

---

# SISTEMAS WEB

## CURSO 2023/2024

### HTTP - HyperText Transfer Protocol

Solicitud y respuesta. Ejemplo utilizando Python: librería *requests*

Códigos de Error: 400 y 404.

Redirecciones (códigos 3XX)

Envío de datos en formato formulario



Web Sistemak by [Oskar Casquero](#) & [María Luz Álvarez](#) is licensed under a [Creative Commons Reconocimiento 4.0 Internacional License](#).

# EJEMPLO UTILIZANDO PYTHON : LIBRERÍA *REQUESTS*

---

- Utilizando la librería *requests* de Python solicita el siguiente recurso:

<http://www.httpwatch.com/httpgallery/chunked/chunkedimage.aspx>

```
import requests

# La petición tiene 4 partes: metodo, uri, cabecera y cuerpo
metodo = 'GET'
uri = "http://www.httpwatch.com/httpgallery/chunked/chunkedimage.aspx"
cabeceras= {'Host': 'www.httpwatch.com'}
cuerpo= ''
respuesta= requests.request(metodo, uri, headers=cabeceras, data=cuerpo)

# La respuesta tiene también 4 apartados: codigo, descripción, cabeceras eta
cuerpo
codigo= respuesta.status_code
descripcion= respuesta.reason
print(str(codigo) + " " + descripcion)
for cabecera in respuesta.headers:
    print(cabecera + ": " + respuesta.headers[cabecera])
cuerpo= respuesta.content
print(cuerpo)

fichero= open("imagen.jpg", 'wb')
fichero.write(cuerpo)
fichero.close()
```

Documentación de requests:  
<https://pypi.org/project/requests/>

# FUNCIONAMIENTO DE HTTP:

## ERRORES

---

- ¿Qué ocurre si una petición no puede ser satisfecha?

### Códigos de respuesta 4xx y 5xx.

- Cuando el cliente o el servidor no puede completar una solicitud, se devuelven los siguientes códigos de estado (STATUS) en la respuesta.

Errores provocados por el **cliente**:

**4xx** ([RFC 2616, Sección 10.4](#))

- 400 Bad Request
- 401 Unauthorized
- 403 Forbidden
- 404 Not Found
- ...

Errores provocados por el **servidor**:

**5xx** ([RFC 2616, Sección 10.5](#)):

- 500 Internal Server Error
- 503 Service Unavailable
- ...

# FUNCIONAMIENTO DE HTTP

## ERROR 400

---

- **400 Bad Request**

- El servidor web no entiende la petición, su sintaxis/semántica es incorrecta.
- Supongamos que se solicita el recurso */html/main page.html*

### Petición de ejemplo

```
GET /html/main page.html HTTP/1.1
Host: sw2024.com
Accept: text/html
Accept-Encoding: gzip,identity;q=0.5
Accept-Language: en-US,es-ES;q=0.8
User-Agent: Mozilla Windows Escritorio
```

### Respuesta de ejemplo

```
HTTP/1.1 400 Bad Request
Date: Wed, 25 Nov 2015 08:07:43 GMT
Content-Length: 138
Content-Type: text/html; charset=UTF-8

<html><head><title>Error 400 (Bad
Request)</title></head><body><p>Your
client has issued a malformed or illegal
request.</p></body></html>
```

Lo que interpreta el servidor al procesar la petición:

- Método: GET
- RequestURI: /html/main
- Versión del protocolo: page.html -> SEMÁNTICA ERRONEA (La cadena debe ser HTTP/1.1)
- Cadena adicional: HTTP/1.1 -> SINTAXIS ERROEA (la primera línea tiene que tener 3 elementos)

