CS 1: Introduction to Computer Programming — Fall 2025

Mini-Project: Searching for Information - Techniques and Trade-offs!

In this mini-project, you (and up to one partner) will compare and contrast various information sources. The main goal is to come away with your own thesis on when various approaches are good or bad and why.

Some Motivating Examples

As you go through this mini-project, we will ask you to categorize and conceptualize queries that you might need to know the answers to for various reasons. Here are some (fairly disparate) examples of the sorts of questions we might be trying to answer:

- What paper does the following quote come from?

  > "To test the efficacy of constructivist versus exposition-centered course designs, we focused on the design of class sessions—as opposed to laboratories, homework assignments, or other exercises."

- What is George Washington's birthday?
- How do I find the minimum of a list in python?
- Is a banana a fruit?
- What are current soybean prices?
- What is the significance of 67?

Two Approaches (both alike in dignity)

In the modern world, we've come to accept the internet as a common source of information. More recently, that definition has expanded to Large Language Models (LLMs). While there are many sources of information (and you will, in fact, have to choose your own later in this mini-project), the two we're most interested in comparing are Google and ChatGPT. As you progress through this mini-project, one of the main things to keep in mind is that the **quality** and **correctness** of the responses is not the only criterion to worry about. Some criteria we've identified that might be interesting to think about are the following:

- How much energy does making the query take? Does a simple click of a button have larger implications?
- How much context does the answer give? Enough to understand? Enough to be dangerous? Is the answer a sound-byte that sounds good but doesn't actually have any nuance?
- How much time is the asker spending sifting through useful (or useless!) words and information. How long does the asker have to wait before even receiving any response at all?
- How accurate is the answer? If events are changing in real time does the answer reflect that? Is the answer itself even correct to begin with? Is some semblance of evidence provided to back up the answer?

Dear student, it's important to understand that we do not have a "correct" answer or even a bias toward one option over the other. Our goal is for you to think critically as you try to find information in the future.

Automating the Boring Work

In a world where everything is programs, it would be convenient for search results to be returned in a form that's easy for a program to consume as input. In other words, visual websites can be great for humans, but programs would almost certainly prefer links and text.

Application Programming Interfaces (APIs)

Enter APIs. An API is a library (program written by someone else) that allows you to access an external resource (e.g., Google or ChatGPT) with your program. They provide a way for you to give inputs and receive outputs from web-based services as if you were just calling a normal function!

SerpAPI: An API for Google Search

SerpAPI is an API for Google. It allows you to execute searches and get back search results. For our purposes, all we really care about are the links in the search results themselves. In this mini-project, we will be using it to fetch results from Google and compare them to results from ChatGPT.

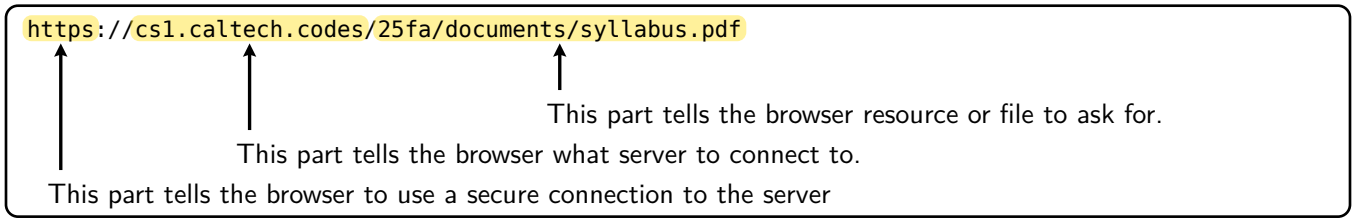
Step 1: Getting Your API Key

To use SerpAPI, you'll need to register and generate an "API key" (the API equivalent of a password). To do so, follow the following steps:

- Go to `https://serpapi.com/users/sign_in`.
- Login to SerpAPI using your **Caltech Google Account**.
- You may or may not have to verify your email. The verification email might take a bit – we're not sure why.
- Register your phone number to gain access to the API.
- Click "subscribe" to submit.
- On the resulting page, you should see a string of random-looking characters labeled "API Key". Save this string somewhere safe and **do not share it with anyone**.

