

TDDD97 - Web Programming

# Server-side development

Sahand Sadjadee

Dept. of Computer and Information Science

Linköping University



# Outline

- Python
  - Flask
  - SQL/SQLite
  - Rest API
  - Validation
  - Security
-

# Server-side languages

Websites require two key components to function: a client and a web server. Clients, as we've learned, are any web browser or device that's used to view and interact with a website. All of the files and data associated with displaying a website on a client are stored on a web server.

<https://www.codeschool.com/beginners-guide-to-web-development/server-side-languages>

in the SPA world, server-side languages are used to create REST API, **implement part of the logic**, implement validation, communicate with the database and ... at the server-side.

# Server-side languages

Programming languages used in most popular websites\*

| Websites ↕                | Popularity<br>(unique visitors per month) <sup>[1]</sup> ↕ | Front-end<br>(Client-side) ↕ | Back-end<br>(Server-side) ↕   | Database ↕  | Notes  |
|---------------------------|--|------------------------------|---|---|--|
| Google.com <sup>[2]</sup> | 1,600,000,000  | JavaScript                   | C, C++, Go, <sup>[3]</sup> Java, Python   | Bigtable, <sup>[4]</sup> MariaDB <sup>[5]</sup>                   | The most used search engine in the world                                       |
| Facebook.com              | 1,100,000,000  | JavaScript                   | Hack, PHP (HHVM), Python, C++, Java, Erlang, D, <sup>[6]</sup> Xhp, <sup>[7]</sup> Haskell <sup>[8]</sup> | MariaDB, MySQL, <sup>[9]</sup> HBase<br>Cassandra <sup>[10]</sup> | The most visited social networking site  |
| YouTube.com               | 1,100,000,000  | JavaScript                   | C, C++, Python, Java, <sup>[11]</sup> Go <sup>[12]</sup>  | Vitess, BigTable, MariaDB <sup>[5]</sup> <sup>[13]</sup>          | The most visited video sharing site  |
| Yahoo                     | 750,000,000  | JavaScript                   | PHP   | MySQL, PostgreSQL, <sup>[14]</sup> VB.NET                         | Yahoo is presently <sup>[when?]</sup> transitioning to Node.js <sup>[15]</sup> |
| Amazon.com                | 500,000,000  | JavaScript                   | Java, C++, Perl <sup>[16]</sup>   | Oracle Database <sup>[17]</sup>                                   | Popular internet shopping site   |
| Wikipedia.org             | 475,000,000  | JavaScript                   | PHP, Hack   | MySQL <sup>[citation needed]</sup> , MariaDB <sup>[18]</sup>      | "MediaWiki" is programmed in PHP, runs on HHVM; free online encyclopedia       |
| Twitter.com               | 290,000,000  | JavaScript                   | C++, Java, Scala, Ruby <sup>[19]</sup>  | MySQL <sup>[20]</sup>   | Popular social network.  |
| Bing                      | 285,000,000  | JavaScript                   | ASP.NET   | Microsoft SQL Server  |  |
| eBay.com                  | 285,000,000  | JavaScript                   | Java, <sup>[21]</sup> JavaScript, <sup>[22]</sup> Scala <sup>[23]</sup>                                   | Oracle Database   | Online auction house   |
| MSN.com                   | 280,000,000  | JavaScript                   | ASP.NET   | Microsoft SQL Server  | An email client, for simple use. Mostly known as "messenger".                  |
| Microsoft                 | 270,000,000  | JavaScript                   | ASP.NET   | Microsoft SQL Server  | One of the world's largest software companies.                                 |
| Linkedin.com              | 260,000,000  | JavaScript                   | Java, JavaScript, <sup>[24]</sup> Scala   | Voldemort <sup>[25]</sup>   | World's largest professional network.  |
| Pinterest                 | 250,000,000  | JavaScript                   | Django, <sup>[26]</sup> Erlang  | MySQL, Redis <sup>[27]</sup>                                      |  |
| WordPress.com             | 240,000,000  | JavaScript                   | PHP, JavaScript <sup>[28]</sup> (Node.js)   | MariaDB, MySQL  |  |

[https://en.wikipedia.org/wiki/Programming\\_languages\\_used\\_in\\_most\\_popular\\_websites](https://en.wikipedia.org/wiki/Programming_languages_used_in_most_popular_websites)

# Python



- High-level, interactive, object-oriented programming language
- No need for compilation
- You can interact with the interpreter directly through the Python command prompt
- History
  - Developed by Guido van Rossum in the late-1980s and early-1990s at the National Research Institute for Mathematics and Computer Science in the Netherlands
  - Inspired by many other languages (e.g., ABC, Modula-3, C, C++, Algol-68, SmallTalk, and various scripting languages)

# Python

```
def add5(x):  
    return x+5  
  
def dotwrite(ast):  
    nodename = getNodeName()  
    label=symbol.sym_name.get(int(ast[0]),ast[0])  
    print '      %s [label="%s' % (nodename, label),  
    if isinstance(ast[1], str):  
        if ast[1].strip():  
            print '= %s'];' % ast[1]  
        else:  
            print '"]'  
    else:  
        print '"]';'  
        children = []  
        for n, childrenumerate(ast[1:]):  
            children.append(dotwrite(child))  
        print , '      %s -> {' % nodename  
        for in :namechildren  
            print '%s' % name,
```

The sample code is taken from Wikipedia.

