ESERCIZIO 1

```
import cherrypy
import json

class Measurement(object):
    exposed = True
    def GET(self, *uri, **params):
        try:
            value = int(params["value"])
        except ValueError:
            raise cherrypy.HTTPError(400, "Valore non valido");

        if params["originalUnit"] == "C" or params["originalUnit"] == "K" or params["originalUnit"] == "F":
            originalUnit = params["originalUnit"]
        else:
           raise cherrypy.HTTPError(400, "OriginalUnit non valido");

        if params["targetUnit"] == "C" or params["targetUnit"] == "K" or params["targetUnit"] == "F":
            targetUnit = params["targetUnit"]
        else:
            raise cherrypy.HTTPError(400, "TargetUnit non valido");


        if str(uri[0]) != "converter":
            raise cherrypy.HTTPError(404, "Parametro errato");


        if originalUnit == "C" and targetUnit == "K":#ok
            new_value = value + 273

        elif originalUnit == "C" and targetUnit == "F":
            new_value = (value * 9/5) + 32

        elif originalUnit == "K" and targetUnit == "C":#ok
            new_value = value - 273

        elif originalUnit == "K" and targetUnit == "F":#ok
            new_value = (value - 273,15) * 9/5 + 32

        elif originalUnit == "F" and targetUnit == "C":
            new_value = (value - 32)*5/9

        elif originalUnit == "F" and targetUnit == "K":#ok
            new_value = (value - 32) * 5/9 + 273,15

        else: #valore in ingresso uguale a quello in uscita
              #perchè originalUnit = targetUnit
            new value = value

        dict_output = {
            'Value' : value,
            'OriginalUnit' : originalUnit,
            'AfterConversion' : new_value,
            'FinalUnit' : targetUnit
            }

        j = json.dumps(dict_output)
        return j


if __name__=="__main__":
#Standard configuration to serve the url "localhost:8080"
    conf={
        '/':{
                'request.dispatch':cherrypy.dispatch.MethodDispatcher(),
                'tool.session.on':True
                }
        }
    cherrypy.tree.mount(Measurement(),'/',conf)
    cherrypy.config.update({'server.socket_port': 8080})
    cherrypy.engine.start()
    cherrypy.engine.block()
```

Il programma espone un servizio Web REST per convertire un valore di temperatura in un'unità di misura differente. Questo tipo di servizio è un modo per fornire una interoperabilità tra vari computer tramite un approccio client-server.
Il codice, quindi, riporta una funzione HTTP GET con, in input, tre parametri: il valore da convertire, l'unità di misura originale e l'unità di misura finale. Questi parametri sono riportati nella URL e memorizzati nel dizionario params; la URL, inoltre, riporta nel path un comando memorizzato nella lista URI.
La GET, quindi, estrae i parametri dal dizionario, li elabora secondo le specifiche e riporta in output un JSON contenente i parametri precedenti insieme al parametro convertito.
Il *main* permette di rendere disponibile il servizio Web e di farlo funzionare.

Considerata la richiesta di ottenere un file JSON in output, abbiamo optato per usare la funzione *dumps()* della libreria JSON, che permette di convertire il dizionario *dict_output*, contenente anche il valore convertito, in una stringa JSON.

ESERCIZIO 2

```
import cherrypy
import json

class Measurement(object):
    exposed = True
    def GET(self, *uri):
        try:
            value = int(uri[1])
        except ValueError:
            raise cherrypy.HTTPError(400, "Valore non valido");

        if uri[2] == "C" or uri[2] == "K" or uri[2] == "F":
            originalUnit = uri[2]
        else:
            raise cherrypy.HTTPError(400, "OriginalUnit non valido");

        if uri[3] == "C" or uri[3] == "K" or uri[3] == "F":
            targetUnit = uri[3]
        else:
            raise cherrypy.HTTPError(400, "TargetUnit non valido");

        if str(uri[0]) != "converter":
            raise cherrypy.HTTPError(404, "Parametro errato");
```

L'esercizio vuole riproporre l'implementazione dello stesso servizio web fornito dall'esercizio precedente, ma con una leggera variazione sul tema: il valore da convertire, l'unità di misura di partenza e quella finale, cioè tutti i dati in input, sono passati al servizio web tramite il path dell'URL (separati da slash), e non attraverso la query. Ad essi si aggiunge la stringa "converter", primo parametro del path, presente anche nell'esercizio precedente. Questa diversa modalità di invio di parametri in input al servizio web sarà mappata nella lista *uri, dove troveremo in ordine i valori inviati nel path. Facciamo allora i dovuti controlli e otteniamo i dati che ci servono, implementando il convertitore non diversamente dall'esercizio precedente.