**Solución: Codificar el espacio en la RequestURI: /html/main%20page.html**

# EJEMPLO: ERROR 400 (SEMÁNTICA ERRÓNEA)

- En el código del ejemplo anterior, añadir al final del nombre de la cabecera Host un espacio.

```
import requests

# La petición tiene 4 partes: metodo, uri, cabecera y cuerpo
metodo = 'GET'
uri = "http://www.httpwatch.com/httpgallery/chunked/chunkedimage.aspx"
cabeceras= {'Host ': 'www.httpwatch.com'}
cuerpo= ''
respuesta= requests.request(metodo, uri, headers=cabeceras, data=cuerpo)

# La respuesta tiene también 4 apartados: codigo, descripción, cabeceras eta
cuerpo
codigo= respuesta.status_code
descripcion= respuesta.reason
print(str(codigo) + " " + descripcion)
for cabecera in respuesta.headers:
    print(cabecera + ": " + respuesta.headers[cabecera])
cuerpo= respuesta.content
print(cuerpo)

fichero= open("imagen.jpg", 'wb')
fichero.write(cuerpo)
fichero.close()
```

# FUNCIONAMIENTO DE HTTP

## ERROR 404

---

- **404 Not Found**

- La petición esta bien formada, pero el servidor web **no encuentra el recurso**.
- Supongamos que se solicita al servidor sw2024.com el recurso */login/index.php*, que no existe en el servidor

### Petición de ejemplo

```
GET /login/index.php HTTP/1.1
Host: sw2024.com
Accept: text/html
Accept-Encoding: gzip,identity;q=0.5
Accept-Language: en-US,es-ES;q=0.8
User-Agent: Mozilla Windows Escritorio
```

### Respuesta de ejemplo

```
HTTP/1.1 404 Not Found
Date: Wed, 25 Nov 2015 08:07:43 GMT
Content-Length: 134
Content-Type: text/html; charset=UTF-8

<html><head><title>Error 404 (Not
Found)</title></head><body><p>The
requested resource was not found in this
server.</p></body></html>
```

# EJEMPLO: ERROR 404

- En el código del ejemplo anterior, cambiar el nombre del recurso:

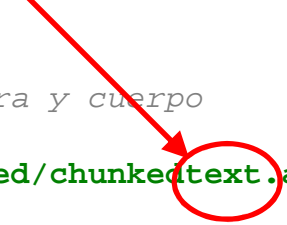
/httpgallery/chunked/chunkedtext.aspx

```
import requests

# La petición tiene 4 partes: metodo, uri, cabecera y cuerpo
metodo = 'GET'
uri = "http://www.httpwatch.com/httpgallery/chunked/chunkedtext.aspx"
cabeceras= {'Host': 'www.httpwatch.com'}
cuerpo= ''
respuesta= requests.request(metodo, uri, headers=cabeceras, data=cuerpo)

# La respuesta tiene también 4 apartados: codigo, descripción, cabeceras eta
cuerpo
codigo= respuesta.status_code
descripcion= respuesta.reason
print(str(codigo) + " " + descripcion)
for cabecera in respuesta.headers:
    print(cabecera + ": " + respuesta.headers[cabecera])
cuerpo= respuesta.content
print(cuerpo)

fichero= open("imagen.jpg", 'wb')
fichero.write(cuerpo)
fichero.close()
```



# FUNCIONAMIENTO DE HTTP:

## REDIRECCIONES

---

- ¿Qué son las redirecciones?
  - Algunas veces, el URI de un recurso puede cambiarse o adecuarse. Por ejemplo, en el año académico 2017-18, el URI <http://egela.ehu.eus/> fue dirigido al URI <http://egela2017-18.ehu.eus/>; en cambio, durante el año académico 2018-19, se dirigió al URI <http://egela2018-19.ehu.eus/>.
  - En ambos casos, el usuario no necesitaba conocer el URI real, porque al solicitar <http://egela.ehu.eus/> el navegador redirige automáticamente al usuario al URI.
- ¿Cómo se hacen las redirecciones?
  - El protocolo HTTP ofrece una forma de redirigir a otro URI. Para realizar las redirecciones utiliza:
    - Los códigos de respuesta 301, 302 o 303
    - y el encabezado “*Location*”.



