask
3. First steps
4. Routing
5. Templates

- ✓ Basic template rendering
- ✓ Using resources
- ✓ Template inheritance
- ✓ Template macros
- ✓ Template language

6. Requests

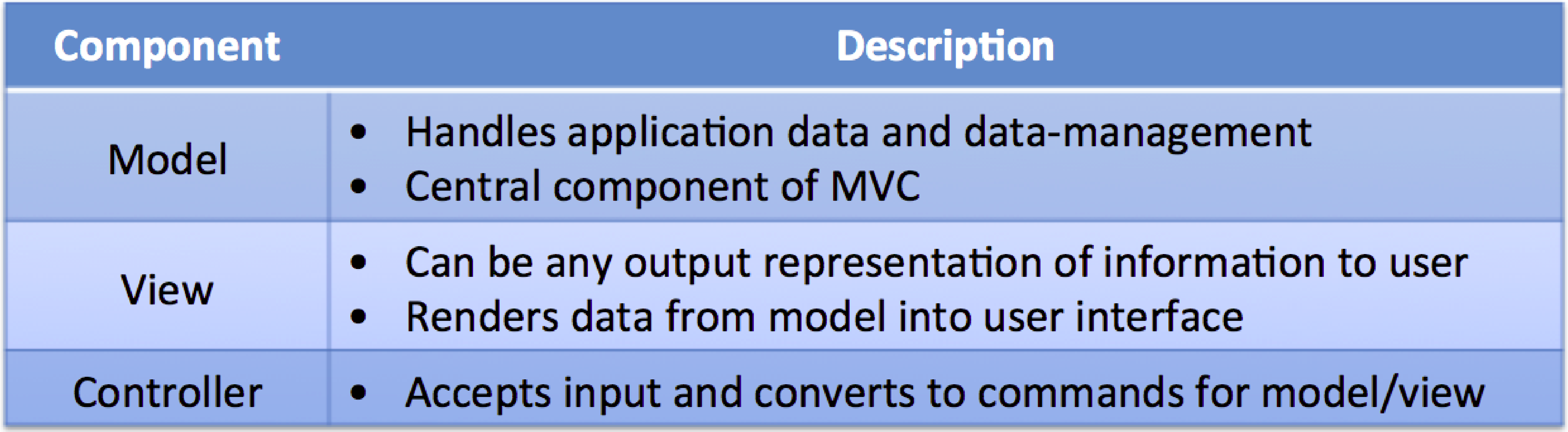
MVC

The pattern MVC, which stands for Model-View-Controller, separates an application in three components:

Model: Business logic of the application

View: Application's user interface and representation of the data that have been retrieved by the Model.

Controller: Orchestrator of all the interactions among the other components and the users



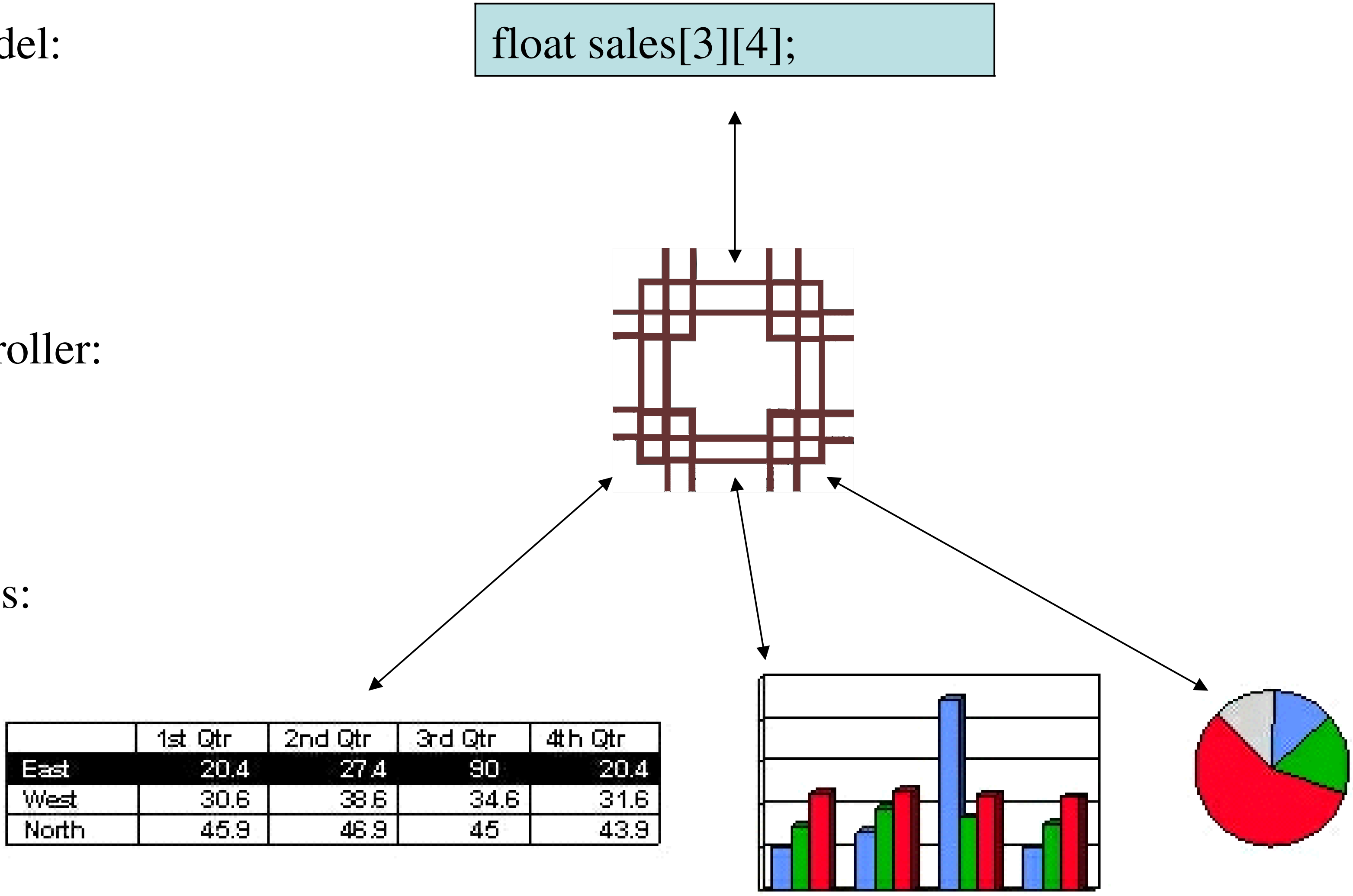
Component	Description
Model	• Handles application data and data-management • Central component of MVC
View	• Can be any output representation of information to user • Renders data from model into user interface
Controller	• Accepts input and converts to commands for model/view

Model View Controller

model:

controller:

views:



float sales[3][4];

	1st Qtr	2nd Qtr	3rd Qtr	4th Qtr
East	20.4	27.4	90	20.4
West	30.6	38.6	34.6	31.6
North	45.9	46.9	45	43.9

Model View Controller

- MVC's the paradigm for arranging your code,into functional segments so your brain does not explode.