ESERCIZIO 3

Vogliamo sempre realizzare un convertitore di valori di temperatura, da un'unità di misura di partenza a una target. Ma questa volta vogliamo convertire non un solo valore, ma un' intera lista; per farlo, non inviamo più i valori in input tramite l'URL, perché risulterebbe troppo carico e, come sappiamo, l'URL ha una lunghezza massima prefissata, non è adatto all'invio di una quantità considerevole di dati. Allora la lista di valori di temperatura da convertire, l'unità di misura iniziale e quella finale vengono passati nel body di una richiesta PUT, come file JSON. Per ottenere il JSON usiamo la funzione *cherrypy.request.body.read()* e la inseriamo in una *json.loads()* per ottenere un dizionario che possiamo manipolare con facilità. Facciamo, come sempre, le opportune verifiche lanciando codici di errore in caso di valori in input errati.

Costruiamo allora un nostro vettore di valori di temperatura ottenuto convertendo ad uno ad uno i valori ricevuti nella lista (che è il valore associato alla chiave "values" del dizionario), tenendo conto dell'unità di misura desiderata e quella ricevuta, usando le formule di conversione già viste negli esercizi precedenti. Il vettore sarà il valore associato alla chiave "AfterConversion" del nostro dizionario, insieme alle altre chiavi con i valori richiesti dal testo, e il dizionario sarà ritornato dalla PUT come file JSON.

```
import cherrypy
import json

class Measurement(object):
    exposed = True
    def PUT(self, **params):
        self.content = json.loads(cherrypy.request.body.read())
        value = self.content["values"]

        if self.content["originalUnit"] == "C" or self.content["originalUnit"] == "K" or self.content["originalUnit"] == "F":
            originalUnit = self.content["originalUnit"]
        else:
            raise cherrypy.HTTPError(400, "OriginalUnit non valido");

        if self.content["targetUnit"] == "C" or self.content["targetUnit"] == "K" or self.content["targetUnit"] == "F":
            targetUnit = self.content["targetUnit"]
        else:
            raise cherrypy.HTTPError(400, "TargetUnit non valido");

        new_values = []
```

ESERCIZIO 4.

```
import cherrypy
import os
import json

class Freeboard():

    exposed = True
    def GET(self, *uri, **params):
        return open("index.html","r").read()

    def POST(self, *uri, **params):
        if uri[0] == "saveDashboard":
            with open("dashboard/dashboard.json", "w") as outfile:
                outfile.write(params['json_string'])


if __name__ == "__main__":
    conf={

        "/":{
                'request.dispatch': cherrypy.dispatch.MethodDispatcher(),
                'tools.sessions.on': True,
                'tools.staticdir.root': os.getcwd()
        },
        "/css":{
                'tools.staticdir.on':True,
                'tools.staticdir.dir':"css"
        },
        "/js":{
                'tools.staticdir.on':True,
                'tools.staticdir.dir':"js"
        },
        "/img":{
                'tools.staticdir.on':True,
                'tools.staticdir.dir':"img"
        },
        "/plugins":{
                'tools.staticdir.on':True,
                'tools.staticdir.dir':"plugins"
        },
        "/dashboard":{
                'tools.staticdir.on':True,
                'tools.staticdir.dir':"dashboard"
        }
}
```

```
this.saveDashboard=function(){freeboard.showDialog("Would you like to replace default dashboard with the current one?","Set Default","Replace","Cancel",
    function(){$.post("saveDashboard", { json_string:JSON.stringify(d.serialize())})})},

this.setDefault=function(){freeboard.showDialog("Would you like to replace this dashboard with the default one?","Set Default","Replace","Cancel",
    function(){$.post("saveDashboard", { json_string: "{\"version\":1,\"allow_edit\":true,\"plugins\":[],\"panes\":[],\"datasources\":[],\"columns\":3}"})})},
```

L'obiettivo di questo esercizio consiste nello sviluppare un servizio Web REST per implementare una freeboard, ovvero un software open-source, utile per la costruzione di dashboards interattive e real-time.
La funzione GET permette di fornire la pagina index.html sfruttando il metodo *read()*, per leggere il contenuto del file.
La funzione POST salva la nuova configurazione della dashboard, contenuta nel valore del dizionario params, avente come chiave "json_string", e, attraverso il metodo *write()*, memorizzarla nel file dashboard.json.
Le stringhe "json_string" e "saveDashboard", usate nella POST, sono state ricavate analizzando il file .js della freeboard riguardo la funzione "saveDashboard" e "setDefault" (in figura sono riportate le due funzioni analizzate a tale scopo).

Configurazione nel main: dal momento che le applicazioni Web sono anche costituite da file javascript, .css, immagini e altre risorse, il dizionario conf, oltre ad indicare il direttorio root, indica anche i path di questi contenuti statici. In questo modo, il nostro browser si aspetta di trovare i file .css nella cartella chiamata "css", i file javascript nella cartella chiamata "javascript" e via dicendo. Inoltre, il main permette di rendere disponibile il servizio Web e di far partire il suo funzionamento.