# FUNCIONAMIENTO DE HTTP:

## REDIRECCIONES

---

- Supongamos que un cliente ubicado en Madrid solicita un recurso cuya URI es: <http://www.google.com/>
- El servidor web que atiende la petición HTTP detecta que dirección IP de origen del paquete es de España, por lo que devuelve una respuesta al cliente indicándole que realice una redirección a <http://www.google.es/>
- El cliente, al detectar un código de respuesta 302, extrae el contenido de la cabecera “**Location**” y realiza una nueva petición a esta URI. En un navegador, este proceso ocurre de forma transparente para el usuario.

### Ejemplo de Solicitud

```
GET / HTTP/1.1
Host: www.google.com
Accept: text/html
Accept-Encoding: identity
Accept-Language: en-US,es-ES;q=0.8
User-Agent: Mozilla Windows Escritorio
```

### Ejemplo de Respuesta

```
HTTP/1.1 302 Found
Content-Length: 137
Content-Type: text/html; charset=UTF-8
Location: http://www.google.es/

<html><head><title>Redirection
302</title></head><body><a
href="http://www.google.es/>Redirect to
http://www.google.es/</a></body></html>
```

# FUNCIONAMIENTO DE HTTP:

## REDIRECCIONES

---

- <https://tools.ietf.org/html/rfc7231#section-6.4>
  - **301 Moved Permanently:** Este código de respuesta significa que el URI del recurso solicitado ha sido cambiado. Probablemente la nueva URI sea devuelta en la respuesta.
  - **302 Found:** Este código de respuesta significa que el recurso de la URI solicitada ha sido cambiado temporalmente. En el futuro se realizarán nuevos cambios en la URI. Por lo tanto, la misma URI debe ser usada por el cliente en futuras solicitudes.
  - **303 See Other:** El servidor envía esta respuesta para dirigir al cliente a un recurso diferente, indicado en la cabecera. El cliente deberá solicitar el nuevo recurso.

# EJEMPLO USANDO LAS HERRAMIENTAS DE DESARROLLO DEL NAVEGADOR

- Abre una nueva pestaña en su navegador, solicita el recurso <http://egela.ehu.eus/> usando las herramientas de desarrollador (F12).

The screenshot displays a web browser window with the URL [egela.ehu.eus](http://egela.ehu.eus/). The website content includes a header with language options (Deutsch, English, Español, Euskara, Français, Italiano) and the UPV EHU logo. The main body features a section for 'Gradu, Master Oficial, Berezko Titulu eta Doktoregorako irakasgaiantzeko plataforma' and a 'Blackboard collaborate' advertisement. Below this, there are three featured articles: 'eGela 18/19 sarbidea (aurreko ikasturtea)', 'Ikasgela Birtualen Kudeatzailea', and 'Erabiltzailearentzako Arreta Zerbitzua'.

