REST

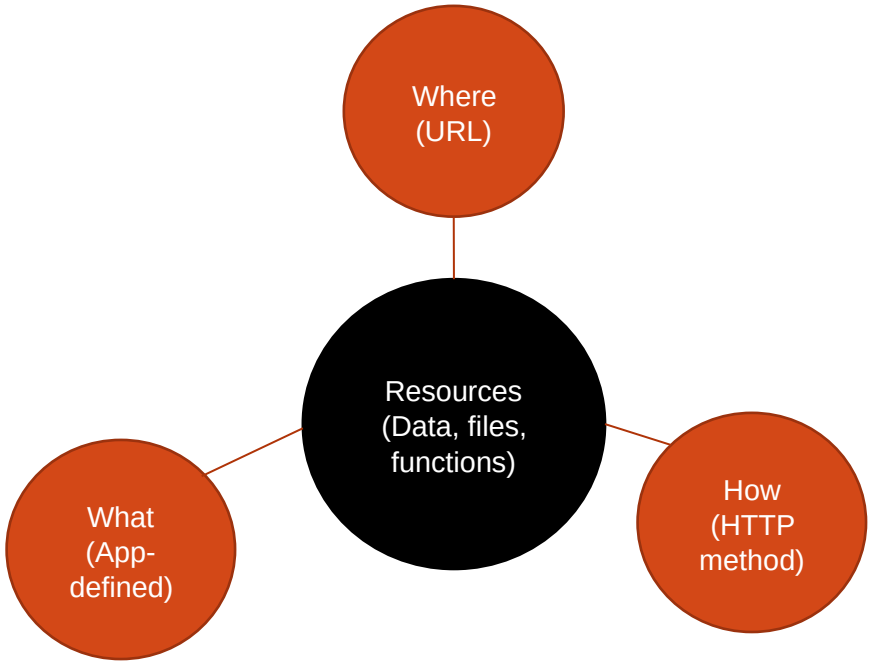
Web-based APIs

REST

- Representational State Transfer
- Style of web software architecture that simplifies application
- Not a standard, but a design pattern

REST

- Take all resources for web application (data, files, functions)
  - Identify each resource and action on resource via an HTTP method and URL.
  - Method selects action
  - Send arguments via the HTTP request (e.g. in URL, URL parameters, or payload)



REST toy example

- http://foo.com/bar/users
  - Server foo.com
  - Database bar
  - Table users
  - URL returns table users in database bar in a particular format (XML, JSON)
- Common examples
  - Twitter, Flickr, Amazon

REST and HTTP methods

- HTTP request methods indicate the desired action
- GET
  - Requests a representation of the specified resource.
  - Use for operations that have NO side-effects (safe operations)
  - Works with robots and crawlers.
- POST
  - Submits data to be processed (e.g., from an HTML form) to the identified resource. Data is included in the body of the request.
- PUT
  - Uploads a representation of the specified resource.
- DELETE
  - Deletes the specified resource.

REST and security

- Each API call must ensure request is authenticated and authorized
  - Requires attention to many of the OWASP Top 10
    - A4: Insecure Direct Object Access
    - A7: Missing Function Level Access Control
    - A2: Broken Authentication and Session Management
    - A1: Injection
  - Now in OWASP Top 10 for 2017 draft

Top 10 2013	Top 10 2017
A1 – Injection	A1 – Injection
A2 – Broken Authentication and Session Management	A2 – Broken Authentication and Session Management
A3 – Cross-Site Scripting (XSS)	A3 – Cross-Site Scripting
A4 – Insecure Direct Object References	A4 – Broken Access Control
A5 – Security Misconfiguration	A5 – Security Misconfiguration
A6 – Sensitive Data Exposure	A6 – Sensitive Data Exposure
A7 – Missing Function Level Access Control	A7 – Insufficient Attack Protection
A8 – Cross-site Request Forgery (CSRF)	A8 - Cross-site Request Forgery (CSRF)
A9 – Using Components with Known Vulnerabilities	A9 – Using Components with Known Vulnerabilities
A10 – Unvalidated Redirects and Forwards	A10 – Unprotected APIs

