# Capítulo 2

## Flask: Archivos - Seguridad

# Archivos

- Método POST:
  - Permite cargar archivos de texto y binarios
- Se puede recibir cargas de archivos de cualquier cliente web compatible con RFC 1867
  - “Form-based File Upload in HTML”
- Se debe asegurar que el formulario tenga el atributo:

```
enctype="multipart/form-data"
```

# Archivos

- Formulario para carga

```
<!DOCTYPE html>
<html>
<body>
   <h1>Ejemplo de upload de archivo</h1>
   <form enctype = "multipart/form-data" action = "archivo.py" method = "post">
      <p>Archivo: <input type = "file" name = "filename" /></p>
      <p><input type = "submit" value = "Enviar archivo" /></p>
   </form>
</body>
</html>
```

# Archivos

- Al recibir archivos, se almacenarán en memoria o en una ubicación temporal
- Se puede acceder a estos archivos usando el atributo "`files`" del "`request`"
  - Todos los archivos que se subieron se mantienen en ese diccionario
  - Se tiene el método "`save()`" que permite almacenar ese archivo en el sistema de archivos del servidor

# Archivos

```python
from flask import request

@app.route('/upload', methods=['GET', 'POST'])
def upload_file():
    if request.method == 'POST':
        f = request.files['the_file']
        f.save('/var/www/uploads/uploaded_file.txt')
    ...
```

- Se puede saber el nombre del archivo en el computador del cliente usando el atributo "`filename`"
  - Nunca debe confiar en ese valor

# Archivos

```python
import os
from flask import Flask, flash, request, redirect, url_for
from werkzeug.utils import secure_filename

UPLOAD_FOLDER = '/path/to/the/uploads'
ALLOWED_EXTENSIONS = {'txt', 'pdf', 'png', 'jpg', 'jpeg', 'gif'}

app = Flask(__name__)
app.config['UPLOAD_FOLDER'] = UPLOAD_FOLDER
```

- En este ejemplo definimos al principio:
  - El directorio en el cual se almacenarán los archivos
  - Conjunto de extensiones válidas para los archivos que se esperan en el servidor

```
def allowed_file(filename):
    return '.' in filename and \
           filename.rsplit('.', 1)[1].lower() in ALLOWED_EXTENSIONS

@app.route('/', methods=['GET', 'POST'])
def upload_file():
    if request.method == 'POST':
        # check if the post request has the file part
        if 'file' not in request.files:
            flash('No file part')
            return redirect(request.url)
        file = request.files['file']
        # If the user does not select a file, the browser submits an
        # empty file without a filename.
        if file.filename == '':
            flash('No selected file')
            return redirect(request.url)
        if file and allowed_file(file.filename):
            filename = secure_filename(file.filename)
            file.save(os.path.join(app.config['UPLOAD_FOLDER'], filename))
            return redirect(url_for('download_file', name=filename))
    return '''
    <!doctype html>
    <title>Upload new File</title>
    <h1>Upload new File</h1>
    <form method=post enctype=multipart/form-data>
      <input type=file name=file>
      <input type=submit value=Upload>
    </form>
    '''
```

# Archivos

- Peticiones GET:
  - Retorna HTML que dibuja formulario para subir archivo
    - Considera incluir último mensaje recordado con función flash
- Peticiones POST:
  - Revisa que se reciba un archivo y que esté entre las extensiones permitidas
    - Almacena en directorio UPLOAD
    - Redirecciona al usuario a URL que permite obtener el contenido del archivo
  - En caso de errores, redirecciona al usuario a URL de inicio

# Archivos

- Se debe tener cuidado con:
  - Tamaño del archivo
    - Comprobar si hay restricciones de tamaño en configuración de servidor web
    - Verificar tamaños máximos o mínimos en la aplicación
  - Tipo de archivo
    - Aplicaciones permiten solo cierto tipo de archivos
      - Imágenes, Documentos, Archivos comprimidos, etc.
    - No confiar en tipo de archivo informado por cliente
    - Nunca confiar en extensión de nombre de archivo

# Archivos

```python
from flask import Flask, Request

app = Flask(__name__)
app.config['MAX_CONTENT_LENGTH'] = 16 * 1000 * 1000
```

- Podemos definir el tamaño máximo que se permite para un upload
  - Definir la llave de configuración MAX_CONTENT_LENGTH
  - En este caso se define un máximo de 16 megabytes
- En caso de superar el tamaño máximo se generará error HTTP “`413 Request Entity Too Large`”

# Archivos

- Tipo de archivo
  - Es necesario revisar el tipo de archivo que se recibe en el servidor
    - NUNCA confiar en la extensión del nombre
  - Podemos usar el módulo `filetype` para determinar el tipo de archivo a partir de "`magic numbers`"
    - https://pypi.org/project/filetype/

```
kind = filetype.guess('tests/fixtures/sample.jpg')
if kind is None:
    print('Cannot guess file type!')
    return

print('File extension: %s' % kind.extension)
print('File MIME type: %s' % kind.mime)
```

