
API

# It is all about communication

Whenever two pieces of code communicate they do so with a prescribed format.

The two pieces of code need to agree on:

- What is the input that needs to be sent?
- What is the format of the input?
- What is the name/location/identity of the code to be run?
- What is the output that will be returned?
- What is the format of the output?

All of this together describes an Application Programming Interface, or API.

The **secret** is that as we've been writing methods we've already been writing and *using* APIs!

# Simple example

Let's look at a simple example:

```
int AddTwoNumbers(int firstNumber, int secondNumber)
{
  return firstNumber + secondNumber;
}
```

```
int AddTwoNumbers(int firstNumber, int secondNumber)
{
  return firstNumber + secondNumber;
}
```

## Let's apply our list:

| | |
|---|---|
| What is the input that needs to be sent? | `firstNumber and secondNumber` |
| What is the format of the input? | `firstNumber is an integer and secondNumber is an integer` |
| What is the name/location/identity of the code to run? | `The method is called AddTwoNumbers` |
| What is the output that will be returned? | `The sum of the two numbers we gave` |
| What is the format of the output? | `A single int` |

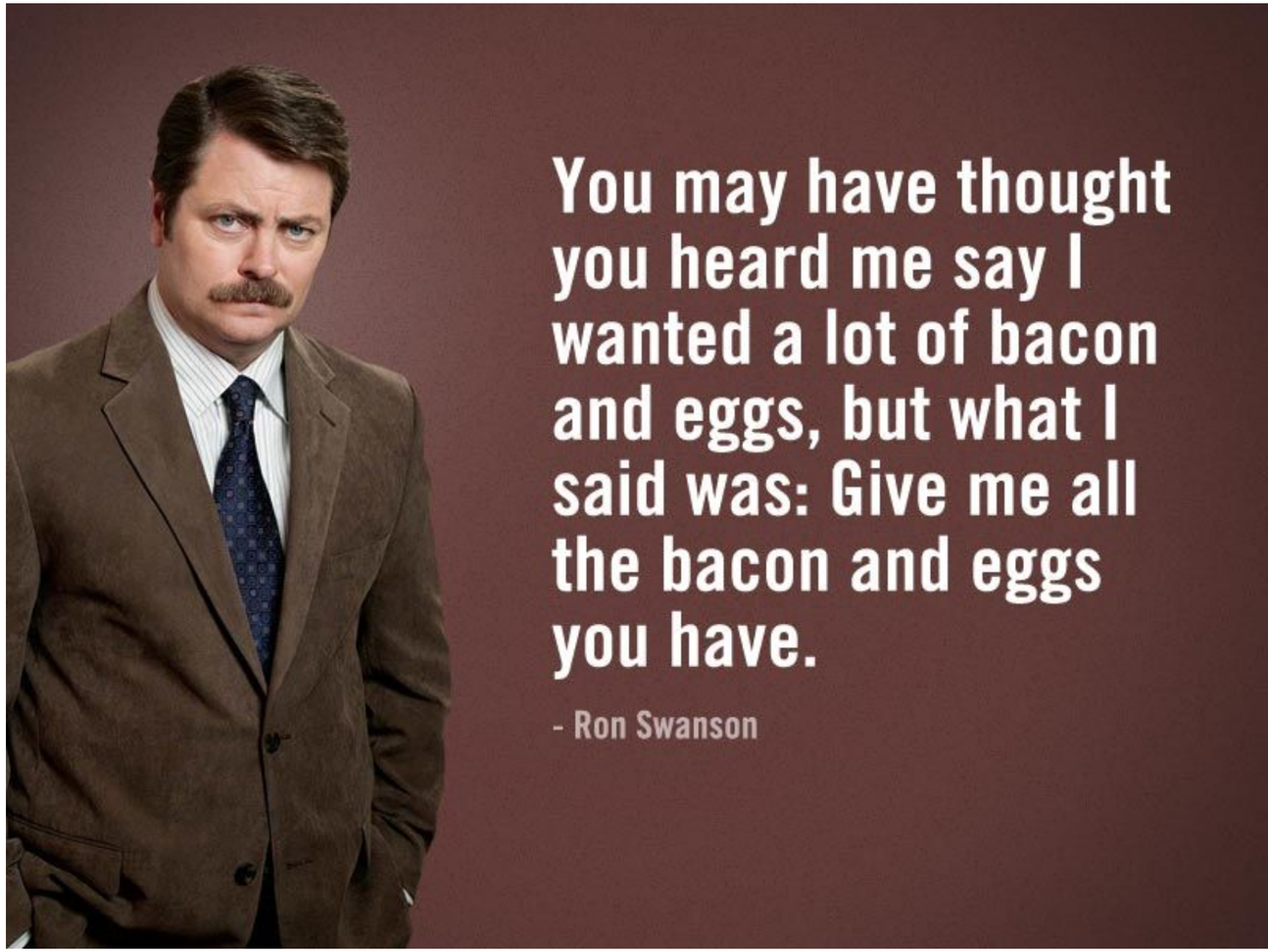
# Remote APIs

The most interesting use of APIs is when we can use an existing API that we do not need to incorporate directly into our code.

These APIs are typically hosted on other computers, maintained by others, and provided to us for free or with a cost.

We will typically access these APIs over the internet and thus the answer to our five questions gets slightly more complicated.

# Using our first simple API



https://github.com/jamesseanwright/ron-swanson-quotes

We'll be looking at all the different options, but let's take a look at the simplest option, one that just gives us back a single quote.

| Question | Answer |
| --- | --- |
| What is the input that needs to be sent? | None |
| What is the format of the input? | None |
| What is the name/location/identity of the code to be run? | `http://ron-swanson-quotes.herokuapp.com/v2/quotes` |
| What is the output that will be returned? | `A list of quotes from Ron` |
| What is the format of the output? | `A JSON array of strings` |

# Tools

The first tool we will look at is `httpie`

Let's run the `http` command (for `httpie`) and supply it the location of our API.

```