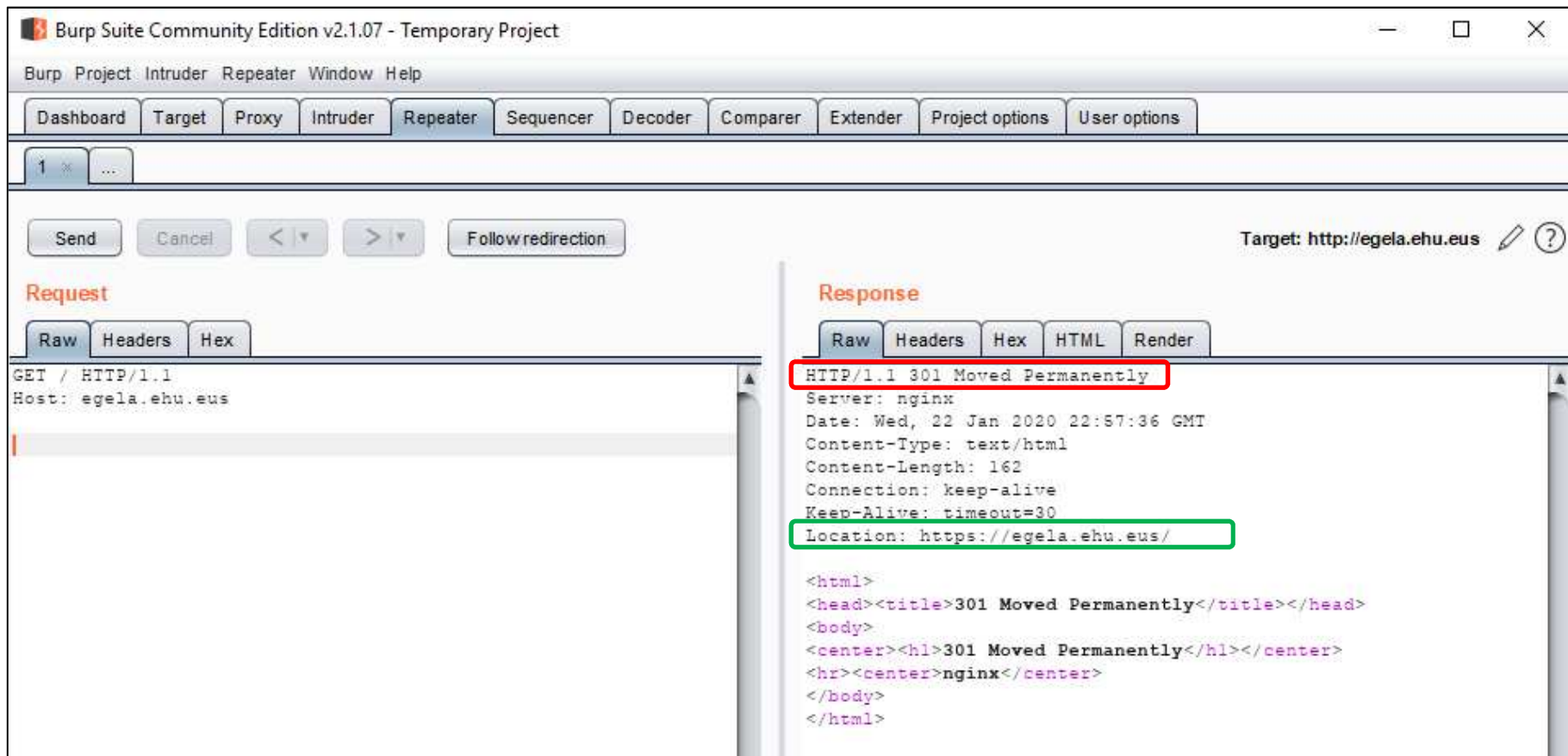
Overlaid on the right side of the browser window is the Chrome DevTools 'Network' tab. It shows a list of network requests. The first two requests, 'egela.ehu.eus' (Status 301) and 'egela.ehu.eus' (Status 200), are highlighted with a red box. The table below shows the details of these and other requests:

