* SSH
* React
* Webpack + Parcel + Vite
* Node.js + Express
* Front End Performance Optimizations
* Back End Performance Optimization
* Progressive Web Apps
* Typescript
* Software Testing
* Server Side Rendering and Client Side Rendering
* Front End Security Optimizations
* Back End Security Optimizations
* Docker
* Redis
* Sessions + JWT
* Amazon Web Services ("AWS")
* Serverless Architecture
* Continuous Integration
* Continuous Delivery
* Continuous Deployment

# Learn Shells and Terminals

### How to open a terminal on Mac

Open the "Terminal" application. You can find it by searching for "Terminal" in Spotlight (the magnifying glass in the top-right corner of your screen).

## What's a CLI?

You'll often hear the terms "terminal", "shell", "command line", "CLI", and "command prompt" used interchangeably, but often they all refer to the same thing: a program that allows you to interact with your computer in a text-based way.

## What's a GUI?

If you don't have a technical background, you're probably used to interacting with your phone or computer using a [graphical user interface](https://en.wikipedia.org/wiki/Graphical_user_interface) (GUI). When you use a mouse to click on fancy icons, buttons, and menus, you're using a GUI.

To be fair, it's usually easier to teach someone how to use a GUI than a CLI. You can simply point to things and say "click here" or "drag this there". But GUIs do have some drawbacks:

* **They're weak.** You are given much more control over your computer through a CLI. With a GUI you're limited to the options that the developer of the GUI has given you.
* **They're slow.** Once you know the commands to type, it's much faster to type them than to click through endless menus with a mouse.
* **They're not as reproducible.** If you want to share a set of instructions, you can just copy and paste commands without worrying about screen sizes and user preferences.
* **They're not automatable.** It's easy to write code that manipulates text (as you've seen in Python), but it's much harder to write code that manipulates GUIs.
* **They're not as cool.** You will be invited to 90% fewer romantic outings if you are a GUI user.

## Assignment

Find and open a terminal on your computer (see below for instructions).

Once you've opened a terminal, type the following command and press enter:

echo "Hello world"

When you're done, answer the question.

### How to open a terminal on Mac

Open the "Terminal" application. You can find it by searching for "Terminal" in Spotlight (the magnifying glass in the top-right corner of your screen).

### How to open a terminal on Windows

Open the "Command Prompt" application. You can find it by searching for "Command Prompt" in the Start Menu.

### How to open a terminal on Linux

If you're on Linux, I refuse to believe that you don't already know how.

Ah, greetings fellow traveler in the land of programming! When you typed echo "Hello world" into the terminal and pressed 'enter', it was a command asking the terminal to display whatever follows the word echo. So, the terminal faithfully obeyed and printed the text "Hello world" onto its console for you to see.

This is why the correct choice is "The text 'Hello world' printed to the terminal's console." It's like asking the computer to speak, and it gladly echoes your words back to you! Isn't it just magical?

Search in video

the words terminal shell command line

CLI and command prompt are often just

thrown around halfhazard by programmers

and even though the precise definitions

vary slightly we're usually talking

about a program that lets you interact

with your machine through text if you're

new to the command line you're already

probably familiar with goys or graphical

user interfaces and a guy is just a

fancy visual interface that lets you get

stuff done on your computer think of a

Windows desktop for example it has menus

to sift through icons to click on and

even complex window Management Systems a

CLI or command line interface is a much

simpler alternative instead of using

complex visual buttons and menus you

interact solely through text once to run

a program don't click on an icon just

type the name of the program want to

view a file don't open Microsoft Word

just type a command now while goys do

have a lower barrier to entry that is

they're often self-explanatory to use

they're also pretty limied a command

line interfac is is more powerful

because you're not limited to the

buttons and widgets that are provided by

application developers instead you get

direct access to the lower levels of

your system as a programmer it's

non-negotiable you need to be

comfortable on a command line

## Learning goals

In previous Boot.dev courses you're likely used to running your code by clicking a "run" button. As a developer, you'll most likely run your code using a command line interface (CLI) instead. For example, in Python, you will probably run code on your own machine by typing this into a terminal:

python main.py

# Installing Windows Subsystem for Linux

You can skip this step If you're on Mac OS or Linux, or already have WSL 2 configured.

The built-in Windows command line (which we'll explain more later) is terrible. It's so bad that if you're on Windows we're just going to have you install WSL 2 ([Windows Subsystem for Linux](https://learn.microsoft.com/en-us/windows/wsl/about)). This will allow you to use your normal Windows desktop but also have a [Linux](https://www.linux.org/) command line, operating system, and filesystem for development.

WSL 2 was a huge step forward for developers on Windows, and in my opinion, is a much better experience than dual-booting. Don't worry, WSL is a Microsoft product that's built to work with Windows, it won't mess up your Windows install, and you'll still be able to run Steam.

Here are a couple of things to keep in mind about WSL 2:

* There are two versions of WSL. We will be using the latest: version 2.
* The WSL 2 filesystem is completely separate from your Windows file system. Use your normal Windows file system normally: for games, documents, windows apps, whatever. I recommend using the Linux filesystem for all your code.

## Install Process

### 1. Make sure you're on Windows 10 or 11

If you are on an older version, **please upgrade**. Your life as a developer will not be fun if you're on an older version of Windows.

### 2. Run Windows Update

Click on the start menu and begin typing "Windows update". You should see a program called "Windows Update Settings", open that. Click "check for updates", and if there are any updates, install them and restart your computer as instructed by the updater.

### 3. Open the Windows Command Prompt and install WSL

3a. Click on the start menu and begin typing "cmd.exe". You should see a program called "Command Prompt", right-click on it and select "Run as administrator".

3b. Type wsl --install in the prompt and press enter.

3c. Restart your computer

3d. Confirm wsl is installed by running wsl -l -v in Command Prompt.

### 4. Setting up the Ubuntu distro

[Ubuntu](https://ubuntu.com/about) is a very popular distribution (or distro) of Linux, and it was installed automatically alongside WSL.

4a. After restarting your computer in step 3, you should see an "Ubuntu" Window open automatically once you log back in. If you don't, search for the "Ubuntu" program in the start menu and launch it.

4b. The Ubuntu window will prompt you to enter a username and password ([which may not appear when typed](https://learn.microsoft.com/en-us/windows/wsl/setup/environment#set-up-your-linux-username-and-password)). Make sure you remember these! These are the credentials for your Linux user. Later when you need to enter your password for certain commands, you'll need to enter this password. Note: by default Ubuntu only accepts usernames in lowercase.

Once you're logged in, you should see a command prompt! Go ahead and type:

echo "Hello world"

You should see the text "Hello world" printed to the console. If you do, you're good to move on to the next step.

### Having trouble?

Here is the link to the [official installation guide](https://learn.microsoft.com/en-us/windows/wsl/install). Take a look and see if you missed any steps. Ask in the [Boot.dev Discord](https://boot.dev/community) if you're still having trouble.

#### Error: A required feature is not installed

Depending on the system you are working on you may run into an error telling you A required feature is not installed when attempting to start the virtual machine. To fix this you will need to enable Hyper-V (Virtualization). You can find this in your Windows features. You will need both Hyper-V Management Tools, and Hyper-V Platform enabled. You may also need to enable virtualization in your BIOS. If you are still having trouble, ask for help on the Boot.dev Discord.

I'm done with this step

**Struggling?** I, Boots the Master of Code and Casting, am happy to help! Ask me and I'll answer for a price. If you find my explanations suspect, ask a human for help in [the Discord](https://boot.dev/community)!

# What is a terminal?

As we talked about, the terms "shell", "CLI", and "terminal" are often used to refer to the same thing: the program that lets you issue text-based commands.

However, to get pedantic, the "terminal" is just one specific part of that program. Historically, the word "terminal" meant a physical device that you could type commands into, essentially a keyboard and a screen.



These days, when we say "terminal", we really mean "terminal emulator". A terminal emulator is a program that emulates a physical terminal. It's a program that lets you type commands into a window on your computer.

Which commands you're able to use isn't determined by the terminal emulator that you happen to be using. It's determined by the "shell", which we'll talk about later.

Your terminal emulator is just responsible for drawing text on the screen and processing your keystrokes.

Ah, greetings, young coder! In this question, we're delving into the role of a terminal emulator. Despite the whimsical choices involving baked goods, the correct answer is that a terminal emulator is indeed "a program that accepts text-based commands and can render text on the screen."

Why, you ask? Well, a terminal emulator is like a trusty scribe that translates your keystrokes into commands that the system can understand. It doesn't function as a standalone operating system or a set of commands; rather, it provides the interface through which commands are issued and responses are displayed. Keep this in mind as you wield your keyboard in the land of code!

# What is a shell?

So if your terminal is just a program that lets you issue text-based commands and renders the output of those commands...

...What is the program that runs those commands???

That's a shell.

Shells do a lot of things, but their main job is to interpret the commands you type and execute them.

## REPL

Shells are often referred to as "REPL"s. REPL stands for

* Read
* Eval (evaluate)
* Print
* Loop

This is a fancy way of saying that shells are programs that:

1. Read the commands you type
2. Evaluate those commands, usually by running other programs on your computer
3. Print the output of those commands
4. Give you a new prompt to type another command and repeat

## Assignment

In your own shell, run the following command:

expr 123456 + 7890

If you're on Windows CMD, use *set /a 123456+7890* instead.

Paste the output of that command into the text box to the right.

If you're confused about prompts vs outputs, here's a quick explanation.

wagslane@MacBook-Pro ~ % echo hello

hello

wagslane@MacBook-Pro ~ %

In the example above, I started with an empty prompt, and typed echo hello:

wagslane@MacBook-Pro ~ % echo hello

The shell then evaluated that command, and printed the output:

hello

Finally, it gave me a new prompt to type another command:

wagslane@MacBook-Pro ~ %

Profile

2



0

**9**

Ch 1: Terminals and Shells

6: Variables

sharpshooter0

milestone5

daily streak2

0

sharpshooter armor0

0

0

frozen flame0

# Variables

* If you're using Ubuntu on WSL, you're probably running a [Bash](https://en.wikipedia.org/wiki/Bash_(Unix_shell)) shell.
* If you're using macOS, you're probably running a [Zsh](https://en.wikipedia.org/wiki/Z_shell) shell.
* If you're running full Linux, I pray you already know what you're using.

The point is that you're probably using Bash or Zsh, and for this course, they're basically the same.

Both Bash and Zsh are shells, and they also happen to be powerful programming languages. They have variables, functions, loops, and more. That said, only [crazy people](https://bashsta.cc/) write large programs in shell languages... shells are optimized for running other programs and writing small scripts, not for writing programs.

## Create a variable

name="Lane"

## Use a variable

echo $name

# Lane

Unlike in Python where you can just use a variable's name, in your shell you need to prefix the variable name with a $ when you want to use it.

## Assignment

For the rest of this course, you'll be a forensic developer. You've been hired by "WorldBanc" to internally investigate a security breach. Like most mega-corps, they don't know what they're doing, but they're more than happy to overpay you to come in so that they can say a "third-party expert" has reviewed their systems.

Turns out, you're **not** an expert, but when you can learn on someone else's dime, why not jump at the opportunity? Right?...

...Anyhow, you'll be using your terminal and shell to traverse files, manipulate text, run programs, and find suspicious activity.

Paste the following into your shell:

bankname="WorldBanc"

founded="1969"

ceo="Jeff Gates"

Next, write and execute a bash command that will print out the following:

WorldBanc was founded in 1969 by Jeff Gates

Your command should dynamically use the variables from above.

Once you've made sure it's working properly, paste **the last command that generated the correct output** into the input field and submit it.

1. echo $bankname was founded in $founded by $ceo
2. echo "$bankname was founded in $founded by $ceo"
3. echo "${bankname} was founded in ${founded} by ${ceo}"
4. echo $bankname "was founded in" $founded "by" $ceo
5. echo $bankname 'was founded in' $founded 'by' $ceo
6. echo ${bankname} was founded in ${founded} by ${ceo}
7. echo ${bankname} "was founded in" ${founded} "by" ${ceo}
8. echo ${bankname} 'was founded in' ${founded} 'by' ${ceo}
9. echo "$bankname" was founded in "$founded" by "$ceo"

# History

When you're working in a REPL, it's really helpful to be able to see the commands you've typed in the past. That way you can easily re-run them, or copy and paste them into a script.

## Assignment

Let's take a look at your shell's history. In your shell, run the following command:

history

Of course, young learner! When you run the history command in a shell, it provides you with a list of the commands you've entered previously. This list is displayed with each command on a separate line, presented in the order from oldest to newest. This makes it easy to scroll back and see what was done in earlier sessions, just like flipping back through pages in a wizard's spell book. So, the correct answer is: "A list of commands, one per line, sorted oldest to newest." Happy coding adventures!

# Navigate History

## Arrows

You'll often want to re-run a command that you've run before. You could just type it out again, but assuming you don't have the [WPM](https://en.wikipedia.org/wiki/Words_per_minute) of ThePrimeagen, that's going to be a pain.

Instead, you can use the up and down arrows to cycle through your command history.

Focus your terminal window and use the "up" arrow key to start cycling through your command history. If you recently restarted your terminal type a few commands first, like:

echo hello

echo world

Once you've cycled back through your history with the "up" arrow, you can use the "down" arrow to cycle back to the most recent command.

## Clear

If your terminal is feeling cluttered with text, you can clear it with the clear command, or by pressing ctrl + l.

This won't delete your history, it will just clear the screen.

Ah, greetings young apprentice! This question is all about navigating your command history in the terminal. When you're looking to revisit commands you've used before, you press the "up" arrow to go back in your command history. To return to more recent commands, the "down" arrow is your ally. Hence, the correct answer is "up, down," just like exploring the shelves of a great library!

Ah, indeed! The clear command is like my magic spell for tidying up. When you type clear or press ctrl + l in the terminal, it cleans up your screen, making it nice and neat without actually altering your history.

So, the correct choice is "Clears the terminal's screen." It doesn’t open a new window or delete your history; it simply gives you a fresh view!

# Terminal Alternatives

So far you've been likely working in the default terminal that came with your operating system, and that's fine. However, there are a number of other options, I want to highlight a couple of them just in case you want to check them out (but you don't have to).

## Editor/IDE built-in terminals

Most text editors for developers have a built-in terminal.

VS Code is a popular text editor that also has a built-in terminal. I use VS Code for all of my development work, and I use the built-in terminal.

However, in this course, I do not recommend using VS Code. It's overkill for what we're doing.

## iTerm2

[iTerm2](https://iterm2.com/) is a popular terminal for Mac OS. It's a bit more powerful than the default terminal, and it has some nice features like split panes.

## Alacritty

[Alacritty](https://github.com/alacritty/alacritty) is a popular terminal for Linux with a focus on flexibility and extensive configuration. GPU-accelerated, it's great if you're doing a lot of heavy lifting in the terminal.

## Windows Terminal

[Windows Terminal](https://apps.microsoft.com/detail/9N0DX20HK701?rtc=1&hl=en-us&gl=US) is Microsoft's substitute for the Linux terminal. Use the "cmd.exe" program settings to change the default terminal. Be sure to start WSL whenever you open a new terminal window.

I'm done with this step

**Struggling?** I, Boots the Efficient Bubble Sorter, am happy to help! Ask me and I'll answer for a price. If you find my explanations suspect, ask a human for help in [the Discord](https://boot.dev/community)!



# What is a Filesystem?

All the data stored on your computer is organized into files and directories. Files and directories are organized into a tree-like structure called a filesystem.

* Directories (same as "folders" on Windows) are just containers that hold files and other directories.
* Files are just a dump of raw binary data: 1's and 0's. The bytes in a file can represent anything: text, images, videos, etc.

The filesystem tree starts with a single directory called the [root directory](https://en.wikipedia.org/wiki/Root_directory). The root directory contains files and directories, which can contain more files and directories, and so on.

When you open your terminal, your working directory (the one you're "in") is going to be... somewhere. Most commonly it is your ["home" directory](https://en.wikipedia.org/wiki/Home_directory).

## Assignment

You've just remotely logged onto a suspicious employee's machine at WorldBanc. First, you need to determine where on their filesystem you are...

Run the "print working directory" command to see the filepath of your current working directory:

pwd

Answer the question.

Ah, the mysteries of the filesystem unravel before us! When you use the pwd command, it reveals the *absolute* path to your current directory. On most Unix-like systems, absolute paths start from the root directory, represented by the / character.

So, the first character in the path is typically /, guiding you like the North Star from the root directory downward through the filesystem tree. Happy navigating!

# Filepaths

The output of your pwd command is a filepath. A filepath is a string that describes the location of a file or directory on your computer. Yours should look something like this:

/Users/wagslane

The text might be different, but the structure should be the same. Let's break it down:

* The first slash (/) represents the root directory. It's the tippy-top of the filesystem tree.
* The next part (Users) is the name of a directory inside the root directory.
* Finally, the last part (wagslane) is the name of a directory inside the Users directory.

So this path represents a directory 2 levels down from the root directory:

root

└── Users

└── wagslane

## Assignment

Time to start digging for evidence. Copy/paste the command below and run it to download the worldbanc directory from GitHub. It contains files and directories that you'll need throughout this course. If prompted for a password, use the password for your machine's user account, or the one you used when setting up Ubuntu in WSL.

curl -L https://github.com/bootdotdev/worldbanc/archive/refs/heads/main.zip -o worldbanc.zip

unzip worldbanc.zip

rm worldbanc.zip

mv worldbanc-main worldbanc

sudo chown -R $(whoami) worldbanc

sudo chmod -R 755 worldbanc

We'll cover what these commands do later.  
Normally, don't run commands if you don't know what they do. If you run into an issue, see the **troubleshooting tips** below.

**You should now have a worldbanc directory in your current working directory. When learning terminal commands in this course, it's possible to make a mistake and ruin your version of the worldbanc repo. If that happens, just come back to this lesson and download worldbanc again.**

Note: WSL users need make sure that the *worldbanc* directory is added to their Linux subsystem and not to the Windows filesystem

Run the "list" command to see the contents of your current working directory:

ls

You should see a worldbanc directory listed.

Run the "change directory" command to move into the worldbanc directory:

cd worldbanc

Finally, use the ls command again to see the contents of the worldbanc directory. Paste the console output into the input field and submit your answer.

## Troubleshooting

If you're having issues with the download due to [curl](https://curl.se/docs/manpage.html) or [unzip](https://www.linux.org/docs/man1/unzip.html) not being installed on Ubuntu/WSL, you can install them with the following commands:

sudo apt install unzip

sudo apt install curl

Certain Windows versions may not let you paste into the Command Line. See [how to enable ctrl+shift+v](https://superuser.com/questions/1410026/how-to-enable-ctrl-shift-v-in-windows-subsystem-for-linux-wsl-command-pr).

*The following checks run against your input:*

1. Input text must contain:
   1. private
   2. public

when you're working with a command line

file paths are the bread and butter of

navigation you're probably already used

to thinking of files and directories as

nested tree structures after all that is

what they are and that's how Windows

Explorer or finder on Mac display them a

file path is just a text-based

representation of one file or directory

in the hierarchy all absolute file paths

start at the root of the file system

which is just the very top directory in

the hierarchy from there everything is

nested inside a tree of directories with

each directory in the file path

separated by a slash now if the path

you're looking at points to a file then

the last section of the file path will

just be the name of the file but if

you're looking at a path to a directory

then the last section will be the name

of the directory to give an example in

this hierarchy the path to homework. txt

is SL documents SL school/ homework. txt

and the path to the work directory is

just SL documents slork

# Parent Directories

We talked about how you can "change directory" to move into a directory. But how do you move back out of a directory?

The answer is two dots: ..

.. is a special directory name that means "the parent directory". It's a shortcut that you can use to move up one level in the directory tree.

## Assignment

Navigate back out of the worldbanc directory so that you're in its parent directory. If you're in the top level of the worldbanc directory, just type:

cd ..

Then run the ls command again, but pass in the name of the worldbanc directory as an argument to list its contents from the parent directory:

ls worldbanc

Paste the output into the input box and submit your answer.

*The following checks run against your input:*

1. Input text must contain:
   1. README.md
   2. go.mod
   3. private
   4. public

# Absolute vs Relative Paths

We've mostly been dealing with [relative filepaths](https://www.redhat.com/sysadmin/linux-path-absolute-relative) which are just paths that take into account your current directory.

For example, let's say we have the following directory structure in our filesystem:

vehicles

├── cars

│ ├── fords

│ │ ├── mustang.txt

│ │ └── focus.txt

When inside the top-level vehicles directory, the relative path to the mustang.txt file is:

cars/fords/mustang.txt

However, when we're inside the cars directory, the relative path to the mustang.txt file is just:

fords/mustang.txt

Or when inside the fords directory, just:

mustang.txt

## Absolute Paths

An absolute path is a path that starts at the root of the filesystem. On [Unix-like systems](https://en.wikipedia.org/wiki/Unix-like) (Mac OS/Linux/WSL), the root is denoted by a forward slash /. So, if the vehicles directory is in the filesystem root, the absolute path to the mustang.txt file is

/vehicles/cars/fords/mustang.txt

So, when inside the fords directory, you can use either:

/vehicles/cars/fords/mustang.txt

or

mustang.txt

to refer to the same file.

## Which should I use?

It depends.

Relative paths are easier to read and write, and as long as you're in the correct directory (or the directory you expect), they're easier to reason about.

Absolute paths are more explicit. They're useful when you're not sure what directory you're currently in. For example, maybe you're giving someone instructions on how to find a file on their computer. You can't be sure what directory they'll be in when they start following your instructions, so you'll need to use an absolute path.

Ah, a fine question! When distinguishing between relative and absolute paths, the essence lies in where the path begins.

A relative path, like "home/ryan," is typically mentioned without a leading slash and is dependent on the current directory you're in. It's like giving directions starting from wherever you currently stand—a more localized approach, if you will.

On the other hand, absolute paths begin from the root of the filesystem. They start with a forward slash / and indicate a fixed journey from the beginning of the filesystem, like /home/ryan.

Thus, "home/ryan" is correct as a relative path since it lacks that initial slash and implies a path from some current directory. Keep exploring, trusty learner, and you'll master these paths in no time!

# Files

You're probably familiar with the concept of files from using a GUI like Windows Explorer or Finder.

At their core, files are just blobs of data. The raw bytes in a file can represent anything: text, images, videos, etc.

## The cat command

The cat command is used to view the contents of a file. It's short for "concatenate", which is a fancy way of saying "put things together". It can feel like a confusing name if you're using the cat command to view a single file, but it makes more sense when you're using it to view multiple files at once.

# Print the contents of a file to the terminal

cat file1.txt

# Concatenate the contents of multiple files and print them to the terminal

cat file1.txt file2.txt

## Assignment

Use the cd command to navigate into the worldbanc directory if you are not there already. Use ls to confirm that you're in the right place. You should see a public directory listed. cd into the public directory.

You should see a file called "pr\_ideas.txt". Use the cat command to view its contents.

Paste the contents of that file from your terminal into the input field and submit it.

*The following checks run against your input:*

1. Input text must exactly match:
   1. Money Trees for Everyone!: We are literally invested in the environment!
   2. WorldBanc will plant a tree for every new account opened.
   3. But here's the twist – each tree will be adorned with fake dollar bills featuring the CEO's face.
   4. Customers can come and 'harvest' the faux cash from their personal trees, exchanging it for real money at a hilariously low exchange rate.
   5. WorldBanc Cares: Recognizing the importance of personal touch in the digital age,
   6. we introduce a new campaign where the CEO personally writes an email to every customer Mr. Beast style.
   7. To top it off, each email signs off with a digitally autographed photo of the CEO giving a thumbs up.
   8. The WorldBanc Blimp Bonanza: The best way to stay 'above' the competition is quite literal – by investing in a fleet of blimps.
   9. These blimps, emblazoned with the WorldBanc logo, will hover over major cities, occasionally dropping WorldBanc-branded swag like pens, notepads, and stress balls.
   10. The pièce de résistance? Each blimp plays a jingle, the lyrics of which are
2. just the company's stock ticker symbol repeated to a catchy tune.

# Head and Tail

Sometimes you don't want to print everything in a file. Files can be really big after all.

## The head command

The [head](https://www.ibm.com/docs/en/aix/7.3?topic=h-head-command) command prints the first n lines of a file, where n is a number you specify.

head -n 10 file1.txt

If you don't specify a number, it will default to 10.

## The tail command

The [tail](https://www.ibm.com/docs/en/aix/7.3?topic=t-tail-command) command prints the last n lines of a file, where n is a number you specify.

tail -n 10 file1.txt

## Assignment

Time to start investigating the finances. Use the cd command to get into the worldbanc/private/transactions/ directory.

Run the cat command to view the contents of the 2023.csv file. You'll notice that it's a really long file. We don't want to print the whole thing.

1. Use the head command to print the first 6 lines of the 2023.csv file.

We are printing the first 6 lines because the first line in the file is the header, and we want to include that along with the first 5 transactions.

1. Use the tail command to print the last 5 lines of the 2023.csv file
2. Paste the first 6 lines, followed by the last 5 lines, into the input field and submit it.

## Tip

Remember, you can use .. as an alias for a parent directory. So if you're in worldbanc/private/transactions/ and you want to get to worldbanc, you can run cd .. twice:

cd ..

cd ..

Alternatively, you can just run:

cd ../..

once.

*The following checks run against your input:*

1. Input text must exactly match:
   1. amount,from\_user\_id,to\_user\_id,from\_name,to\_name,created\_at
   2. 80.16,18,14,Steve,Oscar,2023-07-05 08:43:55
   3. 962.70,5,10,Frank,Kyle,2023-01-07 18:39:14
   4. 407.51,4,4,Eva,Eva,2023-05-31 15:26:04
   5. 827.96,4,1,Eva,Bob,2023-06-03 16:53:02
   6. 472.91,5,13,Frank,Nora,2023-02-05 06:47:19
   7. 393.56,14,4,Oscar,Eva,2023-09-04 14:04:54
   8. 861.65,11,6,Lily,Grace,2023-08-10 01:28:46
   9. 210.75,5,5,Frank,Frank,2023-04-20 03:23:14
   10. 737.84,18,4,Steve,Eva,2023-11-18 16:21:57
   11. 683.91,10,8,Kyle,Ivan,2023-11-11 18:57:27

# More and Less

The [more](https://www.ibm.com/docs/en/aix/7.3?topic=m-more-command) and [less](https://man7.org/linux/man-pages/man1/less.1.html) commands let you view the contents of a file, one page (or line) at a time.

As the adage goes, less is more.

In the context of these commands, less is literally more. The less command does everything that the more command does but also has more features. As a general rule, you should use less instead of more.

You would only use more if you're on a system that doesn't have less installed.

## Assignment

You found nothing suspicious in the first and last transactions of 2023, but you're not done yet! It's time to dig through the middle of the file as well.

1. Run less and pass in the path to the 2023.csv file, located at the root in the worldbanc/private/transactions directory.

less 2023.csv

Notice that you're now in an interactive mode and you've lost your shell prompt! That's because less has taken over your terminal window.

1. Press "enter" a few times to scroll down a few lines, just to see how that works. Press "q" to exit the less program and return to your shell prompt.
2. Re-run the less command, but this time, pass in the -N flag to show line numbers:

less -N 2023.csv

You can use the spacebar to scroll down a page at a time, and you can go back up by pressing "b".

1. Find line 153. Copy and paste the contents of that line into the input field and submit it.

You can use "q" to exit less at any time.

*The following checks run against your input:*

1. Input text must exactly match one of:
   1. 876.65,12,2,Mia,Charlie,2023-07-30 03:44:00
   2. 153 876.65,12,2,Mia,Charlie,2023-07-30 03:44:00
   3. 245.61,8,3,Ivan,David,2023-06-11 11:11:56
   4. 153 245.61,8,3,Ivan,David,2023-06-11 11:11:56

# Touch

The [touch command](https://www.ibm.com/docs/en/aix/7.1?topic=t-touch-command) is designed to update the access and modification timestamps of a file. By default, if the specified file does not exist, touch will create an empty file with the given filename. Because of this, you’ll often see this command used to quickly create new files.

touch new\_file.txt

You can also create multiple files at once by listing them:

touch some\_file.txt some\_other\_file.txt

touch can be very handy when writing scripts because it ensures certain files exist without altering existing ones, creating new files only if necessary.

## Assignment

You discovered a discrepancy with the credit card files, worldbanc is supposed to be keeping credit history records but they aren't there! Change into the worldbanc/public/products/credit\_cards directory and create a new file named credithistory.txt so they can keep track of that information.

Use ls to verify that the file has been created successfully. Then paste the output of that command into the input field and submit it.

*The following checks run against your input:*

1. Input text must contain:
   1. jointrewardsplux.txt
   2. creditplus.txt
   3. freedomunited.txt
   4. economypoints.txt
   5. travelrewards.txt
   6. worldbanccard.txt
   7. credithistory.txt

# Directories

As we mentioned before, a directory is just a location in a filesystem that can contain files and other directories. On some systems, directories are called "folders", but it's the same thing.

## The mkdir command

The ["make directory" command](https://www.ibm.com/docs/en/aix/7.3?topic=m-mkdir-command) creates a new directory inside the current directory.

mkdir my\_directory

## Assignment

During your digging, you find that a file appears to be out of place. Make sure you're in the worldbanc/public/products/credit\_cards directory and run ls from there. You should see a file called "tbills.txt".

Treasury Bills (T-Bills) are a type of investment, not a credit card!

Go back to the products directory and create a new directory called "investments".

Once you've done that, from inside the products directory, run:

ls

Paste the output of that command into the input field and submit it.

*The following checks run against your input:*

1. Input text must contain:
   1. accounts
   2. cds
   3. credit\_cards
   4. loans
   5. investments

# Move

The [move command](https://www.ibm.com/docs/en/aix/7.3?topic=files-moving-renaming-mv-command) moves a file or directory from one location to another. You can use it to rename a file or to move it to a different directory altogether. Your working directory can't be the directory you're moving.

Renaming a file:

mv some\_file.txt some\_other\_name.txt

Moving a file from the current directory to another nested directory:

mv some\_file.txt some\_directory/some\_file.txt

If you don't want to rename the file and you're just moving it to a different directory, you can omit the filename:

mv some\_file.txt some\_directory/

## Assignment

Move the tbills.txt file, which can be found at worldbanc/public/products/credit\_cards/tbills.txt, into the new worldbanc/public/products/investments directory. Keep the filename the same.

Use ls to verify that the file has been moved correctly.

## Tips

* If you mess up the mv command, you'll need to figure out where you accidentally moved the file to, then move it from that location back to where it belongs.
* Both the target and destination have to be valid paths from the current working directory. Use pwd to see where you are and adjust the source and destination paths accordingly.



When using move, which is the correct syntax?

mv <source> <destination>

mv <destination> <source>

Ah, the magic of the mv command! When you're using it, the correct syntax is like giving directions where you say, "Take this item from this spot and place it over there."

That's why mv <source> <destination> is correct. The <source> is where the file currently resides, and the <destination> is where you're sending it. Much like moving a spellbook from one shelf to another!

The incorrect choice, mv <destination> <source>, would be like trying to take an empty spot and place something into it — it just doesn't make sense.

# Remove

The [remove command](https://www.ibm.com/docs/en/aix/7.3?topic=files-deleting-rm-command) deletes a file or empty directory:

rm some\_file.txt

You can optionally add a -r flag to tell the rm command to delete a directory and all of its contents recursively. "Recursively" is just a fancy way of saying "do it again on all of the subdirectories and their contents".

rm -r some\_directory

## Assignment

Change directories to the worldbanc/private directory using the cd command.

There's a big problem here! Use the cat command to view the contents of the worldbanc/private/passwords/passwords.txt file.

It looks like someone is storing passwords in plain text! Even on a private machine, this is a big security risk. Delete both the passwords directory and the passwords.txt file within it.

Once you've done that, list the contents of the "private" directory again to make sure that the file is gone. Paste the output of that command into the input field and submit it.

*The following checks run against your input:*

1. Input text must contain:
   1. bin
   2. customers
   3. cmd
   4. logs
   5. transactions
2. Input text must not contain:
   1. passwords

# Copy

The [copy command](https://www.ibm.com/docs/en/aix/7.3?topic=c-cp-command) does what you would (hopefully) expect: it copies a file from one location to another.

cp source\_file.txt destination/

You can also copy a directory and all of its contents recursively by adding the -R flag:

cp -R my\_dir new\_dir

## Assignment

Take a look inside the worldbanc/private/transactions/ directory. You should see a few files containing transactions from different years.

Next, look inside the backups directory inside of transactions. It looks like something is missing!

Copy the missing file from the transactions directory into the backups directory so there is a backup of all of the transactions.

Finally, ls the contents of the backups directory and paste the output into the input field and submit it.

*The following checks run against your input:*

1. Input text must contain:
   1. 2020.csv
   2. 2021.csv
   3. 2022.csv
   4. 2023.csv
2. Input text must not contain:
   1. backups

# Home

In a Unix-like operating system, a user's [home directory](https://en.wikipedia.org/wiki/Home_directory) is the directory where their personal files are stored. It is also the directory that a user starts in when logging into the system.

I recommend doing all of your development work in your home directory. For example, I like to create a workspace directory in my home directory, and all my projects live in subdirectories there.

## Danger

Your home directory is where you want to spend most of your time. Many of the other directories on your machine are critical to the operating system or other programs. Be careful when working in [other directories](https://www.ibm.com/docs/en/aix/7.3?topic=reference-directories) like /bin, /etc, /var, etc.

You can mess up your system if you're not careful.

## The ~ alias

My home directory (on Mac) is located at /Users/wagslane. The ~ character is an alias for your home directory. So when I want to go home, I don't have to type out cd /Users/wagslane, I can just type:

cd ~

Ah, my eager student, delve into the magic of file paths! In Unix-like systems, the tilde (~) is a shorthand for the home directory. So when we look at the path ~/workspace/names.csv, it's essentially pointing to a file named names.csv within a directory called workspace, all residing snugly within the home directory.

Hence, the correct answer indeed is: "A 'names.csv' file in a 'workspace' directory in the home directory." Just imagine it as a cozy nook for your files in the labyrinth of directories!



CSV Path

What does the filepath ~/workspace/names.csv refer to?

A 'workspace' file in a 'names.csv' directory in the home directory

A 'names.csv' file in a 'workspace' directory in the root directory

A 'names.csv' file in a 'workspace' directory in the home directory

A 'names.csv' file in a 'workspace' directory in the current directory

# Grep

You might be used to nice graphical interfaces that allow you to search for text in files, usually with ctrl+f or cmd+f. But what about when you're working on a terminal?

As it turns out, once you're used to it, searching for text in files on a CLI can be much faster than using a GUI.

## The grep command

The [grep command](https://www.digitalocean.com/community/tutorials/grep-command-in-linux-unix) allows you to search for text in files. It has a ton of capability, and we'll only be scratching the surface of its true power.

### Basic usage

The most basic usage of grep is to search for a string in a file. For example, if we wanted to search for the word "hello" in the file hello.txt, we could run:

grep "hello" hello.txt

This will print out every line in hello.txt that contains the word "hello". It's a case-sensitive search, so it will not match "Hello" or "HELLO".

## Assignment

Applications often write their logs to files on disk. These logs can contain useful information about what the application is doing, and can also be used to debug problems. As a security auditor, you need to dig through these logs to find any evidence of suspicious activity.

Use the grep command to find any lines with the text "CRITICAL" (all caps) in the worldbanc/private/logs/2024-01-10.log file.

Paste the output of your grep command into the input field and submit it.

## Tip

The tab key is your friend! If you start typing the name of a file or directory and then press tab, your shell will try to autocomplete the name for you. If there are multiple possible completions, it will show you a list of them. I rarely type out full file names, I type the first few characters and then press tab.

*The following checks run against your input:*

1. Input text must contain:
   1. CRITICAL: Boots is loose in the server room.
   2. CRITICAL: Server is on fire, please evacuate.
   3. CRITICAL: Someone rm -rf'd the root directory.
   4. CRITICAL: Database files are corrupt.
2. Input text must not contain:
   1. ALERT
   2. WARNING
   3. INFO
   4. DEBUG

*The following checks run against your input:*

1. Input text must contain:
   1. CRITICAL: Boots is loose in the server room.
   2. CRITICAL: Server is on fire, please evacuate.
   3. CRITICAL: Someone rm -rf'd the root directory.
   4. CRITICAL: Database files are corrupt.
2. Input text must not contain:
   1. ALERT
   2. WARNING
   3. INFO
   4. DEBUG

# Grep Multiple Files

You can also search multiple files at once. For example, if we wanted to search for the word "hello" in hello.txt and hello2.txt, we could run:

grep "hello" hello.txt hello2.txt

### Recursive search

You can also search an entire directory, including all subdirectories. For example, if we wanted to search for the word "hello" in the current directory and all subdirectories, we could run:

grep -r "hello" .

The *.* is a special alias for the current directory.

## Assignment

Use the grep command to find all the lines with the text "CRITICAL" (all caps) in the entire worldbanc/private/logs directory.

Paste the output of your grep command into the input field and submit it.

**Struggling?** I, Boots the Undercaffeinated and Overfed, am happy to help! Ask me and I'll answer for a price. If you find my explanations suspect, ask a human for help in [the Discord](https://boot.dev/community)!

*The following checks run against your input:*

1. Input text must contain:
   1. CRITICAL: Someone rm -rf'd the root directory.
   2. CRITICAL: Security scan is being maliciously manipulated.
   3. CRITICAL: Boots is loose in the server room.
   4. CRITICAL: Server is on fire, please evacuate.
   5. CRITICAL: Database files are corrupt.
   6. CRITICAL: Boots is out of Salmon again.
   7. CRITICAL: Power to the main datacenter has been lost. Attempting to reconnect.
   8. CRITICAL: Kernel panic - not syncing: VFS: Unable to mount root fs on unknown-block(0,0).
   9. CRITICAL: No one paid Dennis Nedry for his work. T-Rex released.
   10. CRITICAL: Worldbanc monitoring systems offline.
2. Input text must not contain:
   1. ALERT
   2. WARNING
   3. INFO
   4. DEBUG

# Find

The find command is a powerful tool for finding files and directories by name, not by their contents.

## Find a file by name

Let's say you're looking for a file named hello.txt somewhere in your home directory. You can use the find command to search for exactly that title:

find some\_directory -name hello.txt

## Pattern search

The find command can also search for files that match a pattern. For example, if you wanted to find all files that end in .txt, you could run:

find some\_directory -name "\*.txt"

The \* character is a wildcard that matches anything. If you're trying to find filenames that contain a specific word, you can use the \* character to match the rest of the filename:

# Find all filenames that contain the word "chad"

find some\_directory -name "\*chad\*"

## Assignment

You've been tipped off that fraudulent activity seems to happen often with "joint" accounts. "joint" means accounts that have more than one owner.

Use the find command to search the worldbanc/public/products directory for all files that contain the word "joint" in their name.

Paste the output of your find command into the input field and submit it.

*The following checks run against your input:*

1. Input text must contain:
   1. loans/jointloanshark.txt
   2. credit\_cards/jointrewardsplux.txt
   3. accounts/jointchecking.txt
   4. accounts/jointsavings.txt
2. Input text must not contain:
   1. businessplus.txt
   2. caymanislands.txt
   3. childrens.txt
   4. economypoints.txt

# Users

Unix-like systems (like the one you're using) support multiple users. Each user has their own home directory, their own files, and their own permissions.

If you're like most people these days, you're the only user on your machine. It used to be more common for multiple people to share a single computer, or for multiple people to do their work on the same computer over a network.

## Sudo

The sudo keyword lets you run a command as a "superuser". It's short for ["superuser do"](https://www.linux.com/training-tutorials/linux-101-introduction-sudo/). To use it, you'll need a password with superuser privileges, which you should already have if you're the only user of your machine.

Some people are pedantic and pronounce sudo as "sue-doo". Others are correct and pronounce it "sue-dough".

## Assignment

First, run the ["Who am I?" command](https://www.ibm.com/docs/en/zos/3.1.0?topic=wtsc-using-whoami-command) to see which user you're logged in as:

whoami

Next, run the same command as root:

sudo whoami

You'll be prompted for your password. Enter it. Copy and paste the output of the command into the input box and submit your answer.

Note: Using sudo grants unrestricted access and can risk system damage if you don't know what you're doing, apart from following this course.

*The following checks run against your input:*

1. Input text must contain:
   1. root

# Permissions

In a Unix-like operating system, permissions control who can do what to which files and directories. The permissions of an individual file or directory are visually represented as a 10-character string:

drwxrwxrwx

Let's break down each character. The first one just tells you whether you're looking at a file or a directory:

* -: Regular file (e.g. -rwxrwxrwx)
* d: Directory (e.g. drwxrwxrwx)

The next 9 characters are broken up into 3 sets of rwx and represent the permissions themselves for the "owner", "group", and "others", in order. Each group of 3 represents the permissions for reading, writing, and executing, in order. So, for example:

* rwx: All permissions
* rw-: Read and write, but not execute
* r-x: Read and execute, but not write
* The first 3 characters are "owner" permissions. The "owner" is usually just the user who created the file or directory, but it can be manually changed.
* The next 3 characters are "group" permissions. Unix-like systems support groups of users and a file or directory can be assigned to a single group. To be honest, unless you're a system administrator, you won't often worry about groups.
* The last 3 characters are "others" permissions. This is everyone else.

In my experience, when you're doing programming work on your own local machine, you mostly just care about the "owner" permissions because that's usually you. Here are some full examples:

* -rwxrwxrwx: A file where everyone can do everything
* -rwxr-xr-x: A file where everyone can read and execute, but only the owner can write
* drwxr-xr-x: A directory where everyone can read (ls the contents) and execute (cd into it), but only the owner can write (modify the contents)
* drwx------: A directory where only the owner can read, write and execute

Ah, a fine question indeed! Let's delve into the mysteries of Unix-like permissions. What we're seeking here is a set of permissions allowing everyone—the owner, the group, and others—to read.

Now let's examine the options:

* -rwxrwx---: Here, the owner and group can read, write, and execute, but others have no permissions.
* -rwxr-xr--: Here, the owner has all permissions, the group can read and execute, and others can read. This is the choice where everyone, including "others," has the read permission!
* -rwx--x--x: Here, the owner can read and write, but only the execute permission is granted to others, not read.
* -rwx------: Here, only the owner can read and write, with no permissions for group or others.

Thus, the answer -rwxr-xr-- provides reading permissions to the owner, group, and others, which is precisely what we're looking for. Keep unraveling the magic of permissions, young coder!

in a unix-based operating system like

Mac OS or Linux all files and

directories have a set of permissions

associated with them now there are two

ways that we break down those

permissions the first is who has the

permission the second is what permission

do they have now in regard to the what

any user accessing a file or a directory

may or may not have access to read it

write to it or execute it now all of

those permissions both The Who and the

what are represented by a character

string the first character is always

either a d or a dash if it's a d all

that means is this is a directory if

it's a dash it just means this is a file

the next three characters R WX represent

the three permissions that we talked

about read write and execute and The Who

that they apply to is the owner of the

file or directory the owner is usually

just the user that created it but it can

be changed manually now this rwx section

can have several different states each

permission can either be granted in

which case the letter is present or it

can be a dash in which case the

permission is not granted so for example

rwx means that the owner can read write

and execute while

r-x means that they can only read and

execute but not write and R Das Das

means that they can only read finally

this set of rwx permissions is just

repeated twice more the next set applies

to the group instead of the owner and

the last set applies to everyone else so

to break down a full character string

let's take the example of d rwx

r-x r d d the d means that this is a

directory the rwx means that the owner

can read write and execute the r-x means

the group can read and execute and the

r-- just means that everyone else can

only read this directory

# Permissions

In a Unix-like operating system, permissions control who can do what to which files and directories. The permissions of an individual file or directory are visually represented as a 10-character string:

drwxrwxrwx

Let's break down each character. The first one just tells you whether you're looking at a file or a directory:

* -: Regular file (e.g. -rwxrwxrwx)
* d: Directory (e.g. drwxrwxrwx)

The next 9 characters are broken up into 3 sets of rwx and represent the permissions themselves for the "owner", "group", and "others", in order. Each group of 3 represents the permissions for reading, writing, and executing, in order. So, for example:

* rwx: All permissions
* rw-: Read and write, but not execute
* r-x: Read and execute, but not write
* The first 3 characters are "owner" permissions. The "owner" is usually just the user who created the file or directory, but it can be manually changed.
* The next 3 characters are "group" permissions. Unix-like systems support groups of users and a file or directory can be assigned to a single group. To be honest, unless you're a system administrator, you won't often worry about groups.
* The last 3 characters are "others" permissions. This is everyone else.

In my experience, when you're doing programming work on your own local machine, you mostly just care about the "owner" permissions because that's usually you. Here are some full examples:

* -rwxrwxrwx: A file where everyone can do everything
* -rwxr-xr-x: A file where everyone can read and execute, but only the owner can write
* drwxr-xr-x: A directory where everyone can read (ls the contents) and execute (cd into it), but only the owner can write (modify the contents)
* drwx------: A directory where only the owner can read, write and execute

Certainly, dear apprentice!

In Unix-like systems, the permission string consists of segments that determine what actions can be performed on a file or directory by the owner, group, and others. The segments rwx stand for Read, Write, and Execute permissions respectively.

The question asks which set of permissions allows the owner (the first group of rwx) to execute a file.

Looking at the choices:

* -rw-rw-rw- – The owner can read and write, but not execute.
* -rw-rw-r-- – Again, the owner can read and write, but not execute.
* -r-xr-xr-- – Here, the owner has execute permission (x is present), along with read permission.
* -r--r--r-- – The owner can only read, with no execute or write permission.

The correct answer is -r-xr-xr-- because only here does the owner have execute permissions.

Remember, the presence of that x in the owner's part is what grants the magic of execution! Happy adventuring through the forest of knowledge!

# Changing Permissions

The [chmod command](https://www.ibm.com/docs/en/aix/7.3?topic=c-chmod-command) lets you change the permissions of a file or directory. It's short for "change mode" (I wish it was called "change permissions", but alas).

## Assignment

As part of your security audit, you need to know who has access to the files in the private directory. The ls command has a -l option (lowercase "L") that will print out the permissions of each file and directory in long format. Run this command inside the worldbanc/private directory:

ls -l

Change the permissions of the private directory and all of its contents so that:

* The owner can read, write, and execute
* The group can do nothing
* Others can do nothing

chmod -R u=rwx,g=,o= DIRECTORY

In the command above, u means "user" (aka "owner"), g means "group", and o means "others". The = means "set the permissions to the following", and the rwx means "read, write and execute". The g= and o= mean "set group and other permissions to nothing". The -R means "recursively", which means "do this to all of the contents of the directory as well".

Be sure to replace DIRECTORY with the path to the private directory.

Remember, *.* is a special alias for the current directory.

Once you've changed the permissions, run the ls -l command again to make sure they've been updated.

Paste the **10-character** permission string of the updated private directory into the input field and submit your answer.

## Troubleshooting

If you're using WSL and chmod is not updating the permissions: your worldbanc directory may not be within your Linux subsystem. You can either move the directory or [adjust your wsl.conf file to allow for editing permissions](https://stackoverflow.com/questions/46610256/chmod-wsl-bash-doesnt-work).

*The following checks run against your input:*

1. Input text must exactly match one of:
   1. drwx------@
   2. drwx------

# Executables

You're probably familiar with the idea of reading and writing files. But what about executing them? Executable files are just programs that you can run on your computer.

Files with a .sh extension are [shell scripts](https://en.wikipedia.org/wiki/Shell_script). They're just text files that contain shell commands. You can run a file in your shell by just typing its filepath:

mydir/program.sh

Interestingly, if the program is in the current directory (in this example, the mydir directory), you need to prefix it with ./:

./program.sh

As far as file paths go, ./program.sh and program.sh are the same. The dot (.) is an alias for the current directory. We need the prefix when running executables so that the shell knows we're trying to run a file from a file path, not an installed command like ls, mkdir, chmod, etc.

## Assignment

There seems to be a suspicious script in the worldbanc/private/bin directory called genids.sh. It looks like it generates some type of identifier... First, let's **remove our ability to run it**, just to see what happens. The chmod command has a convenient -x flag that will simply remove the executable permission from the file.

Run the following commands in worldbanc/private/bin:

chmod -x genids.sh

Let's test those permission changes. Try to run the script:

./genids.sh

You should get an error like this:

permission denied

That error occurs because the executable permission is not set for your user (the owner). Let's re-add the executable permission for the owner:

chmod +x FILENAME

Once you've done that, try running the script again. Paste the output of the script into the input field and submit your answer.

## Tips

You'll frequently download scripts from the internet to run on your machine, and often you'll need to make them executable before you can run them. I use chmod +x quite often as a developer.

The internet is a shady place. Only run verified scripts from trusted publishers and authors.

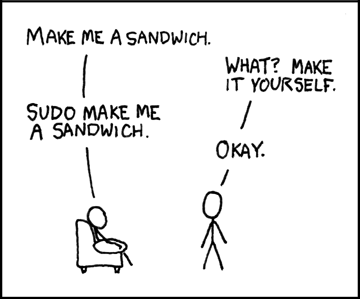
*The following checks run against your input:*

1. Input text must exactly match one of:
   1. ID-3478058268
   2. ID-3441885333
   3. ID-3439307794
   4. ID-3388271495
   5. ID-3358344960
   6. ID-3393565321
   7. ID-3416071694
   8. ID-3226680739
   9. ID-3247149348
   10. ID-3127757497
   11. ID-3149227582
   12. ID-3118240695
   13. -n ID-
   14. 3478058268
   15. -n ID-
   16. 3441885333
   17. -n ID-
   18. 3439307794
   19. -n ID-
   20. 3388271495
   21. -n ID-
   22. 3358344960
   23. -n ID-
   24. 3393565321
   25. -n ID-
   26. 3416071694
   27. -n ID-
   28. 3226680739
   29. -n ID-
   30. 3247149348
   31. -n ID-
   32. 3127757497
   33. -n ID-
   34. 3149227582
   35. -n ID-
   36. 3118240695

# Root User

The "root" user is a superuser. It has access to everything on the system and can do anything. When you use the sudo command, you're running as the root user (as long as your system hasn't been configured differently).

The sudo keyword is convenient because it quickly gives you elevated permissions to run a single command.



-- [xkcd](https://xkcd.com/149/)

However, it can also be dangerous because it gives you access to everything. If you run a command with sudo that you don't understand, you could do serious damage to your system.

For example, rm with the r and f flags run on the root directory (/), will **delete all the files on your system**. Don't do that. The r flag is for "recursive" (delete everything inside) and the f flag is for "force". Most systems will prevent you from doing this, but if you run it with sudo, you've just turned your computer into a very expensive paperweight.

Some modern systems will actually prevent you from deleting everything by default as a safeguard unless you use *--no-preserve-root*, but it's still a very bad idea.

## Should I use sudo?

Sure, as long as you understand what the command you're running does. Just be careful.

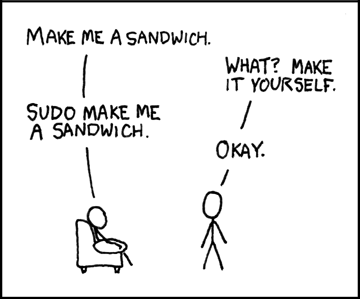
Ah, an expedition into the world of users! The "root" user is indeed the superuser of a system. This special user has the power to access all parts and perform any action on the system. It's like having the magic staff of ultimate authority!

While options like "administrator" and "admin" do suggest roles with significant control, the "root" user specifically is the highest level with unrestricted access. Just be careful with such power, as it can both create and destroy with a few swift commands!

# Root User

The "root" user is a superuser. It has access to everything on the system and can do anything. When you use the sudo command, you're running as the root user (as long as your system hasn't been configured differently).

The sudo keyword is convenient because it quickly gives you elevated permissions to run a single command.



-- [xkcd](https://xkcd.com/149/)

However, it can also be dangerous because it gives you access to everything. If you run a command with sudo that you don't understand, you could do serious damage to your system.

For example, rm with the r and f flags run on the root directory (/), will **delete all the files on your system**. Don't do that. The r flag is for "recursive" (delete everything inside) and the f flag is for "force". Most systems will prevent you from doing this, but if you run it with sudo, you've just turned your computer into a very expensive paperweight.

