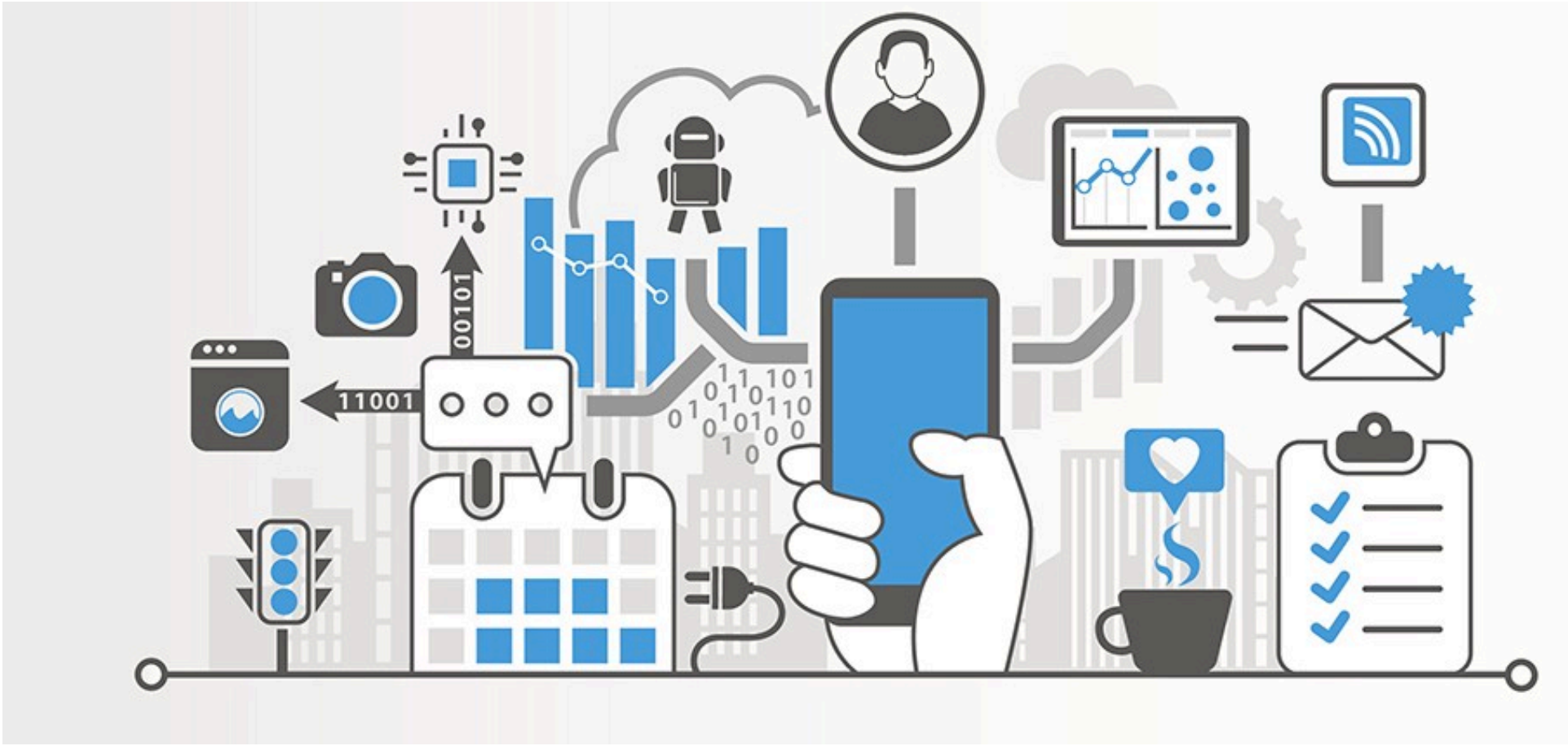
Programming for IoT Applications

Edoardo Patti

Lecture 6



Web Services introduction

CHERRYPY - A MINIMALIST PYTHON WEB FRAMEWORK

What did we study during the last lecture?

Webserver

- A webserver exposes services to clients
- A webserver can be reached using host:port
- A port is a communication endpoint identifing a specific process, application or a type of service running on server. It is 16-bit unsigned numbers
  - http://www.mywebsite.com:8080
  - http://192.168.1.34:8080

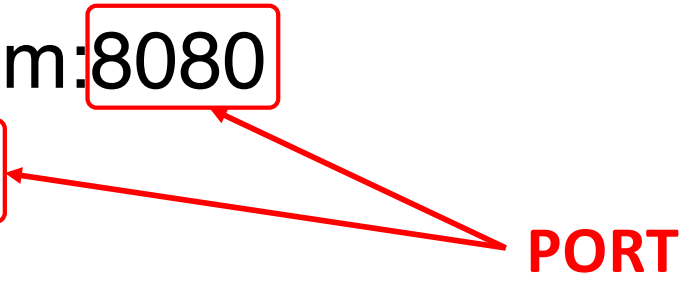
Webserver

- A webserver exposes services to clients
- A webserver can be reached using host:port
- A port is a communication endpoint identifing a specific process, application or a type of service running on server. It is 16-bit unsigned numbers
  - http://www.mywebsite.com:8080
  - http://192.168.1.34:8080

HOST

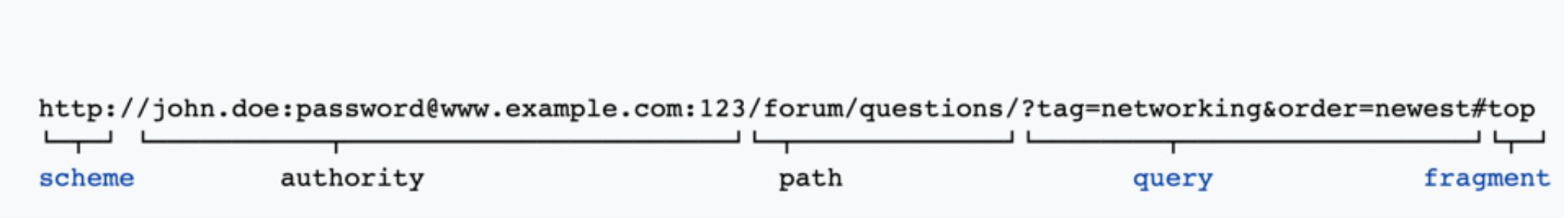
Webserver

- A webserver exposes services to clients
- A webserver can be reached using host:port
- A port is a communication endpoint identifing a specific process, application or a type of service running on server. It is 16-bit unsigned numbers
  - http://www.mywebsite.com:8080
  - http://192.168.1.34:8080



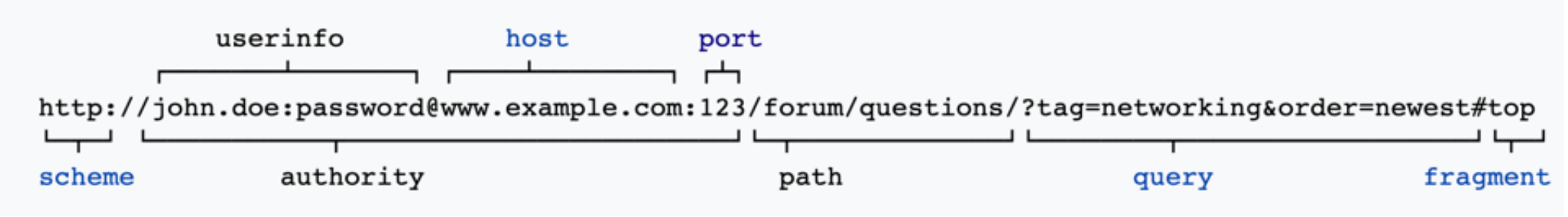
Webserver

- A webserver exposes services to clients
- A webserver can be reached using host:port
- A port is a communication endpoint identifing a specific process, application or a type of service running on server. It is 16-bit unsigned numbers



Webserver

- A webserver exposes services to clients
- A webserver can be reached using host:port
- A port is a communication endpoint identifing a specific process, application or a type of service running on server. It is 16-bit unsigned numbers



Webserver

A webserver must always be up and running ready to provide information

Webclient

- A webclient consumes services exposed by webserver
- A webclient starts the communication by indicating host:port

What is a Web Service?

The W3C (World Wide Web Consortium) defines a web service as *a software system designed to support interoperable machine-to-machine interaction over a network.*

What is a Web Service?

A web service is a **service** offered by an electronic device to another electronic device **communicating with each other via the World Wide Web**.

Web technology such as **HTTP is used for machine-to-machine communication**, transferring machine-readable file formats such as JSON and XML.


Let’s start coding…

Why CherryPy?

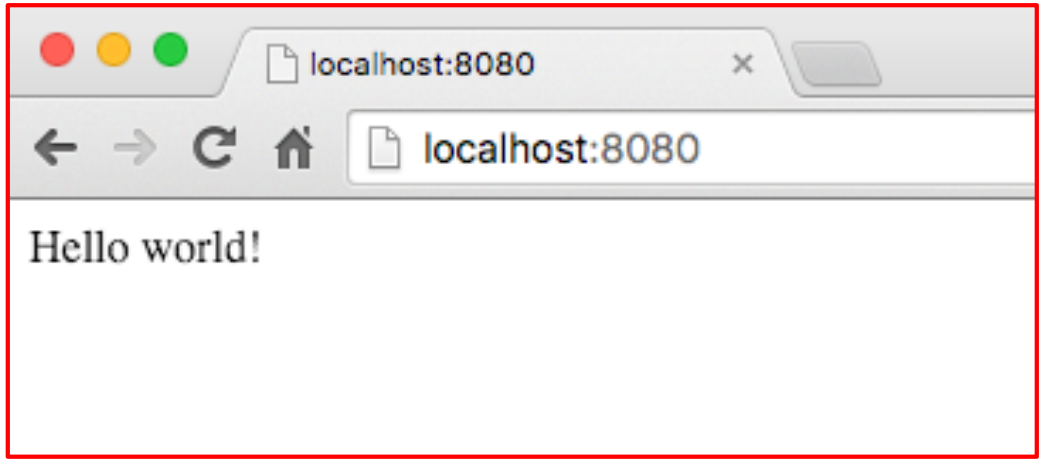
CherryPy is a pythonic, object-oriented web framework.

CherryPy allows developers to build web applications in much the same way they would build any other object-oriented Python program. This results in smaller source code developed in less time.

