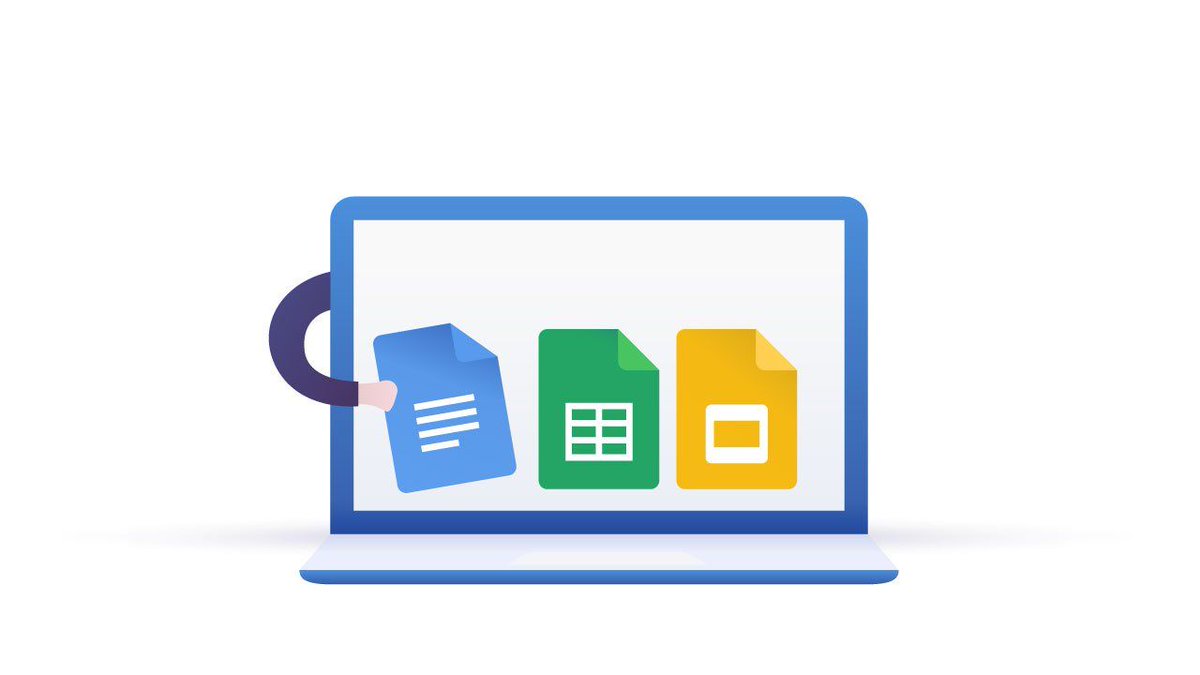
**Introduction**

Data is stored on disks under certain addresses. For a computer, such addresses are numbers. It uses them to find the corresponding information. However, humans are not that good at memorizing sets of numbers. It's much easier for us to name our data, for example, *family photo*, *November report*, and so on. Therefore, the concept of a file was invented: in a file, one can store some information under a user-friendly name. Let's take a closer look at what files are.

**What is a file**



A file is associated with a piece of data. However, there are some restrictions on naming a file. For example, filenames must contain only those characters that are supported by a specific file system. There are different types of contents of the files: text, photo, music, video, etc. The type of information stored in a file defines the **file format**. In order for the computer to be able to distinguish what format a certain file has, **file extensions**were invented.