# Python Highlights

- Easy to learn, read, and maintain
- Broad standard library
- Interactive mode (prompt)
- Available on many platforms
- Support for functional and structured programming methods as well as OOP
- Interfaces to databases available

# Python basic syntax

- Lines and indentation
  - No braces { } to define blocks
  - Blocks are defined by indentation

```
if True:
    print "It is true!"
else:
    print "This is false."
```

- In Python, all continuous lines indented with similar number of spaces form a block.
- Multiple statements on the same line should be separated by semicolon (;)



# Reserved words

|          |         |        |
|----------|---------|--------|
| and      | exec    | not    |
| assert   | finally | or     |
| break    | for     | pass   |
| class    | from    | print  |
| continue | global  | raise  |
| def      | if      | return |
| del      | import  | try    |
| elif     | in      | while  |
| else     | is      | with   |
| except   | lambda  | yield  |

# Python basic syntax

- Comments

- A hash sign (#) comments out the rest of the line

```
# First comment  
print "Hello, Python!"; # second comment
```

- Multi-line comments are made by using three consequent quotations for starting and three other consequent quotations for closing the comment.

- Blank lines

- Ignored by the Python interpreter
- In interactive sessions, an empty line terminates multiline statements

# Multiline statements

- Normally, statements end with a new line
- It is possible to use multiple lines with (\)

```
grand_total = variable_one + \  
              variable_two + \  
              variable_three
```

# Quotation in Python

- String literals
  - Single quote (')
  - Double quote (")
  - Same type at the start and end of the string required
- Triple quotes can be used to create multiline strings

```
word = 'Hello'  
two_words = "Hello World"
```

# Variables

- No need to declare them
- Assignments:

```
cars = 200
parking_spaces = 150
parking_type = "Residential"

print cars
print parking_spaces
print parking_type
```

- Standard data types: Numbers, String, List, Tuple, Dictionary

# Strings

- Plus (+) is used for string concatenation and asterisk (\*) for repetition (\*)

```
str = 'Hello World'

print str          # Prints complete string
print str[0]       # Prints first character of the string
print str[2:5]     # Prints characters starting from 3rd to 5th
print str[2:]      # Prints string starting from 3rd character
print str * 2      # Prints string two times
print str + "TEST" # Prints concatenated string
```

# Lists

- Compound data type

```
list = [ 'Hello', 123, 456, 'world', 16.2 ]
shortlist = [97, 'TDDD']

print list           # Prints the list
print list[0]        # Prints first element
print list[1:3]       # Prints elements starting from 2nd till 3rd
print list[2:]        # Prints elements starting from 3rd element
print shortlist * 2    # Prints list twice
print list + shortlist # Prints concatenated lists
```

# Tuples

- Sequences similar to lists
- Tuples cannot be updated (i.e., they are *read-only* lists)

```
tuple = ( 'Hello', 123, 456, 'world', 16.2 )
shorttuple = (97, 'TDDD')

print tuple           # Prints the tuple
print tuple[0]        # Prints first element
print tuple[1:3]      # Prints elements starting from 2nd till 3rd
print tuple[2:]       # Prints elements starting from 3rd element
print shorttuple * 2   # Prints tuple twice
print tuple + shorttuple # Prints concatenated tuple
```



# Dictionary

- Associative array
- Key–value pairs

```
dict = {'name': 'John Doe', 'code': 1337, 'dept': 'CS'}  
  
print dict['name']      # Prints value for 'name' key  
print dict              # Prints complete dictionary  
print dict.keys()       # Prints all the keys  
print dict.values()     # Prints all the values
```

# Control structures

- If-statements

```
if expression:  
    statement(s)
```

```
if expression:  
    statement(s)  
else:  
    statement(s)
```

```
if expression1:  
    statement(s)  
elif expression2:  
    statement(s)  
else:  
    statement(s)
```

```
if expression1:  
    statement(s)  
    if expression2:  
        statement(s)  
    elif expression3:  
        statement(s)  
    else  
        statement(s)  
elif expression4:  
    statement(s)  
else:  
    statement(s)
```

# Control structures

- While loop

```
while expression:  
    statement(s)
```

- For loop

```
for iterating_var in sequence:  
    statements(s)
```

- Break, continue, and else (!) statements available

# Functions

- The keyword **def** is used for defining functions
  - Parameters passed by reference
  - General syntax:

```
def function_name( parameters ):  
    "function documentation string"  
    function_body  
    return [expression]
```

- Example:

```
def add_numbers( i, j ):  
    return i + j  
  
>>> add_numbers(2, 3)  
5  
>>>
```

# Function arguments

- Different types of arguments
  - Required arguments
    - “Normal” arguments, runtime error if not provided
  - Keyword arguments
    - Caller provides the argument name
    - Definition: `def hello( s ):`
    - Call: `hello( s = "World" )`
  - Default arguments
    - Uses default value of argument if not passed by the caller
    - Definition: `def hello( s = "World" )`
    - Call: `hello() => "World"      hello("Universe") => "Universe"`
  - Value-length arguments
    - Varying number of arguments

# Classes

- Class definitions

```
class Person:
    def __init__(self, name, age):
        self.name = name
        self.age = age

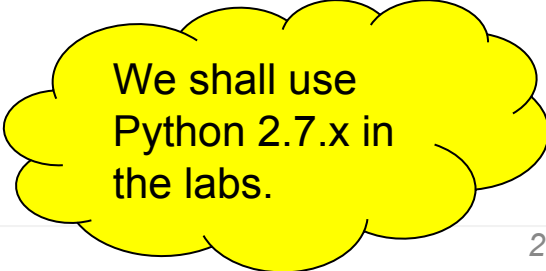
    def printPerson(self):
        print "Name : ", self.name, " , Age: ", self.age
```

- Instantiation

```
p1 = Person("Kalle", 22)
p2 = Person("Anna", 23)
printPerson p1
printPerson p2
print (p1.age + p2.age)
```

# Python 2.x versus 3.x

- Python 2.x
  - Now considered “legacy”
- Python 3.x
  - The present and future of the language
  - A.k.a. Python 3000 or Py3K
  - Fixes some well-known annoyances and warts
  - Print statement replaced with a **print()** function



We shall use  
Python 2.7.x in  
the labs.