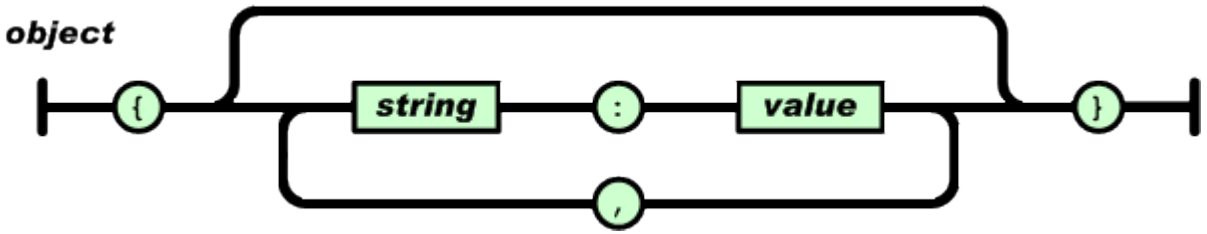
JSON

JSON

- JavaScript Object Notation
  - De-facto web object data format
    - Subset of JavaScript
    - Minimal, lightweight, text-based syntax
    - Easy to parse and generate
  - Prevalent in most web sites
  - Prevalent in many web APIs, often as part of a REST architecture
  - Designed to enable stateful, real-time communication between browser and web application
    - Often used to allow web server to directly modify elements of a page without refresh
    - Initially AJAX (Asynchronous JavaScript and XML) where XML exchanged (e.g. homework site)
    - Now mostly ‘AJAJ’ where JSON exchanged instead

