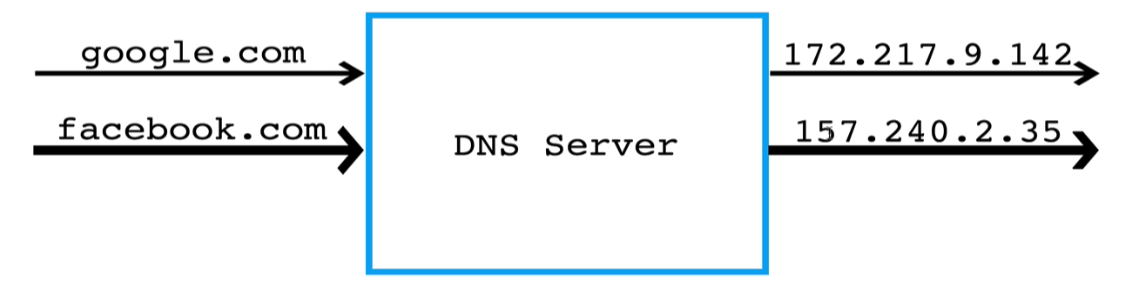
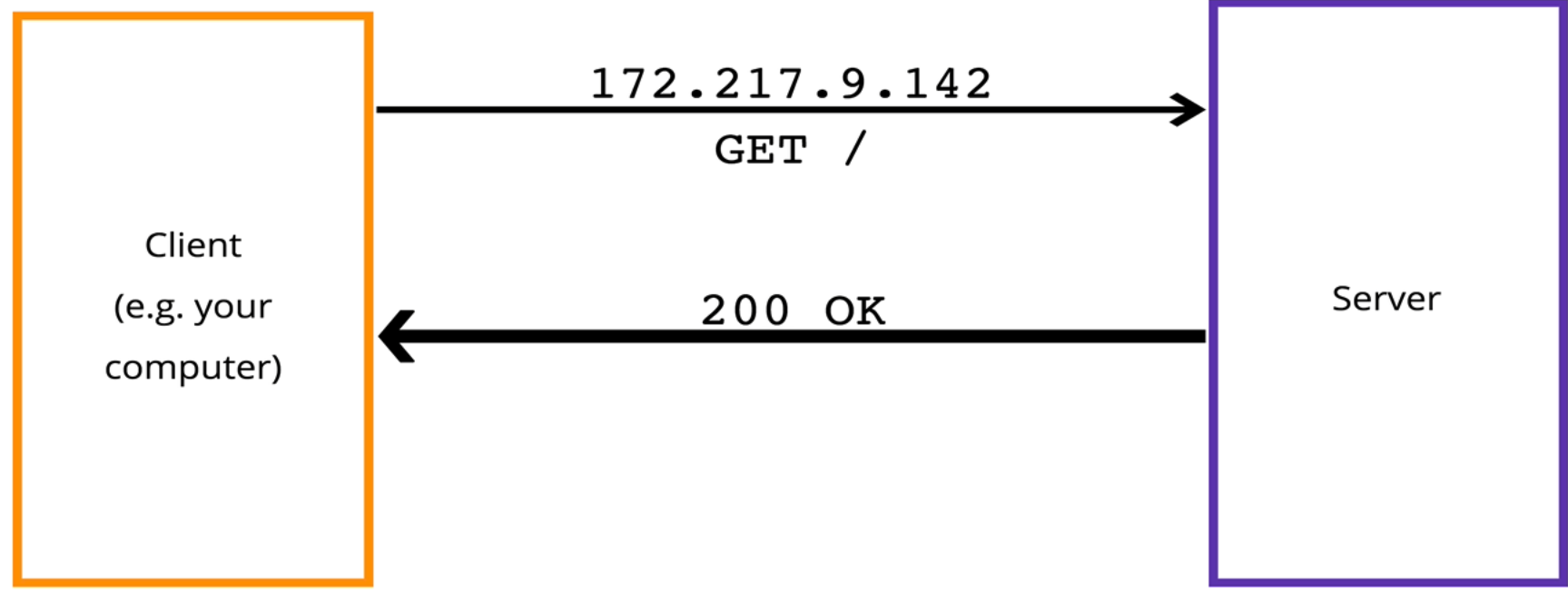
* The goal of this section is to learn how to write Python code that can communicate with websites and obtain data from APIs
  + We’ll do this using a package that makes HTTP requests
* HTTP Introduction
  + HTTP stands for **h**yper**t**ext **t**ransfer **p**rotocol
  + The internet is a bunch of computers connected together by a series of tubes/cables/wires
  + When you do an internet search, you are making a request:
    1. First, a DNS lookup occurs in order to find the correct address to send the request to
    2. Second, the computer (client) makes an HTTP request to a server
    3. Third, the server receives the request and processes it
    4. Fourth, the server compiles and issues a response

Steps 2-4 are known as the **request/response cycle**

* + The **DNS lookup** is like a phonebook for the internet
    - It takes domain names and turns them into IP addresses that we actually send the request to



* + Requests and responses
    - The client computer sends a request (there are different types) to an IP address
    - The server processes a request, oftentimes by communicating with a database, and compiles the response
    - Then the server will respond with the page you are looking for
      * When this doesn’t work, you’ll get one of many status errors (404 error, page has moved error, page does not exist error, etc.)
      * Regardless of whether the requests succeeds or fails, a response is sent back in HTML, which the browser constructs as a viewable webpage