  To achieve re-usability you gotta keep those boundaries clean,Model on the one side, View on the other, the Controller's in between.Most of the modern web development frameworks follow the Model-View-Controller (MVC model):

- ✓ The model: representation of data. Usually, have a strong relation with the database
- ✓ The views: what is shown to the user. Can be any kind

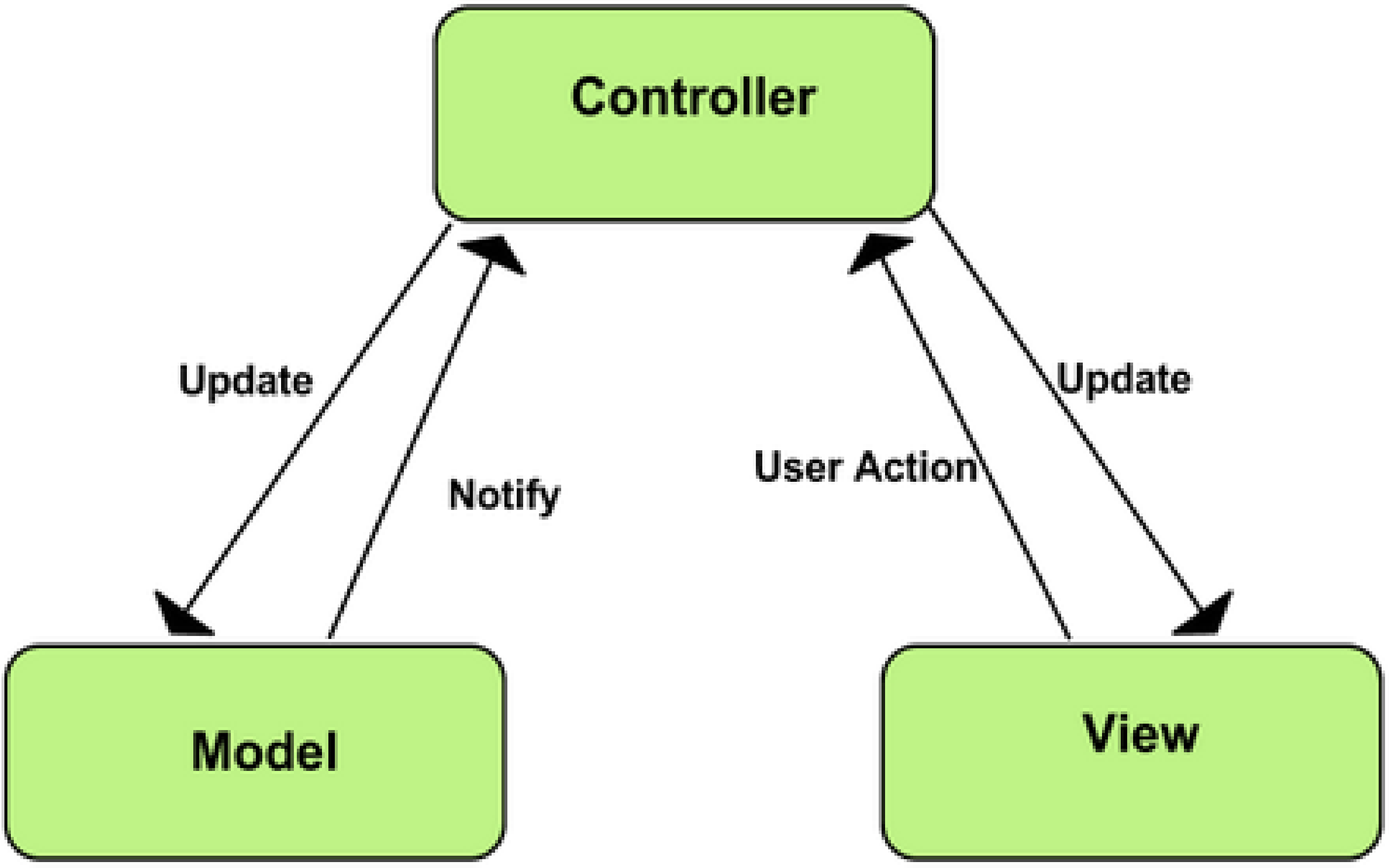
  of user interface, usually HTML pages with Javascript.
- ✓ The controls: what operation are done on the data.

  It’s a rather convenient way to design software projects

  involving user interfaces presenting and manipulating data.

MVC

- A user requests to view a page by entering a URL.
- The Controller receives that request.
- It uses the Models to retrieve all of the necessary data, organizes it, and sends it off to the…
- View, which then uses that data to render the final webpage presented to the the user in their browser.



MVC

Example for Model-View-Controller:

An online management game

- The rule of the game, updating the state of each player⇒ the model
- The HTML pages, showing the various screen of the game ⇒ the views
- The methods called when a user click on the screen ⇒ the controllers

MVC

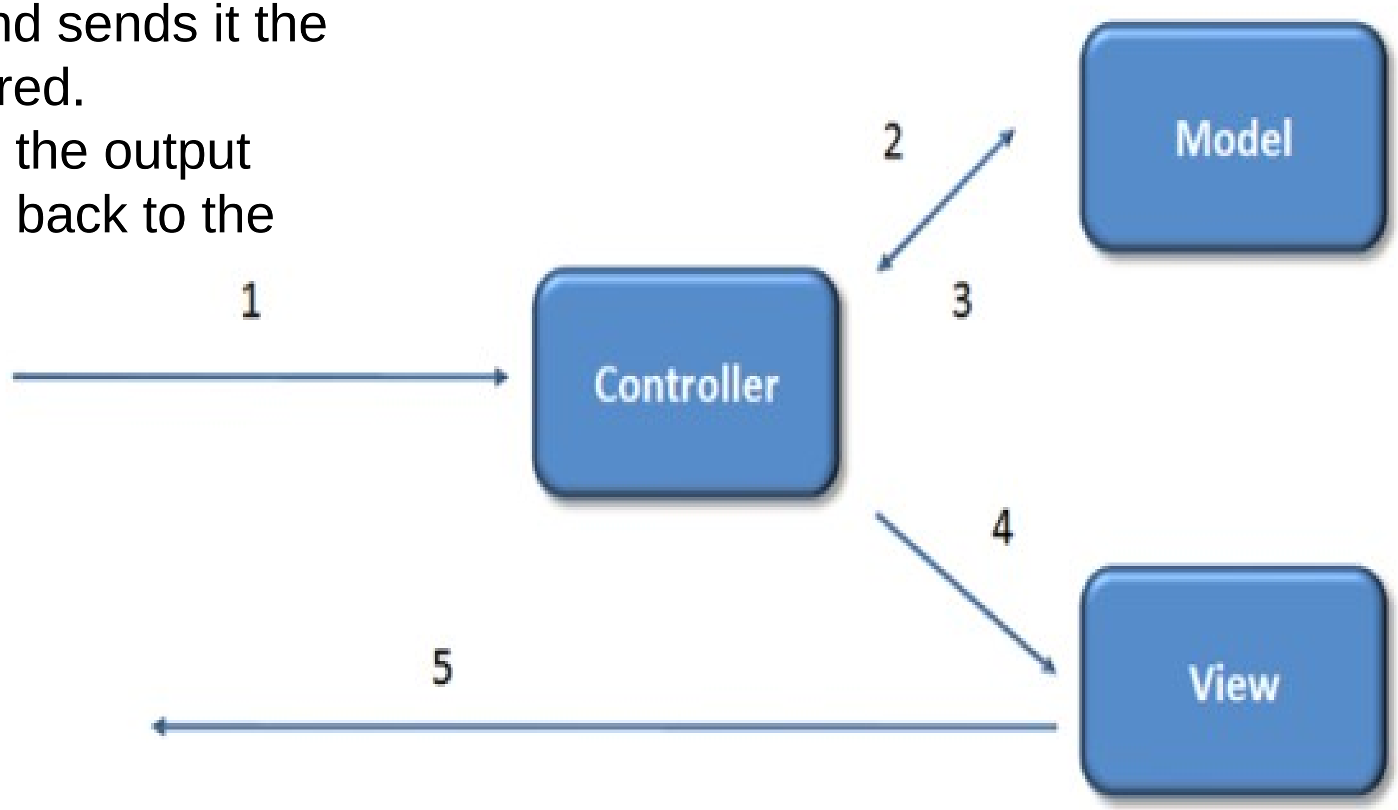
Example for Model-View-Controller:

An online shop

- The list of products, the payment rules, delivery orders ⇒ the model
- The HTML pages, showing the various screen of the shop ⇒ the views
- The methods for payment, order, shopping cart ⇒the controllers

Interaction

1. The request comes from the client and hits the Controller.
2. The Controller calls the Model in order to perform some "business" operations.
3. The Model returns the results of the operations back to the Controller.
4. The Controller decides which View needs to be rendered and sends it the data that must be rendered.
5. Finally the View renders the output and sends the response back to the client.



