Parallelism- multiple things or operations happening simultaneously

The amount of cores in CPU denotes the number of parallel operations that can happen in an instant.

Thread- one operation that runs on a core of a CPU

Concurrency– two threads running at the same time or multiple tasks which start,run and complete in overlapping time periods , in no specific order.

*Concurrency means executing multiple tasks at the same time but not necessarily simultaneously.* Concurrency means that an application **is making progress** on more than one task at the same time (concurrently). Well, if the computer only has one CPU the application may not make progress on more than one task at exactly the same time, but more than one task is being processed at a time inside the application. It does not completely finish one task before it begins the next.

Parallelism means that an application splits its tasks up into smaller subtasks which can be processed in parallel, for instance on multiple CPUs at the exact same time.

in single-core CPU, you may get concurrency but NOT parallelism.

Concurrency is about**dealing with lots of things** at once. Parallelism is about **doing lots of things at once**.

An application can be concurrent — but not parallel, which means that it processes more than one task at the same time, but no two tasks are executing at the same time instant. More like, doing bit of a task at a time rather than completing it there and then. Again shifting back to the task after doing a bit of another task.

An application can be parallel — but not concurrent, which means that it processes multiple sub-tasks of a task in multi-core CPU at the same time.

An application can be neither parallel — nor concurrent, which means that it processes all tasks one at a time, sequentially. More like finishing a task completely and then moving to another task; one after the other.

Threading is one of the most well-known approaches to attaining Python concurrency and parallelism. Threading is a feature usually provided by the operating system. Threads are lighter than processes, and share the same memory space.

Using threading to divide up the work greatly decreases the execution time as the operation gets divided among the different cores present in the CPU rather than having all of it done in one CORE. More like shifting from concurrency to parallelism.

BASICS OF STORAGE AND SESSION

IndexedDB is a low-level API for client-side storage of significant amounts of structured data, including files/blobs. This API uses indexes to enable high-performance searches of this data. While [Web Storage](https://developer.mozilla.org/en-US/docs/Web/API/Web_Storage_API) is useful for storing smaller amounts of data, it is less useful for storing larger amounts of structured data

Difference between CACHE AND COOKIES 🡪

Cookies expire after some time, but cache is kept in the client’s machine until they are removed manually by the user.

The main difference between Cache and Cookie is that, Cache is used to store online page resources during a browser for the long run purpose or to decrease the loading time. On the other hand, cookies are employed to store user choices such as browsing session to trace the user preferences.

<https://www.geeksforgeeks.org/difference-between-cache-and-cookies/#:~:text=The%20main%20difference%20between%20Cache,to%20trace%20the%20user%20preferences>.

WEB CACHING

Web caching works by caching the HTTP responses for requests according to certain rules. Subsequent requests for cached content can then be fulfilled from a cache closer to the user instead of sending the request all the way back to the web server.

* **Cache hit ratio**: A cache’s effectiveness is measured in terms of its cache hit ratio or hit rate. This is a ratio of the requests able to be retrieved from a cache to the total requests made. A high cache hit ratio means that a high percentage of the content was able to be retrieved from the cache. This is usually the desired outcome for most administrators.
* **Freshness**: Freshness is a term used to describe whether an item within a cache is still considered a candidate to serve to a client. Content in a cache will only be used to respond if it is within the freshness time frame specified by the caching policy.
* **Important header of HTTP 🡪 (well defined ) Cache-Control**: This is the more modern replacement for the Expires header. It is well supported and implements a much more flexible design. In almost all cases, this is preferable to Expires, but it may not hurt to set both values. We will discuss the specifics of the options you can set with Cache-Control a bit later.
* The no-store option supersedes the no-cache if both are present. For responses to unauthenticated requests, public is implied. For responses to authenticated requests, private is implied. These can be overridden by including the opposite option in the Cache-Control header.

Certain content lends itself more readily to caching than others. Some very cache-friendly content for most sites are:

* Logos and brand images
* Non-rotating images in general (navigation icons, for example)
* Style sheets
* General Javascript files
* Downloadable Content
* Media Files

**Locations Where Web Content Is Cached**

Content can be cached at many different points throughout the delivery chain:

* **Browser cache**: Web browsers themselves maintain a small cache. Typically, the browser sets a policy that dictates the most important items to cache. This may be user-specific content or content deemed expensive to download and likely to be requested again.
* **Intermediary caching proxies**: Any server in between the client and your infrastructure can cache certain content as desired. These caches may be maintained by ISPs or other independent parties.
* **Reverse Cache**: Your server infrastructure can implement its own cache for backend services. This way, content can be served from the point-of-contact instead of hitting backend servers on each request

The majority of caching behavior is determined by the caching policy, which is set by the content owner. These policies are mainly articulated through the use of specific HTTP headers.

* **(!!! Most important and has other attributes) Cache-Control**: This is the more modern replacement for the Expires header. It is well supported and implements a much more flexible design. In almost all cases, this is preferable to Expires, but it may not hurt to set both values. We will discuss the specifics of the options you can set with Cache-Control a bit later.

<https://www.digitalocean.com/community/tutorials/web-caching-basics-terminology-http-headers-and-caching-strategies>

Whenever you connect to a website, the web server sends something called a Hypertext Transfer Protocol (HTTP) header to your local computer. This header contains information about the last time the web page was updated. This information is stored on your local computer for the web server to access when you next browse that particular page. What the HTTP header does is allow the web server to know that you already have part of the web page on your computer so it does not have to send it to you again; it can load that part of the web page from your local hard drive, both speeding the page rendering for you and reducing the load on the server.