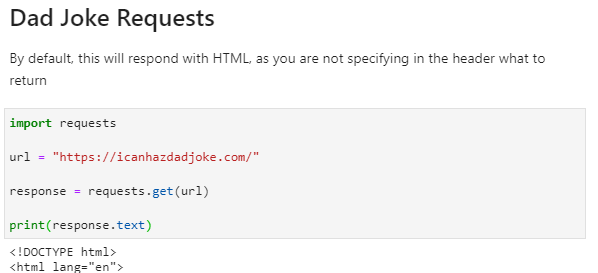
* + HTTP **headers** are contained within requests and responses
    - These provide additional information about the request or response
    - Request headers
      * **Accept** - acceptable content types for responses (e.g. html, json, xml)
      * **Cache-control** – specifying caching behavior
      * **User-Agent** – information about the software used to make the request
    - Response headers
      * Access-Control-Allow-Origin – specify domains that can make requests
      * Allowed – HTTP verbs that are allowed in requests
  + Response **Status Codes** are attached to all responses and provide additional information about the response
    - 2xx – Success
    - 3xx – Redirect
    - 4xx – Client Error (this is your fault, not the server’s fault)
    - 5xx – Server Error (this is NOT your fault, something happens on the backend)
* HTTP Verbs
  + It seems like any client request should work the same way, but that’s not the case
  + The two most fundamental request types are **get** and **post**
    - GET request
      * Useful for retrieving data
      * Think of it as browsing the web
      * Data is passed in a query string
      * Should have no “side-effects”; you are not sending any data that is going to get added somewhere
      * Can be cached
      * Can be bookmarked
    - Post request
      * Useful for writing data
      * Think of it as posting a photo or a comment on Facebook
      * Data is passed in request body
      * This may have “side-effects”, in that the data you are sending may alter something in the backend
      * Not cached
      * Cannot be bookmarked
* What are APIs?
  + API – Application Programming Interface
    - All that really means is that it’s a version of a website intended for computers to talk between computers or between code
    - Contrast this with human-viewable websites
  + APIs allow you to get data from another application without needed to understand how the application works
    - Many well-known websites have APIs were data can be grabbed from
    - Examples: GitHub, Spotify, Google all have APIs
  + The idea here will be to use Python to grab data from APIs that it can then immediately do stuff with
    - Oftentimes you will want to assign this data to a variable, then use it!
    - JSON – JavaScript object notation, so-named because object is a data format in JavaScript
    - XML
* The **requests** module is one of the most popular modules for making HTTP requests
  + <https://realpython.com/python-requests/#the-get-request>
  + Must be installed using pip in the terminal
  + It is very useful for web scraping/crawling, grabbing data from other APIs, etc.
  + Requests can be used to send a request to a normal website, as shown below. This isn’t particularly useful for us



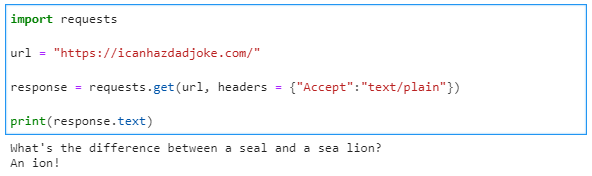
* + We are able to specify headers with the requests module in order to get a response that is friendlier to handle with Python
    - We can put in headers as a dictionary (keyword argument), indicating the header accepts, or the type of value that we will accept back



* + - First, another example with no header that return unreadable HTML



* + - We can edit this request to return **plain text** instead of HTML. However, keep in mind that most sites are NOT set up to return plain text.



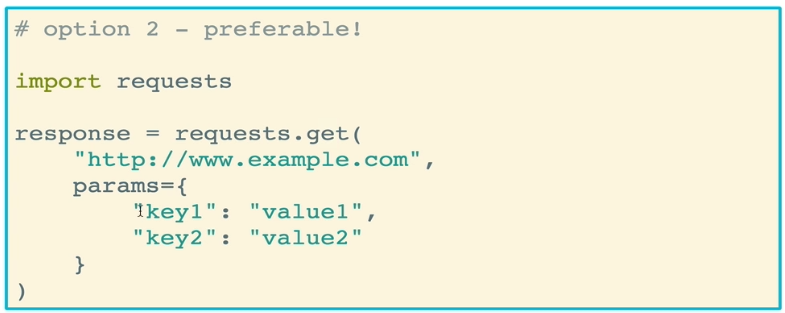
* + **JSON** is the BEST form to request data, if the website is set up to return it
    - Python can very easily change JSON into valid Python code
    - All you need to do is use the .json() method on the response
    - In the example below, the first response is JSON (a string), while the second is Python (a dictionary), even though they look very similar



* A **query string** is a way to pass data to a server as part of a get request. This allows you to provide the server with more information about your request, with the intent being that when the server has a better idea of your request, the response will be closer to what you actually want
  + Take a look at your next Google search. The resulting URL will be a base URL, some gibberish, and a bunch of keywords
  + Many times an application will need the user to provide what they want to search for
  + So what you do is pass what you’re looking for into a query string
  + One way to do this is by hardcoding the key-value pairs into the URL, like so



* + The preferred way, however, is by using the params keyword argument in requests.get() to pass in your key-value pairs



* + Example: Dad joke request with terms