First web application

Example for starting a server and hosts an application that will be served at request reaching **http://127.0.0.1:8080/**

```
import cherrypy

class HelloWorld(object):
      @cherrypy.expose
      def index(self):
            return "Hello world!"

if __name__ == '__main__':
      cherrypy.tree.mount (HelloWorld())
      cherrypy.engine.start()
      cherrypy.engine.block()
```

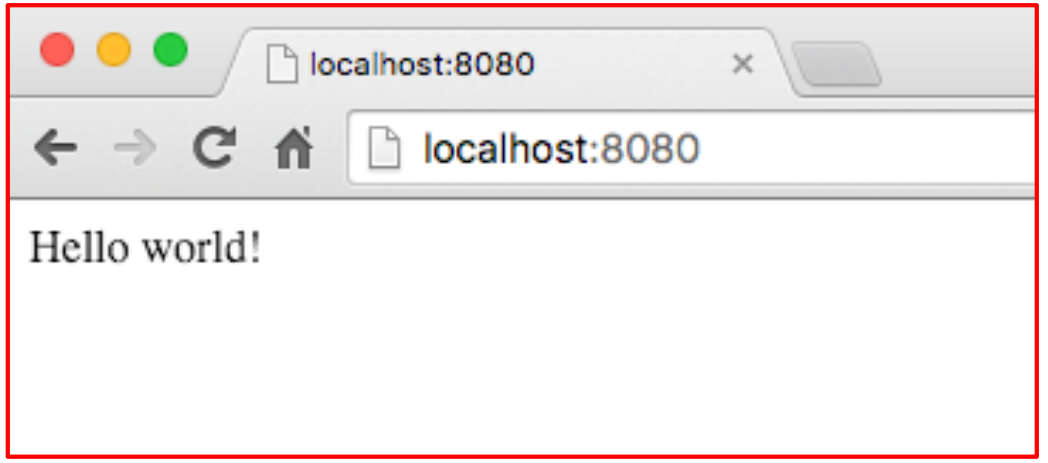


First web application

Example for starting a server and hosts an application that will be served at request reaching **http://127.0.0.1:8080/**

```
import cherrypy

class HelloWorld(object):
    @cherrypy.expose
    def index(self):
        return "Hello world!"

if __name__ == '__main__':
    cherrypy.tree.mount (HelloWorld())
    cherrypy.engine.start()
    cherrypy.engine.block()
```

Expose the method as an URL



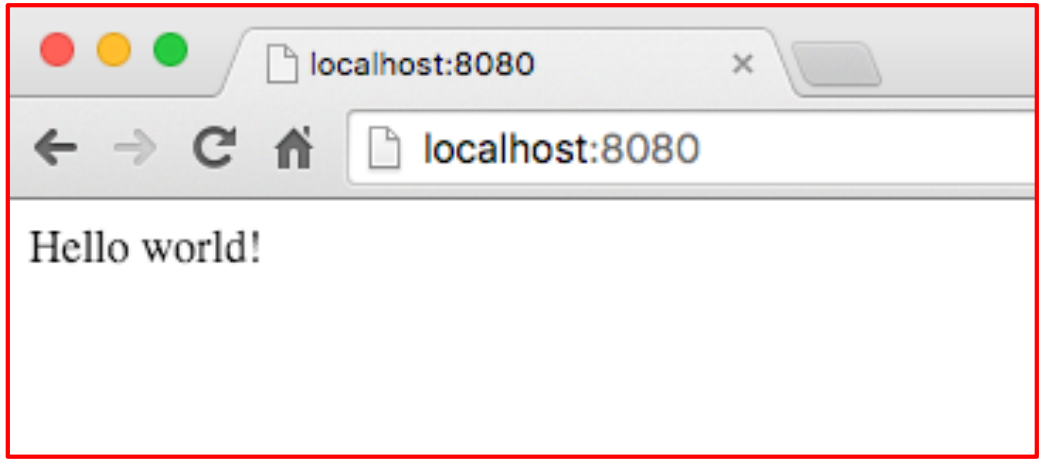
First web application

Example for starting a server and hosts an application that will be served at request reaching **http://127.0.0.1:8080/**

```
import cherrypy

class HelloWorld(object):
    @cherrypy.expose
    def index(self):
        return "Hello world!"

if __name__ == '__main__':
    cherrypy.tree.mount (HelloWorld())
    cherrypy.engine.start()
    cherrypy.engine.block()
```

Expose the method as an URL

Deploy the HelloWorld class and start the Web Server



First web application

The running code will display something like:

```
[24/Feb/2014:21:01:46] ENGINE Listening for SIGHUP.
[24/Feb/2014:21:01:46] ENGINE Listening for SIGTERM.
[24/Feb/2014:21:01:46] ENGINE Listening for SIGUSR1.
[24/Feb/2014:21:01:46] ENGINE Bus STARTING
CherryPy Checker:
The Application mounted at '' has an empty config.

[24/Feb/2014:21:01:46] ENGINE Started monitor thread 'Autoreloader'.
[24/Feb/2014:21:01:46] ENGINE Started monitor thread '_TimeoutMonitor'.
[24/Feb/2014:21:01:46] ENGINE Serving on http://127.0.0.1:8080
[24/Feb/2014:21:01:46] ENGINE Bus STARTED
```

Indicating the server that will handle **signals** for you

First web application

The running code will display something like:

```
[24/Feb/2014:21:01:46] ENGINE Listening for SIGHUP.
[24/Feb/2014:21:01:46] ENGINE Listening for SIGTERM.
[24/Feb/2014:21:01:46] ENGINE Listening for SIGUSR1.
[24/Feb/2014:21:01:46] ENGINE Bus STARTING
CherryPy Checker:
The Application mounted at '' has an empty config.

[24/Feb/2014:21:01:46] ENGINE Started monitor thread 'Autoreloader'.
[24/Feb/2014:21:01:46] ENGINE Started monitor thread '_TimeoutMonitor'.
[24/Feb/2014:21:01:46] ENGINE Serving on http://127.0.0.1:8080
[24/Feb/2014:21:01:46] ENGINE Bus STARTED
```

Current State of the server

First web application

The running code will display something like:

```
[24/Feb/2014:21:01:46] ENGINE Listening for SIGHUP.
[24/Feb/2014:21:01:46] ENGINE Listening for SIGTERM.
[24/Feb/2014:21:01:46] ENGINE Listening for SIGUSR1.
[24/Feb/2014:21:01:46] ENGINE Bus STARTING
CherryPy Checker:
The Application mounted at '' has an empty config.

[24/Feb/2014:21:01:46] ENGINE Started monitor thread 'Autoreloader'.
[24/Feb/2014:21:01:46] ENGINE Started monitor thread '_TimeoutMonitor'.
[24/Feb/2014:21:01:46] ENGINE Serving on http://127.0.0.1:8080
[24/Feb/2014:21:01:46] ENGINE Bus STARTED
```

application has no specific configuration set

First web application

The running code will display something like:

```
[24/Feb/2014:21:01:46] ENGINE Listening for SIGHUP.
[24/Feb/2014:21:01:46] ENGINE Listening for SIGTERM.
[24/Feb/2014:21:01:46] ENGINE Listening for SIGUSR1.
[24/Feb/2014:21:01:46] ENGINE Bus STARTING
CherryPy Checker:
The Application mounted at '' has an empty config.

[24/Feb/2014:21:01:46] ENGINE Started monitor thread 'Autoreloader'.
[24/Feb/2014:21:01:46] ENGINE Started monitor thread '_TimeoutMonitor'.
[24/Feb/2014:21:01:46] ENGINE Serving on http://127.0.0.1:8080
[24/Feb/2014:21:01:46] ENGINE Bus STARTED
```

Ready to accept incoming communications listening to http://127.0.0.1:8080

First web application with different paths

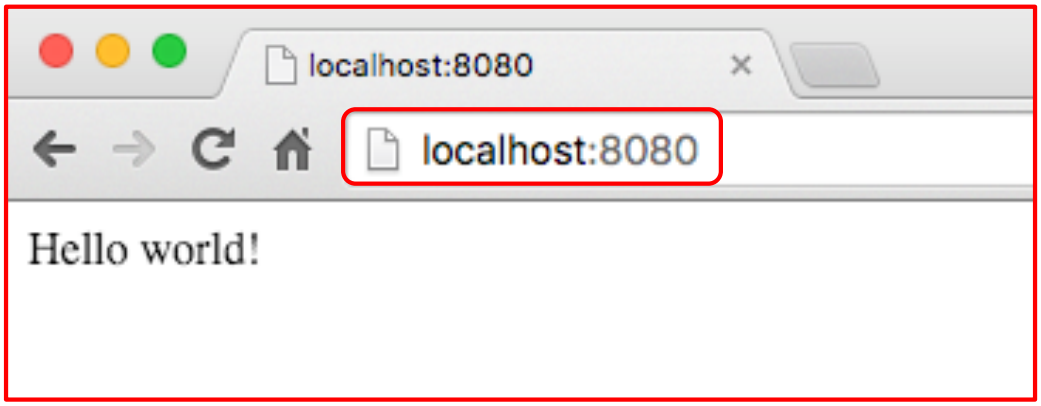
This example uses the **index()** method to handle **http://localhost:8080/** and the **generate()** method to handle **http://localhost:8080/generate**

```
import random
import string
import cherrypy

class StringGenerator(object):
        @cherrypy.expose
        def index(self):
                return "Hello world!"

        @cherrypy.expose
        def generate(self):
                return ''.join(random.sample(string.hexdigits, 8))

if __name__ == '__main__':
        cherrypy.tree.mount (StringGenerator())
        cherrypy.engine.start()
        cherrypy.engine.block()
```

First web application with different paths

This example uses the **index()** method to handle **http://localhost:8080/** and the **generate()** method to handle **http://localhost:8080/generate**

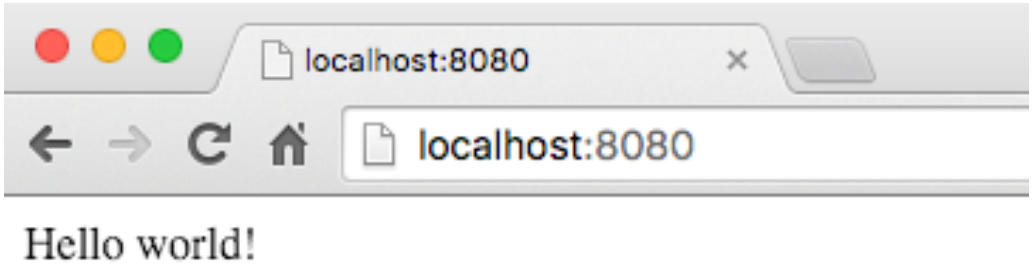
```
import random
import string
import cherrypy

class StringGenerator(object):
        @cherrypy.expose
        def index(self):
                return "Hello world!"

        @cherrypy.expose
        def generate(self):
                return ''.join(random.sample(string.hexdigits, 8))

if __name__ == '__main__':
        cherrypy.tree.mount (StringGenerator())
        cherrypy.engine.start()
        cherrypy.engine.block()
```