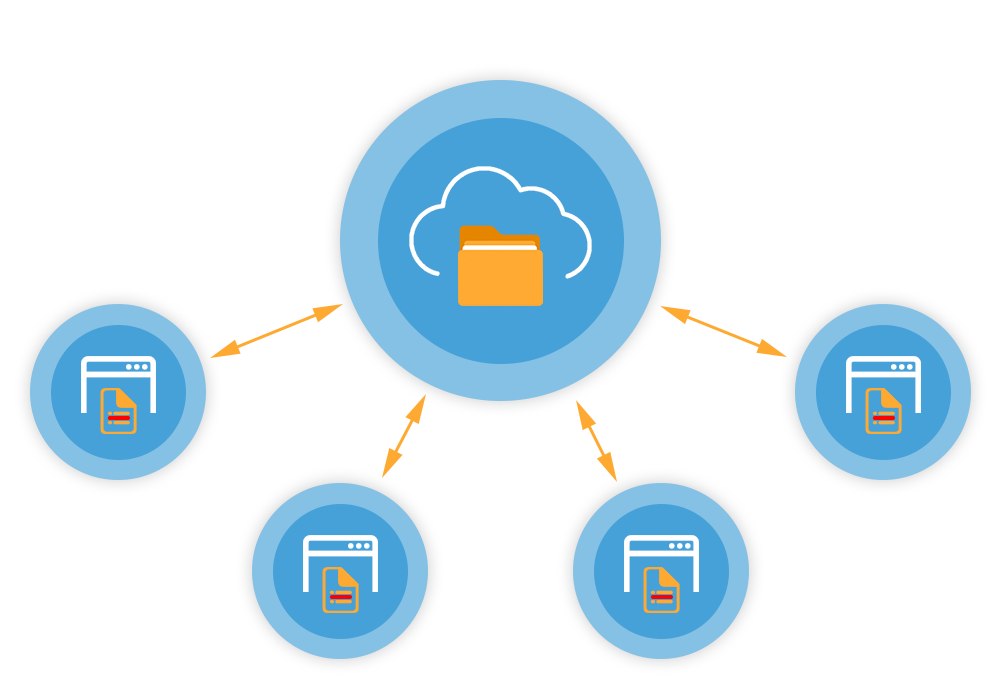
**File extensions**



Finding out the format of a file right away is pretty useful. One way to do it involves **filename extensions**. The end of a filename informs users or programs about the file format. The file format designation usually goes after a period, so you get a name ending with "*.<extension>*". As for more specific examples, here are some of the most common extensions: text files will have the *.txt* extension, for example, november\_report.txt; files with photos may have the *.jpg* extension, like my\_photo.jpg; for videos, the most usual is the *.mp4* extension and for music files, it is *.mp3*. Operating systems use filename extensions to remember which program to use to open files with a certain extension. Filename extensions aren’t strictly necessary, though: they just eliminate the need to guess the format of a file.

Now you know that a file extension is the service information used by a computer. Let's find out what other information a file has that helps computers understand what properties it has. This information is called**file metadata**.

**File metadata**



Metadata stands for "*data about data*"*.* One of the most common pieces of such data is the filename as we've discussed above*.* Other examples are the file size, creation time, the last access time, etc.

Also, metadata consists of **file attributes**. Each attribute has two possible states: **set** (toggled on) or **cleared** (toggled off). File attributes tell the file system or operating system if a file should get some special treatment. For example, if a file has a read-only attribute set, its contents can be read, but all attempts to modify them will be prevented by the file system until this attribute gets cleared. If a file has a **hidden** attribute set, it won’t show up in a graphical user interface unless the user explicitly tells the operating system to show all hidden files. Attributes can also be used to restrict file access to specific groups of users.

**Absolute and relative paths**

In order to find a file, we need to know the **path** to it. The path is a character set indicating the location of a file in the system. The file path can be seen in the file manager (Explorer in Windows). There are two types of paths: an **absolute** or a **relative** path.

A path to a file that starts with a root directory is called an **absolute path** and serves as the file’s unique identifier. If you try to create another file named "*my\_file*" in the same subdirectory, the file system won’t allow you to do that. If both files have the same identifier, how will the system tell them apart? Creating a file named "*my\_file"* in the root directory, however, would be okay: *"root\_directory/sub\_directory/my\_file*" and *"root\_directory/my\_file"* are different identifiers.