Why MVC?

- Allows a better separation of concerns, which results in code independence, better reusability and easier testing.
- Allow for a simple way of building complex web applications.
- Helps organize the work
  - some work on views : graphics designers
  - some work on model : database, software architects
  - some work on controls : low level or specialized code
- The Model should not be contaminated with control code or display code
- The View should represent the Model as it really is, not some remembered status
- The Controller should *talk to* the Model and View, not *manipulate* them
  - The Controller can set variables that the Model and View can read

Separation of concerns(how?)

- Model
  - Handles all the "business logic" of the application
  - Stores the application state
- View
  - Responsible for generating a view for the user of the data from the model
  - Usually a simple templating system to display the data from the model
- Controller
  - Responsible for taking input from the user, fetching the correct data from the model, then calling the correct view to display the data
  - Should be very simple

Modern Web Application Frameworks

- Many different frameworks for all languages (not a comprehensive list)
- Ruby
  - Ruby on Rails
- Python
  - Django
  - Flask
- PHP
  - CakePHP
- ASP
  - ASP.NET MVC
- Java
  - Spring MVC

Web application with script language

Why using a scripting language for a web application ?

- More adapted language to paste together various components (database, rendering, routing, . . .)
- Make its easier to release early & often
- Easier to maintain & modify
- Speed good enough for many use cases

Web application with script language

Why not PHP, or PHP framework ?

- Designed to make simple web pages, not large web applications
- Awfully designed programming language
- Very inconsistent libraries
- Very little help for debugging
- Many security issues
- Many better alternatives

Detailed explanation here

http://me.veekun.com/blog/2012/04/09/php-a-fractal-of-bad-design

Web application with script language

Why not using Java/JSP/JBoss/Apache/Hibernate/Spring ?

- Even simple changes requires lots of coding
- Big changes takes a lot of planning
- Edit/Compile/Run takes more resource
- General speed of development much reduced
- Working without a big fat IDE is tedious

But you can use those all this with a script-like language :

Grails And Groovy

Why Flask?

- It is small : quick to learn and master
- It is complete : you can use to do serious apps
- It is lean : a shell and a text editor are enough, no need for an IDE to be efficient with it
- It is very well documented

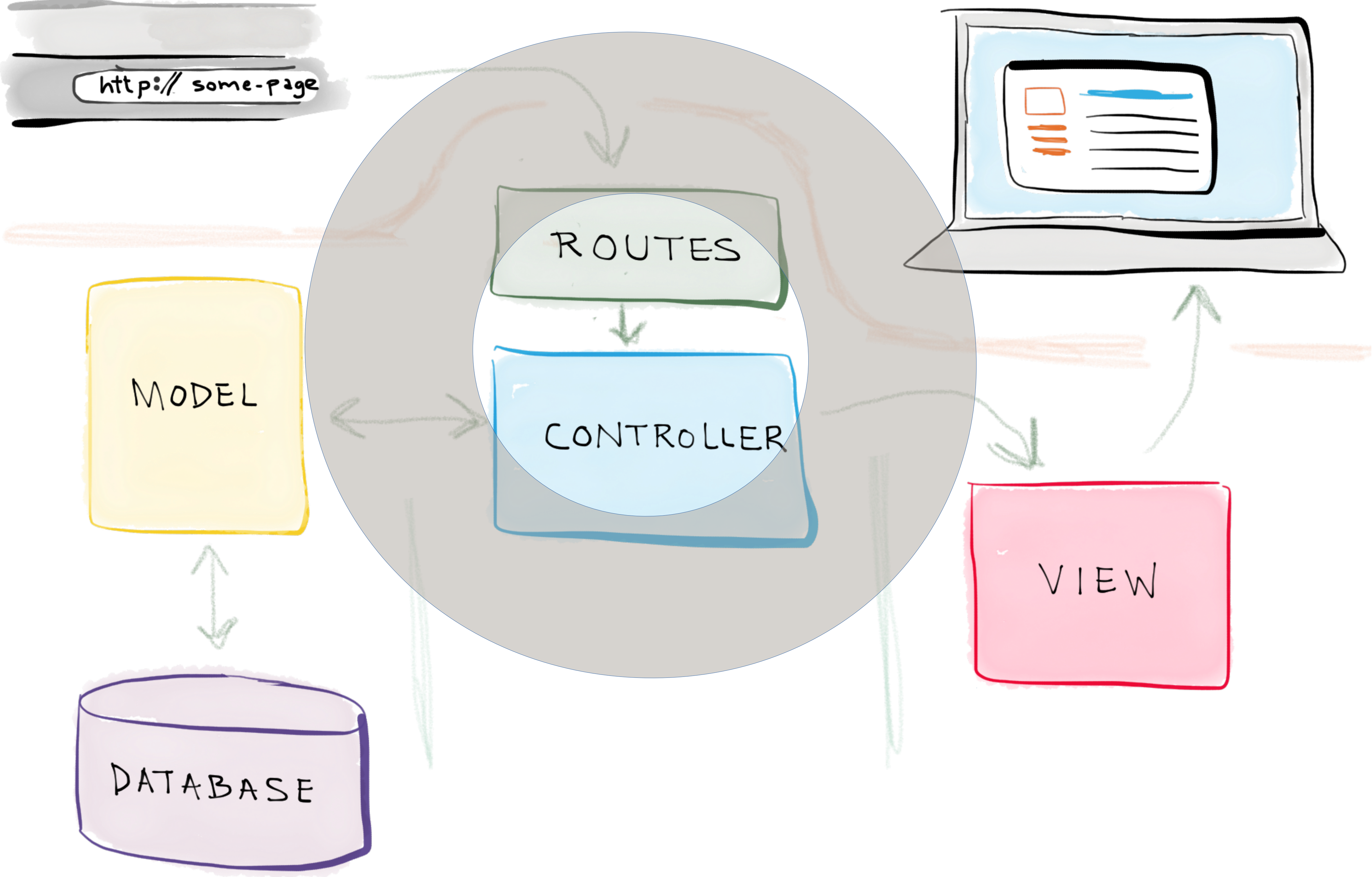
The same ideas can be found in most web development frameworks.

Why Flask?