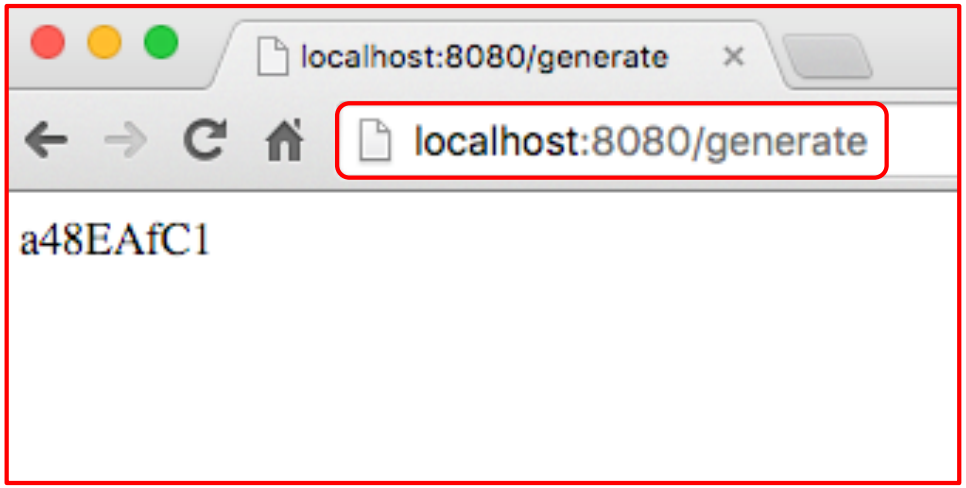


First web application with different paths

This example uses the **index()** method to handle **http://localhost:8080/** and the **generate()** method to handle **http://localhost:8080/generate**

```
import random
import string
import cherrypy

class StringGenerator(object):
        @cherrypy.expose
        def index(self):
                return "Hello world!"

        @cherrypy.expose
        def generate(self):
                return ''.join(random.sample(string.hexdigits, 8))

if __name__ == '__main__':
        cherrypy.tree.mount (StringGenerator())
        cherrypy.engine.start()
        cherrypy.engine.block()
```

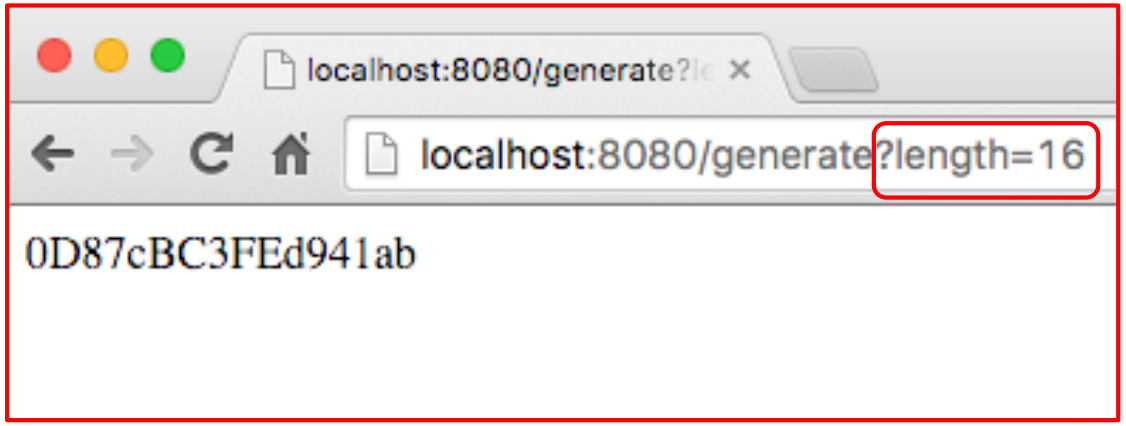




First web application with different paths and parameters

Let's now assume you wish to indicate the length of that string dynamically. This parameters provided **length** parameter e.g. **http://localhost:8080/generate?length=16**

```
import random
import string
import cherrypy

class StringGenerator(object):
        @cherrypy.expose
        def index(self):
                return "Hello world!"

        @cherrypy.expose
        def generate(self, length=8):
                return ''.join(random.sample(string.hexdigits, int(length)))

if __name__ == '__main__':
        cherrypy.tree.mount (StringGenerator())
        cherrypy.engine.start()
        cherrypy.engine.block()
```



In a URL, the section after **?** is called a query-string. **?** is unique for the whole URL

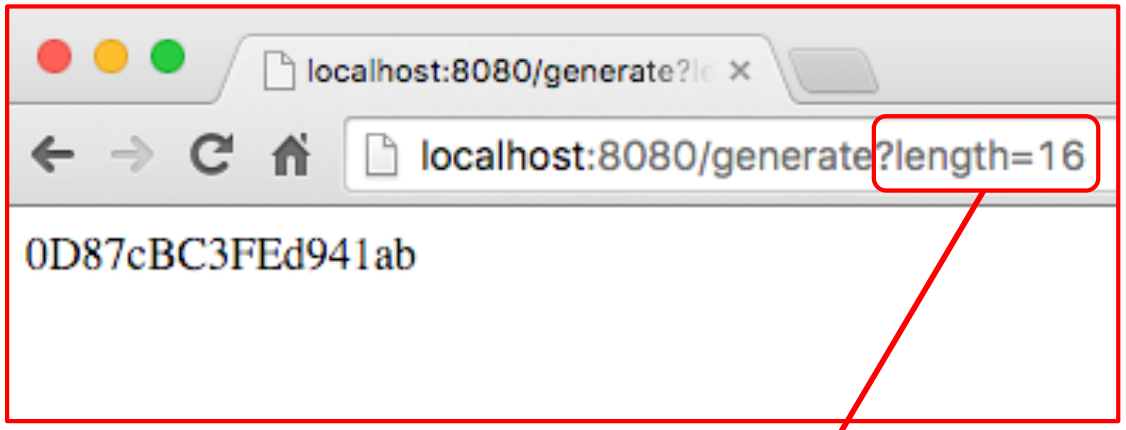
The query-string is used to contextualize the URL by passing a set of (key, value) pairs.

- The format is **key=value**.
- Each pair separated by a **&** character.

First web application with different paths and parameters

Let's now assume you wish to indicate the length of that string dynamically. This parameters provided **length** parameter e.g. **http://localhost:8080/generate?length=16**

```
import random
import string
import cherrypy

class StringGenerator(object):
        @cherrypy.expose
        def index(self):
                return "Hello world!"

        @cherrypy.expose
        def generate(self, length=8):
                return ''.join(random.sample(string.hexdigits, int(length)))

if __name__ == '__main__':
        cherrypy.tree.mount (StringGenerator())
        cherrypy.engine.start()
        cherrypy.engine.block()
```

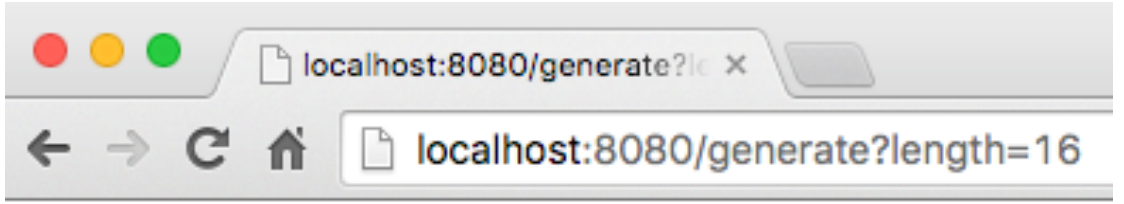


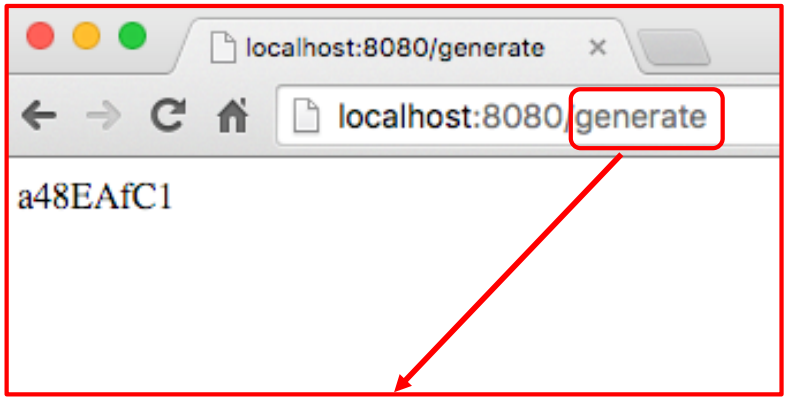
values are sent out from the client to our server as strings. In that case, the conversion to integer is needed

First web application with different paths and parameters

Let's now assume you wish to indicate the length of that string dynamically. This parameters provided **length** parameter e.g. **http://localhost:8080/generate?length=16**

```
import random
import string
import cherrypy

class StringGenerator(object):
        @cherrypy.expose
        def index(self):
                return "Hello world!"

        @cherrypy.expose
        def generate(self, length=8):
                return ''.join(random.sample(string.hexdigits, int(length)))

if __name__ == '__main__':
        cherrypy.tree.mount (StringGenerator())
        cherrypy.engine.start()
        cherrypy.engine.block()
```



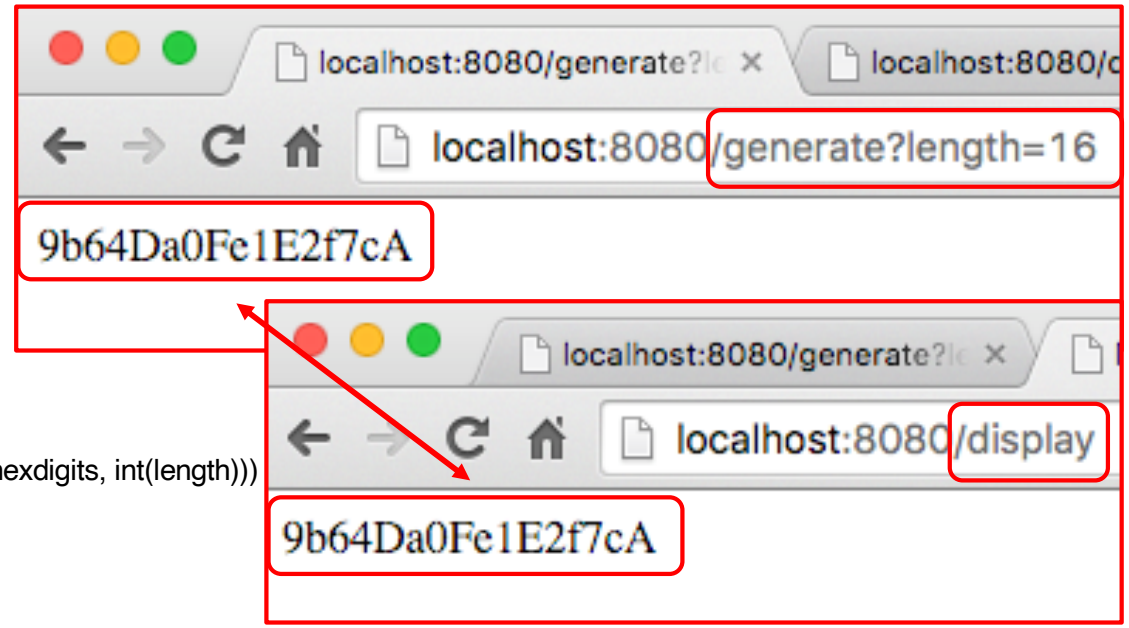


If **length** is not given, default argument values will support the URL.
Hence http://localhost:8080/generate still works.