http --verbose http://ron-swanson-quotes.herokuapp.com/v2/quotes
```

```
GET /v2/quotes HTTP/1.1
Accept: */*
Accept-Encoding: gzip, deflate
Connection: keep-alive
Host: ron-swanson-quotes.herokuapp.com
User-Agent: HTTPie/2.0.0



HTTP/1.1 200 OK
Access-Control-Allow-Origin: *
Connection: keep-alive
Content-Length: 56
Content-Type: application/json; charset=utf-8
Date: Wed, 22 Apr 2020 15:24:46 GMT
Etag: W/"38-8d39bb7c"
Server: Cowboy
Via: 1.1 vegur
X-Powered-By: Express

[
    "Crying: acceptable at funerals and the Grand Canyon."
]
```

# Break it down

# An analogy

Let's make an analogy to our understanding of methods. This API above is similar to a method in C# named:

```
List<string> GetOneQuote()
{
  // Body of code here
}
```

and our `http` command line is like running this code with

```
var result = GetOneQuote()
```

# Ok, but what about if we need to send the API some input?

If we need to send the API input there are several ways of doing so.

We will investigate some here and visit the rest when we discuss how to create, update, and delete data.

# In the URL itself

The first way is to provide input as part of the URL itself.

Looking at the documentation for the Ron Swanson API we see that if we tack on /<count> where <count> is a number we can receive more than one quote.

*This is called a URL parameter. That is, the parameter is part of the URL itself.*

# URL parameters

To use this we might do:

```
http --verbose http://ron-swanson-quotes.herokuapp.com/v2/quotes/5
```

# The answers to our API questions are now:

| Question | Answer |
| --- | --- |
| What is the input that needs to be sent? | `Count of quotes` |
| What is the format of the input? | `as part of the URL` |
| What is the name/location/identity of the code to be run? | `http://ron-swanson-quotes.herokuapp.com/v2/quotes/<count>` |
| What is the output that will be returned? | `A list of quotes from Ron` |
| What is the format of the output? | `A JSON array (of length 5) of strings` |

And the result of calling the API will look like:

```
[
    "Breakfast food can serve many purposes.",
    "Children are terrible artists and artists are crooks.",
    "Crying: acceptable at funerals and the Grand Canyon.",
    "Keep your tears in your eyes where they belong.",
    "Friends: one to three is sufficient."
]
```

# Query Parameters

These work similar to the URL parameters we've seen but they go *after* the URL and a ? character.