The key is to strike a balance that favors aggressive caching where possible while leaving opportunities to invalidate entries in the future when changes are made. Your site will likely have a combination of:

* Aggressively cached items
* Cached items with a short freshness time and the ability to re-validate
* Items that should not be cached at all

The goal is to move content into the first categories when possible while maintaining an acceptable level of accuracy.

**Conclusion**

Taking the time to ensure that your site has proper caching policies in place can have a significant impact on your site. Caching allows you to cut down on the bandwidth costs associated with serving the same content repeatedly. Your server will also be able to handle a greater amount of traffic with the same hardware. Perhaps most importantly, clients will have a faster experience on your site, which may lead them to return more frequently. While effective web caching is not a silver bullet, setting up appropriate caching policies can give you measurable gains with minimal work.

WEB STORAGE

The **Web Storage API** provides mechanisms by which browsers can store key/value pairs, in a much more intuitive fashion than using [cookies](https://developer.mozilla.org/en-US/docs/Glossary/cookie). Cookies let you store a small amount of data (nearly 4KB), the web storage allows you to store up to 5MB of data.

<https://www.tutorialrepublic.com/html-tutorial/html5-web-storage.php>

[**Web Storage concepts and usage**](https://developer.mozilla.org/en-US/docs/Web/API/Web_Storage_API#Web_Storage_concepts_and_usage)

The two mechanisms within Web Storage are as follows:

* sessionStorage maintains a separate storage area for each given origin that's available for the duration of the page session (as long as the browser is open, including page reloads and restores)
  + Stores data only for a session, meaning that the data is stored until the browser (or tab) is closed.
  + Data is never transferred to the server.
  + Storage limit is larger than a cookie (at most 5MB).
  + The sessionStorage object work in the same way as localStorage, except that it stores the data only for one session i.e. the data remains until the user closes that window or tab.
* localStorage does the same thing, but persists even when the browser is closed and reopened.
  + Stores data with no expiration date, and gets cleared only through JavaScript, or clearing the Browser cache / Locally Stored Data.
  + Storage limit is the maximum amongst the three.

As stated earlier, the localStorage object stores the data with no expiration date. Each piece of data is stored in a key/value pair

These mechanisms are available via the [Window.sessionStorage](https://developer.mozilla.org/en-US/docs/Web/API/Window/sessionStorage) and [Window.localStorage](https://developer.mozilla.org/en-US/docs/Web/API/Window/localStorage) properties (to be more precise, in supporting browsers the Window object implements the WindowLocalStorage and WindowSessionStorage objects, which the localStorage and sessionStorage properties hang off) — invoking one of these will create an instance of the [Storage](https://developer.mozilla.org/en-US/docs/Web/API/Storage) object, through which data items can be set, retrieved and removed. A different Storage object is used for the sessionStorage and localStorage for each origin — they function and are controlled separately.

Windows Session Storage –

The read-only **sessionStorage** property accesses a session [Storage](https://developer.mozilla.org/en-US/docs/Web/API/Storage) object for the current [origin](https://developer.mozilla.org/en-US/docs/Glossary/origin). sessionStorage is similar to [localStorage](https://developer.mozilla.org/en-US/docs/Web/API/Window/localStorage); the difference is that while data in localStorage doesn't expire, data in sessionStorage is cleared when the *page session* ends.

* Whenever a document is loaded in a particular tab in the browser, a unique page session gets created and assigned to that particular tab. That page session is valid only for that particular tab.
* A page session lasts as long as the tab or the browser is open, and survives over page reloads and restores.
* **Opening a page in a new tab or window creates a new session with the value of the top-level browsing context, which differs from how session cookies work.**
* Opening multiple tabs/windows with the same URL creates sessionStorage for each tab/window.
* Closing a tab/window ends the session and clears objects in sessionStorage.

Difference between COOKIES and SESSION:

**Cookie**

* Stores data that has to be sent back to the server with subsequent XHR requests. Its expiration varies based on the type and the expiration duration can be set from either server-side or client-side .
* Cookies are primarily for server-side reading (can also be read on client-side), localStorage and sessionStorage can only be read on client-side.
* Size must be less than 4KB.
* Cookies can be made secure by setting the httpOnly flag as true for that cookie. This prevents client-side access to that cookie.

## What is a service worker

A service worker is a script that your browser runs in the background, separate from a web page, opening the door to features that don't need a web page or user interaction. Today, they already include features like [push notifications](https://developers.google.com/web/updates/2015/03/push-notifications-on-the-open-web) and [background sync](https://developers.google.com/web/updates/2015/12/background-sync). In the future, service workers might support other things like periodic sync or geofencing. The core feature discussed in this tutorial is the ability to intercept and handle network requests, including programmatically managing a cache of responses.

Cookies –browser and server, local and session storage- browser only

Cookies- less data than local and session storage

Concurrency and Parallelism in Python Example 3: Distributing to Multiple Workers

While the threading and multiprocessing modules are great for scripts that are running on your personal computer, what should you do if you want the work to be done on a different machine, or you need to scale up to more than the CPU on one machine can handle? A great use case for this is long-running back-end tasks for web applications. If you have some long-running tasks, you don’t want to spin up a bunch of sub-processes or threads on the same machine that need to be running the rest of your application code. This will degrade the performance of your application for all of your users. What would be great is to be able to run these jobs on another machine, or many other machines.

A worker is something you give a task and continue in your process, while the worker (or multiple workers) process the task on a different thread. When they finish they let you know about it via a call back method. I.e. a special method provided on the initial call gets called.

**When Is Concurrency Useful?**

Concurrency can make a big difference for two types of problems. These are generally called CPU-bound and I/O-bound.

I/O-bound problems cause your program to slow down because it frequently must wait for input/output (I/O) from some external resource. They arise frequently when your program is working with things that are much slower than your CPU.