# Flask, the micro-framework



# Web frameworks

“Server-side web frameworks (a.k.a. "web application frameworks") are software frameworks that make it easier to write, maintain and scale web applications. They provide tools and libraries that simplify common web development tasks, including routing URLs to appropriate handlers, interacting with databases, supporting sessions and user authorization, formatting output (e.g. HTML, JSON, XML), and improving security against web attacks.”, MDN.

[https://developer.mozilla.org/en-US/docs/Learn/Server-side/First\\_steps/Web\\_frameworks](https://developer.mozilla.org/en-US/docs/Learn/Server-side/First_steps/Web_frameworks)

[https://en.wikipedia.org/wiki/Comparison\\_of\\_web\\_frameworks](https://en.wikipedia.org/wiki/Comparison_of_web_frameworks)

# Web frameworks

It is possible to create Rest API and also do server-side templating both in most of the popular web frameworks today.

For example, FLASK!

# Flask

- Light-weight web development framework
- Python-based
- Microframework: small, but extensible core
- Examples of extensions include:
  - object-relational mappers
  - form validation
  - upload handling
  - various open authentication technologies
- Open source, BSD license
- Web site: <http://flask.pocoo.org/>



# Flask features

- built-in development server and fast debugger
- integrated support for unit testing
- RESTful request dispatching
- [Jinja2](#) templating
- support for secure cookies (client side sessions)
- WSGI 1.0 compliant
- Unicode based

<http://quintagroup.com/cms/python/flask>

# Hello World Flask Application

- In the Python file `hello.py`:

```
from flask import Flask
app = Flask(__name__)

@app.route('/')
def hello_world():
    return 'Hello World!'

if __name__ == '__main__':
    app.run()
```

- Start the application with:

```
> python hello.py
* Running on http://127.0.0.1:5000/
```

# What it means

- `from flask import Flask`
  - Import Flask stuff
- `app = Flask(__name__)`
  - Create the Flask application instance
- `@app.route('/')`
  - Function *decoration*
  - Tells flask to route requests for “/” here
- `if __name__ == '__main__':`  
    `app.run()`
  - Start the application
  - Similar to the main method/function in other languages

# Flask Routing

- The route() decorator binds a function to a URL

- Examples:

```
@app.route('/')
def index():
    return 'Index Page'

@app.route('/hello')
def hello():
    return 'Hello World'
```

- Variable URLs:

```
@app.route('/user/<username>')
def show_user_profile(username):
    # show the name of a user
    return 'User %s' % username
```

# Flask Template Rendering

- Based on the Jinja2 template language/engine for Python
- HTML templates should be located in the templates directory
- Template rendering function: `render_template()`
- Example:

```
from flask import render_template

@app.route('/hello/')
@app.route('/hello/<name>')
def hello(name=None):
    return render_template('hello.html', name=name)
```



# Flask Template Rendering (cont.)

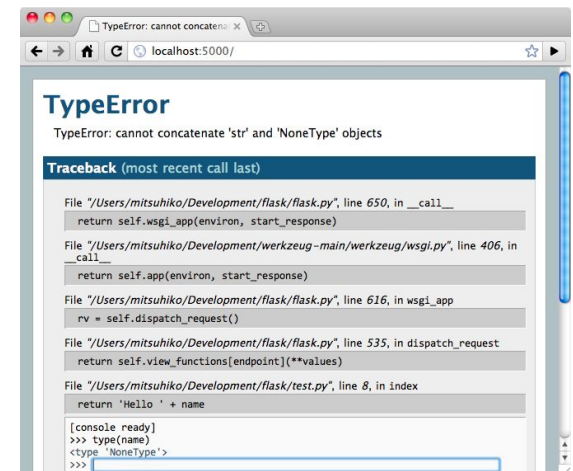
- In the file templates/hello.html

```
<!doctype html>
<title>Hello from Flask</title>
{% if name %}
    <h1>Hello {{ name }}!</h1>
{% else %}
    <h1>Hello World!</h1>
{% endif %}
```

# Flask Debug Mode

- Interactive debugger in the web browser
  - Shows interactive stack traces on errors
  - Python command prompt in the web page
- Reloads the server automatically on code changes
- Do not use for production servers (because of security)
- Enable by setting the debug flag:

```
app.debug = True  
app.run()
```



# Flask under Linux (IDA)

- Make sure you are running python 2.7.x
  - Check with `python -V`
- Create a virtual environment
  - Create a new environment for the virtual environment. In the Python prompt: `virtualenv <directory>`
  - Use the install tool *pip* to install Flask. In the virtual environment root directory: `bin/pip install <package>`
- Run the application using the Python executable in the virtual environment (i.e., not the global Python)
  - In the virtual environment root directory: `bin/python`

