Networking
Networking

- Networking is key and fundamental to today’s high-tech world
- You need access to the internet to do most of the interesting things you can do with computers
- There is a famous saying that goes - “the network IS the computer”.
  - In our cloud-centric world, that rings truer than ever
  - This is especially true with the Internet of Things
According to Gartner: there will be over 20 billion devices on the Internet of Things by 2020.

IPv4 ("old-style"): 
- 128.101.68.110
- $2^{32} = 4,294,967,296$ combinations (in practice, much fewer, because some are reserved)

IPv6 
- 2001:0db8:0000:0000:0000:ff00:0042:8329
- $2^{128} = 340,282,366,920,938,463,463,374,607,431,768,211,456$ combinations (enough to assign an IP address to each atom on the surface of the earth)
Basic concepts of a computer network:

- At present, the dominant network protocol is TCP/IP.

- One computer communicates with another computer by connecting via sockets (ports)
  - A port is a “door” through which the outside world can talk to a computer.
  - Typically about 65K ports available in a computer = 65K ways hackers can try to get into your computer!

- There is a set of predefined ports and usages
  - For example: Web servers - 80 (http/insecure), 443 (https/secure)
  - All others can be turned off - made unusable
Networking

Your app needs to be able to talk to networks based on both IPv4 and IPv6.
Networking with iOS

There are essentially three ways to do networking in iOS apps:

1. Use Apple’s **NSURLConnection framework**:
   - Apple’s original networking framework
   - archaic: most people do not use it anymore

2. Use Apple’s **NSURLSession framework**:
   - has for the most part replaced **NSURLConnection**
   - is MUCH easier to use than **NSURLConnection**
   - recommended by Apple
Networking with iOS

3. Use the third party AFNetworking framework:
   - very popular: used by a lot of developers
   - stable
   - prior to v2.0: built on top of NSURLConnection
   - v2.0 onward: built on top of NSURLSession
   - AFNetworking is actually the Objective-C version
   - AlamoFire is the Swift version

   Excellent tutorial:
   http://www.raywenderlich.com/85080/beginning-alamofire-tutorial
The `NSURLSession` class, and related classes, provide an API for downloading and uploading content over the internet.

In addition to downloading and uploading, there is built-in support for authentication and execution of background tasks (downloads).

`NSURLSession` natively supports the `data`, `file`, `ftp`, `http`, and `https` URL schemes.

There is also support for proxy servers and SOCKS gateways.
It is fully thread safe

Supports canceling, restarting / resuming, and suspending tasks

Can resume suspended / cancelled / failed downloads where they left off

You can create one or more sessions, each of which coordinates a group of related tasks
NSURLSessionConfiguration defines connection behaviors and policies for uploading and downloading data.

- The operations are asynchronous, and return data to your app by either:
  - Executing a completion handler block
  - Executing a session’s delegate methods
There are 4 types of `NSURLSession` policies:

- **Shared**: used for basic requests
- **Default**: allows you to receive data incrementally
- **Ephemeral**: does not write caches, cookies or credentials to the persistent store
- **Background**: allows upload / download in the background

Other configuration settings:

- Timeout Values
- Caching Policies
- Security Policies
- Background Transfers
- HTTP and proxy policies
NSURLSessionTask performs the actual work for retrieving data. You can create 3 types of tasks:

- **Data tasks** send and receive data using `NSData` objects
  - Intended for short requests and small amounts of data
  - Uses HTTP GET

- **Upload tasks** send and receive large amounts of data
  - Uses HTTP POST and PUT

- Download Tasks retrieve data in the form of a file
Basic Process to Use URLSession

Create a custom protocol to handle errors and return data.

protocol DataProtocol {

    func responseDataHandler(data: [return type])

    func responseErrorMessage(message: String)

}

Place this definition in a separate file, just like any other protocol / class.
Basic Process to Use URLSession

- **Create an** `NSURLSession`, the high-level session object
- **Create one or more** `NSURLSessionDataTask` objects.
  - These objects contains the code that do the work
  - Create them by calling `dataTaskWithRequest` or `dataTaskWithURL`
- **Include code in** `responseDataHandler` to process the received response
- **All tasks start in a** `suspended` state. You must start the task executing by calling `resume`.
Payloads refer to the data sent across the network. It is structured to be read on both ends. iOS can handle multiple payload formats:

- **JSON** (JavaScript Object Notation)
- **XML** (eXtensible Markup Language)
- **Property Lists**: Apple-specific format used for `info.plist` file
Tags defined by angle brackets

Content placed within tags

Tags can be nested

Nested elements are children of the enclosing elements
XML example


<?xml version="1.0" encoding="UTF-8" ?>
<data>
  <current_condition>
    <cloudcover>16</cloudcover>
    <humidity>59</humidity>
    <observation_time>09:09 PM</observation_time>
  </current_condition>
</data>
JSON

- Model for objects and arrays of data
- Presents hierarchical structures
- Easier to structure and parse than XML
- Objects are unordered name/value pairs:
  ```json
  {name1: value1, name2: value2}
  ```
- Arrays are ordered collections of values:
  ```json
  [value1, value2]
  ```
- Objects and arrays can be values
JSON example

http://www.json.org

```json
{
    "data": {
        "current_condition": [
            {
                "cloudcover": "16",
                "humidity": "59",
                "observation_time": "09:09 PM"
            }
        ]
    }
}
```

Note that JSON is more “space efficient” than XML.
info.plist is simply an XML file!

If you know XML, you can edit it directly; otherwise, you can edit it from within Xcode.
Apple’s documentation on Property Lists:

Xcode provides a user interface into the `info.plist` file.