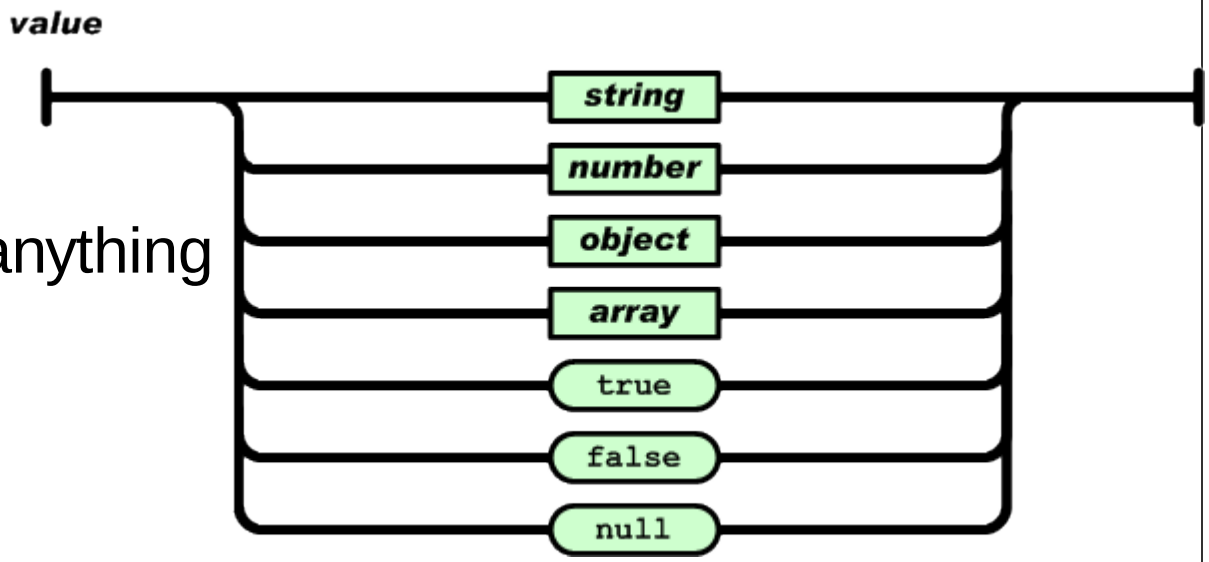
JSON objects

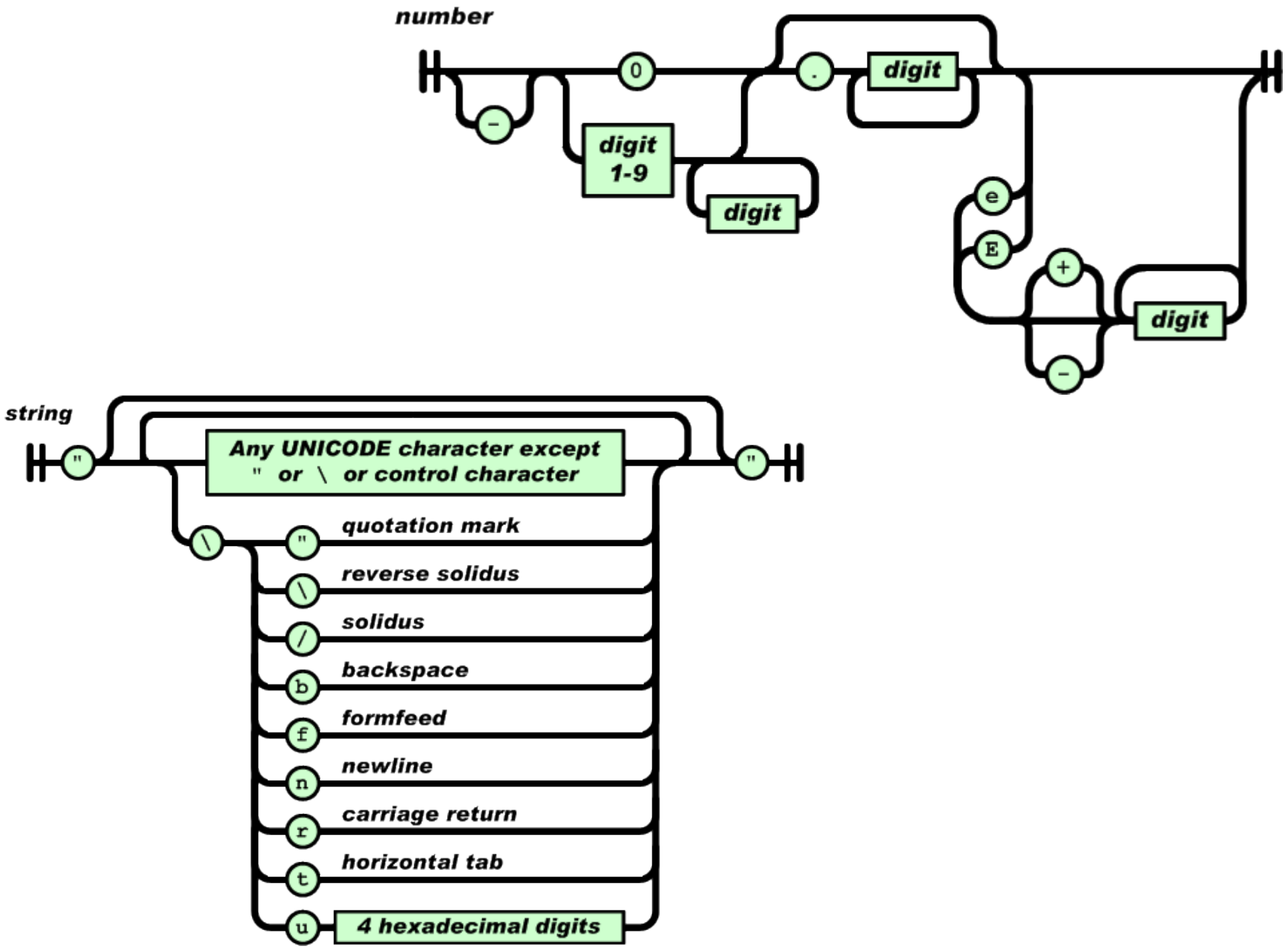
- Objects are unordered containers of key/value pairs
  - Keys are strings
  - Values are JSON values
  - Wrapped in **{ }**
    - **,** separates key/value pairs
    - **:** separates keys and values
- Parsed into local data structures as struct, record, hashtable, or dictionary



JSON Values

- Strings
  - Sequence of 0 or more Unicode characters wrapped in double quotes
- Numbers
  - Integer, Real, Scientific
  - No octal or hex
  - No **`NaN`** or **`Infinity`** (Uses **`null`** instead!)
- Booleans
  - **`true, false`**
- **`null`**
  - A value that isn't anything
- Objects
- Arrays



