Python Cherry Py tutorial

Tutorial 1: Egy alap webes alkalmazás

A következő példa a legalapabb webes alkalmazás demonstrációja. Elindít egy szervert, ami tulajdonképpen egy olyan alkalmazást hostol ami http requesteket (kéréseket) fog fogadni a http://127.0.0.1:8080/ címen:

```
import cherrypy


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


if __name__ == '__main__':
    cherrypy.quickstart(HelloWorld())
```

Mentsd el a kódot egy **tut01.py** file-ba és futtasd az alkamazást!

Valami hasonlót kell látnod:

```
[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 Serving on http://127.0.0.1:8080
[24/Feb/2014:21:01:46] ENGINE Bus STARTED
```

Tutorial 2: Különböző URL-ek különböző funkcióknak

A következő alkalmazás többre lesz képes, mint egy darab egyszerű URL kiszolgálására. Készítsünk alkalmazást, mely random stringeket generál minden egyes hívásra!

```
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.quickstart(StringGenerator())
```

Mentsd el a kódot egy **tut02.py** file-ba és futtasd az alkamazást!

Látogass most el a http://localhost:8080/generate oldalra!

Álljunk meg egy pillanatra és elemezzük mi történt. Kezdjük magával az URL-lel:

1. http:// →Definiálja, hogy a kapcsolatom a szerver felé http protokollon keresztül történjen
2. localhost→A szerver címe (hostname)
3. :8080→ Annak a portnak a száma, amin keresztül adatot kérek a szervertől
4. /generate →Egyfajta „útvonal” ami egy bizonyos helyre irányítja a kérést. Jelen esetben a „generate” nevezetű függvényünk visszatérési értékéhez

A jelenlegi példában tehát 2 útvonalunk van:

1. index (az index ekvivalens azzal, ha a port után nem írunk semmit)
2. generate

Tutorial 3: Az URL-emben legyenek paraméterek

Az előző példában random stringeket generáltunk. Fejlesszük az alkalmazást úgy, hogy a stringek hossza megadható legyen a felhasználó által, vagyis a hossz egy paraméter.

```
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.quickstart(StringGenerator())
```

Mentsd el a kódot egy **tut03.py** file-ba és futtasd az alkamazást!

Látogassunk el most a http://localhost:8080/generate?length=16 címre. Az látható, hogy a generált string 16 karakter hosszú, mert a címbe azt írtuk, hogy *length=16* (ha átírod, más lesz a hossza).

> Megj.: Ha megnézed a kódot, a *length*-nek van default értéke

Az ilyen URL-ekben a **?** utáni részt *query-string*-nek nevezik (magyarul valami olyasmi lenne, hogy *lekérdező szöveg*). A *query-string*-eknek pont az a lényege, hogy értékeket adjunk át az oldalnak. A formátuma **kulcs=érték**. Megadható több is, ebben az esetben az **&** jel az elválasztó karakter.

A CherryPY ezeket a kulcs érték párokat a függvény paramétereihez használja fel.

Tutorial 4: Űrlap kitöltés

A CherryPy egy webes framework (magyarul talán úgy lehetne mondani, hogy keretrendszer). Általában a webes alkalmazásokat a felhasználók nem így ebben a formában szeretik használni, hanem valamiféle HTML interfészen keresztül.

Készítsünk egy olyat python alkalmazást, melyben a függvény egy olyan stringgel tér vissza, mely egy valid HTML leírás.

```
import random
import string

import cherrypy


class StringGenerator(object):
    @cherrypy.expose
    def index(self):
        return """<html>
          <head></head>
          <body>
            <form method="get" action="generate">
              <input type="text" value="8" name="length" />
              <button type="submit">Give it now!</button>
            </form>
          </body>
        </html>"""

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


if __name__ == '__main__':
```

```
    cherrypy.quickstart(StringGenerator())
```

Mentsd el a kódot egy **tut04.py** file-ba és futtasd az alkamazást!

Ha most ellátogatsz a http://localhost:8080/ címre, akkor egy beviteli mezőt és egy gombot kell látnod. Ha beírod mondjuk a **10**-et a mezőbe és kattintasz a gombra, akkor az URL sávban a cím megváltozik arra, hogy: http://127.0.0.1:8080/generate?length=10

Megj.: Vegyük észre, hogy ez pont olyan, mint amit az előző tutorialban csináltunk.

Tutorial 5: A felhasználó nyomonkövetése

Eléggé gyakori a webes alkalmazásokban, hogy a felhasználó tevékenységét valmilyen módon nyomon kell követni (például ha bejelentkezel egy weboldalra és utána navigálsz tovább a lapok között, a szervernek a „jogokat” ugyan úgy rádvonatkozóan kell tekinteni, azaz a következő oldalon is az előzőek szerint kell eljárni. Képzeljük el, hogy egyszerre többen is meglátogatják a weboldalunkat. Ezesetben mindenkinek más és más adatot kell kiszolgálni. A webes keretrendszerek ezt a fajta „azonosítást” session-nek hívják (fogalmam sincsen, hogy erre van-e magyar szó).

```
import random
import string

import cherrypy


class StringGenerator(object):
    @cherrypy.expose
    def index(self):
        return """<html>
          <head></head>
          <body>
            <form method="get" action="generate">
              <input type="text" value="8" name="length" />
              <button type="submit">Give it now!</button>
            </form>
          </body>
        </html>"""

    @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.quickstart(StringGenerator(), '/', conf)
```

Mentsd el a kódot egy **tut05.py** file-ba és futtasd az alkamazást!

Az alkalmazás hasonlóan működk mint az előző példában. Ha azonban most ellátogatsz a http://localhost:8080/display oldalra, akkor az előzőekben generált szöveget kell látnod.

A 30-34. sorban azt látod, hogy a CherryPy számára „aktiváltuk" a session-ök kezelését. By default (alapból) a session-öket a folyamat memóriájában tárolja a CherryPy.

Tutorial 6: Mi a helyzet a javascriptekkel, CSS-sel és képekkel?

A webes alkalmazások általában statikus tartalmakból épülnek fel, melyek javascript, CSS fileokból, vagy éppek képekből áll. A CherryPy ezeket is támogatja.

Tegyük fel, hogy stílusokkal szeretnék kiegészíteni az alkalmazást, hogy a hátteret kékre állíthassuk.

Először is mentenünk kell a stílus fileunkat (CSS). Legyen a neve **style.css**. Mentsük ezt a **public/css** könyvtárba.

```
body {
  background-color: blue;
}
```

Módosítsuk most a HTML kódot annyiban, hogy a stílust odalinkeljük. Használd a http://localhost:8080/static/css/style.css URL-t.

Mentsd el a kódot egy **tut06.py** file-ba és futtasd az alkamazást!

A CherryPy támogatja egyszerű fileok, vagy teljes mappa struktúrák átadásához. Általában pontosan ezt akarjuk csinálni, mint ahogyan azt a mostani példában is. Először is a **root** mappa an beállítva az összes static tartalomhoz. Ennek **abszolút elérési út**nak kell lennie biztonsági okokból. A CherryPy egyébként nyavalyogni fog, ha **relatív út**vonalat használunk.