Flask is a nice glue around existing tools

- Python⇒programming language
- SQL Alchemy⇒database
- Jinja2⇒HTML template system
- Werkzeug⇒WSCGI handling (CGI, but better)

MVC

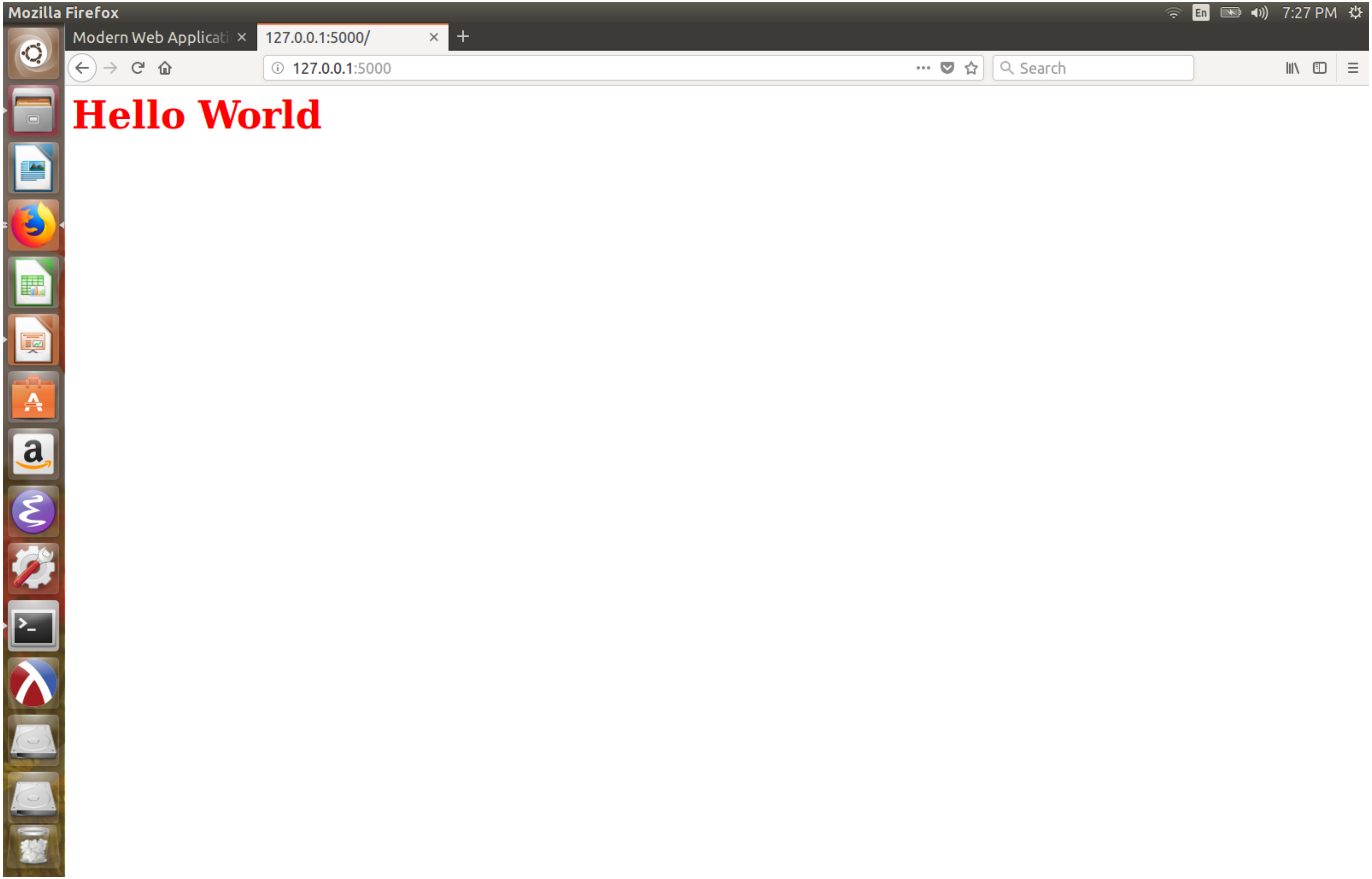


Hello World! A minimal Flask application

```
from flask import Flask

#create the application.
app = Flask(__name__)

@app.route('/')
def helloworld():
    """ Displays hello world string at '/'"""
    return "<h1 style=color:red>Hello World</h1>"


if __name__ == '__main__':
    app.debug = True
    app.run()
```

Run this, and open your web browser at
http://127.0.0.1:5000

Hello World! A minimal Flask application



Hello World

Hello World!A minimal Flask application

```
 from flask import Flask

 #create the application.
 app = Flask(__name__)

if __name__ == '__main__':
# if loaded as main module
    app.run()
```

This creates an application instance and run it.

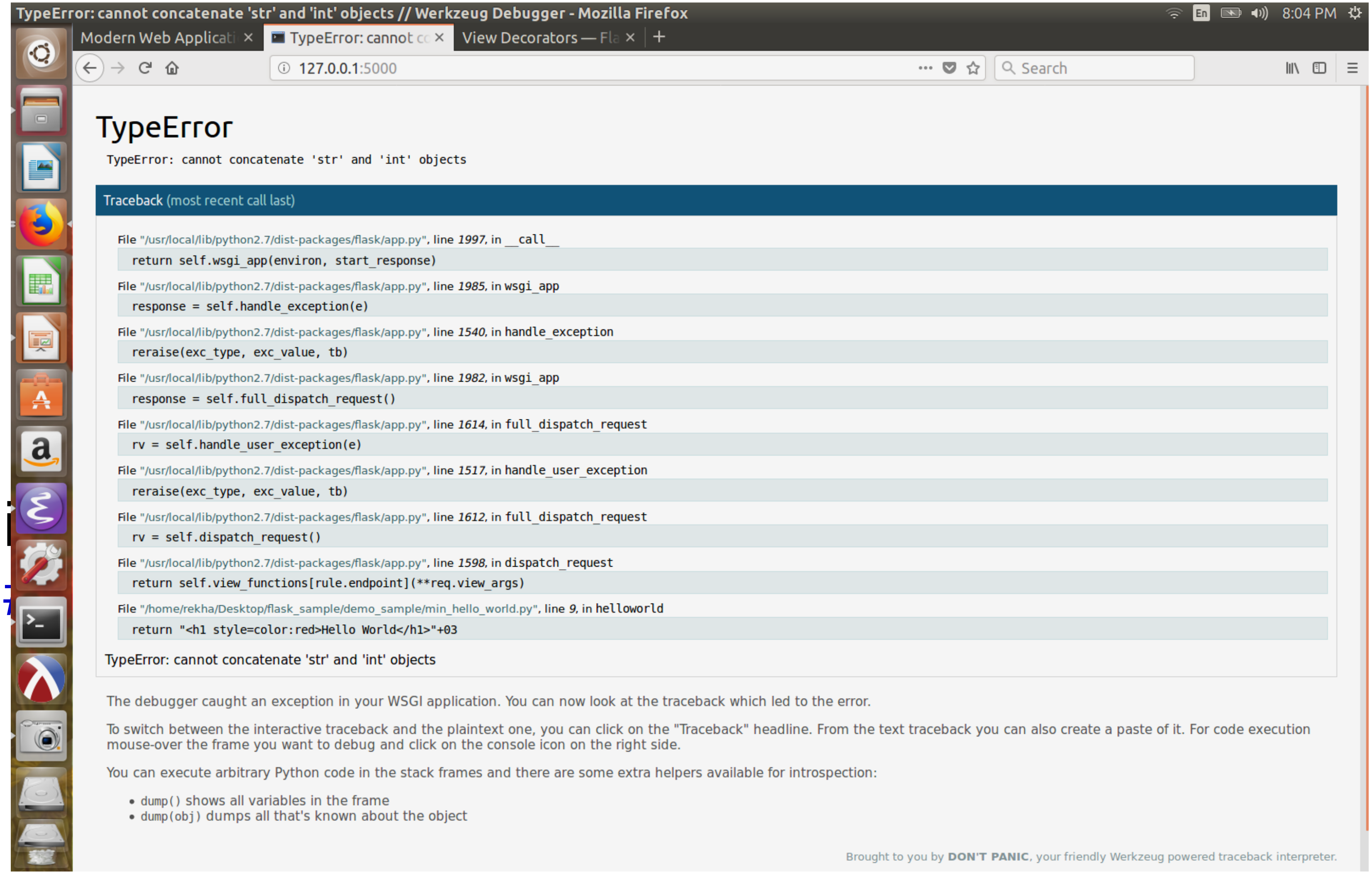
Hello World!