# Flask under Linux (IDA)

```
> python -V
Python 2.7.6

> virtualenv test
New python executable in test/bin/python
Installing setuptools.....done.
Installing pip.....done.

> cd test
> bin/pip install flask
Downloading/unpacking flask...

> bin/python
Python 2.7.6 (default, Oct 26 2016, 20:30:19)
[GCC 4.8.4] on linux2
Type "help", "copyright", "credits" or "license" for more information.
>>>
```

# Python virtual environments

- Isolated Python environments — sandboxes
- Separate directory for each virtual environment
- Activate/deactivate commands
  - Unix/bash: `source bin/activate`
  - MS Windows: `scripts\activate`
  - Both: `deactivate`
- Examples of use:
  - Separated development environments
  - Mixing Python version on the same machine
- Command to create: **`virtualenv <directory>`**

# HTTP status codes

- To abort a HTTP request early, call `return`
- Example: `return "", 404`
- Common error codes
  - 404 Not Found
  - 403 Forbidden
  - 410 Gone
  - 500 Internal Server Error

[http://en.wikipedia.org/wiki/List\\_of\\_HTTP\\_status\\_codes](http://en.wikipedia.org/wiki/List_of_HTTP_status_codes)

# HTTP methods

- [GET](#) The GET method requests a representation of the specified resource. Requests using GET should only retrieve data.
- [POST](#) The POST method is used to submit an entity to the specified resource, often causing a change in state or side effects on the server.
- [PUT](#) The PUT method replaces all current representations of the target resource with the request payload.
- [DELETE](#) The DELETE method deletes the specified resource.

<https://developer.mozilla.org/en-US/docs/Web/HTTP/Methods>

# HTTP methods

<https://assertible.com/blog/7-http-methods-every-web-developer-should-know-and-how-to-test-them>

**store** Access to Petstore orders

**GET** `/store/inventory` Returns pet inventories by status

**POST** `/store/order` Place an order for a pet

**GET** `/store/order/{orderId}` Find purchase order by ID

**DELETE** `/store/order/{orderId}` Delete purchase order by ID

| HTTP Method | Safe | Idempotent |
|-------------|------|------------|
| GET         | ✓    | ✓          |
| POST        | ✗    | ✗          |
| PUT         | ✗    | ✓          |
| DELETE      | ✗    | ✓          |
| OPTIONS     | ✓    | ✓          |
| HEAD        | ✓    | ✓          |