Some modern systems will actually prevent you from deleting everything by default as a safeguard unless you use *--no-preserve-root*, but it's still a very bad idea.

## Should I use sudo?

Sure, as long as you understand what the command you're running does. Just be careful.

**Struggling?** I, Boots the Coffee Connoisseur, am happy to help! Ask me anything about the lesson. If you find my explanations suspect, ask a human for help in [the Discord](https://boot.dev/community)!



What happens if you run 'sudo rm -rf /'?

Your computer will be wiped clean of all files, including files needed to function

Your computer will be wiped clean of all files, but will still function normally

You will lose your 'collection' of dank memes, everything else will be fine

Somewhere in the world Linus Torvalds chuckles

Explain 🪄

Ah, adventurous learner! This question plays with the awesome, yet dangerous power of the sudo command combined with rm -rf /.

When you run this command as the root user, you instruct your system to delete everything, including files essential for the operating system to function. It’s like asking a dragon to torch your castle—bold, but not without consequence!

Therefore, the correct choice is: "Your computer will be wiped clean of all files, including files needed to function." So, tread carefully when using sudo, as it grants immense power, but with great power comes great responsibility!

# Chown

So when do you need to use sudo? chmod allows you to change the permissions of any file or directory that you own. But what if you don't own the file or directory? That's where sudo is required. Let's change ownership of a directory to see how that works.

The chown command, which stands for "change owner", allows you to change the owner of a file or directory, and it requires root privileges.

## Assignment

Navigate into the worldbanc/private directory. Take a look at the contents:

ls

You should see a directory called contacts. Take a look at its contents.

You should see a file called emergency.txt. Take a look at its contents with cat.

Hmmm, seems like it has some personal contact info for employees. We don't know whose this is, but we're pretty sure it should have its access strictly limited. Let's change the owner of the entire contacts directory to root:

sudo chown -R root contacts

Let's break down this command:

* sudo - Run as the root user
* chown - Command to change the owner
* -R - "Recursive", meaning also apply the changes to everything inside the directory
* root - The name of the new owner
* contacts - The directory to change the owner of

Next, from the worldbanc/private directory, run:

ls -l

Paste only the line from the output that represents the contacts directory into the input field and submit your answer.

*The following checks run against your input:*

1. Input text must contain:
   1. contacts
   2. root
2. Input text must not contain:
   1. bin/
   2. cmd/
   3. customers/
   4. logs/
   5. transactions/

# Using sudo

Before you start this assignment, make sure the permissions on the worldbanc/private/contacts directory are drwx------, that the owner is root, and that you're not signed in as root. If you've been following along, all of those should be true. You can check with the ls -l command.

## Assignment

You told senior staff about the contacts directory and its contents, and they've decided to delete it entirely. Go ahead and try to delete it with the rm command:

rm -r contacts

The contacts directory should fail to delete! You do not have permission to delete it. It's owned by the root user, and you're not root.

Try again as root by using sudo.

Once you have confirmed that you were able to delete it, run the ls command from inside the private directory. Paste the output into the input field and submit your answer.

*The following checks run against your input:*

1. Input text must contain:
   1. bin
   2. cmd
   3. customers
   4. logs
   5. transactions
2. Input text must not contain:
   1. contacts

a program is just a bunch of

instructions written in a way that a

computer can carry out and in a nutshell

there are two kinds of programs compiled

programs and interpreted programs for

our purposes the real difference between

the two is just that compiled programs

do not rely on any other programs to be

able to run they're usually written in a

language like C rust or go but are then

compiled down to Binary the raw language

of your computer's Hardware interpreted

programs on the other hand do require

other software in order to be executed

python Ruby JavaScript or shell scripts

rely on an interpreter to read the code

as they run when you run a program on

your machine that has no other

dependencies it can simply be ran by

typing the name of the program into your

shell however when you run an

interpreted program you need to tell

your machine which interpreter it should

use for example to run a compiled

program called text editor you might

just type do/ text editor to run an

interpreted Python program program you

might have to type python text editor.

Pui and that's because the first one is

a compiled binary it's just ones and

zeros that your machine knows how to

execute in the second example text

editor. py is just a file of python

source code so it needs access to the

python interpreter

# Compiled vs Interpreted

A program is just a set of instructions that a computer can execute. We talked about executables in the last lesson, an executable is just a file that contains a program. The words "program" and "executable" are often used interchangeably. Broadly speaking, there are two types of programs:

* Compiled programs
* Interpreted programs

## Compiled programs

A compiled program is a program that has been converted from human-readable source code into machine code (binary). [Machine code](https://en.wikipedia.org/wiki/Machine_code) is a set of instructions that a computer can execute directly: your computer's CPU is hardware that's been designed to execute machine code.

Programming languages like Go, C, and Rust are compiled languages that produce compiled programs.

## Interpreted programs

An interpreted program is a program that is executed by another program. The program that executes the interpreted program is called an [interpreter](https://en.wikipedia.org/wiki/Interpreter_%28computing%29). The interpreter reads the source code of the interpreted program and executes it.

Programming languages like Python, Ruby, and JavaScript, are interpreted languages. Your computer needs to have an interpreter installed to run programs written in those languages.

Another example is the .sh shell script files we talked about. Those are interpreted by the shell program.

## Assignment

In your shell, run the following command:

which sh

The which command tells you the location of an installed command line program. In this case, we're asking for the location of the sh (shell) program.

Mine is located at /bin/sh. Next, take a look at the contents of that file:

cat /bin/sh

Keep in mind, your sh program is a compiled executable, probably written in C. That's why when you try to view its contents with cat, you see... what you see.

A file with a .sh extension on the other hand is a shell script. It's a text file that contains commands that will be interpreted and run by the sh program. They are both executable programs, but only one can be run without the help of another program.

**Struggling?** I, Boots the Sleepy Spellcaster, am happy to help! Ask me and I'll answer for a price. If you find my explanations suspect, ask a human for help in [the Discord](https://boot.dev/community)!



What is the output of 'cat'ing the sh executable?

Raw binary encoded as text in hexadecimal

Raw binary encoded as text in 1's and 0's

Nothing

A bunch of gibberish (raw bytes attempting to be interpreted as text)

Become a member for quiz access

# Shebang

As we talked about before, you can run any executable file by typing its file path into your shell. For example:

bin/genids.sh

That works out-of-the-box for files that are compiled executables. But what about scripts that need to be interpreted by another program? The computer needs to be told what program to use to interpret the file.

A ["shebang"](https://en.wikipedia.org/wiki/Shebang_(Unix)) is a special line at the top of a script that tells your shell which program to use to execute the file.

The format of a shebang is:

#! interpreter [optional-arg]

For example, if your script is a Python script and you want to use Python 3, your shebang might look like this:

#!/usr/bin/python3

This tells the system to use Python 3 located at /usr/bin/python3 to run the script.

## Assignment

Use the cat or less command to view the contents of the private/bin/genids.sh file. Paste the shebang into the input field and submit your answer.

# Bourne Shell

For your main shell REPL, as we talked about before:

* If you're using Ubuntu on WSL, you're probably running a [Bash](https://en.wikipedia.org/wiki/Bash_(Unix_shell)) shell.
* If you're using macOS, you're probably running a [Zsh](https://en.wikipedia.org/wiki/Z_shell) shell.
* If you're running a raw Linux installation, I pray you already know what you're using.

To get hand-wavy about it, I want to explain the difference between the 3 shells you're likely to encounter:

* sh - The Bourne shell. This is the original Unix shell and is [POSIX-compliant](https://en.wikipedia.org/wiki/POSIX). It's very basic and doesn't have many quality-of-life features.
* bash - The Bourne Again shell. This is the most popular shell on Linux. It builds on sh, but also has a lot of extra features.
* zsh - The Z shell. This is the most popular shell on macOS. Like bash, it does what sh can do, but also has a lot of extra features.

Both zsh and bash are "sh-compatible" shells, meaning they can run .sh scripts, but they also have extra features that generally make them more pleasant to use. For your purposes, the differences between zsh and bash are not super significant. Everything we do in this course will work in both shells.

# Shell Configuration

Bash and Zsh both have [configuration files](https://en.wikipedia.org/wiki/Unix_shell#Configuration_files) that run automatically each time you start a new shell session. These files are used to set up your shell environment. They can be used to set up aliases, functions, and environment variables.

These files are located in your home directory (~) and are hidden by default. The ls command has a -a flag that will show hidden files:

ls -a ~

* If you're using Bash, .bashrc is probably the file you want to edit.
* If you're using Zsh, .zshrc is probably the file you want to edit or create if it doesn't yet exist.

## Assignment

As part of your audit, you're trying to figure out what shell commands are run automatically when a user logs in on the machine. You need to confirm which shell configuration file is being used.

Edit the file you believe to be your shell configuration file. You can use the nano command to edit files in your shell:

nano ~/.bashrc

Add the following line to the bottom:

echo "Hello from the shell!"

You can use Ctrl+O to save the file (confirm any prompts with "enter"), and then Ctrl+X to exit the editor. (There should be a list of commands at the bottom of the screen.)

Finally, close your terminal and open a new one. If you see the message "Hello from the shell!", you've found the right file!

Feel free to remove the line you added to the file so that you don't see that message every time you open a new shell.

**Struggling?** I, Boots the Bear with a Back-End, am happy to help! Ask me and I'll answer for a price. If you find my explanations suspect, ask a human for help in [the Discord](https://boot.dev/community)!



Why might you restart your shell after changing a shell configuration file?

Because the config file runs when you type 'uwu senpai pweez'

Because the config file runs automatically when you start a new shell session

Because the config file runs automatically when you save a shell configuration file

Because the config file runs automatically when you close a shell

Become a member for quiz access

# Environment Variables

We talked about how you can create and use local variables in your shell:

name="Lane"

echo $name

# Lane

There is another type of variable called [environment variables](https://en.wikipedia.org/wiki/Environment_variable). They are available to all programs that you run in your shell.

You can view all of the environment variables that are currently set in your shell with the command env.

If you want to set a variable in your shell, you can use the export command:

export NAME="Lane"

You can then use the variable in your shell, just as before:

echo $NAME

# Lane

The interesting part is that programs and scripts you run in your shell can also use that variable:

For example, we can create a file called introduce.sh with the following contents:

#!/bin/sh

echo "Hi I'm $NAME"

Then we run it:

chmod +x ./introduce.sh

./introduce.sh

# Hi I'm Lane

## Assignment

Take a look at the contents of the warn.sh script in the worldbanc/private/bin directory. It looks like it's supposed to print nicely formatted warning messages with worldbanc branding.

export the required environment variables:

* WARN\_MESSAGE: "The auditor is here"
* WARN\_FROM\_NAME: "Your worst nightmare"

Run the script and paste the output into the input field and submit your answer.

## Tip

You can also temporarily set a variable for a single command, instead of exporting it (exporting means the variable will persist until you close the shell).

For example:

WARN\_MESSAGE="this works too" ./warn.sh

# PATH

This is one of the most important lessons in this entire course! Listen up.

There are environment variables that are sort of "built-in" to your shell. By "built-in" I just mean that different programs and parts of your system know about them and use them. The PATH variable is one of those.

## Why do we care about the PATH?

If it weren't for the PATH, you'd have to remember the filesystem path of every executable you wanted to run. Instead of just running ls, you'd have to run /bin/ls (or whatever the location of the ls executable is on your system). That's not very convenient.

The PATH variable is a list of directories that your shell will look into when you try to run a command. If you type ls, your shell will look in each directory listed in your PATH variable for an executable called ls. If it finds one, it will just run it. If it doesn't, it will give you an error like: "command not found".

## What's in the PATH variable?

Take a look at your current PATH variable:

echo $PATH

You should see a giant list of directories separated by colons (:). Each of those directories is a place where your shell will look for executables. For example, with a PATH like this:

/usr/local/bin:/usr/bin:/bin:/usr/sbin:/sbin

Your shell will look for executables in the following directories:

* /usr/local/bin
* /usr/bin
* /bin
* /usr/sbin
* /sbin

## Assignment

As something of a security engineer yourself, you want to temporarily disable the PATH variable so that you can only run executables by using their full path. You know, just so you don't accidentally run something you don't mean to.

Reset your PATH variable to an empty string:

export PATH=""

This will only affect your current shell session. If you open a new shell, it will have the default *PATH* variable again.

When you're done, try running some simple commands like ls, pwd, echo, etc – some will no longer work, while others that do not rely on PATH will. Then answer the question.

**Struggling?** I, Boots the Efficient Bubble Sorter, am happy to help! Ask me and I'll answer for a price. If you find my explanations suspect, ask a human for help in [the Discord](https://boot.dev/community)!



After deleting your PATH, what happens when you try to run 'ls'?

The command isn't found

It lists the contents of the root directory

It lists the contents of the current directory

It gives a security warning

Become a member for quiz access

# Change your PATH

First, restart your shell session to reset your PATH variable to its default. You can do that by closing your terminal window and opening a new one.

A common problem you'll run into in the future is that you install a new program on your machine, but when you try to run it from your terminal, you get an error like:

$ my-new-program

-bash: my-new-program: command not found

Nine times out of ten, this is because the program is installed in a location that's not in your PATH variable. Oftentimes when you install a program using the CLI, it will print a message during the installation process that tells you where the command was installed. **Don't let your eyes glaze over when your terminal prints important messages!** Sometimes you just gotta [rtfm](https://www.dictionary.com/browse/rtfm).

## Assignment

The worldbanc directory that you downloaded has some executables that would be useful to have in your PATH so that you can run them from anywhere. Add the worldbanc/private/bin directory to your PATH variable. You'll need to add the absolute path, not the relative path. You can get the absolute path by running pwd in the worldbanc/private/bin directory, or by using the realpath command.

To add a directory to your PATH without overwriting all of the existing directories, use the export command and reference the existing PATH variable:

export PATH="$PATH:/path/to/new"

The $PATH part is a reference to the existing PATH variable. The : separates the existing directories from the new directory (/path/to/new) that you're adding.

Once you've done that, cd outside of the worldbanc directory entirely, maybe to your home directory:

cd ~

Then try running the worldbanc.sh CLI Tool. Follow the prompts, then copy/paste the entire interaction with the tool into the input field and submit your answer. (Starting with "Welcome" and ending with "Goodbye!")

# PATH Config

In the last lesson, you changed your PATH variable for your current shell session. Trouble is, the next time you restart your shell, it will be reset to its default value. You won't be able to use the worldbanc CLI Tool from anywhere unless you change your PATH variable permanently.

The most common way to do this is to add the same export command that you used in the last lesson to your shell's configuration file.

## Assignment

Edit your .zshrc / .bashrc file (whatever your shell config file is) and add an export command to the end of the file that adds the worldbanc/private/bin directory to your PATH variable.

Once you've done that, restart your shell session.

Then run the worldbanc.sh CLI tool again from your home directory and make sure it works.

What changed when we added the PATH change to the config file?

We need to use the full path to the 'worldbanc.sh' script to run it

The 'worldbanc.sh' script is available from anywhere, even after a session restart

The 'worldbanc.sh' script is available from anywhere, but only in the current session

Become a member for quiz access

# Man

The [man](https://www.ibm.com/docs/en/aix/7.3?topic=m-man-command) command is short for "manual". It's a program that displays the manual for other programs.

I know that manuals and documentation can feel intimidating, heck, that's why there are courses like this one to give you a gentler introduction. That said, manuals and documentation should become more useful to you as you become a more experienced developer. They're not as scary as they seem when you actually take the time to read them.

## Using man

The man command will only work for programs that it has a manual for, but most built-in commands and Unix programs are supported. You just pass the name of the command you want to read the manual of as an argument. The most intuitive place to start, of course, is reading the manual-manual:

# open the man pages for the 'man' command

man man

## Searching

Most people don't just [read man pages for fun](https://www.youtube.com/watch?v=rT-fbLFOCy0). More often, the manual is used as a reference to quickly look up usage or special flags.

You can search for text in the manual by pressing the / key and typing your search, then pressing enter. Let's try to look up what the -r flag does for the ls command:

man ls

# type '/-r' to start searching

# press 'n' to jump to the next result

# press 'N' to go back if you went too far

## Assignment

As a systems person, the grep command is going to be your best friend. It has a lot of power, but it can be a little confusing to use. Open its manual.

You'll notice that the manual is an interactive session. You can page through the manual with the spacebar, and quit with q.

Read the first couple paragraphs of the grep manual and answer the question(s).

What does the '-v' flag do?

selects non-matching lines

prints the version

matches lines that start with 'v'

Become a member for quiz access

# Flags

As you've already seen in this course, some commands can take [flags](https://www.ibm.com/docs/en/aix/7.3?topic=names-command-flags). Flags are options that you can pass to a command to change its behavior.

For example, the ls command can take a -l flag to show a "long" listing of files:

ls -l

The ls command can also take a -a flag to show "all" files, including hidden files:

ls -a

You can combine flags:

ls -al

## Conventions

Whether or not a command takes flags, and what those flags are, is up to the developer of the command. That said there are some common conventions:

* Single-character flags are prefixed with a single dash (.e.g -a)
* Multi-character flags are prefixed with two dashes (e.g. --help)
* Sometimes the same flag can be used with a single dash or two dashes (e.g. -h or --help)

## Assignment

You've found that some files have been tampered with! To figure out which one it is, you need to know the number of bytes contained in each file.

Use the ["word count" command](https://www.ibm.com/docs/en/aix/7.3?topic=af-counting-words-lines-bytes-in-files-wc-command), wc, to count the number of bytes in the worldbanc/public/pr\_ideas.txt file. Use the man command to figure out which flag you need to use.

Paste the number of bytes (just the number!) into the input field and submit your answe

# Positional Arguments

Programming languages have functions, and functions take arguments. For example, this Python function takes two arguments: xp and level:

def print\_player(xp, level):

print("Player has", xp, "xp and is level", level)

It can then be called with two arguments:

print\_player(100, 2)

# Player has 100 xp and is level 2

In a shell, commands (programs) can also take arguments. For example, the cd command takes a single argument (the directory to change to):

cd /home/wagslane

Other commands might take multiple arguments. For example, the mv command takes two arguments: the file to move, and the destination to move it to:

mv file.txt newfile.txt

## Assignment

Take a look inside the worldbanc/public directory. cat the contents of the suspicious-looking key.txt file. This looks like some kind of private key... why is it in the public directory?

[Move it](https://www.ibm.com/docs/en/aix/7.3?topic=m-mv-command) from worldbanc/public/key.txt file to worldbanc/private/.

When you're done, list the contents of the worldbanc/private directory. Paste the output into the input field and submit it.

# Positional Arguments

Programming languages have functions, and functions take arguments. For example, this Python function takes two arguments: xp and level:

def print\_player(xp, level):

print("Player has", xp, "xp and is level", level)

It can then be called with two arguments:

print\_player(100, 2)

# Player has 100 xp and is level 2

In a shell, commands (programs) can also take arguments. For example, the cd command takes a single argument (the directory to change to):

cd /home/wagslane

Other commands might take multiple arguments. For example, the mv command takes two arguments: the file to move, and the destination to move it to:

mv file.txt newfile.txt

## Assignment

Take a look inside the worldbanc/public directory. cat the contents of the suspicious-looking key.txt file. This looks like some kind of private key... why is it in the public directory?

[Move it](https://www.ibm.com/docs/en/aix/7.3?topic=m-mv-command) from worldbanc/public/key.txt file to worldbanc/private/.

When you're done, list the contents of the worldbanc/private directory. Paste the output into the input field and submit it.

# Help

By convention, most production-ready CLI tools have a "help" option that will print out some information about how to use the tool. It's usually accessed with one of the following:

* --help (flag)
* -h (flag)
* help (first positional argument)

The "help" output is often easier to parse than a full man page. It's usually more of a quick start guide than a full manual.

## Assignment

You need to make some network requests as part of this audit: you want to make sure that this machine you're auditing only has access to authorized systems.

To make those requests, you'll need to know how to use [curl](https://curl.se/). It's a cli tool that lets you make network requests from the command line.

Print out the help menu for curl.

Paste the command you used to print the menu (not the menu itself) into the input field and submit your answer.

# Exit Codes

[Exit codes](https://en.wikipedia.org/wiki/Exit_status) (sometimes called "return codes" or "status codes") are how programs communicate back whether they ran successfully or not.

0 is the exit code for success. Any other exit code is an error. 9 times out of 10, if a non-zero exit code is returned (meaning an error) it will be 1, which is the "catch-all" error code.

Programs that call other programs use error codes to figure out if execution was successful. For example, if the Boot.dev server program exits with a non-zero exit code, we have another program that will automatically restart it and log the error.

In a shell, you can access the exit code of the last program you ran with the question mark (?) variable. For example, if you run a program that exits with a non-zero exit code, you can see what it was with the echo command:

ls ~

echo $?

# 0

ls /does/not/exist

echo $?

# 1

## Assignment

Commands will usually exit with error codes if they've been run without the proper arguments or configuration. For example, the private/bin/warn.sh script will exit with a non-zero code if you don't have the required environment variables set.

Run the warn.sh script without setting the two environment variables (WARN\_MESSAGE and WARN\_FROM\_NAME) that it requires.

Run the echo command to see the exit code. Paste the exit code in the text box and submit your answer.

## Tips

* You can use the unset command to unset an environment variable:

unset ENV\_VAR\_NAME

Alternatively, you can set the environment variable to an empty string:

export ENV\_VAR\_NAME=""

# Standard Output

You might not even know it yet, but you're already a pro at using standard output. You've been using it since you started the first exercise in this course.

["Standard Output"](https://en.wikipedia.org/wiki/Standard_streams#Standard_output_%28stdout%29), usually called "standard out" or "stdout", is the default place where programs print their output. It's just a stream of data that prints to your terminal, but we'll talk later about how it can be redirected to other places.

All programming languages have a simple way to print to stdout. In Python, it's the print function:

print("Hello world")

# Hello world

In a shell, it's the echo command:

echo "Hello world"

# Hello world

## Assignment

Use the grep command to print all the lines in the worldbanc/private/transactions/2020.csv file that contain the word "Marshal" to stdout.

Copy the output into the input field and submit your answer.

# Standard Error

["Standard Error"](https://en.wikipedia.org/wiki/Standard_streams#Standard_error_%28stderr%29), usually called "stderr", is the same thing as standard output, but for error messages. It's a stream completely separate from stdout so that you can redirect it to a different place if need be, but by default, it prints to your terminal just like stdout.

## Redirecting streams

You can redirect stdout and stderr to different places using the > and 2> operators. > redirects stdout, and 2> redirects stderr.

### Redirect stdout to a file

echo "Hello world" > hello.txt

cat hello.txt

# Hello world

### Redirect stderr to a file

cat doesnotexist.txt 2> error.txt

cat error.txt

# cat: doesnotexist.txt: No such file or directory

## Assignment

There is a process\_transactions.sh script in the worldbanc/private/bin directory.

* It accepts a path to a transactions CSV file as a [positional argument](https://en.wikipedia.org/wiki/Command-line_interface#Arguments)
* It prints modern transactions (after the year 2000) to stdout, and old transactions (before the year 2000) to stderr

Run the script, and pass it the path to the 2020.csv file in the worldbanc/private/transactions directory. Be sure to redirect stderr to a temporary file called /tmp/worldbanc.log.

Next, cat the /tmp/worldbanc.log file to see the old transactions then paste that output into the input field and submit your answer.

## Tip

Files in the [temporary directory](https://en.wikipedia.org/wiki/Temporary_folder) (/tmp) are deleted by your system routinely. It's a great place to store temporary files that you don't need to keep around.

# Standard In

If there's a standard output, there must be a standard input, right?

["Standard Input"](https://en.wikipedia.org/wiki/Standard_streams#Standard_input_%28stdin%29), usually called "standard in" or "stdin", is the default place where programs read their input. It's just a stream of data that programs can read from as they run.

All programming languages have a simple way to read from stdin. In Python, it's the input function:

# execution stops until the user types

# something (in this case "Lane") and presses enter

name = input("What is your name? ")

print("Hello,", name)

# Hello, Lane!

## Assignment

Run the worldbanc/private/bin/worldbanc.sh program again. Notice that it makes use of stdin to read your name and email.

Take a closer look at the code in the worldbanc.sh file. What command does it use to read from stdin?

Copy the command's name into the input field and submit your answer.

# Standard In

If there's a standard output, there must be a standard input, right?

["Standard Input"](https://en.wikipedia.org/wiki/Standard_streams#Standard_input_%28stdin%29), usually called "standard in" or "stdin", is the default place where programs read their input. It's just a stream of data that programs can read from as they run.

All programming languages have a simple way to read from stdin. In Python, it's the input function:

# execution stops until the user types

# something (in this case "Lane") and presses enter

name = input("What is your name? ")

print("Hello,", name)

# Hello, Lane!

## Assignment

Run the worldbanc/private/bin/worldbanc.sh program again. Notice that it makes use of stdin to read your name and email.

Take a closer look at the code in the worldbanc.sh file. What command does it use to read from stdin?

Copy the command's name into the input field and submit your answer.

# Piping

One of the most beautiful things about the shell is that you can [pipe](https://en.wikipedia.org/wiki/Pipeline_%28Unix%29) the output of one program into the input of another program. With this one simple concept, you can run incredibly powerful automation tasks.

## Pipe

The pipe operator is |. It's the character that looks like a vertical line. It's usually on the same key as the backslash (\) above the enter key. The pipe operator takes the stdout of the program on the left and "pipes" it into the stdin of the program on the right.

echo "Have you heard the tragedy of Darth Plagueis the Wise?" | wc -w

# 10

In the example above, the echo command sends "Have you heard the tragedy of Darth Plagueis the Wise?" to stdout.

However, instead of that text being sent to your terminal, the pipe operator pipes it into the wc (word count) command. The wc command counts the number of words in the input it receives. The -w flag tells wc to only count words.

This only works because the wc command, like most shell commands, can optionally read from stdin instead of a filepath.

## Assignment

We need to figure out just how many transactions "Bob" has been involved in... Bob's looking sus fr.

Use the grep command to find all lines in all of the transactions that include the word "Bob". You'll want to include all of the files in the worldbanc/private/transactions directory, but not the files in the backups directory. Use the --exclude-dir flag:

grep -R "Bob" PATH\_TO\_TRANSACTIONS\_DIR --exclude-dir="backups"

Then, pipe the output of that command into the wc -l command to count the number of lines.

Paste the number of transactions (just the number!) into the input field and submit your answer.

# Interrupt

Sometimes a program will get stuck and you'll want to stop it. Common reasons for this are:

* You made a typo in the command and it's not doing what you want
* It's trying to access the internet but you're not connected
* It's processing too much data and you don't want to wait for it to finish
* There is a bug in the program causing it to hang

In these cases, you can stop the program by pressing ctrl + c. This sends a "SIGINT" signal to the program, which tells it to stop.

## Assignment

Inside the worldbanc/private/bin/ directory, there is a program called malicious.sh. Sounds scary!

As a good security auditor, and knowing that you are on a secure system with no internet access, you want to see what this program does! Go ahead and run it!

It will run forever if you let it... use ctrl + c to send a SIGINT signal to the program and stop it. Paste the last line of its output into the input box and submit your answer.

# Kill

Sometimes a program is in such a bad state (or is so malicious) that it doesn't respond to the SIGINT, in which case the best option is to use another shell session (new terminal window) to manually [kill](https://www.ibm.com/docs/en/aix/7.3?topic=k-kill-command) the program.

## Syntax

kill PID

PID stands for "process ID". Every process that's running on your machine has a unique ID. The [ps](https://www.ibm.com/docs/en/zos/3.1.0?topic=jobs-using-ps-command), "process status" command can be used to list the processes running on your machine, and their IDs:

ps aux

The "aux" options just mean "show all processes, including those owned by other users, and show extra information about each process".

## Assignment

You did some digging and found out that the malicious program actually has another mode: "force".

1. Run the program again in force mode:

./malicious.sh force

1. Try to kill the program using CTRL+C. You shouldn't be able to kill it. Copy the line that was printed to your terminal when trying to kill the malicious program with ctrl+c. Paste it into the input field, but **do not submit yet**.
2. Open a second terminal window and pipe (this thing: |) the output of ps aux into grep malicious.sh to find the process ID of the malicious program. You will probably get 2 results: one for the malicious program, and one for the grep command itself. **Make sure you use the correct PID, and that you don't miss any characters.** You don't want to accidentally kill a system-critical process.
3. Once you have the PID (it should just be a number like "93838") kill it:

kill PID

1. Once you've killed the program, you should see your original terminal window return to a new prompt because it stopped running the malicious program.
2. With control of your terminal restored, **submit the message the malicious program** printed in step 2 in your text input.

## Tip

Run ps aux | head -n 1 to see the header row of the ps aux table.

!

# Unix Philosophy

The [Unix Philosophy](https://en.wikipedia.org/wiki/Unix_philosophy) is a simple set of principles that have guided the development of Unix-like operating systems for decades. It can be summarized as:

1. Write programs that do one thing and do it well.
2. Write programs to work together.
3. Write programs to handle text streams, because that is a universal interface.

## 1. Write programs that do one thing and do it well

This is why programs like ls, grep, and less exist. They do one thing, and they do it well. They don't try to do too much.

* ls lists files and directories
* grep searches for text
* less displays text

## 2. Write programs to work together

Because, at least according to the Unix Philosophy, programs should do one thing and do it well, it's easy to write programs that work together. For example, you can use grep to search for text in a file, and then pipe the output of grep into less to display the results interactively:

grep "hello" some\_file.txt | less

## 3. Write programs to handle text streams, because that is a universal interface

This point is more the "how" of the previous point. Programs work together easily when they all use the same interface: text streams. A text stream is just a sequence of characters that can be read or written sequentially. In other words, a text stream is just text.

This hearkens back to the point we talked about at the beginning of this course: the shell is a command-line (text) interface. Text-based interfaces are much more powerful and extensible than graphical interfaces. That's why developers have been using them for decades, and why what we can do with them looks like magic to the uninitiated.

# Unix Philosophy

The [Unix Philosophy](https://en.wikipedia.org/wiki/Unix_philosophy) is a simple set of principles that have guided the development of Unix-like operating systems for decades. It can be summarized as:

1. Write programs that do one thing and do it well.
2. Write programs to work together.
3. Write programs to handle text streams, because that is a universal interface.

## 1. Write programs that do one thing and do it well

This is why programs like ls, grep, and less exist. They do one thing, and they do it well. They don't try to do too much.

* ls lists files and directories
* grep searches for text
* less displays text

## 2. Write programs to work together

Because, at least according to the Unix Philosophy, programs should do one thing and do it well, it's easy to write programs that work together. For example, you can use grep to search for text in a file, and then pipe the output of grep into less to display the results interactively:

grep "hello" some\_file.txt | less

## 3. Write programs to handle text streams, because that is a universal interface

This point is more the "how" of the previous point. Programs work together easily when they all use the same interface: text streams. A text stream is just a sequence of characters that can be read or written sequentially. In other words, a text stream is just text.

This hearkens back to the point we talked about at the beginning of this course: the shell is a command-line (text) interface. Text-based interfaces are much more powerful and extensible than graphical interfaces. That's why developers have been using them for decades, and why what we can do with them looks like magic to the uninitiated.

1: Package Managers

sharpshooter0

milestone17

daily streak2

0

sharpshooter armor0

0

0

frozen flame0

You're in guest mode!

# Package Managers

A package manager is a software tool that helps you install other software. Its primary functions include:

* Downloading software from official sources
* Installing software
* Updating software
* Removing software
* Managing dependencies

As a developer, you'll frequently use package managers to get access to the software you need to get your work done.

## APT (Ubuntu)

APT, or "Advanced Package Tool", is the primary package manager for Ubuntu. To be fair, you can use other package managers on Ubuntu, but APT is the default and most common.

If you're on WSL and Ubuntu, you'll be using APT. If you're on another Linux setup, I pray you already know what package manager you're using. If you're on WSL or Ubuntu, check to make sure you have APT installed by running the following command:

apt --version

## Brew (Mac OS)

There isn't a "default" package manager for Mac OS. The most popular (but unofficial) package manager is [Homebrew](https://brew.sh/).

If you're on Mac OS, and you don't have Homebrew installed, you can install it by running the following command:

/bin/bash -c "$(curl -fsSL https://raw.githubusercontent.com/Homebrew/install/HEAD/install.sh)"

See the [Homebrew](https://brew.sh/) site for more information if needed.

## Assignment

Let's install something! [Neovim](https://neovim.io/) is a hyperextensible (and newer) version of Vim, which is a "new" version of Vi, which is a popular command line text editor.

### WSL/Ubuntu Instructions

First, make sure that apt itself is up to date and will install the latest version of Neovim. Run the following command:

sudo apt update

This command only upgrades apt, it doesn't upgrade anything else.

Next, install Neovim:

sudo apt install neovim

### Mac OS Instructions

If you're using Homebrew, on Mac, you can install Neovim by running the following command:

brew install neovim

### Other Installation Methods

If the above methods don't work, you can try [other installation methods](https://github.com/neovim/neovim/blob/master/INSTALL.md).

## Check Installation

Let's make sure we installed it properly. Check your version:

nvim --version

Paste the output into the input field and submit it.

# Using Neovim

This isn't a course on [Neovim](https://neovim.io/) (we'd need **at least** 420 more lessons for that), but let's at least figure out how to edit a file and exit the program.

Don't be fooled, exiting Vim (or Neovim) is one of the greatest hurdles you must overcome as a developer. It's a rite of passage. It's a badge of honor.

## Assignment

There's a public file containing a company password!

Use Neovim to open the worldbanc/public/company\_info.md file. It's a text file that's using [markdown](https://www.markdownguide.org/) syntax.

You can open a file with Neovim by passing in the file path as an argument:

nvim FILEPATH\_HERE

Once it's open, you might notice that you can't type anything. That's because you're in "normal" mode, the mode for navigating and manipulating text. Use the arrow keys (I know, Vim sacrilege) to move your cursor down to the last line.

Enter "insert" mode by pressing the i key. You should see -- INSERT -- appear at the bottom of the screen. You should now be able to delete the password. Replace just the password with the text "REDACTED" in all caps, no quotes.

When you're done, press the esc key to return to normal mode. Then type :w and press enter to save your changes. Finally, type :q and press enter to quit Neovim.

You've successfully edited a file (and escaped from) Neovim!

Use cat to view the contents of the edited file. Paste the contents into the input field and submit it.

# Webi

There is one more package manager (actually more of an anti-package manager) I want to introduce you to: [Webi](https://webinstall.dev/). Also known as webinstall.dev.

Webi lets you install command line tools directly from the web, with no need for a local command line tool like apt or brew. You don't need to install webi itself at all, instead, you just run a shell command that downloads and runs the tool's official installer script.

## Assignment

Let's install [lsd](https://github.com/lsd-rs/lsd), a more modern version of the ls command. It's essentially ls with a bunch of extra features.

1. Go to [Webi](https://webinstall.dev/) and look for the lsd (LSDeluxe) entry. Navigate to [its page](https://webinstall.dev/lsd/) and select the correct instructions to install it on your system (Linux/WSL or MacOS). You should simply need to copy/paste and run a single command in your terminal.
2. Make sure lsd works by listing all the files in the worldbanc/private/transactions directory.
3. Try running lsd with the --tree flag on the worldbanc/private directory. One of the coolest features of lsd is that you can view your files in a tree-like structure.
4. Run it again with both --tree and --classic flags on the worldbanc/private/transactions directory. Paste the output into the input field and submit it.

Optional Icons: You must install [the nerdfont](https://webinstall.dev/nerdfont/) and update the font in your terminal's settings for lsd to show icons.

### Note on security

As a general rule, it's not smart to pipe a shell script from the internet directly into your terminal, because that script could do malicious things. That said, as long as you're doing it via HTTPS (which you are), and you trust the source (in this case webinstall.dev), it's not as big of a concern. If you want to hear more about it, here's the [section of the Backend Banter podcast](https://youtu.be/zSkDandxcQ0?t=1447) episode where the creator talks about it more in detail.

# VS Code

Up to this point in Boot.dev, you've written code in our in-browser editor and you've used a "standard" terminal. Some developers do their work inside a terminal and use a terminal editor like [Neovim](https://neovim.io/), but a lot of developers, myself included, prefer to work in a more visual editor. One that doesn't take months to learn how to use.

Enter [VS Code](https://code.visualstudio.com/). It's a fairly lightweight editor that's easy to use for beginners, but powerful enough for professionals. VS Code (not to be confused with Visual Studio) is the [most popular](https://survey.stackoverflow.co/2024/technology#1-integrated-development-environment) code editor among developers today, and for good reason! I love it because it:

* Is easy to use for beginners
* Is powerful enough for professionals
* Is flexible through a marketplace of plugins
* Has out-of-the-box support for practically every popular coding language
* It visibly upsets Vim users who think they're better than everyone else

## Assignment

If you don't already have a preferred editor, [install VS Code on your machine](https://code.visualstudio.com/) now.

# WSL in VS Code

If you're on Mac or Linux (non-WSL), you can skip this step. However, if you're on WSL, you need to [set up VS Code to work with WSL](https://code.visualstudio.com/docs/remote/wsl#_from-vs-code). This will allow you to use the full power of your Linux filesystem within VS Code.

## Assignment

1. Open VS Code and navigate to the "Extensions" menu on the left-hand toolbar.
2. Search for "WSL" and install [the extension created by Microsoft](https://marketplace.visualstudio.com/items?itemName=ms-vscode-remote.remote-wsl).
3. In the very bottom-left corner of VS Code, there should be a green or blue button. Click on that and select "Connect to WSL using Distro" and select "Ubuntu".

You should now have a new WSL-ready VS Code editor! You can close the other VS Code window. I recommend pinning this new window to your task-bar so that you can always open the WSL-enabled version of VS Code in one click. You won't have access to your Linux filesystem if you're in the "regular" VS Code window, so I'd just avoid it completely.

1. Select "file" -> "open folder" and navigate to your worldbanc directory and open it. You should see all your files and directories in the sidebar on the left. You now have an editor that you can use for your future projects!

Note: For more detailed instructions, see the VS Code WSL extension [*Getting Started*](https://code.visualstudio.com/docs/remote/wsl#_getting-started) page.

You're good to move on.

4



0

**10**

Ch 1: What is Functional Programming?

1: What is Functional Programming?

sharpshooter0

milestone17

daily streak2

0

sharpshooter armor0

0

0

frozen flame0

# What is Functional Programming?

Functional programming is a style (or "paradigm" if you're pretentious) of programming where we compose functions instead of mutating state (updating the value of variables).

* [Functional programming](https://en.wikipedia.org/wiki/Functional_programming) is more about declaring what you want to happen, rather than how you want it to happen.
* [Imperative](https://en.wikipedia.org/wiki/Imperative_programming) (or procedural) programming declares both the what and the how.

**Example of imperative code:**

car = create\_car()

car.add\_gas(10)

car.clean\_windows()

**Example of functional code:**

return clean\_windows(add\_gas(create\_car()))

The important distinction is that in the functional example, we never change the value of the car variable, we just compose functions that return new values, with the outermost function, clean\_windows in this case, returning the final result.

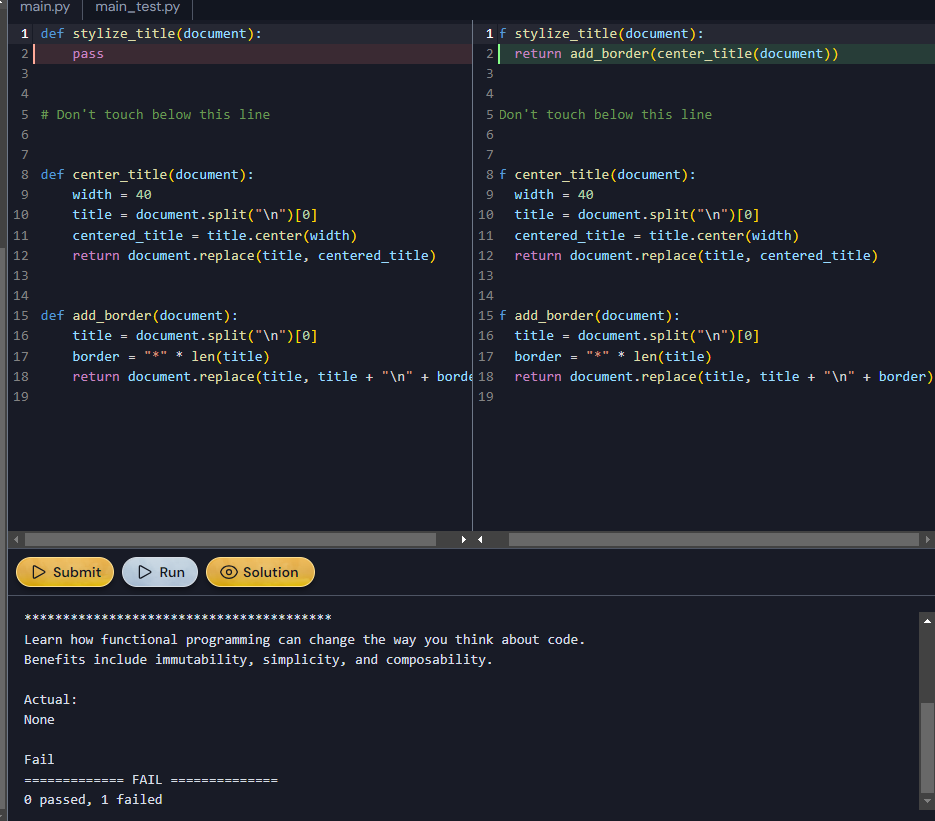
## Doc2Doc

In this course, we're working on "Doc2Doc", a command line tool for converting documents from one format to another. If you're familiar with [Pandoc](https://pandoc.org/), the idea is similar.

## Assignment

Complete the stylize\_title function. It should take a single string as input, and return a single string as output. The returned string should have both the title centered and a border added.

* Use the provided functions center\_title and add\_border.
* Center the title before adding the border.
* Do not create any variables.
* Use only 1 line of code in the function body.



Main\_test.py

from main import \*

run\_cases = [

(

"""The Importance of FP

Learn how functional programming can change the way you think about code.

Benefits include immutability, simplicity, and composability.""",

""" The Importance of FP

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

Learn how functional programming can change the way you think about code.

Benefits include immutability, simplicity, and composability.""",

),

]

submit\_cases = run\_cases + [

(

"""Short Title

Equally short story""",

""" Short Title \n\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

Equally short story""",

),

(

"""DocToDoc: A Guide

Understanding the art of document conversion.

We write cool functional code to make it happen.""",

""" DocToDoc: A Guide

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

Understanding the art of document conversion.

We write cool functional code to make it happen.""",

),

]

def test(input1, expected\_output):

print("---------------------------------")

print(f"Inputs:")

print(f" \* document: {input1}\n")

print(f"Expecting:\n{expected\_output}\n")

result = stylize\_title(input1)

print(f"Actual:\n{result}\n")

if result == expected\_output:

print("Pass")

return True

print("Fail")

return False

def main():

passed = 0

failed = 0

for test\_case in test\_cases:

correct = test(\*test\_case)

if correct:

passed += 1

else:

failed += 1

if failed == 0:

print("============= PASS ==============")

else:

print("============= FAIL ==============")

print(f"{passed} passed, {failed} failed")

test\_cases = submit\_cases

if "\_\_RUN\_\_" in globals():

test\_cases = run\_cases

main()

# Why Python?

Frankly, Python is not the best language for functional programming. Reasons include:

1. No [static typing](https://developer.mozilla.org/en-US/docs/Glossary/Static_typing).
2. Everything is [mutable](https://en.wikipedia.org/wiki/Immutable_object).
3. No [tail call optimization](https://exploringjs.com/es6/ch_tail-calls.html).
4. [Side effects](https://en.wikipedia.org/wiki/Side_effect_(computer_science)) are common.
5. Imperative and OOP styles abound in popular libraries.
6. [Purity](https://en.wikipedia.org/wiki/Pure_function) is not enforced (and sometimes not even encouraged).
7. [Sum Types](https://en.wikipedia.org/wiki/Algebraic_data_type) are hard to define.
8. [Pattern matching](https://en.wikipedia.org/wiki/Pattern_matching) is weak at best.

So seriously, why are we using Python? One reason trumps all others: you already know Python. Python is a great choice for learning coding basics, OOP, Algorithms, and Data Structures, and the tradeoff of learning a new language at this point in the curriculum isn't worth it.

We can still cover the most important concepts of functional programming in Python, even if we have to jump through a hoop or two to do it. Functional programming is a paradigm of useful techniques for writing better code, and they apply to all languages, not just purely functional ones.

Note: We also plan to release a "Functional Programming 2" course in a more functional language. Likely one of these:

* [Haskell](https://www.haskell.org/)
* [OCaml](https://ocaml.org/)
* [Elixir](https://elixir-lang.org/)

# Immutability

In FP, we strive to make data [*immutable*](https://en.wikipedia.org/wiki/Immutable_object). Once a value is created, it cannot be changed. Mutable data, on the other hand, can be changed after it's created.

## Who cares?

Immutable data is easier to think about and work with. When 10 different functions have access to the same variable, and you're debugging a problem with that variable, you have to consider the possibility that any of those functions could have changed the value.

When a variable is immutable, you can be sure that it hasn't changed since it was created. It's a helluva lot easier to work with.

Generally speaking, immutability means fewer bugs and more maintainable code.

## Tuples vs Lists

Tuples and lists are both ordered collections of values, but tuples are immutable and lists are mutable.

You can append to a list, but you can not append to a tuple. You can create a new copy of a tuple using values from an existing tuple, but you can't change the existing tuple.

### Lists are mutable

ages = [16, 21, 30]

# 'ages' is being changed in place

ages.append(80)

# [16, 21, 30, 80]

### Tuples are immutable

ages = (16, 21, 30)

more\_ages = (80,) # note the comma! It's required for a single-element tuple

# 'all\_ages' is a brand new tuple

all\_ages = ages + more\_ages

# (16, 21, 30, 80)

## Assignment

The add\_prefix function accepts 2 arguments:

* "document": a string
* "documents": the current tuple of strings

It should do 2 things:

* Add a prefix of X. to the beginning of the new document, where X is the next index in the tuple. (The first document should be 0. , next should be 1. , etc.)
* Return the documents tuple with the new document added to the end.

1. **Run the code to see the error.** Whoever wrote this code assumed that documents is a list, but it's a tuple!
2. **Fix the bug.** Instead of attempting to mutate the input tuple, create a brand new tuple with the new document added to the end and return that.

def add\_prefix(document, documents):

prefix = f"{len(documents)}. "

new\_doc = prefix + document

documents.append(new\_doc)

return documents

from main import \*

run\_cases = [

(

("hello there", "sonny", "how ya doing"),

("0. hello there", "1. sonny", "2. how ya doing"),

)

]

submit\_cases = run\_cases + [

(

("go", "python", "java", "javascript"),

("0. go", "1. python", "2. java", "3. javascript"),

),

(

("boots", "everyone else"),

("0. boots", "1. everyone else"),

),

]

def test(input1, expected\_output):

print("---------------------------------")

print(f"Inputs:")

print(f" \* document: {input1}")

print(f"Expecting: {expected\_output}")

try:

documents = ()

for doc in input1:

documents = add\_prefix(doc, documents)

except Exception as e:

documents = f"Error: {e}"

print(f"Actual: {documents}")

if documents == expected\_output:

print("Pass")

return True

print("Fail")

return False

def main():

passed = 0

failed = 0

for test\_case in test\_cases:

correct = test(\*test\_case)

if correct:

passed += 1

else:

failed += 1

if failed == 0:

print("============= PASS ==============")

else:

print("============= FAIL ==============")

print(f"{passed} passed, {failed} failed")

test\_cases = submit\_cases

if "\_\_RUN\_\_" in globals():

test\_cases = run\_cases

main()

there are many parts of functional

programming recursion pure functions

immutability higher order functions and

these are just a few of the concepts but

I'd argue that the main purpose of

functional programming is to make our

code more declarative CSS I know

disgusting uh is actually a great

example of declarative language take a

look at this simple snippet of CSS that

turns the color of a button red there

are no step-by-step procedures we simply

declare I want button buttons to be read

the browser then takes that declaration

and implements it it applies it so that

all the buttons on the website are

actually rendered as read on the screen

now let's take a look at a similar

example that's implemented in procedural

non-declarative python first we get

right into the nitty-gritty of the

implementation details by importing the

tkinter library next we'll create a new

guey window we'll set the size of the

window we draw a red button on the

screen and finally we call the main Loop

that keeps the window open and listens

for any incoming events

the key difference here is that in the

procedural python example we're very

concerned about the how in addition to

the what when we're writing declarative

code we really just want to worry about

the what right the fact that we want red

buttons good old mathematics is another

great example of a declarative language

that's why functional programming is

often more popular with mathematicians

take a look at this equation x = \*

+ \* the equation equation doesn't

tell us in which order we need to carry

out each computation on our hardware and

it also doesn't tell us exactly where we

should store each value in the memory on

our computer we could write the code

this way or we could swap the ordering

around like this and we'd get the same

result either way purely functional code

is mostly a pipe train because at the

end of the day somewhere we need to

specify how to get stuff done but we can

continuously try to make our code more

and more declarative which means more

and more functional by hiding the

implementation details at a lower level

so the goal of the functional

programming style is just to bring us

closer and closer to this declarative

ideal

Functional programming aims to be declarative. We prefer to declare what we want the computer to do, rather than muck around with the details of how to do it.

Let's take an extreme example and pretend we wanted to style a webpage with [CSS](https://developer.mozilla.org/en-US/docs/Web/CSS) (Obviously a hypothetical because, well, why would anyone want to work on the frontend???)

## Declarative styling

The following CSS changes all [button](https://developer.mozilla.org/en-US/docs/Web/HTML/Element/button) elements to have red text:

button {

color: red;

}

It does not execute line-by-line like an imperative language. Instead, it simply declares the desired style, and it's up to a web browser to figure out how to apply and display it.

## Imperative styling

Unlike functional programming (and CSS), a lot of code is imperative. We write out the exact step-by-step implementation details. This Python script draws a red button on a screen using the [Tkinter](https://docs.python.org/3/library/tkinter.html) library:

from tkinter import \* # first, import the library

master = Tk() # create a window

master.geometry("200x100") # set the window size

button = Button(master, text="Submit", fg="red").pack() # create a button

master.mainloop() # start the event loop

# It's Math

Functional programming tends to be popular amongst developers with a strong mathematical background. After all, a math equation isn't procedural: it's declarative. Take the following math equation:

avg = Σx/N

To put this calculation in plain English:

1. Σ is just the Greek letter [Sigma](https://en.wikipedia.org/wiki/Sigma), and it represents "the [sum](https://en.wikipedia.org/wiki/Summation) of a collection".
2. x is the collection of numbers we're averaging.
3. N is the number of elements in the collection.
4. avg is equal to the sum of all the numbers in collection "x" divided by the number of elements in collection "x".

So, the equation really just says that avg is the average of all the numbers in collection "x". This math equation is a declarative way of writing "calculate the average of a list of numbers". Here's some imperative Python code that does the same thing:

def get\_average(nums):

total = 0

for num in nums:

total += num

return total / len(nums)

However, with functional programming, we would write code that's a bit more declarative:

def get\_average(nums):

return sum(nums) / len(nums)

Here we're not keeping track of state (the total variable in the first example is ["stateful"](https://en.wikipedia.org/wiki/State_(computer_science)#:~:text=In%20information%20technology%20and%20computer,known%20as%20its%20state%20space.)). We're simply composing functions together to get the result we want.

## Assignment

In the world of document conversion, we sometimes need to handle fonts and font sizes.

Complete the get\_median\_font\_size function. Given a list of numbers representing font sizes, return the [median](https://en.wikipedia.org/wiki/Median) of the list.

For example:

[1, 2, 3] => 2

[10, 8, 7, 5] => 7

Notice the second list is out of order. Order the list, then find the middle index, and return the middle number. If there is an even amount of numbers, return the smaller of the two middle numbers (I know it's not a true median, but good for our purposes). If the list is empty, just return None.