A minimal Flask application

```
from flask import Flask

#create the application.
app = Flask(__name__)

if __name__ == '__main__':
# if loaded as main module
    app.debug = True
    app.run()
```

In debug mode, you can edit the code while the server runs : it will auto-restart !

Hello World! A minimal Flask application



TypeError: cannot concatenate 'str' and 'int' objects // Werkzeug Debugger - Mozilla Firefox
8:04 PM
Modern Web Applicati
TypeError: cannot cc
View Decorators — Fla
127.0.0.1:5000
Search

TypeError

```
TypeError: cannot concatenate 'str' and 'int' objects
```

Traceback (most recent call last)

```
File "/usr/local/lib/python2.7/dist-packages/flask/app.py", line 1997, in __call__
  return self.wsgi_app(environ, start_response)
File "/usr/local/lib/python2.7/dist-packages/flask/app.py", line 1985, in wsgi_app
  response = self.handle_exception(e)
File "/usr/local/lib/python2.7/dist-packages/flask/app.py", line 1540, in handle_exception
  reraise(exc_type, exc_value, tb)
File "/usr/local/lib/python2.7/dist-packages/flask/app.py", line 1982, in wsgi_app
  response = self.full_dispatch_request()
File "/usr/local/lib/python2.7/dist-packages/flask/app.py", line 1614, in full_dispatch_request
  rv = self.handle_user_exception(e)
File "/usr/local/lib/python2.7/dist-packages/flask/app.py", line 1517, in handle_user_exception
  reraise(exc_type, exc_value, tb)
File "/usr/local/lib/python2.7/dist-packages/flask/app.py", line 1612, in full_dispatch_request
  rv = self.dispatch_request()
File "/usr/local/lib/python2.7/dist-packages/flask/app.py", line 1598, in dispatch_request
  return self.view_functions[rule.endpoint](**req.view_args)
File "/home/rekha/Desktop/flask_sample/demo_sample/min_hello_world.py", line 9, in helloworld
  return "<h1 style=color:red>Hello World</h1>"+03
TypeError: cannot concatenate 'str' and 'int' objects
```

The debugger caught an exception in your WSGI application. You can now look at the traceback which led to the error.

To switch between the interactive traceback and the plaintext one, you can click on the "Traceback" headline. From the text traceback you can also create a paste of it. For code execution mouse-over the frame you want to debug and click on the console icon on the right side.

You can execute arbitrary Python code in the stack frames and there are some extra helpers available for introspection:

- dump() shows all variables in the frame
- dump(obj) dumps all that's known about the object

Brought to you by **DON'T PANIC**, your friendly Werkzeug powered traceback interpreter.

The debug mode will also helps a lot to point where the problem is.

Hello World! A minimal Flask application

```
@app.route('/')
def helloworld():
    """ Displays hello world string at '/'"""
    return "<h1 style=color:red>Hello World</h1>"
```

- helloworld() will be called every time the address / is requested
- helloworld()returns the text data for the web browser

Route

@app.route('/')

Routes are, essentially, URL patterns associated with different pages.

URL --->

matching predefined route --->

associated controller action

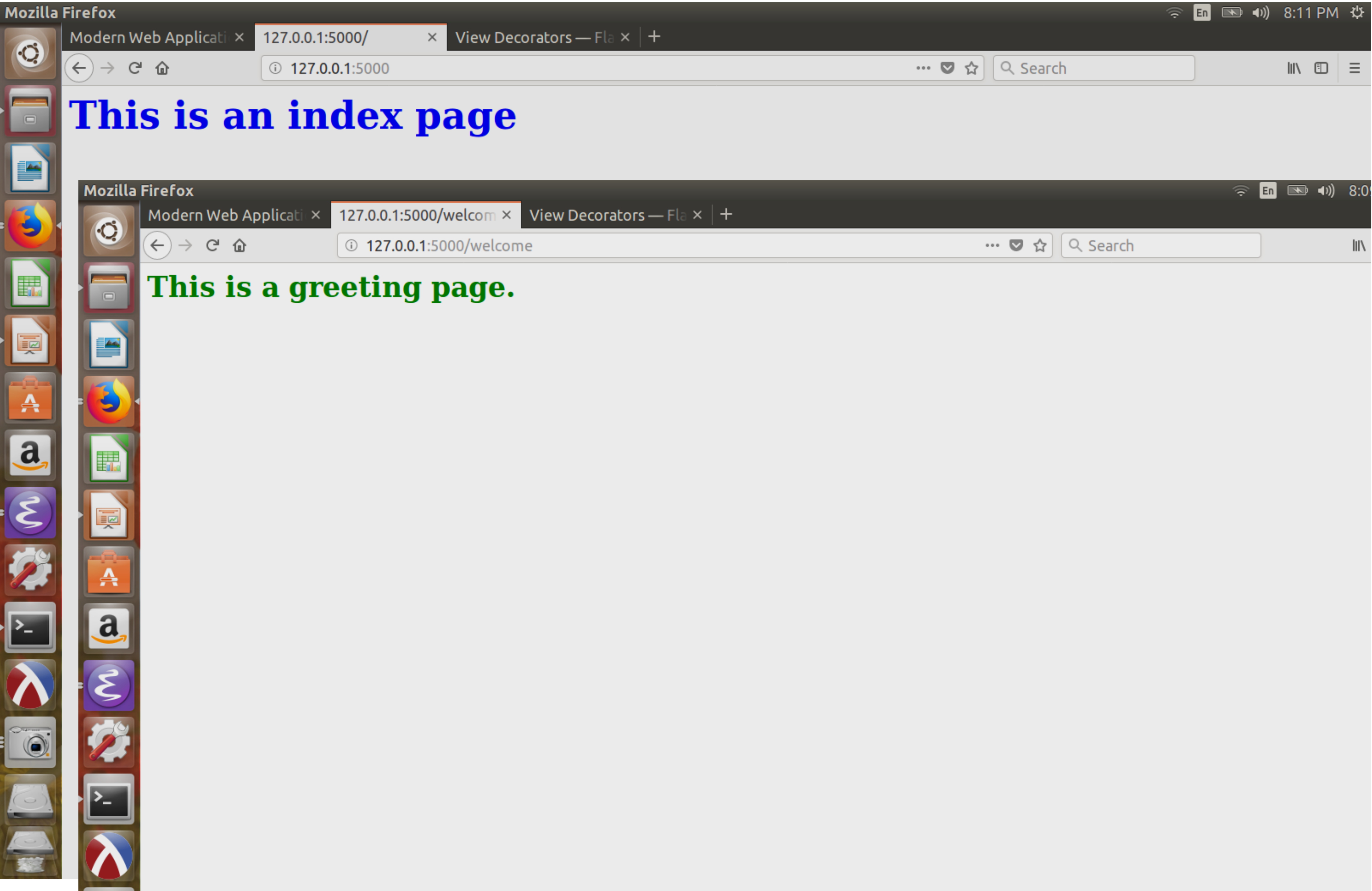
URL to function mapping

When an URL is requested, Flask will look for its corresponding function.

```
from flask import Flask

#create the application.
app = Flask(__name__)

@app.route('/')
def index():
    return "<h1 style=color:blue>This is an index page</h1>"

@app.route('/welcome')
def welcome():
    return "<h2 style=color:green>This is a greeting page.</h2>"

if __name__ == '__main__':
    app.debug = True
    app.run()
```

When function returns text data, It can be HTM, XML, JSON,etc.

URL to function mapping



This is an index page

This is a greeting page.