<https://blog.4psa.com/rest-best-practices-choosing-http-methods/>



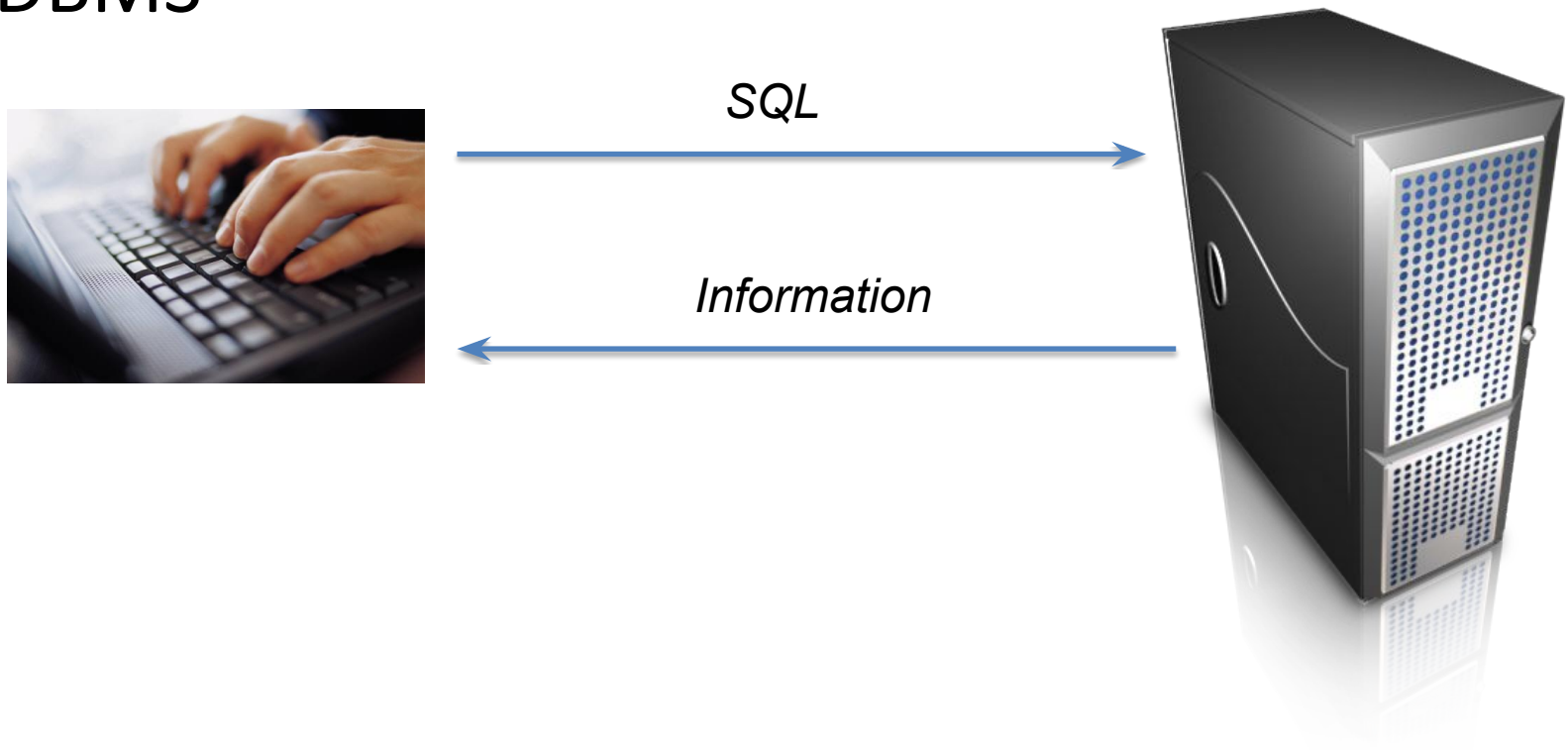
# Useful Resources

- Python tutorial: <http://docs.python.org/2/tutorial/>
- Another Python tutorial: <http://www.learnpython.org>
- Flask homepage: <http://flask.pocoo.org/docs/>
- Flask tutorial: <http://flask.pocoo.org/docs/tutorial/>
- Another Flask tutorial:  
<http://blog.miguelgrinberg.com/post/the-flask-mega-tutorial-part-i-hello-world>

SQL

# SQL

Standard language for working/interacting with a DBMS



# SQL- History

- IBM Sequel language developed as part of System R project at the IBM San Jose Research Laboratory
- Renamed Structured Query Language (SQL)
- ANSI and ISO standard SQL:
  - SQL-86, SQL-89, SQL-92
  - SQL:1999, SQL:2003, SQL:2008
- Commercial systems offer most, if not all, SQL-92 features, plus varying feature sets from later standards and special proprietary features.

# SQL and Flask

- Two possible methods for working with the database
- SQL
  - Using SQL directly in the code
  - Will be used in the labs
- Using an ORM like SQLAlchemy
  - Python SQL toolkit and Object Relational Mapper
  - More powerful and flexible—suitable for larger applications



# SQLite



- SQL database engine
- Serverless
- In-process library
  - Small footprint, compact library
  - Well-suited for use in applications and embedded devices
- Implemented in ANSI-C
  - Cross platform: Unix (Linux, OS-X, Android, iOS) and MS Windows
- SQLite software and documentation in the public domain <https://www.sqlite.org/>

# SQLite Example

- Create a new database
  - At the command prompt: `>sqlite3 test.db`
- Enter SQL commands to create and populate the database
  - `sqlite> create table table1 (course varchar(6), credits smallint);`
  - `sqlite> insert into table1 values('TDDD24', 4);`
  - `sqlite> insert into table1 values('TDDD97', 6);`
  - `sqlite> select * from table1;`
  - TDDD24|4
  - TDDD97|6
  - `sqlite> select credits from table1 where course='TDDD97';`
  - 6
  - `sqlite>`

# SQLite suitability

SQLite ***works well*** for

- Application file format
  - Load/save application data
- Embedded devices and applications
- Websites
  - Low to medium traffic web sites (say < 100k hits/day)
- Replacement for ad-hoc disk files
- Internal or temporary databases
  - In-memory databases supported
- Command-line dataset analysis tool
- Stand-in for enterprise database during demos and testing
- Learning databases and SQL



# SQLite unsuitability

SQLite may be *inappropriate* for:

- Client-server applications
  - Where a separate database server is needed
  - SQLite will work over a network file system, however
- High-volume websites
  - Where a separate database machine is needed
- Very large datasets
  - SQLite databases limited to 140 TB
  - Often, the underlying file system is the actual limit
- High concurrency
  - Supports unlimited number of readers, but only one writer at the time

# Sample Flask Server

```
import sqlite3
from flask import g

def connect_db():
    return sqlite3.connect("mydatabase.db")

def get_db():
    db = getattr(g, 'db', None)
    if db is None:
        db = g.db = connect_db()
    return db

def init():
    c = get_db()
    c.execute("drop table if exists entries")
    c.execute("create table entries (id integer primary key, name text,message text)")
    c.commit()

def add_message(name,message):
    c = get_db()
    c.execute("insert into entries (name,message) values (?,?)", (name,message))
    c.commit()

def close():
    get_db().close()
```

# Sample Flask Server (cont.)

- Checking the database content from the command line:

```
> sqlite3 mydatabase.db
SQLite version 3.8.2 2013-12-06 14:53:30
Enter ".help" for instructions
Enter SQL statements terminated with a ";"
sqlite> select * from entries;

1|Nisse|Hi from Nisse!
2|Mary|Supercalifragilisticexpialidocious
3|John|What is this?
4|Joe|This looks cool.
sqlite> select * from entries where name='Nisse';

1|Nisse|Hi from Nisse!
sqlite>
```

# Sample Flask Server (cont.)

```
def get_message(name):  
    c = get_db()  
    cursor = c.cursor()  
    cursor.execute("select name, message from entries where name = '" + name + "'")  
    entries = [dict(name=row[0], message=row[1]) for row in cursor.fetchall()]  
    c.close()  
    return entries[0]['name'] + " says: " + entries[0]['message']  
  
def close():  
    get_db().close()
```

# In addition to constructing SQL queries, database programming entails...

- Connect and disconnect to DB
- Validate data to put into DB
  - Assumed by DB to be correct
- Manage UI creation in relation to SQL queries
- Manage object to SQL transformation of data
- Manage security concerns and potentially pass them on to DB
  - Cannot rely on DB security alone
  - Manage where DB-related security is handled and how it is exposed