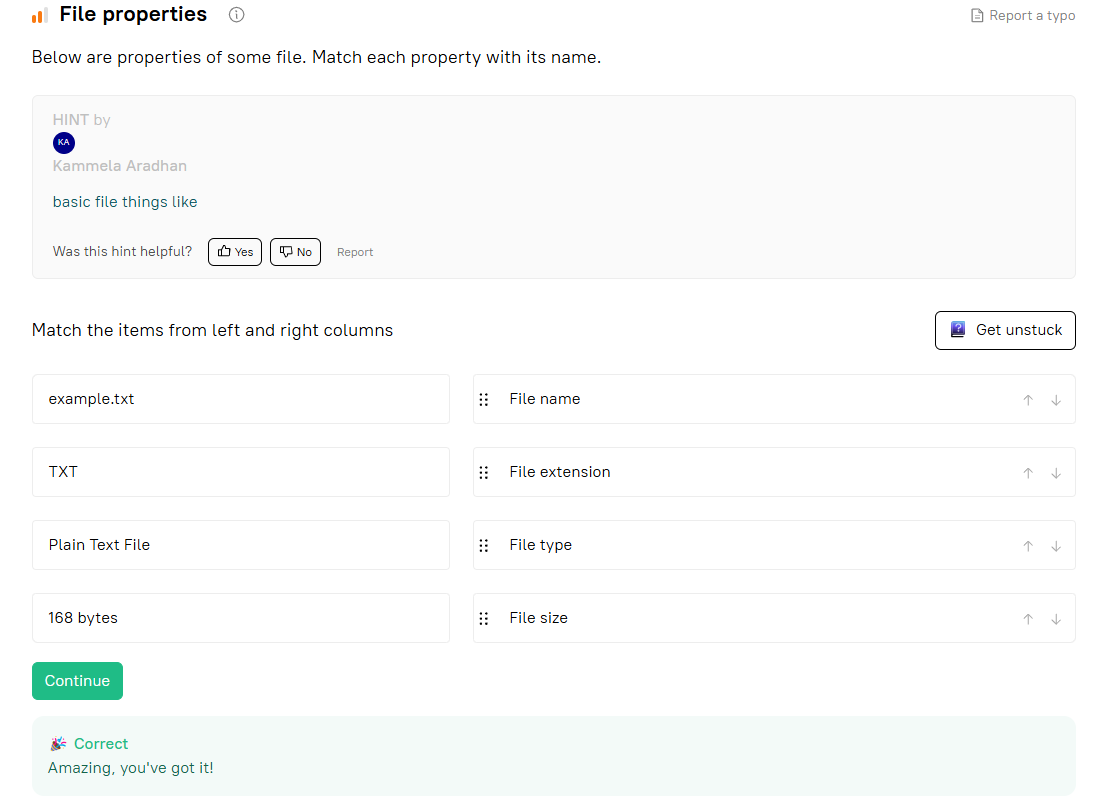
There is a catch with absolute paths though. When you write a program that will be installed on different computers, you know your own program’s directory, but you don’t know where other users of this program will install it. Your program’s directory can end up in any parent directory on a user’s computer, so you can't use an absolute path in your program to point to its directory.

This is where **relative paths** come into play. Each process that runs on a computer is associated with a **working directory** on this computer; it is tracked and managed by the operating system. This basically means that the operating systems of other users will be focused on your program’s directory when they run it. Your program can address the working directory by using a special character . instead of the directory’s actual name, so you can use a path like "*./my\_file*" without specifying the whole path from a root. You just let the user’s operating system figure it out! You can also use .. to address a parent directory of the working directory.

**Summary**

To sum up, in this topic you've learned that

* information on computers is stored in files
* the files are named in accordance with the system's instructions
* depending on the type of data in the file, the file will have a specific extension so that the computer can distinguish one format from another
* the detailed information about a file is called the file metadata
* to find the file you need to look at its path in the file explorer
* a path can be absolute or relative



How is a relative path different from an absolute one?

Select one option from the list

It does not start with the root directory.

What are file attributes for?

They tell the system what it can do with a file.

What are file attributes?

"File permissions" is just one part what "file attributes" define. There are other functionalities as well, where "file attributes" come into play. "File attributes" are responsible for how the system will behave.

✅ **Correct answer:**  
**Metadata that define file system behaviour**

### Explanation:

* **File attributes** are **metadata** that influence how the **file system handles** a file.
* This includes:
  + Read-only
  + Hidden
  + System file
  + Archive
  + And yes, also **permissions**, but not only that.

So the most complete and accurate option is:  
**Metadata that define file system behaviour.**

Why do we need a file extension?

distinguishing the file format

A file is a data container

**Correct answers:**

* **creation time**
* **file size**
* **last access time**
* **file attributes**

### ❌ Incorrect:

* **file content**  
  → This is the **actual data** in the file, **not metadata**.
* **general information about the OS**  
  → This is **not stored in file metadata**; it's system-level info, unrelated to individual files.