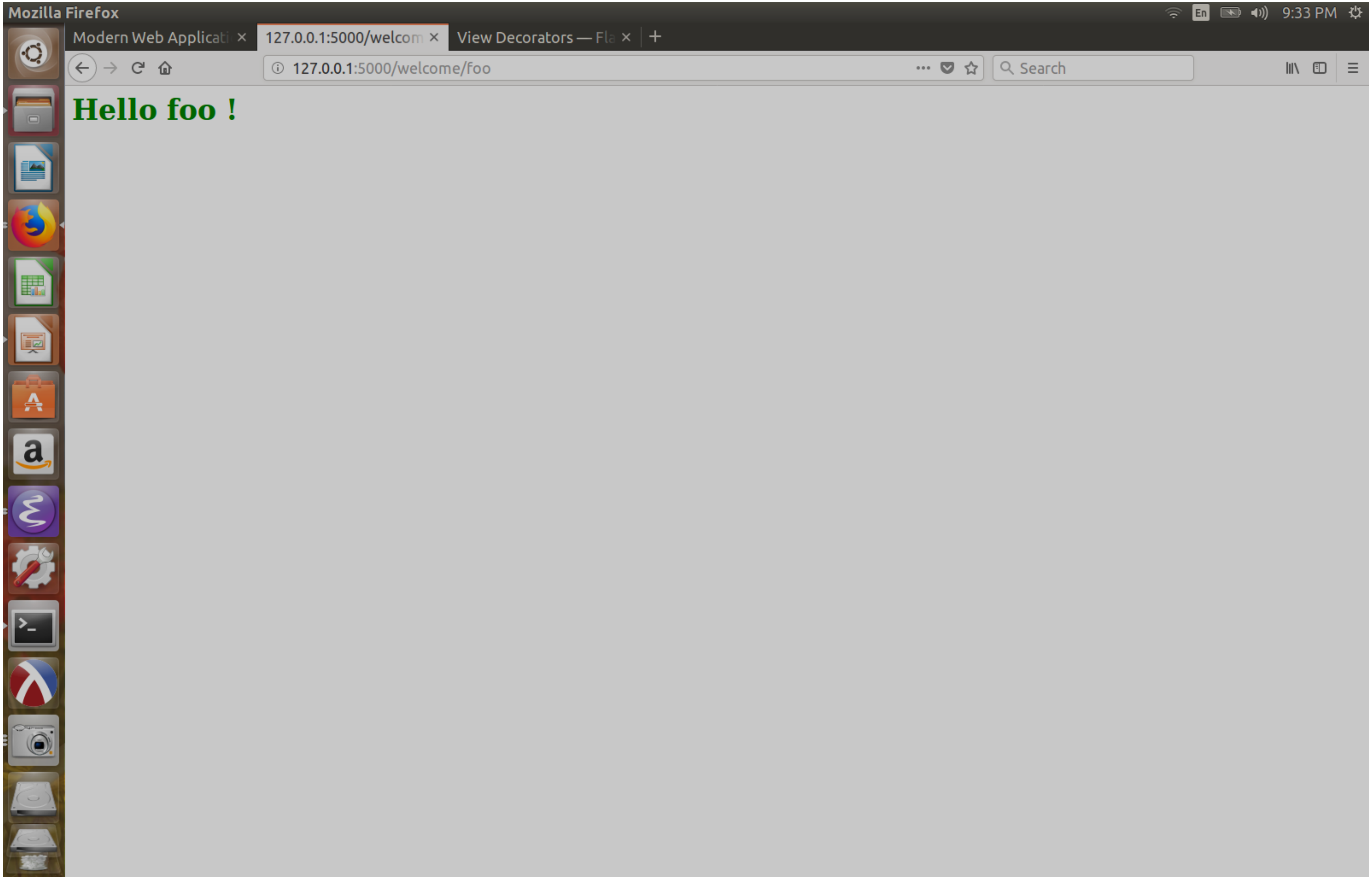
URL with arguments

You can defines URL with parameters

```
from flask import Flask

#create the application.
app = Flask(__name__)

@app.route('/')
def index():
    return "<h1 style=color:blue>This is an index page</h1>"

@app.route('/welcome/<name>')
def welcome(name):
    return "<h2 style=color:green>Hello %s !</h2>" % name

if __name__ == '__main__':
    app.debug = True
    app.run()
```

It gives a nice way, intuitive way to define ressources on a website.

URL with arguments



Hello foo !

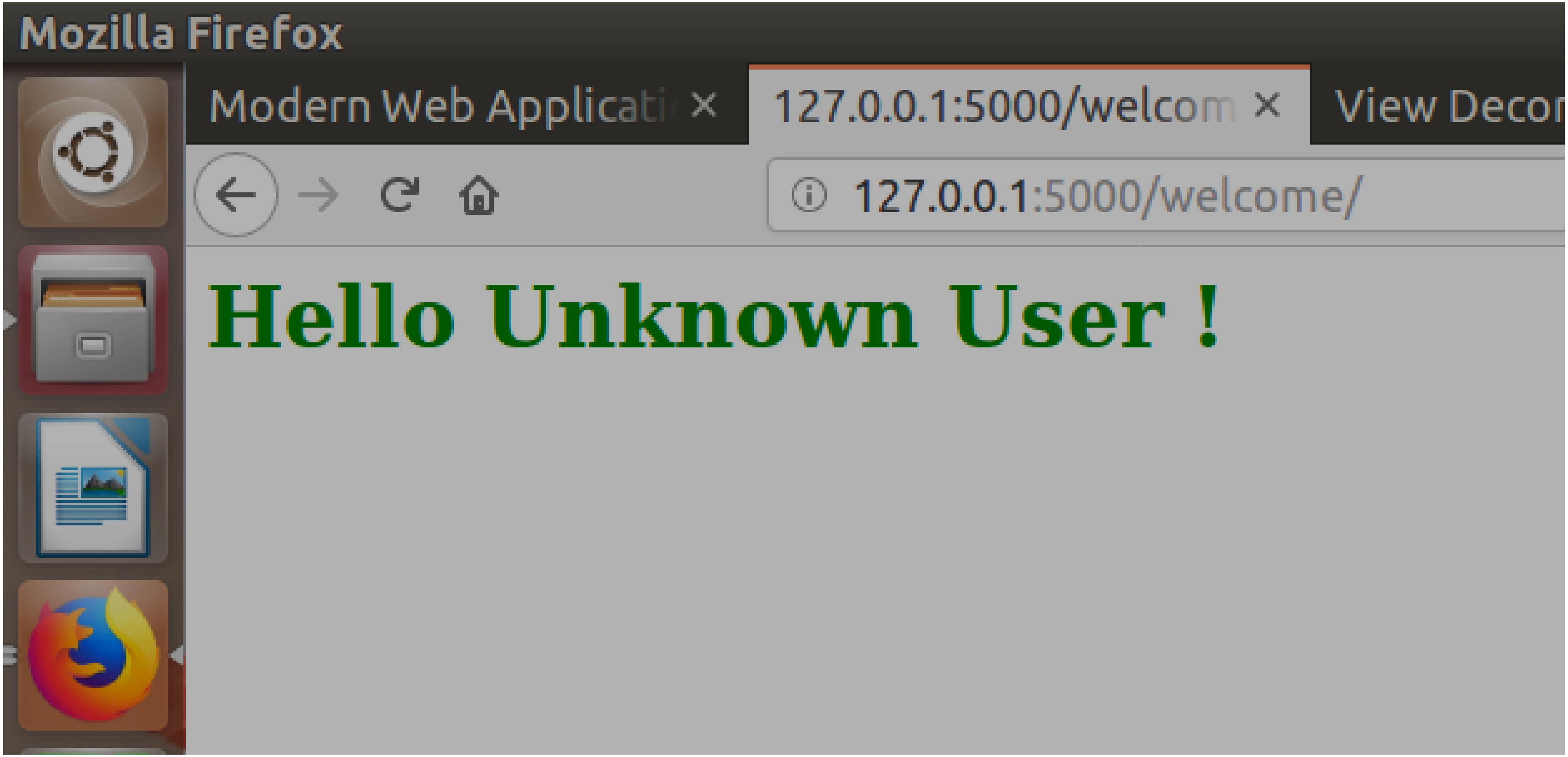
It gives a nice way, intuitive way to define ressources on a website.