Rest API, Representational State transfer

# Rest constraints

- Client-Server
- Cacheable
- Code on demand
- Layered system
- Uniform interface
- Stateless

<http://www.restapitutorial.com/lessons/whatisrest.html#>

# REST API

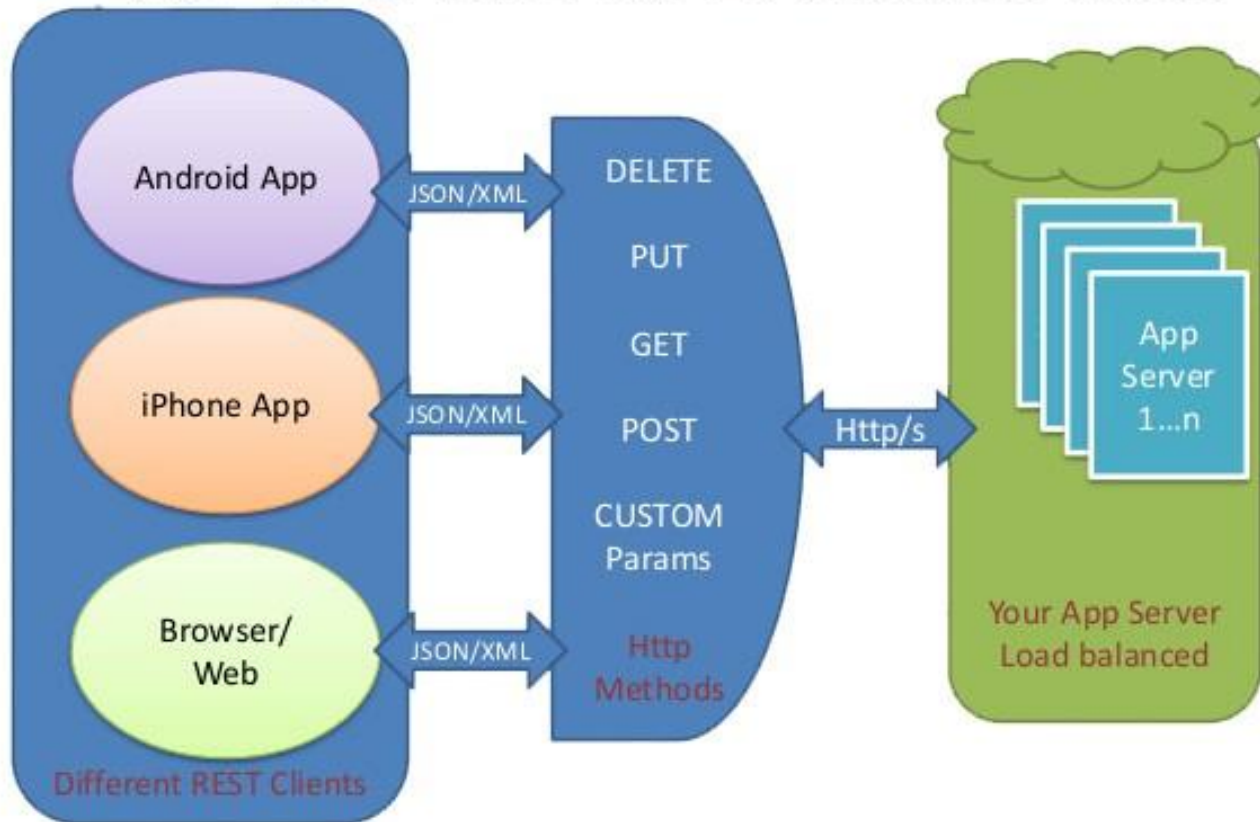
A set of server-side web services, functions, designed and implemented according to Rest constraints which are callable from client-side by using HTTP protocol are called REST API. JSON is most common format for returning data to the client-side.

Good read

<https://restful.io/rest-api-back-to-basics-c64f282d972>



# REST API Architecture

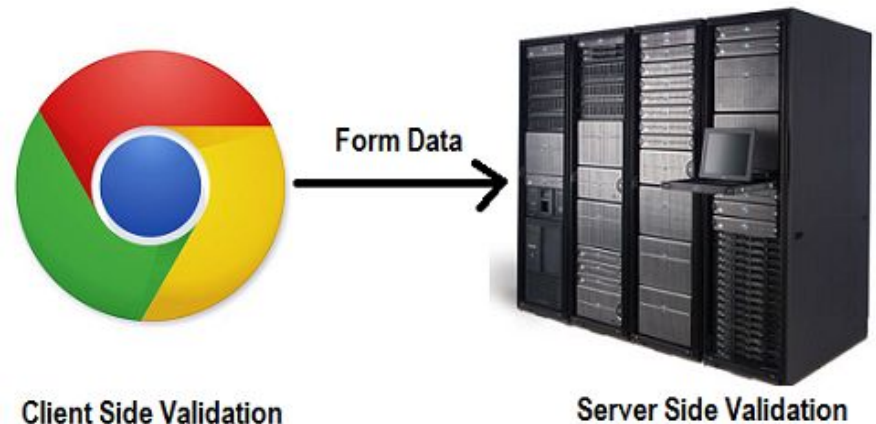


<https://shareurcodes.com/blog/creating%20a%20simple%20rest%20api%20in%20php>

# Validation

# Validation

- Client-side code can be inactivated or not used by the user at all.
- Data needs to be validated both at the server-side and the client-side.
- Client-side validation improves responsiveness and user experience.
- Client-side validation has nothing to do with assuring data integrity and consistency.
- Server-side validation improves data integrity and consistency.
- In real-world all data sent to the server need to be validated before any operation is done on it.