Tracking end user activities

Often an application needs to track the user activities. **Session Identifier** is used for achieve this purpose. **http://localhost:8080/generate** generates a random string, then **http://localhost:8080/display** will show the string just generated.

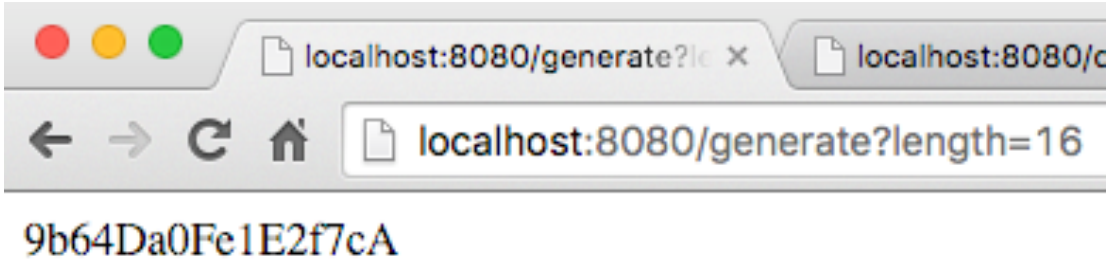
```
import random
import string
import cherrypy

class StringGenerator(object):
        @cherrypy.expose
        def index(self):
                return "Hello world!"

        @cherrypy.expose
        def generate(self, length=8):
                some_string = ''.join(random.sample(string.hexdigits, int(length)))
                cherrypy.session['mystring'] = some_string
                return some_string

        @cherrypy.expose
        def display(self):
                return cherrypy.session['mystring']
```

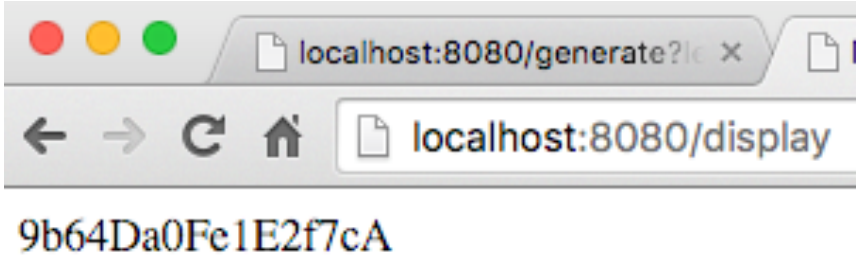


Tracking end user activities

Often an application needs to track the user activities. **Session Identifier** is used for achieve this purpose. **http://localhost:8080/generate** generates a random string, then **http://localhost:8080/display** will show the string just generated.

```
import random
import string
import cherrypy

class StringGenerator(object):
        @cherrypy.expose
        def index(self):
                return "Hello world!"

        @cherrypy.expose
        def generate(self, length=8):
                some_string = ''.join(random.sample(string.hexdigits, int(length)))
                cherrypy.session['mystring'] = some_string
                return some_string

        @cherrypy.expose
        def display(self):
                return cherrypy.session['mystring']
```






```
if __name__ == '__main__':
        conf = {
                '/': { 'tools.sessions.on': True
                }
        }
        cherrypy.tree.mount (StringGenerator(), '/', conf)
        cherrypy.engine.start()
        cherrypy.engine.block()
```

Enable the **session support** in the application

Providing static resources

Web applications are also made of static content such as javascript, files or images. CherryPy provides support to serve static content to end-users.

```
import os, os.path
import random
import string
import cherrypy

class StringGenerator(object):
        @cherrypy.expose
        def index(self):
                return """<html> <head>
                        <link href="/static/css/style.css" rel="stylesheet">
                        </head>
                        <body> <p>Hello World!!</p> </body>
                        </html>"""
```

```
if __name__ == '__main__':
        conf = {
          '/': {
                  'tools.sessions.on': True,
                  'tools.staticdir.root': os.path.abspath(os.getcwd())
           },
           '/static': {
                   'tools.staticdir.on': True,
                   'tools.staticdir.dir': './public'
           }
        }
        cherrypy.tree.mount (StringGenerator(), '/', conf)
        cherrypy.engine.start()
        cherrypy.engine.block()
```

Providing static resources

Web applications are also made of static content such as javascript, files or images. CherryPy provides support to serve static content to end-users.

CherryPy provides support to serve a single file or a complete directory structure. First, we indicate the **root** directory of all of our static content.

```
import os, os.path
import random
import string
import cherrypy

class StringGenerator(object):
        @cherrypy.expose
        def index(self):
                return """<html> <head>
                        <link href="/static/css/style.css" rel="stylesheet">
                        </head>
                        <body> <p>Hello World!!</p> </body>
                        </html>"""
```

```
if __name__ == '__main__':
        conf = {
         '/': {
                 'tools.sessions.on': True,
                 'tools.staticdir.root': os.path.abspath(os.getcwd())
         },
          '/static': {
                  'tools.staticdir.on': True,
                  'tools.staticdir.dir': './public'
          }
        }
        cherrypy.tree.mount (StringGenerator(), '/', conf)
        cherrypy.engine.start()
        cherrypy.engine.block()
```

Providing static resources

Web applications are also made of static content such as javascript, files or images. CherryPy provides support to serve static content to end-users.

CherryPy provides support to serve a single file or a complete directory structure. First, we indicate the **root** directory of all of our static content.

Then we indicate that all URLs which path segment starts with */static* will be served as static content. We map that URL to the *public* directory, a direct child of the *root* directory. The entire sub-tree of the *public* directory will be served as static content. CherryPy will map URLs to path within that directory. This is why */static/css/style.css* is found in *public/css/style.css* (in the local Hard Disk).

```
import os, os.path
import random
import string
import cherrypy

class StringGenerator(object):
        @cherrypy.expose
        def index(self):
                return """<html> <head>
                        <link href="/static/css/style.css" rel="stylesheet">
                        </head>
                        <body> <p>Hello World!!</p> </body>
                        </html>"""
```

```
if __name__ == '__main__':
        conf = {
          '/': {
                  'tools.sessions.on': True,
                  'tools.staticdir.root': os.path.abspath(os.getcwd())
          },
          '/static': {
                  'tools.staticdir.on': True,
                  'tools.staticdir.dir': './public'
          }
        }
        cherrypy.tree.mount (StringGenerator(), '/', conf)
        cherrypy.engine.start()
        cherrypy.engine.block()
```

Let’s REST…

REST Web Services

Representational state transfer *(REST) or* ***RESTful*** *web services is a way of providing interoperability between computer systems on the Internet following a* ***Client-server approach***

REST-compliant Web services allow requesting systems to ***access and manipulate textual representations of Web resources using a uniform and predefined set of stateless operations****.*

REST – Methods

- **HTTP Methods are a key corner stone in REST**
- They define the actions to be taken with a URL
- Proper **RESTful** webservices **expose all CRUD methods** (Create, Read, Update, Delete)

HTTP	REST - CRUD
POST	Create
GET	Read
PUT	Update or Modify or Replace
DELETE	Delete

Dealing with REST

Example of a web API following REST principles

```
import random
import string
import cherrypy

class StringGeneratorWebService(object):
        exposed = True


        def GET (self):
                return cherrypy.session['mystring']

        def POST (self, *uri, **params):
                some_string = ''.join(random.sample(string.hexdigits, int(params ['length']})))
                cherrypy.session['mystring'] = some_string
                return some_string

        def PUT (self, *uri, **params):
                cherrypy.session['mystring'] = params ['another_string']

        def DELETE (self):
                cherrypy.session.pop('mystring', None)
```

```
if __name__ == '__main__':
    conf = {
        '/': {
            'request.dispatch': cherrypy.dispatch.MethodDispatcher(),
             'tools.sessions.on': True,
        }
    }
    cherrypy.tree.mount (StringGeneratorWebService(), '/string', conf)

    cherrypy.config.update({'server.socket_host': '0.0.0.0'})
    cherrypy.config.update({'server.socket_port': 8080})

    cherrypy.engine.start()
    cherrypy.engine.block()
```

Dealing with REST

Example of a web API following REST principles

```
import random
import string
import cherrypy

class StringGeneratorWebService(object):
        exposed = True


        def GET (self):
                return cherrypy.session['mystring']

        def POST (self, *uri, **params):
                some_string = ''.join(random.sample(string.hexdigits, int(params ['length']})))
                cherrypy.session['mystring'] = some_string
                return some_string

        def PUT (self, *uri, **params):
                cherrypy.session['mystring'] = params ['another_string']

        def DELETE (self):
                cherrypy.session.pop('mystring', None)
```

```
if __name__ == '__main__':
    conf = {
        '/': {
            'request.dispatch': cherrypy.dispatch.MethodDispatcher(),
            'tools.sessions.on': True,
        }
    }
    cherrypy.tree.mount (StringGeneratorWebService(), '/string', conf)

    cherrypy.config.update({'server.socket_host': '0.0.0.0'})
    cherrypy.config.update({'server.socket_port': 8080})

    cherrypy.engine.start()
    cherrypy.engine.block()
```

Define a class with **CRUD** methods

Dealing with REST

Example of a web API following REST principles

```
import random
import string
import cherrypy

class StringGeneratorWebService(object):
        exposed = True

        def GET (self):
                return cherrypy.session['mystring']

        def POST (self, *uri, **params):
                some_string = ''.join(random.sample(string.hexdigits, int(params ['length'])))
                cherrypy.session['mystring'] = some_string
                return some_string

        def PUT (self, *uri, **params):
                cherrypy.session['mystring'] = params ['another_string']

        def DELETE (self):
                cherrypy.session.pop('mystring', None)
```

```
if __name__ == '__main__':
    conf = {
        '/': {
            'request.dispatch': cherrypy.dispatch.MethodDispatcher(),
            'tools.sessions.on': True,
        }
    }
    cherrypy.tree.mount (StringGeneratorWebService(), '/string', conf)

    cherrypy.config.update({'server.socket_host': '0.0.0.0'})
    cherrypy.config.update({'server.socket_port': 8080})

    cherrypy.engine.start()
    cherrypy.engine.block()
```

