# Capítulo 2

## Framework Flask: Formularios y Conexión Base de datos

# Datos desde cliente

- Toda aplicación web debe procesar información que el cliente envía al servidor
  - En Flask, esa información la contiene el objeto global `request`
  - Se usa el atributo `method` para saber que tipo de solicitud es (GET, POST, PUT…)
  - Para acceder a información de un formulario transmitido por POST se debe usar el atributo `form`
  - Para acceder a información enviada en la URL de la forma `URL?key=value` se usa el atributo `args`

```
searchword = request.args.get('key', '')
```

# Datos desde cliente

```python
@app.route('/login', methods=['POST', 'GET'])
def login():
    error = None
    if request.method == 'POST':
        if valid_login(request.form['username'],
                       request.form['password']):
            return log_the_user_in(request.form['username'])
        else:
            error = 'Invalid username/password'
    # the code below is executed if the request method
    # was GET or the credentials were invalid
    return render_template('login.html', error=error)
```

- Si una llave no existe en el atributo form tendremos KeyError
  - Si no se controla la excepción, retornará `HTTP 400 Bad Request`

# Interacción con Base de Datos

# Base de Datos Relacionales

- Comenzaron a principios de los años 70
- Base de datos
  - Conjunto de tablas “rectangulares”
  - Cada fila de la tabla es un registro de alguna cosa
  - El registro contiene varias piezas de información, llamados campos
  - Ejemplo:

| LASTNAME | FIRSTNAME | ID | POSTAL_CODE | AGE | SEX |
|---|---|---|---|---|---|
| Gauss | Karl | 119 | 19107 | 30 | M |
| Smith | Mark | 3 | T2V 3V4 | 53 | M |
| Noether | Emmy | 118 | 19107 | 31 | F |
| Smith | Jeff | 28 | K2G 5J9 | 19 | M |
| Hamilton | William | 247 | 10139 | 2 | M |

# Base de Datos Relacionales

- En el ejemplo
  - Los nombres de los campos son `lastname`, `firstname`, `id`, `postal_code`, `age` y `sex`
  - Cada línea de una tabla es llamada registro, fila o tupla
- SQL: Structured Query Language
  - Inventado por IBM en los 70
  - Lenguaje para escribir búsquedas y modificaciones en una base de datos relacional
  - Enorme éxito: simple de usar
  - Soportado por los sistemas de bases de datos

# Base de Datos Relacionales

- Operaciones que se pueden hacer con una tabla:
  - SELECT: busca todos los registros que cumplan cierta propiedad
  - INSERT: agrega nuevos registros
  - DELETE: elimina registros existentes
  - UPDATE: modifica registros existentes
- Son los 4 comandos SQL más importantes
  - También se llaman sentencias o *queries*

# Base de Datos Relacionales

- Ejemplos: obtener el nombre de todas las personas con apellido “Smith”:

```
SELECT firstname FROM people WHERE lastname = 'Smith'
```

  - Borrar 'Mark Smith' de la tabla

```
DELETE FROM people WHERE id = 3
```

  - Agregar un año a 'William'

```
UPDATE people SET age = age+1 WHERE id = 247
```

# Base de Datos Relacionales

- Ejemplo
  - Insertar nuevo registro

```
INSERT INTO people VALUES ('Euler', 'Leonhard', 248, NULL, 58, 'M')
```

- Existe varios otros comandos SQL:
  - Crear, eliminar tablas
  - Dar, quitar permisos de acceso
  - Para confirmar o abandonar transacciones
  - Mostrar todas las tablas
  - Interactuar con tablas especiales

# Base de datos: PyMySQL

- https://pypi.org/project/PyMySQL/
- Biblioteca para comunicar Python con MySQL
  - Implementa PEP 249

    https://www.python.org/dev/peps/pep-0249/
  - Documentación

    https://pymysql.readthedocs.io/en/latest/
- Permite las funciones básicas
  - Conexión a base de datos
  - Ejecución de sentencias SQL
    - Obtener, insertar, actualizar, eliminar
  - Manejo de errores

# PyMySQL

- Se consideran dos objetos principales:
- `Connection`: representación de un *socket* de comunicación con el servidor MySQL
  - Establece la conexión con la base de datos
  - Acepta varios argumentos:
    - Host
    - User
    - Password
    - Database
    - Port
    - Charset
    - init_command
    - autocommit

https://pymysql.readthedocs.io/en/latest/modules/connections.html

# PyMySQL

- `Cursor`: se utiliza para interactuar con la base de datos
  - Se debe obtener desde el objeto `Connection`
- Provee funciones para ejecutar sentencias SQL
  - `execute(query, args=None)`
    - Query: sentencia SQL
    - Args: opcional, parámetros que se usarán en la sentencia SQL
    - Retorna un número entero que indica la cantidad de filas que fueron afectadas por la consulta

# PyMySQL

- Conexión a base de datos

```
import pymysql

# conexion a base de datos
conn = pymysql.connect(
    db='ejemplo',
    user='cc5002',
    passwd='cc5002',
    host='localhost',
    charset='utf8')
c = conn.cursor()
```

# PyMySQL

- Agregar información
  - Definir SQL de inserción
  - Ejecutar agregando datos
  - Revisar cantidad de filas afectadas
  - Controlar errores

```
sql = "INSERT INTO persona (nombre, apellido) VALUES (%s, %s)"
try:
    resultado = c.cursor().execute(sql, (nombre, apellido))
    c.commit()
    return resultado == 1
except pymysql.Error as e:
    app.logger.error("Error con base de datos: {0} {1} ".format(e.args[0], e.args[1]))
```

# PyMySQL

- Recuperar información
  - Definir SQL de lectura
  - Ejecutar agregando información para condiciones
  - Verificar cantidad de resultados
  - Recorrer resultados

```
def get_personas(c):
    sql = "SELECT id, nombre, apellido FROM persona";
    cursor = c.cursor()
    cursor.execute(sql)
    c.commit()
    personas = cursor.fetchall()
    listaPersonas = []
    if len(personas) > 0:
        for per in personas:
            personaBD = Persona(per[0], per[1], per[2])
            listaPersonas.append(personaBD)
    return listaPersonas
```