You may have seen this when looking at the URL bar for a google search. Your URL might look something like this: `https://www.google.com/search?q=API`.

Try this: `https://google.com?q=Breakfast`

# Examples

The Ron Swanson API does not use this mode for specifying its inputs but if it had, those queries would look like this:

```
http://ron-swanson-quotes.herokupapp.com/v2/quotes?count=5
http://ron-swanson-quotes.herokupapp.com/v2/quotes?term=Breakfast
```

# Verbs

In addition to the URL we send, and any parameters we send, the HTTP protocol also specifies a `verb`, the *kind* of request this is.

Every request you make by typing a URL into your browser is known as a `GET` request.

# GET

A `GET` request indicates to the remote server that we wish to fetch information and we are not *sending* it any data that it needs to keep. You'll see that these kinds of requests fit that style:

- Give me a Ron Swanson quote
- Give me 12 Ron Swanson quotes
- Give me any Ron Swanson quotes where he mentions work

# Not a GET

However it would *not* fit a style of:

- Here is a new Ron Swanson quote I want you to keep around.
- Please remove the 4th Ron Swanson quote.
- There is a misspelling in the 12th Ron Swanson quote, here is the corrected quote.

All of those requests modify information and thus a `GET` request is not

# PUT, POST, DELETE

These are the next most common verbs. They are typically used to CREATE, UPDATE, and DELETE information.

If we need to supply additional information, say a `POST` request to create data, we will put the information in the body of the request. We will see some examples when we try a few APIs that allow us to create and modify data.

# Insomnia

# One List

The `One List` API is a sample API made by SDG for learning how to consume APIs.

The One List API is an API for managing a list of todo items.

The examples below are based on the online documention of the One List API.

Each URL we investigate will have an access_token and this will allow us to maintain different lists for different users.

# `GET /items?access_token={access_token}`

This API URL will get the list of all todo items. If we run `http` to fetch data from the API like:

```
http "https://one-list-api.herokuapp.com/items?access_token=illustriousvoyage"
```

# Create a new item

We are now going to see our first API usage where we send data to the API to be stored/updated/deleted.

We use the URL `POST /items?access_token={access_token}` to **CREATE** an item.

Notice for the first time we are using a different verb, the `POST`.

The `POST` verb cannot be generated from the URL bar, but can from a web form, which makes sense since we often use forms to create data, as well as from code or the command line.

# Sending data in the body

The JSON we will send to the server should look like this:

```
{
  "item": {
    "text": "New text here",
    "complete": false
  }
}
```

# Update an item

To update an item with this API we use a very similar structure as the `POST` used to create, but this time we change the VERB to `PUT` and include the `id` in the URL:

```
PUT /items/{id}?access_token={access_token}
```

The body of the method has the same structure as the `POST`, that is the JSON object containing `text` and `complete` keys.

# Deleting items

To delete an item we switch to using the `DELETE` verb and specify the item in the URL.

```
DELETE /items/{id}?access_token={access_token}
```

However, we do not specify a body since no additional information is needed to find the item and remove it.

# Review our API questions

Each API URL (known as an *end point*) has different answers to our *API questions*

| | |
|---|---|
| What is the input that needs to be sent? | `Any URL parameters, like {id} and query parameters like {access_token} as well as the body of the request if required` |
| What is the format of the input? | `The body will be in JSON format` |
| What is the name/location/identity of the code to be run? | `This will be the URL and the specific verb to use` |
| What is the output that will be returned? | `The response body and the response code` |
| What is the format of the output? | `The body will be in JSON and the response code will be a number` |