Needed for exposing the Web Services

Dealing with REST

Example of a web API following REST principles

```
import random
import string
import cherrypy

class StringGeneratorWebService(object):
        exposed = True


        def GET (self):
                return cherrypy.session['mystring']

        def POST (self, *uri, **params):
                some_string = ''.join(random.sample(string.hexdigits, int(params ['length']})))
                cherrypy.session['mystring'] = some_string
                return some_string

        def PUT (self, *uri, **params):
                cherrypy.session['mystring'] = params ['another_string']

        def DELETE (self):
                cherrypy.session.pop('mystring', None)
```

```
if __name__ == '__main__':
    conf = {
        '/': {
            'request.dispatch': cherrypy.dispatch.MethodDispatcher(),
            'tools.sessions.on': True,
        }
    }
    cherrypy.tree.mount (StringGeneratorWebService(), '/string', conf)

    cherrypy.config.update({'server.socket_host': '0.0.0.0'})
    cherrypy.config.update({'server.socket_port': 8080})

    cherrypy.engine.start()
    cherrypy.engine.block()
```

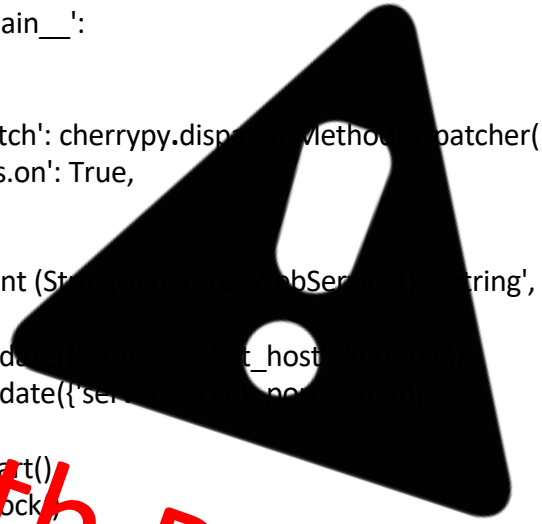
This will switch from the default mechanism of matching URLs to the HTT-compliant approach

Dealing with REST

Example of a web API following REST principles

```
import random
import string
import cherrypy

class StringGeneratorWebService(object):
        exposed = True


        def GET (self):
                return cherrypy.session['mystring']

        def POST (self, *uri, **params):
                some_string = ''.join(random.sample(string.hexdigits, int(params['length'])))
                cherrypy.session['mystring'] = some_string
                return some_string

        def PUT (self, *uri, **params):
                cherrypy.session['mystring'] = params ['another_string']

        def DELETE (self):
                cherrypy.session.pop('mystring', None)
```

```
if __name__ == '__main__':
    conf = {
        '/': {
            'request.dispatch': cherrypy.dispatch.MethodDispatcher(),
            'tools.sessions.on': True,
        }
    }
    cherrypy.tree.mount (StringGeneratorWebService(), 'string', conf)

    cherrypy.config.update({'server.socket_host': '0.0.0.0',
                            'server.socket_port': 8080,
                           })

    cherrypy.engine.start()
    cherrypy.engine.block()
```



We will work with REST

USE THIS APPROACH

RESTful-style (paths)

Cherrypy is able to read and manage the **paths**

```
import random
import string
import cherrypy

class StringGeneratorWebService(object):
        exposed = True

        def GET (self, * uri):
            # “uri” can be managed as an array
            return (”uri: %s; uri length %s" % (str (uri), len(uri)))
```

```
if __name__ == '__main__':
    conf = {
        '/': {
            'request.dispatch': cherrypy.dispatch.MethodDispatcher(),
             'tools.sessions.on': True,
        }
    }
    cherrypy.tree.mount (StringGeneratorWebService(), '/string', conf)

    cherrypy.config.update({'server.socket_host': '0.0.0.0’})
    cherrypy.config.update({'server.socket_port’: 8080})

    cherrypy.engine.start()
    cherrypy.engine.block()
```

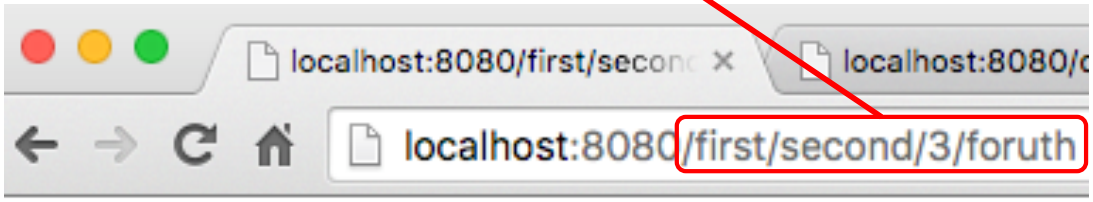
RESTful-style (paths)

Cherrypy is able to read and manage the **paths**

```
import random
import string
import cherrypy

class StringGeneratorWebService(object):
        exposed = True

        def GET (self, *uri):
            # "uri" can be managed as an array
            return ("uri: %s; uri length %s" % (str (uri), len(uri)))
```

NB: single star *

```
if __name__ == '__main__':
    conf = {
        '/': {
            'request.dispatch': cherrypy.dispatch.MethodDispatcher(),
            'tools.sessions.on': True,
        }
    }
    cherrypy.tree.mount (StringGeneratorWebService(), '/string', conf)

    cherrypy.config.update({'server.socket_host': '0.0.0.0'})
    cherrypy.config.update({'server.socket_port': 8080})

    cherrypy.engine.start()
    cherrypy.engine.block()
```



uri: ('first', 'second', '3', 'foruth'); uri length 4

RESTful-style (parameters)

Cherrypy is able to read and manage **parameters** in the **URL**

```
import random
import string
import cherrypy

class StringGeneratorWebService(object):
        exposed = True

        def GET (self, ** params):
            # “params” can be managed as a dictonary
            return ("Params: %s; params length %s"
                                        % (str (params), len(params)))
```

```
if __name__ == '__main__':
    conf = {
        '/': {
            'request.dispatch': cherrypy.dispatch.MethodDispatcher(),
             'tools.sessions.on': True,
        }
    }
    cherrypy.tree.mount (StringGeneratorWebService(), '/string', conf)

    cherrypy.config.update({'server.socket_host': '0.0.0.0’})
    cherrypy.config.update({'server.socket_port’: 8080})

    cherrypy.engine.start()
    cherrypy.engine.block()
```

RESTful-style (parameters)

Cherrypy is able to read and manage **parameters** in the **URL**

```
import random
import string
import cherrypy

class StringGeneratorWebService(object):
        exposed = True

        def GET (self,**params):
            # "params" can be managed as a dictonary
            return ("Params: %s; params length %s"
                                        % (str (params), len(params)))
```

NB: double star **

```
if __name__ == '__main__':
    conf = {
        '/': {
            'request.dispatch': cherrypy.dispatch.MethodDispatcher(),
             'tools.sessions.on': True,
        }
    }
    cherrypy.tree.mount (StringGeneratorWebService(), '/string', conf)

    cherrypy.config.update({'server.socket_host': '0.0.0.0'})
    cherrypy.config.update({'server.socket_port': 8080})

    cherrypy.engine.start()
    cherrypy.engine.block()
```

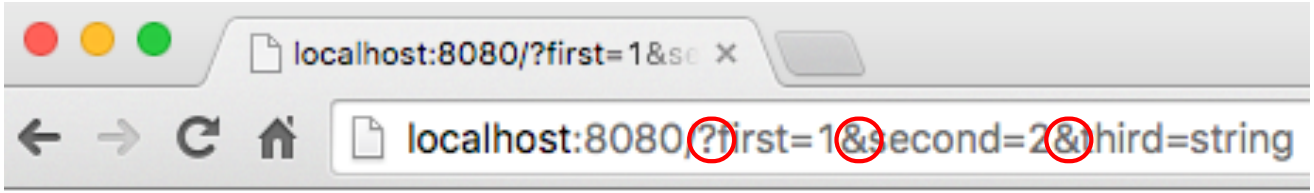
params: {'second': u'2', 'third': u'string', 'first': u'1'}; params length 3

RESTful-style (parameters)

Cherrypy is able to read and manage **parameters** in the **URL**

```
import random
import string
import cherrypy

class StringGeneratorWebService(object):
        exposed = True

        def GET (self, ** params):
            # "params" can be managed as a dictonary
            return ("Params: %s; params length %s"
                                    % (str (params), len(params)))
```

```
if __name__ == '__main__':
    conf = {
        '/': {
            'request.dispatch': cherrypy.dispatch.MethodDispatcher(),
            'tools.sessions.on': True,
        }
    }
    cherrypy.tree.mount (StringGeneratorWebService(), '/string', conf)

    cherrypy.config.update({'server.socket_host': '0.0.0.0'})
    cherrypy.config.update({'server.socket_port': 8080})

    cherrypy.engine.start()
    cherrypy.engine.block()
```



In a URL, the section after **?** is called a query-string. **?** is unique for the whole URL

The query-string is used to contextualize the URL by passing a set of (key, value) pairs.

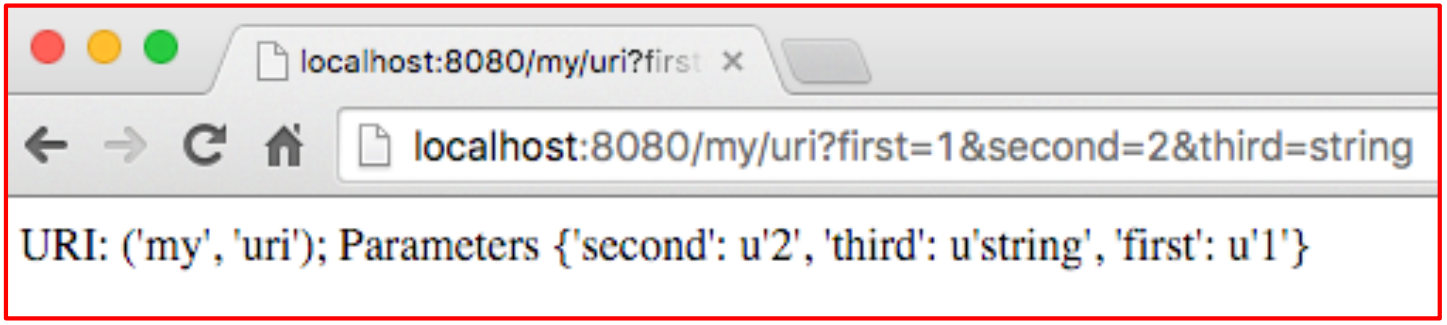
- The format is **key=value**.
- Each pair being separated by a **&** character.

