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.

Tutorial 7: Hogyan néznének ki a REST kérések?

Először is beszéljünk a REST-ről! Manapság a web API-t és a REST-et együtt emlegetik, pedig a kettő nem ugyan az. Az APi az *Application Programming Interface*, magyarul *alkalmazásprogramozási felület*. Gyakorlatilag az a felület, amin keresztül el lehet érni egy alkalmazás eljárásait, esetleg adatait. Vegyük példának okáért a mi akasztófa projektünket. Azt szeretnénk, hogy egy program egy adott kérésre adjon válaszul egy véletlen (vagy valamilyen eljárással generált) szót, amit majd ki kell találnia a felhasználónak. Ez esetben meg kell írnunk a szógeneráló eljárást, de ezen felül ezt elérhetővé kell tennünk valamiféle API-n, hiszen a megjelenítőnk (egy weboldal) meg szeretné hívni ezt az eljárást bizonyos felhasználói kérésre (mondjuk megnyom egy gombot a weboldalon). Az érdekes a helyzetben, hogy az eljárást nem is egy hús-vér ember intézi például úgy, hogy lefuttatja a python alkalmazást, hanem egy másik program, a megjelenítő alkalmazásunk. Az API-nak tehát a következő feltételeknek kell megfelelnie:

1. Egyszerre több kérést is ki kell szolgálnia, hiszen a megjelenítő több ember böngészőjében futhat egyszerre
2. A megoldásnak „webesnek” kell lennie, azaz a hálózaton elérhetőnek kell lennie, hiszen a programunk egy szerveren fut, nem akarjuk minden egyes felhasználónak odaadni a programot
3. A megoldásnak olyannak kell lennie, amit egy másik program könnyen tud intézni sőt, csak program fogja hívni (a felhasználó közvetlen nem kapja meg a szót, csak a weblapon keresztül amit meglátogat).

Amit fentebb összeszedtünk gyakorlatilag minden webes alkalmazás problémája: Van egy kliens oldali alkalmazás, ami a felhasználó böngészőjében fut (frontend), és adatokat kér egy központi alkalmazástól (backend).

A leggyakoribb megoldás a REST API.

Elsőként lássuk a gyakorlatban egy rest apit:

1. A következő REST API-n elérhető alkalmazás véletlenszerű macska tényeket ad vissza: https://catfact.ninja/fact
2. Ez a REST api nemzetiséget talál ki névből: https://api.genderize.io/?name=luc

A példákból az látszik, hogy a REST majdnem pont olyan mint a http: URL-ek vannak, a szokásos struktúrával (protokol, szervernév, elérési út, http query).

Nem mennék bele a részletekbe annyira, mert nem olyan fontos most. A szemléletes lényeg a használat módjában van. A http-t weboldalakra használják, a REST-et, pedig adatkiszolgálásra. A másik fontos különbség az eljárások, amikkel el lehet érni.

Mind a http, mind a REST eljárásokkal működik. Ez azt jelenti, hogy amikor intézel egy kérést, akkor először is meg kell mondanod, a kérésed milyen „igényt” takar. Ha rákattintasz egy linkre (pl. a fentiekre), akkor ezzel azért nem találkozol, mert a böngésző automatikusan kérést, azaaz GET-et kér. A http eljárásai: GET, HEAD, POST, PUT, DELETE, CONNECT, OPTIONS, TRACE, PATCH. A RESTé: POST, GET, PUT, PATCH.