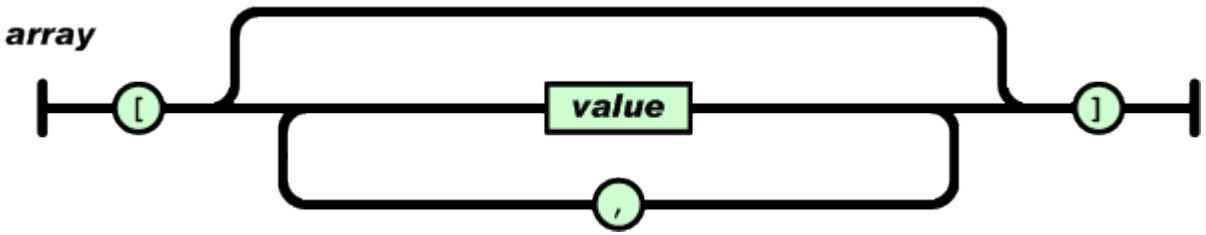

number
0
-
digit
1-9
digit
.
digit
e
E
+
-
digit
string
"
Any UNICODE character except
" or \ or control character
"
\
"
quotation mark
\
reverse solidus
/
solidus
b
backspace
f
formfeed
n
newline
r
carriage return
t
horizontal tab
u
4 hexadecimal digits

Arrays

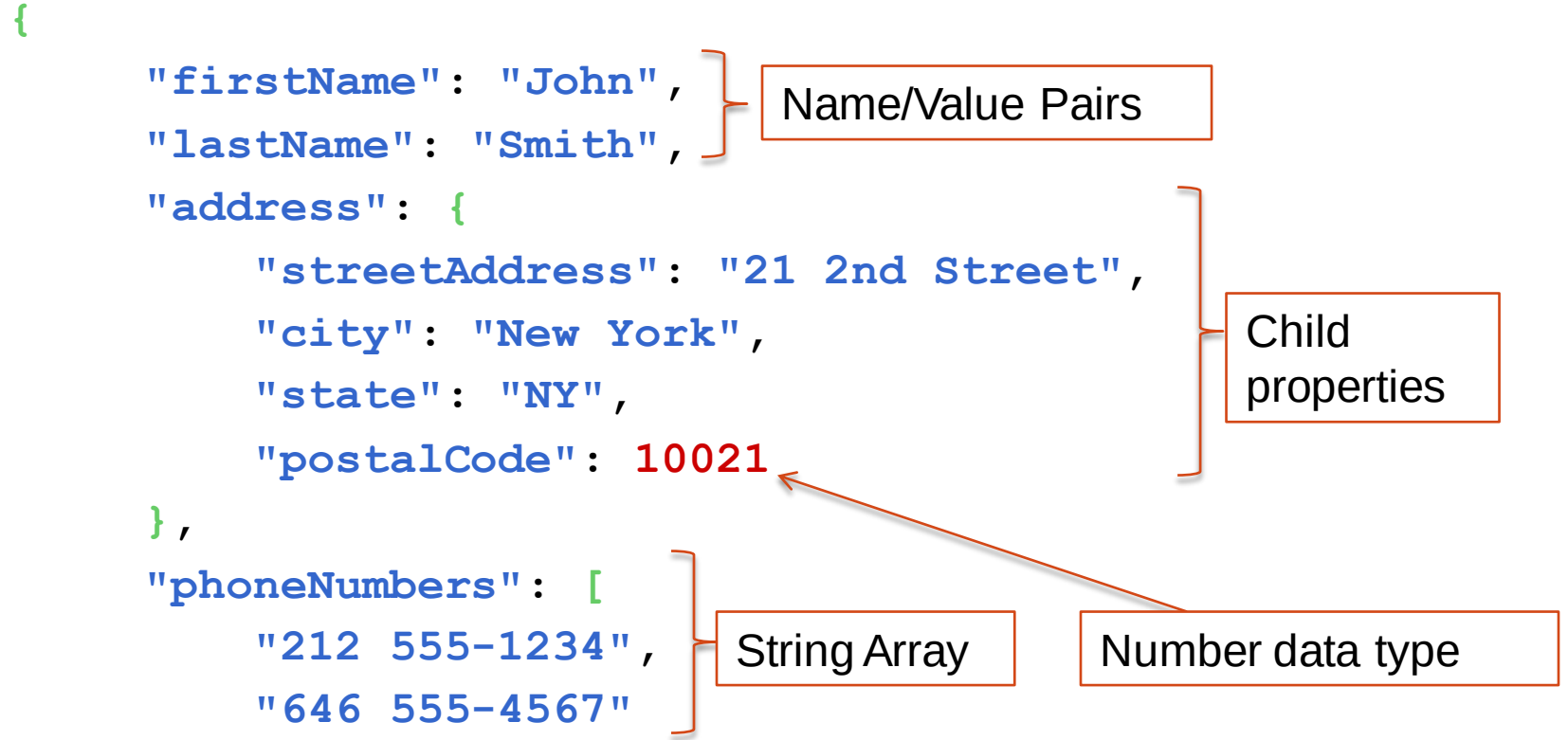
- Ordered sequences of values wrapped in `[]`
  - `,` separates values
- JSON does not specify indexing.
  - Array is parsed by web program language
  - Implementation can start array indexing at 0 or 1.

```
["Sunday", "Monday", "Tuesday", "Wednesday",
  "Thursday", "Friday", "Saturday"]
```

```
[
  [0, -1, 0],
  [1, 0, 0],
  [0, 0, 1]
]
```