Here are some helpful docs:

* [sorted](https://docs.python.org/3/library/functions.html#sorted)
* [len](https://docs.python.org/3/library/functions.html#len)
* [//](https://docs.python.org/3/library/stdtypes.html#numeric-types-int-float-complex) (floor division)

To be a good little functional programmer, your code for this lesson should **not**:

1. Use loops
2. Mutate any variables (it's okay to create new ones)

def get\_median\_font\_size(font\_sizes):

if len(font\_sizes) == 0:

return None

return sorted(font\_sizes)[(len(font\_sizes) - 1) // 2]

from main import \*

run\_cases = [

([4, 3, 2, 1, 5], 3),

([20, 14, 16], 16),

([9, 11, 16, 20], 11),

]

submit\_cases = run\_cases + [

([8, 8, 8], 8),

([30, 18, 14, 22], 18),

([6, 24, 6, 6, 24, 24, 2, 1, 3], 6),

([], None),

]

def test(input, expected\_output):

print("---------------------------------")

print(f"Input: {input}")

print(f"Expected: {expected\_output}")

input\_copy = input.copy()

result = get\_median\_font\_size(input)

print(f"Actual: {result}")

if result != expected\_output:

print("Fail")

return False

if input != input\_copy:

print(f"Expected font\_sizes: {input\_copy}")

print(f"Actual font\_sizes: {input}")

print("font\_sizes was modified")

print("Fail")

return False

print("Pass")

return True

def main():

passed = 0

failed = 0

for test\_case in test\_cases:

correct = test(\*test\_case)

if correct:

passed += 1

else:

failed += 1

if failed == 0:

print("============= PASS ==============")

else:

print("============= FAIL ==============")

print(f"{passed} passed, {failed} failed")

test\_cases = submit\_cases

if "\_\_RUN\_\_" in globals():

test\_cases = run\_cases

main()

# Classes vs Functions

I run into new developers who, after learning about classes, want to use them everywhere. They assume that because they learned about functions first, functions are somehow inferior.

**Nope.** They're just different.

## Should I use functions or classes?

Here's my rule of thumb:

**If you're unsure, default to functions.** I find myself reaching for classes when I need something long-lived and stateful that would be easier to model if I could share behavior and data structure via inheritance. This is often the case for:

* Video games
* Simulations
* GUIs

The difference is:

**Classes** encourage you to think about the world as a hierarchical collection of objects. Objects bundle behavior, data, and state together in a way that draws boundaries between instances of things, like chess pieces on a board.

**Functions** encourage you to think about the world as a series of data transformations. Functions take data as input and return a transformed output. For example, a function might take the entire state of a chess board and a move as inputs, and return the new state of the board as output.

Use what feels right to you in your projects, and adjust and refactor as you improve your skills.

# Classes vs Functions

I run into new developers who, after learning about classes, want to use them everywhere. They assume that because they learned about functions first, functions are somehow inferior.

**Nope.** They're just different.

## Should I use functions or classes?

Here's my rule of thumb:

**If you're unsure, default to functions.** I find myself reaching for classes when I need something long-lived and stateful that would be easier to model if I could share behavior and data structure via inheritance. This is often the case for:

* Video games
* Simulations
* GUIs

The difference is:

**Classes** encourage you to think about the world as a hierarchical collection of objects. Objects bundle behavior, data, and state together in a way that draws boundaries between instances of things, like chess pieces on a board.

**Functions** encourage you to think about the world as a series of data transformations. Functions take data as input and return a transformed output. For example, a function might take the entire state of a chess board and a move as inputs, and return the new state of the board as output.

Use what feels right to you in your projects, and adjust and refactor as you improve your skills.

# Debugging FP

It's nearly impossible, even for tenured senior developers, to write perfect code the first time. That's why debugging is such an important skill. The trouble is, sometimes you have these "elegant" (sarcasm intended) one-liners that are tricky to debug:

def get\_player\_position(position, velocity, friction, gravity):

return calc\_gravity(calc\_friction(calc\_move(position, velocity), friction), gravity)

If the output of get\_player\_position is incorrect, it's hard to know what's going on inside that black box. Break it up! Then you can inspect the moved, slowed, and final variables more easily:

def get\_player\_position(position, velocity, friction, gravity):

moved = calc\_move(position, velocity)

slowed = calc\_friction(moved, friction)

final = calc\_gravity(slowed, gravity)

print("Given:")

print(f"position: {position}, velocity: {velocity}, friction: {friction}, gravity: {gravity}")

print("Results:")

print(f"moved: {moved}, slowed: {slowed}, final: {final}")

return final

Once you've run it, found the issue, and solved it, you can remove the print statements.

## Assignment

Fix the format\_line function. It should apply the following transformations in order:

1. Strip whitespace from the beginning and end of the line.
2. Capitalize every character in the line.
3. Remove any periods from the line.
4. Suffix the line with an ellipsis: words go here...

Run the code. You should see that some subtle bugs are present.

Break up the function to make it easier to debug. Use print() statements to see what's going on at each step.

## Tips

Be careful about whitespace! It's easy to miss in console output. I sometimes add a character, like a | to the beginning and end of a string to make whitespace more obvious when print debugging.

* [.replace(old, new)](https://docs.python.org/3/library/stdtypes.html#str.replace) can be used to replace all instances of a character in a string.
* [.upper()](https://docs.python.org/3/library/stdtypes.html#str.upper) capitalizes an entire string, [.capitalize()](https://docs.python.org/3/library/stdtypes.html#str.capitalize) capitalizes the first letter.
* [.strip()](https://docs.python.org/3/library/stdtypes.html#str.strip) removes whitespace from the beginning and end of a string, [.lstrip()](https://docs.python.org/3/library/stdtypes.html#str.lstrip) and [.rstrip()](https://docs.python.org/3/library/stdtypes.html#str.rstrip) remove whitespace from the left and right respectively.

def format\_line(line):

stripped = line.strip()

capitalized = stripped.upper()

rm\_punc = capitalized.replace(".", "")

suffixed = f"{rm\_punc}..."

return suffixed

from main import \*

run\_cases = [

(

"You can't spell America without Erica",

"YOU CAN'T SPELL AMERICA WITHOUT ERICA...",

),

("Friends don't lie.", "FRIENDS DON'T LIE..."),

(" She's our friend and she's crazy!", "SHE'S OUR FRIEND AND SHE'S CRAZY!..."),

]

submit\_cases = run\_cases + [

(" You're gonna slay 'em dead, Nance. ", "YOU'RE GONNA SLAY 'EM DEAD, NANCE..."),

]

def test(input, expected\_output):

print("---------------------------------")

print(f"Input: {input}")

print(f"Expected: {expected\_output}")

result = format\_line(input)

print(f"Actual: {result}")

if result != expected\_output:

print("Fail")

return False

print("Pass")

return True

def main():

passed = 0

failed = 0

for test\_case in test\_cases:

correct = test(\*test\_case)

if correct:

passed += 1

else:

failed += 1

if failed == 0:

print("============= PASS ==============")

else:

print("============= FAIL ==============")

print(f"{passed} passed, {failed} failed")

test\_cases = submit\_cases

if "\_\_RUN\_\_" in globals():

test\_cases = run\_cases

main()

# Functional vs OOP

Functional programming and object-oriented programming are **styles for writing code**. One isn't inherently superior to the other, but to be a well-rounded developer you should understand both well and use ideas from each when appropriate.

You'll encounter developers who love functional programming and others who love object-oriented programming. However, contrary to popular opinion, FP and OOP are not always at odds with one another. They aren't opposites. Of the four pillars of OOP, [inheritance](https://en.wikipedia.org/wiki/Inheritance_(object-oriented_programming)) is the only one that doesn't fit with functional programming.

Inheritance isn't seen in functional code due to the mutable classes that come along with it. Encapsulation, polymorphism and abstraction are still used all the time in functional programming.

When working in a language that supports ideas from both FP and OOP (like Python, JavaScript, or Go) the best developers are the ones who can use the best ideas from both paradigms effectively and appropriately.

# Functional vs OOP

Functional programming and object-oriented programming are **styles for writing code**. One isn't inherently superior to the other, but to be a well-rounded developer you should understand both well and use ideas from each when appropriate.

You'll encounter developers who love functional programming and others who love object-oriented programming. However, contrary to popular opinion, FP and OOP are not always at odds with one another. They aren't opposites. Of the four pillars of OOP, [inheritance](https://en.wikipedia.org/wiki/Inheritance_(object-oriented_programming)) is the only one that doesn't fit with functional programming.

Inheritance isn't seen in functional code due to the mutable classes that come along with it. Encapsulation, polymorphism and abstraction are still used all the time in functional programming.

When working in a language that supports ideas from both FP and OOP (like Python, JavaScript, or Go) the best developers are the ones who can use the best ideas from both paradigms effectively and appropriately.

# Toggle Case

We need to add a feature to Doc2Doc that switches the capitalization of all the words in a line.

## Assignment

Complete the toggle\_case function using [string methods](https://docs.python.org/3/library/stdtypes.html#string-methods). It takes a string as input line, and returns a string.

1. If line is in [titlecase](https://en.wikipedia.org/wiki/Title_case), convert it to all uppercase and add three "!" to the end.
2. If line is all uppercase, convert it to all lowercase except for the first letter and remove any trailing "!".
3. If line is all lowercase or only the first letter is capitalized, convert it to title case.
4. Otherwise, just return line unchanged.

## Tips

* You will have to use the link to the official Python documentation to find the right string methods. Reading documentation is a skill all developers need to master.

def toggle\_case(line):

if line.istitle():

return line.upper() + "!!!"

if line.isupper():

return line.capitalize().rstrip("!")

if len(line) > 0 and line[1:].islower():

return line.title()

return line

from main import \*

run\_cases = [

(

"live long and prosper",

"Live Long And Prosper",

),

(

"...Khan",

"...KHAN!!!",

),

("BEAM ME UP, BOOTS!", "Beam me up, boots"),

]

submit\_cases = run\_cases + [

(

"",

"",

),

(

"I aM a DoCtOr, nOt A fUnCtIoNaL pRoGrAmMeR!!",

"I aM a DoCtOr, nOt A fUnCtIoNaL pRoGrAmMeR!!",

),

(

"TO BOLDLY GO WHERE NO BEAR HAS GONE BEFORE!!!!",

"To boldly go where no bear has gone before",

),

(

"Illogical",

"ILLOGICAL!!!",

),

]

def test(input, expected\_output):

print("---------------------------------")

print(f" Input: {input}")

print(f"Expected: {expected\_output}")

result = toggle\_case(input)

print(f" Actual: {result}")

if result != expected\_output:

print("Fail")

return False

print("Pass")

return True

def main():

passed = 0

failed = 0

for test\_case in test\_cases:

correct = test(\*test\_case)

if correct:

passed += 1

else:

failed += 1

if failed == 0:

print("============= PASS ==============")

else:

print("============= FAIL ==============")

print(f"{passed} passed, {failed} failed")

test\_cases = submit\_cases

if "\_\_RUN\_\_" in globals():

test\_cases = run\_cases

main()

# Debug Hex to RGB

Doc2Doc should seamlessly convert [hex](https://en.wikipedia.org/wiki/Hexadecimal) [triplet color codes](https://en.wikipedia.org/wiki/Web_colors#Hex_triplet) to RGB values. Hex colors are an efficient means of representing color with only 6 characters. RGB values combine red, green and blue light to electronically display the entire color spectrum.

## Assignment

Debug the hex\_to\_rgb function. hex\_to\_rgb should take a hex triplet color code and return three integers for the RGB values using [int()](https://docs.python.org/3/library/functions.html#int). [int()](https://docs.python.org/3/library/functions.html#int) is not being used correctly, examine the documentation to see how to convert hexadecimal values.

Use the provided is\_hexadecimal function to check if hex\_to\_rgb's input is valid. If the input is not a six character long hexadecimal string, raise an exception "not a hex color string".

Example:

red\_val, green\_val, blue\_val = hex\_to\_rgb("A020F0")

print(red\_val)

# prints 160

print(green\_val)

# prints 32

print(blue\_val)

# prints 240

def hex\_to\_rgb(hex\_color):

if not is\_hexadecimal(hex\_color) or len(hex\_color) != 6:

raise Exception("not a hex color string")

r = int(hex\_color[:2], 16)

g = int(hex\_color[2:4], 16)

b = int(hex\_color[4:], 16)

return r, g, b

# Don't edit below this line

def is\_hexadecimal(hex\_string):

try:

int(hex\_string, 16)

return True

except Exception:

return False

from main import \*

run\_cases = [

(

"00FFFF",

(0, 255, 255),

),

(

"FFFF00",

(255, 255, 0),

),

(

"Hello!",

"not a hex color string",

),

(

"42",

"not a hex color string",

),

]

submit\_cases = run\_cases + [

(

"",

"not a hex color string",

),

(

1\_000\_000,

"not a hex color string",

),

(

"FF00FF",

(255, 0, 255),

),

(

"000000",

(0, 0, 0),

),

(

"FFFFFF",

(255, 255, 255),

),

]

def test(input, expected\_output):

print("---------------------------------")

print(f" Inputs: '{input}'")

print(f"Expected: {expected\_output}")

try:

result = hex\_to\_rgb(input)

except Exception as e:

result = str(e)

print(f" Actual: {result}")

if result != expected\_output:

print("Fail")

return False

print("Pass")

return True

def main():

passed = 0

failed = 0

for test\_case in test\_cases:

correct = test(\*test\_case)

if correct:

passed += 1

else:

failed += 1

if failed == 0:

print("============= PASS ==============")

else:

print("============= FAIL ==============")

print(f"{passed} passed, {failed} failed")

test\_cases = submit\_cases

if "\_\_RUN\_\_" in globals():

test\_cases = run\_cases

main()

# Dedupe Lists

We need to add a feature to Doc2Doc that merges two lists of items, removes any duplicates, and returns a sorted list of unique items. This is important for consolidating certain documents and metadata.

## Assignment

Complete the deduplicate\_lists function. It takes two lists as input lst1 and lst2 and an optional reverse boolean, and returns a sorted list of unique elements. If reverse is True, then the returned list should be sorted in descending order. Use [sorted](https://docs.python.org/3/library/functions.html#sorted) and pass it the [reverse](https://docs.python.org/3/howto/sorting.html#ascending-and-descending) parameter.

def deduplicate\_lists(lst1, lst2, reverse=False):

return sorted(set(lst1 + lst2), reverse=reverse)

from main import \*

run\_cases = [

(

[1, 2, 3, 4, 5],

[4, 5, 6, 7, 8],

True,

[8, 7, 6, 5, 4, 3, 2, 1],

),

(

["tent", "sleeping bag", "camp stove", "lantern", "backpack"],

["flashlight", "tent", "camp chair", "sleeping bag", "water bottle"],

False,

[

"backpack",

"camp chair",

"camp stove",

"flashlight",

"lantern",

"sleeping bag",

"tent",

"water bottle",

],

),

(

["milk", "bread", "eggs", "cheese", "apples"],

["milk", "bananas", "bread", "oranges", "cheese"],

True,

["oranges", "milk", "eggs", "cheese", "bread", "bananas", "apples"],

),

(

["soccer ball", "tennis racket", "basketball", "baseball glove"],

["baseball bat", "soccer ball", "tennis balls", "basketball", "helmet"],

False,

[

"baseball bat",

"baseball glove",

"basketball",

"helmet",

"soccer ball",

"tennis balls",

"tennis racket",

],

),

]

submit\_cases = run\_cases + [

(

["notebooks", "pencils", "backpack", "textbooks", "laptop"],

["highlighters", "notebooks", "erasers", "backpack", "calculator"],

False,

[

"backpack",

"calculator",

"erasers",

"highlighters",

"laptop",

"notebooks",

"pencils",

"textbooks",

],

),

(

["tent", "milk", "soccer ball", "notebooks"],

["bread", "tent", "swim goggles", "pencils", "milk"],

True,

[

"tent",

"swim goggles",

"soccer ball",

"pencils",

"notebooks",

"milk",

"bread",

],

),

]

def test(input1, input2, input3, expected\_output):

print("---------------------------------")

print(f"List 1: {input1}")

print(f"List 2: {input2}")

if input3:

print(f"Reversed")

print(f"Expected: {expected\_output}")

result = deduplicate\_lists(input1, input2, input3)

print(f" Actual: {result}")

if result != expected\_output:

print("Fail")

return False

print("Pass")

return True

def main():

passed = 0

failed = 0

for test\_case in test\_cases:

correct = test(\*test\_case)

if correct:

passed += 1

else:

failed += 1

if failed == 0:

print("============= PASS ==============")

else:

print("============= FAIL ==============")

print(f"{passed} passed, {failed} failed")

test\_cases = submit\_cases

if "\_\_RUN\_\_" in globals():

test\_cases = run\_cases

main()

# Functions as Values

In Python, functions are just values, like strings, integers, or objects. For example, we can assign an existing function to a variable:

def add(x, y):

return x + y

# assign the function to a new variable

# called "addition". It behaves the same

# as the original "add" function

addition = add

print(addition(2, 5))

# 7

## Assignment

With the popularity of generative AI (like ChatGPT), we need to be able to convert files into pure text to be injected into prompts.

**Complete the file\_to\_prompt function.** It should take a file dictionary and a to\_string function as inputs and return a formatted string. The provided to\_string function is responsible for converting the file dictionary into a string: you don't need to implement it, it's a value passed to your function.

However, your function should wrap the result of the to\_string function in triple backticks (```) to format it as a code block in Markdown. For example:

an example string

should become:

```

an example string

```

## Tips

Notice the two newlines in the example above! You don't need a trailing newline, but you do need the 2 newlines between the triple backticks. You can achieve this by using the newline [\n](https://en.wikipedia.org/wiki/Newline) escape character. Here's an example:

print("I wish the ring had never come to me.\nI wish none of this had happened.")

becomes:

I wish the ring had never come to me.

I wish none of this had happened.

def file\_to\_prompt(file, to\_string):

stringified = to\_string(file)

return f"```\n{stringified}\n```"

rom main import \*

def to\_string(file):

return (

f"File: {file['filename']}\n"

f"Author: {file['author\_first\_name']} {file['author\_last\_name']}\n"

f"Content: {file['content']}"

)

run\_cases = [

(

{

"filename": "essay.txt",

"content": "Dear Mr. Vernon, we accept the fact that we had to sacrifice a whole Saturday in detention for whatever it was we did wrong...",

"author\_first\_name": "Brian",

"author\_last\_name": "Johnson",

},

"```\nFile: essay.txt\nAuthor: Brian Johnson\nContent: Dear Mr. Vernon, we accept the fact that we had to sacrifice a whole Saturday in detention for whatever it was we did wrong...\n```",

),

(

{

"filename": "letter.txt",

"content": "But we think you're crazy to make us write an essay telling you who we think we are.",

"author\_first\_name": "Brian",

"author\_last\_name": "Johnson",

},

"```\nFile: letter.txt\nAuthor: Brian Johnson\nContent: But we think you're crazy to make us write an essay telling you who we think we are.\n```",

),

]

submit\_cases = run\_cases + [

(

{

"filename": "note.txt",

"content": "Does Barry Manilow know that you raid his wardrobe?",

"author\_first\_name": "John",

"author\_last\_name": "Bender",

},

"```\nFile: note.txt\nAuthor: John Bender\nContent: Does Barry Manilow know that you raid his wardrobe?\n```",

),

]

def test(input1, expected\_output):

print("---------------------------------")

print("Inputs:")

print(f" filename: {input1['filename']}")

print(f" content: {input1['content'][:30]}...") # Truncate for display

print(f" author\_first\_name: {input1['author\_first\_name']}")

print(f" author\_last\_name: {input1['author\_last\_name']}")

print(f"Expecting:\n{expected\_output}")

result = file\_to\_prompt(input1, to\_string)

print(f"Actual:\n{result}")

if result == expected\_output:

print("Pass")

return True

print("Fail")

return False

def main():

passed = 0

failed = 0

for test\_case in test\_cases:

correct = test(\*test\_case)

if correct:

passed += 1

else:

failed += 1

if failed == 0:

print("============= PASS ==============")

else:

print("============= FAIL ==============")

print(f"{passed} passed, {failed} failed")

test\_cases = submit\_cases

if "\_\_RUN\_\_" in globals():

test\_cases = run\_cases

main()

# Anonymous Functions

Anonymous functions have no name, and in Python, they're called [lambda functions](https://docs.python.org/3/reference/expressions.html#lambda) after [lambda calculus](https://en.wikipedia.org/wiki/Lambda_calculus). Here's a lambda function that takes a single argument x and returns the result of x + 1:

lambda x: x + 1

Notice that the [expression](https://docs.python.org/3/reference/expressions.html#expressions) x + 1 is returned automatically, no need for a return statement. And because functions are just values, we can assign the function to a variable named add\_one:

add\_one = lambda x: x + 1

print(add\_one(2))

# 3

Lambda functions might look scary, but they're still just functions. Because they simply return the result of an expression, they're often used for small, simple evaluations. Here's an example that uses a lambda to get a value from a dictionary:

get\_age = lambda name: {

'lane': 29,

'hunter': 69,

'allan': 17

}.get(name, 'not found')

print(get\_age('lane'))

# 29

## Assignment

Complete the file\_type\_getter function. This function accepts a list of tuples, where each tuple contains:

1. A "file type" (e.g. "code", "document", "image", etc)
2. A list of associated file extensions (e.g. ['.py', '.js'] or ['.docx', '.doc'])

First, use loops to create a dictionary that maps each file extension to its corresponding file type, based on the input tuples. For example, the resulting dictionary might be:

{

'.doc': 'text',

'.docx': 'document',

'.py': 'code',

'.jpg': 'image'

}

Next, return a lambda function that accepts a string (a file extension) and returns the corresponding file type. If the extension is not found in the dictionary, the lambda function should return "Unknown". I used the [.get](https://docs.python.org/3/library/stdtypes.html#dict.get) dictionary method to handle this.

def file\_type\_getter(file\_extension\_tuples):

file\_extensions\_dict = {}

for tup in file\_extension\_tuples:

for ext in tup[1]:

file\_extensions\_dict[ext] = tup[0]

return lambda ext: file\_extensions\_dict.get(ext, "Unknown")

from main import \*

run\_cases = [

(

[("document", [".doc", ".docx"]), ("image", [".jpg", ".png"])],

".doc",

"document",

),

(

[("document", [".doc", ".docx"]), ("image", [".jpg", ".png"])],

".png",

"image",

),

]

submit\_cases = run\_cases + [

(

[("document", [".doc", ".docx"]), ("image", [".jpg", ".png"])],

".txt",

"Unknown",

),

(

[("code", [".py", ".js"]), ("markup", [".html", ".xml"])],

".js",

"code",

),

]

def test(file\_extension\_tuples, ext, expected\_output):

try:

print("---------------------------------")

print("Input tuples:")

for file\_type, exts in file\_extension\_tuples:

print(f" {file\_type}: {exts}")

print(f"Extension: {ext}")

print(f"Expecting: {expected\_output}")

getter\_function = file\_type\_getter(file\_extension\_tuples)

result = getter\_function(ext)

print(f"Actual: {result}")

if result == expected\_output:

print("Pass")

return True

print("Fail")

return False

except Exception as e:

print("Fail")

print(e)

return False

def main():

passed = 0

failed = 0

for test\_case in test\_cases:

correct = test(\*test\_case)

if correct:

passed += 1

else:

failed += 1

if failed == 0:

print("============= PASS ==============")

else:

print("============= FAIL ==============")

print(f"{passed} passed, {failed} failed")

test\_cases = submit\_cases

if "\_\_RUN\_\_" in globals():

test\_cases = run\_cases

main()

# First Class and Higher Order Functions

A programming language "supports first-class functions" when functions are treated like any other variable. That means functions can be passed as arguments to other functions, can be returned by other functions, and can be assigned to variables.

* **First-class function:** A function that is treated like any other value
* **Higher-order function:** A function that accepts another function as an argument or returns a function

Python supports first-class and higher-order functions.

## First-class example

def square(x):

return x \* x

# Assign function to a variable

f = square

print(f(5))

# 25

## Higher-order example

def square(x):

return x \* x

def my\_map(func, arg\_list):

result = []

for i in arg\_list:

result.append(func(i))

return result

squares = my\_map(square, [1, 2, 3, 4, 5])

print(squares)

# [1, 4, 9, 16, 25]

# First Class and Higher Order Functions

A programming language "supports first-class functions" when functions are treated like any other variable. That means functions can be passed as arguments to other functions, can be returned by other functions, and can be assigned to variables.

* **First-class function:** A function that is treated like any other value
* **Higher-order function:** A function that accepts another function as an argument or returns a function

Python supports first-class and higher-order functions.

## First-class example

def square(x):

return x \* x

# Assign function to a variable

f = square

print(f(5))

# 25

## Higher-order example

def square(x):

return x \* x

def my\_map(func, arg\_list):

result = []

for i in arg\_list:

result.append(func(i))

return result

squares = my\_map(square, [1, 2, 3, 4, 5])

print(squares)

# [1, 4, 9, 16, 25]

# Map

"Map", "filter", and "reduce" are three commonly used [higher-order functions](https://en.wikipedia.org/wiki/Higher-order_function) in functional programming.

In Python, the built-in [map](https://docs.python.org/3/library/functions.html#map) function takes a function and an [iterable](https://docs.python.org/3/glossary.html#term-iterable) (in this case a list) as inputs. It returns an iterator that applies the function to every item, yielding the results.

With map, we can operate on lists without using loops and nasty stateful variables. For example:

def square(x):

return x \* x

nums = [1, 2, 3, 4, 5]

squared\_nums = map(square, nums)

print(list(squared\_nums))

# [1, 4, 9, 16, 25]

The [*list type constructor*](https://docs.python.org/3/library/stdtypes.html#list), *list()* converts the *map* object back into a standard list.

## Assignment

[Markdown](https://www.markdownguide.org/cheat-sheet/) supports two different styles of bullet points, - and \*. We prefer \*, so, we need a function to convert any - bullet points to \* bullet points.

Complete the change\_bullet\_style function. It takes a document (a string) as input, and returns a single string as output. The returned string should have any lines that start with a - character replaced with a \* character.

For example, this:

- This is a bullet

- This is a bullet

Becomes:

\* This is a bullet

\* This is a bullet

Use the built-in [map](https://docs.python.org/3/library/functions.html#map) function to apply the provided convert\_line function to each line of the input string. Use [.split()](https://docs.python.org/3/library/stdtypes.html#str.split) and [.join()](https://docs.python.org/3/library/stdtypes.html#str.join) to split the document into a list of lines, and then join the lines back together. This should preserve the original line breaks. Don't use the .replace() string method.

Examples of split and join:

# my\_document is a string with newlines

lines\_list = my\_document.split("\n")

rejoined\_doc = "\n".join(lines\_list)

def change\_bullet\_style(document):

return "\n".join(map(convert\_line, document.split("\n")))

# Don't edit below this line

def convert\_line(line):

old\_bullet = "-"

new\_bullet = "\*"

if len(line) > 0 and line[0] == old\_bullet:

return new\_bullet + line[1:]

return line

from main import \*

run\_cases = [

(

"\* Alai\n- Dink Meeker\n",

"\* Alai\n\* Dink Meeker\n",

),

(

"\* Ender Wiggin\n- Petra Arkanian\n\* Bean\n",

"\* Ender Wiggin\n\* Petra Arkanian\n\* Bean\n",

),

]

submit\_cases = run\_cases + [

(

"- Bonzo Madrid\n- Stilson\n- The Formics\n- Peter Wiggin\n- Valentine Wiggin\n- Colonel Graff\n",

"\* Bonzo Madrid\n\* Stilson\n\* The Formics\n\* Peter Wiggin\n\* Valentine Wiggin\n\* Colonel Graff\n",

),

]

def test(input\_document, expected\_output):

print("---------------------------------")

print("Input document:")

print(input\_document)

print("Expected output:")

print(expected\_output)

result = change\_bullet\_style(input\_document)

print("Actual output:")

print(result)

if result == expected\_output:

print("Pass")

return True

print("Fail")

return False

def main():

passed = 0

failed = 0

for test\_case in test\_cases:

correct = test(\*test\_case)

if correct:

passed += 1

else:

failed += 1

if failed == 0:

print("============= PASS ==============")

else:

print("============= FAIL ==============")

print(f"{passed} passed, {failed} failed")

test\_cases = submit\_cases

if "\_\_RUN\_\_" in globals():

test\_cases = run\_cases

main()

# Filter

The built-in [filter](https://docs.python.org/3/library/functions.html#filter) function takes a function and an iterable (in this case a list) and returns a new iterable that only contains elements from the original iterable where the result of the function on that item returned True.

In Python:

def is\_even(x):

return x % 2 == 0

numbers = [1, 2, 3, 4, 5, 6]

evens = list(filter(is\_even, numbers))

print(evens)

# [2, 4, 6]

## Assignment

Complete the remove\_invalid\_lines function. It accepts a document as input. It should:

1. Use the built-in filter function and a lambda to return a copy of the input document
2. Remove any lines that start with a - character.
3. Keep all other lines and **preserve trailing newlines**.

For example, this:

\* Star Wars episode 1 is underrated

- Star Wars episode 9 is fine

\* Star Wars episode 3 is the best

Should become:

\* Star Wars episode 1 is underrated

\* Star Wars episode 3 is the best

## Tips

[.join](https://docs.python.org/3/library/stdtypes.html#str.join)

"\n".join(["a", "b", "c"])

# a

# b

# c

[.startswith](https://docs.python.org/3/library/stdtypes.html#str.startswith)

s = "hello"

s.startswith("h")

# True

s.startswith("o")

# False

[.split](https://docs.python.org/3/library/stdtypes.html#str.split)

s = """hello

world"""

lines = s.split("\n")

# ['hello', 'world']

def remove\_invalid\_lines(document):

return "\n".join(

filter(lambda line: not line.startswith("-"), document.split("\n"))

)

# Reduce

The built-in [functools.reduce()](https://docs.python.org/3/library/functools.html#functools.reduce) function takes a function and a list of values, and applies the function to each value in the list, accumulating a single result as it goes.

# import functools from the standard library

import functools

def add(sum\_so\_far, x):

print(f"sum\_so\_far: {sum\_so\_far}, x: {x}")

return sum\_so\_far + x

numbers = [1, 2, 3, 4]

sum = functools.reduce(add, numbers)

# sum\_so\_far: 1, x: 2

# sum\_so\_far: 3, x: 3

# sum\_so\_far: 6, x: 4

# 10 doesn't print, it's just the final result

print(sum)

# 10

## Assignment

Complete the join and the join\_first\_sentences functions.

### join()

This is a helper function we'll use in join\_first\_sentences. It takes two inputs:

1. A "doc\_so\_far" accumulator string. It's similar to the sum\_so\_far variable in the example above.
2. A "sentence" string. This is the next string we want to add to the accumulator.

It returns the result of concatenating the "doc" and "sentence" strings together, with a period and a space in between. For example:

doc = "This is the first sentence"

sentence = "This is the second sentence"

print(join(doc, sentence))

# This is the first sentence. This is the second sentence

### join\_first\_sentences()

It accepts two arguments:

1. A list of sentence strings
2. An integer n

It should use the built-in [functools.reduce()](https://docs.python.org/3/library/functools.html#functools.reduce) function alongside your join function to return a single string: the result of joining the first n sentences in the list. It should also add a final period (but no trailing space) to the end of the final "reduced" string.

If n is zero, just return an empty string.

Use [list slicing](https://docs.python.org/3/library/stdtypes.html#common-sequence-operations) to get the first n sentences. For example:

fruits = ["apple", "banana", "cherry", "date"]

print(fruits[:2])

# ["apple", "banana"]

Here's an example of the expected behavior:

joined = join\_first\_sentences(

["This is the first sentence", "This is the second sentence", "This is the third sentence"],

2

)

print(joined)

# This is the first sentence. This is the second sentence.

import functools

def join(doc\_so\_far, sentence):

return doc\_so\_far + ". " + sentence

def join\_first\_sentences(sentences, n):

if n == 0:

return ""

from main import \*

run\_cases = [

(

["I don't feel safe", "Are you cussing with me?"],

2,

"I don't feel safe. Are you cussing with me?.",

),

(

["You're fantastic", "He's just another rat", "Where'd the food come from?"],

2,

"You're fantastic. He's just another rat.",

),

]

submit\_cases = run\_cases + [

(["I'm not different"], 0, ""),

(

[

"You wrote a bad song",

"This is a good idea",

"Just buy the tree",

"It's going to flood",

"Tell us what to do",

"Here comes the train",

"Are you cussing with me?",

"This is just cider",

"Get me a bandit hat",

],

3,

"You wrote a bad song. This is a good idea. Just buy the tree.",

),

]

def test(input\_sentences, input\_n, expected\_output):

print("---------------------------------")

print("Inputs:")

print(f" \* sentences: {input\_sentences}")

print(f" \* n: {input\_n}")

print("Expecting:")

print(f" \* {expected\_output}")

result = join\_first\_sentences(input\_sentences, input\_n)

print("Actual:")

print(f" \* {result}")

if result == expected\_output:

print("Pass")

return True

print("Fail")

return False

def main():

passed = 0

failed = 0

for test\_case in test\_cases:

correct = test(\*test\_case)

if correct:

passed += 1

else:

failed += 1

if failed == 0:

print("============= PASS ==============")

else:

print("============= FAIL ==============")

print(f"{passed} passed, {failed} failed")

test\_cases = submit\_cases

if "\_\_RUN\_\_" in globals():

test\_cases = run\_cases

main()

# Map, Filter, and Reduce Review

Higher-order functions like map, filter, and reduce, allow us to avoid stateful iteration and mutations of variables.

Take a look at this [imperative](https://en.wikipedia.org/wiki/Imperative_programming) code that calculates the [factorial](https://en.wikipedia.org/wiki/Factorial) of a number:

def factorial(n):

# a procedure that continuously multiplies

# the current result by the next number

result = 1

for i in range(1, n + 1):

result \*= i

return result

Here's the same factorial function using reduce:

import functools

def factorial(n):

return functools.reduce(lambda x, y: x \* y, range(1, n + 1))

In the functional example, we're just combining functions to get the result we want. There's no need to reassign variables or keep track of the program's state in a loop.

A loop is inherently stateful. Depending on which iteration you're on, the i variable has a different value.

# Map, Filter, and Reduce Review

Higher-order functions like map, filter, and reduce, allow us to avoid stateful iteration and mutations of variables.

Take a look at this [imperative](https://en.wikipedia.org/wiki/Imperative_programming) code that calculates the [factorial](https://en.wikipedia.org/wiki/Factorial) of a number:

def factorial(n):

# a procedure that continuously multiplies

# the current result by the next number

result = 1

for i in range(1, n + 1):

result \*= i

return result

Here's the same factorial function using reduce:

import functools

def factorial(n):

return functools.reduce(lambda x, y: x \* y, range(1, n + 1))

In the functional example, we're just combining functions to get the result we want. There's no need to reassign variables or keep track of the program's state in a loop.

A loop is inherently stateful. Depending on which iteration you're on, the i variable has a different value.

# Intersect

The [.intersection()](https://docs.python.org/3/library/stdtypes.html#frozenset.intersection) method calculates the intersection of two sets.

The intersection of two sets is a new set that contains all of the elements that are in both original sets

For example:

a = {1, 2, 3, 4}

b = {3, 4, 5, 6}

c = a.intersection(b)

print(c)

# {3, 4}

## Assignment

Complete the get\_common\_formats function using the .intersection() method. It should take in two arguments, formats1 and formats2, each a list of strings representing the file formats supported by two different pieces of software.

It should return a [set](https://docs.python.org/3/tutorial/datastructures.html#sets) of strings representing the file formats that are supported by both pieces of software.

def get\_common\_formats(formats1, formats2):

return set(formats1).intersection(set(formats2))

from main import \*

run\_cases = [

(["PDF", "DOCX", "TXT"], ["PDF", "MD", "HTML"], set(["PDF"])),

(

["PDF", "DOCX", "TXT", "HTML"],

["PDF", "MD", "HTML", "TXT"],

set(["PDF", "TXT", "HTML"]),

),

]

submit\_cases = run\_cases + [

(["TXT"], ["TXT"], set(["TXT"])),

(["PDF", "DOCX", "TXT"], ["JPEG", "GIF", "PNG"], set()),

(["PDF", "DOCX"], ["DOCX", "PDF", "TXT"], set(["PDF", "DOCX"])),

]

def test(formats1, formats2, expected\_output):

print("---------------------------------")

print(f"Formats for Software 1: {formats1}")

print(f"Formats for Software 2: {formats2}")

print(f"Expecting: {expected\_output}")

result = get\_common\_formats(formats1, formats2)

print(f"Actual: {result}")

if result == expected\_output:

print("Pass")

return True

print("Fail")

return False

def main():

passed = 0

failed = 0

for test\_case in test\_cases:

correct = test(\*test\_case)

if correct:

passed += 1

else:

failed += 1

if failed == 0:

print("============= PASS ==============")

else:

print("============= FAIL ==============")

print(f"{passed} passed, {failed} failed")

test\_cases = submit\_cases

if "\_\_RUN\_\_" in globals():

test\_cases = run\_cases

main()

# Zip

The [zip](https://docs.python.org/3/library/functions.html#zip) function takes two iterables (in this case lists), and returns a new iterable where each element is a tuple containing one element from each of the original iterables.

a = [1, 2, 3]

b = [4, 5, 6]

c = list(zip(a, b))

print(c)

# [(1, 4), (2, 5), (3, 6)]

## Assignment

Complete the pair\_document\_with\_format function. It takes two lists as input: doc\_names and doc\_formats. Each list contains strings. The doc\_names list contains the names of documents, and the doc\_formats list contains the file formats of the documents.

First, zip up the lists into a single list of tuples with the names as the first index and the formats as the second index in each tuple.

Next, [filter](https://docs.python.org/3/library/functions.html#filter) the list of tuples to only include tuples where the format is one of the given valid\_formats.

Return the result.

valid\_formats = [

"docx",

"pdf",

"txt",

"pptx",

"ppt",

"md",

]

# Don't edit above this line

def pair\_document\_with\_format(doc\_names, doc\_formats):

zipped = list(zip(doc\_names, doc\_formats))

return list(filter(lambda x: x[1] in valid\_formats, zipped))

from main import \*

run\_cases = [

(

(["Proposal", "Invoice", "Contract"], ["docx", "pdoof", "txt"]),

[("Proposal", "docx"), ("Contract", "txt")],

),

(

(["Presentation", "Summary"], ["pptx", "docx"]),

[("Presentation", "pptx"), ("Summary", "docx")],

),

]

submit\_cases = run\_cases + [

(([], []), []),

((["Test", "Example"], ["ppt", "docx"]), [("Test", "ppt"), ("Example", "docx")]),

(

(

["Python Cheatsheet", "Java Cheatsheet", "Malware", "Golang Cheatsheet"],

["pdf", "docx", "trash", "docx"],

),

[

("Python Cheatsheet", "pdf"),

("Java Cheatsheet", "docx"),

("Golang Cheatsheet", "docx"),

],

),

]

def test(input1, expected\_output):

print("---------------------------------")

print(f"Inputs:")

print(f" \* doc\_names: {input1[0]}")

print(f" \* doc\_formats: {input1[1]}")

print(f"Expecting: {expected\_output}")

try:

result = list(pair\_document\_with\_format(\*input1))

except Exception as e:

result = f"Error: {e}"

print(f"Actual: {result}")

if result == expected\_output:

print("Pass")

return True

print("Fail")

return False

def main():

passed = 0

failed = 0

for test\_case in test\_cases:

correct = test(\*test\_case)

if correct:

passed += 1

else:

failed += 1

if failed == 0:

print("============= PASS ==============")

else:

print("============= FAIL ==============")

print(f"{passed} passed, {failed} failed")

test\_cases = submit\_cases

if "\_\_RUN\_\_" in globals():

test\_cases = run\_cases

main()

Ch 2: First Class Functions

C3: Restore

sharpshooter0

milestone18

daily streak2

0

sharpshooter armor0

0

0

frozen flame0

This is an optional challenge

You can [skip to the next chapter](https://www.boot.dev/lessons/48e1e0f3-7153-4a56-94e2-711db955b6ca) if you want

# Restore

Doc2Doc needs a way to restore documents from saved backups. However, not all original documents may have backups, and some backups might be corrupted.

## Assignment

Complete the restore\_documents function in one line. It takes two tuples of document strings, originals and backups, as input and returns a set.

1. Convert all documents to the same case with .upper() for comparison.
2. Filter out documents that are corrupted strings of random numbers with .isdigit().
3. Return the combined originals and backups documents with any duplicates removed using a set.

from main import \*

run\_cases = [

(

(

("Mortgage", "Marriage Certificate", "Boot.dev Certificate"),

("VEHICLE TITLE", "MORTGAGE"),

),

{"MORTGAGE", "MARRIAGE CERTIFICATE", "BOOT.DEV CERTIFICATE", "VEHICLE TITLE"},

),

(

(

("1235023451345", "ANNUITY", "WATER BILL"),

("Photo Album", "Year Book"),

),

{"ANNUITY", "WATER BILL", "PHOTO ALBUM", "YEAR BOOK"},

),

]

submit\_cases = run\_cases + [

(((), ()), set()),

(

(

("RECEIPT FOR 1st AND LAST RENT", "314159", "School Loan"),

("SCOOTER REGISTRATION", "ENGLISH MAJOR DEGREE"),

),

{

"RECEIPT FOR 1ST AND LAST RENT",

"SCHOOL LOAN",

"SCOOTER REGISTRATION",

"ENGLISH MAJOR DEGREE",

},

),

]

def test(input1, expected\_output):

print("---------------------------------")

print(f"Inputs:")

print(f" \* damaged documents: {input1[0]}")

print(f" \* back-up documents: {input1[1]}")

print(f"Expecting: {expected\_output}")

try:

result = restore\_documents(\*input1)

except Exception as e:

result = f"Error: {e}"

print(f"Actual: {result}")

if result == expected\_output:

print("Pass")

return True

print("Fail")

return False

def main():

passed = 0

failed = 0

for test\_case in test\_cases:

correct = test(\*test\_case)

if correct:

passed += 1

else:

failed += 1

if failed == 0:

print("============= PASS ==============")

else:

print("============= FAIL ==============")

print(f"{passed} passed, {failed} failed")

test\_cases = submit\_cases

if "\_\_RUN\_\_" in globals():

test\_cases = run\_cases

main()

pure functions are almost always a good

idea just like using tabs for

indentation and pure functions have two

properties that make them pure they

always return the same values given the

same arguments and they cause no side

effects let's look at the first point

here's an example we have an add

function that takes a and b as input and

Returns the sum now no matter how many

times you call add with the same

arguments you'll get the same result

I'll add with two and three one billion

times in a loop and you'll go crazy

waiting for anything but five as the

result now let's take a look at a

similar but impure example first we'll

declare a global variable called result

and set it equal to zero next we'll

Define a function called add to result

now this one only takes one input we'll

call it a the first thing it does is

declare that the result value or I

should say the result variable is a

global variable in Python this is

required just so that we're allowed to

access that Global variable from within

a function next we'll add a to the

result and return that sum now the

interesting thing about this function is

that every time you call it even with

different inputs you could potentially

get a different result for example the

first time I call add to result with say

the number five I expect to get back

five because we're going to add zero to

five and return it but if I call add to

result again with the input five I won't

get five and that's what makes it impure

because the result variable now has five

in it already we'll be adding five to

five and we'll get back and this is

why pure functions are so convenient to

use in the majority of cases every time

you call them with the same inputs you

expect the same outputs you don't have

to worry about they're being hidden

random number generators uh the state of

a database or whether or not an internet

connection is available in order for the

function to work as expected and because

they're predictable they're easy to test

and debug you can write an entire Suite

of unit tests for a pure function in

assuming they pass you can expect not

just expect but they will work the same

way in production given the same inputs

if you've ever experienced the well it

works on my machine meme that's not a

problem with pure functions impure

functions are the ones that made that

Meme popular so if pure functions are so

great then why don't we just make all

functions pure well by definition pure

functions can't have side effects and a

program that has no side side effects is

effectively useless it just makes your

computer heat up but to no result if

your program has no side effects it

can't even print to the console it can't

access the internet it can't update a

database it can't even draw buttons on a

screen so what should you do well my

rule of thumb is I try to make as many

of my functions that can reasonably be

pure I try to keep them pure and I try

to keep the number of side effects in my

code contained to just a few places

usually somewhere near the top level of

my code that way when other developers

are working with my code it's obvious

where the side effects are and

everything else is fairly easy to test

and de bug

# Pure Functions

If you take nothing else away from this course, please take this: [**Pure functions**](https://en.wikipedia.org/wiki/Pure_function)**are fantastic.** They have two properties:

* They always return the same value given the same arguments.
* Running them causes no side effects

In short: **pure functions don't do anything with anything that exists outside of their scope.**

## Example of a pure function

def find\_max(nums):

max\_val = float("-inf")

for num in nums:

if max\_val < num:

max\_val = num

return max\_val

## Example of an impure function

# instead of returning a value

# this function modifies a global variable

global\_max = float("-inf")

def find\_max(nums):

global global\_max

for num in nums:

if global\_max < num:

global\_max = num

## Assignment

There's a bug in the convert\_file\_format function! As it stands, it depends on some data that isn't scoped locally within the function. Those global values are mutated by external functions and are not guaranteed to be the same every time convert\_file\_format is called.

Fix the bug by making convert\_file\_format a pure function. It should only depend on data that is scoped inside of the function.

def convert\_file\_format(filename, target\_format):

valid\_extensions = ["docx", "pdf", "txt", "pptx", "ppt", "md"]

valid\_conversions = {

"docx": ["pdf", "txt", "md"],

"pdf": ["docx", "txt", "md"],

"txt": ["docx", "pdf", "md"],

"pptx": ["ppt", "pdf"],

"ppt": ["pptx", "pdf"],

"md": ["docx", "pdf", "txt"],

}

current\_format = filename.split(".")[-1]

if (

current\_format in valid\_extensions

and target\_format in valid\_conversions[current\_format]

):

return filename.replace(current\_format, target\_format)

return None

import main as main\_

run\_cases = [

("Proposal.docx", "pdf", "Proposal.pdf"),

("Invoice.txt", "md", "Invoice.md"),

]

submit\_cases = run\_cases + [

("Presentation.ppt", "pptx", "Presentation.pptx"),

("Intro.pptx", "jpeg", None),

("Summary.md", "txt", "Summary.txt"),

("Contract.pdf", "pdoof", None),

]

def mutate\_globals():

main\_.valid\_extensions = ["docx", "txt", "pptx", "ppt", "md"]

main\_.valid\_conversions = {

"docx": ["jpeg"],

"pdf": ["docx", "txt", "md"],

"txt": ["docx"],

"ppt": ["pptx", "jpeg"],

"md": ["png"],

"jpeg": ["png"],

}

def test(input1, input2, expected\_output):

print("---------------------------------")

print(f"Inputs:")

print(f" \* filename: {input1}")

print(f" \* target\_format: {input2}")

print(f"Expecting: {expected\_output}")

result = main\_.convert\_file\_format(input1, input2)

print(f"Actual: {result}")

if result == expected\_output:

print("Pass")

return True

print("Fail")

return False

def main():

passed = 0

failed = 0

mutate\_globals()

for test\_case in test\_cases:

correct = test(\*test\_case)

if correct:

passed += 1

else:

failed += 1

if failed == 0:

print("============= PASS ==============")

else:

print("============= FAIL ==============")

print(f"{passed} passed, {failed} failed")

test\_cases = submit\_cases

if "\_\_RUN\_\_" in globals():

test\_cases = run\_cases

main()

# Pure Function Review

Pure functions have a lot of benefits. Whenever possible, good developers try to use pure functions instead of impure functions. Remember, pure functions:

* Return the same result if given the same arguments. They are [deterministic](https://en.wikipedia.org/wiki/Deterministic_system).
* Do not change the external state of the program. For example, they do not change any variables outside of their scope.
* Do not perform any [I/O operations](https://en.wikipedia.org/wiki/Input/output) (like reading from disk, accessing the internet, or writing from the console).

These properties result in pure functions being easier to test, debug, and think about.

Refer to the following examples and answer the questions.

## Example 1

def multiply\_by2(nums):

products = []

for num in nums:

products.append(num\*2)

return products

## Example 2

balance = 1000

cars = []

def buy\_car(new\_car):

cars.append(new\_car)

balance -= 69

## Example 3

import random

def roll\_die(num\_sides):

return random.randint(1, num\_sides)

# Reference vs. Value

When you pass a value into a function as an argument, one of two things can happen:

* It's passed by **reference**: The function has access to the original value and can change it
* It's passed by **value**: The function only has access to a copy. Changes to the copy within the function don't affect the original

There is a bit more nuance, but this explanation mostly works.

These types are passed by **reference**:

* Lists
* Dictionaries
* Sets

These types are passed by **value**:

* Integers
* Floats
* Strings
* Booleans
* Tuples

Most collection types are passed by reference (except for tuples) and most primitive types are passed by value.

## Example of Pass by Reference (mutable)

def modify\_list(inner\_lst):

inner\_lst.append(4)

# the original "outer\_lst" is updated

# because inner\_lst is a reference to the original

outer\_lst = [1, 2, 3]

modify\_list(outer\_lst)

# outer\_lst = [1, 2, 3, 4]

## Example of Pass by Value (immutable)

def attempt\_to\_modify(inner\_num):

inner\_num += 1

# the original "outer\_num" is not updated

# because inner\_num is a copy of the original

outer\_num = 1

attempt\_to\_modify(outer\_num)

# outer\_num = 1

## Assignment

We have a way for Doc2Doc users to set their supported formats in their settings. In memory, we store those settings as a simple dictionary:

settings = {

"docx": True,

"pdf": True,

"txt": False

}

Unfortunately, there is a bug in our code! When a new format is added or removed, it not only updates the new dictionary, but it changes the defaults themselves! That's not good. We want to create a new dictionary with the updates, not change the original.

Fix the bug by making add\_format and remove\_format pure functions that don't mutate their inputs.

## Tip

The [.copy()](https://docs.python.org/3/library/copy.html) method can be used to create a new copy of a dictionary.

def add\_format(default\_formats, new\_format):

new\_formats = default\_formats.copy()

new\_formats[new\_format] = True

return new\_formats

def remove\_format(default\_formats, old\_format):

new\_formats = default\_formats.copy()

new\_formats[old\_format] = False

return new\_formats

from main import \*

run\_cases = [

(

{"docx": True, "pdf": True},

add\_format,

"txt",

{"docx": True, "pdf": True, "txt": True},

),

(

{"md": True, "txt": False},

add\_format,

"ppt",

{"md": True, "txt": False, "ppt": True},

),

({"md": True, "txt": False}, remove\_format, "md", {"md": False, "txt": False}),

]

submit\_cases = run\_cases + [

({}, add\_format, "docx", {"docx": True}),

(

{"docx": True, "pdf": True, "txt": False},

remove\_format,

"pdf",

{"docx": True, "pdf": False, "txt": False},

),

(

{"docx": True, "pdf": True, "txt": False},

add\_format,

"jpg",

{"docx": True, "pdf": True, "txt": False, "jpg": True},

),

(

{"docx": False, "pdf": True, "txt": True},

add\_format,

"docx",

{"docx": True, "pdf": True, "txt": True},

),

]

def test(input1, formatter, input2, expected\_output):

print("---------------------------------")

print(f"Inputs:")

print(f" \* default\_formats: {input1}")

print(f" \* formatter: {formatter.\_\_name\_\_}")

print(f" \* new\_format: {input2}")

print(f"Expecting: {expected\_output}")

input1\_copy = input1.copy()

result = formatter(input1, input2)

print(f"Actual: {result}")

if result != expected\_output:

print("Fail")

return False

if input1 != input1\_copy:

print("Default\_formats was mutated!")

print("Fail")

return False

print("Pass")

return True

def main():

passed = 0

failed = 0

for test\_case in test\_cases:

correct = test(\*test\_case)

if correct:

passed += 1

else:

failed += 1

if failed == 0:

print("============= PASS ==============")

else:

print("============= FAIL ==============")

print(f"{passed} passed, {failed} failed")

test\_cases = submit\_cases

if "\_\_RUN\_\_" in globals():

test\_cases = run\_cases

main()

# Pass by Reference Impurity

Because certain types in Python are passed by reference, we can mutate values that we didn't intend to. This is a form of function impurity.

Remember, a pure function should have no side effects. It shouldn't modify anything outside of its scope, including its inputs. It should return new copies of inputs instead of changing them.