Name	Status	Type	Initiator	Size	Time	Waterfall
egela.ehu.eus	301	text/html	Other	214 B	82 ms	
egela.ehu.eus	200	document	egela.ehu.eus/	12.9 KB	329 ms	
yui_combo.php?rollup/3.17.2/yui-moodie-simple-min.css	200	stylesheet	(index)	1.5 KB	29 ms	
all	200	stylesheet	(index)	132 KB	360 ms	
yui_combo.php?rollup/3.17.2/yui-moodie-simple-min.js	200	script	(index)	82.9 KB	363 ms	
jquery-3.2.1.min.js	200	script	(index)	30.0 KB	123 ms	
javascript-static.js	200	script	(index)	7.1 KB	89 ms	
ResizeSensor.js	200	script	(index)	8.8 KB	87 ms	
ElementQueries.js	200	script	(index)	19.9 KB	113 ms	
js?id=UA-98997857-3	200	script	(index)	27.6 KB	127 ms	
logoECAMPUS.png	200	png	(index)	1.8 KB	28 ms	
require.min.js	200	script	(index)	6.9 KB	28 ms	
scrolltop.js	200	script	(index)	768 B	27 ms	
tooltipfix.js	200	script	(index)	658 B	29 ms	
header_egela.svg	200	svg+xml	(index)	478 KB	557 ms	
egela1819.png	200	png	(index)	157 KB	845 ms	
GAV.png	200	png	(index)	219 KB	258 ms	
caup.png	200	png	(index)	237 KB	732 ms	
yui_combo.php?rollup/3.17.2/cssbutton/cssbutton-min.css	200	stylesheet	yui_combo.php?rollup/3.1...	1.6 KB	27 ms	
fontawesome-webfont.woff2?v=4.7.0	200	font	(index)	75.8 KB	105 ms	
first.js	200	script	require.min.js	260 KB	357 ms	
data:image/svg+xml...	200	svg+xml	jquery-3.2.1.min.js	(memory cache)	0 ms	
data:image/svg+xml...	200	svg+xml	jquery-3.2.1.min.js	(memory cache)	0 ms	
analytics.js	200	script	js?id=UA-98997857-3	17.8 KB	80 ms	
EGelaportfolio.png	200	png	jquery-3.2.1.min.js	82.2 KB	184 ms	
collect?v=1&vjs=79&a=2021447936&t=pageview&s=1&d...	302	text/html	analytics.js	301 B	46 ms	
collect?v=1&aip=1&t=dc&r=3&tid=UA-98997857-3&cid=...	200	gif	collect	407 B	123 ms	
Blackboard-collaborate.jpg	200	jpeg	jquery-3.2.1.min.js	35.6 KB	72 ms	
yui_combo.php?m/1579685327/core/event/event-min.js...	200	script	yui_combo.php?rollup/3.1...	6.7 KB	27 ms	
yui_combo.php?m/1579685327/filter_mathjaxloader/loader...	200	script	yui_combo.php?rollup/3.1...	1.1 KB	26 ms	
event.js	200	script	require.min.js	260 KB	247 ms	
jquery-3.2.1.min.js	200	script	require.min.js	29.9 KB	52 ms	
yui_combo.php?m/1579685327/event-mousewheel/event-mousew...	200	script	yui_combo.php?rollup/3.1...	5.2 KB	27 ms	
service.php?sesskey=nDpEUp36n&info=core_fetch_notifica...	200	xhr	jquery-3.2.1.min.js	577 B	47 ms	

At the bottom of the Network tab, a summary shows: 34 requests, 2.1 MB transferred, 4.7 MB resources, Finish: 1.77 s, DOMContentLoaded: 1.00 s, Load: 1.78 s.

# EJEMPLO UTILIZANDO BURP

- Utilizando la pestaña *Repeater* de Burp, solicita el siguiente recurso:  
<http://egela.ehu.es/>



# EJEMPLO UTILIZANDO PYTHON

- Utilizando la librería *requests* en Python-en, realiza el siguiente ejemplo

```
import requests

metodo = 'GET'
uri = "http://egela.ehu.eus/"
cabeceras = {'Host': 'egela.ehu.eus'}
cuerpo = ''
respuesta = requests.request(metodo, uri, headers=cabeceras, data=cuerpo, allow_redirects=False)

codigo = respuesta.status_code
descripcion = respuesta.reason
print(str(codigo) + " " + descripcion)
for cabecera in respuesta.headers:
    print(cabecera + ": " + respuesta.headers[cabecera])
cuerpo = respuesta.content
print(cuerpo)

metodo = 'GET'
uri = respuesta.headers['Location']
cabeceras = {'Host': uri.split('/')[2]}
cuerpo = ''
respuesta = requests.request(metodo, uri, headers=cabeceras, data=cuerpo, allow_redirects=False)

codigo = respuesta.status_code
descripcion = respuesta.reason
print(str(codigo) + " " + descripcion)
for cabecera in respuesta.headers:
    print(cabecera + ": " + respuesta.headers[cabecera])
cuerpo = respuesta.content
print(cuerpo)
```