Good read

<http://net-informations.com/fag/asp/validation.htn>

# Validation

## Examples:

- The password is not long enough as required.
- The email address is not in the right form.
- The posted message contains not allowed words or symbols.
- Therequested field is empty.
- ...

# Security

# CIA triad

A simple but widely-applicable security model is the CIA triad; standing for **Confidentiality**, **Integrity** and **Availability**; three key principles which should be guaranteed in any kind of secure system.

## **Confidentiality**

Confidentiality is the ability to hide information from those people unauthorised to view it.

## **Integrity**

The ability to ensure that data is an accurate and unchanged representation of the original secure information.

## **Availability**

It is important to ensure that the information concerned is readily accessible to the authorised viewer at all times.

<http://www.doc.ic.ac.uk/~ajs300/security/CIA.htm>

# CIA principles



# Some common attacks

SQL Injection: can target Confidentiality, Integrity and availability.

XSS(Cross-site scripting): can target confidentiality, integrity and availability.

DDOS(Distributed Denial of Service): can target only availability



# SQL Injection

## Phonebook Record Manager

Username

John

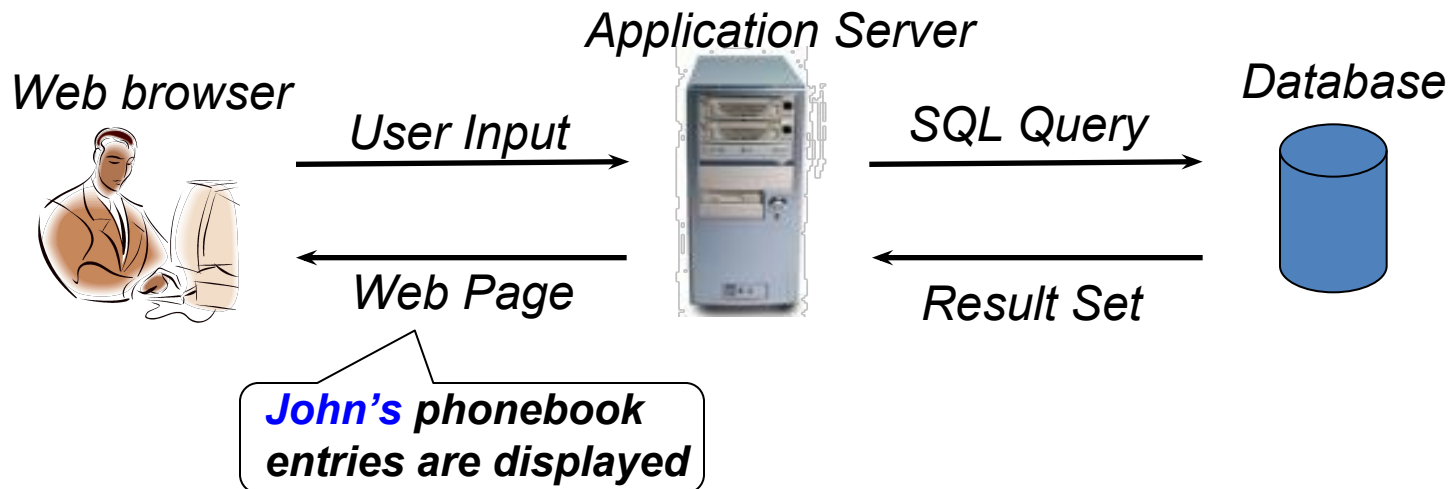
Password

abcd

Submit

*SQL: Structured Query Language.  
Used for query, delete, insert, and  
update database records.*