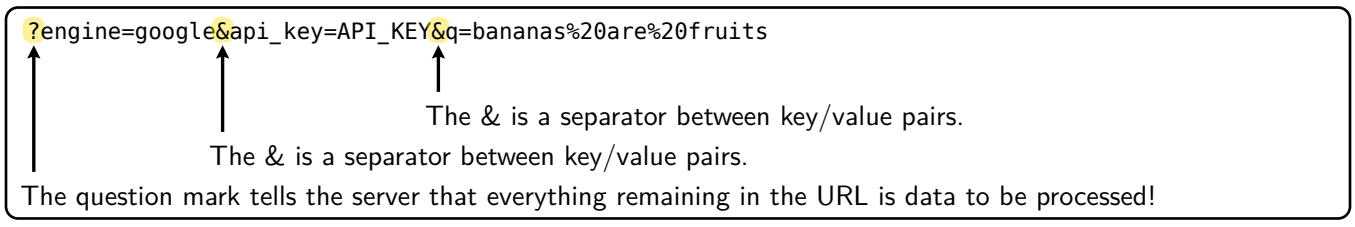
Step 2: Making an API Request with Your Browser

When you go to a website in your browser, you are implicitly making a "web request" to a server. The string you type into the address bar of the browser is called a URL. For example, to get the syllabus from the CS 1 website, you'd use the following URL:



An API is really just "yet another server". The only difference in the request is that the URL has a "query string" at the end (started by a "?"). For example, here is a URL that makes a request to the Google API:

`https://serpapi.com/search.json?engine=google&api_key=API_KEY&q=bananas%20are%20fruits`



When the server receives the request, it turns the query string into a dictionary with the following contents:

```
{
  "engine": "google",
  "api_key": "API_KEY",
  "q": "bananas are fruits"
}
```

There are two very important things to note about this final dictionary:

1) "API_KEY" was interpreted literally! You need to put your actual `API_KEY` in the URL directly to make this work.
2) "`bananas%20are%20fruits`" was read as "bananas are fruits". This is because `%20` is a "URL encoding" of space. There are certain characters (e.g., " ") that cannot appear in URLs. We have written a function for you

that converts a regular string (e.g., "bananas are fruits") into the URL encoded version (e.g., "bananas%20are%20fruits"):

```
def quote_for_url(s: str) -> str:
  import urllib.parse
  return urllib.parse.quote_plus(s)
```

Please use this program wherever you are passing a query to a URL throughout this mini-project!

Finally, you are ready to make your first API request…by pasting it right into your browser! Replacing `API_KEY` with the weird string you got from `serpapi.com` in step 1, copy/paste the following URL into your browser:

`https://serpapi.com/search.json?engine=google&q=cs1%20caltech&api_key=API_KEY`

The page you get back should have a bunch of text inside brackets and curly braces. If this is the case, success! If not, please get help from a member of course staff.

Step 3: Making an API Request with Python

Now that we know we can make API requests to Google, it's time to have Python do it for us. We've actually already seen the library we need (called `requests`) in class (on day one, even!), but now it's time to understand it. For our purposes, we need to know two main things:

1) We must import the library using `import requests` before using it.
2) The only function we care about is called `get` and looks like the following: `requests.get(URL).json()`; this function will return a dictionary with the results from the API.

Step 3.5: Getting the first link on the page

Once we've used `requests` to get back **all** the results on the page, we want to find the link corresponding to the first result. The results are stored as a list inside the returned dictionary and can be found at the key `organic_results`. Each result inside `organic_results` **is itself** a dictionary which has a `link` key. Put everything together to write a short function called `google` which returns the link corresponding to the first result for the search `query`:

```
def google(query: str, api_key: str) -> str:
  url = ...
  data = requests.get(url + quote_for_url(query)).json()
  return ...
```

An API for ChatGPT

Now that we've successfully used a Google API, it's time to move to bigger game: ChatGPT!