RESTful-style (paths and parameters)

Cherrypy is able to read and manage **parameters** in the **URL**

```
import random
import string
import cherrypy

class StringGeneratorWebService(object):
        exposed = True

        def GET (self, *uri, ** params):
            return ("URI: %s; Parameters %s" % (str (uri), str(params)))
```

```
if __name__ == '__main__':
    conf = {
        '/': {
            'request.dispatch': cherrypy.dispatch.MethodDispatcher(),
            'tools.sessions.on': True,
        }
    }
    cherrypy.tree.mount (StringGeneratorWebService(), '/string', conf)

    cherrypy.config.update({'server.socket_host': '0.0.0.0'})
    cherrypy.config.update({'server.socket_port': 8080})

    cherrypy.engine.start()
    cherrypy.engine.block()
```

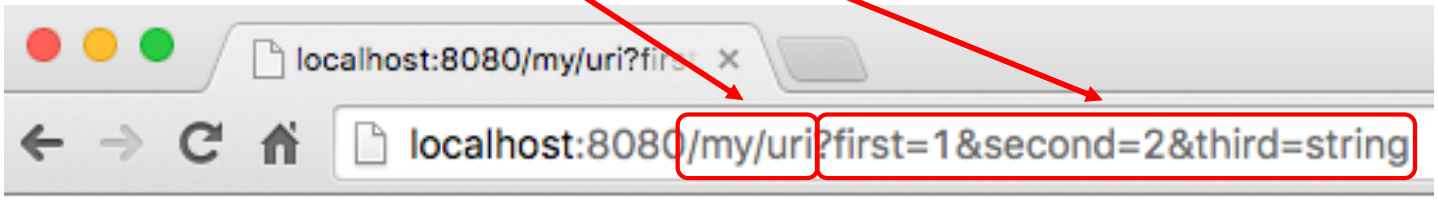


RESTful-style (paths and parameters)

Cherrypy is able to read and manage **parameters** in the **URL**

```
import random
import string
import cherrypy

class StringGeneratorWebService(object):
        exposed = True

        def GET (self, *uri, ** params):
            return ("URI: %s; Parameters %s" % (str (uri), str(params)))
```

```
if __name__ == '__main__':
    conf = {
        '/': {
            'request.dispatch': cherrypy.dispatch.MethodDispatcher(),
            'tools.sessions.on': True,
        }
    }
    cherrypy.tree.mount (StringGeneratorWebService(), '/string', conf)

    cherrypy.config.update({'server.socket_host': '0.0.0.0'})
    cherrypy.config.update({'server.socket_port': 8080})

    cherrypy.engine.start()
    cherrypy.engine.block()
```



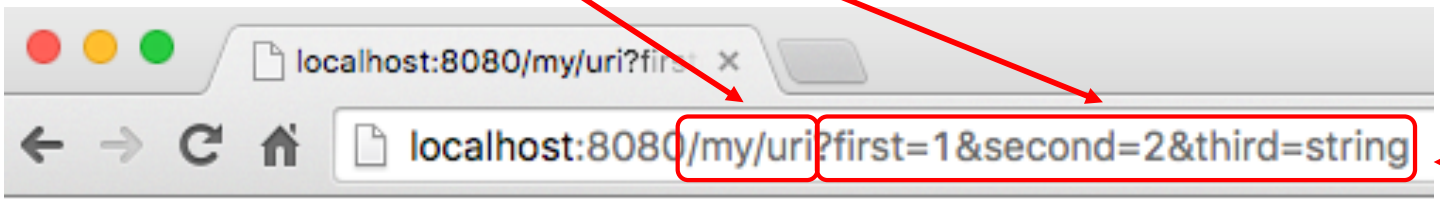
URI: ('my', 'uri'); Parameters {'second': u'2', 'third': u'string', 'first': u'1'}

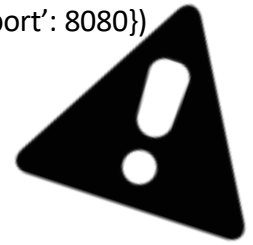
RESTful-style (paths and parameters)

Cherrypy is able to read and manage **parameters** in the **URL**

```
import random
import string
import cherrypy

class StringGeneratorWebService(object):
        exposed = True

        def GET (self, *uri, ** params):
            return ("URI: %s; Parameters %s" % (str (uri), str(params)))
```

```
if __name__ == '__main__':
    conf = {
        '/': {
            'request.dispatch': cherrypy.dispatch.MethodDispatcher(),
            'tools.sessions.on': True,
        }
    }
    cherrypy.tree.mount (StringGeneratorWebService(), '/string', conf)

    cherrypy.config.update({'server.socket_host': '0.0.0.0'})
    cherrypy.config.update({'server.socket_port': 8080})

    cherrypy.engine.start()
    cherrypy.engine.block()
```





Parameters always end the URL

RESTful-style (paths and parameters)

Cherrypy is able to read and manage **parameters** in the **URL**
This is valid also for **POST**, **PUT** and **DELETE** methods.

```
import random
import string
import cherrypy

class StringGeneratorWebService(object):
        exposed = True


        def GET (self, *uri, ** params):
                # do something
                return something

        def POST (self, *uri, ** params):
                # do something
                mystring = "POST RESPONSE: “
                my_string += cherrypy.request.body.read()
                return my_string

        def PUT (self, *uri, ** params):
                # do something
                mystring = ”PUT RESPONSE: “
                my_string += cherrypy.request.body.read()

        def DELETE (self *uri, ** params):
                # do something
```

```
if __name__ == '__main__':
    conf = {
        '/': {
            'request.dispatch': cherrypy.dispatch.MethodDispatcher(),
            'tools.sessions.on': True,
        }
    }
    cherrypy.tree.mount (StringGeneratorWebService(), '/string', conf)

    cherrypy.config.update({'server.socket_host': '0.0.0.0’})
    cherrypy.config.update({'server.socket_port’: 8080})

    cherrypy.engine.start()
    cherrypy.engine.block()
```

RESTful-style (paths and parameters)

Cherrypy is able to read and manage **parameters** in the **URL**
This is valid also for **POST**, **PUT** and **DELETE** methods.

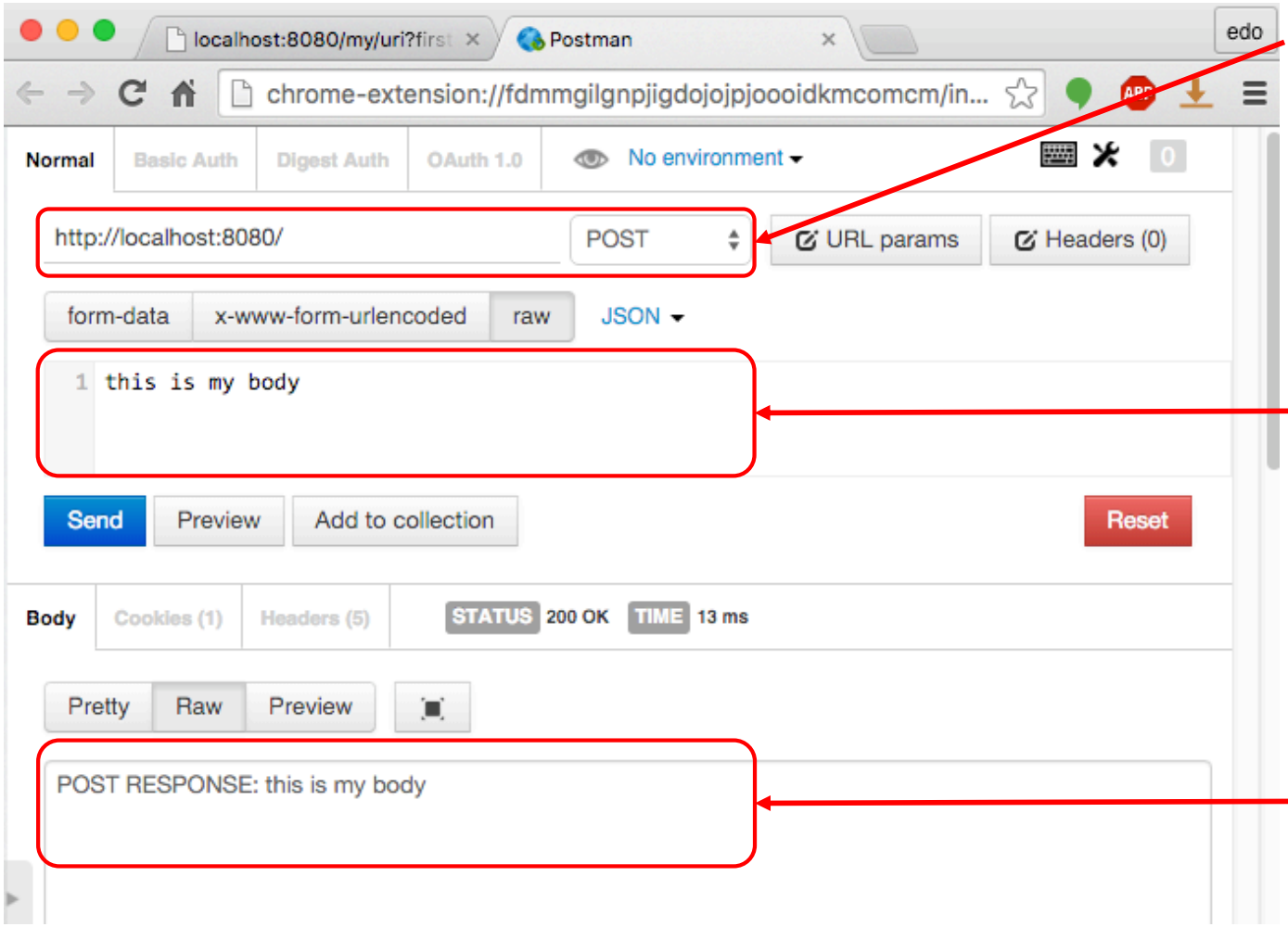
```
import random
import string
import cherrypy

class StringGeneratorWebService(object):
        exposed = True


        def GET (self, *uri, ** params):
                # do something
                return something

        def POST (self, *uri, ** params):
                # do something
                mystring = "POST RESPONSE: "
                my_string += cherrypy.request.body.read()
                return my_string

        def PUT (self, *uri, ** params):
                # do something
                mystring = "PUT RESPONSE: "
                my_string += cherrypy.request.body.read()

        def DELETE (self *uri, ** params):
                # do something
```

```
if __name__ == '__main__':
    conf = {
        '/': {
            'request.dispatch': cherrypy.dispatch.MethodDispatcher(),
            'tools.sessions.on': True,
        }
    }
    cherrypy.tree.mount (StringGeneratorWebService(), '/string', conf)

    cherrypy.config.update({'server.socket_host': '0.0.0.0'})
    cherrypy.config.update({'server.socket_port': 8080})

    cherrypy.engine.start()
    cherrypy.engine.block()
```