## Pure function

def remove\_format(default\_formats, old\_format):

new\_formats = default\_formats.copy()

new\_formats[old\_format] = False

return new\_formats

## Impure function

def remove\_format(default\_formats, old\_format):

default\_formats[old\_format] = False

return default\_formats

## Why do we care?

One of the biggest differences between good and great developers is how often they incorporate pure functions into their code. Pure functions are easier to read, easier to reason about, easier to test, and easier to combine. Even if you're working in an imperative language like Python, you can (and should) write pure functions whenever reasonable.

There's nothing worse than trying to debug a program where the order functions are called needs to be juuuuust right because they all read and modify the same global variable.

# Input and Output

Comic by [xkcd](https://xkcd.com/1790/).

The term "i/o" stands for input/output. In the context of writing programs, i/o refers to anything in our code that interacts with the "outside world". "Outside world" just means anything that's not stored in our application's memory (like variables).

## Examples of i/o

* Reading from or writing to a file on the hard drive
* Accessing the internet
* Reading from or writing to a database
* Even simply printing to the console is considered i/o!

All i/o is a form of "side effect".

## Assignment

In Doc2Doc, we frequently need to change the casing of some text. For example:

### TitleCase

Every Day Once A Day Give Yourself A Present

### LowerCase

every day once a day give yourself a present

### UpperCase

EVERY DAY ONCE A DAY GIVE YOURSELF A PRESENT

There is an issue in the convert\_case function, our test suite can't test its behavior because it's printing to the console (eww... a side-effect) instead of returning a value. Fix the function so that it returns the correct value instead of printing it.

def convert\_case(text, target\_format):

if not text or not target\_format:

raise ValueError(f"No text or target format provided")

if target\_format == "uppercase":

return text.upper()

if target\_format == "lowercase":

return text.lower()

if target\_format == "titlecase":

return text.title()

raise ValueError(f"Unsupported format: {target\_format}")

from main import \*

run\_cases = [

(

"Through the darkness of future past",

"uppercase",

"THROUGH THE DARKNESS OF FUTURE PAST",

),

("The magician longs to see", "lowercase", "the magician longs to see"),

]

submit\_cases = run\_cases + [

(

"One chants out between two worlds",

"titlecase",

"One Chants Out Between Two Worlds",

),

("Fire walk with me", "garbagecase", "Unsupported format: garbagecase"),

]

def test(input1, input2, expected\_output):

print("---------------------------------")

print(f"Input:")

print(f'"{input1}", {input2}')

print(f"Expecting:")

print(f'"{expected\_output}"')

try:

result = convert\_case(input1, input2)

except Exception as e:

result = str(e)

print(f"Actual:")

print(f'"{result}"')

if result == expected\_output:

print("Pass")

return True

print("Fail")

return False

def main():

passed = 0

failed = 0

for test\_case in test\_cases:

correct = test(\*test\_case)

if correct:

passed += 1

else:

failed += 1

if failed == 0:

print("============= PASS ==============")

else:

print("============= FAIL ==============")

print(f"{passed} passed, {failed} failed")

test\_cases = submit\_cases

if "\_\_RUN\_\_" in globals():

test\_cases = run\_cases

main()

# Should I i/o?

A program that doesn't do any i/o is pretty useless. What's the point of computing something if you can't see the results?

In functional programming, i/o is viewed as dirty but necessary. We know we can't eliminate i/o from our code, so we just contain it as much as possible. There should be a clear place in your project that does nasty i/o stuff, and the rest of your code can be pure.

For example, a Python program might:

1. Read a file from the hard drive as the program starts
2. Run a bunch of pure functions to analyze the data
3. Write the results of the analysis to a file on the hard drive at the end

# No-Op

A [no-op](https://en.wikipedia.org/wiki/NOP_(code)) is an operation that does... nothing.

If a function doesn't return anything, it's probably impure. If it doesn't return anything, the only reason for it to exist is to perform a side effect.

## Example no-op

This function performs a useless computation because it doesn't return anything or perform a side-effect. It's a no-op.

def square(x):

x \* x

## Example side-effect

This function performs a side effect. It changes the value of the y variable that is outside of its scope. It's impure.

y = 5

def add\_to\_y(x):

global y

y += x

add\_to\_y(3)

# y = 8

The [*global*](https://docs.python.org/3/reference/simple_stmts.html#global) keyword just tells Python to allow access to the outer-scoped *y* variable.

## print()

Even the print() function (technically) has an impure side effect! It doesn't return anything, but it does print text to the console, which is a form of i/o.

## Assignment

Complete the remove\_emphasis, remove\_emphasis\_from\_line, and remove\_emphasis\_from\_word functions. They are currently no-ops.

remove\_emphasis is the parent function. It takes a full document and removes any single or double \* characters that are at the start or end of a word. ([Emphasis](https://www.markdownguide.org/basic-syntax/#emphasis) in markdown)

For example, this:

I \*love\* markdown.

I \*\*really love\*\* markdown.

Should become:

I love markdown.

I really love markdown.

Write the helper functions, they will make the remove\_emphasis function much easier to write:

* The remove\_emphasis\_from\_line function should remove emphasis from a single line of text.
* The remove\_emphasis\_from\_word function should remove emphasis from a single word.

## Tips

For the sake of practice, try it without the .replace() string method. I used some of these built-ins:

* [str.split](https://docs.python.org/3/library/stdtypes.html#str.split)
* [str.strip](https://docs.python.org/3/library/stdtypes.html#str.strip)
* [map](https://docs.python.org/3/library/functions.html#map)
* [join](https://docs.python.org/3/library/stdtypes.html#str.join)

def remove\_emphasis\_from\_word(word):

return word.strip("\*")

def remove\_emphasis\_from\_line(line):

words = line.split()

new\_words = map(remove\_emphasis\_from\_word, words)

return " ".join(new\_words)

def remove\_emphasis(doc\_content):

lines = doc\_content.split("\n")

new\_lines = map(remove\_emphasis\_from\_line, lines)

return "\n".join(new\_lines)

from main import \*

run\_cases = [

(

"\*Don't\* panic.\n",

"Don't panic.\n",

),

(

"The \*\*answer to the ultimate question\*\* of life, the universe and everything is \*42\*\n",

"The answer to the ultimate question of life, the universe and everything is 42\n",

),

(

"For a moment, \*nothing\* happened.\nThen, after a second or so, nothing \*\*continued\*\* to happen.\n",

"For a moment, nothing happened.\nThen, after a second or so, nothing continued to happen.\n",

),

]

submit\_cases = run\_cases + [

(

"",

"",

),

(

"In the beginning the \*universe\* was created.\nThis has made a lot of people very \*angry\* and been widely regarded as a bad move.\n",

"In the beginning the universe was created.\nThis has made a lot of people very angry and been widely regarded as a bad move.\n",

),

(

"Ford, you're turning into a \*penguin\*\n",

"Ford, you're turning into a penguin\n",

),

(

"\*Space\* is big.\nYou just won't \*\*believe\*\* how vastly, hugely, mind-bogglingly big it is.\n",

"Space is big.\nYou just won't believe how vastly, hugely, mind-bogglingly big it is.\n",

),

]

def test(input\_doc, expected\_output):

print("---------------------------------")

print(f"Input document:\n{input\_doc}")

print(f"Expected output:\n{expected\_output}")

result = remove\_emphasis(input\_doc)

print(f"Actual output:\n{result}")

if result == expected\_output:

print("Pass")

return True

print("Fail")

return False

def main():

passed = 0

failed = 0

for test\_case in test\_cases:

correct = test(\*test\_case)

if correct:

passed += 1

else:

failed += 1

if failed == 0:

print("============= PASS ==============")

else:

print("============= FAIL ==============")

print(f"{passed} passed, {failed} failed")

test\_cases = submit\_cases

if "\_\_RUN\_\_" in globals():

test\_cases = run\_cases

main()

# Memoization

At its core, [memoization](https://en.wikipedia.org/wiki/Memoization) is just [caching](https://en.wikipedia.org/wiki/Cache_(computing)) (storing a copy of) the result of a computation so that we don't have to compute it again in the future.

For example, take this simple function:

def add(x, y):

return x + y

A call to add(5, 7) will always evaluate to 12. So, if you think about it, once we know that add(5, 7) can be replaced with 12, we can just store the value 12 somewhere in memory so that we don't have to do the addition operation again in the future. Then, if we need to add(5, 7) again, we can just look up the value 12 instead of doing a (potentially expensive) CPU operation.

The slower and more complex the function, the more memoization can help speed things up.

Note: It's pronounced "memOization", not "memORization". This confused me for quite a while in college, I thought my professor just didn't speak goodly...

## Assignment

Counting the words in a document can be slow, so we want to memoize it.

Complete the word\_count\_memo function. It takes two inputs:

1. A document string.
2. A memos dictionary. The keys are full document strings, and the values are the word count of the document.

It should return two values:

1. The word count of the given document
2. An updated memos dictionary.

Here are the steps to follow:

1. Create a [.copy()](https://docs.python.org/3/library/copy.html#module-copy) of the memos dictionary.
2. If the document is [in](https://docs.python.org/3/library/stdtypes.html#dict) the memos dictionary, just return the associated word count and the memos copy. No need to recompute the word count.
3. Otherwise, use the provided word\_count function to count the words in the given document.
4. Store the word count in the memos copy.
5. return the word count and the updated memos copy.

def word\_count\_memo(document, memos):

new\_memos = memos.copy()

if document in new\_memos:

return new\_memos[document], new\_memos

count = word\_count(document)

new\_memos[document] = count

return count, new\_memos

# Don't edit below this line

def word\_count(document):

count = len(document.split())

return count

from main import \*

run\_cases = [

(

"My hovercraft is full of eels",

{

"My hovercraft is full of eels": 6,

"He's a lumberjack and he's okay. He sleeps all night and he works all day": 15,

},

(

6,

{

"My hovercraft is full of eels": 6,

"He's a lumberjack and he's okay. He sleeps all night and he works all day": 15,

},

),

),

(

"Spam, spam, spam, spam, spam, spam, baked beans, spam, spam, and spam",

{},

(

12,

{

"Spam, spam, spam, spam, spam, spam, baked beans, spam, spam, and spam": 12

},

),

),

]

submit\_cases = run\_cases + [

(

"This is an ex-parrot",

{"This parrot is no more": 5},

(4, {"This parrot is no more": 5, "This is an ex-parrot": 4}),

),

(

"This doc should 'incorrectly' have 9999 words to test that the memoization is working",

{

"My hovercraft is full of eels": 6,

"This doc should 'incorrectly' have 9999 words to test that the memoization is working": 9999,

},

(

9999,

{

"My hovercraft is full of eels": 6,

"This doc should 'incorrectly' have 9999 words to test that the memoization is working": 9999,

},

),

),

]

def test(input\_document, input\_memos, expected\_output):

print("---------------------------------")

print(f"Input document:\n {input\_document}")

print(f"Input memos:")

for key, value in input\_memos.items():

print(f" {key}: {value}")

print(f"Expected word count: {expected\_output[0]}")

print(f"Expected memos:")

for key, value in expected\_output[1].items():

print(f" {key}: {value}")

input\_memos\_copy = input\_memos.copy()

result = word\_count\_memo(input\_document, input\_memos\_copy)

print(f"Actual word count: {result[0]}")

print(f"Actual memos:")

for key, value in result[1].items():

print(f" {key}: {value}")

if input\_memos\_copy != input\_memos:

print("Mutated input memos\nFail")

return False

if input\_memos == expected\_output[1] and result[1] != expected\_output[1]:

print("Expected word count from the input memos\nFail")

return False

if result != expected\_output:

print("Fail")

return False

print("Pass")

return True

def main():

test\_cases = submit\_cases

if "\_\_RUN\_\_" in globals():

test\_cases = run\_cases

passed = 0

failed = 0

for test\_case in test\_cases:

correct = test(\*test\_case)

if correct:

passed += 1

else:

failed += 1

if failed == 0:

print("============= PASS ==============")

else:

print("============= FAIL ==============")

print(f"{passed} passed, {failed} failed")

main()

# Referential transparency

Pure functions are always [referentially transparent](https://www.baeldung.com/cs/referential-transparency#referential-transparency).

"Referential transparency" is a fancy way of saying that a function call can be replaced by its would-be return value because it's the same every time. **Referentially transparent functions can be safely memoized.** For example add(2, 3) can be smartly replaced by the value 5.

The great thing about pure functions is that they can always be safely memoized. Impure functions can't be because they might do something in addition to returning a static value, or they might return different values given the same arguments.

## Should I always memoize?

No! Memoization is a tradeoff between memory and speed. If your function is fast to execute, it's probably not worth memoizing, because the amount of RAM (memory) your program will need to store the results will go way up.

It's also a bunch of extra code to write, so you should only do it if you have a good reason to.

# Custom Commands

Doc2Doc is customizable. New commands can be configured to use whichever function suits the user. However, the new commands are causing bugs in other parts of the application by mutating global values and other unintended side effects.

## Assignment

Fix the functions add\_custom\_command, add\_format, save\_document and add\_line\_break to make them pure functions without side effects. Here are the reported issues:

* add\_custom\_command: is an impure function that is mutating an input
* add\_format: is an impure function that is mutating an input
* save\_document: is an impure function that is mutating an input
* add\_line\_break: is a no-op function with a side-effect

default\_commands = {}

default\_formats = ["txt", "md", "html"]

saved\_documents = {}

# Don't edit above this line

def add\_custom\_command(commands, new\_command, function):

commands\_copy = commands.copy()

commands\_copy[new\_command] = function

return commands\_copy

def add\_format(formats, format):

formats\_copy = formats.copy()

formats\_copy.append(format)

return formats\_copy

def save\_document(docs, file\_name, doc):

docs\_copy = docs.copy()

docs\_copy[file\_name] = doc

return docs\_copy

def add\_line\_break(line):

return line + "\n\n"

from main import \*

run\_cases = [

(

("add\_format", add\_format),

default\_formats,

[("rtf",), ("csv",)],

["txt", "md", "html", "rtf", "csv"],

),

(

("save\_document", save\_document),

saved\_documents,

[

("My\_Princess\_Diaries.txt", "I can't be a princess!"),

(

"The\_Devil\_Wears\_Boots.md",

"Please, bore someone else with your questions.",

),

],

{

"My\_Princess\_Diaries.txt": "I can't be a princess!",

"The\_Devil\_Wears\_Boots.md": "Please, bore someone else with your questions.",

},

),

(

("add\_line\_break", add\_line\_break),

"It's not you, it's me.",

[()],

"It's not you, it's me.\n\n",

),

]

submit\_cases = run\_cases + [

(

("add\_format", add\_format),

default\_formats,

[

("doc",),

("docx",),

("pdf",),

],

["txt", "md", "html", "doc", "docx", "pdf"],

),

(

("save\_document", save\_document),

saved\_documents,

[

("Function\_Club.txt", "The types you own end up owning you"),

("Shrek.doc", "Functions are like onions."),

],

{

"Function\_Club.txt": "The types you own end up owning you",

"Shrek.doc": "Functions are like onions.",

},

),

(

("add\_line\_break", add\_line\_break),

"Go be free.",

[()],

"Go be free.\n\n",

),

]

def test(input1, input2, input3, expected\_output):

print("---------------------------------")

print(f"Inputs:")

print(f" \* new command: {input1[0]}")

print(f" \* starting input: {input2}")

result = input2

commands = default\_commands

input2\_length = len(input2)

default\_commands\_length = len(default\_commands)

# add and test new command

commands = add\_custom\_command(commands, \*input1)

for item in input3:

if len(item) > 0:

print(f" \* input: {item}")

result = commands[input1[0]](result, \*item)

# check result

print(f"Expecting: '{expected\_output}'")

print(f" Actual: '{result}'")

if result == expected\_output:

# check inputs not mutated

if len(input2) == input2\_length:

if len(default\_commands) == default\_commands\_length:

print("Pass")

return True

else:

print("default\_commands modified")

else:

print("Starting input modified")

print("Fail")

return False

def main():

passed = 0

failed = 0

for test\_case in test\_cases:

correct = test(\*test\_case)

if correct:

passed += 1

else:

failed += 1

if failed == 0:

print("============= PASS ==============")

else:

print("============= FAIL ==============")

print(f"{passed} passed, {failed} failed")

test\_cases = submit\_cases

if "\_\_RUN\_\_" in globals():

test\_cases = run\_cases

main()

# Sort Dates

Datetimes are infamously a [pain in the neck](https://gist.github.com/timvisee/fcda9bbdff88d45cc9061606b4b923ca) for programming. The least of the list of problems are the order of the year, month and day of a calendar date. Some countries use day-month-year format, others use year-month-day. Some [insane countries](https://iso.mit.edu/americanisms/date-format-in-the-united-states/) use month-day-year because they want everyone else to be miserable.

## Assignment

Fix the sort\_dates function. It takes as input a list of dates in "MONTH-DAY-YEAR" format and returns a list of the dates sorted in ascending order.

## Tip

The built-in [sorted](https://docs.python.org/3/library/functions.html#sorted) function might work better here than the .sort() list method. Create some function to transform the dates to more easily compare them with the sorted function.

def sort\_dates(dates):

return sorted(dates, key=format\_date)

def format\_date(date):

month, day, year = date.split("-")

return year + month + day

from main import \*

run\_cases = [

(

[

"07-21-2023",

"12-25-2022",

"01-01-2023",

"01-15-2023",

"10-31-2023",

"04-10-2023",

],

[

"12-25-2022",

"01-01-2023",

"01-15-2023",

"04-10-2023",

"07-21-2023",

"10-31-2023",

],

),

(

[

"08-17-2023",

"11-05-2022",

"02-28-2023",

"06-30-2023",

"09-19-2024",

"05-22-2023",

],

[

"11-05-2022",

"02-28-2023",

"05-22-2023",

"06-30-2023",

"08-17-2023",

"09-19-2024",

],

),

(

[

"07-04-2023",

"12-01-2024",

"01-20-2023",

"03-10-2023",

"10-05-2023",

"04-25-2023",

],

[

"01-20-2023",

"03-10-2023",

"04-25-2023",

"07-04-2023",

"10-05-2023",

"12-01-2024",

],

),

(

[

"08-12-2023",

"11-15-2022",

"02-10-2023",

"06-25-2023",

"09-05-2023",

"05-05-2023",

],

[

"11-15-2022",

"02-10-2023",

"05-05-2023",

"06-25-2023",

"08-12-2023",

"09-05-2023",

],

),

]

submit\_cases = run\_cases + [

(

[

"07-15-2024",

"12-18-2022",

"03-30-2023",

"03-20-2023",

"10-20-2023",

"04-05-2023",

],

[

"12-18-2022",

"03-20-2023",

"03-30-2023",

"04-05-2023",

"10-20-2023",

"07-15-2024",

],

),

(

[

"08-22-2023",

"11-30-2022",

"02-05-2023",

"06-10-2023",

"09-25-2023",

"05-10-2023",

],

[

"11-30-2022",

"02-05-2023",

"05-10-2023",

"06-10-2023",

"08-22-2023",

"09-25-2023",

],

),

(

[

"07-10-2024",

"12-10-2022",

"01-25-2023",

"03-05-2023",

"10-15-2023",

"04-15-2023",

],

[

"12-10-2022",

"01-25-2023",

"03-05-2023",

"04-15-2023",

"10-15-2023",

"07-10-2024",

],

),

(

[

"08-02-2023",

"11-25-2022",

"02-15-2024",

"06-05-2023",

"09-10-2023",

"05-01-2023",

],

[

"11-25-2022",

"05-01-2023",

"06-05-2023",

"08-02-2023",

"09-10-2023",

"02-15-2024",

],

),

]

def test(input1, expected\_output):

print("---------------------------------")

print(f"Input: {input1}")

print(f"Expected: {expected\_output}")

result = sort\_dates(input1)

print(f" Actual: {result}")

if result != expected\_output:

print("Fail")

return False

print("Pass")

return True

def main():

passed = 0

failed = 0

for test\_case in test\_cases:

correct = test(\*test\_case)

if correct:

passed += 1

else:

failed += 1

if failed == 0:

print("============= PASS ==============")

else:

print("============= FAIL ==============")

print(f"{passed} passed, {failed} failed")

test\_cases = submit\_cases

if "\_\_RUN\_\_" in globals():

test\_cases = run\_cases

main()

# Index Keywords

By indexing keywords, documents are better organized, enabling users to navigate through their documents more efficiently. It also reduces the computational overhead of repeatedly searching for the same terms, improving the performance and responsiveness of Doc2Doc. This is useful for notebooking, research and project management.

## Assignment

Complete the find\_keywords function. It takes as input a document string and returns a list of the keyword substrings in the document.

The keywords list is being modified at the start of code execution. Turn find\_keywords into a pure function that does not rely on a global variable.

1. Move the keywords list inside of the find\_keywords function so changes to the global scope do not change this list.
2. Filter the keywords to find the keyword substrings in the document.
3. Be sure to return the found keyword substrings as a list.

Fix the index\_keywords function. It takes a string document input and a dictionary index input, and returns a list of keywords substrings found in the document and a dictionary index with the document added if needed. The index dictionary has document strings as keys, and their values are a list of keywords found in the key string, in the order they appear in the keywords list.

1. Do not modify or return the actual input index.
2. If the document is already in the index, return its keywords and the index copy.
3. Use find\_keywords to get the keyword substrings in the document.
4. Add the document and its keyword substrings to the index copy.
5. Return the document's keyword substrings and the index copy.

## Tip

Check the test suite to understand how variables are being changed in the global scope.

def index\_keywords(document, index):

new\_index = index.copy()

if document in new\_index:

return new\_index[document], new\_index

found\_keywords = find\_keywords(document)

new\_index[document] = found\_keywords

return found\_keywords, new\_index

def find\_keywords(document):

keywords = [

"functional",

"immutable",

"declarative",

"higher-order",

"lambda",

"deterministic",

"side-effects",

"memoization",

"referential transparency",

]

return list(filter(lambda keyword: keyword in document, keywords))

import main as main\_

run\_cases = [

(

"Key parts of functional programming are higher-order functions and lambda expressions.",

{},

["functional", "higher-order", "lambda"],

{

"Key parts of functional programming are higher-order functions and lambda expressions.": [

"functional",

"higher-order",

"lambda",

]

},

),

(

"Results are deterministic by using referential transparency and the absence of side-effects.",

{},

["deterministic", "side-effects", "referential transparency"],

{

"Results are deterministic by using referential transparency and the absence of side-effects.": [

"deterministic",

"side-effects",

"referential transparency",

]

},

),

(

"Storing the results of deterministic functions uses memoization to help optimize functional code",

{},

["functional", "deterministic", "memoization"],

{

"Storing the results of deterministic functions uses memoization to help optimize functional code": [

"functional",

"deterministic",

"memoization",

]

},

),

(

"Functional programming emphasizes immutable data.",

{"Functional programming emphasizes immutable data.": ["test\_keyword"]},

["test\_keyword"],

{"Functional programming emphasizes immutable data.": ["test\_keyword"]},

),

]

submit\_cases = run\_cases + [

(

"The immutable state in functional programming ensures referential transparency.",

{},

["functional", "immutable", "referential transparency"],

{

"The immutable state in functional programming ensures referential transparency.": [

"functional",

"immutable",

"referential transparency",

]

},

),

(

"Functional programming often uses higher-order functions to handle side-effects declaratively.",

{},

["declarative", "higher-order", "side-effects"],

{

"Functional programming often uses higher-order functions to handle side-effects declaratively.": [

"declarative",

"higher-order",

"side-effects",

]

},

),

(

"A concise method for implementing higher-order functions is through lambda functions.",

{},

["higher-order", "lambda"],

{

"A concise method for implementing higher-order functions is through lambda functions.": [

"higher-order",

"lambda",

]

},

),

(

"Pure functions simplify testing by eliminating dependencies on external state.",

{},

[],

{

"Pure functions simplify testing by eliminating dependencies on external state.": [],

},

),

]

def mutate\_globals():

main\_.keywords = []

def test(document, index, expected\_keywords, expected\_index):

print("---------------------------------")

print("Inputs:")

print(f"\* {document}")

print(f"Index:")

for key, value in index.items():

print(f" {key}: {value}")

print(f"Expected Keywords: {expected\_keywords}")

print(f"Expected Index:")

for key, value in expected\_index.items():

print(f" {key}: {value}")

index\_copy = index.copy()

result\_keywords, result\_index = main\_.index\_keywords(document, index)

print(f" Actual Keywords: {result\_keywords}")

print(f" Actual Index:")

for key, value in result\_index.items():

print(f" {key}: {value}")

if index\_copy != index:

print("Fail: Mutated input index")

return False

if index == expected\_index and result\_index != expected\_index:

print("Fail: Expected keywords from the input index")

return False

if len(result\_keywords) == 0 and len(expected\_keywords) != 0:

print("Fail: the global scope keywords changed, causing this failure.")

print("How can you use the keywords without them changing in the global scope?")

return False

if (result\_keywords, result\_index) != (expected\_keywords, expected\_index):

print("Fail")

return False

print("Pass")

return True

def main():

test\_cases = submit\_cases

if "\_\_RUN\_\_" in globals():

test\_cases = run\_cases

passed = 0

failed = 0

mutate\_globals()

for test\_case in test\_cases:

correct = test(\*test\_case)

if correct:

passed += 1

else:

failed += 1

if failed == 0:

print("============= PASS ==============")

else:

print("============= FAIL ==============")

print(f"{passed} passed, {failed} failed")

main()

recursion is a word used to strike fear

into the hearts of new programmers it's

a simple concept that just takes a

second to learn how to use properly

recursion is just when a function calls

itself now that might sound fairly

straightforward and it is but it is one

of the hardest concepts for new

developers to learn so don't beat

yourself up if it takes just a little

extra practice and a few more examples

before you fully understand what's going

on let's take a look at this simple

example we've got a function called

countdown and it takes as input n an

integer first it just prints n after it

prints n it calls itself again but only

after subtracting one from n so what

happens if we call countdown and pass in

five as n well first five is printed to

the console then countdown is called

again but this time with n minus one or

- which is four so we go back to the

top of the function now we'll print n

which is four all count down again this

time with three and on and on forever so

we' figured out the recursive part but

the problem is this function will never

stop I can't stop I can't stop I can't

stop it will keep calling countdown over

and over and over decrementing by one

every time even into the negatives

countd down with negative negative -

negso what we need is a base case to

stop the recursive Madness let's add an

if statement to the top of the countdown

function if if n is less than or equal

to zero instead of printing n and

recursing again with n minus one we'll

just print the word blast off to the

console and return immediately again

stopping the recursion so what happens

now if we call countdown with n of five

first five is printed then we call

countdown again and that's because five

is greater than zero so we're not going

to trigger that base case next countdown

is called with an N of four again we're

skipping the base case because n is

greater than zero four is printed to the

console this continues all the way down

until n is less than or equal to zero so

when n is zero we'll enter that if

statement print blast off and then

return and because we're returning there

will be no more recursive calls to the

countdown function so the final result

is that when we call countdown of five

blast off is printed to the

console now let's take a look at one

more example let's say we want to write

a function called dict depth that

Returns the depth of an arbitrarily

nested dictionary for example this

dictionary only has a depth of one it

has two key value pairs but there's no

nested dictionaries within it but if you

take a look at this dictionary it has

three nested levels melee weapons

corresponds to a second dictionary which

has the key stabies which corresponds to

another nested dictionary a total of

three levels now because the dictionary

passed into our function can be

arbitrarily deep recursion is a really

good fit for this problem so let's

figure out what our function signature

should be we'll Define the new function

we'll call it dict depth and we'll pass

in two parameters D the dictionary

itself and then max depth so far which

is just going to keep track of how far

into the nested dictionary we are at

each level of recursion now this time

let's start with the base case just so

we have a different way of approaching

this problem and our base case should be

when there's no more dictionaries to

recurse into so if D is not a valid

dictionary for example if it's just a

number like the number that corresponds

with the Bose key six then we're going

to stop recursing so we'll just add this

if statement that says if D is not a

valid dictionary immediately return the

max depth so far okay now we need a

recursive case we need to figure out how

to go from the top level dictionary to

the next level nested dictionary or or

dictionaries because there can be

multiple key value pairs so first we'll

set a current Max variable equal to the

max depth so far and the purpose of this

variable will be to keep track of the

max depth of all the nested dictionaries

at this level next we need to iterate

over each potential dictionary at this

level in Python there's a DOT values

method that we can call on a dictionary

object to get every value in that level

of the dictionary now at each key value

pair we'll recursively call dict depth

but this time we'll only pass in that

nested dictionary so instead of passing

in D we're going to pass in V we also

need to increment the max depth so far

by one because we're going one level

deeper now once that recursive call

returns it will return with the maximum

depth of that sub dictionary so we

should track that max depth and at the

end of the loop if that max depth is the

greatest of all the nested sub diaries

that will be the one will return so with

the first call to Dick depth we're going

to pass in a dictionary d as you can see

it has three nested levels the first key

melee weapons is the only key in the top

level it corresponds to a dictionary

which has two entries stabies which

corresponds to another dictionary and

Bose which corresponds to the number six

within the stabies dictionary the third

level we've got two keys Spears and

knives each corresponding to a single

number we'll also pass in a max depth so

far of one because by default a

dictionary just has one level so what

happens well first in this initial call

to di depth the base case is going to be

skipped because we passed it a valid

dictionary next the current Max we'll

set to the maximum depth so far which is

one and that will be overridden within

our Loop if we find a recursive call

that returns a greater number now we

iterate over all the key value pairs in

the dictionary which in this case is

just the melee weapons key we enter the

recursive call where we pass in the

dictionary that corresponds with the

melee weapons key that subdictionary we

also increment the max depth so far to

two within that recursive call again D

is a valid dictionary so we'll set the

current Max to two which is what the max

depth so far was then we'll iterate over

each key value pair again this time

there are two the stabies key and the

Bose key because the stabies key is

first we'll start there we'll kick off

another recursive call this time passing

in the dictionary associated with the

stabies key we'll also increment the max

depth so far to three we'll iterate over

each key value pair again there are two

Spears four and knives three starting

with the spears key we move into another

recursive call this time the value four

is not a valid dictionary so we trigger

the base case and immediately return

four which was the max depth so far that

we passed in after incrementing it now

we're back up a level after returning

four we move on to the next key in the

dictionary the knives key again it kicks

off a recursive call the max depth so

far again is four because again we've

incremented it from its parent function

which was three again D is not a valid

dictionary so we'll immediately return

four now back up again in that parent

function we've iterated over both key

value Pairs and we're going to return

the maximum of the two return values

vales which because they were both four

is just four so that will return up the

call stack again now we're back into the

second call there's another key for us

to process the Bose key that will kick

off a recursive call this time D again

is not a valid dictionary it's just the

value six so we'll return three because

three is the max depth so far after

incrementing from its parent function

which was just two now we've iterated

over each key in the dictionary the

stabies key and the Bose key the

recursive call to the stabies dictionary

returned a four while the recursive call

to the Bose key returned a three so this

call will return the maximum of the two

which is four now we're back in the

original

function we've iterated over all the

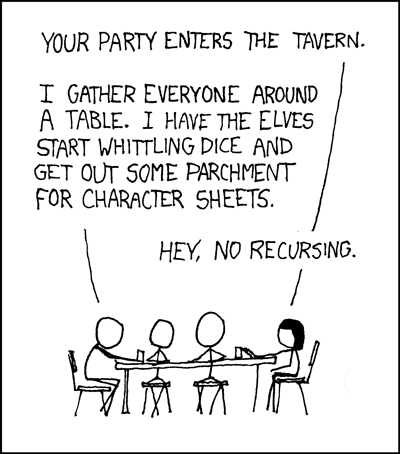
keys which in this case was just the

melee weapons key and its Associated

dictionary and we'll return four and the

recursion is done

# Recursion Review



-- [xkcd](https://xkcd.com/244/)

I hate explaining jokes, but in case you don't get the comic: The joke is that the characters within the Dungeons and Dragons game are also playing their own Dungeons and Dragons game. Maybe their character's game of DnD also has characters playing DnD, and so on, recursively forever.

## Another example

def print\_chars(word, i):

if i == len(word):

return

print(word[i])

print\_chars(word, i + 1)

print\_chars("Hello", 0)

# H

# e

# l

# l

# o

What happens if a recursive function has no base case?

It's no longer considered recursive

It will keep calling itself forever, resulting in an infinite loop

Alan Turing comes back from the dead to haunt the author

Become a member for quiz access

## Another example

def print\_chars(word, i):

if i == len(word):

return

print(word[i])

print\_chars(word, i + 1)

print\_chars("Hello", 0)

# H

# e

# l

# l

#

# Zipmap

Let's practice another simple recursive function.

You may not understand recursion just yet, but by following the instructions, you will begin to grasp the fundamentals.

## Assignment

Within Doc2Doc we need to map certain properties from one document to properties of another document. Complete the recursive zipmap function.

It takes two lists as input and returns a [dictionary](https://docs.python.org/3/tutorial/datastructures.html#dictionaries) where the first list provides the keys and the second list provides the values.

Example usage:

zipped = zipmap(

["Avatar: The Last Airbender", "Avatar (in Papyrus font)", "The Last Airbender (Live Action)"],

[9.9, 6.1, 2.1]

)

print(zipped)

# {

# 'Avatar: The Last Airbender': 9.9,

# 'Avatar (in Papyrus font)': 6.1,

# 'The Last Airbender (Live Action)': 2.1,

# }

Here's the [pseudocode](https://en.wikipedia.org/wiki/Pseudocode):

1. If either the keys or values list is empty, return an empty dictionary (base case)
2. Recursively call zipmap on all but the first elements from keys and values
3. Add the first element of keys to the resulting dictionary, and set its value to the first element in values
4. Return the updated dictionary

# Recursion Quiz

Consider the following function:

def countdown(n):

print(n)

if n == 0:

return

else:

countdown(n - 1)

# Which line contains the base case? Recursion Quiz

Consider the following function:

def countdown(n):

print(n)

if n == 0:

return

else:

countdown(n - 1)

# Nested Sum

Recursion is hard for all new developers. If you're struggling, that's okay! Take your time. That's why we're doing a few extra practice problems.

## Assignment

In Doc2Doc, users can process files or entire directories. We need to know the total size of those files and directories (measured in bytes).

Due to the nested nature of directories, we represent a root directory as a list of lists. Each list represents a directory, and each number represents the size of a file in that directory. For example, here's a directory that contains 2 files at the root level, then a nested directory with its own two files:

root = [

1,

2,

[3, 4]

]

print(sum\_nested\_list(root))

# 10

Here's a more complex example:

root

├── scripts.txt (5 bytes)

├── characters (dir)

│ ├── zuko.txt (6 bytes)

│ └── aang.txt (7 bytes)

└── seasons (dir)

├── season1 (dir)

│ ├── the\_avatar\_returns.docx (8 bytes)

│ └── the\_southern\_air\_temple.docx (9 bytes)

└── season2\_notes.txt (10 bytes)

Which would be represented as:

root = [

5,

[6, 7],

[[8, 9], 10]

]

print(sum\_nested\_list(root))

# 45

**Complete the sum\_nested\_list function**. It takes a nested list of integers as input and should return the total size of all files in the list. It's a recursive function.

Here's some pseudocode to help you get started:

1. Create an integer variable to keep track of the total size.
2. For each item in the list (use a loop here):
   1. If the item is an integer, add it to the total size.
   2. If the item is a list, use a recursive call to sum\_nested\_list to get the size of that list. Add that size to the total size.
3. Return the total size when you're done iterating.

## Tips

You can use loops with recursion. While functional programming avoids loops, recursion can be used outside functional programming.

You can use the built-in [isinstance](https://docs.python.org/3/library/functions.html#isinstance) function to check if an item is an integer or a list:

isinstance(5, list)

# False

isinstance([5, 6], list)

# True

# Recursion Review

Recursion is so dang useful with tree-like structures because we don't always know how deep they're nested. Stop and think about how you would write nested loops to traverse a tree of arbitrary depth... it's not easy, is it?

for item in tree:

for nested\_item in item:

for nested\_nested\_item in nested\_item:

for nested\_nested\_nested\_item in nested\_nested\_item:

# ... WHEN DOES IT END???

I most often use recursion on tree-like problems (file systems, nested dictionaries, etc). If I'm just iterating over a one-dimensional list then a loop (gasp...) is typically simpler, even if it's not as "pure" in the academic sense.

Remember: The rules of functional programming are just philosophies to help you write better code, but it's not always the right tool for the job. The same goes for any programming paradigm.

# Recursion on a tree

Recursion is often used in "tree-like" structures. For example:

* Nested dictionaries
* File systems
* HTML documents
* JSON objects

That's because trees can have unknown depth. It's hard to write a series of loops because you don't know how many levels deep the tree goes.

for entry\_i in directory:

if entry\_i.is\_dir:

for entry\_j in entry\_i:

if entry\_j.is\_dir

for entry\_k in entry\_j:

...

## Assignment

You're responsible for a module in Doc2Doc that can scan a file system (represented in our code as nested dictionaries) and create a list of the filenames.

Complete the recursive list\_files function. It accepts two arguments:

* current\_filetree (dict): A dictionary representing the file system
* current\_path (str): A string representing the current path (e.g. root/dir1/dir2)

It should return a list of file paths (strings). For example, this filetree:

{

"Documents": {

"Proposal.docx": None,

"Receipts": {

"January": {

"receipt1.txt": None,

"receipt2.txt": None

},

"February": {

"receipt3.txt": None

}

}

},

}

Should result in the following list of file paths:

[

"/Documents/Proposal.docx",

"/Documents/Receipts/January/receipt1.txt",

"/Documents/Receipts/January/receipt2.txt",

"/Documents/Receipts/February/receipt3.txt"

]

## Steps

1. Create an empty list to store the file paths.
2. For each "node" in the current dictionary (just use a loop):
   1. If the node's entry is None (meaning it's a final file, not a directory), append the full path for that node to the list.
   2. Otherwise, [extend](https://docs.python.org/3/tutorial/datastructures.html#more-on-lists) the file list with the results of a recursive list\_files call.
      1. The new current\_filetree for the recursive call is the one nested under the current node.
      2. The new current\_path should be the old one with the new node's name added after a slash. e.g. old\_current\_path/node\_name.
3. Return the list of file paths.

## Tips

What's the base case? It looks a bit different than before, but it's just when a nested node's value is None because in that case we don't have any more directories to explore.

You might be wondering why we used a loop! Loops are nice when we know how many times we need to iterate (the number of keys in the dictionary). Recursion is nice when we don't know how many times we need to iterate (the number of nested dictionaries).

# Dangers of Recursion

Recursion is great because it's simple and elegant, often being the most straightforward way to solve a problem. But there are some things to watch out for:

1. **Stack Overflow**: Each function call requires a bit of memory. So, if you recurse too deeply, you can run out of ["stack" memory](https://en.wikipedia.org/wiki/Stack-based_memory_allocation) which will crash your program. (This is what the famous [website](https://stackoverflow.com/) is named after)
2. If you don't have a solid base case, you can end up in an infinite loop (which will likely lead to a stack overflow).
3. Recursion (especially in a language like Python) is often slower than a for loop because each function call requires some memory. [Tail call optimization](https://exploringjs.com/es6/ch_tail-calls.html) can help with this, but Python doesn't support it.

# Count Nested Levels

In Doc2Doc, we might have documents nested inside other documents, forming a kind of tree. You know how crazy .docx files can get...

Anyways, we want to find out how deeply nested a given document is.

## Assignment

Complete the count\_nested\_levels function.

It should recursively search for the target\_document\_id in the nested\_documents dictionary and return the number of nested levels of that document.

If the target document doesn't exist, the function should return -1. You may use loops with recursion.

## Example

In this dictionary, the document with id 3 is nested 2 levels deep. Document 2 is nested 1 level deep.

{

1: {

3: {}

},

2: {}

}

from main import \*

run\_cases = [

({1: {2: {3: {}, 4: {5: {}}}, 6: {}, 7: {8: {9: {10: {}}}}}}, 2, 2),

({1: {2: {3: {}, 4: {5: {}}}, 6: {}, 7: {8: {9: {10: {}}}}}}, 9, 4),

]

submit\_cases = run\_cases + [

({}, 1, -1),

({1: {2: {3: {}, 4: {5: {}}}, 6: {}, 7: {8: {9: {10: {}}}}}}, 5, 4),

({1: {2: {3: {}, 4: {5: {}}}, 6: {}, 7: {8: {9: {10: {}}}}}}, 20, -1),

]

def test(input1, input2, expected\_output):

print("---------------------------------")

print(f"Input tree: {input1}")

print(f"Input document id: {input2}")

print(f"Expecting: {expected\_output}")

result = count\_nested\_levels(input1, input2)

print(f"Actual: {result}")

if result == expected\_output:

print("Pass")

return True

print("Fail")

return False

def main():

passed = 0

failed = 0

for test\_case in test\_cases:

correct = test(\*test\_case)

if correct:

passed += 1

else:

failed += 1

if failed == 0:

print("============= PASS ==============")

else:

print("============= FAIL ==============")

print(f"{passed} passed, {failed} failed")

test\_cases = submit\_cases

if "\_\_RUN\_\_" in globals():

test\_cases = run\_cases

main()

# Recursive String Reversal

At Doc2Doc we need to be able to reverse strings. Our bad-word detection algorithms need to work on those crazy teenagers who write unpleasantries backward to try to get around our filters!

However, being an ex-[Haskell](https://www.haskell.org/)-loving university professor, our team lead has asked us to implement it using recursion instead of a loop.

## Assignment

Complete the reverse\_string function.

It should take a string as a parameter and return the reversed string by recursively reversing the substrings inside. Your function should recurse once for each character in the string.

from main import \*

run\_cases = [

("Functional programming", "gnimmargorp lanoitcnuF"),

("Python", "nohtyP"),

]

submit\_cases = run\_cases + [

("", ""),

(

"Haskell code has no side effects because no one writes it",

"ti setirw eno on esuaceb stceffe edis on sah edoc lleksaH",

),

(

"Lisp is the #1 programming language if you measure by parentheses",

"sesehtnerap yb erusaem uoy fi egaugnal gnimmargorp 1# eht si psiL",

),

("OCaml is for Haskell dropouts", "stuopord lleksaH rof si lmaCO"),

]

def test(input, expected\_output):

print("---------------------------------")

print(f"Input: {input}")

print(f"Expected Output: {expected\_output}")

result = reverse\_string(input)

print(f"Actual Output: {result}")

if result == expected\_output:

print("Pass")

return True

print("Failure")

return False

def main():

passed = 0

failed = 0

for test\_case in test\_cases:

correct = test(\*test\_case)

if correct:

passed += 1

else:

failed += 1

if failed == 0:

print("============= PASS ==============")

else:

print("============= FAIL ==============")

print(f"{passed} passed, {failed} failed")

test\_cases = submit\_cases

if "\_\_RUN\_\_" in globals():

test\_cases = run\_cases

main()

# Longest Word

In Doc2Doc, we have a search function to find the longest word in a document.

## Assignment

Complete the find\_longest\_word function without a loop. It accepts a string input, document, and a string input, longest\_word, that is the current longest word and which defaults to an empty string.

* Check if the first word is longer than the current longest\_word, then recur for the rest of the document.
* Ensure there are no potential [index errors](https://docs.python.org/3/library/exceptions.html#IndexError).

from main import \*

run\_cases = [

("Either that wallpaper goes, or I do.", "wallpaper"),

(

"Then I die happy",

"happy",

),

(

"Et tu, Brute?",

"Brute?",

),

]

submit\_cases = run\_cases + [

(

"",

"",

),

(

" ",

"",

),

(

"Let us cross over the river and rest under the shade of the trees",

"cross",

),

]

def test(input1, expected\_output):

print("---------------------------------")

print(f"Input: '{input1}'")

print(f"Expecting: '{expected\_output}'")

result = find\_longest\_word(input1)

print(f"Actual: '{result}'")

if result == expected\_output:

print("Pass")

return True

print("Fail")

return False

def main():

passed = 0

failed = 0

for test\_case in test\_cases:

correct = test(\*test\_case)

if correct:

passed += 1

else:

failed += 1

if failed == 0:

print("============= PASS ==============")

else:

print("============= FAIL ==============")

print(f"{passed} passed, {failed} failed")

test\_cases = submit\_cases

if "\_\_RUN\_\_" in globals():

test\_cases = run\_cases

main()

# Function Transformations

"Function transformation" is just a more concise way to describe a specific type of [higher order function](https://en.wikipedia.org/wiki/Higher-order_function). It's when a function takes a function (or functions) as input and returns a new function. Let's look at an example:

def multiply(x, y):

return x \* y

def add(x, y):

return x + y

# self\_math is a higher order function

# input: a function that takes two arguments and returns a value

# output: a new function that takes one argument and returns a value

def self\_math(math\_func):

def inner\_func(x):

return math\_func(x, x)

return inner\_func

square\_func = self\_math(multiply)

double\_func = self\_math(add)

print(square\_func(5))

# prints 25

print(double\_func(5))

# prints 10

The self\_math function takes a function that operates on two different parameters (e.g. multiply or add) and returns a new function that operates on one parameter twice (e.g. square or double).

## Assignment

Doc2Doc needs a good logging system so that users and developers alike can see what's going on under the hood. **Complete the get\_logger function.**

It takes a formatter function as a parameter and returns a new function. Steps:

1. Define a new function, logger, inside get\_logger (see self\_math above as an example). It accepts two strings. You can just name them first and second if you like.
2. The logger function should not return anything. It should simply print the result of calling the given formatter function with the first and second strings as arguments.
3. Return the new logger function for the test suite to use.

## Tip

The colon\_delimit and dash\_delimit functions are "formatters" that will be passed into our get\_logger function by the tests. You don't need to touch them, but it's important to understand that when you call formatter() in the get\_logger function, you're calling one of these functions.

def get\_logger(formatter):

# ?

# Don't edit below this line

def test(first, errors, formatter):

print("Logs:")

logger = get\_logger(formatter)

for err in errors:

logger(first, err)

print("====================================")

def colon\_delimit(first, second):

return f"{first}: {second}"

def dash\_delimit(first, second):

return f"{first} - {second}"

def main():

db\_errors = [

"out of memory",

"cpu is pegged",

"networking issue",

"invalid syntax",

]

test("Doc2Doc FATAL", db\_errors, colon\_delimit)

mail\_errors = [

"email too large",

"non alphanumeric symbols found",

]

test("Doc2Doc WARNING", mail\_errors, dash\_delimit)

main()

# Transformations Review

Example of a function transformation:

def multiply(x, y):

return x \* y

def add(x, y):

return x + y

def self\_math(math\_func):

# inner\_func is defined inside self\_math.

# It can only be referenced directly

# inside self\_math's scope. However, it is then

# returned and can be captured into a new variable

# like square\_func or double\_func, and called that way

def inner\_func(x):

return math\_func(x, x)

return inner\_func

square\_func = self\_math(multiply)

double\_func = self\_math(add)

print(square\_func(5))

# 25

print(double\_func(5))

# 10

# More Transformations

Here's some example code for you to reference as you work through the assignment:

def multiply(x, y):

return x \* y

def add(x, y):

return x + y

def self\_math(math\_func):

def inner\_func(x):

return math\_func(x, x)

return inner\_func

square\_func = self\_math(multiply)

double\_func = self\_math(add)

print(square\_func(5))

# prints 25

print(double\_func(5))

# prints 10

## Assignment

Complete the doc\_format\_checker\_and\_converter function.

It takes a conversion\_function and a list of valid\_formats as parameters. It should return a new function that takes two parameters of its own:

* filename: The name of the file to be converted
* content: The content (body text) of the file to be converted

If the file extension of the filename is in the valid\_formats list, then it should return the result of calling the conversion\_function on the content. Otherwise, it should [raise](https://docs.python.org/3/tutorial/errors.html#raising-exceptions) a [ValueError](https://docs.python.org/3/library/exceptions.html#ValueError) with the message Invalid file format.

## Tip

I used the [.split()](https://docs.python.org/3/library/stdtypes.html#str.split) method on the filename to get the file extension. You can use the in keyword to check if a value is in a list.

The capitalize\_content and reverse\_content are "conversion functions" that will be passed into our doc\_format\_checker\_and\_converter function by the tests.

def doc\_format\_checker\_and\_converter(conversion\_function, valid\_formats):

pass

# Don't edit below this line

def capitalize\_content(content):

return content.upper()

def reverse\_content(content):

return content[::-1]

# Why Transform?

You might be wondering:

* "When would I use function transformations in the real world?"
* "Isn't it simpler to just define functions at the top level of the code, and call them as needed?"

Good questions. To be clear, we don't just transform functions at [runtime](https://en.wikipedia.org/wiki/Execution_(computing)#Runtime) for the fun of it! We only use advanced techniques like function transformations when they make our code simpler than it would otherwise be.

## Code Reusability

Creating variations of the same function dynamically can make it a lot easier to share common functionality. Take a look at this formatter function. It accepts a "pattern" and returns a new function that formats text according to that pattern:

def formatter(pattern):

def inner\_func(text):

result = ""

i = 0

while i < len(pattern):

if pattern[i:i+2] == '{}':

result += text

i += 2

else:

result += pattern[i]

i += 1

return result

return inner\_func

Now we can create new formatters easily:

bold\_formatter = formatter("\*\*{}\*\*")

italic\_formatter = formatter("\*{}\*")

bullet\_point\_formatter = formatter("\* {}")

And use them like this:

print(bold\_formatter("Hello"))

# \*\*Hello\*\*

print(italic\_formatter("Hello"))

# \*Hello\*

print(bullet\_point\_formatter("Hello"))

# \* Hello

## Closures

90% of the time, when I use function transformations, it's because I want to create a closure. We'll talk about closures in the next chapter!

# Why Transform?

You might be wondering:

* "When would I use function transformations in the real world?"
* "Isn't it simpler to just define functions at the top level of the code, and call them as needed?"

Good questions. To be clear, we don't just transform functions at [runtime](https://en.wikipedia.org/wiki/Execution_(computing)#Runtime) for the fun of it! We only use advanced techniques like function transformations when they make our code simpler than it would otherwise be.

## Code Reusability

Creating variations of the same function dynamically can make it a lot easier to share common functionality. Take a look at this formatter function. It accepts a "pattern" and returns a new function that formats text according to that pattern:

def formatter(pattern):

def inner\_func(text):

result = ""

i = 0

while i < len(pattern):

if pattern[i:i+2] == '{}':

result += text

i += 2

else:

result += pattern[i]

i += 1

return result

return inner\_func

Now we can create new formatters easily:

bold\_formatter = formatter("\*\*{}\*\*")

italic\_formatter = formatter("\*{}\*")

bullet\_point\_formatter = formatter("\* {}")

And use them like this:

print(bold\_formatter("Hello"))

# \*\*Hello\*\*

print(italic\_formatter("Hello"))

# \*Hello\*

print(bullet\_point\_formatter("Hello"))

# \* Hello

## Closures

90% of the time, when I use function transformations, it's because I want to create a closure. We'll talk about closures in the next chapter!

# Filter Command

In Doc2Doc, users are asking for a filtering feature. They want a command that has dynamic options so they can work as quickly as possible.

## Assignment

Complete the get\_filter\_cmd function. It takes two functions as input, filter\_one and filter\_two, and returns a function, filter\_cmd.

filter\_cmd should take as input a string content to be filtered, and a string option with a default value of --one. The filter\_cmd should filter and return the content according to the input option. Do not use the builtin filter function.

1. If --one, use filter\_one
2. If --two, use filter\_two
3. If --three, use filter\_one first, then use filter\_two
4. If another option is passed, raise an exception, "invalid option"

def get\_filter\_cmd(filter\_one, filter\_two):

def filter\_cmd(content, option):

pass

return filter\_cmd

# don't touch below this line

def replace\_bad(text):

return text.replace("bad", "good")

def replace\_ellipsis(text):

return text.replace("..", "...")

def fix\_ellipsis(text):

return text.replace("....", "...")

# Upgrade Filter Command

Users like the filter command, but think it could be better. They want access to more than two filter options and to customize the words that get filtered. It's time to upgrade the filter command feature.

## Assignment

1. Complete the get\_filter\_cmd function. It should take a dictionary as input, filters, and return a function, filter\_cmd.

* filters contains option string and filter function key/value pairs.

1. filter\_cmd should take as input a string content to be filtered, a list of strings options, and a list of tuples word\_pairs.