URL with optional params

You can make param to url optional.

```
@app.route('/welcome/')
@app.route('/welcome/<name>')
def welcome(name = None):
    if name is None:
        return "<h2 style=color:green>Hello %s !</h2>" % "Unknown User"
    else:
        return "<h2 style=color:green>Hello %s !</h2>" % name
```

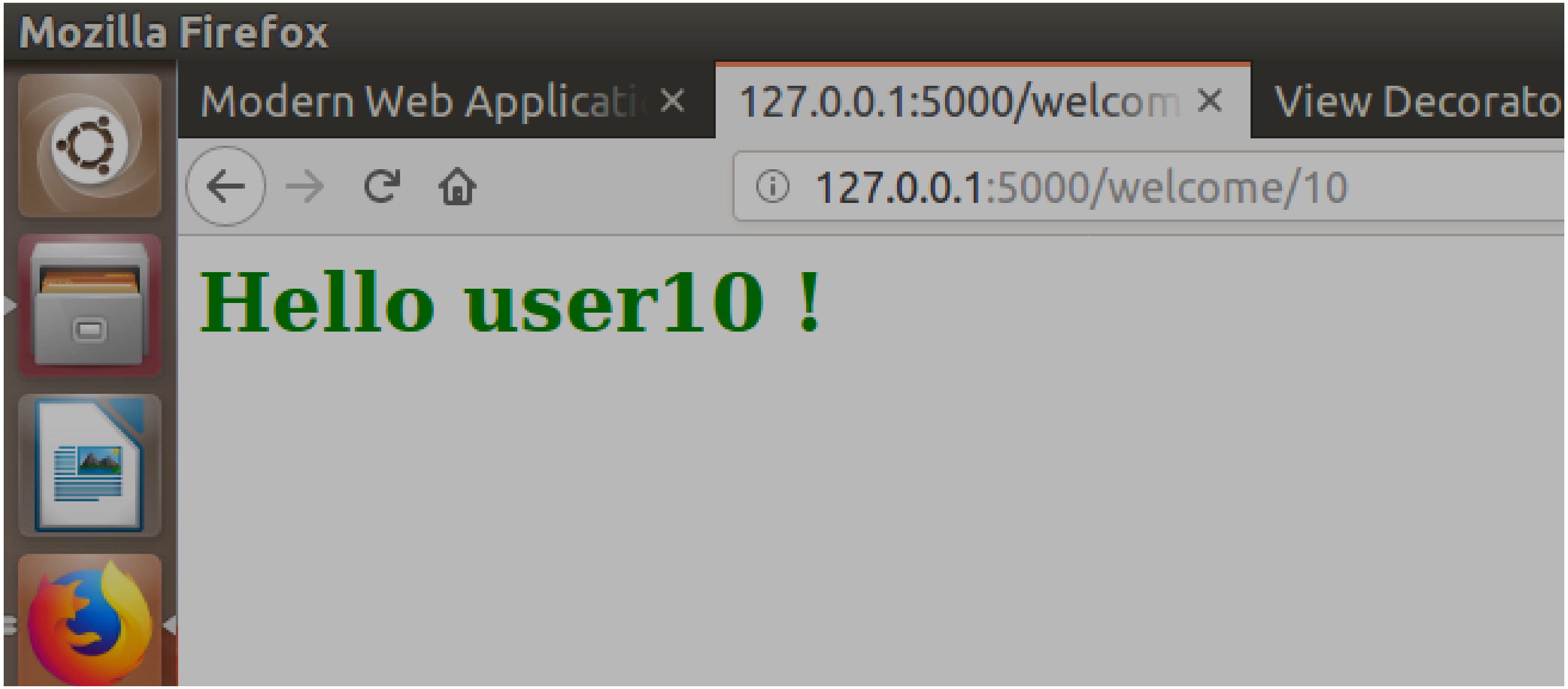
URL with optional params



Hello Unknown User !

URL with typed param

```
@app.route('/welcome/')
@app.route('/welcome/<int:id>')
def welcome(id = None):
    if id is None:
        return "<h2 style=color:green>Hello %s !</h2>" % "Unknown User"
    else:
        return "<h2 style=color:green>Hello user%d !</h2>" % id
```

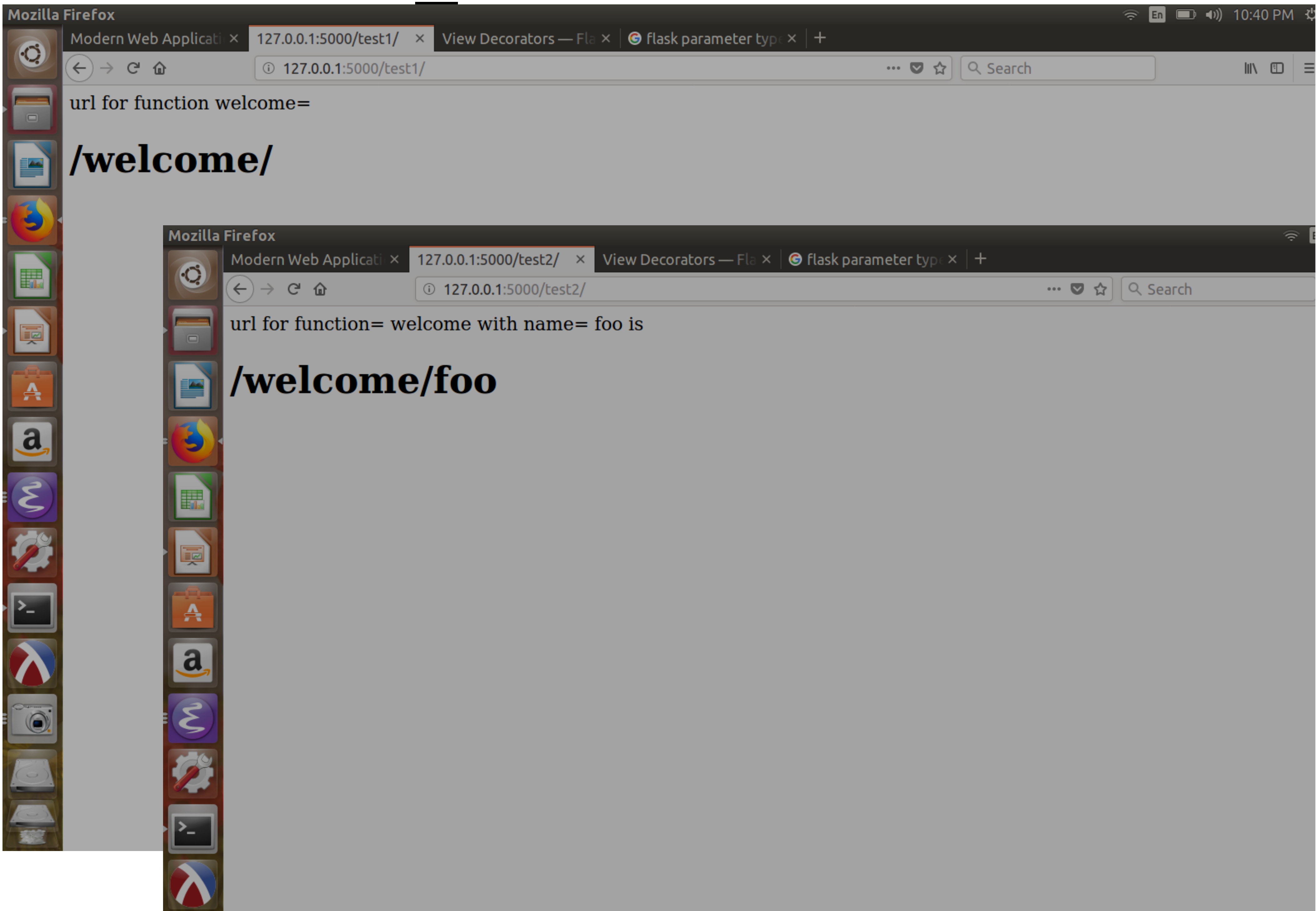


Hello user10 !