**Tener en cuenta que en la nueva URI el host puede cambiar !!**

# FUNCIONAMIENTO DE HTTP: FORMULARIOS

---

- **¿Cómo se envían información desde el cliente al servidor?**
  - Un cliente HTTP puede enviar diferentes tipos de datos a un servidor web:
    - datos en formato binario (ejemplo: imágenes)
    - datos en formato JSON o XML
    - **datos en formato formulario**
- En esta sesión vamos a estudiar el envío de datos en **formato formulario**, es decir, cadenas cortas de datos que se envían como pares de nombre-valor, por ejemplo:
  - Modelo genérico: name1=value1&name2=value2&name3=value3
  - Ejemplo del modelo: dni=12345678N&nombre=marta&apellido=perez

# FUNCIONAMIENTO DE HTTP:

## ENVÍO DE DATOS EN FORMATO FORMULARIO (POST)

---

- Supongamos que en un servidor web reside un recurso que calcula la letra del DNI. Si se envía una solicitud HTTP con el número DNI en un parámetro denominado **dni**, este recurso devuelve una respuesta HTTP que contiene la letra del DNI
- Para enviar el dato en formato formulario, el cliente define la siguiente petición HTTP:
  - Método: **POST**
  - Cabeceras para caracterizar el cuerpo del mensaje:
    - **Content-Type**: application/x-www-form-urlencoded
    - **Content-Length**: la longitud del cuerpo del mensaje
  - Los pares nombre-valor deben codificarse de una manera particular (**codificación UTF-8 + por ciento**) y formatearse (como un nombre = valor y un separador entre pares "&").

### Solicitud de ejemplo

```
POST /recurso HTTP/1.1
Host: sw2020.com
Accept: text/plain
Content-Type: application/x-www-form-urlencoded
Content-Length: 12
User-Agent: Cliente Python

dni=12345678
```

### Respuesta de ejemplo

```
HTTP/1.1 200 OK
Date: Wed, 25 Nov 2015 08:07:43 GMT
Content-Length: 1
Content-Type: text/plain

Z
```

# FUNCIONAMIENTO DE HTTP:

## ENVÍO DE DATOS EN FORMATO FORMULARIO (GET)

---

- En lugar de enviar los pares de valores en el cuerpo del mensaje, se puede enviar al URI utilizando el carácter "?" → *query string*
  - Método: **GET**
  - **No** se deben usar los encabezados Content-Type y Content-Length
    - En *query string* solo puede enviar datos formateados.
    - La estructura de la solicitud HTTP limita la *query string*: los pares nombre- valor se encuentra entre el carácter "?" y la cadena "HTTP / 1.1".
  - La codificación y el formato de los datos se realizan como antes.
  - Desventaja: la cantidad de datos que se pueden enviar está limitada a 1024 octetos.

### Petición de ejemplo

```
GET /recurso?dni=12345678 HTTP/1.1
Host: sw2016.com
Accept: text/plain
User-Agent: Cliente Python
```

### Respuesta de ejemplo

```
HTTP/1.1 200 OK
Date: Wed, 25 Nov 2015 08:07:43 GMT
Content-Length: 1
Content-Type: text/plain

Z
```