* The filter\_cmd should filter and return the content, filtered according to the input options
* If there are no options in the options list, raise an exception "missing options".
* For each option, if its option string is in the filters dictionary, then filter the content by passing the content and word\_pairs to the option's filter.
* If an option is not in the filters dictionary, raise an exception "invalid option".

# Closures

A [closure](https://en.wikipedia.org/wiki/Closure_(computer_programming)) is a function that references variables from outside its own function body. The function definition and its environment are bundled together into a single entity.

Put simply, a closure is just a function that **keeps track of some values** from the place where it was defined, no matter where it is executed later on.

## Example

The concatter() function returns a function called doc\_builder (yay higher-order functions!) that has a reference to an enclosed doc value.

def concatter():

doc = ""

def doc\_builder(word):

# "nonlocal" tells Python to use the 'doc'

# variable from the enclosing scope

nonlocal doc

doc += word + " "

return doc

return doc\_builder

# save the returned 'doc\_builder' function

# to the new function 'harry\_potter\_aggregator'

harry\_potter\_aggregator = concatter()

harry\_potter\_aggregator("Mr.")

harry\_potter\_aggregator("and")

harry\_potter\_aggregator("Mrs.")

harry\_potter\_aggregator("Dursley")

harry\_potter\_aggregator("of")

harry\_potter\_aggregator("number")

harry\_potter\_aggregator("four,")

harry\_potter\_aggregator("Privet")

print(harry\_potter\_aggregator("Drive"))

# Mr. and Mrs. Dursley of number four, Privet Drive

When concatter() is called, it creates a new "stateful" function that remembers the value of its internal doc variable. Each successive call to harry\_potter\_aggregator appends to that same doc.

## nonlocal

Python has a keyword called [nonlocal](https://docs.python.org/3/reference/simple_stmts.html#nonlocal) that's required to access variables from an enclosing scope. Most programming languages don't require this keyword, but Python does.

## Assignment

Doc2Doc keeps track of how many words are in a collection of documents.

Complete the word\_count\_aggregator function. It should return a function that calculates the number of words in its input (doc, a string). It should then add that number to an enclosed count value and return the new count. In other words, it keeps a running total of the count variable within a closure.

## Tip

I used [.split()](https://docs.python.org/3/library/stdtypes.html#str.split) to count the number of words in the doc string.

# Closure Review

The whole point of a closure is that it's stateful. It's a function that "remembers" the values from the enclosing scope even after the enclosing scope has finished executing.

It's as if you're saving the state of a function at a particular point in time, and then you can use and update that state later on.

def concatter():

doc = ""

def inner\_func(word):

# "nonlocal" tells Python to use the doc

# variable from the enclosing scope

nonlocal doc

doc += word + " "

return doc

return inner\_func

harry\_potter\_aggregator = concatter()

harry\_potter\_aggregator("Mr.")

harry\_potter\_aggregator("and")

harry\_potter\_aggregator("Mrs.")

harry\_potter\_aggregator("Dursley")

harry\_potter\_aggregator("of")

harry\_potter\_aggregator("number")

harry\_potter\_aggregator("four,")

harry\_potter\_aggregator("Privet")

print(harry\_potter\_aggregator("Drive"))

# Mr. and Mrs. Dursley of number four, Privet Drive

That means that in many cases, closures are not pure functions. They can mutate state outside of their scope and have side effects.

# Closure Review

The whole point of a closure is that it's stateful. It's a function that "remembers" the values from the enclosing scope even after the enclosing scope has finished executing.

It's as if you're saving the state of a function at a particular point in time, and then you can use and update that state later on.

def concatter():

doc = ""

def inner\_func(word):

# "nonlocal" tells Python to use the doc

# variable from the enclosing scope

nonlocal doc

doc += word + " "

return doc

return inner\_func

harry\_potter\_aggregator = concatter()

harry\_potter\_aggregator("Mr.")

harry\_potter\_aggregator("and")

harry\_potter\_aggregator("Mrs.")

harry\_potter\_aggregator("Dursley")

harry\_potter\_aggregator("of")

harry\_potter\_aggregator("number")

harry\_potter\_aggregator("four,")

harry\_potter\_aggregator("Privet")

print(harry\_potter\_aggregator("Drive"))

# Mr. and Mrs. Dursley of number four, Privet Drive

That means that in many cases, closures are not pure functions. They can mutate state outside of their scope and have side effects.

# Closure Practice

Remember, a [closure](https://en.wikipedia.org/wiki/Closure_(computer_programming)) is a function that retains the state of its environment. That makes it useful for tracking data as it changes over time, but it can come at the cost of understandability.

When not to use the nonlocal keyword: when the variable is mutable (such as a list, dictionary or set), and you are modifying its contents rather than reassigning the variable. You only need the nonlocal keyword if you are reassigning a variable instead of modifying its contents (which you must do to change immutable values such as strings and integers).

Let's try a closure without the nonlocal keyword.

## Assignment

**Complete the new\_collection function.**

It accepts a list of strings, initial\_docs.

It should return a function that closes over a **copy** of initial\_docs. This function should accept a string, append that string to the closed-over list, and return the new list.

Do not modify the original initial\_docs list!

## Example usage

new\_collection = new\_collection(["doc1", "doc2", "doc3"])

print(new\_collection("doc4"))

# ['doc1', 'doc2', 'doc3', 'doc4']

print(new\_collection("doc5"))

# ['doc1', 'doc2', 'doc3', 'doc4', 'doc5']

# Copy/Paste

Doc2Doc has a simple clipboard for copying to and pasting from a cache. Initialize the clipboard to set and get strings.

## Assignment

Complete the new\_clipboard function. It accepts a dictionary as input and returns two functions, copy\_to\_clipboard and paste\_from\_clipboard. It should not modify the original input dictionary.

* copy\_to\_clipboard: It takes a key and value string pair and adds them to the clipboard dictionary.
* paste\_from\_clipboard: It takes a key string and returns its value from the clipboard dictionary. If the key is missing from the clipboard, return an empty string.

# CSS Styles

Doc2Doc should be able to add css styling to an html file. CSS uses selectors to identify the html element to add the style property. Essentially, styles are a chain of keys and values.

## Assignment

Complete the css\_styles function. It accepts a nested dictionary as input, initial\_styles, and returns a function, add\_style.

1. It should not modify the original input dictionary. Make a copy of the initial\_styles dictionary.
2. Create the add\_style inner function. It takes 3 strings as input, selector, property and value. add\_style should modify and return the copy of the initial\_styles dictionary.
3. The selector is the first key and its value should be a dictionary.
4. property is the key for the selector's dictionary and value is its value.
5. Create the selector's dictionary if it doesn't already exist within the styles dictionary, then add or replace the property and value in that dictionary.

For example:

initial\_styles = {

"body": {

"background-color": "white",

"color": "black"

},

"h1": {

"font-size": "16px",

"padding": "10px"

}

}

add\_style = css\_styles(initial\_styles)

new\_styles = add\_style("p", "color", "grey")

# {

# "body": {

# "background-color": "white",

# "color": "black"

# },

# "h1": {

# "font-size": "16px",

# "padding": "10px"

# },

# "p": {

# "color": "grey",

# }

# }

# Pagination

Users should be able to paginate lengthy documents to make them manageable. Pagination is simply dividing a document into pages. This idea can also be applied to other data besides raw text, such as a list of search results or API responses.

## Assignment

Complete the inner add\_word\_to\_pages function. It takes a list of strings, pages, and a string word, as inputs and returns pages with the word added.

1. If pages is empty, return the word in a list
2. Every page will have at least one word, even if that word is longer than the page\_length
3. The last page in pages is the current page
4. If the length of the current page and the word and a space is more than the page\_length, start a new page
5. Else, add the word to the current page in pages

paginate = paginator(11)

pages = paginate("Boots loves salmon because he is a bear.")

# pages: ["Boots loves", "salmon", "because he", "is a bear."]

# Currying

Function [currying](https://en.wikipedia.org/wiki/Currying) is a specific kind of function transformation where we translate a single function that accepts multiple arguments into multiple functions that each accept a single argument.

This is a "normal" 3-argument function:

box\_volume(3, 4, 5)

This is a "curried" series of functions that does the same thing:

box\_volume(3)(4)(5)

Here's another example that includes the implementations:

def sum(a, b):

return a + b

print(sum(1, 2))

# prints 3

And the same thing curried:

def sum(a):

def inner\_sum(b):

return a + b

return inner\_sum

print(sum(1)(2))

# prints 3

The sum function only takes a single input, a. It returns a new function that takes a single input, b. This new function when called with a value for b will return the sum of a and b. We'll talk later about why this is useful.

## Assignment

In Doc2Doc, depending on the type of text file we're working with, we sometimes need to transform the font size of the text when it comes time to render it on the screen.

Fix the converted\_font\_size function. We are using a 3rd party code library that expects our function to be a curried series of functions that each take a single argument.

* converted\_font\_size should just take a single argument, font\_size and return a function that takes a single argument, doc\_type. That function should return the font\_size multiplied by the appropriate value for the given doc\_type.

## Tip

You can always Reset the code to see the proper font\_size multipliers if you accidentally change them.

# Why Curry?

It's fairly obvious that:

def sum(a, b):

return a + b

is simpler than:

def sum(a):

def inner\_sum(b):

return a + b

return inner\_sum

So why would we ever want to do the more complicated thing? Well, currying is often used to **change a function's signature** to make it conform to a specific shape. For example:

def colorize(converter, doc):

# ...

converter(doc)

# ...

The colorize function accepts a function called converter as input, and at some point during its execution, it calls converter with a single argument. That means that it expects converter to accept exactly one argument. So, if I have a conversion function like this:

def markdown\_to\_html(doc, asterisk\_style):

# ...

I can't pass markdown\_to\_html to colorize because markdown\_to\_html wants two arguments. To solve this problem, I can curry markdown\_to\_html into a function that takes a single argument:

def markdown\_to\_html(asterisk\_style):

def asterisk\_md\_to\_html(doc):

# do stuff with doc and asterisk\_style...

return asterisk\_md\_to\_html

markdown\_to\_html\_italic = markdown\_to\_html('italic')

colorize(markdown\_to\_html\_italic, doc)

# Why Curry?

It's fairly obvious that:

def sum(a, b):

return a + b

is simpler than:

def sum(a):

def inner\_sum(b):

return a + b

return inner\_sum

So why would we ever want to do the more complicated thing? Well, currying is often used to **change a function's signature** to make it conform to a specific shape. For example:

def colorize(converter, doc):

# ...

converter(doc)

# ...

The colorize function accepts a function called converter as input, and at some point during its execution, it calls converter with a single argument. That means that it expects converter to accept exactly one argument. So, if I have a conversion function like this:

def markdown\_to\_html(doc, asterisk\_style):

# ...

I can't pass markdown\_to\_html to colorize because markdown\_to\_html wants two arguments. To solve this problem, I can curry markdown\_to\_html into a function that takes a single argument:

def markdown\_to\_html(asterisk\_style):

def asterisk\_md\_to\_html(doc):

# do stuff with doc and asterisk\_style...

return asterisk\_md\_to\_html

markdown\_to\_html\_italic = markdown\_to\_html('italic')

colorize(markdown\_to\_html\_italic, doc)

# Currying Practice

Remember, [currying](https://en.wikipedia.org/wiki/Currying) is when we take a function that accepts multiple arguments:

final\_volume = box\_volume(3, 4, 5)

print(final\_volume)

# 60

And convert it into a series of functions that each accept a single argument:

final\_volume = box\_volume(3)(4)(5)

print(final\_volume)

# 60

* box\_volume(3) returns a new function that accepts a single integer and returns a new function
* box\_volume(3)(4) returns another new function that accepts a single integer and returns a new function
* box\_volume(3)(4)(5) returns the final result

Here's another way of calling it, where each function is stored in a variable before being called:

with\_length\_3 = box\_volume(3)

with\_len\_3\_width\_4 = with\_length\_3(4)

final\_volume = with\_len\_3\_width\_4(5)

print(final\_volume)

# 60

Here are the function definitions:

def box\_volume(length):

def box\_volume\_with\_len(width):

def box\_volume\_with\_len\_width(height):

return length \* width \* height

return box\_volume\_with\_len\_width

return box\_volume\_with\_len

## Assignment

Doc2Doc needs to be able to find the number of lines in a document that contain a specific sequence of characters. For example, given the following document:

aaaa

bbbb

ccdd

aabb

How many lines contain the sequence "aa"? The answer is 2: "aaaa" and "aabb".

**Complete the lines\_with\_sequence function.** It should return a series of curried functions so it can be called like this:

num\_lines = lines\_with\_sequence(char)(length)(doc)

The "sequence" is generated by the first with\_char that has been provided for you. It works like this:

| **Character** | **Length** | **Sequence** |
| --- | --- | --- |
| "a" | 3 | "aaa" |
| "b" | 2 | "bb" |
| "\*" | 4 | "\*\*\*\*" |

**You need to define and return a second curried function.** I called mine with\_length. It should accept the final parameter, a doc string, and return the number of lines that contain the sequence.

1. Define the with\_length function inside the with\_char function, it should accept a doc.
2. [Split](https://docs.python.org/3/library/stdtypes.html#str.split) the doc into lines.
3. Use a loop (or if you're feeling fancy, use [reduce](https://docs.python.org/3/library/functools.html#functools.reduce)) to count the number of lines that contain the sequence [in](https://docs.python.org/3/reference/expressions.html#membership-test-details) them.
4. Return the count from the with\_length function.
5. Return the with\_length function from the with\_char function.

# HTML Table

Doc2Doc should have tools to create HTML boilerplate. One of the features should create a table. An [HTML table](https://developer.mozilla.org/en-US/docs/Learn/HTML/Tables/Basics) has a header row and data rows. A header row has headers for the columns. Each normal row has data cells which contain the information in the table. It is essentially a 2-dimensional list.

Example HTML Table:

<table>

<tr>

<th>Row 1, Header 1</th>

<th>Row 1, Header 2</th>

</tr>

<tr>

<td>Row 2, Cell 1</td>

<td>Row 2, Cell 2</td>

</tr>

<tr>

<td>Row 3, Cell 1</td>

<td>Row 3, Cell 2</td>

</tr>

</table>

* "td": Each item in a table goes in its own data cell, which are arranged in rows.
* "tr": The table row tag goes around each row of cells.
* "th": The header cells hold the headers for each column and belong in the first row.
* "table": This is the parent tag of the entire table.

Result:

| **Row 1, Header 1** | **Row 1, Header 2** |
| --- | --- |
| Row 2, Cell 1 | Row 2, Cell 2 |
| Row 3, Cell 1 | Row 3, Cell 2 |

## Assignment

Complete the create\_html\_table function. It takes a list of lists of strings, data\_rows, and returns a function, create\_table\_headers. create\_table\_headers should take a list of strings, headers, and convert them into the header row of the table, then return the complete HTML table as a string without any added whitespace or indentation.

Use the provided functions to complete the create\_html\_table function:

1. Use the accumulators and the map function to convert the data\_rows into a single string of html rows.
   * Use map to convert each string in the data\_rows list into data cells within a table row.
   * Accumulate the table rows together as a single string.
2. Within the create\_table\_headers function:
   * Access the nonlocal rows string.
   * Accumulate the strings in the headers list as header cells in a single table row.
   * Add the row of headers to the beginning of the rows string.
   * Add the final tag, "table", around all of the rows.
   * Return one single string containing the HTML table.

# Markdown Image

Markdown makes [displaying images](https://www.markdownguide.org/basic-syntax/#images-1) as simple as possible. To add an image to a markdown document, just use this syntax:

![alt text](url "title")

* alt text a brief description for screen readers and web scrapers. Required for accessibility.
* url url or relative path to image.
* title shown on mouse hover. Optional.

## Assignment

Doc2Doc makes using markdown a breeze. This includes adding images to markdown documents.

1. Complete the create\_markdown\_image function using currying. It takes a string input, alt\_text, and returns an inner function.
   * It should enclose the alt\_text in square brackets prefixed with an exclamation point ![alt\_text].
2. Create the inner function returned by create\_markdown\_image. It also takes a string input, url, and returns an innermost function.
   * The inner function should first escape any parentheses in the URL by replacing them with [encoded sequences](https://www.url-encode-decode.com/).
     + Use the .replace() string method to change any opening parenthesis ( into %28.
     + Do the same to change any closing parenthesis ) into %29.
   * Enclose the url with parentheses (url).
   * Add the enclosed url to the end of the enclosed alt\_text: ![alt\_text](url)
3. Create the innermost function returned by the inner function. It should take an optional string input for the title.
   * If a title is passed:
     + Enclose it in double quotes.
     + Then add the quoted title to the image syntax by first removing the closing parenthesis ) from the end of the image syntax.
     + Add a space and the quoted title with a closing parenthesis ) to the end of the image syntax: ![alt\_text](url "title")
   * Return the finished image syntax.

# Markdown Image

Markdown makes [displaying images](https://www.markdownguide.org/basic-syntax/#images-1) as simple as possible. To add an image to a markdown document, just use this syntax:

![alt text](url "title")

* alt text a brief description for screen readers and web scrapers. Required for accessibility.
* url url or relative path to image.
* title shown on mouse hover. Optional.

## Assignment

Doc2Doc makes using markdown a breeze. This includes adding images to markdown documents.

1. Complete the create\_markdown\_image function using currying. It takes a string input, alt\_text, and returns an inner function.
   * It should enclose the alt\_text in square brackets prefixed with an exclamation point ![alt\_text].
2. Create the inner function returned by create\_markdown\_image. It also takes a string input, url, and returns an innermost function.
   * The inner function should first escape any parentheses in the URL by replacing them with [encoded sequences](https://www.url-encode-decode.com/).
     + Use the .replace() string method to change any opening parenthesis ( into %28.
     + Do the same to change any closing parenthesis ) into %29.
   * Enclose the url with parentheses (url).
   * Add the enclosed url to the end of the enclosed alt\_text: ![alt\_text](url)
3. Create the innermost function returned by the inner function. It should take an optional string input for the title.
   * If a title is passed:
     + Enclose it in double quotes.
     + Then add the quoted title to the image syntax by first removing the closing parenthesis ) from the end of the image syntax.
     + Add a space and the quoted title with a closing parenthesis ) to the end of the image syntax: ![alt\_text](url "title")
   * Return the finished image syntax.

# Resize Image

Doc2Doc should include a feature for image resizing, allowing users to adjust image dimensions to specified ranges. This ensures that images in documents fit and aren't freakishly large or hilariously small.

## Assignment

1. Complete the new\_resizer function using currying. It takes integer inputs max\_width and max\_height and returns an inner function.
2. The inner function should take optional integer inputs min\_width and min\_height — with default values 0 — and return an innermost function.
   * If min\_width is more than max\_width or min\_height is more than max\_height, raise an exception "minimum size cannot exceed maximum size".
3. The innermost function should take two integer inputs width and height and return two integers.
   * Use the built-in [min](https://docs.python.org/3/library/functions.html#min) and [max](https://docs.python.org/3/library/functions.html#max) functions to reduce or increase the width and height as needed, then return them.

## Example

If our new\_resizer function returns a set\_min\_size function, and set\_min\_size returns a resize\_image function, we would use it like this:

# Step 1: Create the resizer with maximum dimensions

set\_min\_size = new\_resizer(800, 600)

# Step 2: Set the minimum dimensions

resize\_image = set\_min\_size(200, 100)

# Step 3: Resize the image

new\_width, new\_height = resize\_image(1000, 500)

# Step 4: Output the result

print(new\_width, new\_height) # Output: 800, 500

# With currying syntax

print(new\_resizer(800, 600)(200, 100)(1000, 500)) # Output: (800, 500)

# Decorators

[Python decorators](https://book.pythontips.com/en/latest/decorators.html) are just [syntactic sugar](https://en.wikipedia.org/wiki/Syntactic_sugar) for [higher-order functions](https://en.wikipedia.org/wiki/Higher-order_function).

**Example:**

def vowel\_counter(func\_to\_decorate):

vowel\_count = 0

def wrapper(doc):

nonlocal vowel\_count

vowels = "aeiou"

for char in doc:

if char in vowels:

vowel\_count += 1

print(f"Vowel count: {vowel\_count}")

return func\_to\_decorate(doc)

return wrapper

@vowel\_counter

def process\_doc(doc):

print(f"Document: {doc}")

process\_doc("What")

# Vowel count: 1

# Document: What

process\_doc("a wonderful")

# Vowel count: 5

# Document: a wonderful

process\_doc("world")

# Vowel count: 6

# Document: world

The @vowel\_counter line is "decorating" the process\_doc function with the vowel\_counter function. vowel\_counter is called once when process\_doc is defined with the @ syntax, but the wrapper function that it returns is called every time process\_doc is called. That's why vowel\_count is preserved and printed after each time.

## It's just syntactic sugar

Python decorators are just another (sometimes simpler) way of writing a higher-order function. These two pieces of code are identical:

### With decorator

@vowel\_counter

def process\_doc(doc):

print(f"Document: {doc}")

process\_doc("Something wicked this way comes")

### Without decorator

def process(doc):

print(f"Document: {doc}")

process\_doc = vowel\_counter(process)

process\_doc("Something wicked this way comes")

## Assignment

The provided file\_type\_aggregator function is intended to decorate other functions. It assumes that the function it decorates has exactly 2 positional arguments.

**Create a process\_doc function** that's decorated by file\_type\_aggregator. It should return the following string:

f"Processing doc: '{doc}'. File Type: {file\_type}"

Where doc and file\_type are its positional arguments. (See line 11 for where it's being called)

**Struggling?** I, Boots the Undercaffeinated and Overfed, am happy to help! Ask me and I'll answer for a price. If you find my explanations suspect, ask a human for help in [the Discord](https://boot.dev/community)!



* main.py
* main\_test.py

1

2

3

4

5

6

7

8

9

10

11

12

13

14

15

16

17

18

19

20

21

def file\_type\_aggregator(func\_to\_decorate):

# dict of file\_type -> count

counts = {}

def wrapper(doc, file\_type):

nonlocal counts

if file\_type not in counts:

counts[file\_type] = 0

counts[file\_type] += 1

result = func\_to\_decorate(doc, file\_type)

return result, counts

return wrapper

# don't touch above this line

# ?

# Args and kwargs

In Python, [\*args and \*\*kwargs](https://book.pythontips.com/en/latest/args_and_kwargs.html) allow a function to accept and deal with a variable number of arguments.

* \*args collects positional arguments into a tuple
* \*\*kwargs collects keyword (named) arguments into a dictionary

def print\_arguments(\*args, \*\*kwargs):

print(f"Positional arguments: {args}")

print(f"Keyword arguments: {kwargs}")

print\_arguments("hello", "world", a=1, b=2)

# Positional arguments: ('hello', 'world')

# Keyword arguments: {'a': 1, 'b': 2}

## Positional arguments

Positional arguments are the ones you're already familiar with, where the order of the arguments matters. Like this:

def sub(a, b):

return a - b

# a=3, b=2

res = sub(3, 2)

# res = 1

## Keyword arguments

[Keyword arguments](https://docs.python.org/3/tutorial/controlflow.html#keyword-arguments) are passed in by name. Order does not matter. Like this:

def sub(a, b):

return a - b

res = sub(b=3, a=2)

# res = -1

res = sub(a=3, b=2)

# res = 1

## A note on ordering

Any positional arguments must come before keyword arguments. This will not work:

sub(b=3, 2)

## Assignment

At Doc2Doc, we need better internal debugging tools. **Complete the args\_logger function.** It takes a variable number of positional and keyword arguments and prints them to the console.

1. Print each positional argument sequentially using numbers and periods as the prefixes. For example:

args\_logger("what's", "up", "doc") # prints:

# 1. what's

# 2. up

# 3. doc

1. Print each keyword argument alphabetically by key using asterisks (\*) as the prefix with a colon (:) inbetween. For example:

args\_logger("hi", "there", age=17, date="July 4 1776") # prints:

# 1. hi

# 2. there

# \* age: 17

# \* date: July 4 1776

Use the [*sorted()*](https://docs.python.org/3/library/functions.html#sorted) function to get the order right.

## Tips

* Don't feel guilty about using loops
* kwargs is a dictionary, not a list. My recommendation is to use the [.items()](https://docs.python.org/3/library/stdtypes.html#dict.items) method to get its key-value pairs as a list of tuples, then sort that list before printing

# Decorators

The \*args and \*\*kwargs syntax is great for decorators that are intended to work on functions with different [signatures](https://developer.mozilla.org/en-US/docs/Glossary/Signature/Function).

## Example

The log\_call\_count function below doesn't care about the number or type of the decorated function's (func\_to\_decorate) arguments. It just wants to count how many times the function is called. However, it still needs to pass any arguments through to the wrapped function.

def log\_call\_count(func\_to\_decorate):

count = 0

def wrapper(\*args, \*\*kwargs):

nonlocal count

count += 1

print(f"Called {count} times")

# The \* and \*\* syntax unpacks the arguments

# and passes them to the decorated function

return func\_to\_decorate(\*args, \*\*kwargs)

return wrapper

## Assignment

**Complete the markdown\_to\_text\_decorator function.** It can decorate a function with any number of string arguments, no matter if they're positional or keyword args. It will run the decorated function, but first strip out any Markdown heading symbols (see below for an explanation of Markdown headings).

It should return a wrapper function that takes \*args and \*\*kwargs. The wrapper should:

1. Map the \*args to a new [list](https://docs.python.org/3/library/stdtypes.html#list) where each string is converted to plain text using convert\_md\_to\_txt.
2. Map the \*\*kwargs to a new [dictionary](https://docs.python.org/3/library/stdtypes.html#dict) where each "value" is converted to plain text using convert\_md\_to\_txt. The "key" should remain the same.
3. Return the result of calling the decorated function with the new arguments.

## Tips

* Take a look at the editor's formatters.py file tab to see what the formatter functions do. What arguments are they expecting? You can use \* tuple unpacking and \*\* dictionary unpacking operators to pass variables as the correct arguments.
* The [map](https://docs.python.org/3/library/functions.html#map) function
* Use the [list()](https://docs.python.org/3/library/stdtypes.html#list) function to convert map results to a list
* Use the [dict()](https://docs.python.org/3/library/stdtypes.html#dict) function to convert map results to a dictionary
* The [.items()](https://docs.python.org/3/library/stdtypes.html#dict.items) method can be used on a dictionary to get an iterable of key-value [tuple](https://docs.python.org/3/library/stdtypes.html#tuple) pairs
* The provided convert\_md\_to\_txt function takes a string of [Markdown](https://www.markdownguide.org/cheat-sheet/) text and returns a string of text with any "heading" symbols removed. For example:

| **Input** | **Output** |
| --- | --- |
| # This is a heading | This is a heading |
| ## This is also a heading | This is also a heading |
| This is not a heading | This is not a heading |
| \* This is also not a heading | \* This is also not a heading |

Decorators.py

def markdown\_to\_text\_decorator(func):

def wrapper(\*args, \*\*kwargs):

pass

return wrapper

# don't touch below this line

def convert\_md\_to\_txt(doc):

lines = doc.split("\n")

for i in range(len(lines)):

line = lines[i]

lines[i] = line.lstrip("# ")

return "\n".join(lines)

formatters.py

from decorators import \*

@markdown\_to\_text\_decorator

def concat(first\_doc, second\_doc):

return f""" First: {first\_doc}

Second: {second\_doc}"""

@markdown\_to\_text\_decorator

def format\_as\_essay(title, body, conclusion):

return f""" Title: {title}

Body: {body}

Conclusion: {conclusion}"""

# Decorators Review

A Python decorator is just [syntactic sugar](https://en.wikipedia.org/wiki/Syntactic_sugar) for [higher-order functions](https://en.wikipedia.org/wiki/Higher-order_function). "Syntactic sugar" just means "a more convenient syntax".

Not all programming languages have built-in decorators, but most do support higher-order functions and closures.

Some of the famous [functional languages](https://en.wikipedia.org/wiki/Purely_functional_programming) like:

* [Haskell](https://en.wikipedia.org/wiki/Haskell_(programming_language))
* [Erlang](https://en.wikipedia.org/wiki/Erlang_(programming_language))
* [Clojure](https://en.wikipedia.org/wiki/Clojure)
* [Lisp](https://en.wikipedia.org/wiki/Lisp_(programming_language))

do not have special syntax decorators, but they do have higher-order functions and closures, meaning the [decorator pattern](https://en.wikipedia.org/wiki/Decorator_pattern) can still be used. So, if you understand those concepts, they will serve you well in many different languages.

# Decorators Review

A Python decorator is just [syntactic sugar](https://en.wikipedia.org/wiki/Syntactic_sugar) for [higher-order functions](https://en.wikipedia.org/wiki/Higher-order_function). "Syntactic sugar" just means "a more convenient syntax".

Not all programming languages have built-in decorators, but most do support higher-order functions and closures.

Some of the famous [functional languages](https://en.wikipedia.org/wiki/Purely_functional_programming) like:

* [Haskell](https://en.wikipedia.org/wiki/Haskell_(programming_language))
* [Erlang](https://en.wikipedia.org/wiki/Erlang_(programming_language))
* [Clojure](https://en.wikipedia.org/wiki/Clojure)
* [Lisp](https://en.wikipedia.org/wiki/Lisp_(programming_language))

do not have special syntax decorators, but they do have higher-order functions and closures, meaning the [decorator pattern](https://en.wikipedia.org/wiki/Decorator_pattern) can still be used. So, if you understand those concepts, they will serve you well in many different languages.

# Configure Plugin

Doc2Doc should be extensible to allow for third-party plugins. These plugins will be configurable.

## Assignment

Complete the configure\_plugin\_decorator function. It decorates a func that takes keyword arguments \*\*kwargs, but the wrapper function it returns takes positional arguments \*args. The arguments passed to the wrapper will be a series of tuples, each a key/value pair.

1. Convert the args into a dictionary with the [dict](https://docs.python.org/3/library/stdtypes.html#dict) function.
2. Get the result of passing this dictionary into the func as keyword arguments unpacked with the [\*\*](https://www.bitecode.dev/i/140708565/unpacking-arguments) operator.
3. Return the result.

plugin\_config = configure\_backups(("path", "~/duplicates"), ("prefix", "duplicate\_"), ("extension", ".rtf"))

# plugin\_config:

# {

# "path": "~/duplicates",

# "prefix": "duplicate\_",

# "extension": ".rtf",

# }

Plugins.py

from decorators import \*

@configure\_plugin\_decorator

def configure\_backups(path="~/backups", prefix="copy\_", extension=".txt"):

return {

"path": path,

"prefix": prefix,

"extension": extension,

}

@configure\_plugin\_decorator

def configure\_login(user=None, password=None, token=None):

return {

"user": user,

"password": password,

"token": token,

}

# Escape HTML

You can stack decorators, and you can use currying with decorators.

def to\_uppercase(func):

def wrapper(document):

return func(document.upper())

return wrapper

def get\_truncate(length):

def truncate(func):

def wrapper(document):

return func(document[:length])

return wrapper

return truncate

@to\_uppercase

@get\_truncate(9) # currying

def print\_input(input):

print(input)

print\_input("Keep Calm and Carry On")

# prints: "KEEP CALM"

Observe that to\_uppercase wrapped get\_truncate(9), and get\_truncate(9) returned truncate which wrapped print\_input, then print\_input printed "KEEP CALM" from "Keep Calm and Carry On".

You might not know anything about HTML. That's fine. This challenge isn't about HTML directly. Just understand that it's a markup language like markdown. Certain characters are interpreted as part of HTML tags. In order to show these characters without interpreting them, they must be encoded as [escape sequences](https://www.w3.org/MarkUp/html-spec/html-spec_9.html#SEC9.7.1). tldr: replace "<" with "&lt;".

Doc2Doc needs a feature that can take care of encoding such characters for you. This is particularly helpful if you want to show raw HTML on a vanilla HTML webpage. HTML even has a semantic <pre> tag for designating pre-formatted text.

## Assignment

Complete the replacer function.

1. It takes as input two strings, old and new, and returns a function, replace.
2. replace takes an input function, decorated\_func, and returns a wrapper function.
3. wrapper takes as input a string text. It uses .replace() string method to replace instances of old with new in the text. Then it returns the result of passing the modified text to the decorated\_func.
4. Use a series of the replacer function to decorate tag\_pre. Pass the following pairs of strings to these decorators to encode the escape sequences:

* Replace "&" with "&amp;"
* Replace "<" with "&lt;"
* Replace ">" with "&gt;"
* Replace '"' with "&quot;"
* Replace "'" with "&#x27;"

# lru\_cache

[lru\_cache from the functools module](https://docs.python.org/3/library/functools.html#functools.lru_cache) is an example of a decorator and an example of memoization.

lru\_cache memoizes the inputs and outputs of the decorated function in a size-restricted dictionary. It speeds up repeated calls to a slow function with the same inputs. For instance, if the function reads from disc, makes network requests, or requires a lot of computation AND it is used repeatedly with the same inputs.

Here's an example from the Python documentation that perfectly illustrates how and why to use the lru\_cache decorator:

from functools import lru\_cache

@lru\_cache()

def factorial\_r(x):

if x == 0:

return 1

else:

return x \* factorial\_r(x - 1)

factorial\_r(10) # no previously cached result, makes 11 recursive calls

# 3628800

factorial\_r(5) # just looks up cached value result

# 120

factorial\_r(12) # makes two new recursive calls, the other 11 are cached

# 479001600

Since the factorial function is recursive and the inputs are sequential numbers, it's called repeatedly with the same inputs. Without the cache, the function would be called 30 times. With lru\_cache, the function is only called 13 times. While you don't often need to compute factorials, this example ties together how to use a decorator and memoization and recursion.

## Assignment

The creator of Doc2Doc is a huge fan of [palindromes](https://en.wikipedia.org/wiki/Palindrome) for some nerdy reason. Add a feature to check if a word is a palindrome.

1. Import the lru\_cache function from the functools module. Use it to decorate the incomplete is\_palindrome function.
2. Complete the is\_palindrome function. It takes as input a word string and returns True if the word is a palindrome (such as "racecar"), or False otherwise. Try to use recursion. Check the outer characters first, then move inwards until you reach the base case or find the word is not a palindrome.

# Sum Types

Remember when I said, "Pure functions are my favorite part of functional programming"? Well, [sum types](https://en.wikipedia.org/wiki/Tagged_union) are a close second.

A "sum" type is the opposite of a "product" type. This Python object is an example of a product type:

man.studies\_finance = True

man.has\_trust\_fund = False

The total number of combinations a man can have is 4, the product of 2 \* 2:

| **studies\_finance** | **has\_trust\_fund** |
| --- | --- |
| True | True |
| True | False |
| False | True |
| False | False |

If we add a third attribute, perhaps a has\_blue\_eyes boolean, the total number of possibilities multiplies again, to 8!

| **studies\_finance** | **has\_trust\_fund** | **has\_blue\_eyes** |
| --- | --- | --- |
| True | True | True |
| True | True | False |
| True | False | True |
| True | False | False |
| False | True | True |
| False | True | False |
| False | False | True |
| False | False | False |

But let's pretend that we live in a world where there are really only [three types of people](https://www.youtube.com/watch?v=tEt0IuQJX2o) that our program cares about:

1. Dateable
2. Undateable
3. Maybe dateable

We can reduce the number of cases our code needs to handle by using a (admittedly fake Pythonic) sum type with only 3 possible types:

class Person:

def \_\_init\_\_(self, name):

self.name = name

class Dateable(Person):

pass

class MaybeDateable(Person):

pass

class Undateable(Person):

pass

Then we can use the [isinstance](https://docs.python.org/3/library/functions.html#isinstance) built-in function to check if a Person is an instance of one of the subclasses. It's a clunky way to represent sum types, but hey, it's Python.

def respond\_to\_text(guy\_at\_bar):

if isinstance(guy\_at\_bar, Dateable):

return f"Hey {guy\_at\_bar.name}, I'd love to go out with you!"

elif isinstance(guy\_at\_bar, MaybeDateable):

return f"Hey {guy\_at\_bar.name}, I'm busy but let's hang out sometime later."

elif isinstance(guy\_at\_bar, Undateable):

return "Have you tried being rich?"

else:

raise ValueError("invalid Person type")

## Sum Types

As opposed to product types, which can have many (often infinite) combinations, sum types have a fixed number of possible values. To be clear: **Python doesn't really support sum types**. We have to use a workaround and invent our own little system and enforce it ourselves.

## Assignment

Whenever a document is parsed by Doc2Doc, it can either succeed or fail. In functional programming, we often represent errors as data (e.g. the ParseError class) rather than by raiseing exceptions, because exceptions are side effects. (This isn't standard Python practice, but it's useful to understand from an FP perspective)

**Complete the Parsed and ParseError subclasses.**

* Parsed represents success. It should accept a doc\_name string and a text string and save them as properties of the same name.
* ParseError represents failure. It should accept a doc\_name string and an err string and save them as properties of the same name.

The test suite uses the isinstance function to see if an error occurred based on the class type.

enom are one of the best things to

happen to programming and that's

including the release of the best

operating system of all time Windows

Vista but before we talk about enums

let's brush up on a little bit of

category Theory and despite the word

theory being added to scare

non-academics it's really not too bad in

programming types are how we categorize

things for example this partner class

contains two variables both of which are

Boolean types so the partner type

actually contains other types and this

is this is common when working with

collection types like lists tupal and

classes the interesting thing about this

partner class is that it really has four

possible combinations of values studies

Finance can be true has trust fund also

true studies Finance true has trust fund

false studies Finance false has trust

fund true studies Finance false has

trust fund false and that seems

manageable but what happens if we add a

third field well if it's also a Boolean

then the number of possibilities

actually jumps up to eight because we

have to multiply four by the available

combinations again which is two getting

eight possible total combinations if we

add another we get and if we go up to

five then we get a total number of

combinations and if we add a string into

the mix instead of a Boolean then the

number of total possible values

essentially goes up to Infinity because

a string can contain any number of

combinations of characters now this is

where the term product type and some

type come into play and they're also

sometimes referred to as algebraic data

types just to scare the non- phds the

partner class is a product type because

the total number of possible

combinations of values is the product of

the number of fields right \* \* \*

\* however many Boolean fields we had

in there but product types AR very

interesting almost everything is a

product type by default lists tupal and

classes with you know a number of

properties and the problem is that

product types result in many

possibilities and even primitive types

like strings and integers have the same

problem when you have a string type you

don't know if that string value contains

the entire text of the novel Moby Dick

or the string hello world or the empty

string there's there's so many possible

combinations that could be stored in

that type and that makes them hard to

work with right when you write a

function that accepts one of those types

as as a parameter you have endless if

else statements and switch cases and

fallbacks to handle all of the different

possibilities that could arise when your

function is called with different values

that's where some types come into play

they're just types where the number of

possibilities is limited to a finite set

a Boolean is a good example of a some

type it has two possibilities true and

false when you're working with a Boolean

all you have to think about is whether

this variable is true or false okay so

now we can finally talk about enums

enums are another some type like a

Boolean and really I like to think about

them as kind of extended booleans for

example let's say we want to write a

button class and in this button class we

want to specify that it can be one of

three colors we could have the buttons

Constructor accept just a string to

Define which color it's going to be and

we'd accept either the string red the

string green or the string blue now the

problem is that our button class now

needs to take into account that whoever

is creating a new button could pass in

any string right they could pass in the

string teal or they could just pass in a

misspelled blue and we need to handle

both of those cases the better option is

to use an enum and at least in Python

the way you can define an enum is using

the enum package you pass in the name of

the enum in our case we'll use color and

then the possible values that are valid

in this new type so in our case red

green and blue now in our code if we use

color. red it will work it will resolve

to the color red and as far as the color

type is concerned everything will work

smoothly but if we try to use a color

that was not specified when we created

the color type like teal the python

interpreter itself will actually throw

an error so now the language itself will

raise an exception if we ever try to use

the wrong value and that means that in

all of our code we no longer need to

have all of these if else statements

checking and making sure that the

variable maps to a valid value right

there might be different places in

code that have to check the value and we

don't want to rewrite all that code in a

language like python The Interpreter

will yell at us when we run the code if

we ever use an invalid value in an enum

type but in statically typed languages

like a lot of the compiled languages

like go or typescript or C we'd actually

get the errors either at compile time or

even better right in our editor telling

us that those values are not allowed for

that type that's said enom are still

useful even in a language like python

Python's not statically typed so it does

mean we'll have to actually run the code

to see the type errors but at least the

valid values red green and blue are

defined when we create the enum type so

there's a single place to reference so

we as developers can see what colors are

valid types and we won't have to keep

redefining it every time we check the

value

# Enums

Doing the admittedly weird class and isinstance() thing works, but it turns out, there's a better way in some cases. If you're trying to represent a fixed set of values (but not store additional data within them) [enums](https://docs.python.org/3/library/enum.html) are the way to go.

Let's say we have a Color variable that we want to restrict to only three possible values:

* RED
* GREEN
* BLUE

We could use a plain-old string to represent these values, but that's annoying because we have to remember all the "valid" values and defensively check for invalid ones all over our codebase. Instead, we can use an Enum:

from enum import Enum

Color = Enum('Color', ['RED', 'GREEN', 'BLUE'])

print(Color.RED) # this works, prints 'Color.RED'

print(Color.TEAL) # this raises an exception

Now Color is a sum type! At least, as close as we can get in Python.

There are a few benefits:

1. A "Color" can only be RED, GREEN, or BLUE. If you try to use Color.TEAL, Python raises an exception.
2. There is a central place to see the "valid" values for a Color.
3. Each "Color" has a "name" (e.g. Color.RED) and a "value" (e.g. 1). The value is an integer and is used under the hood instead of the name. Integers take up less memory than strings, which helps with performance.

## Assignment

Create an Enum called Doctype with values:

* PDF
* TXT
* DOCX
* MD
* HTML

## Tip

Don't forget to import the Enum class from the enum module!

# Sum Types

Unfortunately, Python does not support sum types as well as some of the other [statically typed](https://developer.mozilla.org/en-US/docs/Glossary/Static_typing) languages.

Python [does not enforce](https://docs.python.org/3/library/typing.html) your types before your code runs. That's why we need this line here to raise an Exception if a color is invalid:

def color\_to\_hex(color):

if color == Color.GREEN:

return '#00FF00'

elif color == Color.BLUE:

return '#0000FF'

elif color == Color.RED:

return '#FF0000'

# handle the case where the color is invalid

raise Exception('Unknown color')

In a language like [Rust](https://www.rust-lang.org/) we could write the same thing like this:

fn color\_to\_hex(color: Color) -> String {

match color {

Color::Green => "#00FF00".to\_string(),

Color::Blue => "#0000FF".to\_string(),

Color::Red => "#FF0000".to\_string(),

}

}

Notice how there isn't any case for an unknown value? That's because the Rust code will fail to compile (a step that happens before the code runs at all) if the Color is a different value. **This static enforcement is a huge benefit of sum types**, and it's a shame we can't get that in Python.

# Match

Let's take another look at our example [Enum](https://docs.python.org/3/library/enum.html) from the previous lesson:

Color = Enum("Color", ["RED", "GREEN", "BLUE"])

## Working with enums

Python has a match statement that tends to be a lot cleaner than a series of if/else/elif statements when we're working with a fixed set of possible values (like a sum type, or more specifically an enum):

def get\_hex(color):

match color:

case Color.RED:

return "#FF0000"

case Color.GREEN:

return "#00FF00"

case Color.BLUE:

return "#0000FF"

# default case

# (invalid Color)

case \_:

return "#FFFFFF"

If you have two values to match, you can use a tuple:

def get\_hex(color, shade):

match (color, shade):

case (Color.RED, Shade.LIGHT):

return "#FFAAAA"

case (Color.RED, Shade.DARK):

return "#AA0000"

case (Color.GREEN, Shade.LIGHT):

return "#AAFFAA"

case (Color.GREEN, Shade.DARK):

return "#00AA00"

case (Color.BLUE, Shade.LIGHT):

return "#AAAAFF"

case (Color.BLUE, Shade.DARK):

return "#0000AA"

# default case

# (invalid combination)

case \_:

return "#FFFFFF"

The value we want to compare is set after match keyword, which is then compared against different cases/patterns. If a match is found, the code in the block is executed.

## Assignment

Complete the convert\_format function. It should support 3 types of conversions:

**From MD to HTML:**

Assume the content is a single h1 tag in markdown syntax - it's a single string representing a line. Replace the leading # with an <h1> and add a </h1> to the end.

# This is a heading -> <h1>This is a heading</h1>

**From TXT to PDF:**

Simply add a [PDF] tag to the beginning and end of the content. Notice the spaces between [PDF] tags and the content:

This is some text -> [PDF] This is some text [PDF]

**From HTML to MD:**

Replace any <h1> tags with # and remove any </h1> tags.

<h1>This is a heading</h1> -> # This is a heading

**Any other conversion:**

If the input format is invalid, raise an Exception with the string Invalid type.

# Match

Let's take another look at our example [Enum](https://docs.python.org/3/library/enum.html) from the previous lesson:

Color = Enum("Color", ["RED", "GREEN", "BLUE"])

## Working with enums

Python has a match statement that tends to be a lot cleaner than a series of if/else/elif statements when we're working with a fixed set of possible values (like a sum type, or more specifically an enum):

def get\_hex(color):

match color:

case Color.RED:

return "#FF0000"

case Color.GREEN:

return "#00FF00"

case Color.BLUE:

return "#0000FF"

# default case

# (invalid Color)

case \_:

return "#FFFFFF"

If you have two values to match, you can use a tuple:

def get\_hex(color, shade):

match (color, shade):

case (Color.RED, Shade.LIGHT):

return "#FFAAAA"

case (Color.RED, Shade.DARK):

return "#AA0000"

case (Color.GREEN, Shade.LIGHT):

return "#AAFFAA"

case (Color.GREEN, Shade.DARK):

return "#00AA00"

case (Color.BLUE, Shade.LIGHT):

return "#AAAAFF"

case (Color.BLUE, Shade.DARK):

return "#0000AA"

# default case

# (invalid combination)

case \_:

return "#FFFFFF"

The value we want to compare is set after match keyword, which is then compared against different cases/patterns. If a match is found, the code in the block is executed.

## Assignment

Complete the convert\_format function. It should support 3 types of conversions:

**From MD to HTML:**

Assume the content is a single h1 tag in markdown syntax - it's a single string representing a line. Replace the leading # with an <h1> and add a </h1> to the end.

# This is a heading -> <h1>This is a heading</h1>

**From TXT to PDF:**

Simply add a [PDF] tag to the beginning and end of the content. Notice the spaces between [PDF] tags and the content:

This is some text -> [PDF] This is some text [PDF]

**From HTML to MD:**

Replace any <h1> tags with # and remove any </h1> tags.

<h1>This is a heading</h1> -> # This is a heading

**Any other conversion:**

If the input format is invalid, raise an Exception with the string Invalid type.

# Export CSV

Doc2Doc should be able to prepare and export a CSV file of whatever data you input. Comma-Separated Values are a ubiquitous text format that allows for information to be structured in a table. There is usually a header row, followed by data rows. Within rows, items are separated by commas.

## Assignment

Complete the get\_csv\_status function. It should use a match case statement to select the correct response depending on the status of the export operation. Create a function to handle each operation as follows:

### PENDING:

Return a tuple with the string "Pending..." and the data converted from a list of lists of anything, to a list of lists of strings. Try to use nested map functions to convert the data items into strings. Remember to convert from a map object back into a list.

### PROCESSING:

Return a tuple with the string "Processing..." and the data converted from a list of lists of strings into one string in CSV format.

* For each list of strings, combine the strings with join with commas inbetween to form a row.
* For each row string, combine the strings with join with newlines "\n" inbetween to form a table.

### SUCCESS:

Return a tuple with the string "Success!" and simply return the data as is.

### FAILURE:

Return a tuple with the string "Unknown error, retrying..." and the data after it has been prepared and processed into a CSV string, by combining the steps for Pending and Processing.

### Any other status:

If the input status is none of the above, raise an Exception with the string "Unknown export status".

## Tip

It's better if you try this challenge without using loops for practice, but you may use loops.

# Edit Document

Doc2Doc should be able to track changes in documents. Tracking changes is important for undoing and redoing edits. Some editors save changes and some file formats do as well.

## Assignment

Complete the handle\_edit function. It takes as input a document string, an edit\_type EditType enum, and an edit dictionary. It should use a match case statement to select the correct operation depending on the EditType. Create a function to handle each operation as follows:

### NEWLINE:

Use the edit dictionary to modify and return a copy of the document. The edit dictionary will only contain a line\_number key and integer value. Add a newline \n at the end of the line of the document corresponding to the line\_number.

### SUBSTITUTE:

Use the edit dictionary to modify and return a copy of the document. The edit dictionary will contain a insert\_text key and string value, a line\_number key and integer value, a start key and integer value and an end key and integer value. Substitute the insert\_text into the line of the document corresponding to the line\_number between the start and end indexes.

### INSERT:

Use the edit dictionary to modify and return a copy of the document. The edit dictionary will contain a insert\_text key and string value, a line\_number key and integer value, and a start key and integer value. Insert the insert\_text into the line of the document corresponding to the line\_number at the start index.

### DELETE:

Use the edit dictionary to modify and return a copy of the document. The edit dictionary will contain a line\_number key and integer value, a start key and integer value, and an end key and integer value. Remove the substring of the line of the document corresponding to the line\_number between the start and end indexes.

### Exceptions:

* If the edit\_type is none of the above, raise an Exception with the string "Unknown edit type".
* If line\_number is greater than or equal to the number of lines in the document, raise an Exception "Invalid line number".
* If start is greater than the length of the line, raise an Exception "Invalid start index".
* If end is greater than the length of the line or less than start, raise an Exception "Invalid end index".

## Tip

Take a look at the test cases in the main\_test.py file tab of the editor to see the input documents and expected\_outputs to visualize what is happening for each operation.

# Welcome to Web Servers

In a way, this course is where everything you've learned so far on the Boot.dev back-end track comes together. Building web servers is the bread-and-butter of a backend developer's day-to-day work.

This course assumes you already have a solid understanding of Go. If you don't, take a step back and take our [Go course](https://www.boot.dev/courses/learn-golang).

## Goals of this course

## What is a server?

A web [server](https://en.wikipedia.org/wiki/Server_%28computing%29) is just a computer that serves data over a network, typically the Internet. Servers run software that listens for incoming requests from clients. When a request is received, the server responds with the requested data.

Any server worth its salt can handle many requests at the same time. In Go, we use a new [goroutine](https://go.dev/tour/concurrency) for each request to handle them concurrently. Let's start by practicing with goroutines.

## Assignment

In this course, we'll be working on a product called "Chirpy". Chirpy is a social network similar to Twitter.