Read the content sent in the POST/PUT body

RESTful-style (paths and parameters)

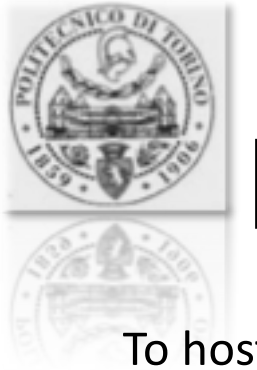
SUGGESTION: Use **Postman**, a chrome plugin, for testing REST web services by managing HTTP requests



Calling the POST method

Message sent in the BODY

POST response



Hosting single application

To host a single application the easy way is to use **cherrypy.tree.mount()** function. It takes at least an argument, the instance of the application to host. Use it together with **cherrypy.engine.start()** and **cherrypy.engine.block()**

```
class StringGeneratorWebService(object):
        exposed = True
        def GET (self):
                ...
        def POST (self, length=8):
                …
        def PUT (self, another_string):
                …
        def DELETE (self):
                …
```

```
if __name__ == '__main__':
   conf = {
      '/': {
         'request.dispatch': cherrypy.dispatch.MethodDispatcher(),
         'tools.sessions.on': True,
      }
   }
   cherrypy.tree.mount (StringGeneratorWebService())

   cherrypy.config.update({'server.socket_host': '0.0.0.0'})
   cherrypy.config.update({'server.socket_port': 8080})

   cherrypy.engine.start()
   cherrypy.engine.block()
```

cherrypy.tree.mount() with 1 parameters.

The application will be available at http://hostname:port/

Hosting single application

To host a single application the easy way is to use **cherrypy.tree.mount()** function.
It takes at least an argument, the instance of the application to host. Use it together with **cherrypy.engine.start()** and **cherrypy.engine.block()**
Two other settings are optional:

1. the base path at which the application will be accessible from;

```
class StringGeneratorWebService(object):
        exposed = True
        def GET (self):
                ...
        def POST (self, length=8):
                …
        def PUT (self, another_string):
                …
        def DELETE (self):
                …
```

```
if __name__ == '__main__':
  conf = {
    '/': {
      'request.dispatch': cherrypy.dispatch.MethodDispatcher(),
       'tools.sessions.on': True,
    }
  }
  cherrypy.tree.mount(StringGeneratorWebService(), '/string')

  cherrypy.config.update({'server.socket_host': '0.0.0.0'})
  cherrypy.config.update({'server.socket_port': 8080})

  cherrypy.engine.start()
  cherrypy.engine.block()
```

cherrypy.tree.mount() with 2 parameters.

The application will be available at http://hostname:port**/string**

Hosting single application

To host a single application the easy way is to use **cherrypy.tree.mount()** function.
It takes at least an argument, the instance of the application to host. Use it together with **cherrypy.engine.start()** and **cherrypy.engine.block()**
Two other settings are optional:

1. the base path at which the application will be accessible from;
2. the dictionary to configure the application.

```
class StringGeneratorWebService(object):
        exposed = True
        def GET (self):
                ...
        def POST (self, length=8):
                …
        def PUT (self, another_string):
                …
        def DELETE (self):
                …
```

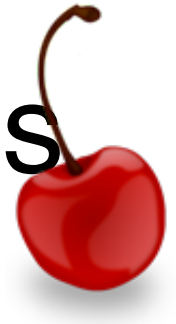
```
if __name__ == '__main__':
  conf = {
    '/': {
      'request.dispatch': cherrypy.dispatch.MethodDispatcher(),
       'tools.sessions.on': True,
    }
  }
  cherrypy.tree.mount(StringGeneratorWebService(), '/string', conf)

  cherrypy.config.update({'server.socket_host': '0.0.0.0'})
  cherrypy.config.update({'server.socket_port': 8080})

  cherrypy.engine.start()
  cherrypy.engine.block()
```

cherrypy.tree.mount() with 3 parameters.

The application will be available at http://hostname:port**/string.**

"conf" provides specific settings for the application

Hosting multiple applications

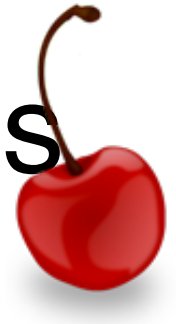


cherrypy.tree.mount() is used also to host multiple applications. Use it together with **cherrypy.engine.start()** and **cherrypy.engine.block()**

```
class StringGeneratorWebService(object):
        exposed = True
        def GET (self):
                …
        def POST (self, length=8):
                …
        def PUT (self, another_string):
                …
        def DELETE (self):
                …

class OtherWebService(object):
        exposed = True
        def GET (self):
                …
        def POST (self, length=8):
                …
        def PUT (self, another_string):
                …
        def DELETE (self):
                …
```

```
if __name__ == '__main__':
   conf = {
      '/': {
         'request.dispatch': cherrypy.dispatch.MethodDispatcher(),
          'tools.sessions.on': True,
      }
   }
   cherrypy.tree.mount (StringGeneratorWebService(), '/string', conf)
   cherrypy.tree.mount (OtherWebService(), '/other', conf)

   cherrypy.config.update({'server.socket_host': '0.0.0.0'})
   cherrypy.config.update({'server.socket_port': 8080})

   cherrypy.engine.start()
   cherrypy.engine.block()
```

Hosting multiple applications



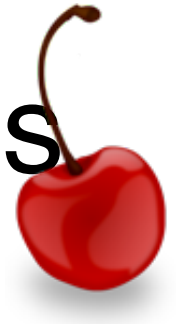
cherrypy.tree.mount() is used also to host multiple applications. Use it together with **cherrypy.engine.start()** and **cherrypy.engine.block()**

```
class StringGeneratorWebService(object):
        exposed = True
        def GET (self):
                …
        def POST (self, length=8):
                …
        def PUT (self, another_string):
                …
        def DELETE (self):
                …

class OtherWebService(object):
        exposed = True
        def GET (self):
                …
        def POST (self, length=8):
                …
        def PUT (self, another_string):
                …
        def DELETE (self):
                …
```

```
if __name__ == '__main__':
    conf = {
        '/': {
            'request.dispatch': cherrypy.dispatch.MethodDispatcher(),
            'tools.sessions.on': True,
        }
    }
    cherrypy.tree.mount (StringGeneratorWebService(), '/string', conf)
    cherrypy.tree.mount (OtherWebService(), '/other', conf)

    cherrypy.config.update({'server.socket_host': '0.0.0.0'})
    cherrypy.config.update({'server.socket_port': 8080})

    cherrypy.engine.start()
    cherrypy.engine.block()
```

Hosting two applications

Hosting multiple applications



cherrypy.tree.mount() is used also to host multiple applications. Use it together with **cherrypy.engine.start()** and **cherrypy.engine.block()**

```
class StringGeneratorWebService(object):
        exposed = True
        def GET (self):
                …
        def POST (self, length=8):
                …
        def PUT (self, another_string):
                …
        def DELETE (self):
                …

class OtherWebService(object):
        exposed = True
        def GET (self):
                …
        def POST (self, length=8):
                …
        def PUT (self, another_string):
                …
        def DELETE (self):
                …
```

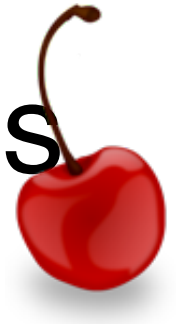
```
if __name__ == '__main__':
   conf = {
      '/': {
         'request.dispatch': cherrypy.dispatch.MethodDispatcher(),
          'tools.sessions.on': True,
      }
   }
   cherrypy.tree.mount (StringGeneratorWebService(), '/string', conf)
   cherrypy.tree.mount (OtherWebService(), '/other', conf)

   cherrypy.config.update({'server.socket_host': '0.0.0.0'})
   cherrypy.config.update({'server.socket_port': 8080})

   cherrypy.engine.start()
   cherrypy.engine.block()
```

Hosting two applications

http://hostname:port**/string**

http://hostname:port**/other**

Hosting multiple applications



cherrypy.tree.mount() is used also to host multiple applications. Use it together with **cherrypy.engine.start()** and **cherrypy.engine.block()**

```
class StringGeneratorWebService(object):
        exposed = True
        def GET (self):
                …
        def POST (self, length=8):
                …
        def PUT (self, another_string):
                …
        def DELETE (self):
                …

class OtherWebService(object):
        exposed = True
        def GET (self):
                …
        def POST (self, length=8):
                …
        def PUT (self, another_string):
                …
        def DELETE (self):
                …
```

```
if __name__ == '__main__':
   conf = {
      '/': {
         'request.dispatch': cherrypy.dispatch.MethodDispatcher(),
          'tools.sessions.on': True,
      }
   }
   cherrypy.tree.mount (StringGeneratorWebService(), '/string', conf)
   cherrypy.tree.mount (OtherWebService(), '/other', conf)

   cherrypy.config.update({'server.socket_host': '0.0.0.0'})
   cherrypy.config.update({'server.socket_port': 8080})

   cherrypy.engine.start()
   cherrypy.engine.block()
```

Can be the same or two different "configuration" dictionaries

Cherrypy basic settings

cherrypy.tree.mount() is used also to host multiple applications. Use it together with **cherrypy.engine.start()** and **cherrypy.engine.block()**

```
class StringGeneratorWebService(object):
        exposed = True
        def GET (self):
                …
        def POST (self, length=8):
                …
        def PUT (self, another_string):
                …
        def DELETE (self):
                …

class OtherWebService(object):
        exposed = True
        def GET (self):
                …
        def POST (self, length=8):
                …
        def PUT (self, another_string):
                …
        def DELETE (self):
                …
```

```
if __name__ == '__main__':
   conf = {
      '/': {
         'request.dispatch': cherrypy.dispatch.MethodDispatcher(),
          'tools.sessions.on': True,
      }
   }
   cherrypy.tree.mount (StringGeneratorWebService(), '/string', conf)
   cherrypy.tree.mount (OtherWebService(), '/other', conf)

   cherrypy.config.update({'server.socket_host': '0.0.0.0'})
   cherrypy.config.update({'server.socket_port': 8080})

   cherrypy.engine.start()
   cherrypy.engine.block()
```

To make the webservice reachable from other machines (by default cherrypy allows only connections from the localhost)

Cherrypy basic settings

cherrypy.tree.mount() is used also to host multiple applications. Use it together with **cherrypy.engine.start()** and **cherrypy.engine.block()**

```
class StringGeneratorWebService(object):
        exposed = True
        def GET (self):
                …
        def POST (self, length=8):
                …
        def PUT (self, another_string):
                …
        def DELETE (self):
                …

class OtherWebService(object):
        exposed = True
        def GET (self):
                …
        def POST (self, length=8):
                …
        def PUT (self, another_string):
                …
        def DELETE (self):
                …
```

```
if __name__ == '__main__':
   conf = {
      '/': {
         'request.dispatch': cherrypy.dispatch.MethodDispatcher(),
          'tools.sessions.on': True,
      }
   }
   cherrypy.tree.mount (StringGeneratorWebService(), '/string', conf)
   cherrypy.tree.mount (OtherWebService(), '/other', conf)

   cherrypy.config.update({'server.socket_host': '0.0.0.0'})
   cherrypy.config.update({'server.socket_port': 8080})

   cherrypy.engine.start()
   cherrypy.engine.block()
```

To change the default cherrypy port (8080) with another of your choice between 1025 e 65534.