Url _for also works for URL with parameters

```
@app.route('/')
def index():
    return "<h1 style=color:blue>This is an index page</h1>"

@app.route('/welcome/')
@app.route('/welcome/<name>')
def welcome(name = None):
    if name is None:
        return "<h2 style=color:green>Hello %s !</h2>" % "Unknown User"
    else:
        return "<h2 style=color:green>Hello %s !</h2>" % name

@app.route('/test1/')
def test1():
    fname = 'welcome'
    return "url for function %s=<h1/>%s" % (fname, url_for(fname))

@app.route('/test2/')
def test2():
    fname, name = 'welcome', 'foo'
    return "url for function= %s with name= %s is <h1/> %s" %
(fname,name,url_for(fname, name = name))
```

Url _for also works for URL with parameters



url for function welcome=

/welcome/

url for function= welcome with name= foo is

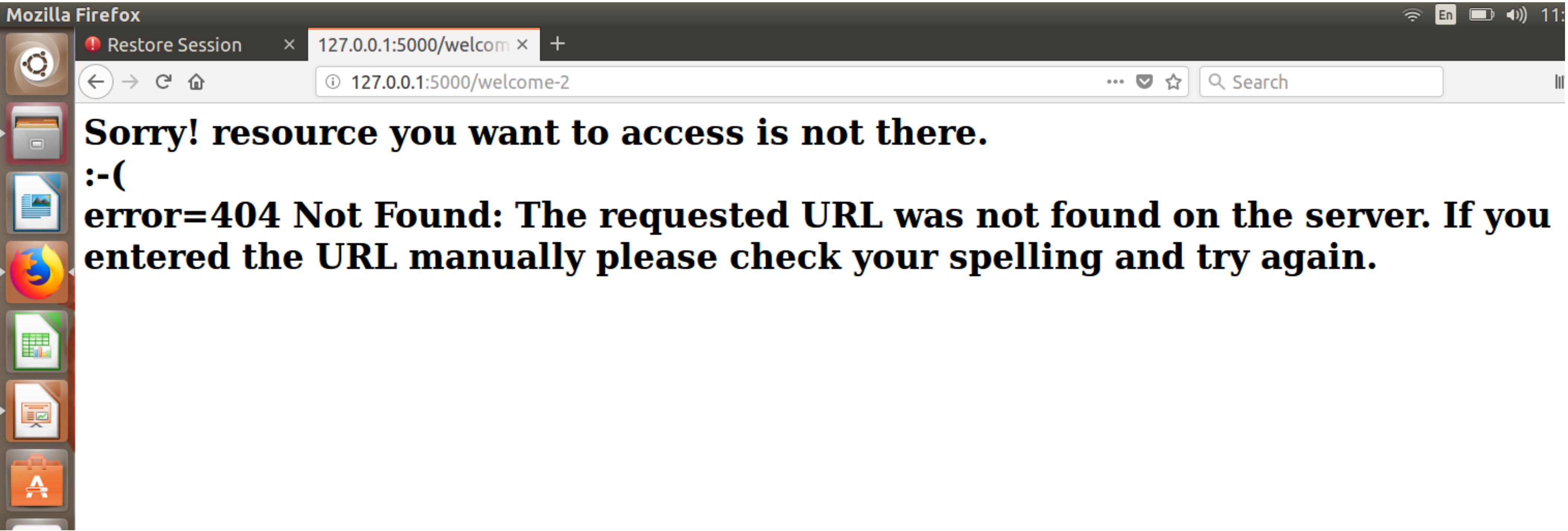
/welcome/foo

Redirect URL

```
@app.route('/test1/')
def test1():
    fname = 'welcome'
    return redirect(url_for(fname))

@app.route('/test2/')
def test2():
    fname, name = 'welcome', 'foo'
    return redirect (url_for(fname, name = name))
```

Error Handler

```
@app.errorhandler(404)
def page_error(error):
    return '<h2/>Sorry! resource you want to access is not there.<br>:-(<br> error=%s'%error
```



```
Sorry! resource you want to access is not there.
:-(
error=404 Not Found: The requested URL was not found on the server. If you
entered the URL manually please check your spelling and try again.
```

Error Handler

```
@app.route('/show_account/')
def show_account():
    logged_in = False
    if not logged_in:
        abort(401)
    return "balance is ..."
```

HTTP protocol defines several status codes.

400 BadRequest

401 Unauthorized

402 Payment Required

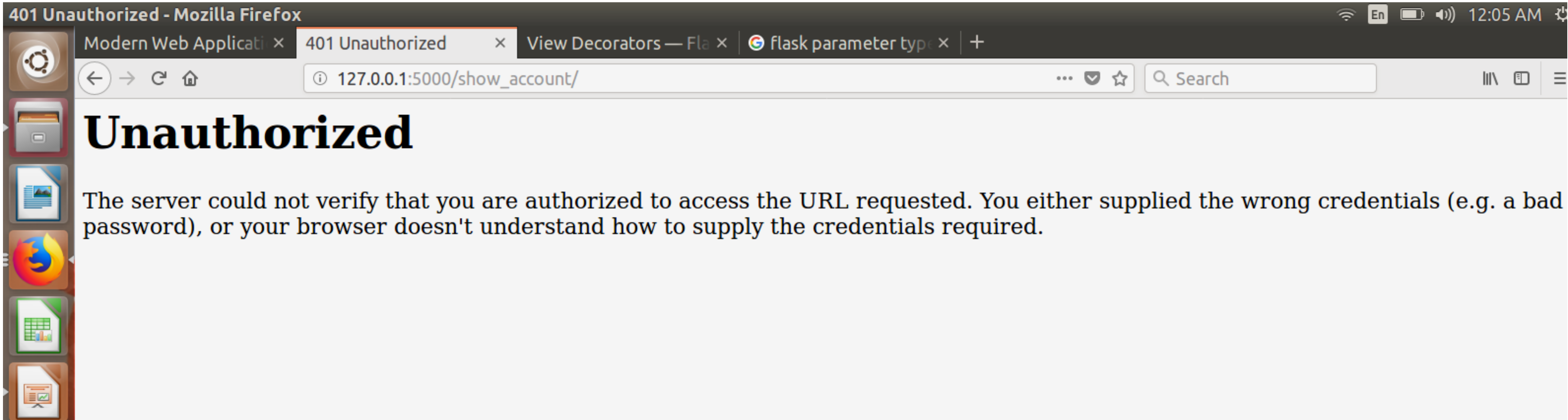
403 Forbidden

404 Not Found

500 Internal Server Error

501 Not Implemented

503 Service Unavailable



Unauthorized

The server could not verify that you are authorized to access the URL requested. You either supplied the wrong credentials (e.g. a bad password), or your browser doesn't understand how to supply the credentials required.