***SELECT \* FROM phonebook WHERE  
username = 'John' AND  
password = 'abcd'***



# SQL Injection

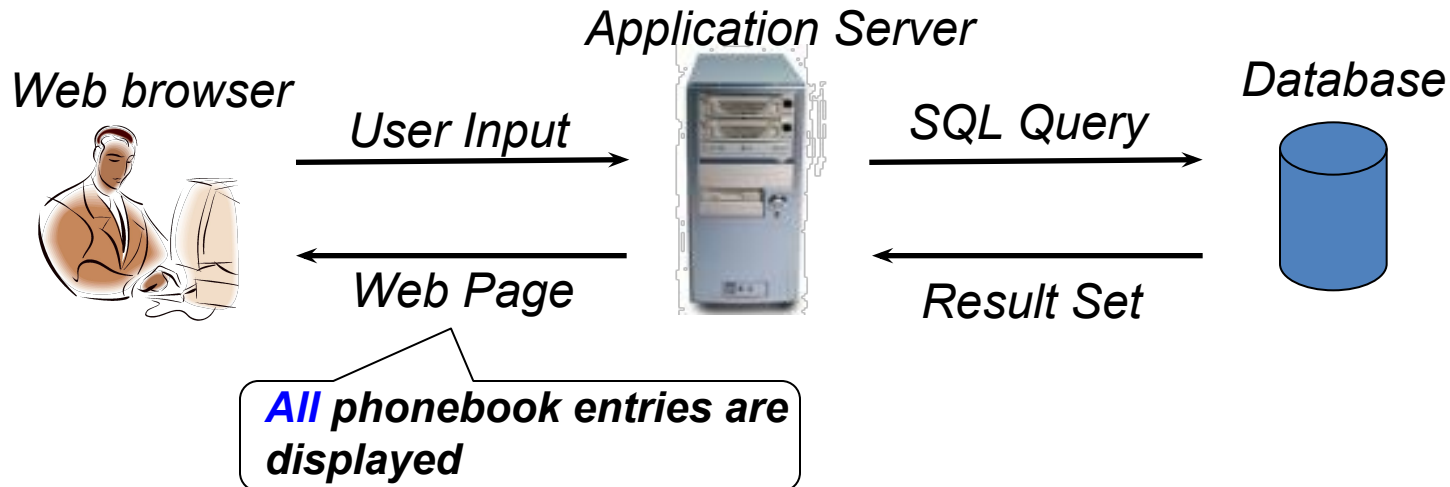
## Phonebook Record Manager

Username **John ' OR 1=1 --**  
Password **not needed**

**Submit**

```
SELECT * FROM phonebook WHERE  
username = 'John' OR 1=1 --' AND  
password = 'not needed'
```

*Everything after -- is ignored!*

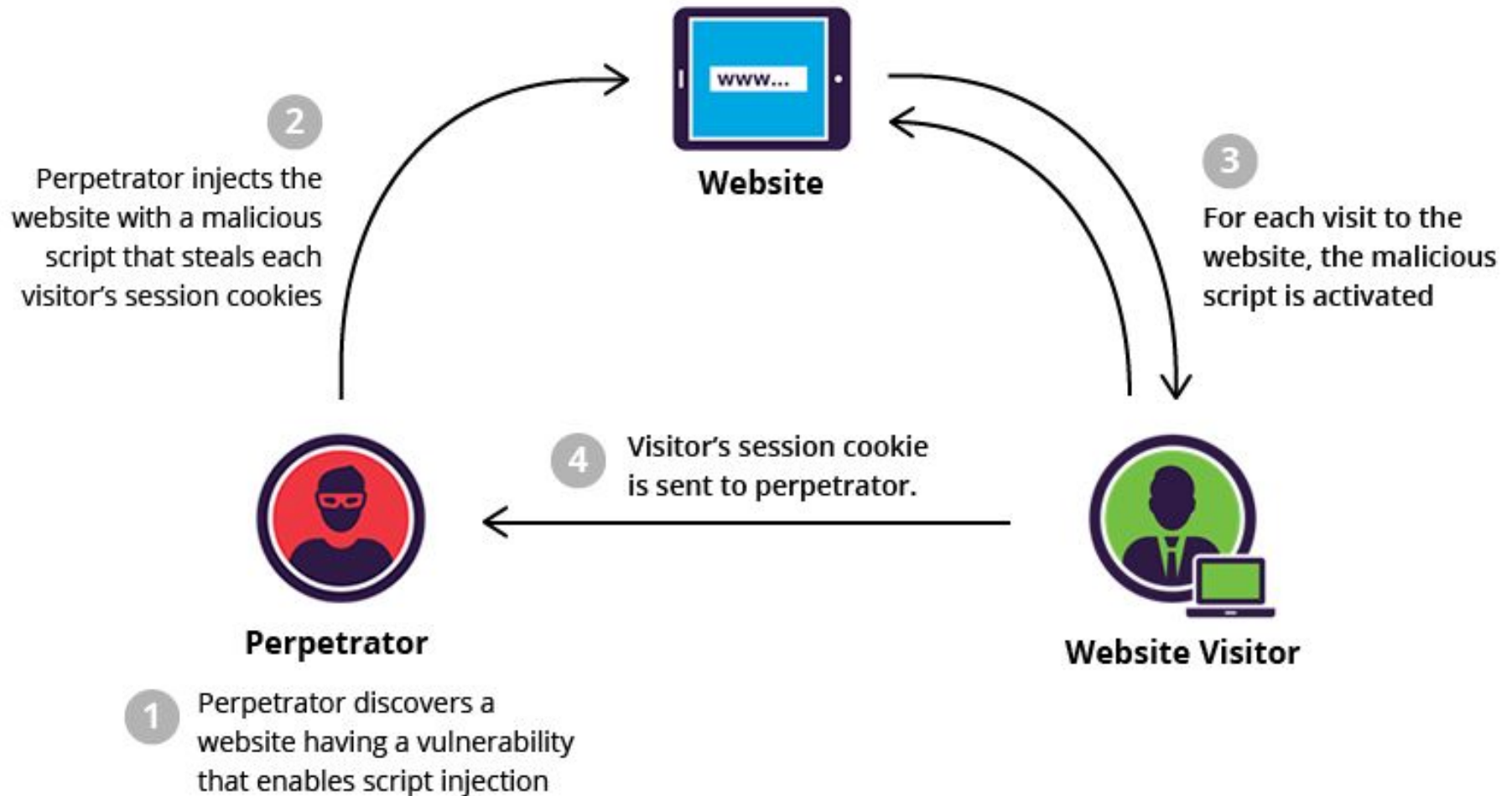


# Cross-site scripting

Cross-Site Scripting (XSS) attacks are a type of injection, in which malicious scripts are injected into otherwise benign and trusted web sites. XSS attacks occur when an attacker uses a web application to send malicious code, generally in the form of a browser side script, to a different end user. Flaws that allow these attacks to succeed are quite widespread and occur anywhere a web application uses input from a user within the output it generates without validating or encoding it.

[https://www.owasp.org/index.php/Cross-site\\_Scripting\\_%28XSS%29](https://www.owasp.org/index.php/Cross-site_Scripting_%28XSS%29)

# Cross-site scripting





**Thanks for listening!**