Cherrypy basic settings

cherrypy.tree.mount() is used also to host multiple applications. Use it together with **cherrypy.engine.start()** and **cherrypy.engine.block()**

```
class StringGeneratorWebService(object):
        exposed = True
        def GET (self):
                …
        def POST (self, length=8):
                …
        def PUT (self, another_string):
                …
        def DELETE (self):
                …

class OtherWebService(object):
        exposed = True
        def GET (self):
                …
        def POST (self, length=8):
                …
        def PUT (self, another_string):
                …
        def DELETE (self):
                …
```

```
if __name__ == '__main__':
  conf = {
    '/': {
      'request.dispatch': cherrypy.dispatch.MethodDispatcher(),
       'tools.sessions.on': True,
    }
  }
  cherrypy.tree.mount (StringGeneratorWebService(), '/string', conf)
  cherrypy.tree.mount (OtherWebService(), '/other', conf)

  cherrypy.config.update({'server.socket_host': '0.0.0.0'})
  cherrypy.config.update({'server.socket_port': 9090})

  cherrypy.engine.start()
  cherrypy.engine.block()
```

To change the default cherrypy port (8080) with another of your choice between 1025 e 65534.

Cherrypy basic settings

cherrypy.tree.mount() is used also to host multiple applications. Use it together with **cherrypy.engine.start()** and **cherrypy.engine.block()**

```
class StringGeneratorWebService(object):
        exposed = True
        def GET (self):
                …
        def POST (self, length=8):
                …
        def PUT (self, another_string):
                …
        def DELETE (self):
                …

class OtherWebService(object):
        exposed = True
        def GET (self):
                …
        def POST (self, length=8):
                …
        def PUT (self, another_string):
                …
        def DELETE (self):
                …
```

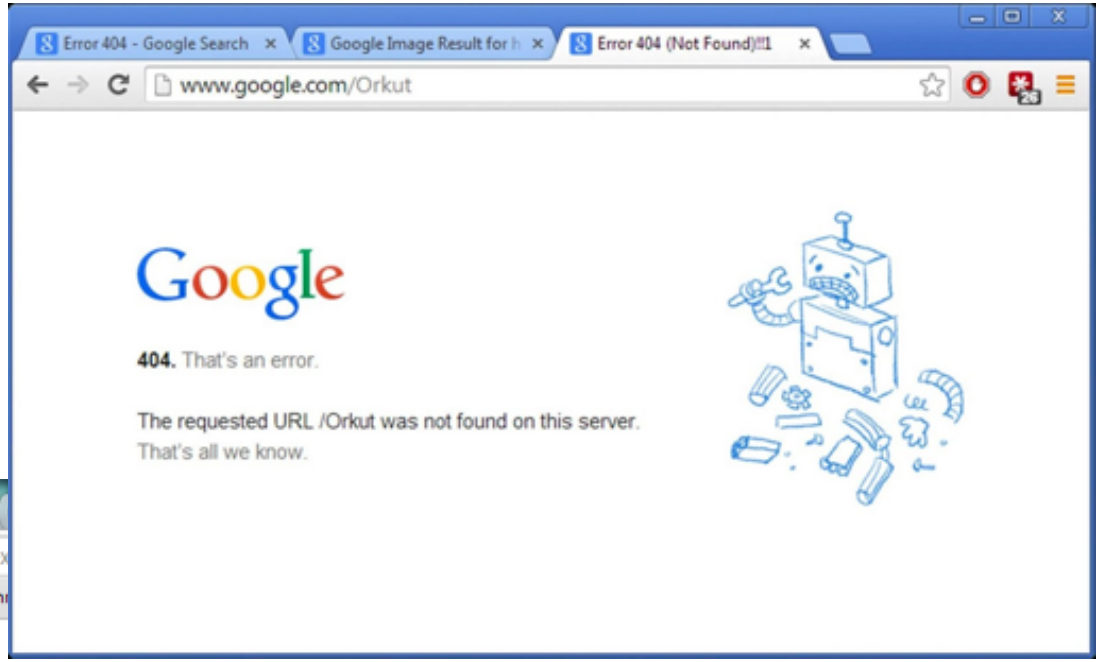
```
if __name__ == '__main__':
   conf = {
      '/': {
         'request.dispatch': cherrypy.dispatch.MethodDispatcher(),
          'tools.sessions.on': True,
      }
   }
   cherrypy.tree.mount (StringGeneratorWebService(), '/string', conf)
   cherrypy.tree.mount (OtherWebService(), '/other', conf)

   cherrypy.config.update({'server.socket_host': '0.0.0.0'})
   cherrypy.config.update({'server.socket_port': 8080})

   cherrypy.engine.start()
   cherrypy.engine.block()
```

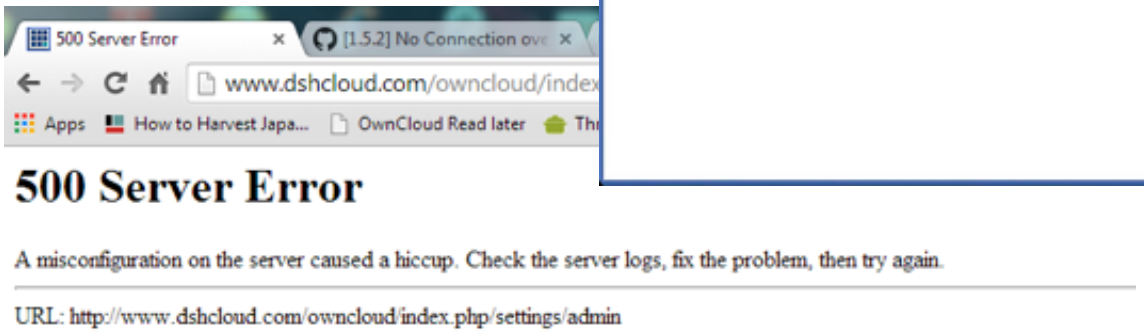
To start cherrypy engine

HTTP status codes

HTTP provides status codes for defining the outcome of the communication. They are organized in five classes:

- **1xx** – Informational
- **2xx** – Successful
- **3xx** – Redirection
- **4xx** – Client Error
- **5xx** – Server Error





HTTP status codes: 2xx – Sucessful

This class indicates that the client's request was successfully received, understood, and accepted.

Code	Name	Description
200	OK	The request has succeeded.
201	Created	The request has been fulfilled and resulted in a new resource being created.
202	Accepted	The request has been accepted for processing, but the processing has not been completed.
…	…	…

HTTP status codes: 4xx – Client Error

This class is intended for cases in which the client seems to have erred. The server SHOULD include an entity containing an explanation of the error situation, and whether it is a temporary or permanent condition.

Code	Name	Description
400	Bad Request	The request could not be understood by the server due to malformed syntax.
401	Unauthorized	The request requires user authentication.
403	Forbidden	The server understood the request, but is refusing to fulfill it.
404	Not Found	The server has not found anything matching the Request-URI.
…	…	…

HTTP status codes: 5xx – Server Error

This class indicates cases in which the server is aware that it has erred or is incapable of performing the request. The server SHOULD include an entity containing an explanation of the error situation, and whether it is a temporary or permanent condition.

Code	Name	Description
500	Internal Server Error	The server encountered an unexpected condition which prevented it from fulfilling the request.
501	Not Implemented	The server does not support the functionality required to fulfill the request. This is the appropriate response when the server does not recognize the request method and is not capable of supporting it for any resource.
503	Service Unavailable	The server is currently unable to handle the request due to a temporary overloading or maintenance of the server.
…	…	…

Raising HTTP Status codes

Use **raise cherrypy.HTTPError()** to manage an HTTP status codes

```
class StringGeneratorWebService(object):
        exposed = True
        def GET (self):
            if (len(uri) > 0 and uri[0] == "ciao"):
                return ("URI: %s; Parameters %s" % (str (uri), str(params))
            else:
                raise cherrypy.HTTPError(404, "Error message")
        def POST (self, length=8):
                …
        def PUT (self, another_string):
                …
        def DELETE (self):
                …
```

```
if __name__ == '__main__':
    conf = {
        '/': {
            'request.dispatch': cherrypy.dispatch.MethodDispatcher(),
            'tools.sessions.on': True,
        }
    }
    cherrypy.tree.mount (StringGeneratorWebService(), '/string', conf)
    cherrypy.tree.mount (OtherWebService(), '/other', conf)

    cherrypy.config.update({'server.socket_host': '0.0.0.0'})
    cherrypy.config.update({'server.socket_port': 8080})

    cherrypy.engine.start()
    cherrypy.engine.block()
```

Raising HTTP Status codes

Use **raise cherrypy.HTTPError()** to manage an HTTP status codes

```
class StringGeneratorWebService(object):
        exposed = True
        def GET (self):
            if (len(uri) > 0 and uri[0] == "ciao"):
                return ("URI: %s; Parameters %s" % (str (uri), str(params))
            else:
                raise cherrypy.HTTPError(404, "Error message")
        def POST (self, length=8):
                …
        def PUT (self, another_string):
                …
        def DELETE (self):
                …
```

```
if __name__ == '__main__':
    conf = {
        '/': {
            'request.dispatch': cherrypy.dispatch.MethodDispatcher(),
            'tools.sessions.on': True,
        }
    }
    cherrypy.tree.mount (StringGeneratorWebService(), '/string', conf)
    cherrypy.tree.mount (OtherWebService(), '/other', conf)

    cherrypy.config.update({'server.socket_host': '0.0.0.0'})
    cherrypy.config.update({'server.socket_port': 8080})

    cherrypy.engine.start()
    cherrypy.engine.block()
```

Raising HTTP 404 Error

- The firs parameter is the HTTP status code given as integer
- The second optional parameter is a string with your error message

Reference

- http://docs.cherrypy.org/en/latest/
- http://www.w3.org/Protocols/rfc2616/rfc2616-sec10.html