One of Chirpy's servers is processing requests unbelievably slowly. Use a goroutine to fix the bug in the handleRequests (not handleRequest) function. The server should be able to process all the requests within the time limit.

package main

import (

"fmt"

"time"

)

func handleRequests(reqs <-chan request) {

for req := range reqs {

handleRequest(req)

}

}

// don't touch below this line

type request struct {

path string

}

func main() {

reqs := make(chan request, 100)

go handleRequests(reqs)

for i := 0; i < 4; i++ {

reqs <- request{path: fmt.Sprintf("/path/%d", i)}

time.Sleep(500 \* time.Millisecond)

}

time.Sleep(5 \* time.Second)

fmt.Println("5 seconds passed, killing server")

}

func handleRequest(req request) {

fmt.Println("Handling request for", req.path)

time.Sleep(2 \* time.Second)

fmt.Println("Done with request for", req.path)

}

Intro

go is blazingly fast when it comes to

writing back and web servers let's talk

about what makes ghost servers so fast

and how it goes approach to handling

HTTP requests compares to JavaScript and

python specifically node.js in Python so

first of all what am I talking about I'm

talking about web servers so I'll draw a

Web Server Performance

server over here on the right and we're

concerned in this video about the

performance of web servers and generally

speaking when we're talking about

performance of web servers we're talking

about how many requests

how many requests

can the server handle

right how many can it handle at the same

time and how quickly can it process

those requests it would be pretty

useless if I were browsing Facebook that

you couldn't be browsing Facebook at the

same time right a good server should be

able to handle many many clients at the

same time so the question is how do

different server Technologies handle

this kind of concurrent processing of

requests and we're going to start with

JavaScript and then we'll talk about

Python and go I kind of want to finish

with go because it's where I've spent

most of my career and I'm pretty excited

about talking about how go approaches

this whole concurrency problem I think

it's really simple elegant and it

results in really fast uh server side

code so how does node.js handle this

How does Node.js handle concurrency?

concurrency problem well right out of

the gate we have a problem in node.js

and that's that node.js is single

threaded so most computers these days

have multiple cores let me kind of draw

this out

so most computers within their CPU have

multiple cores let's say it's a quad

core processor right so it has four

cores

and what that means is the computer can

actually do four different computations

at the same time and many programs can

take advantage of that and kind of run

their code in parallel across those four

cores so if you have a program that

would typically take say eight seconds

running on One Core you might be able to

if you've written your code well run it

in two seconds by utilizing all four

cores right eight divided by four is two

well at the end of the day node

basically just doesn't allow this a node

program is single threaded so at any

given time it will only be using one

core in your processor now this might be

sounding pretty bad for node.js but it's

actually not too much of a problem let's

talk about why so most web servers

aren't doing too much with the CPU at

any given time most web servers are

processing a request and kind of

offloading most of the the

computationally heavy stuff to another

server typically to a data database so

for example if you have down here let's

say this is a a postgres database

we'll draw like that

what typically happens with any given

request is that the request comes in for

the Ser to the server just for a

millisecond or so before the server

sends a second request down to the

database and now we're waiting for

potentially milliseconds or so for

the postgres database to respond with

its request and the second it comes back

or I should say the the millisecond it

comes back the server is able to respond

to the client when we have a workflow

that looks like this we call it an i o

bound workflow i o bound

and again basically what that means is

the server itself isn't doing anything

too computationally heavy right maybe

it's checking an authentication token

but that's something that can happen

very very quickly what what's taking up

the majority of the time when processing

an individual request is actually

sending another Network request and

waiting on another kind of third-party

server when we have a workflow like that

we call it an i o bound workflow and

again this is very very common in kind

of crud apps right create read update

delete apps or restful back-end Services

because a lot of the computation is

being offloaded to a database like

postgres or MySQL so when we have an i o

bound workflow like this right where

we're mostly waiting on network

connections node.js actually performs

really really well the way node.js

processes multiple requests at the same

time is through the use of promises

promises right which you may be familiar

with through the promise syntax in

JavaScript or maybe the async await

syntax but the point is the way promises

work is they essentially defer

processing of something

um until uh sometime in the future so

what ends up happening is a request

comes into the server

right and then almost immediately the

server is able to kind of relay its

Associated let's say database request

down to the database which may take

let's say several hundred milliseconds

and in that time while the server is

still waiting to respond to this first

request another request comes in

now because the node.js runtime has

effectively deferred the processing of

that first request it's able to now

handle the second request right so the

second request comes in and let's say

that second request maybe doesn't even

require i o it doesn't even need to go

to the database to service that request

maybe it's just responding with some

HTML right well that request can be

responded to

and then when when this one finally

comes back from the database then that

first process or that first request can

be processed so any individual request

is only being processed one at a time

but they're kind of interleaved or

interwoven in the order in which they're

processed based on uh kind of whether or

not promises or asynchronous processes

are being weighted for so like I said as

long as you're not pounding the CPU as

long as you have an i o bound workflow

this works really really well so the

question is when does node.js start

slowing down right when is it not such a

good choice at least from a performance

standpoint and the answer is when you

have a CPU bound workflow

like we talked about node.js only takes

advantage of One Core at a time so if

you have four or eight cores available

on your server and you're running a

node.js process it's not going to take

advantage of all of the hardware that

you have available to you so if you have

a service that's not just offloading

requests to a database but instead is

maybe doing some pretty heavy processing

let's say image processing Right video

processing things that actually take a

lot of Hardware resources on the server

itself well node.js is going to struggle

a little bit more so while there are

always exceptions to every rule as a

general rule node.js does really well

with i o bound workflows and tends to

How does Python handle concurrency?

really struggle with CPU bound workflows

okay so how does python play into this

right well the interesting thing is when

you're deploying python back-end servers

you'll typically be using a framework

like Django or flask that's not

necessarily always the case but it's

pretty commonly the case so I'm going to

talk mostly about how the model Works

within the context of those Frameworks

the architecture for deploying a node

server is fairly simple you have the

node runtime running your application

code and it handles all the HTTP

requests itself deploying python service

to production is a little more involved

there's basically one more moving part

and it's called the whiskey

or the web server Gateway interface and

it's another piece of software that sits

right here and you may have heard of a

US key that would be one kind of example

of the web server Gateway interface

there are other uh popular Gateway

interfaces as well but for now we'll

just kind of

wave our hands and say this is the this

is the whizkey interface

the requests actually come in from the

internet to the web server Gateway

interface

and from there they're sent to your

python application

this is the

python code

now I want you to keep in mind that I am

waving my hands a lot here there are a

lot of details that I'm not mentioning

that go on under the hood but it is

important to understand that from a high

level python code generally speaking

runs on a single core and is not

asynchronous now there is growing

support for async python but a lot of

these web Frameworks especially if

you're working on Legacy code won't

support it so python servers can only

process a single request at a time and

even when they offload that request to a

database again let's just say like a

postgres database

they can't handle the next request until

that first one

comes back so you might be thinking

that's insane like that's going to just

take you're going to have such a

performance hit that will be unusable

and that would be the case except for

the fact that the web server Gateway

interface orchestrates with your python

application code your framework to kind

of pool these applications so we'll

actually spin up multiple instances or

multiple processes

of the Python code

right and it will kind of offload

some of the requests kind of in parallel

to different instances of the Python

application code

so let's compare how JavaScript and

python handle these HTTP requests and

what we could kind of expect again at

least from a high level when it comes to

Performance well first of all the python

way of doing things

um or at least the Django flask whiskey

way of doing things can take advantage

of all the cores on a machine kind of

out of the box which is great if you

have a lot of cores available to you

let's say or cores then it's likely

that the python deployment will be able

to do CPU intensive workloads a little

better right because the node.js server

will be bound to Just One Core out of

the eight or one out of the now

again I'm not doing any benchmarks here

this is a very high level overview and

kind of estimation of how the

performance is going to shake out

anytime you're doing really hardcore

performance engineering on a live system

of course you would want to take

benchmarks right actually time things

and see how changing things up affects

the runtime when it comes to IO sound

workflows on the other hand I would

actually expect the node server to do a

little better and the reason is when the

python setup wants to process two kind

of i o requests at the same time right

so something like this

right two different requests are being

sent off to say a database

there's actually some overhead there

we're essentially spinning up you know a

new operating system thread right new

instance of The Interpreter

um there's extra stuff going on in order

to just essentially send another HTTP

request whereas the node Server doesn't

have that overhead all it does is sit

there and wait while it shoots off yet

How does Golang handle concurrency?

another HTTP request into the Aether now

let's talk about go go has its own

architecture for dealing with concurrent

HTTP requests so we have the same

diagram I'll draw it really quickly

and we've got our go server now go like

node has no kind of intermediary between

it and the requests coming in unless

it's like a load balance or something

like that there is no web server Gateway

interface the big difference between go

and node is that out of the box go has

great support for concurrency

right so in our little CPU diagram here

we talked about how JavaScript or the

node.js runtime can only use one core at

a time well go has absolutely no problem

using all four cores which just means

that if we're doing CPU intensive stuff

we are able to get kind of the best bang

for our buck out of our Hardware so

because of that you could say that go is

really great it's really great for CPU

intensive workloads so the question is

how does go stack up when it comes to i

o bound workflows and the answer is that

it does a really great job and the

reason is every single HTTP request that

comes in gets its own go routine

but a go routine is not the same as an

operating system thread which is what

python would rely on to kind of do uh

many things at the same time go routines

are much more lightweight so more

similar to node when we need to handle

multiple requests at the same time we're

spinning up multiple go routines but

that doesn't come with nearly as much

overhead as you would get with the

python architecture so just to kind of

reiterate we're running one go server

here right one instance of the

application and that one instance is

going to be able to handle many requests

and spin off a new go routine for every

HTTP request that comes in and that's

okay because go routines are very

lightweight so we kind of get the best

of both worlds in terms of being able to

do pretty heavy CPU intensive workloads

and handle i o bound workloads without a

How do you decide which technology to use?

problem so we've gotten to this point in

the video and you might be thinking well

great so shouldn't we just always write

go what well I'm a huge go Advocate I

really like go but there are valid

reasons to use node.js Django and flask

let's talk about some of those the first

thing to mention is that performance

isn't everything right if I were to redo

this video and talk about Go versus rust

rust will generally outperform go there

are other things that we consider when

we're picking a programming language for

a specific job for example python is

very easy to get started with it's easy

to write it's easy to read if you have a

team of python developers I wouldn't

recommend writing your server in go

right it's just going to be easier

you're going to have less problems if

you stick with the stack that your team

is already familiar with another thing

you might consider when choosing a

technology for a back-end project is

what libraries are available right for

example python is well known for having

amazing machine learning deep learning

libraries that you can kind of just

import and use right so if you know that

there are some libraries that you want

to use in Python it can be worth it to

just take the performance hit and use

those libraries right you'll save

yourself a ton of time one great reason

to use node.js is simply that your front

end is probably written in JavaScript so

by writing your back end in JavaScript

as well you can save yourself some

Final thoughts

headache and keep your stack kind of

smaller in scope so to wrap this up I

would be very surprised if anyone ever

switches away from go to python or

JavaScript for performance reasons right

there are valid reasons to use other

programming languages on the back end

but I would just be amazed if someone

chose JavaScript or python over go

for performance reasons

# Goroutines in servers

In Go, goroutines are used to serve many requests at the same time, but not all servers are quite so performant.

Go was built by Google, and one of the purposes of its creation was to power Google's massive web infrastructure. Go's goroutines are a great fit for web servers because they're lighter weight than operating system threads, but still take advantage of multiple cores. Let's compare a Go web server's concurrency model to other popular languages and frameworks.

## Node.js / Express.js

In JavaScript land, servers are typically single-threaded. A [Node.js](https://nodejs.org/en/) server (often using the [Express](https://expressjs.com/) framework) only uses one CPU core at a time. It can still handle many requests at once by using an [async event loop](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Event_loop). That just means whenever a request has to wait on I/O (like to a database), the server puts it on pause and does something else for a bit.

This might sound horribly inefficient, but it's not too bad. Node servers do just fine with the I/O workloads associated with most CRUD apps (Where processing is offloaded to the Database). You only start to run into trouble with this model when you need your server to do CPU-intensive work.

## Python / Django / Flask

[Django](https://www.djangoproject.com/) and [Flask](https://flask.palletsprojects.com/en/2.2.x/) are two of the most popular back-end Python frameworks. They're both built on top of the [WSGI](https://wsgi.readthedocs.io/en/latest/what.html) standard, which is a specification for how web servers and web applications should communicate.

Generally speaking, Python application code only processes a single request at a time. This means that if a request is being processed, the application won't do anything else until it's finished.

The only reason this isn't completely debilitating is there is a separate WSGI process (for example [uwsgi](https://uwsgi-docs.readthedocs.io/en/latest/)) that handles the concurrency of the application code. It can spawn multiple processes of the Python application to handle different requests at once.

In other words, uwsgi + Flask or uwsgi + Django are both needed to handle the same things that a single Node.js or Go server does alone.

## Takeaways

* Go servers are great for performance whether the workload is I/O or CPU-bound
* Node.js and Express work well for I/O-bound tasks, but struggle with CPU-bound tasks
* Python and Django/Flask do just fine with I/O bound tasks, but frankly, no one picks Python for its performance

I'm not saying Go is always "better" than Python or JavaScript when it comes to back-end development, but it generally outperforms them when it comes to speed

# Server

We're building a fully-fledged web server from scratch on your local machine. The test suite will make HTTP requests to your local server over [localhost](https://www.hostinger.com/tutorials/what-is-localhost). Your server will run in one terminal, while you submit tests with the [Boot.dev CLI](https://github.com/bootdotdev/bootdev) in another terminal.

## Setup

### Tools you'll need

1. A code editor. I use [VS code](https://code.visualstudio.com/), but you can use whatever you're comfortable with.
2. A command line. I work on Mac OS/Linux, so my instructions will be in Bash. I recommend [WSL 2](https://docs.microsoft.com/en-us/windows/wsl/install) if you're on Windows so you can still use Linux commands.
3. The [Go toolchain](https://golang.org/doc/install) with version 1.22+.
4. The [Boot.dev CLI](https://github.com/bootdotdev/bootdev) to run the tests. Go ahead and install it following the instructions in the README, then run bootdev login to authenticate.

The lessons in this course require at least version 1.22 of Go. If you're using an older version, you'll run into some frustrating issues!

### Set up your project

Create a new GitHub/GitLab repository for your Chirpy project, and clone it down onto your local machine. Use go mod init to create a new Go module for the project, and add a main.go file. That's where you'll be writing your code for each assignment.

Do not delete your work after each assignment! Each lesson will build upon the previous ones so we'll be reusing a lot of code.

## Assignment

The Go standard library makes it easy to build a simple server. Your task is to build and run a server that binds to localhost:8080 and always responds with a 404 Not Found response.

### Steps

1. Create a [new http.ServeMux](https://pkg.go.dev/net/http#NewServeMux)
2. Create a new [http.Server](https://pkg.go.dev/net/http#Server) struct.
   * Use the new "ServeMux" as the server's handler
   * Set the .Addr field to ":8080"
3. Use the server's [ListenAndServe](https://pkg.go.dev/net/http#Server.ListenAndServe) method to start the server
4. Build and run your server (e.g. go build -o out && ./out)
5. Open http://localhost:8080 in your browser. You should see a 404 error because we haven't connected any handler logic yet. Don't worry, that's what is expected for the tests to pass for now.

**While your server is still running**, run and submit the HTTP tests using the [Boot.dev CLI tool](https://github.com/bootdotdev/bootdev) in another terminal window.

## Tips

* Use go mod init to create a Go module for your project
* Each time you change your code you'll need to rebuild and restart your server
* Use Git to save your work as you go

bootdev run 861ada77-c583-42c8-a265-657f2c453103

**Submit**

bootdev run 861ada77-c583-42c8-a265-657f2c453103 -s

* Default base URL: http://localhost:8080
* Optionally use the -b flag to override the default baseURL

Run the CLI commands to test your solution.

1. 1. **GET /**
   * 1.

Expecting status code: 404

# Fileserver quiz

Let's go over some of the code that we used for our simple fileserver and make sure we understand the nuts and bolts.

Consider this code from the main() function and answer the questions:

srv := &http.Server{

Addr: ":" + port,

Handler: mux,

}

log.Printf("Serving files from %s on port: %s\n", filepathRoot, port)

log.Fatal(srv.ListenAndServe())

# Serving Images

You may be wondering how the fileserver knew to serve the index.html file to the root of the server. It's such a common convention on the web to use a file called index.html to serve the webpage for a given path, that the Go standard library's [FileServer](https://pkg.go.dev/net/http#FileServer) does it automatically.

When using a standard fileserver, the path to a file on disk is the same as its URL path. An exception is that index.html is served from / instead of /index.html.

## Try it out

Run your chirpy server again, and open http://localhost:8080/index.html in a new browser tab. You'll notice that you're redirected to http://localhost:8080/.

This works for all directories, not just the root!

For example:

* /index.html will be served from /
* /pages/index.html will be served from /pages
* /pages/about/index.html will be served from /pages/about

Alternatively, try opening a URL that doesn't exist, like http://localhost:8080/doesntexist.html. You'll see that the fileserver returns a 404 error.

## Assignment

Let's serve another type of file from our server: an image. Chirpy has a slick logo, and we need to serve it so that our users can load it in their browsers and mobile apps.

Download the Chirpy logo from below and add it to your project directory.

Configure its filepath so that it's accessible from this URL:

http://localhost:8080/assets/logo.png

# Workflow Tips

Servers are interesting because they're always running. A lot of the code we've written in Boot.dev up to this point has acted more like a command line tool: it runs, does its thing, and then exits.

Servers are different. They run forever, waiting for requests to come in, processing them, sending responses, and then waiting for the next request. If they didn't work this way, websites and apps would be down and unavailable all the time!

## Debugging a Server

Debugging a CLI app is simple:

1. Write some code.
2. Build and run the code.
3. See if it did what you expected.
4. If it didn't, add some logging or fix the code, and go back to step 2.

Debugging a server is a little different. The simplest way (minimal tooling) is to:

1. Write some code.
2. Build and run the code.
3. Send a request to the server using a browser or some other HTTP client.
4. See if it did what you expected.
5. If it didn't, add some logging or fix the code, and go back to step 2.

Make sure you're testing your server by hitting endpoints in the browser before submitting your answers.

## Restarting a Server

I usually use a single command to build and run my servers, assuming I'm in my main package directory:

go run .

This builds the server and runs it in one command.

To stop the server, I use ctrl+c. This sends a signal to the server, telling it to stop. The server then exits.

To start it again, I just run the same command.

Alternately, you can compile a binary and run it instead:

go build -o out && ./out

## CLI Tip

If you didn't know, you can continuously press the up arrow key on the command line to see the commands you've previously run. That way you don't need to re-type commands that you use often!

# Custom Handlers

In the previous exercise, we used the [http.FileServer](https://pkg.go.dev/net/http#FileServer) function, which simply returns a built-in [http.Handler](https://pkg.go.dev/net/http#Handler).

An http.Handler is just an interface:

type Handler interface {

ServeHTTP(ResponseWriter, \*Request)

}

Any type with a ServeHTTP method that matches the [http.HandlerFunc](https://pkg.go.dev/net/http#HandlerFunc) signature above is an http.Handler. Take a second to think about it: it makes a lot of sense! To handle an incoming HTTP request, all a function needs is a way to write a response and the request itself.

## Assignment

Let's add a readiness endpoint to the Chirpy server! Readiness endpoints are commonly used by external systems to check if our server is ready to receive traffic.

The endpoint should be accessible at the /healthz path using any HTTP method.

The endpoint should simply return a 200 OK status code indicating that it has started up successfully and is listening for traffic. The endpoint should return a Content-Type: text/plain; charset=utf-8 header, and the body will contain a message that simply says "OK" (the text associated with the 200 status code).

Later this endpoint can be enhanced to return a *503 Service Unavailable* status code if the server is not ready.

### 1. Add the readiness endpoint

I recommend using the [mux.HandleFunc](https://pkg.go.dev/net/http#ServeMux.HandleFunc) to register your handler. Your handler can just be a function that matches the signature of [http.HandlerFunc](https://pkg.go.dev/net/http#HandlerFunc):

handler func(http.ResponseWriter, \*http.Request)

Your handler should do the following:

1. Write the Content-Type header
2. Write the status code using [w.WriteHeader](https://pkg.go.dev/net/http#ResponseWriter.WriteHeader)
3. Write the body text using [w.Write](https://pkg.go.dev/net/http#ResponseWriter.Write)

### 2. Update the fileserver path

Now that we've added a new handler, we don't want potential conflicts with the fileserver handler. Update the fileserver to use the /app/ path instead of /.

Not only will you need to [mux.Handle](https://pkg.go.dev/net/http#ServeMux.Handle) the /app/ path, you'll also need to strip the /app prefix from the request path before passing it to the fileserver handler. You can do this using the [http.StripPrefix](https://pkg.go.dev/net/http#StripPrefix) function.

**Struggling?** I, Boots the Coffee Connoisseur, am happy to help! Ask me and I'll answer for a price. If you find my explanations suspect, ask a human for help in [the Discord](https://boot.dev/community)!



Copy/paste one of the following commands into your terminal:

**Run**

bootdev run 174d13f0-f887-46c6-a633-d963662fde39

**Submit**

bootdev run 174d13f0-f887-46c6-a633-d963662fde39 -s

* Default base URL: http://localhost:8080
* Optionally use the -b flag to override the default baseURL

Run the CLI commands to test your solution.

1. 1. **GET /app**
   * 1.

Expecting status code: 200

* + 2.

Expecting body to contain: <html

* + 3.

Expecting body to contain: Welcome to Chirpy

* + 4.

Expecting body to contain: </html>

1. 2. **GET /app/assets**
   * 1.

Expecting status code: 200

* + 2.

Expecting body to contain: <pre>

* + 3.

Expecting body to contain: <a href="logo.png">logo.png</a>

* + 4.

Expecting body to contain: </pre>

1. 3. **GET /healthz**
   * 1.

Expecting status code: 200

* + 2.

Expecting "Content-Type" header to contain "text/plain; charset=utf-8"

* + 3.

Expecting body to contain: OK

Solution

***Using the Bootdev CLI***

*The Bootdev CLI is the only way to submit your solution for this type of lesson. We need to be able to make HTTP requests outside of the confines of the browser.*

*You can*[*install it here*](https://github.com/bootdotdev/bootdev)*. It's a Go program hosted on GitHub, so you'll need Go installed as well. Instructions are on the GitHub page.*

# Handler Review

## Handler

An [http.Handler](https://pkg.go.dev/net/http#Handler) is any [defined type](https://go.dev/ref/spec#Type_definitions) that implements the set of methods defined by the Handler [interface](https://go.dev/tour/methods/9), specifically the ServeHTTP method.

type Handler interface {

ServeHTTP(ResponseWriter, \*Request)

}

The [ServeMux](https://pkg.go.dev/net/http#ServeMux) you used in the previous exercise is an http.Handler.

You will typically use a Handler for more complex use cases, such as when you want to implement a custom router, middleware, or other custom logic.

## HandlerFunc

type HandlerFunc func(ResponseWriter, \*Request)

You'll typically use a HandlerFunc when you want to implement a simple handler. The HandlerFunc type is just a function that matches the ServeHTTP signature above.

## Why this signature?

The Request argument is fairly obvious: it contains all the information about the incoming request, such as the HTTP method, path, headers, and body.

The ResponseWriter is less intuitive in my opinion. The response is an argument, not a return type. Instead of returning a value all at once from the handler function, we write the response to the ResponseWriter.

# Stateful Handlers

It's frequently useful to have a way to store and access state in our handlers. For example, we might want to keep track of the number of requests we've received, or we may want to pass around an open connection to a database, or credentials to an API.

## Assignment

The product managers at Chirpy want to know how many requests are being made to serve our homepage - in essence, they want to know how many people are viewing the site!

They have asked for a simple HTTP endpoint they can hit to get the number of requests that have been processed. It will return the count as plain text in the response body.

For now, they just want the number of requests that have been processed since the last time the server was started, we don't need to worry about saving the data between restarts.

## Steps

1. Create a struct that will hold any stateful, in-memory data we'll need to keep track of. In our case, we just need to keep track of the number of requests we've received.

type apiConfig struct {

fileserverHits atomic.Int32

}

The [atomic.Int32](https://pkg.go.dev/sync/atomic#Int32) type is a really cool standard-library type that allows us to safely increment and read an integer value across multiple goroutines (HTTP requests).

1. Next, write a new [middleware](https://en.wikipedia.org/wiki/Middleware) method on a \*apiConfig that increments the fileserverHits counter every time it's called. Here's the method signature I used:

func (cfg \*apiConfig) middlewareMetricsInc(next http.Handler) http.Handler {

// ...

}

The atomic.Int32 type has an [.Add()](https://pkg.go.dev/sync/atomic#Int32.Add) method.

1. [Wrap](https://en.wikipedia.org/wiki/Wrapper_function) the http.FileServer handler with the middleware method we just wrote. For example:

mux.Handle("/app/", apiCfg.middlewareMetricsInc(handler))

1. Create a new handler that writes the number of requests that have been counted as plain text in this format to the HTTP response:

Hits: x

Where x is the number of requests that have been processed. This handler should be a method on the \*apiConfig struct so that it can access the fileserverHits data.

1. Register that handler with the serve mux on the /metrics path.
2. Finally, create and register a handler on the /reset path that, when hit, will reset your fileserverHits back to 0.

It should follow the same design as the previous handlers.

# Remember, similar to the metrics endpoint, /reset will need to be a method on the \*apiConfig struct so that it can also access the fileserverHitsMiddleware

Middleware is a way to wrap a handler with additional functionality. It is a common pattern in web applications that allows us to write DRY code. For example, we can write a middleware that logs every request to the server. We can then wrap our handler with this middleware and every request will be logged without us having to write the logging code in every handler.

Here are examples of the middleware that we've written so far.

## Keeping track of the number of times a handler has been called

func (cfg \*apiConfig) middlewareMetricsInc(next http.Handler) http.Handler {

return http.HandlerFunc(func(w http.ResponseWriter, r \*http.Request) {

cfg.fileserverHits.Add(1)

next.ServeHTTP(w, r)

})

}

## Logging every request

We haven't written this one yet, but it would look something like this:

func middlewareLog(next http.Handler) http.Handler {

return http.HandlerFunc(func(w http.ResponseWriter, r \*http.Request) {

log.Printf("%s %s", r.Method, r.URL.Path)

next.ServeHTTP(w, r)

})

}

**Struggling?** I, Boots the Coffee Connoisseur, am happy to help! Ask me and I'll answer for a price. If you find my explanations suspect, ask a human for help in [the Discord](https://boot.dev/community)!



Middleware...

Is a way to speed up your application

Is a way to keep secret environment variables safe

Is a way to inject code into HTTP handlers

Is a security feature of the Go language

# Routing

The Go standard library has a lot of powerful HTTP features and, as of version 1.22, comes equipped with method-based pattern matching for routing.

Note that there are other powerful routing libraries like [Gorilla Mux](https://github.com/gorilla/mux) and [Chi](https://github.com/go-chi/chi), however, the instructions for this course will assume you are using Go's standard library. Just know that it isn't your only option!

In this lesson, we are going to limit which endpoints are available via which HTTP methods. In our current implementation, we can use any HTTP method to access any endpoint. This is not ideal.

## Try it!

Run this command to send an empty POST request to your running server:

curl -X POST http://localhost:8080/healthz

You should get an OK response - but we want this endpoint to only be available via GET requests!

## Method specific routing

Using the Go standard library, you can specify a method like this: [METHOD ][HOST]/[PATH]. For example:

mux.HandleFunc("POST /articles", handlerArticlesCreate)

mux.HandleFunc("DELETE /articles", handlerArticlesDelete)

## Assignment

1. Update the following paths to only accept GET requests:

* /healthz
* /metrics

When a request is made to one of these endpoints with a method other than GET, the server should return a 405 (Method Not Allowed) response (this is handled automatically!).

1. Update the /reset endpoint to only accept POST requests.

**Run and submit the HTTP tests using the CLI tool**.

Copy/paste one of the following commands into your terminal:

**Run**

bootdev run d1c4962d-f55e-4f0f-ac9c-913db8ef8ae8

**Submit**

bootdev run d1c4962d-f55e-4f0f-ac9c-913db8ef8ae8 -s

* Default base URL: http://localhost:8080
* Optionally use the -b flag to override the default baseURL

Run the CLI commands to test your solution.

1. 1. **GET /app**
   * 1.

Expecting status code: 200

* + 2.

Expecting body to contain: Welcome to Chirpy

1. 2. **GET /healthz**
   * 1.

Expecting status code: 200

1. 3. **POST /healthz**
   * 1.

Expecting status code: 405

1. 4. **GET /metrics**
   * 1.

Expecting status code: 200

1. 5. **POST /metrics**
   * 1.

Expecting status code: 405

1. 6. **GET /reset**
   * 1.

Expecting status code: 405

1. 7. **POST /reset**
   * 1.

Expecting status code: 200

Solution

***Using the Bootdev CLI***

*The Bootdev CLI is the only way to submit your solution for this type of lesson. We need to be able to make HTTP requests outside of the confines of the browser.*

*You can*[*install it here*](https://github.com/bootdotdev/bootdev)*. It's a Go program hosted on GitHub, so you'll need Go installed as well. Instructions are on the GitHub page.*

# Patterns

A pattern is a string that specifies the set of URL paths that should be matched to handle HTTP requests. Go's ServeMux router uses these patterns to dispatch requests to the appropriate handler functions based on the URL path of the request. As we saw in the previous lesson, patterns help organize the handling of different routes efficiently.

As previously mentioned, patterns generally look like this: [METHOD ][HOST]/[PATH]. Note that all three parts are optional.

## Rules and Definitions

### Fixed URL Paths

A pattern that exactly matches the URL path. For example, if you have a pattern /about, it will match the URL path /about and no other paths.

### Subtree Paths

If a pattern ends with a slash /, it matches all URL paths that have the same prefix. For example, a pattern /images/ matches /images/, /images/logo.png, and /images/css/style.css. As we saw with our /app/ path, this is useful for serving a directory of static files or for structuring your application into sub-sections.

### Longest Match Wins

If more than one pattern matches a request path, the longest match is chosen. This allows more specific handlers to override more general ones. For example, if you have patterns / (root) and /images/, and the request path is /images/logo.png, the /images/ handler will be used because it's the longest match.

### Host-specific Patterns

We won't be using this but be aware that patterns can also start with a hostname (e.g., www.example.com/). This allows you to serve different content based on the Host header of the request. If both host-specific and non-host-specific patterns match, the host-specific pattern takes precedence.

If you're interested, you can read more in the [ServeMux docs](https://pkg.go.dev/net/http#ServeMux).

# Monoliths and Decoupling

"Architecture" in software can mean many different things, but in this lesson, we're talking about the high-level architecture of a web application from a structural standpoint. More specifically, we are concerned with the separation (or lack thereof) between the back-end and the front-end.

When we talk about "coupling" in this context, we're talking about the coupling between the data and the presentation logic of that data. Loosely speaking, when I say "a tightly coupled front-end and back-end", what I mean is:

### Front-end: The presentation logic

If it's a web app, then this is the HTML, CSS, and JavaScript that is served to the browser which will then be used to render any dynamic data. If it's a mobile app, then this is the compiled code that is downloaded on the mobile device.

### Back-end: Raw data

For an app like YouTube, this would be videos and comments. For an app like Twitter, this might be tweets and users data. You can't embed the YouTube videos directly into the Youtube app, because a user's feed changes each time they open the app. The app downloads new raw data from Google's back-end each time the app is opened.

## Monolithic

A monolith is a single, large program that contains all of the functionality for both the front-end and the back-end of an application. It's a common architecture for web applications, and it's what we're building here in this course.

Sometimes monoliths host a REST API for raw data (like JSON data) within a subpath, like /api as shown in the image. That said, there are even more tightly coupled kinds of monoliths that inject the dynamic data directly into the HTML as well. The nice thing about separate data endpoints is that they can be consumed by any client, (like a mobile app) and not just the website. That said, injection is typically more performant, so it's a trade-off. WordPress and other website builders typically work this way.

## Decoupled

A "decoupled" architecture is one where the front-end and back-end are separated into different codebases. For example, the front-end might be hosted by a static file server on one domain, and the back-end might be hosted on a subdomain by a different server.

Depending on whether or not a load balancer is sitting in front of a decoupled architecture, the API server might be hosted on a separate domain (as shown in the image) or on a subpath, as shown in the monolithic architecture. A decoupled architecture allows for either approach.

## Assignment

For now, Chirpy is technically a monolith. That said we are keeping all the API logic decoupled in the sense that it will be served from its own namespace (path prefix). We serve the website from the app path, and we'll be serving the API from the /api path.

Let's move our non-website endpoints to the /api namespace in our routing.

To do this, prepend /api to the beginning of each of our API (non fileserver) endpoints.

**Struggling?** I, Boots the Primeval 10x Developer, am happy to help! Ask me and I'll answer for a price. If you find my explanations suspect, ask a human for help in [the Discord](https://boot.dev/community)!



Copy/paste one of the following commands into your terminal:

**Run**

bootdev run 0b877128-7dd0-4643-8fef-5d8faf9a1cc6

**Submit**

bootdev run 0b877128-7dd0-4643-8fef-5d8faf9a1cc6 -s

* Default base URL: http://localhost:8080
* Optionally use the -b flag to override the default baseURL

Run the CLI commands to test your solution.

1. 1. **POST /api/reset**
   * 1.

Expecting status code: 200

1. 2. **GET /app**
   * 1.

Expecting status code: 200

* + 2.

Expecting body to contain: Welcome to Chirpy

1. 3. **GET /app/assets/**
   * 1.

Expecting status code: 200

* + 2.

Expecting body to contain: <a href="logo.png">logo.png</a>

1. 4. **GET /api/metrics**
   * 1.

Expecting status code: 200

* + 2.

Expecting body to contain: Hits: 2

1. 5. **GET /api/healthz**
   * 1.

Expecting status code: 200

* + 2.

Expecting body to contain: OK

Solution

***Using the Bootdev CLI***

*The Bootdev CLI is the only way to submit your solution for this type of lesson. We need to be able to make HTTP requests outside of the confines of the browser.*

*You can*[*install it here*](https://github.com/bootdotdev/bootdev)*. It's a Go program hosted on GitHub, so you'll need Go installed as well. Instructions are on the GitHub page.*

/

•

Intro

Monoliths vs decoupled front and back ends | engineer explains

Unlisted

Boot dev

K subscribers

Subscribe

Share

Download

Clip

Save

views Mar

Let's go over the differences between a monolithic architecture and a decoupled one as it relates to web development.

Learn back-end development - https://boot.dev

Like & subscribe for the algo if you enjoyed the video!

== Chapter Contents ==

- Intro

- What is a Monolithic Architecture?

- Using a Monolith with a dynamic web app (mutation)

- Using a Monolith with a dynamic web app (API)

- Decoupled Architecture

- Tradeoffs between Monolithic and Decoupled Architecture

- When do Monoliths cause problems?

- Another reason to decouple: Mobile Application

- High Level Summary

Chapters

View all

Transcript

Follow along using the transcript.

Show transcript

Boot dev

K subscribers

Videos

About

Comments

Spriha N Motley Crew

Add a comment...

@vm

year ago

Thanks a lot lane as a cse undergrad these real world applications architecture always fascinated me and you certainly helped me clear my doubts as cse degree mostly focus on theories and math proofs.

Reply

Boot dev

·

reply

@saaransh

year ago

beautifully explained bro.... keep going

Reply

Boot dev

·

reply

@thecoderguy\_

year ago

Nice explanation Lane.

Reply

Boot dev

·

reply

@BraisonsCrece

year ago

Thanks for sharing Lane. Your videos are amazing, very interesting and helpful.

Keep it going!

Reply

Boot dev

·

reply

Transcript

Intro

should you use a monolith for your next

web app let's talk about it now when I

What is a Monolithic Architecture?

say monolith or monolithic architecture

what I'm talking about is a single

program that handles both the front end

and the back end of a web application

let me show you what I mean when a user

loads our web application in their

browser what happens is an HTTP request

is sent to our server this is HTTP

request and this is our server over here

and the server is going to respond with

some HTML

some CSS and maybe some JavaScript if

the website is interactive now if the

website we're talking about is just a

website or a static site where there's

nothing Dynamic happening right the

content isn't changing on a per user

basis for example then this model works

very well and we don't even really need

to think about architecture we're kind

of done at this point if instead of

working with a static site we're working

with a dynamic site then I call that a

web app and we're going to need a

different architecture in order to build

that application and all I mean by web

app here or web application is something

a little more Dynamic so imagine Twitter

for example Twitter isn't a static

website depending on who is browsing

Twitter they're going to see a different

list of tweets right the data changes

depending on who the user is how they're

using it and just the state of the world

at any given time so we already

diagrammed a simple static website but

Using a Monolith with a dynamic web app (mutation)

let's talk about how a monolith might

work in regards to a dynamic web

application and there's kind of two

different options so option number one

I'm just going to call it the mutation

option mutation

and the way it works is let's say we

have

three different users of our application

and because they're different users and

they have different data stored in our

app they're going to kind of get

different data when they load the

application right again in the case of

Twitter they'll get different Tweets in

the case of YouTube they might get

different videos when they load their

feed

so we'll draw the server over here

and the way this kind of a monolith

would work

is the HTTP request to load the page

comes in from the client

and rather than the server just

responding with the same HTML CSS and

JavaScript files for that user instead

the server is going to do a little bit

of processing maybe it's going to for

example contact

a database

right look up things like who this user

is following or what their interests are

right and then it's going to use that

information

to build a specific response for that

client and it's actually going to mutate

and respond with some HTML

CSS

and maybe some JavaScript that's unique

to that client or you unique to that

customer

um again just by literally mutating the

files so the HTML that's being sent back

is going to be a different HTML file

with different data inside of it so when

the second client sends their HTTP

request to the server again the server

is going to do some processing and

respond with a different a different

HTML CSS and JavaScript file so this is

kind of the most traditional version of

a monolith things like WordPress work

this way PHP when you're using it to do

inline HTML templating things like

golang's built-in HTML templating or the

way kind of Ruby on Rails or Python's

Django work out of the box they all work

under this assumption that you're going

to want to dynamically change the layout

of a web page on a per request basis

Using a Monolith with a dynamic web app (API)

let's talk about option number two for

monoliths which is to kind of break take

the application up into an API side and

a side that serves the front end now it

is still a monolith so we're still

talking about one single program that

does both things but in this case what

we will typically do is

change what is served based on a route

so for example maybe the root path so my

site

dot com right slash just at the root

this will serve

the front end

of the application so it'll serve kind

of a static

um a static set of HTML CSS and

JavaScript files and then under a

different route we would serve kind of

raw data via an API so it might be

something like

my site

dot com

slash API

and under that namespace we can go put

all of the routes that are going to

serve raw data so things like mysite.com

API slash users right and then all the

pages on the site would be under just

mysite.com so maybe something like

mysite.com page slash one something like

that so in kind of a high level hand

wavy way I'm going to say that part of

the monolith is serving the front end

serving the HTML the CSS and the

JavaScript and the other side of the

monolith is serving the back end via an

API so what does this look like when a

client wants to access the website so

client loads up the website what happens

is first they send a request to the

front end my site.com right and they get

back

some HTML

HTML some CSS and maybe some JavaScript

this works kind of in a very similar way

to the static file server that we talked

about earlier right the HTML the Cs as

in the JavaScript probably aren't

changing much maybe a little bit right

depending on how the monolith is written

maybe it changes a bit but most likely

it doesn't change a whole lot and very

often in this type of monolith this this

bundle of HTML CSS and JavaScript will

actually be a framework

um like a bundled output generated by a

framework like react or view this could

be

a single page application for example so

when I say single page application what

I really mean is that this HTML CSS in

JavaScript has enough logic embedded

within it that it knows how to go fetch

additional data at a later point and

kind of update its views think about an

app on your mobile phone an app on your

mobile phone doesn't have all of the

data pre-installed on it right it

doesn't have all of the YouTube videos

in the world pre-installed on the app on

your phone so the application on your

phone has to know how to reach out to

some server and download those

additional files and that's essentially

what we're talking about here there's

typically enough JavaScript

in the application that it's smart

enough to know how to do this from the

browser so what happens next the HTML

the CSS and JavaScript come back and the

browser is going to load kind of a

skeleton of the app

and once that skeleton has been loaded

and mounted into the browser the browser

is able to reach out

by using an additional HTTP request and

get some raw data from the back end

typically structured in like a Json

format or something like that the data

comes back

and then the browser is able to render

that Dynamic data within the site right

again maybe some tweets maybe some

videos something like that now it's

important to understand here that we are

still talking about a monolith right

this program over here that's running on

a server is one program and it's just

splitting up the types of requests that

are coming in based on a path in the

route right so the architecture of the

code is very different You could argue

that the architecture of the code inside

of this monolith is decoupled but it's

not decoupled in the sense that it's

still one program now let's talk about a

Decoupled Architecture

decoupled architecture again in the

context of web apps the only difference

between a decoupled architecture and

that monolith that was partially

decoupled is that we're actually

building two different servers two

different server programs right one of

them will be responsible for serving

that front end right which is again

probably a static site or a static

bundle generated by react review or one

of these front-end Frameworks

um

and the other the other program is going

to be the back end which is probably

just an API

right maybe a restful API or a graphql

API that's responsible for serving raw

data and the way the client interacts

with it is exactly the same they get

their front-end bundle

once it loads in their browser they're

able to go fetch

some data now I want to talk about some

Tradeoffs between Monolithic and Decoupled Architecture

of the trade-offs between these three

different ways of architecting a web app

so the first architecture we talked

about was a monolith that is kind of

mutating the HTML the CSS and the

JavaScript on page load again something

like a WordPress works this way there

are definitely a couple of advantages to

using an architecture that's very

monolithic like this the first is that

it's super simple right we only have one

piece of code right that we're compiling

into one web server and that server is

responsible for doing all of the data

manipulation and responding with

everything in one request so there's a

lot of Simplicity there just baked into

how it all works another reason to

mutate the HTML on the Fly is that

you'll actually have faster page loads

generally speaking if you remember from

the decoupled architecture we actually

have to go out fetch the HTML the

JavaScript and the CSS bring it back to

the browser render it at least partially

render it or at least partially parse it

and then go make a second Network

request for the data and bring that back

so potentially we're talking about

spending almost twice as long getting

the data and rendering the page when

we're talking about using a decoupled

architecture for web apps as far as

option number two where we have a

monolith it's still all one code base

but we've separated the routes for the

front end and the routes for the API

into two different kind of sections of

the application the reason to do this is

really simple it's do you need faster

deploys are you a small company right if

you're a small company you need to be

able to work quickly on the front end

and the back end at the same time maybe

all of your developers are full stack

then you can save a lot of time by

keeping everything in one place right

you'll just have one GitHub repo and

you'll just have to deploy one single

binary up to one server to get

everything working so again monoliths

are great when it comes to Simplicity

there are really two instances where we

When do Monoliths cause problems?

start to have trouble with monoliths

monoliths again tend to work really

great for small companies who are

growing and need to be rapidly

prototyping

monoliths start to have a lot of trouble

when we need to scale either in terms of

the performance of our application like

we need to serve more and more users or

we need to scale up the size of our team

the nice thing about decoupling the

front end from the back end when it

comes to Performance is that we can

scale those servers differently right if

we're getting a lot of load in terms of

having to just serve HTML and CSS and

JavaScript then we can scale up the

number of static file servers that we

have or maybe the amount of caching or

something like that on the other hand if

it's the API that's getting hammered we

can scale that up independently right

maybe we can add more compute to the

database or we can just horizontally

scale and add more and more servers to

the API side

when you bundle everything together

everything has to scale at the same pace

which just means you have a potential

for buying more Hardware than you

actually need it can also be useful to

decouple your architecture when you talk

about scaling up your teams generally

speaking you don't want all of your

engineers working on your entire

application once your application gets

to a certain size it's just too much

code to reason about so by splitting the

back end and the front end into separate

applications you can give people a

smaller kind of area of control and

their arguments we made there are other

ways to split up your application those

arguments are super valid I just want to

point out that this is one way to split

up the areas of concern between say a

back-end team and a front end the next

Another reason to decouple: Mobile Application

reason you might want to decouple your

API and your static files from each

other is because you have a mobile app

so let's draw this out

let's say you want to support kind of a

browser

a browser version of your site

as well as as well as a mobile mobile

version of your site

problem with that first architecture

that we talked about was all of the data

mutation was happening directly inside

HTML CSS and JavaScript files

HTML

HTML CSS

and JavaScript files the problem is

mobile apps don't rely on HTML CSS and

JavaScript files to render the

application these are typically compiled

applications written in Swift or kotlin

or Java or something like that so A

mobile application really needs access

to the raw data right raw video data raw

tweet data usually encoded in Json form

maybe in maybe encoded a different way

but the point is it's raw data that can

be parsed and displayed within the app

it's not embedded within HTML

so how how do we do that right well the

problem with a monolith of the first

type that we talked about is that we

don't have a way to service the mobile

app

so we kind of have two options at that

point

we can have a back end for the website

that's a monolith

and we can have a separate back ends

it's just an API

that serves

that serves the mobile app the problem

here is that we're going to be

duplicating all of our backend logic

we're going to have to have that backend

logic that mutates the the HTML over

here on the monolith and we're gonna

have to rewrite it in the API that's why

when you need a mobile app and a website

you almost always will want to split out

the API into a separate piece of

architecture now again you can do that

within the monolith right you can you

can do kind of path based routing and

the monolith can serve the API but what

you don't want to do is be kind of

mucking with the HTML because you're

just going to be doing everything twice

at that point now there are other

architectures out there specifically

when we talk about how we architect the

API we might talk about Services versus

monolithic apis versus microservices

that's kind of out of the scope of what

we're talking about in this video we're

mostly just talking about how we split

up the logic for the the back end or the

the data and the logic for the front end

in a later video we'll talk about

services and microservices so to wrap

High Level Summary

all of this up I just want to reiterate

from a high level when you might use

these different architectures

none of these architectures are bad

inherently there are always trade-offs

involved so generally speaking a more

monolithic architecture and a more

tightly coupled architecture is going to

be better for smaller simpler

applications that are being worked on by

smaller teams so if you're a startup I

think it makes a lot of sense to start

with a monolith as the team grows and as

usage of the application grows it can

make a lot of sense to start decoupling

the architecture you'll get added

benefits of scale and performance as

well as the added benefits of kind of

reduced cognitive load on each

individual team member because they only

need to work on a smaller part of the

application hope this helped be sure to

like And subscribe if you liked the

video

# Which is better?

There is always a trade-off.

## Pros for monoliths

* Simpler to get started with
* Easier to deploy new versions because everything is always in sync
* In the case of the data being embedded in the HTML, the performance can result in better UX and SEO

## Pros for decoupled architectures

* Easier to scale as traffic grows
* Easier to practice good separation of concerns as the codebase grows
* Can be hosted on separate servers and using separate technologies
* Embedding data in the HTML is still possible with pre-rendering (similar to how Next.js works), it's just more complicated

## Can we have the best of both worlds?

Perhaps. My recommendation to someone building a new application from scratch would be to start with a monolith, but to keep the API and the front-end decoupled logically within the project from the start (like we're doing with Chirpy).

That way, our app is easy to get started with, but we can migrate to a fully decoupled architecture later if we need to.

# Admin Namespace

Let's add an "admin" namespace. This is where we'll put endpoints intended for internal administrative use. Note: there's nothing inherently more secure about this namespace, it's just an organizational structure.

## Assignment

1. Swap out the GET /api/metrics endpoint, which just returns plain text, for a GET /admin/metrics that returns HTML to be rendered in the browser. Use this template:

<html>

<body>

<h1>Welcome, Chirpy Admin</h1>

<p>Chirpy has been visited %d times!</p>

</body>

</html>

Where %d is replaced with the number of times the page has been loaded.

* Make sure you use the Content-Type header to set the response type to text/html so that the browser knows how to render it.
* Try loading http://localhost:8080/admin/metrics in your browser, and in another tab load http://localhost:8080/app a few times. Refreshing the admin page should show the updated count.

1. Update the POST /api/reset to POST /admin/reset. It should behave the same way.

**Run and submit the HTTP tests using the CLI tool**.

**Struggling?** I, Boots the Bane of End-Users, am happy to help! Ask me and I'll answer for a price. If you find my explanations suspect, ask a human for help in [the Discord](https://boot.dev/community)!



Copy/paste one of the following commands into your terminal:

**Run**

bootdev run 892b38f7-d154-4591-ac63-a9fbc2a38187

**Submit**

bootdev run 892b38f7-d154-4591-ac63-a9fbc2a38187 -s

* Default base URL: http://localhost:8080
* Optionally use the -b flag to override the default baseURL

Run the CLI commands to test your solution.

1. 1. **POST /admin/reset**
   * 1.

Expecting status code: 200

1. 2. **GET /admin/metrics**
   * 1.

Expecting status code: 200

* + 2.

Expecting body to contain: Welcome, Chirpy Admin

* + 3.

Expecting body to contain: Chirpy has been visited 0 times!

1. 3. **GET /app**
   * 1.

Expecting status code: 200

* + 2.

Expecting body to contain: Welcome to Chirpy

1. 4. **GET /admin/metrics**
   * 1.

Expecting status code: 200

* + 2.

Expecting body to contain: Chirpy has been visited 1 times!

1. 5. **GET /app**
   * 1.

Expecting status code: 200

* + 2.

Expecting body to contain: Welcome to Chirpy

1. 6. **GET /app**
   * 1.

Expecting status code: 200

* + 2.

Expecting body to contain: Welcome to Chirpy

1. 7. **GET /app**
   * 1.

Expecting status code: 200

* + 2.

Expecting body to contain: Welcome to Chirpy

1. 8. **GET /admin/metrics**
   * 1.

Expecting status code: 200

* + 2.

Expecting body to contain: Chirpy has been visited 4 times!

Solution

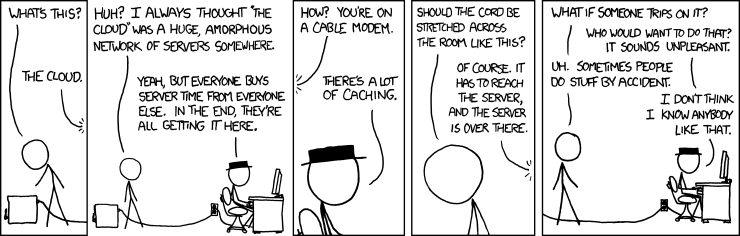
***Using the Bootdev CLI***

*The Bootdev CLI is the only way to submit your solution for this type of lesson. We need to be able to make HTTP requests outside of the confines of the browser.*

*You can*[*install it here*](https://github.com/bootdotdev/bootdev)*. It's a Go program hosted on GitHub, so you'll need Go installed as well. Instructions are on the GitHub page.*

# Deployment Options

We won't go in-depth with deployment instructions right now; that said, let's talk about how our choice of project architecture affects our deployment options, and how we could deploy our application in the future. We'll only talk about cloud deployment options here, and by the "cloud" I'm just referring to a remote server that's managed by a third-party company like Google or Amazon.



* [xkcd](https://xkcd.com/908/)

Using a cloud service to deploy applications is super common these days because it's easy, fast, and cheap.

That said, it's still possible to deploy to a local or on-premise server, and some companies still do that, but it's not as common as it used to be.

## Monolithic deployment

Deploying a monolith is straightforward. Because your server is just one program, you just need to get it running on a server that's exposed to the internet and point your DNS records to it.

You could upload and run it on classic server, something like:

* AWS EC2
* GCP Compute Engine (GCE)
* Digital Ocean Droplets
* Azure Virtual Machines

Alternatively, you could use a platform that's specifically designed to run web applications, like:

* Heroku
* Google App Engine
* Fly.io
* AWS Elastic Beanstalk

## Decoupled deployment

With a decoupled architecture, you have two different programs that need to be deployed. You would typically deploy your back-end to the same kinds of places you would deploy a monolith.

For your front-end server, you can do the same, or you can use a platform that's specifically designed to host static files and server-side rendered front-end apps, something like:

* Vercel
* Netlify
* GitHub Pages

Because the front-end bundle is likely just static files, you can host it easily on a [CDN (Content Delivery Network)](https://www.cloudflare.com/learning/cdn/what-is-a-cdn/) inexpensively.

## More powerful options

If you want to be able to scale your application up and down in specific ways, or you want to add other back-end servers to your stack, you might want to look into container orchestration options like Kubernetes and Docker Swarm.

## Don't worry about all this stuff!

I'm trying to gently introduce you to some popular technologies and how they work together, but you don't need to memorize all of these products and options.

# HTTP Clients

So far, you have probably been using a browser to test your server. That works fine with simple GET requests (the kind of request a browser sends when you type a URL into the address bar), but it's not very useful for any other HTTP methods or requests with custom headers and bodies.

## Debugging your endpoints

Servers are built to be used by clients. As you develop your code, you should be using a tool that makes sending one-off requests to your server easy! Here are some of my favorites:

* [Thunder Client for VS Code](https://marketplace.visualstudio.com/items?itemName=rangav.vscode-thunder-client)
* [REST Client for VS Code](https://marketplace.visualstudio.com/items?itemName=humao.rest-client)
* [Postman for VS Code](https://marketplace.visualstudio.com/items?itemName=Postman.postman-for-vscode)
* [cURL](https://curl.se/)
* [Postman](https://www.postman.com/)

Use whichever client you like, but make sure you're using one!

## Why do I like Thunder Client?

Thunder Client is built into VS Code, so I don't need to leave my editor, and it's fairly minimalistic.

**Struggling?** I, Boots the Bear with a Back-End, am happy to help! Ask me and I'll answer for a price. If you find my explanations suspect, ask a human for help in [the Discord](https://boot.dev/community)!



Which is a browser NOT good at testing?

POST requests

GET requests

Become a member for quiz access

Profile

4

0

**10**

Ch 4: JSON

2: JSON

sharpshooter0

milestone19

daily streak2

0

sharpshooter armor0

0

0

frozen flame0

You're in guest mode!