# FUNCIONAMIENTO DE HTTP:

## RESUMEN

---

GET /app/servlet/contactUs?name=XXX&email=YYY&message=ZZZ HTTP/1.1

Host: ws2017.com

Accept: text/html

Accept-Language: en-US,en

Connection: keep-alive

POST /app/servlet/contactUs HTTP/1.1

Host: ws2017.com

Accept: text/html

Accept-Language: en-US,en

Connection: keep-alive

Content-Type: application/x-www-form-urlencoded

Content-Length: 30

name=XXX&email=YYY&message=ZZZ

# FUNCIONAMIENTO DE HTTP:

## ENVÍO DE DATOS EN FORMATO FORMULARIO

---

- En el formato de formulario, se especifica para el contenido (las cadenas de nombre y valor de los pares nombre-valor) el siguiente **procedimiento de codificación**:
    - Si aparece un carácter especial (caracteres que son **delimitadores** en el URI o en la cadena de nombre-valor) o un carácter no recogido en el alfabeto [US-ASCII](#) (por ejemplo, un carácter con tilde), éste debe codificarse en dos pasos antes formar la URI:
      1. Codificar el carácter en [UTF-8](#). (valores en hexadecimal)
        - Ejemplo 1: & → 26
        - Ejemplo 2: á → C3 A1
      2. Codificar el resultado anterior en “porcentaje” (percent encoding)
        - Ejemplo 1 : 26 → %26
        - Ejemplo 2 : C3 A1 → %C3 %A1
- NOTA:** En los ejemplos anteriores, solo se codifica un carácter; pero debido al procedimiento de codificación, ¿cuántos caracteres se enviarán?
- Ejemplo 1: %26 → 3 caracteres: %, 2, 6
  - Ejemplo 2: %C3 %A1 → 6 caracteres: %, C, 3, %, A, 1

# FUNCIONAMIENTO DE HTTP:

## ENVÍO DE DATOS EN FORMATO FORMULARIO

---

- Ejemplo:
  - datos a enviar: nombre=Iñaki y apellido=Pérez
  - datos que se codificarán en el cuerpo del mensaje:  
nombre=I%C3%B1aki&apellido=P%C3%A9rez
- Si en la cadena del nombre o del valor de los pares nombre-valor aparece un espacio, éste se codifica con "+":
  - datos a enviar: nombre\_apellidos=Iñaki Pérez
  - datos que se codificarán en el cuerpo del mensaje:  
nombre\_apellidos=I%C3%B1aki+P%C3%A9rez
- Pero si se desea utilizar el "+" como dato en lugar de como separador, éste debe codificarse según lo indicado arriba (primero en UTF-8 y luego en porcentaje):
  - datos a enviar: num1=2, num2=2 y operador=+
  - datos que se codificarán en el cuerpo del mensaje:  
num1=2&num2=2&operato=%2B

# EJEMPLO:

## ENVÍO DE DATOS EN FORMATO FORMULARIO -- BURP

---

Envío de datos a un servidor para obtener la letra del DNI:

<http://gae-sw-2017.appspot.com/>

# FUNCIONAMIENTO DE HTTP: FORMULARIOS (HTML)

---



(\*) image extracted from <http://www.freedback.com/>

```
<html>
  <head>
    <title>Formulario</title>
  </head>
  <body>
    <p>
      <form method="GET" action="/app/servlet/contactUs">
        <b>Name: </b>
        <input type="text" name="name">
        <br/>
        <b>Email: </b>
        <input type="text" name="email">
        <br/>
        <b>Message: </b>
        <input type="text" name="message">
        <br/><br/>
        <input type="submit">
      </form>
    </p>
  </body>
</html>
```

# EJEMPLO:

## ENVÍO DE DATOS EN FORMATO FORMULARIO -- POST

Envío de datos a un servidor para obtener la letra del DNI:

<http://gae-sw-2017.appspot.com/>

Codificación  
de datos

### Petición de Letra DNI POST

**POST** /processForm HTTP/1.1  
Host: gae-sw-2017.appspot.com  
Content-Type : application/x-www-form-urlencoded  
Content-Length:12  
  
dni=12345678

```
import requests
import sys
import urllib

metodo = 'POST'
uri = "http://gae-sw-2017.appspot.com/processForm"
cabeceras = {'Host': 'gae-sw-2017.appspot.com',
             'Content-Type': 'application/x-www-form-urlencoded'}
cuerpo = {'dni': sys.argv[1]}
cuerpo_encoded = urllib.parse.urlencode(cuerpo)
cabeceras['Content-Length'] = str(len(cuerpo_encoded))
respuesta = requests.request(metodo, uri, data=cuerpo_encoded,
                             headers=cabeceras, allow_redirects=False)
```

Generamos la petición

### Respuesta del servidor

HTTP/1.1 200 OK  
Date: Wed, 25 Nov 2015 08:07:43 GMT  
Content-Length: 9  
Server: Google Frontend  
Content-type: text/html; charset=utf-8  
  
12345678Z

```
codigo = respuesta.status_code
descripcion = respuesta.reason
print(str(codigo) + " " + descripcion)
for cabecera in respuesta.headers:
    print(cabecera + ": " + respuesta.headers[cabecera])
cuerpo = respuesta.content
print(cuerpo)
```

Procesamos la respuesta

# EJEMPLO:

## ENVÍO DE DATOS EN FORMATO FORMULARIO -- GET

Envío de datos a un servidor para obtener la letra del DNI:

<http://gae-sw-2017.appspot.com/>

### Petición de Letra DNI GET

GET /processForm?dni=12345678 HTTP/1.1  
Host:gae-sw-2017.appspot.com

### Respuesta del servidor

HTTP/1.1 200 OK  
Date: Wed, 25 Nov 2015 08:07:43 GMT  
Content-Length: 9  
Server: Google Frontend  
Content-type: text/html; charset=utf-8  
  
12345678Z

```
import requests
import urllib
import sys

metodo = 'GET'
uri_base = "http://gae-sw-2017.appspot.com/processForm"
cabeceras = {'Host': 'gae-sw-2017.appspot.com'}
query = {'dni': sys.argv[1]}
query_encoded = urllib.parse.urlencode(query)
uri= uri_base + '?' + query_encoded
respuesta = requests.request(metodo, uri,
                             headers=cabeceras, allow_redirects=False)
```

### Generamos la petición

```
codigo = respuesta.status_code
descripcion = respuesta.reason
print(str(codigo) + " " + descripcion)
for cabecera in respuesta.headers:
    print(cabecera + ": " + respuesta.headers[cabecera])
cuerpo = respuesta.content
print(cuerpo)
```

### Procesamos la respuesta

Datos en la  
URI

# EJERCICIO: CONSULTA EN EL *DIRECTORIO DE EHU* UTILIZANDO BURP Y EN PYTHON

---

- <https://www.ehu.eus/bilatu/buscar/bilatu.php?lang=es>

The screenshot displays a web form titled "BUSCAR" (Search) with the following sections:

- Datos personales** (Personal data):
  - Apellidos/ Nombre (Last name/ Name)
  - Nombre (Name)
  - Primer apellido (First surname)
  - Segundo apellido (Second surname)
  - Teléfono (Phone)
  - Correo electrónico (Email)
- Campus**:
  - ☐ Álava
  - ☐ Bizkaia
  - ☐ Gipuzkoa
- Areas** (Areas):
  - Centro (Center): -- Seleccione una opción --
  - Biblioteca (Library): -- Seleccione una opción --
  - Instituto (Institute): -- Seleccione una opción --
  - Departamento (Department): -- Seleccione una opción --
  - Servicios Centrales (Central Services): -- Seleccione una opción --
  - Rectorado (Rectorate): -- Seleccione una opción --
  - Cargo académico (Academic position): -- Seleccione una opción --

At the bottom of the form, there is a small green text note: "NOTA: Si no se encuentra el resultado de la búsqueda, puede ser que el usuario no exista o que el usuario no esté activo".