Mit jelent ez a gyakorlatban? Hát azt, hogy amikor megcsinálod a REST szervízed, akkor 1 adott végpontra implementálhatsz különböző dolgokat a különböző eljárásokhoz. Lássunk egy példát!

```
import random
import string

import cherrypy


@cherrypy.expose
class StringGeneratorWebService(object):

    @cherrypy.tools.accept(media='text/plain')
    def GET(self):
        return cherrypy.session['mystring']

    def POST(self, length=8):
```

```
        some_string = ''.join(random.sample(string.hexdigits, int(length)))
        cherrypy.session['mystring'] = some_string
        return some_string

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

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


if __name__ == '__main__':
    conf = {
        '/': {
            'request.dispatch': cherrypy.dispatch.MethodDispatcher(),
            'tools.sessions.on': True,
            'tools.response_headers.on': True,
            'tools.response_headers.headers': [('Content-Type',
'text/plain')],
        }
    }
    cherrypy.quickstart(StringGeneratorWebService(), '/', conf)
```

Mentsd el a kódot egy **tut07.py** file-ba és futtasd az alkamazást!

Mielőtt lefuttatnánk, értsük meg a kódot!

Eddig úgy csináltunk, hogy a *cherrpy.expose* dekorátort az eljárásokra raktuk, így a végpont neve ugyan az volt, mint magának az eljárásnak. Most azonban több eljárás tartozik 1 végponthoz, hiszen más-mást szeretnénk csinálni a különböző rest eljárásokra. Éppen ezért a dekorátor az osztályon van. Azért, hogy a végpont megtalálja az eljárást, a 27. sor felelős.

További érdekesség, hogy megkötöttük, hogy a válasz *content-type text/plain* vagyis egyszerű szöveg legyen, ezzel biztosítva, hogy csak olyan indíthasson GET kérést, aki el is tud fogadni sima szöveget.

Ezeket az eljárásokat már nem lehet böngészővel csinálni, ezért szükségünk lesz más programra. Rengeteg ilyen van. Parancssoriból a *curl* a legelterjedtebb, grafikusok közül pedig talán a *Postman*.

Mi most python scriptet fogunk írni, hogy teszteljük. Ehhez először is telepíteni kell a *requests* könyvtárat

```
pip install requests
```

Mostmár ki lehet próbálni:

```
>>> import requests
>>> s = requests.Session()
>>> r = s.get('http://127.0.0.1:8080/')
>>> r.status_code
500
>>> r = s.post('http://127.0.0.1:8080/')
```

```
>>> r.status_code, r.text
(200, u'04A92138')
>>> r = s.get('http://127.0.0.1:8080/')
>>> r.status_code, r.text
(200, u'04A92138')
>>> r = s.get('http://127.0.0.1:8080/', headers={'Accept':
'application/json'})
>>> r.status_code
406
>>> r = s.put('http://127.0.0.1:8080/', params={'another_string': 'hello'})
>>> r = s.get('http://127.0.0.1:8080/')
>>> r.status_code, r.text
(200, u'hello')
>>> r = s.delete('http://127.0.0.1:8080/')
>>> r = s.get('http://127.0.0.1:8080/')
>>> r.status_code
500
```

Az első és az utolsó kérés azért tért vissza 500-as response kóddal, mert nem volt szóveg tárolva az adott pillanatban (amit a posttal csináltunk, de a delete-tel kitöröltünk)

A 12-14. sor azt mutatja, hogy mi van ha nem sima szövegként (plain/test) kérjük. 406-os a hibakód, vagyis nem sikerült (a json-ról majd később).

Megj.: A kérésben a *Session* interfész van használva, ami gondoskodik a session id tárolásáról a request sütiben.

Tutorial 8: Finomítsuk Ajax-val

Most koncentráljunk egy picit a megjelenésre. Ez ugye HTML dokumentum. Van ezzel egy kis gond, méghozzá az, hogy ez egy **dokumentum**. Képzeljük el a következő helyzetet: meglátogatunk egy oldalt, amibe mondjuk be van építve egy like gomb. Ha rákattintasz nő a szám egyel. Ha ez szimpla HTML-lel van megvalósítva, akkor a HTML dokumentum változik, hiszen a szám változott, vagyik a like-ra kattintva az egész oldalnak újra kell töltődnie, hiszen egy új dokumentum „készült”. Másik példa, ha google-n keresel valamit, akkor gépelés közben lejön egy ablak, amiben „ajánlások” vannak, méghozzá a google ajánlásával (kb. keresési gyakoriság, de nem tudom pontosan). Honnan került az oda? Hát a HTML-ben nem lehet benne ami épp be van nálam töltve, hiszen menet közben gépelek, amire folyamatosan kapok adatot szervertől és az megjelenik.