# JSON

Hopefully, by now you already know what JSON is. If not, you should go back and take the Learn HTTP Clients course [here first](https://www.boot.dev/courses/learn-http-clients-golang).

What you may be new to is handling and parsing JSON on the server side, rather than sending it as a client.

If you want to take a super deep dive into JSON in Go, then you can [read this post here](https://blog.boot.dev/golang/json-golang/). With that in mind, you don't need to! I'll give you the relevant info below.

## Decode JSON request body

It's very common for POST requests to send JSON data in the request body. Here's how you can handle that incoming data:

{

"name": "John",

"age": 30

}

func handler(w http.ResponseWriter, r \*http.Request){

type parameters struct {

// these tags indicate how the keys in the JSON should be mapped to the struct fields

// the struct fields must be exported (start with a capital letter) if you want them parsed

Name string `json:"name"`

Age int `json:"age"`

}

decoder := json.NewDecoder(r.Body)

params := parameters{}

err := decoder.Decode(&params)

if err != nil {

// an error will be thrown if the JSON is invalid or has the wrong types

// any missing fields will simply have their values in the struct set to their zero value

log.Printf("Error decoding parameters: %s", err)

w.WriteHeader(500)

return

}

// params is a struct with data populated successfully

// ...

}

## Encode JSON response body

func handler(w http.ResponseWriter, r \*http.Request){

// ...

type returnVals struct {

// the key will be the name of struct field unless you give it an explicit JSON tag

CreatedAt time.Time `json:"created\_at"`

ID int `json:"id"`

}

respBody := returnVals{

CreatedAt: time.Now(),

ID: 123,

}

dat, err := json.Marshal(respBody)

if err != nil {

log.Printf("Error marshalling JSON: %s", err)

w.WriteHeader(500)

return

}

w.Header().Set("Content-Type", "application/json")

w.WriteHeader(200)

w.Write(dat)

}

## Assignment

At Chirpy, we have a silly rule that says all Chirps must be 140 characters long or less.

Add a new endpoint to the Chirpy API that accepts a POST request at /api/validate\_chirp. It should expect a JSON body of this shape:

{

"body": "This is an opinion I need to share with the world"

}

If any errors occur, it should respond with an appropriate HTTP status code and a JSON body of this shape:

{

"error": "Something went wrong"

}

For example, if the Chirp is too long, respond with a 400 code and this body:

{

"error": "Chirp is too long"

}

If the Chirp is valid, respond with a 200 code and this body:

{

"valid": true

}

## Tips

Use an HTTP client like [Thunder Client](https://marketplace.visualstudio.com/items?itemName=rangav.vscode-thunder-client) to test your POST requests.

Use [json.Marshal()](https://pkg.go.dev/encoding/json#Marshal) like the example above to remove whitespace in your encoded data.

# JSON Review

## Struct 1

type parameters struct {

Name string `json:"name"`

Age int `json:"age"`

School struct {

Name string `json:"name"`

Location string `json:"location"`

} `json:"school"`

}

## Struct 2

type parameters struct {

name string `json:"name"`

Age int `json:"age"`

}

## Struct 3

type parameters struct {

Name string

Age int

}

**Struggling?** I, Boots the Primeval 10x Developer, am happy to help! Ask me and I'll answer for a price. If you find my explanations suspect, ask a human for help in [the Discord](https://boot.dev/community)!



What are the keys in the JSON representation of struct 1?

name, age, school, school.name, school.location

name, age, school

Name, Age, School, School.Name, School.Location

Name, Age, School

Become a member for quiz access

# The Profane

Not only do we validate that Chirps are under 140 characters, but we also have a list of words that are not allowed.

## Assignment

We need to update the /api/validate\_chirp endpoint to replace all "profane" words with 4 asterisks: \*\*\*\*.

Assuming the length validation passed, replace any of the following words in the Chirp with the static 4-character string \*\*\*\*:

* kerfuffle
* sharbert
* fornax

Be sure to match against uppercase versions of the words as well, but not punctuation. "Sharbert!" does not need to be replaced, we'll consider it a different word due to the exclamation point. Finally, instead of the valid boolean, your handler should return the cleaned version of the text in a JSON response:

### Example input

{

"body": "This is a kerfuffle opinion I need to share with the world"

}

### Example output

{

"cleaned\_body": "This is a \*\*\*\* opinion I need to share with the world"

}

## Tips

Use an HTTP client like [*Thunder Client*](https://marketplace.visualstudio.com/items?itemName=rangav.vscode-thunder-client) to test your POST requests.

I'd recommend creating two helper functions:

* respondWithError(w http.ResponseWriter, code int, msg string)
* respondWithJSON(w http.ResponseWriter, code int, payload interface{})

These helpers are not required but might help [DRY](https://blog.boot.dev/clean-code/dry-code/) up your code when we add more endpoints in the future.

I'd also recommend breaking the bad word replacement into a separate function. You can even write some unit tests for it!

Here are some useful standard library functions:

* [strings.ToLower](https://pkg.go.dev/strings#ToLower)
* [strings.Split](https://pkg.go.dev/strings#Split)
* [strings.Join](https://pkg.go.dev/strings#Join)

**Struggling?** I, Boots the Magnificent, am happy to help! Ask me and I'll answer for a price. If you find my explanations suspect, ask a human for help in [the Discord](https://boot.dev/community)!



Copy/paste one of the following commands into your terminal:

**Run**

bootdev run 7cde3fa8-f38a-444e-92a6-83166a905cb0

**Submit**

bootdev run 7cde3fa8-f38a-444e-92a6-83166a905cb0 -s

* Default base URL: http://localhost:8080
* Optionally use the -b flag to override the default baseURL

Run the CLI commands to test your solution.

1. 1. **POST /api/validate\_chirp**

**Request Body:**

{

"body": "I had something interesting for breakfast"

}

* + 1.

Expecting status code: 200

* + 2.

Expecting JSON at

.cleaned\_body

 to be equal to

I had something interesting for breakfast

1. 2. **POST /api/validate\_chirp**

**Request Body:**

{

"body": "I hear Mastodon is better than Chirpy. sharbert I need to migrate",

"extra": "this should be ignored"

}

* + 1.

Expecting status code: 200

* + 2.

Expecting JSON at

.cleaned\_body

 to be equal to

I hear Mastodon is better than Chirpy. \*\*\*\* I need to migrate

1. 3. **POST /api/validate\_chirp**

**Request Body:**

{

"body": "I really need a kerfuffle to go to bed sooner, Fornax !"

}

* + 1.

Expecting status code: 200

* + 2.

Expecting JSON at

.cleaned\_body

 to be equal to

I really need a \*\*\*\* to go to bed sooner, \*\*\*\* !

1. 4. **POST /api/validate\_chirp**

**Request Body:**

{

"body": "lorem ipsum dolor sit amet, consectetur adipiscing elit, sed do eiusmod tempor incididunt ut labore et dolore magna aliqua. Ut enim ad minim veniam, quis nostrud exercitation ullamco laboris nisi ut aliquip ex ea commodo consequat. Duis aute irure dolor in reprehenderit in voluptate velit esse cillum dolore eu fugiat nulla pariatur. Excepteur sint occaecat cupidatat non proident, sunt in culpa qui officia deserunt mollit anim id est laborum."

}

* + 1.

Expecting status code: 400

Become a member to submit

Solution

***Using the Bootdev CLI***

*The Bootdev CLI is the only way to submit your solution for this type of lesson. We need to be able to make HTTP requests outside of the confines of the browser.*

*You can*[*install it here*](https://github.com/bootdotdev/bootdev)*. It's a Go program hosted on GitHub, so you'll need Go installed as well. Instructions are on the GitHub page.*

# Storage

Arguably the most important part of your typical web application is the storage of data. It would be pretty useless if each time you logged into your account on YouTube, Twitter or GitHub, all of your subscriptions, tweets, or repositories were gone.

## Memory vs Disk

When you run a program on your computer (like our HTTP server), the program is loaded into memory. Memory is a lot like a scratch pad. It's fast, but it's not permanent. If the program terminates or restarts, the data in memory is lost.

When you're building a web server, any data you store in memory (in your program's variables) is lost when the server is restarted. Any important data needs to be saved to disk via the file system.

## Option 1: Raw Files

We could take our user's data, serialize it to JSON, and save it to disk in .json files (or any other format for that matter). It's simple, and will even work for small applications. Trouble is, it will run into problems fast:

* **Concurrency**: If two requests try to write to the same file at the same time, you'll get overwritten data.
* **Scalability**: It's not efficient to read and write large files to disk for every request.
* **Complexity**: You'll have to write a lot of code to manage the files, and the chances of bugs are high.

## Option 2: A database

At the end of the day, a database technology like MySQL, PostgreSQL, or MongoDB "just" writes files to disk. The difference is that they also come with all the fancy code and algorithms that make managing those files efficient and safe. In the case of a SQL database, the files are abstracted away from us entirely. You just write SQL queries and let the DB handle the rest.

**We will be using option 2:**[**PostgreSQL**](https://www.postgresql.org/)**.** It's a production-ready, open-source SQL database. It's a great choice for many web applications, and as a back-end engineer, it might be the single most important database to be familiar with.

## Assignment

1. Install Postgres.

**Mac OS with**[**brew**](https://brew.sh/)

brew install postgresql@15

**Linux / WSL (Debian). Here are the**[**docs from Microsoft**](https://learn.microsoft.com/en-us/windows/wsl/tutorials/wsl-database#install-postgresql)**, but simply:**

sudo apt update

sudo apt install postgresql postgresql-contrib

1. Ensure the installation worked. The psql command-line utility is the default client for Postgres. Use it to make sure you're on version 14+ of Postgres:

psql --version

1. (Linux only) Update postgres password:

sudo passwd postgres

Enter a password, and be sure you won't forget it. You can just use something easy like postgres.

1. Start the Postgres server in the background

* Mac: brew services start postgresql@15
* Linux: sudo service postgresql start

1. Connect to the server. I recommend simply using the psql client. It's the "default" client for Postgres, and it's a great way to interact with the database. While it's not as user-friendly as a GUI like [PGAdmin](https://www.pgadmin.org/), it's a great tool to be able to do at least basic operations with.

Enter the psql shell:

* Mac: psql postgres
* Linux: sudo -u postgres psql

You should see a new prompt that looks like this:

postgres=#

1. Create a new database. I called mine chirpy:

CREATE DATABASE chirpy;

1. Connect to the new database:

\c chirpy

You should see a new prompt that looks like this:

chirpy=#

1. Set the user password (Linux only)

ALTER USER postgres PASSWORD 'postgres';

For simplicity, I used postgres as the password. Before, we altered the system user's password, now we're altering the database user's password.

1. Query the database

From here you can run SQL queries against the chirpy database. For example, to see the version of Postgres you're running, you can run:

SELECT version();

If everything is working, run and submit the tests to move on.

**Struggling?** I, Boots the Hump Day Holdout, am happy to help! Ask me and I'll answer for a price. If you find my explanations suspect, ask a human for help in [the Discord](https://boot.dev/community)!



Copy/paste one of the following commands into your terminal:

**Run**

bootdev run c3e215a5-1d8f-437b-9f89-3606118800ec

**Submit**

bootdev run c3e215a5-1d8f-437b-9f89-3606118800ec -s

Run the CLI commands to test your solution.

1. **psql --version**
   * Expecting exit code: 0

Become a member to submit

***Using the Bootdev CLI***

*The Bootdev CLI is the only way to submit your solution for this type of lesson. We need to be able to run commands in your environment to verify your solution.*

*You can*[*install it here*](https://github.com/bootdotdev/bootdev)*. It's a Go program hosted on GitHub, so you'll need Go installed as well. Instructions are on the GitHub page.*

# Goose Migrations

[Goose](https://github.com/pressly/goose) is a database migration tool written in Go. It runs migrations from a set of SQL files, making it a perfect fit for this project (we wanna stay close to the raw SQL).

## What is a migration?

A migration is just a set of changes to your database table. You can have as many migrations as needed as your requirements change over time. For example, one migration might create a new table, one might delete a column, and one might add 2 new columns.

An "up" migration moves the state of the database from its current schema to the schema that you want. So, to get a "blank" database to the state it needs to be ready to run your application, you run all the "up" migrations.

If something breaks, you can run one of the "down" migrations to revert the database to a previous state. "Down" migrations are also used if you need to reset a local testing database to a known state.

## Users

Our API needs to support the standard CRUD operations for "users" - the people logging into and using our application.

## Assignment

1. Install Goose.

Goose is just a command line tool that happens to be written in Go. I recommend [installing](https://github.com/pressly/goose#install) it using go install:

go install github.com/pressly/goose/v3/cmd/goose@latest

Run goose -version to make sure it's installed correctly.

1. Create a users migration in a new sql/schema directory.

A "migration" in Goose is just a .sql file with some SQL queries and some special comments. Our first migration should just create a users table. The simplest format for these files is:

number\_name.sql

For example, I created a file in sql/schema called 001\_users.sql with the following contents:

-- +goose Up

CREATE TABLE ...

-- +goose Down

DROP TABLE users;

Write out the CREATE TABLE statement in full, I left it blank for you to fill in. A user should have 4 fields:

* id: a [UUID](https://blog.boot.dev/clean-code/what-are-uuids-and-should-you-use-them/) that will serve as the primary key
* created\_at: a TIMESTAMP that can not be null
* updated\_at: a TIMESTAMP that can not be null
* email: TEXT that can not be null and must be unique

The -- +goose Up and -- +goose Down comments are required. They tell Goose how to run the migration in each direction.

1. Get your connection string. A connection string is just a URL with all of the information needed to connect to a database. The format is:

protocol://username:password@host:port/database

Here are examples:

* Mac OS (no password, your username): postgres://wagslane:@localhost:5432/chirpy
* Linux (password from last lesson, postgres user): postgres://postgres:postgres@localhost:5432/chirpy

Test your connection string by running psql, for example:

psql "postgres://wagslane:@localhost:5432/chirpy"

It should connect you to the chirpy database directly. If it's working, great. exit out of psql and save the connection string.

1. Run the up migration.

cd into the sql/schema directory and run:

goose postgres <connection\_string> up

Run your migration! Make sure it works by using psql to find your newly created users table:

psql chirpy

\dt

1. Run the down migration to make sure it works (it should just drop the table).
2. When you're satisifed, run the up migration again to recreate the table.

With the table created, submit the CLI tests and move on.

**Struggling?** I, Boots the Lover of Salmon, am happy to help! Ask me and I'll answer for a price. If you find my explanations suspect, ask a human for help in [the Discord](https://boot.dev/community)!



Copy/paste one of the following commands into your terminal:

**Run**

bootdev run ea036a3f-6fa3-446a-ba20-c04cb913e12a

**Submit**

bootdev run ea036a3f-6fa3-446a-ba20-c04cb913e12a -s

Run the CLI commands to test your solution.

1. **goose --version**
   * Expecting exit code: 0

Become a member to submit

Solution

***Using the Bootdev CLI***

*The Bootdev CLI is the only way to submit your solution for this type of lesson. We need to be able to run commands in your environment to verify your solution.*

*You can*[*install it here*](https://github.com/bootdotdev/bootdev)*. It's a Go program hosted on GitHub, so you'll need Go installed as well. Instructions are on the GitHub page.*

# SQLC

[SQLC](https://sqlc.dev/) is an amazing Go program that generates Go code from SQL queries. It's not exactly an [ORM](https://www.freecodecamp.org/news/what-is-an-orm-the-meaning-of-object-relational-mapping-database-tools/), but rather a tool that makes working with raw SQL easy and type-safe.

We will be using Goose to manage our database migrations (the schema). We'll be using SQLC to generate Go code that our application can use to interact with the database (run queries).

## Assignment

1. Install SQLC.

SQLC is just a command line tool, it's not a package that we need to import. I recommend [installing](https://docs.sqlc.dev/en/latest/overview/install.html) it using go install. Installing Go CLI tools with go install is easy and ensures compatibility with your Go environment.

go install github.com/sqlc-dev/sqlc/cmd/sqlc@latest

Then run sqlc version to make sure it's installed correctly.

1. Configure [SQLC](https://docs.sqlc.dev/en/latest/tutorials/getting-started-postgresql.html). You'll always run the sqlc command from the root of your project. Create a file called sqlc.yaml in the root of your project. Here is mine:

version: "2"

sql:

- schema: "sql/schema"

queries: "sql/queries"

engine: "postgresql"

gen:

go:

out: "internal/database"

We're telling SQLC to look in the sql/schema directory for our schema structure (which is the same set of files that Goose uses, but sqlc automatically ignores "down" migrations), and in the sql/queries directory for queries. We're also telling it to generate Go code in the internal/database directory.

1. Write a query to create a user. Inside the sql/queries directory, create a file called users.sql. Here's the format:

-- name: CreateUser :one

INSERT INTO users (id, created\_at, updated\_at, email)

VALUES (

...

)

RETURNING \*;

* We'll be using UUIDs for ID values, so you can use [gen\_random\_uuid()](https://www.postgresql.org/docs/current/functions-uuid.html) to generate a new UUID.
* The created\_at and updated\_at fields should be set to the current timestamp. In Postgres, you can use NOW() to get the current timestamp.
* The email should be passed in by our application. Use $1 to represent the first parameter passed into the query. (in future queries, we'll use $2, $3, etc. for additional parameters)

The :one at the end of the query name tells SQLC that we expect to get back a single row (the created user).

Keep the [SQLC postgres docs](https://docs.sqlc.dev/en/latest/tutorials/getting-started-postgresql.html) handy, you'll probably need to refer to them again later.

1. Generate the Go code. Run sqlc generate from the root of your project. It should create a new package of go code in internal/database. You'll notice that the generated code relies on Google's [uuid](https://pkg.go.dev/github.com/google/uuid) package, so you'll need to add that to your module:

go get github.com/google/uuid

1. Import a PostgreSQL driver.

We need to add and import a [Postgres driver](https://github.com/lib/pq) so our program knows how to talk to the database. Install it in your module:

go get github.com/lib/pq

Add this import to the top of your main.go file:

import \_ "github.com/lib/pq"

This is one of my least favorite things working with SQL in Go currently. You have to import the driver, but you don't use it directly anywhere in your code. The underscore tells Go that you're importing it for its side effects, not because you need to use it.

1. Create a [.env](https://dotenvx.com/docs/env-file) file in the root of your project:

DB\_URL="YOUR\_CONNECTION\_STRING\_HERE"

Add a query parameter to the end of the connection string to disable SSL, e.g. postgres://wagslane:@localhost:5432/chirpy?sslmode=disable. **Add it to your .gitignore file**. It's incredibly insecure to commit secret keys to a Git repo.

1. go get github.com/joho/godotenv, then call godotenv.Load() at the beginning of your main() function to load the .env file into your environment variables. Then you can use [os.Getenv](https://pkg.go.dev/os#Getenv) to get the DB\_URL from the environment:

dbURL := os.Getenv("DB\_URL")

1. Next, [sql.Open()](https://pkg.go.dev/database/sql#Open) a connection to your database:

db, err := sql.Open("postgres", dbURL)

Use your SQLC generated database package to create a new \*database.Queries, and store it in your apiConfig struct so that handlers can access it:

dbQueries := database.New(db)

**With the .env file in place, your program successfully compiling, and your database connection set up, run the tests.**

**Run**

bootdev run e5bddf3d-d96b-487e-97e6-7a5aa06b1ee1

**Submit**

bootdev run e5bddf3d-d96b-487e-97e6-7a5aa06b1ee1 -s

Run the CLI commands to test your solution.

1. **cat .env**
   * Expecting exit code: 0
   * Expecting stdout to contain all of:
     + DB\_URL
     + =
     + postgres://
     + sslmode=disable
2. **go build**
   * Expecting exit code: 0

Become a member to submit

Solution

***Using the Bootdev CLI***

*The Bootdev CLI is the only way to submit your solution for this type of lesson. We need to be able to run commands in your environment to verify your solution.*

*You can*[*install it here*](https://github.com/bootdotdev/bootdev)*. It's a Go program hosted on GitHub, so you'll need Go installed as well. Instructions are on the GitHub page.*

# Database Review

It's very standard to use database software to store web server data on disk. Sometimes that database runs on the same host machine as your server (like we're doing on your local machine), but it's also common to have a separate database server that your server connects to over the network.

That's why we use a connection URL: it can point to a local database or a remote one.

## Popular databases

You don't need to know about all of these, but you might be curious about some of the database technologies out there. Here are a few of the most popular ones:

* [PostgreSQL](https://www.postgresql.org/): A fantastic open-source SQL database.
* [MySQL](https://www.mysql.com/): Another open-source SQL database. Less fantastic IMO.
* [MongoDB](https://www.mongodb.com/): A popular open-source NoSQL document database.
* [Firebase](https://firebase.google.com/): A popular cloud-based NoSQL database service.
* [SQLite](https://www.sqlite.org/index.html): A popular embedded SQL database.

Feel free to browse [DB Engine](https://db-engines.com/en/ranking) if you want to dive deeper into the world of database technologies.

# Create User

We've written the SQL query, now it's time to write the API handler that will allow users to create a new user.

## The Context Package

The context package is a part of Go's standard library. It does several things, but the most important thing is that it handles timeouts. All of SQLC's database queries accept a [context.Context](https://pkg.go.dev/context#Context) as their first argument:

user, err := cfg.db.CreateUser(r.Context(), params.Email)

By passing your handler's [http.Request.Context()](https://pkg.go.dev/net/http#Request.Context) to the query, the library will automatically cancel the database query if the HTTP request is canceled or times out.

The benefit is that it will save your server from getting bogged down by long-running queries!

## Assignment

1. Add a new endpoint to your server POST /api/users that allows users to be created. It accepts an email as JSON in the request body and returns the user's ID, email, and timestamps in the response body.

**Request**:

{

"email": "user@example.com"

}

**Response**:

HTTP 201 Created

{

"id": "50746277-23c6-4d85-a890-564c0044c2fb",

"created\_at": "2021-07-07T00:00:00Z",

"updated\_at": "2021-07-07T00:00:00Z",

"email": "user@example.com"

}

1. Update the POST /admin/reset endpoint to delete all users in the database (but don't mess with the schema). You'll need a new SQLC query for this. Add a new value to your .env file called PLATFORM and set it equal to "dev". Read it into your apiConfig. If PLATFORM is not equal to "dev", this endpoint should return a 403 Forbidden. This ensures that this extremely dangerous endpoint can only be accessed in a local development environment.

**Run and submit the HTTP tests using the CLI tool**.

## Tips

I created a User struct in my main package. When the database package returns a database.User, I map it to my main package's User struct before marshalling it to JSON so that I can control the JSON keys:

type User struct {

ID uuid.UUID `json:"id"`

CreatedAt time.Time `json:"created\_at"`

UpdatedAt time.Time `json:"updated\_at"`

Email string `json:"email"`

}

**Struggling?** I, Boots the Bane of End-Users, am happy to help! Ask me and I'll answer for a price. If you find my explanations suspect, ask a human for help in [the Discord](https://boot.dev/community)!



Copy/paste one of the following commands into your terminal:

**Run**

bootdev run 0820daf4-4006-425a-a50c-f45c0eb97d06

**Submit**

bootdev run 0820daf4-4006-425a-a50c-f45c0eb97d06 -s

* Default base URL: http://localhost:8080
* Optionally use the -b flag to override the default baseURL

Run the CLI commands to test your solution.

1. 1. **POST /admin/reset**
   * 1.

Expecting status code: 200

1. 2. **POST /api/users**

**Request Body:**

{

"email": "mloneusk@example.co"

}

* + 1.

Expecting status code: 201

* + 2.

Expecting body to contain: id

* + 3.

Expecting body to contain: created\_at

* + 4.

Expecting body to contain: updated\_at

* + 5.

Expecting JSON at

.email

 to be equal to

mloneusk@example.co

1. 3. **POST /api/users**

**Request Body:**

{

"email": "dackjorsey@example.co"

}

* + 1.

Expecting status code: 201

* + 2.

Expecting body to contain: id

* + 3.

Expecting body to contain: created\_at

* + 4.

Expecting body to contain: updated\_at

* + 5.

Expecting JSON at

.email

 to be equal to

dackjorsey@example.co

Become a member to submit

Solution

***Using the Bootdev CLI***

*The Bootdev CLI is the only way to submit your solution for this type of lesson. We need to be able to make HTTP requests outside of the confines of the browser.*

*You can*[*install it here*](https://github.com/bootdotdev/bootdev)*. It's a Go program hosted on GitHub, so you'll need Go installed as well. Instructions are on the GitHub page.*

# Create Chirp

Our API needs to support standard CRUD operations for "chirps". A "chirp" is just a short message that a user can post to the API, like a tweet.

## Assignment

1. Add a POST /api/chirps handler. It accepts a JSON payload with a body field:

{

"body": "Hello, world!",

"user\_id": "123e4567-e89b-12d3-a456-426614174000"

}

Delete the /api/validate\_chirp endpoint that we created before, but port all that logic into this one. Users should not be allowed to create invalid chirps!

1. If the chirp is valid, you should save it in the database with:
   * A new random id: A UUID
   * created\_at: A non-null timestamp
   * updated\_at: A non null timestamp
   * body: A non-null string
   * user\_id: This should [reference](https://en.wikipedia.org/wiki/Foreign_key) the id of the user who created the chirp, and [ON DELETE CASCADE](https://www.postgresql.org/docs/9.2/ddl-constraints.html), which will cause a user's chirps to be deleted if the user is deleted.

You'll need a new up/down migration for this table.

As a general rule it's always a good idea to use created\_at and updated\_at timestamps for all your resources. It gives you a nice audit trail and makes it easier to debug issues.

1. If creating the record goes well, respond with a 201 status code and the full chirp resource:

{

"id": "94b7e44c-3604-42e3-bef7-ebfcc3efff8f",

"created\_at": "2021-01-01T00:00:00Z",

"updated\_at": "2021-01-01T00:00:00Z",

"body": "Hello, world!",

"user\_id": "123e4567-e89b-12d3-a456-426614174000"

}

Yes, this isn't secure because it means any user can create a chirp on behalf of any other user. We'll fix that in a future assignment.

**Run and submit the CLI tests.**

**Struggling?** I, Boots the Fearless Friday Deployer, am happy to help! Ask me and I'll answer for a price. If you find my explanations suspect, ask a human for help in [the Discord](https://boot.dev/community)!



Copy/paste one of the following commands into your terminal:

**Run**

bootdev run acf357f0-ec20-429f-99ec-265963c2fedf

**Submit**

bootdev run acf357f0-ec20-429f-99ec-265963c2fedf -s

* Default base URL: http://localhost:8080
* Optionally use the -b flag to override the default baseURL

Run the CLI commands to test your solution.

1. 1. **POST /admin/reset**
   * 1.

Expecting status code: 200

1. 2. **POST /api/users**

**Request Body:**

{

"email": "saul@bettercall.com"

}

* + 1.

Expecting status code: 201

* + 2.

Expecting JSON at

.email

 to be equal to

saul@bettercall.com

1. 3. **POST /api/chirps**

**Request Body:**

{

"body": "If you're committed enough, you can make any story work.",

"user\_id": "${userID1}"

}

* + 1.

Expecting status code: 201

* + 2.

Expecting JSON at

.body

 to be equal to

If you're committed enough, you can make any story work.

* + 3.

Expecting JSON at

.user\_id

 to be equal to

${userID1}

Become a member to submit

Solution

***Using the Bootdev CLI***

*The Bootdev CLI is the only way to submit your solution for this type of lesson. We need to be able to make HTTP requests outside of the confines of the browser.*

*You can*[*install it here*](https://github.com/bootdotdev/bootdev)*. It's a Go program hosted on GitHub, so you'll need Go installed as well. Instructions are on the GitHub page.*

Run the CLI commands to test your solution.

1. 1. **POST /admin/reset**
   * 1.

Expecting status code: 200

1. 2. **POST /api/users**

**Request Body:**

{

"email": "saul@bettercall.com"

}

* + 1.

Expecting status code: 201

* + 2.

Expecting JSON at

.email

 to be equal to

saul@bettercall.com

1. 3. **POST /api/chirps**

**Request Body:**

{

"body": "If you're committed enough, you can make any story work.",

"user\_id": "${userID1}"

}

* + 1.

Expecting status code: 201

* + 2.

Expecting JSON at

.body

 to be equal to

If you're committed enough, you can make any story work.

* + 3.

Expecting JSON at

.user\_id

 to be equal to

${userID1}

Become a member to submit

ntro

as a hiring manager for back-end

Engineers one of my favorite things to

ask candidates was how they would design

an API API design is so critical to get

right the first time if you write some

code in a style that you're not happy

with but at least it's internal to your

application you can fix it fairly easily

but if you design an API that other

people are using and later want to

change it it's much harder to make those

changes because you have clients that

are now depending on the API that you're

not happy with in this video we're going

to break down how to write clean API

endpoints specifically the HTTP paths to

those endpoints and all these tips will

apply to kind of a restful API or an API

that follows restful conventions restful

RESTful API's

apis model the world as resources so say

for example you were building YouTube as

a restful API you might have resources

like users videos and comments and for

each of those three resources you need a

way for clients of your API to create

read update and delete those resources

that's why we often refer to restful

apis as crud apis

Broad apis in the context of a restful

GET Endpoint in RESTful API's

API an endpoint is just a path plus

method combination that allows us to

interact with a resource so for example

let's say we have the videos resource if

we want to get a list of all videos on

the server we might have an endpoint

that uses the get HTTP method

and the slash videos

slash videos path

means we're going to be sending an HTTP

get request to the domain of the API

slash videos and we'll expect to get

back kind of a list of all of the videos

that are stored on the server so we'd

expect this endpoint to kind of give us

all all of the videos

so how would we go about getting just

one specific video right maybe we don't

want to paginate through all of the

videos on YouTube but we want to get a

very specific video by its ID well the

way we would do that in kind of a

conventional way

is again get request again because we're

just getting a resource we're not

changing it in any way we're going to

use that same slash videos path

but this time we're going to append

a dynamic ID to the end of the path so

for example to get the video with idwe

would do slash videos slash one or to

get video with idwe would do slash

videos slash two so to draw that out for

you we would make a get request to slash

videos

slash two if we wanted to get video with

idnow one mistake that I've seen

people make here is when making the

endpoint to get all the videos people

properly use the kind of videos now

but when creating the endpoint to get a

single video sometimes people will drop

the S and make it slash video slash ID

and this is not recommended if you're

trying to build a restful server you

should definitely keep the path the same

and just append an ID to the end now to

be clear it will work either way right

and as long as your documentation

reflects how your server actually works

most people are going to be able to

figure out how to use your API but these

conventions are in place for a reason

and most people expect the videos

endpoint that returns a single video in

the video's endpoint that returns all of

the videos to have the same kind of

pluralized form of the noun so how would

POST Endpoint in RESTful API's

we structure the path when we want to

create a video well when we want to

create a video we'll probably use the

post HTTP method and we'll definitely be

using that same slash videos

part of the path the question is here

and I want to pause for a second so you

can take a second to think about it are

we going to provide an ID in the path

when we create a new video or are we not

the answer is that we should not even

though we're creating a single video

we're still just going to use this slash

videos endpoint without an ID and the

reason being that it's the server's job

to create IDs when we upload a new video

we don't tell the server what the ID

should be we don't have that knowledge

right the server knows how it generates

IDs and it manages the creation of those

IDs for example it needs to make sure

that the IDS are unique right so that's

something as the client that we can't

really know we just upload raw video

content and we would expect in the

response that comes back on that rep on

that post request that we would get a

kind of access to that ID so again when

you're creating new resources you

generally do not specify an ID in the

UPDATE (PUT) Endpoint in RESTful API's

path so how we go about updating a

resource well updating is typically done

with the put method put HTTP method

sometimes the patch method

um but typically put and we're going to

make that request to the slash videos

endpoint again

in this case we do want to pass an ID

and that's because the server needs to

know which resource we want to update

right so we have a specific video let's

say and we want to change the title well

we would make a put request to the

endpoint with of the ID of that video

and pass in a new title probably in the

HTTP request body last but not least

DELETE Endpoint in RESTful API's

we've got the delete endpoint

which uses the delete HTTP method

now it is very rare to see a delete

endpoint that can delete all resources

so delete slash videos

Almost doesn't even make sense right

that kind of is implying that we have

the ability to delete every video in the

system so almost always a delete

endpoint will end with an ID so you can

delete a very specific endpoint right I

uploaded a video to YouTube I need to go

delete that video there's no button in

YouTube to delete all videos on the

platform that doesn't really make sense

Additional HTTP Path conventions

before we wrap up the last thing I want

to mention is that really this only

applies to the end of the path so you'll

never really see additional data kind of

out here on the end of the path right

things like videos

name like that that doesn't happen this

like when we were talking about this

convention we were talking about the

convention for the end of the HTTP path

that said sometimes the beginning of the

path does change so for example let's

say we wanted to name space our API

based on the version of the API we might

do slash Vvideos so that in the future

if we wanted to change the way the API

works we could swap that out for slash

Vslash videos right so the beginning

of the path can change a little bit but

the end of the path cannot the

convention really kind of strictly

applies to the end of the HTTP path

Outro

hopefully this explanation has helped

again these are often the kinds of

questions that I would ask to kind of

June your back end Engineers when I'm

hiring for a back-end developer position

so if you're in the market you should

definitely be familiar with this kind of

high level API design stuff if you

enjoyed the video please be sure to like

And subscribe it helps with the

algorithm and I'll see you in the next

one

# Collections and Singletons

We're building a fairly [RESTful API](https://restfulapi.net/).

REST is a set of guidelines for how to build APIs. It's not a standard, but it's a set of conventions that many people follow. Not all back-end APIs are RESTful, but many are. As a back-end developer, you'll need to know how to build RESTful APIs.

## Collections and Singletons

In REST, it's conventional to name all of your endpoints after the resource that they represent and for the name to be plural. That's why we use POST /api/chirps to create a new chirp instead of POST /api/chirp.

To get a collection of resources it's conventional to use a GET request to the plural name of the resource. So we are going to use GET /api/chirps to get all of the chirps.

To get a singleton, or a single instance of a resource, it's conventional to use a GET request to the plural name of the resource, followed by the ID of the resource. So we are going to use GET /api/chirps/94b7e44c-3604-42e3-bef7-ebfcc3efff8f to get the chirp with ID 94b7e44c-3604-42e3-bef7-ebfcc3efff8f.

**Struggling?** I, Boots the Master of Mondays, am happy to help! Ask me and I'll answer for a price. If you find my explanations suspect, ask a human for help in [the Discord](https://boot.dev/community)!



Which is the RESTful way to design an endpoint that returns a single user resource?

GET /users/{id}

GET /users

GET /user/{id}

GET /user

Become a member for quiz access

# Get All Chirps

We need a way to retrieve all the chirps from the database. Later, we'll add sorting and filtering functionality, but you can think of this as a very basic version of an endpoint that might serve a timeline of chirps.

## Assignment

1. Add a new query that retrives all chirps in ascending order by created\_at.
2. Add a GET /api/chirps endpoint that returns all chirps in the database. It should return them in the same structure as the POST /api/chirps endpoint, but as an array. Use a 200 status code for success. Order them by created\_at in ascending order.

[

{

"id": "94b7e44c-3604-42e3-bef7-ebfcc3efff8f",

"created\_at": "2021-01-01T00:00:00Z",

"updated\_at": "2021-01-01T00:00:00Z",

"body": "Yo fam this feast is lit ong",

"user\_id": "123e4567-e89b-12d3-a456-426614174000"

},

{

"id": "f0f87ec2-a8b5-48cc-b66a-a85ce7c7b862",

"created\_at": "2022-01-01T00:00:00Z",

"updated\_at": "2023-01-01T00:00:00Z",

"body": "What's good king?",

"user\_id": "123e4567-e89b-12d3-a456-426614174000"

}

]

**Run and submit the CLI tests.**

**Struggling?** I, Boots the King of the Keyboard, am happy to help! Ask me and I'll answer for a price. If you find my explanations suspect, ask a human for help in [the Discord](https://boot.dev/community)!



Copy/paste one of the following commands into your terminal:

**Run**

bootdev run 341b80d4-556f-4c5b-8afc-ffd12d5238c2

**Submit**

bootdev run 341b80d4-556f-4c5b-8afc-ffd12d5238c2 -s

* Default base URL: http://localhost:8080
* Optionally use the -b flag to override the default baseURL

Run the CLI commands to test your solution.

1. 1. **POST /admin/reset**
   * 1.

Expecting status code: 200

1. 2. **POST /api/users**

**Request Body:**

{

"email": "saul@bettercall.com"

}

* + 1.

Expecting status code: 201

1. 3. **POST /api/chirps**

**Request Body:**

{

"body": "If you're committed enough, you can make any story work.",

"user\_id": "${userID1}"

}

* + 1.

Expecting status code: 201

1. 4. **POST /api/chirps**

**Request Body:**

{

"body": "I once told a woman I was Kevin Costner, and it worked because I believed it.",

"user\_id": "${userID1}"

}

* + 1.

Expecting status code: 201

1. 5. **GET /api/chirps**
   * 1.

Expecting status code: 200

* + 2.

Expecting JSON at

.[0].body

 to be equal to

If you're committed enough, you can make any story work.

* + 3.

Expecting JSON at

.[1].body

 to be equal to

I once told a woman I was Kevin Costner, and it worked because I believed it.

Become a member to submit

Solution

***Using the Bootdev CLI***

*The Bootdev CLI is the only way to submit your solution for this type of lesson. We need to be able to make HTTP requests outside of the confines of the browser.*

*You can*[*install it here*](https://github.com/bootdotdev/bootdev)*. It's a Go program hosted on GitHub, so you'll need Go installed as well. Instructions are on the GitHub page.*

# Get Chirp

Now we need a way to lookup a single chirp by its ID. You might be thinking:

"If I can get all of the chirps, why do I need a way to get just one?"

Imagine there are 10,000 chirps in the database - no, imagine 10,000,000,000! We'll obviously need to change our GET /api/chirps endpoint to only return a subset of chirps at a time.

However, our users will still need a way to view a single chirp - for example, maybe they have a link directly to it.

## Assignment

1. Add a GET /api/chirps/{chirpID} endpoint that returns a single chirp by its ID. The chirp ID will be passed in as a path parameter. For example:

GET /api/chirps/94b7e44c-3604-42e3-bef7-ebfcc3efff8f

You can get the string value of the path parameter like in Go with the [http.Request.PathValue](https://pkg.go.dev/net/http#Request.PathValue) method.

1. If the chirp is found, return it like so with a 200 code:

{

"id": "94b7e44c-3604-42e3-bef7-ebfcc3efff8f",

"created\_at": "2021-01-01T00:00:00Z",

"updated\_at": "2021-01-01T00:00:00Z",

"body": "fr? no clowning?",

"user\_id": "123e4567-e89b-12d3-a456-426614174000"

}

1. Otherwise, return a 404.

**Run and submit the CLI tests.**

**Struggling?** I, Boots the Master of Code and Casting, am happy to help! Ask me and I'll answer for a price. If you find my explanations suspect, ask a human for help in [the Discord](https://boot.dev/community)!



Copy/paste one of the following commands into your terminal:

**Run**

bootdev run 0a07a4a3-c11f-429f-ac70-52fa2e016bc0

**Submit**

bootdev run 0a07a4a3-c11f-429f-ac70-52fa2e016bc0 -s

* Default base URL: http://localhost:8080
* Optionally use the -b flag to override the default baseURL

Run the CLI commands to test your solution.

1. 1. **POST /admin/reset**
   * 1.

Expecting status code: 200

1. 2. **POST /api/users**

**Request Body:**

{

"email": "saul@bettercall.com"

}

* + 1.

Expecting status code: 201

1. 3. **POST /api/chirps**

**Request Body:**

{

"body": "I'm gonna be a damn good developer, and people are gonna know about it.",

"user\_id": "${userID1}"

}

* + 1.

Expecting status code: 201

1. 4. **GET /api/chirps/${chirpID1}**
   * 1.

Expecting status code: 200

* + 2.

Expecting JSON at

.body

 to be equal to

I'm gonna be a damn good developer, and people are gonna know about it.

Become a member to submit

Solution

***Using the Bootdev CLI***

*The Bootdev CLI is the only way to submit your solution for this type of lesson. We need to be able to make HTTP requests outside of the confines of the browser.*

*You can*[*install it here*](https://github.com/bootdotdev/bootdev)*. It's a Go program hosted on GitHub, so you'll need Go installed as well. Instructions are on the GitHub page.*

# Authentication with Passwords

Authentication is the process of verifying who a user is. If you don't have a secure authentication system, your back-end systems will be open to attack!

Imagine if I could make an HTTP request to the YouTube API and upload a video to your channel. YouTube's authentication system prevents this from happening by verifying that I am who I say I am.

## Passwords

Passwords are a common way to authenticate users. You know how they work: When a user signs up for a new account, they choose a password. When they log in, they enter their password again. The server will then compare the password they entered with the password that was stored in the database.

There are 2 really important things to consider when storing passwords:

1. **Storing passwords in plain text is awful.** If someone gets access to your database, they will be able to see all of your users' passwords. If you store passwords in plain text, you are giving away your users' passwords to anyone who gets access to your database.
2. **Password strength matters.** If you allow users to choose weak passwords, they will be more likely to reuse the same password on other websites. If someone gets access to your database, they will be able to log in to your users' other accounts.

We won't be writing code to validate password strength in this course, but you get the idea: you can enforce rules in your HTTP handlers to make sure passwords are of a certain length and complexity.

## Hashing

On the other hand, we will be writing code to store passwords in a way that prevents them from being read by anyone who gets access to your database. This is called hashing. Hashing is a one-way function. It takes a string as input and produces a string as output. The output string is called a hash.

We'll cover how hashing works in-depth in a later course. For now, just know that hashing is a way to store passwords in a way that prevents them from being read by anyone who gets access to your database, but still allows us to compare passwords when a user logs in.

## Assignment

1. Add and run a new migration that adds a non-null TEXT column to the users table called hashed\_password. It should default to "unset" for existing users.
2. Create an internal/auth package and expose two functions:
   * func HashPassword(password string) (string, error): Hash the password using the [bcrypt.GenerateFromPassword](https://pkg.go.dev/golang.org/x/crypto/bcrypt#GenerateFromPassword) function. [Bcrypt](https://blog.boot.dev/cryptography/bcrypt-step-by-step/) is a secure hash function that is intended for use with passwords.
   * func CheckPasswordHash(password, hash string) error: Use the [bcrypt.CompareHashAndPassword](https://pkg.go.dev/golang.org/x/crypto/bcrypt#CompareHashAndPassword) function to compare the password that the user entered in the HTTP request with the password that is stored in the database.

I wrote a couple of simple [unit tests](https://go.dev/doc/tutorial/add-a-test) to ensure the package is working as expected.

1. Update the POST /api/users endpoint. The body parameters should now require a new password field:

{

"password": "04234",

"email": "lane@example.com"

}

Note: As long as your server uses HTTPS in production, it's safe to send raw passwords in HTTP requests, because the entire request is encrypted.

Use your internal package's HashPassword function to hash the password before storing it in the database. Do **NOT** return the hashed password in the response. Again, that would be a security risk.

1. Add a POST /api/login endpoint. This endpoint should allow a user to login. In a future exercise, this endpoint will be used to give the user a token that they can use to make authenticated requests. For now, let's just make sure password validation is working. It should accept this body:

{

"password": "04234",

"email": "lane@example.com"

}

You'll need a new query to look up a user by their email address (you don't have access to an ID here). Once you have the user, check to see if their password matches the stored hash using your internal package. If either the user lookup or the password comparison errors, just return a 401 Unauthorized response with the message "Incorrect email or password".

If the passwords match, return a 200 OK response and a copy of the user resource (without the password of course):

{

"id": "f0f87ec2-a8b5-48cc-b66a-a85ce7c7b862",

"created\_at": "2021-07-07T00:00:00Z",

"updated\_at": "2021-07-07T00:00:00Z",

"email": "lane@example.com"

}

**Struggling?** I, Boots the Magnificent, am happy to help! Ask me and I'll answer for a price. If you find my explanations suspect, ask a human for help in [the Discord](https://boot.dev/community)!



Copy/paste one of the following commands into your terminal:

**Run**

bootdev run 294e5c16-d1e8-4836-871c-dedc98581236

**Submit**

bootdev run 294e5c16-d1e8-4836-871c-dedc98581236 -s

* Default base URL: http://localhost:8080
* Optionally use the -b flag to override the default baseURL

Run the CLI commands to test your solution.

1. 1. **POST /admin/reset**
   * 1.

Expecting status code: 200

1. 2. **POST /api/users**

**Request Body:**

{

"email": "saul@bettercall.com",

"password": "123456"

}

* + 1.

Expecting status code: 201

* + 2.

Expecting body to contain: id

* + 3.

Expecting body to contain: created\_at

* + 4.

Expecting body to contain: updated\_at

* + 5.

Expecting JSON at

.email

 to be equal to

saul@bettercall.com

1. 3. **POST /api/login**

**Request Body:**

{

"email": "saul@bettercall.com",

"password": "123456"

}

* + 1.

Expecting status code: 200

* + 2.

Expecting body to contain: id

* + 3.

Expecting body to contain: created\_at

* + 4.

Expecting body to contain: updated\_at

* + 5.

Expecting JSON at

.email

 to be equal to

saul@bettercall.com

1. 4. **POST /api/login**

**Request Body:**

{

"email": "saul@bettercall.com",

"password": "000011112222"

}

* + 1.

Expecting status code: 401

Become a member to submit

Solution

***Using the Bootdev CLI***

*The Bootdev CLI is the only way to submit your solution for this type of lesson. We need to be able to make HTTP requests outside of the confines of the browser.*

*You can*[*install it here*](https://github.com/bootdotdev/bootdev)*. It's a Go program hosted on GitHub, so you'll need Go installed as well. Instructions are on the GitHub page.*

# Password Review

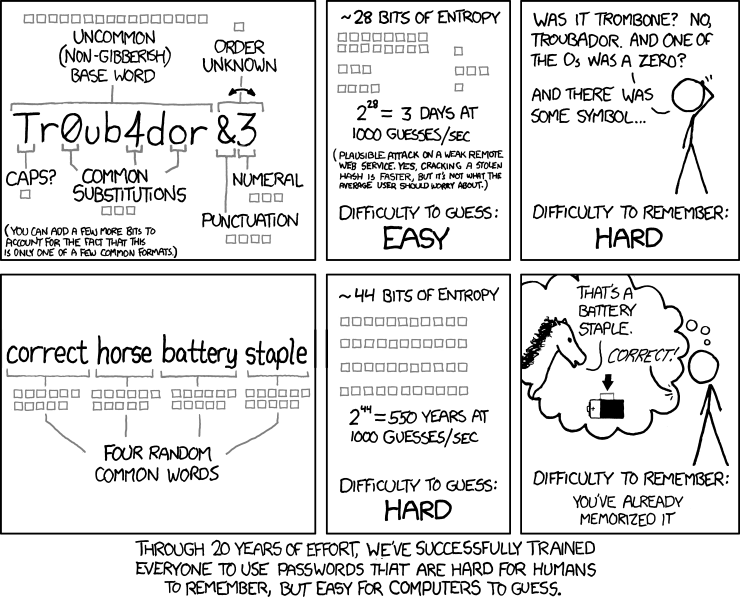
It's a really bad idea for users to reuse the same passwords across sites. If someone figures out their password for one site, they can try it on other sites. If they get lucky, they can log in to and compromise many of their accounts.

Unfortunately, it's very common for users to reuse passwords. We can't force users to not reuse passwords on the server side, but we can take steps to make it harder for them to reuse passwords. Namely, we can require that passwords are strong.

## Passwords should be strong

The most important factor for the strength of a password is its entropy. [Entropy](https://blog.boot.dev/cryptography/what-is-entropy-in-cryptography/) is a measure of how many possible combinations of characters there are in a string. To put it simply:

* The longer the password the better
* Special characters and capitals should always be allowed
* Special characters and capitals aren't as important as length



* [xkcd: Password Strength](https://xkcd.com/936/)

## Passwords should never be stored in plain text

The most critical thing we can do to protect our users' passwords is to never store them in plain text. We should use cryptographically strong key derivation functions (which are a special class of hash functions) to store passwords in a way that prevents them from being read by anyone who gets access to your database.

[Bcrypt](https://blog.boot.dev/cryptography/bcrypt-step-by-step/) is a great choice. [SHA-256](https://blog.boot.dev/cryptography/how-sha-2-works-step-by-step-sha-256/) and [MD5](https://en.wikipedia.org/wiki/MD5) are not.

Which is a good hash function for passwords?

SHA-256

MD5

Bcrypt

Become a member for quiz access

# Password Review

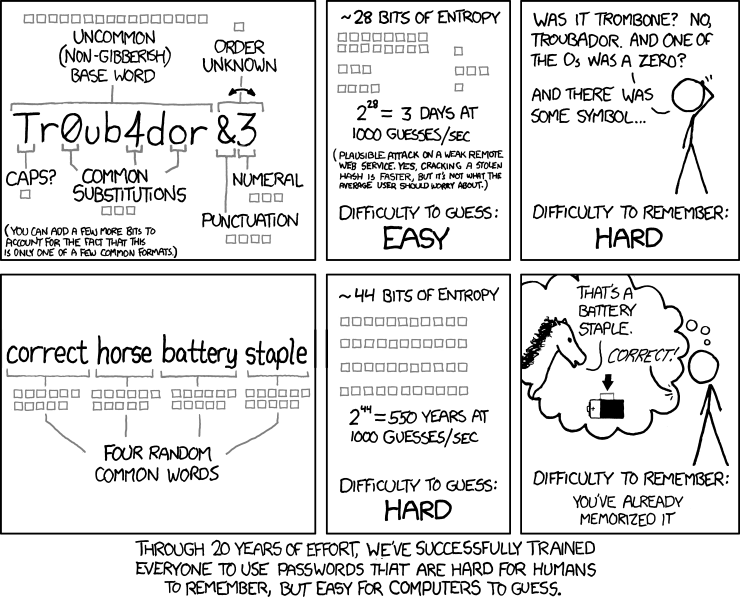
It's a really bad idea for users to reuse the same passwords across sites. If someone figures out their password for one site, they can try it on other sites. If they get lucky, they can log in to and compromise many of their accounts.

Unfortunately, it's very common for users to reuse passwords. We can't force users to not reuse passwords on the server side, but we can take steps to make it harder for them to reuse passwords. Namely, we can require that passwords are strong.

## Passwords should be strong

The most important factor for the strength of a password is its entropy. [Entropy](https://blog.boot.dev/cryptography/what-is-entropy-in-cryptography/) is a measure of how many possible combinations of characters there are in a string. To put it simply:

* The longer the password the better
* Special characters and capitals should always be allowed
* Special characters and capitals aren't as important as length



* [xkcd: Password Strength](https://xkcd.com/936/)

## Passwords should never be stored in plain text

The most critical thing we can do to protect our users' passwords is to never store them in plain text. We should use cryptographically strong key derivation functions (which are a special class of hash functions) to store passwords in a way that prevents them from being read by anyone who gets access to your database.

[Bcrypt](https://blog.boot.dev/cryptography/bcrypt-step-by-step/) is a great choice. [SHA-256](https://blog.boot.dev/cryptography/how-sha-2-works-step-by-step-sha-256/) and [MD5](https://en.wikipedia.org/wiki/MD5) are not.

What is the best measure of password strength?