The ChatGPT API works pretty similarly to the SerpAPI one with some small caveats.

Step 1: Getting Your API Key

Like before, you will need an API Key. You can find `your personal ChatGPT API key here`.

Step 2: Making an API Request with Python

The function you wrote to use SerpAPI is actually quite similar to the function you need to write to use the ChatGPT API. Consider the following scaffold for a `chatgpt` function:

```
def chatgpt(query: str, api_key: str) -> str:
  url = "https://grinch.caltech.edu/chatgpt-api"
  data = requests.post(url, json=data, headers=headers).json()
  return ...
```

We've highlighted the main three differences; namely, the "post", the "data", and the "headers". Let's go through each piece one-by-one:

GET Requests vs. POST Requests

A `GET` request (achieved by the `requests.get` function used in the SerpAPI part) is intended to ask the server for a deterministic answer to a query. Typically, GET requests act like functions that always return the same answer for a provided input.

In contrast, a `POST` request (achieved by the `requests.post` function used in this part) is intended to ask a server to **compute** and/or **store** something based on an input. Indeed, this is exactly what making a request to ChatGPT is (it is called "generative AI", after all).

Query Strings vs. Data

In a `GET` request, we provide the input as a "query string" (the thing that starts with the `?`). In a `POST` request, we provide the input as a dictionary. For ChatGPT, the data should be passed in the `json` argument and look something like this:

```
data = {
  "model": "gpt-4",
  "messages": [{"role": "user", "content": query}]
}
```

Authorization Headers

When using `SerpAPI`, we put the API Key in as part of the query string. More advanced APIs (like the ChatGPT one) place the API key in a "header" which is passed in an encrypted form. For the ChatGPT API, we will be using a "bearer token" which will look like a dictionary that can be passed into the `headers` argument as follows:

```
headers = {'Authorization': 'Bearer ' + api_key}
```

Please note that the ChatGPT API will only let you make a single request every TEN SECONDS. If you try to make multiple requests within that time period, the API will return an error.

Step 3.5: Getting ChatGPT's response

If all goes well, we will get back a dictionary from the ChatGPT API. To get the actual content, we'll need to index into this dictionary in the following way:

- The top-level dictionary will have a key called `choices`
- We want the first choice in the resulting list
- We want the `message` key from that choice.
- We want the `content` key from that message.

Now, the Fun Part!

Now that we have functions to access the ChatGPT and Google APIs, we can write a short python program to be able to ask questions to both to compare their answers, using something like the below:

```
query = None
while query != "q":
  query = input("What would you like to ask?")
  print("ChatGPT:", chatgpt(query))
  print("Google Link:", google(query))
```

Earlier in this document, we've provided you with some motivating examples and some food-for-thought about how to compare the two engines.

Dear student, your actual task on this mini-project is to form a thesis that categorizes when ChatGPT is a good resource, when Google is a good resource, and when something else entirely might be the best resource. The Experiment portion of the DUE will be a (friendly) debate between your group and another group about justifying your theses!

This Miniproject **does** have a mandatory DUE, and you and any partner you worked with must sign up for a **single** slot together that you can both attend. Please show up to the DUE with the following ready:

1) Your code that you wrote in the previous sections of this mini-project. You will not be turning in this code on gitlab or anywhere else, but you will have to Demo it for us during the DUE.
2) Both you and your partner should bring laptops (**both** with the miniproject code) to the DUE. Make sure that both you AND your partner fully understand the project. **You will be asked to complete some of the parts of the DUE individually**.
3) A written thesis as per the previous red box that you are ready to defend and argue. Write as much as necessary to prepare for a substantive discussion with another group. Please don't write more than a paragraph.
4) A question you are not able to immediately answer or resolve about these trade-offs.

Please make sure to `sign up for a DUE` by clicking any of the `green DUE Sign-Up` links `in this sentence`.