# Archivos

- Se debe tener cuidado con:
  - Nombre de archivo:
    - No confiar en nombre enviado por cliente:
      - Puede contener caracteres no soportados
      - Puede intentar sobre-escribir archivos existentes
      - Puede ser de largo no soportado
    - Recomendación: Crear un nombre de forma segura en el servidor:
      - Usar funciones de HASH para generar nombre aleatorio y con baja probabilidad de colisión
      - Nombre enviado por cliente se puede almacenar como texto en una base de datos

# Archivos

- Recomendaciones:
  - Habilitar carga de archivos en servidor web solo si nuestra aplicación lo requiere
  - Definir:
    - Cantidad máxima de archivos que podemos recibir en una petición HTTP
    - Tamaño máximo de cada archivo en petición HTTP y tamaño de toda la llamada HTTP
    - Tiempo máximo de procesamiento de CGI que recibe archivos
    - Memoria máxima que puede usar CGI que recibe archivos
  - Monitorear continuamente espacio disponibles y estadísticas de sistema de archivos

# Archivos

- Recomendaciones
  - Cuidado al borrar archivos que incluyen nombre enviado por cliente
    - ¡Debe validar entrada de datos!
  - Incluya “condiciones del servicio” que el cliente **debe** aceptar para enviar archivos:
    - Considerar archivos con derecho de autor (copyright)
    - Contenido legal dependiendo de territorialidad de la ley aplicable
    - Archivos binarios con fines ilícitos (virus, malware, troyanos, spyware, adware, etc)

# Seguridad

- Consejos
  - Nunca se conecte como super usuario o como el propietario de la base de datos
    - Utilice usuarios personalizados con privilegios muy limitados
    - ¿Es necesario tener permisos de escritura en todas las tablas de la base de datos?
    - ¿Se eliminarán datos de esta aplicación?
      - ¿eliminación física o lógica?
  - Use sentencias preparadas con variables asociadas

# Seguridad

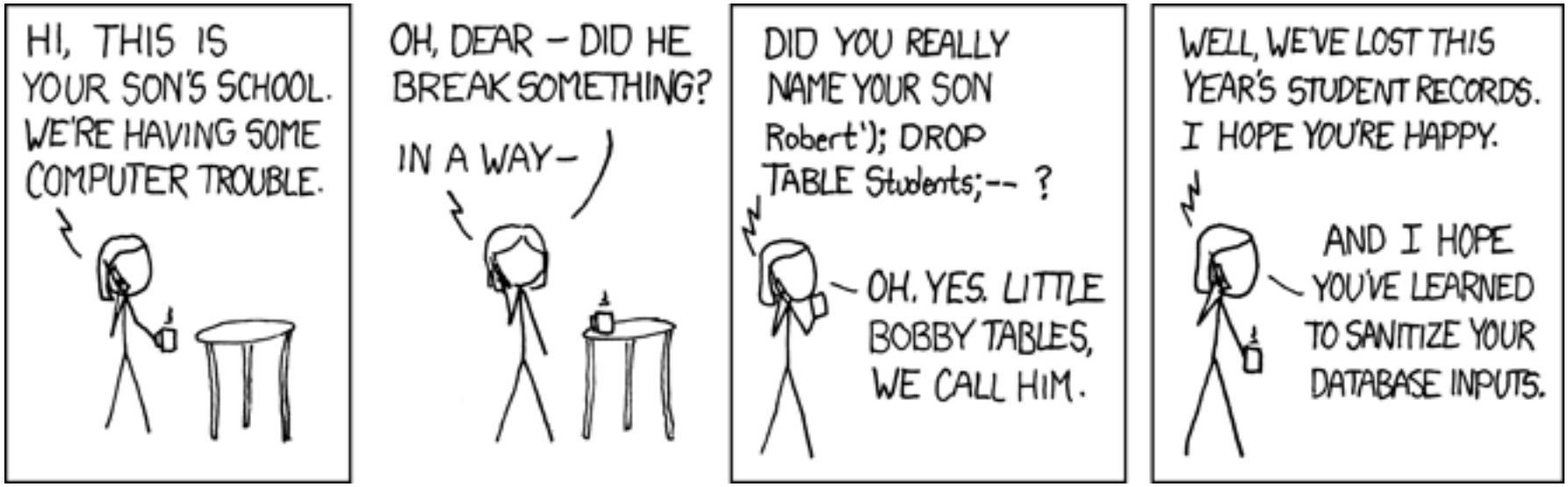


Imagen obtenida desde: http://xkcd.com/327

# Seguridad

- Consejos
  - Revise si la entrada proporcionada tiene el tipo de datos que se espera
    - Python provee varias funciones de validación de tipo de datos
  - Si se espera una entrada numérica:
    - verificar los datos con la función isinstance()

```
marks = 90
result = isinstance(marks, int)
if result :
  print("Yes! given variable is an instance of type int")
else:
  print("No! given variable is not an instance of type int")
```