Egy szó mint száz, nem akarjuk a megjelenésnél mindig újratölteni az egész oldalt, azt szeretnénk, hogy dinamikusan változhassanak bizonyos része (más része nem. például ha van egy kép, akkor az maradhat ott).

Miért fontos ez most nekünk? Hát azért, mert a kliens oldalunk dinamikusan kell majd, hogy kommunikáljon a backenddel.

Először csináljuk meg a CSS stílusokat mondjuk egy css mappában **style.css** néven.

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

```
#the-string {
  display: none;
}
```

Az lesz a célunk, hogy megmutassuk a generált szöveget (alapból ne).Készítsük el a következő HTML kódot is:

```
<!DOCTYPE html>
<html>
  <head>
    <link href="/static/css/style.css" rel="stylesheet">
    <script src="http://code.jquery.com/jquery-2.0.3.min.js"></script>
    <script type="text/javascript">
      $(document).ready(function() {

        $("#generate-string").click(function(e) {
          $.post("/generator", {"length": $("input[name='length']").val()})
           .done(function(string) {
            $("#the-string").show();
            $("#the-string input").val(string);
          });
          e.preventDefault();
        });

        $("#replace-string").click(function(e) {
          $.ajax({
            type: "PUT",
            url: "/generator",
            data: {"another_string": $("#the-string input").val()}
          })
          .done(function() {
            alert("Replaced!");
          });
          e.preventDefault();
        });

        $("#delete-string").click(function(e) {
          $.ajax({
            type: "DELETE",
            url: "/generator"
          })
          .done(function() {
            $("#the-string").hide();
          });
          e.preventDefault();
```

```
        });

      });
    </script>
  </head>
  <body>
    <input type="text" value="8" name="length"/>
    <button id="generate-string">Give it now!</button>
    <div id="the-string">
      <input type="text" />
      <button id="replace-string">Replace</button>
      <button id="delete-string">Delete it</button>
    </div>
  </body>
</html>
```

A kódban használatban van továbbá jquery is, hogy egyszerüsítsük az életünket. Az oldal egy egyszerű felhasználói felületet generál és kéréseket intéz a backend szerviz felé.

Végül a python kód maga, ami tartalmazza a szó generálást, de az index.html kiexpozálását is.

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

import cherrypy


class StringGenerator(object):
    @cherrypy.expose
    def index(self):
        return open('index.html')


@cherrypy.expose
class StringGeneratorWebService(object):

    @cherrypy.tools.accept(media='text/plain')
    def GET(self):
        return cherrypy.session['mystring']

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

    def PUT(self, another_string):
```

```
        cherrypy.session['mystring'] = another_string

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


if __name__ == '__main__':
    conf = {
        '/': {
            'tools.sessions.on': True,
            'tools.staticdir.root': os.path.abspath(os.getcwd())
        },
        '/generator': {
            'request.dispatch': cherrypy.dispatch.MethodDispatcher(),
            'tools.response_headers.on': True,
            'tools.response_headers.headers': [('Content-Type',
'text/plain')],
        },
        '/static': {
            'tools.staticdir.on': True,
            'tools.staticdir.dir': './public'
        }
    }
    webapp = StringGenerator()
    webapp.generator = StringGeneratorWebService()
    cherrypy.quickstart(webapp, '/', conf)
```

Mentsd el a kódot egy **tut08.py** file-ba és futtasd az alkamazást!

Látogass el a *http://localhost:8080/* oldalra és játssz egy picit.