JSON example



JSON example

- `stockfigher.io` stock order

```
{
  'account' : 'SWB1886430',
  'venue' : 'ETKBEX',
  'symbol' : 'EJYW',
  'price' : 8100,
  'qty' : 100,
  'direction' : 'buy',
  'orderType' : 'limit`
}
```

- Twitter direct message
  - https://dev.twitter.com/rest/reference/get/direct_messages

JSON in AJAX & JavaScript

- JSON often exchanged in JavaScript via XMLHttpRequest
  - Example: obtain as **`responseText`**, then parse it

```
responseText is '{ "name": "Jack B. Nimble", "at large":
true, "grade": "A", "format": { "type": "rect", "width":
1920, "height": 1080, "interlace": false, "framerate": 24
} }';

jsonobject = JSON.parse(responseText);
document.write("The object<br>");
document.write("name: ", jsonobject.name, "<br>");
document.write("grade: ", jsonobject.grade, "<br>");
document.write("format: ", jsonobject.format, "<br>");
```

JSON and avoiding eval()

- JSON uses JavaScript syntax to specify objects in a serialized manner
- Can either write a parser to pull out key:value pairs from JSON string or simply “evaluate” JSON string via **`eval`**
  - Parse version

```
jsonobject = JSON.parse(responseText);
```

  - Eval version

```
jsonobject = eval('(' + responseText + ')');
```

- Which one is safer?
- What if JSON object contained rogue JavaScript code?
  - Deserialization attacks
  - Mixing code and data

JSON security

- Deserialization attacks
  - Dependent upon trust
  - On client, not an issue
    - JSON data came from the same server that vended the page.
    - **`eval`** of the data is no less secure than the original html (assuming sent over HTTPS)

JSON security

- What about on the server (i.e. Node.js)?
- Is it OK to ever use **`eval`** to generate object from client?
  - No
- Can never trust the client
  - The client cannot be trusted
  - Server must validate everything the client tells it.
  - Run-time evaluation of untrusted input extremely dangerous!
  - Always use a parser on server running JavaScript (`nodejs`)
    - **`JSON.parse(string)`** instead of **`eval`**.

eval is evil

- Avoid using it in your web applications
  - PHP eval and deserialization issues (`picoCTF`, `natas`)
  - Python eval issues (`picoCTF`)
  - JavaScript eval issues (Pentestlab exercises, deserialization)

Security of JSON vs. XML

JSON	XML
Data Structure	Data Structure
No validation system	XSD
No namespaces	Has namespaces (can use multiples)
Parsing is just an eval Fast Security issues	Parsing requires XML document parsing using things like XPath
In JavaScript you can work with objects – runtime evaluation of types	In JavaScript you can work with strings – may require additional parsing
Security: eval() means that if the source is not trusted anything could be put into it. Libraries exist to make parsing safe(r)	Security: XML is text/parsing – not code execution.

JSON vs Javascript

- Double quotes for strings
- No functions intended to be allowed (text, no code unless someone does an eval)
- No circular references (text, no references)

Questions

- https://sayat.me/wu4f