Whether or not we also store the user's mother's maiden name

Whether or not special characters are used

Whether or not capitals are used

Length

Become a member for quiz access

Intro

passwords API Keys magic links login

with Google there are so many ways to

authenticate with a web service in this

video I'll break down kind of the pros

and cons of each of those different

authentication mechanisms and we'll talk

about which one you should use in your

What is authentication?

next project before we get ahead of

ourselves what is authentication

authentication is just the process of

verifying that a user is who they say

they are without authentication anyone

could log on to Twitter and tweet as me

because Twitter would have no way of

differentiating who all of the different

users are and which accounts they're

allowed to tweet from so passwords are

Password based auth

kind of the most basic form of

authentication I'm not going to explain

how passwords work I'm sure you're

familiar with passwords but what I will

say is the passwords are a bit of a

double-edged sword passwords are

something that users expect right

they're used everywhere and it's

certainly a viable system of

authentication the problem is users

often don't follow best security

practices right many users still don't

use a password manager many users are

choosing weak passwords or reusing

passwords across different web services

not only that users frequently forget

their passwords so you also need to have

a separate authentication system so that

when a user forgets their password they

have a way to reset it this is usually

done by sending a recovery email I'm not

saying that you can't build a

password-based authentication system you

definitely can just make sure that at

the very least you're requiring strong

passwords and if you're running a a

service that's more security critical

you might even want to look into

two-factor authentication as an

additional layer of security next up is

Magic links

magic links so you may have encountered

a system that uses magic links they're

definitely not as popular as passwords

yet but they kind of rely on the same

security mechanisms so when you build a

password-based authentication system

inherently you're almost always also

trusting the user's email because if the

user forgets their password they can

typically recover or reset their

password by having an email sent to them

and obviously if their email isn't

secure then the password system also

isn't secure magic links basically just

take this idea of your email as your

authentication system to the next level

in password-based authentication flow

you would go to a website and type in an

email and a password right and then

you'd be authenticated if the password

matches the one that you typed in when

you first created the account with magic

links you just type in an email and then

you'd click a button that says send me a

magic link and then in your inbox an

email shows up with a specialized link

so the link would look something like

this be https colon slash example.com

right so this is just the domain of the

website that you're logging into but the

first query parameter on the list might

look something like this token equals

and then there's going to be a token

which it's kind of just this long giant

string of seemingly garbage letters and

numbers right and within that encoded

token would actually be some user

information so it'd probably to be my

user ID right so when I clicked the when

I clicked the link on the website the

server generated a token just for me

emailed it to me and now because I'm the

only one with access to my inbox I can

just click the link navigate back to the

website and the website can tell hey

this is Lane's token we should log him

in because he clearly owns this email

address if you're familiar with Json web

tokens or jwts that token could very

well just be a JWT so when should you

use magic links well to be honest I use

magic links on boot.dev and the main

reason being I just don't want to deal

with password recovery and storing users

passwords and all the kind of security

that comes with it magic links are

easier to implement and get started with

the big drawback to magic links is that

some users just don't like them right

people who are good password users

people that have password managers that

take care of all their passwords for

them might not appreciate the fact that

you require them to click into their

inbox every time they log in

it's not as much of a problem if you

have long session durations right so if

you don't automatically log people out

of your site say every day then maybe

magic links would be a good fit but if

you have people logging in every day it

might be kind of cumbersome to be

clicking into your email again every day

to log back into the website next let's

talk about third-party authentication so

rd party auth (login with google)

you almost certainly have have tried

this or at least seen it around this is

stuff like log in with Google log in

with GitHub log in with Facebook the

idea is the website that you're logging

into is kind of offloading or

Outsourcing their authentication to a

third-party service usually a tech giant

that almost certainly the user already

has an account with as a user I love

third-party authentication I hate

creating new passwords anytime I can log

in with Google or GitHub I usually do

there are some huge drawbacks though

there are very few websites out there

that can get away with only allowing

users to log in through a third party

and the reasons are pretty obvious what

if the user doesn't have an account with

a third party right what if they don't

have a GitHub account or a Google

account right they need another option

to be able to log in so it's very

frequently that you'll see kind of a

combo of login with Google or log in

with GitHub or use a magic link right or

log in with Google log in with GitHub or

use a password it's also just worth

pointing out that you're putting a lot

of trust in the third party right when

you as a website Implement login with

Google you have complete trust that

Google has actually done their job of

authentication well now I think that

that's usually a good assumption it's

probably unlikely that your little

startup or your side project has more

secure authentication than what Google's

doing with that in mind it's really

important to understand just how much

trust you're putting into third parties

those three options uh passwords magic

links and logging users in through a

third party are all great options for

websites and mobile applications but

sometimes you aren't authenticating

users through a UI or graphical user

CLI auth and api tokens

interface you're authenticating them

through a CLI or sometimes you're just

authenticating clients that are

themselves pieces of software running on

other servers somewhere so when you're

writing authentication that isn't

necessarily meant to be consumed by

humans in a UI but maybe it's again a

piece of software then API keys are

probably what you want and an API key is

just a long string of letters and

numbers that uniquely identifies a

client an API key might look something

like this it's basically just a long

string of letters and numbers but it

accomplishes two purposes so with

password-based authentication we had an

email which was the identifier and then

we had the password which kind of served

as the credentials an API key is both in

one because API keys are typically

generated by the server they can be

guaranteed to be unique so when you use

an API key the server knows exactly who

you are you don't even need to provide

an email address and this just makes API

Keys even better when you're working in

code because all you need to do is save

the API key somewhere in a configuration

file and you're good to go so if you're

building an authentication system maybe

you're writing a new back-end server and

you need a way to authenticate users

hopefully this video helped you

understand some of the different options

you have and you know when some of these

different options would be better than

others if you found this video useful

will please hit the like and the

Subscribe button it really helps out the

Channel

# Types of Authentication

Here are a few of the most common authentication methods you'll see in the wild:

1. Password + ID (username, email, etc.)
2. 3rd Party Authentication ("Sign in with Google", "Sign in with GitHub", etc)
3. Magic Links
4. API Keys

## 1. Password + ID

This is the most common type of authentication that requires a manual login from a user. When users use password managers, it's one of the more secure ways to authenticate users, unfortunately, many users don't, so it's not as secure as it could be.

That said, it's a valid choice.

## 2. 3rd Party Authentication

3rd party authentication is a way to authenticate users using a service like Google or GitHub (similarly to how we do it here on Boot.dev). 3rd party auth is great for user experience because it allows users to use their existing accounts to log in to your app, lowering friction.

It's also nice because you don't need to worry about storing passwords yourself, meaning you can outsource the security of your users' passwords to a company that, hopefully, does a good job.

The only real drawbacks to 3rd party auth is that you're trusting a 3rd party and if your users don't have an account with that 3rd party, they won't be able to log in.

## 3. Magic Links

Magic links are a way to authenticate users without a password. It relies on the assumption that the user's email is something that they have unique access to.

The webserver sends a link to the user's email and encodes a unique token in that link. When the user clicks the link, the webserver can decode the token and use it to authenticate the user. Eg:

https://example.com/login?token=...

## 4. API Keys

API keys are a fantastic way to authenticate users and systems programmatically. An API Key is just a long, secure string that uniquely identifies a user or system, and that can't be guessed. Because they're intended to be used in code, they don't need to be memorized and, as such, can be much longer and double as an identifier. An API key might look something like this:

bd\_JDS543J3n5NMKspDXNRlowiqw523lKHK32K43kl

# Authentication with JWTs

There are several different ways to handle authentication. We'll use [JWTs](https://blog.boot.dev/cryptography/hmac-and-macs-in-jwts/) in this course. They're a popular choice for APIs that are consumed by web applications and mobile apps.

## Step 1: Login

It would be pretty annoying if you had to enter your username and password every time you wanted to make a request to an API. Instead, after a user enters a username and password, our server should respond with a token (JWT) that's saved in the client's device.

The token remains valid until it expires, at which point the user will need to log in again.

## Step 2: Using the token

When the user wants to make a request to the API, they send the token along with the request in the HTTP headers. The server can then verify that the token is valid, which means the user is who they say they are.

## What is a JWT?

A JWT is a JSON Web Token. It's a cryptographically signed JSON object that contains information about the user. You'll learn about how the cryptography of JWTs work in our [Learn Cryptography](https://boot.dev/courses/learn-cryptography) course, for now, it's just important to know that once the token is created by the server, the data in the token can't be changed without the server knowing.

When your server issues a JWT to Bob, Bob can use that token to make requests as Bob to your API. Bob won't be able to change the token to make requests as Alice.

## Assignment

1. Add a MakeJWT function to your auth package:

func MakeJWT(userID uuid.UUID, tokenSecret string, expiresIn time.Duration) (string, error)

Create a and return a JWT using [this JWT library](https://github.com/golang-jwt/jwt). Again, you'll need to install the library. You'll need to:

* Use [jwt.NewWithClaims](https://pkg.go.dev/github.com/golang-jwt/jwt/v5#NewWithClaims) to create a new token.
  + Use [jwt.SigningMethodHS256](https://pkg.go.dev/github.com/golang-jwt/jwt/v5#SigningMethodHS256) as the signing method.
  + Use [jwt.RegisteredClaims](https://pkg.go.dev/github.com/golang-jwt/jwt/v5#RegisteredClaims) as the claims.
    - Set the Issuer to "chirpy"
    - Set IssuedAt to the current time in UTC
    - Set ExpiresAt to the current time plus the expiration time (expiresIn)
    - Set the Subject to a stringified version of the user's id
  + Use [token.SignedString](https://pkg.go.dev/github.com/golang-jwt/jwt/v5#Token.SignedString) to sign the token with the secret key. Refer to [here](https://golang-jwt.github.io/jwt/usage/signing_methods/#signing-methods-and-key-types) for an overview of the different signing methods and their respective key types.

1. Add a ValidateJWT function to your auth package:

func ValidateJWT(tokenString, tokenSecret string) (uuid.UUID, error)

Use the [jwt.ParseWithClaims](https://pkg.go.dev/github.com/golang-jwt/jwt/v5#ParseWithClaims) function to validate the signature of the JWT and extract the claims into a [\*jwt.Token](https://pkg.go.dev/github.com/golang-jwt/jwt/v5#Token) struct. An error will be returned if the token is invalid or has expired. If the token is invalid, return a 401 Unauthorized response from your handler.

If all is well with the token, use the [token.Claims](https://pkg.go.dev/github.com/golang-jwt/jwt/v5#Claims) interface to get access to the user's id from the claims (which should be stored in the Subject field). Return the id as a uuid.UUID.

1. Add some more unit tests to the auth package. Make sure that you can create and validate JWTs, and that expired tokens are rejected and JWTs signed with the wrong secret are rejected.
2. Add a GetBearerToken function to your auth package:

func GetBearerToken(headers http.Header) (string, error)

Auth information will come into our server in the Authorization header. Its value will look like this:

Bearer TOKEN\_STRING

This function should look for the Authorization header in the headers parameter and return the TOKEN\_STRING if it exists (stripping off the Bearer prefix and whitespace. If the header doesn't exist, return an error. This is an easy one to write a unit test for, and I'd recommend doing so.

1. Create a secret for your server and store it in your .env file. This is the secret used to sign and verify JWTs. By keeping it safe, no other servers will be able to create valid JWTs for your server. Secrets should **NOT** be stored in Git, just in case anyone malicious gains access to your repository. We will yet again use [environment variables](https://en.wikipedia.org/wiki/Environment_variable). You can generate a nice long random string on the command line like this:

openssl rand -base64 64

1. Load the JWT secret from your .env file in your main() function and store it in your apiConfig struct.
2. Update your POST /api/login endpoint. It should accept a new, optional expires\_in\_seconds field in the request body:

{

"password": "04234",

"email": "lane@example.com",

"expires\_in\_seconds": 2

}

If it's specified by the client, use it as the expiration time. If it's not specified, use a default expiration time of 1 hour. If the client specified a number over 1 hour, use 1 hour as the expiration time.

Once you have the token, respond to the request with a 200 code and this body shape:

{

"id": "5a47789c-a617-444a-8a80-b50359247804",

"created\_at": "2021-07-01T00:00:00Z",

"updated\_at": "2021-07-01T00:00:00Z",

"email": "lane@example.com",

"token": "eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJzdWIiOiIxMjM0NTY3ODkwIiwibmFtZSI6IkpvaG4gRG9lIiwiaWF0IjoxNTE2MjM5MDIyfQ.SflKxwRJSMeKKF2QT4fwpMeJf36POk6yJV\_adQssw5c"

}

1. Update the POST /api/chirps endpoint. It is not an authenticated endpoint. To post a chirp, a user needs to have a valid JWT. The JWT will determine which user is posting the chirp. Use your GetBearerToken and ValidateJWT functions. If the JWT is invalid, return a 401 Unauthorized response.

**Struggling?** I, Boots the Gormless Glutton, am happy to help! Ask me and I'll answer for a price. If you find my explanations suspect, ask a human for help in [the Discord](https://boot.dev/community)!



Copy/paste one of the following commands into your terminal:

**Run**

bootdev run 0689e0d0-bdb1-4cc8-b577-f0dd0535ad00

**Submit**

bootdev run 0689e0d0-bdb1-4cc8-b577-f0dd0535ad00 -s

* Default base URL: http://localhost:8080
* Optionally use the -b flag to override the default baseURL

Run the CLI commands to test your solution.

1. 1. **POST /admin/reset**
   * 1.

Expecting status code: 200

1. 2. **POST /api/users**

**Request Body:**

{

"email": "saul@bettercall.com",

"password": "123456"

}

* + 1.

Expecting status code: 201

* + 2.

Expecting JSON at

.email

 to be equal to

saul@bettercall.com

1. 3. **POST /api/users**

**Request Body:**

{

"email": "mike@bettercall.com",

"password": "98765"

}

* + 1.

Expecting status code: 201

* + 2.

Expecting JSON at

.email

 to be equal to

mike@bettercall.com

1. 4. **POST /api/login**

**Request Body:**

{

"email": "saul@bettercall.com",

"password": "123456"

}

* + 1.

Expecting status code: 200

1. 5. **POST /api/login**

**Request Body:**

{

"email": "mike@bettercall.com",

"password": "98765"

}

* + 1.

Expecting status code: 200

1. 6. **POST /api/chirps**

**Headers:**

{

"Authorization": "Bearer ${jwtTokenSaul}"

}

**Request Body:**

{

"body": "Clearly his taste in women is the same as his taste in lawyers: only the very best... with just a right amount of dirty!"

}

* + 1.

Expecting status code: 201

* + 2.

Expecting JSON at

.user\_id

 to be equal to

${userIDSaul}

1. 7. **POST /api/chirps**

**Headers:**

{

"Authorization": "Bearer ${jwtTokenMike}"

}

**Request Body:**

{

"body": "No more half-measures."

}

* + 1.

Expecting status code: 201

* + 2.

Expecting JSON at

.user\_id

 to be equal to

${userIDMike}

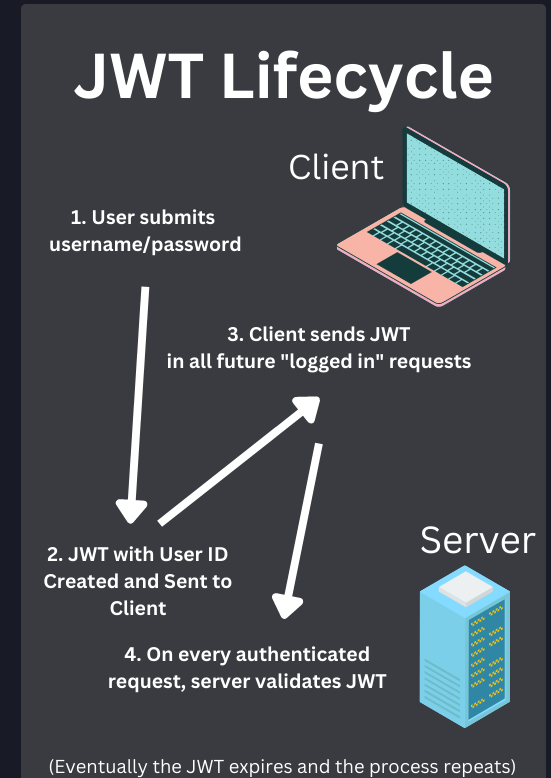
Become a member to submit

Solution

***Using the Bootdev CLI***

*The Bootdev CLI is the only way to submit your solution for this type of lesson. We need to be able to make HTTP requests outside of the confines of the browser.*

*You can*[*install it here*](https://github.com/bootdotdev/bootdev)*. It's a Go program hosted on GitHub, so you'll need Go installed as well. Instructions are on the GitHub page.*



JWTs are cryptographically signed JSON objects that contain information about an authenticated user.

I've heard "JWT" pronounced as "jot", but I pronounce it "jay double yoo tee".

## JWTs can't be changed

We'll talk about [MACs, HMACs](https://blog.boot.dev/cryptography/hmac-and-macs-in-jwts/), and digital signatures in a later course, which are the cryptographic concepts that power JWTs. For now, it's just important to know that once the token is created by a server, the data in the token can't be changed without the server being aware of it.

When your server issues a JWT to Bob, Bob can use that token to make requests as Bob to your API. Bob won't be able to change the token to make requests as Alice.

## JWTs are not encrypted

JWTs are not encrypted. Anyone who has the token can read the data (like the expiry and the user id) in the token. This is why you should never store sensitive information in a JWT. It's just a way to authenticate a user.

I like using [JWT.io](https://jwt.io/) to inspect JWTs. It is a great tool playing around with them and learning how they work.

## JWT Lifecycle

**Struggling?** I, Boots the Hump Day Holdout, am happy to help! Ask me and I'll answer for a price. If you find my explanations suspect, ask a human for help in [the Discord](https://boot.dev/community)!



What happens if a client manually alters the encoded user ID in a JWT?

The server won't know - this is a known vulnerability

The server will know because the token is quantumly entangled with the server

The server will know because the signature will be invalid

Become a member for quiz access

# Revoking JWTs

One of the main benefits of JWTs is that they're stateless. The server doesn't need to keep track of which users are logged in via JWT. The server just needs to issue a JWT to a user and the user can use that JWT to authenticate themselves. Statelessness is fast and scalable because your server doesn't need to consult a database to see if a user is currently logged in.

However, that same benefit poses a potential problem. JWTs can't be revoked. If a user's JWT is stolen, there's no easy way to stop the JWT from being used. JWTs are just a signed string of text.

The JWTs we've been using so far are more specifically access tokens. Access tokens are used to authenticate a user to a server, and they provide access to protected resources. Access tokens are:

* Stateless
* Short-lived (15m-24h)
* Irrevocable

They must be short-lived because they can't be revoked. The shorter the lifespan, the more secure they are. Trouble is, this can create a poor user experience. We don't want users to have to log in every 15 minutes.

## A Solution: Refresh Tokens

Refresh tokens don't provide access to resources directly, but they can be used to get new access tokens. Refresh tokens are much longer lived, and importantly, they can be revoked. They are:

* Stateful
* Long-lived (24h-60d)
* Revocable

Now we get the best of both worlds! Our endpoints and servers that provide access to protected resources can use access tokens, which are fast, stateless, simple, and scalable. On the other hand, refresh tokens are used to keep users logged in for longer periods of time, and they can be revoked if a user's access token is compromised.

# Refresh Tokens

To allow our users to stay logged in for longer periods, let's add refresh tokens to our authentication system. At the same time, we'll reduce the lifespan of our access tokens to improve security.

## Session store

In our case, a refresh token will just be a random 256-bit string. It's a token, but not a JSON Web Token. It doesn't need to be a JWT because we'll store it in our database and associate it with a user server-side. No point in using stateless JWTs if we're going to store them in a database anyway.

To revoke a refresh token, we'll set a revoked\_at timestamp in the database. If revoked\_at is not null, the token is revoked and will be considered invalid.

## Assignment

1. Create a new database table with up/down migrations called refresh\_tokens.

* token: the primary key - it's just a string
* created\_at
* updated\_at
* user\_id: foreign key that deletes the row if the user is deleted
* expires\_at: the timestamp when the token expirees
* revoked\_at: the timestamp when the token was revoked (null if not revoked)

1. Add a func MakeRefreshToken() (string, error) function to your internal/auth package. It should use the following to generate a random 256-bit (32-byte) hex-encoded string:

* [rand.Read](https://pkg.go.dev/crypto/rand#Read) to generate 32 bytes (256 bits) of random data from the crypto/rand package (math/rand's Read function is deprecated).
* [hex.EncodeToString](https://pkg.go.dev/encoding/hex#EncodeToString) to convert the random data to a hex string

1. Update the POST /api/login endpoint to return a refresh token, as well as an access token:

{

"id": "5a47789c-a617-444a-8a80-b50359247804",

"created\_at": "2021-07-01T00:00:00Z",

"updated\_at": "2021-07-01T00:00:00Z",

"email": "lane@example.com",

"token": "eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJzdWIiOiIxMjM0NTY3ODkwIiwibmFtZSI6IkpvaG4gRG9lIiwiaWF0IjoxNTE2MjM5MDIyfQ.SflKxwRJSMeKKF2QT4fwpMeJf36POk6yJV\_adQssw5c",

"refresh\_token": "56aa826d22baab4b5ec2cea41a59ecbba03e542aedbb31d9b80326ac8ffcfa2a"

}

* Access tokens (JWTs) should expire after 1 hour. Expiration time is stored in the exp claim. You can remove the optional expires\_in\_seconds parameter from the endpoint.
* Refresh tokens should expire after 60 days. Expiration time is stored in the database.
* The revoked\_at field should be null when the token is created.

1. Create a POST /api/refresh endpoint. This new endpoint does not accept a request body, but does require a **refresh** token to be present in the headers, in the same Authorization: Bearer <token> format.

Look up the token in the database. If it doesn't exist, or if it's expired, respond with a 401 status code. Otherwise, respond with a 200 code and this shape:

{

"token": "eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJzdWIiOiIxMjM0NTY3ODkwIiwibmFtZSI6IkpvaG4gRG9lIiwiaWF0IjoxNTE2MjM5MDIyfQ.SflKxwRJSMeKKF2QT4fwpMeJf36POk6yJV\_adQssw5c"

}

The token field should be a newly created access token for the given user that expires in 1 hour. I wrote a GetUserFromRefreshToken SQL query.

1. Create a new POST /api/revoke endpoint. This new endpoint does not accept a request body, but does require a refresh token to be present in the headers, in the same Authorization: Bearer <token> format.

Revoke the token in the database that matches the token that was passed in the header of the request by setting the revoked\_at to the current timestamp. Remember that any time you update a record, you should also be updating the updated\_at timestamp.

Respond with a [204 status code](https://www.rfc-editor.org/rfc/rfc9110.html#name-204-no-content). A 204 status means the request was successful but no body is returned.

**Run and submit the CLI tests.**

**Struggling?** I, Boots the Coffee Connoisseur, am happy to help! Ask me and I'll answer for a price. If you find my explanations suspect, ask a human for help in [the Discord](https://boot.dev/community)!



Copy/paste one of the following commands into your terminal:

**Run**

bootdev run f7285cef-5185-4b15-b5fc-9533ccaafe8a

**Submit**

bootdev run f7285cef-5185-4b15-b5fc-9533ccaafe8a -s

* Default base URL: http://localhost:8080
* Optionally use the -b flag to override the default baseURL

Run the CLI commands to test your solution.

1. 1. **POST /admin/reset**
   * 1.

Expecting status code: 200

1. 2. **POST /api/users**

**Request Body:**

{

"email": "saul@bettercall.com",

"password": "123456"

}

* + 1.

Expecting status code: 201

* + 2.

Expecting JSON at

.email

 to be equal to

saul@bettercall.com

1. 3. **POST /api/login**

**Request Body:**

{

"email": "saul@bettercall.com",

"password": "123456"

}

* + 1.

Expecting status code: 200

1. 4. **POST /api/chirps**

**Headers:**

{

"Authorization": "Bearer ${saulRefreshToken}"

}

**Request Body:**

{

"body": "Let’s just say I know a guy... who knows a guy... who knows another guy."

}

* + 1.

Expecting status code: 401

1. 5. **POST /api/chirps**

**Headers:**

{

"Authorization": "Bearer ${saulAccessToken}"

}

**Request Body:**

{

"body": "Let’s just say I know a guy... who knows a guy... who knows another guy."

}

* + 1.

Expecting status code: 201

1. 6. **POST /api/refresh**

**Headers:**

{

"Authorization": "Bearer ${saulRefreshToken}"

}

* + 1.

Expecting status code: 200

1. 7. **POST /api/chirps**

**Headers:**

{

"Authorization": "Bearer ${saulAccessToken2}"

}

**Request Body:**

{

"body": "I'm the guy who's gonna win you this case."

}

* + 1.

Expecting status code: 201

1. 8. **POST /api/revoke**

**Headers:**

{

"Authorization": "Bearer ${saulRefreshToken}"

}

* + 1.

Expecting status code: 204

1. 9. **POST /api/refresh**

**Headers:**

{

"Authorization": "Bearer ${saulRefreshToken}"

}

* + 1.

Expecting status code: 401

Become a member to submit

Solution

***Using the Bootdev CLI***

*The Bootdev CLI is the only way to submit your solution for this type of lesson. We need to be able to make HTTP requests outside of the confines of the browser.*

*You can*[*install it here*](https://github.com/bootdotdev/bootdev)*. It's a Go program hosted on GitHub, so you'll need Go installed as well. Instructions are on the GitHub page.*

# okies

HTTP [cookies](https://en.wikipedia.org/wiki/HTTP_cookie) are one of the most talked about, but least understood, aspects of the web.

When cookies are talked about in the news, they're usually implied to simply be privacy-stealing bad guys. While cookies can certainly invade your privacy, that's not what they are.

## What is an HTTP cookie?

A cookie is a small piece of data that a server sends to a client. The client then dutifully stores the cookie and sends it back to the server on subsequent requests.

Cookies can store any arbitrary data:

* A user's name or other tracking information
* A JWT (refresh and access tokens)
* Items in a shopping cart
* etc.

The server decides what to put in a cookie, and the client's job is simply to store it and send it back.

## How do cookies work?

Simply put, cookies work through HTTP headers.

Cookies are sent from the server to the client in the Set-Cookie header. Cookies are most popular for web (browser-based) applications because browsers automatically send any cookies they have back to the server in the Cookie header.

## Why aren't we using cookies?

Simply put, Chirpy's API is designed to be consumed by mobile apps and other servers. Cookies are primarily for browsers.

A good use-case for cookies is to serve as a more strict and secure transport layer for JWTs within the context of a browser-based application.

For example, when using [httpOnly cookies](https://developer.mozilla.org/en-US/docs/Web/HTTP/Cookies#restrict_access_to_cookies), you can ensure that 3rd party JavaScript that's being executed on your website can't access any cookies. That's a lot better than storing JWTs in the browser's local storage, where it's easily accessible to any JavaScript running on the page.



Cookies can store...

User preferences

Session information

Authentication information

Anything

# Authorization

While authentication is about verifying who a user is, authorization is about verifying what a user is allowed to do.

For example, a hypothetical YouTuber ThePrimeagen should be allowed to edit and delete the videos on his account, and everyone should be allowed to view them. Another absolutely-not-real YouTuber TEEJ should be able to view ThePrimeagen's videos, but not edit or delete them.

Authorization logic is just the code that enforces these kinds of rules.

## Assignment

We already have a bit of authorization built into Chirpy: authenticated users can only create chirps for themselves, not for others.

1. Add a PUT /api/users endpoint so that users can update their own (but not other's) email and password. It requires:
   * An access token in the header
   * A new password and email in the request body
2. Hash the password, then update the hashed password and the email for the authenticated user in the database. Respond with a 200 if everything is successful and the newly updated User resource (omitting the password of course).
3. If the access token is malformed or missing, respond with a 401 status code.

**Run and submit the CLI tests.**

# Delete Chirp

Oh no... the Chirpy CEO is chirping again. He's about to get the entire company cancelled. Let's add delete functionality!

## Assignment

1. Add a new DELETE /api/chirps/{chirpID} route to your server that deletes a chirp from the database by its id.
   * This is an authenticated endpoint, so be sure to check the token in the header. Only allow the deletion of a chirp if the user is the author of the chirp.
   * If they are not, return a 403 status code.
2. If the chirp is deleted successfully, return a 204 status code.
3. If the chirp is not found, return a 404 status code.

**Struggling?** I, Boots the Primeval 10x Developer, am happy to help! Ask me and I'll answer for a price. If you find my explanations suspect, ask a human for help in [the Discord](https://boot.dev/community)!



Copy/paste one of the following commands into your terminal:

**Run**

bootdev run 61628ee7-a227-45a2-ab79-2721a52db32a

**Submit**

bootdev run 61628ee7-a227-45a2-ab79-2721a52db32a -s

* Default base URL: http://localhost:8080
* Optionally use the -b flag to override the default baseURL

Run the CLI commands to test your solution.

1. 1. **POST /admin/reset**
   * 1.

Expecting status code: 200

1. 2. **POST /api/users**

**Request Body:**

{

"email": "walt@breakingbad.com",

"password": "123456"

}

* + 1.

Expecting status code: 201

* + 2.

Expecting JSON at

.email

 to be equal to

walt@breakingbad.com

1. 3. **POST /api/login**

**Request Body:**

{

"email": "walt@breakingbad.com",

"password": "123456"

}

* + 1.

Expecting status code: 200

1. 4. **POST /api/chirps**

**Headers:**

{

"Authorization": "Bearer ${walterAccessToken}"

}

**Request Body:**

{

"body": "I did it for me. I liked it. I was good at it. And I was really... I was alive."

}

* + 1.

Expecting status code: 201

1. 5. **GET /api/chirps/${chirpID}**
   * 1.

Expecting status code: 200

1. 6. **DELETE /api/chirps/${chirpID}**
   * 1.

Expecting status code: 401

1. 7. **POST /api/users**

**Request Body:**

{

"email": "saul@bettercall.com",

"password": "123456"

}

* + 1.

Expecting status code: 201

* + 2.

Expecting JSON at

.email

 to be equal to

saul@bettercall.com

1. 8. **POST /api/login**

**Request Body:**

{

"email": "saul@bettercall.com",

"password": "123456"

}

* + 1.

Expecting status code: 200

1. 9. **DELETE /api/chirps/${chirpID}**

**Headers:**

{

"Authorization": "Bearer ${saulAccessToken}"

}

* + 1.

Expecting status code: 403

1. 10. **DELETE /api/chirps/${chirpID}**

**Headers:**

{

"Authorization": "Bearer ${walterAccessToken}"

}

* + 1.

Expecting status code: 204

1. 11. **GET /api/chirps/${chirpID}**
   * 1.

Expecting status code: 404

Become a member to submit

Solution

***Using the Bootdev CLI***

*The Bootdev CLI is the only way to submit your solution for this type of lesson. We need to be able to make HTTP requests outside of the confines of the browser.*

*You can*[*install it here*](https://github.com/bootdotdev/bootdev)*. It's a Go program hosted on GitHub, so you'll need Go installed as well. Instructions are on the GitHub page.*

# Webhooks

Webhooks sound like a scary advanced concept, but they're quite simple.

A webhook is just an event that's sent to your server by an external service when something happens.

For example, here at Boot.dev we use Stripe as a third-party payment processor. When a student makes a payment, Stripe sends a webhook to the Boot.dev servers so that we can unlock the student's membership.

1. Student makes a payment to stripe
2. Stripe processes the payment
3. If the payment is successful, Stripe sends an HTTP POST request to https://api.boot.dev/stripe/webhook (that's not the real URL, but you get the idea)

That's it! The only real difference between a webhook and a typical HTTP request is that the system making the request is an automated system, not a human loading a webpage or web app. As such, webhook handlers must be [idempotent](https://en.wikipedia.org/wiki/Idempotence) because the system on the other side may retry the request multiple times.

## Idempo... what?

Idempotent, or "idempotence", is a fancy word that means "the same result no matter how many times you do it". For example, your typical POST /api/chirps (create a chirp) endpoint will not be idempotent. If you send the same request twice, you'll end up with two chirps with the same information but different IDs.

Webhooks, on the other hand, should be idempotent, and it's typically easy to build them this way because the client sends some kind of "event" and usually provides its own unique ID.

## Assignment

We recently rolled out a new feature called "Chirpy Red". It's a membership program, and members of "Chirpy Red" get pretty incredible features: like the ability to edit chirps after posting them. But that's beside the point...

Chirpy uses "Polka" as its payment provider. They send us webhooks whenever a user subscribes to Chirpy Red. We need to mark users as Chirpy Red members when we receive these webhooks.

1. Add a migration to the users table to include a new column called is\_chirpy\_red. This column should be a boolean, and it should default to false.
2. Add a database query that upgrades a user to chirpy red based on their ID.
3. Add a POST /api/polka/webhooks endpoint. It should accept a request of this shape:

{

"event": "user.upgraded",

"data": {

"user\_id": "3311741c-680c-4546-99f3-fc9efac2036c"

}

}

* If the event is anything other than user.upgraded, the endpoint should immediately respond with a 204 status code - we don't care about any other events.
* If the event is user.upgraded, then it should update the user in the database, and mark that they are a Chirpy Red member.
* If the user is upgraded successfully, the endpoint should respond with a 204 status code and an empty response body. If the user can't be found, the endpoint should respond with a 404 status code.

Polka uses the response code to know whether or not the webhook was received successfully. If the response code is anything other than *2XX*, they'll retry the request.

1. Update all endpoints that return user resources to include the is\_chirpy\_red field.

# Webhooks Review

A webhook is just an event that's sent to your server by an external service. There are just a couple of things to keep in mind when building a webhook handler:

* The third-party system will probably retry requests multiple times, so your handler should be [idempotent](https://en.wikipedia.org/wiki/Idempotence).
* Be extra careful to never "acknowledge" a webhook request unless you processed it successfully. By sending a 2XX code, you're telling the third-party system that you processed the request successfully, and they'll stop retrying it.
* When you're writing a server, you typically get to define the API. However, when you're integrating a webhook from a service like Stripe, you'll probably need to adhere to their API: they'll tell you what shape the events will be sent in.

## Are webhooks and websockets the same thing?

Nope! A websocket is a persistent connection between a client and a server. Websockets are typically used for real-time communication, like chat apps. Webhooks are a one-way communication from a third-party service to your server.

We'll talk about websockets in a future course.

**Struggling?** I, Boots the Lover of Salmon, am happy to help! Ask me and I'll answer for a price. If you find my explanations suspect, ask a human for help in [the Discord](https://boot.dev/community)!



What will likely happen if your server responds to a webhook with a 5XX status code?

The third party will stop trying to send the event

The third party will never send another event to your server

The third party will keep retrying the exact same request

Become a member for quiz access

# API Keys

You may have noticed that there is an issue with our webhook handler: it's not secure!

Anyone can send a request to our webhook handler, and we'll process it. That means that if Chirpy users figured out our API documentation, they could simply upgrade their account without paying!

## Assignment

Luckily, Polka has a solution for this: API keys. Polka provided us with an API key, and if a request to our webhook handler doesn't use that API key, we should reject the request. This ensures that only Polka can tell us to upgrade a user's account.

Your Polka key: f271c81ff7084ee5b99a5091b42d486e

1. Add a new secret value to your .env file called POLKA\_KEY. This is the api key that polka will send so that we know it's them (and not someone else trying to get free Chirpy red). Load it into your server and store it in your apiConfig.
2. Add a func GetAPIKey(headers http.Header) (string, error) to your auth package. It should extract the api key from the Authorization header, which is expected to be in this format:

Authorization: ApiKey THE\_KEY\_HERE

You'll need to strip out the ApiKey part and the whitespace and return just the key.

1. Update the POST /api/polka/webhooks endpoint. It should ensure that the API key in the header matches the one stored in the .env file. If it doesn't, the endpoint should respond with a 401 status code.

# Documentation

When you're designing a server-side API, no one is going to know how to interact with it unless you tell them. Are you going to force the front-end developers, mobile developers, or other back-end service teams to sift through your code and reverse engineer your API?

Of course not! You're a good person. You're going to write documentation.

## First be obvious, then document it anyway

We've talked a lot about how your REST API should follow conventions as much as possible. That said, the conventions are not enough. You still need to document your endpoints. Without documentation, no one will know:

* Which resources are available
* What the path to the endpoints are
* Which HTTP methods are supported for each resource
* What the shape of the data is for each resource
* etc.

## Assignment

One type of endpoint that's nearly impossible to interact with without documentation is a plural GET endpoint, that is, an endpoint that returns a list of resources. They often have different sorting, filtering, and [pagination](https://developer.squareup.com/docs/build-basics/common-api-patterns/pagination) features.

1. Update the GET /api/chirps endpoint. It should accept an optional query parameter called author\_id.
   * If the author\_id query parameter is provided, the endpoint should return only the chirps for that author.
   * If the author\_id query parameter is not provided, the endpoint should return all chirps as it did before.

For example:

GET http://localhost:8080/api/chirps?author\_id=1

Continue sorting the chirps by *created\_at* in ascending order.

## Tips

The [http.Request](https://pkg.go.dev/net/http#Request) struct has a way to grab the query parameters from the URL:

s := r.URL.Query().Get("author\_id")

// s is a string that contains the value of the author\_id query parameter

// if it exists, or an empty string if it doesn't

**Struggling?** I, Boots the Bane of End-Users, am happy to help! Ask me and I'll answer for a price. If you find my explanations suspect, ask a human for help in [the Discord](https://boot.dev/community)!



Copy/paste one of the following commands into your terminal:

**Run**

bootdev run c1a4f8aa-de85-45fe-9e70-e49a98e14e3a

**Submit**

bootdev run c1a4f8aa-de85-45fe-9e70-e49a98e14e3a -s

* Default base URL: http://localhost:8080
* Optionally use the -b flag to override the default baseURL

Run the CLI commands to test your solution.

# How to document

As far as creating documentation goes, there are 2 main approaches:

1. Manually write documentation
2. Use a tool to generate documentation

Obviously, the first approach is easier to get going with if you have a small API, but as the system grows, it can be really hard to keep the documentation up to date.

**Incorrect documentation is worse than no documentation.**

At least when there is no documentation, your clients will reach out and ask for clarification. When the documentation is incorrect, it can lead to a lot of wasted time and frustration.

## Manually writing documentation

When I've worked on smaller teams, we've generally opted to write our documentation in [Markdown files](https://www.markdownguide.org/) and host them on GitHub. This is a great way to get started because Markdown is a simple format that is easy to write and easy to read.

## Some automated tools to be aware of

I've also written and consumed APIs that have used:

* [Swagger](https://swagger.io/)
* [GraphQL](https://graphql.org/) (not RESTful, but still a networking API)
* [Godoc](https://go.dev/blog/godoc) (which only works for REST APIs if you provide an SDK)
* [Postman](https://learning.postman.com/docs/publishing-your-api/documenting-your-api/) (only useful if your team all uses Postman as their HTTP client)

## Okay, but what should I do now?

I recommend writing documentation for your personal projects in Markdown files and storing them alongside the rest of your code in Git. Your project's README.md file is a great place to start, but it's also common for the README.md file to link to a /docs folder that contains more detailed documentation. The benefits are:

* It's easy to get started writing docs
* The documentation lives alongside your code, so it's easy to keep it up to date
* You'll learn Markdown, which is a great skill to have
* GitHub/GitLab will render your Markdown files for you, so your docs will look great

# How to document

As far as creating documentation goes, there are 2 main approaches:

1. Manually write documentation
2. Use a tool to generate documentation

Obviously, the first approach is easier to get going with if you have a small API, but as the system grows, it can be really hard to keep the documentation up to date.

**Incorrect documentation is worse than no documentation.**

At least when there is no documentation, your clients will reach out and ask for clarification. When the documentation is incorrect, it can lead to a lot of wasted time and frustration.

## Manually writing documentation

When I've worked on smaller teams, we've generally opted to write our documentation in [Markdown files](https://www.markdownguide.org/) and host them on GitHub. This is a great way to get started because Markdown is a simple format that is easy to write and easy to read.

## Some automated tools to be aware of

I've also written and consumed APIs that have used:

* [Swagger](https://swagger.io/)
* [GraphQL](https://graphql.org/) (not RESTful, but still a networking API)
* [Godoc](https://go.dev/blog/godoc) (which only works for REST APIs if you provide an SDK)
* [Postman](https://learning.postman.com/docs/publishing-your-api/documenting-your-api/) (only useful if your team all uses Postman as their HTTP client)

## Okay, but what should I do now?

I recommend writing documentation for your personal projects in Markdown files and storing them alongside the rest of your code in Git. Your project's README.md file is a great place to start, but it's also common for the README.md file to link to a /docs folder that contains more detailed documentation. The benefits are:

* It's easy to get started writing docs
* The documentation lives alongside your code, so it's easy to keep it up to date
* You'll learn Markdown, which is a great skill to have
* GitHub/GitLab will render your Markdown files for you, so your docs will look great

# Sorting Chirps

In a more robust server, especially one that uses [UUIDs](https://blog.boot.dev/clean-code/what-are-uuids-and-should-you-use-them/) or something instead of auto-incrementing integers, we would probably want to sort by created\_at time.Time instead of a numeric id.

That said, this is the same concept, and works just fine for our little server.

## Assignment

Update the GET /api/chirps endpoint. It should accept an optional query parameter called sort. It can have 2 possible values:

* asc - Sort the chirps in the response by created\_at in ascending order
* desc - Sort the chirps in the response by created\_at in descending order

asc is the default if no sort query parameter is provided.

## Examples of valid URLs

* GET http://localhost:8080/api/chirps?sort=asc&author\_id=2
* GET http://localhost:8080/api/chirps?sort=asc
* GET http://localhost:8080/api/chirps?sort=desc
* GET http://localhost:8080/api/chirps

**Struggling?** I, Boots the Magnificent, am happy to help! Ask me and I'll answer for a price. If you find my explanations suspect, ask a human for help in [the Discord](https://boot.dev/community)!



Copy/paste one of the following commands into your terminal:

**Run**

bootdev run 2f20da66-64d8-47b4-8678-4a95cd06767a

**Submit**

bootdev run 2f20da66-64d8-47b4-8678-4a95cd06767a -s

* Default base URL: http://localhost:8080
* Optionally use the -b flag to override the default baseURL

Run the CLI commands to test your solution.

1. 1. **POST /admin/reset**
   * 1.

Expecting status code: 200

1. 2. **POST /api/users**

**Request Body:**

{

"email": "walt@breakingbad.com",

"password": "123456"

}

* + 1.

Expecting status code: 201

* + 2.

Expecting JSON at

.email

 to be equal to

walt@breakingbad.com

1. 3. **POST /api/login**

**Request Body:**

{

"email": "walt@breakingbad.com",

"password": "123456"

}

* + 1.

Expecting status code: 200

1. 4. **POST /api/chirps**

**Headers:**

{

"Authorization": "Bearer ${waltAccessToken}"

}

**Request Body:**

{

"body": "I'm the one who knocks!"

}

* + 1.

Expecting status code: 201

* + 2.

Expecting JSON at

.body

 to be equal to

I'm the one who knocks!

1. 5. **POST /api/chirps**

**Headers:**

{

"Authorization": "Bearer ${waltAccessToken}"

}

**Request Body:**

{

"body": "Gale!"

}

* + 1.

Expecting status code: 201

* + 2.

Expecting JSON at

.body

 to be equal to

Gale!

1. 6. **POST /api/chirps**

**Headers:**

{

"Authorization": "Bearer ${waltAccessToken}"

}

**Request Body:**

{

"body": "Cmon Pinkman"

}

* + 1.

Expecting status code: 201

* + 2.

Expecting JSON at

.body

 to be equal to

Cmon Pinkman

1. 7. **POST /api/chirps**

**Headers:**

{

"Authorization": "Bearer ${waltAccessToken}"

}

**Request Body:**

{

"body": "Darn that fly, I just wanna cook"

}

* + 1.

Expecting status code: 201

* + 2.

Expecting JSON at

.body

 to be equal to

Darn that fly, I just wanna cook

1. 8. **GET /api/chirps?sort=desc**
   * 1.

Expecting status code: 200

* + 2.

Expecting JSON at

.[0].body

 to be equal to

Darn that fly, I just wanna cook

* + 3.

Expecting JSON at

.[1].body

 to be equal to

Cmon Pinkman

* + 4.

Expecting JSON at

.[2].body

 to be equal to

Gale!

* + 5.

Expecting JSON at

.[3].body

 to be equal to

I'm the one who knocks!

1. 9. **GET /api/chirps?sort=asc**
   * 1.

Expecting status code: 200

* + 2.

Expecting JSON at

.[0].body

 to be equal to

I'm the one who knocks!

* + 3.

Expecting JSON at

.[1].body

 to be equal to

Gale!

* + 4.

Expecting JSON at

.[2].body

 to be equal to

Cmon Pinkman

* + 5.

Expecting JSON at

.[3].body

 to be equal to

Darn that fly, I just wanna cook

Become a member to submit

Solution

***Using the Bootdev CLI***

*The Bootdev CLI is the only way to submit your solution for this type of lesson. We need to be able to make HTTP requests outside of the confines of the browser.*

*You can*[*install it here*](https://github.com/bootdotdev/bootdev)*. It's a Go program hosted on GitHub, so you'll need Go installed as well. Instructions are on the GitHub page.*

# Adding a README

You're done building Chirpy! Great work!

I want to take a moment to cover how you should think about your public GitHub profile, and especially how it can help you in your job search.

We have a more [in-depth course](https://www.boot.dev/courses/learn-job-search) that includes building a professional GitHub profile, but for now, I want to cover some basics.

## Do I have to put this on GitHub?

You really should. It doesn't necessarily need to be publicly visible, but it's good to keep copies of all of your code for future reference.

## Projects are your professional portfolio

When you start looking for jobs, you're going to want a couple of great projects on your GitHub or GitLab profile that show off your skills.

This probably isn't one of those because you built it using a guide.

That said, it may be wise to treat this project like one and get your feet wet with the process of presenting a project to the world.

## How to present this project

When someone navigates to your project's link, the first thing they'll see is the README.md file. You should quickly and concisely explain:

* What your project does
* Why someone should care
* How to install and run your project

Take a look at one of my portfolio projects as an example: [go-rabbitmq](https://github.com/wagslane/go-rabbitmq).

Good luck!

**Struggling?** I, Boots the Efficient Bubble Sorter, am happy to help! Ask me and I'll answer for a price. If you find my explanations suspect, ask a human for help in [the Discord](https://boot.dev/community)!



Why might this not be the best portfolio project for your job hunt?

It's not cool

It's not a real project

It wasn't built with a team

It was built by following a guide

Become a member for quiz access

npm –v

npm init –y

npm i nodemon –D

Intro

[Music] hello and welcome to nearly eight hours.Welcome of myrn Stack tutorials and instruction this video is made up of tutorials that build upon each other much like the chapters of a book throughout the lessons in this video I will mention links being available in the description below I've compiled all of these links into one GitHub resource that you will find in the description hi I'm Dave gray and I'm the creator of these mernstack project tutorials you can subscribe to my YouTube channel for more tutorials like this one you can also follow me on Twitter and if you're feeling generous you can even buy me a cup of coffee let's get started learning the myrn stack with chapter one what is the myrn

Chapter : MERN Stack Project

stack myrn is an acronym that uses the first letter of four complementary Technologies M is for mongodb e is for expressjs R is for react and n is for node.js so if the myrn stack is full

stack that leads us to ask what is full stack and why is the myrn stack considered to be full stack a full stack application means it requires code that runs on the server and code that runs in the browser the code that runs on the server is referred to as the back end and the code that runs in the browser is referred to as the front end the front end and the back end are typically two completely separate code repositories in a large Enterprise full stack project there may be a team of developers that work on the front end and another separate team of developers that work on the back end as a full stack developer you should be able to work on both the front end and the back end if needed the back end for the myrn stack is a rest API a rest API also known as a restful API is an interface that two computer systems use to exchange information securely over the Internet

the back end will receive requests from the front end those requests can be classified as crud

operations crud is another four-letter acronym like mern the letters of crud stand for create read update and delete these terms also indicate which type of HTTP request methods will be used in the application for example post relates to create git relates to read patch and put requests relate to update and delete has an exact match hopefully all of that information was not only an intro but also somewhat of a review because this is not a beginner's tutorial series I already have many beginners courses available and I recommend those courses as prerequisites to this series if you have not already completed them specifically I suggest my node.js course

before the first lessons of this mern series where we're going to build the back end rest API I also suggest completing my react course my Redux course and my react login series

playlist before the lessons in this course where we build the front end react app I'll put links to all of these suggested prerequisites in the description along with any links to resources and source code for this mern series

we're going to build a tech Notes app for a small computer repair shop specifically Dan D's repair shop so our stakeholder is Dan D or full name Dan Davidson but he goes by Dan D and this

isn't the type of project where I'm going to give you a tour of the full project before we begin we're more taking a real world approach here for this project and to do that we start by gathering some user stories from our stakeholder Dan D I've got vs code open and I've already done a preliminary

interview with Dan D our stakeholder for the tech notes project and I've come up with user stories that Dan wants now if you've never done an interview for user stories you need to think that the person you're interviewing with will probably describe what they want as a user but they're not that technical so it can be difficult to get those technical requirements out of that interview and of course that does take practice so I definitely wanted to start the project with this just so you can

see the highlights that I've organized for this and we can derive our technical requirements from the descriptions that I have received from DND

so the main goal for this application the first thing Dan told me is he wants to replace his current sticky note system he has a small computer repair shop and right now they use yellow sticky notes to write the problem and they slap it on the side of the computer or whatever other technology an iPhone whatever somebody brings in and it goes on their shelf and that's their whole system and he knows they need to get a better system than that so that's what this Tech notes program's main goal is is to have a local database something they can refer to and everybody knows what everybody else was working on and Dan can manage the whole thing the second thing he wants is just a public-facing web page with basic contact info again he's from a small town small computer shop he doesn't need anything extra for his website he just basically wants it to be a business card and we're okay with that because we want to focus on building this back end application for Dan okay he wants to add an employee login to the notes app because he doesn't want just anybody to be able to access it he just wants his employees to access it and then he wants to provide a welcome page after the login thinks that would just be nice to of course show the username maybe the current day time and of course what they have to work on or what's available to them maybe their current level of the administration whether they're an employee a manager Dan has at

least one manager and then he's the admin of course he wants easy navigation he wants to display the current user and assign role at any time throughout the application and then he wants to provide a log out option of course he wants to require users to log out or log in at least once per week so that is a big requirement that we need to think about when it comes to authorization again.

it's not a public-facing site it's not a financial site it doesn't need Ultimate Security but he wants some security but he doesn't want his employees to have to log in all the time either so we need to think about that with the authorization he wants to provide a way to remove employee access as soon as possible possibly he fires someone and needs to remove that access so they do not disrupt the Notes application notes are assigned to specific employees

so everybody has their own responsibility notes have a ticket number title note body created and updated dates notes are either open or completed notes can be employees managers or

admins notes can be deleted by managers and admins only so that's a consideration when we're applying the roles and permissions of the application anyone can create a note because a customer may come in to check in and Dan doesn't know who may be at the counter but they all need to be able to check in

and create a note for a customer employees can only View and edit their

assigned notes however managers and admins can view edit and delete all notes so that's another thing with the

roles and permissions only managers and admins can access the user settings and only managers and admins can create new

users and then desktop mode is the most important because that's where they'll be using it but it should also be

available in mobile now notice this is a markdown file that I created a checklist in if you're not that familiar with

markdown files they end with a DOT MD and that's also what the readme files

for GitHub are usually created with now in Visual Studio code I can preview this file by pressing Ctrl shift in the

letter V and we can see this markdown file as it would appear possibly on GitHub or somewhere else and the nice

thing is we can also use it as a check well it's not checking right now let me go back and if we put an X in here it

would check but then also we can preview this a different way with control K and then just press the letter V and now let's look at this and we have this on

the right and the edit on the left and now if I check this yes it does check and we see the X over here so you can

work with these markdown files within Visual Studio code and they're sometimes very useful for checklists so I just

wanted to highlight that fact as well but those are user stories and we'll create our technical requirements from

these descriptions go ahead and create an empty folder for your backend code

for this mernstack project act four Tech notes and then if you do not already

have it I sure hope you do but if you don't install node.js from

nodejs.org and you can download that right on the home page and they have currently as of the making of this video

LTS is recommended for most users of course get whatever the most current

is that should give you npm and node I already said I hope you do because again

this is not for beginners you might get through today's lesson but I don't feel like you will be comfortable and get

that far if you're an absolute beginner go back to that node.js course if you are now after you've installed node and

npm or maybe you already have them you can control backtick inside of Visual Studio code and it should open up a

terminal window you can check your versions as well by typing node Dash V it says I have I can type npm-v as

well because that also installs and we need that and it's currently .So today we're going to build the

back end or at least start building the back end rest API for this mern Tech

node.js project and that will use node Express and so three of the four

Technologies in the stack so what can we take away from these user stories already just the basic General thing

that we would need for our server is we need to be able to create notes and as

we mentioned in here view which would be read as related to crud edit and that

would be update delete and then there's also the ability to create notes as well

so we need to perform all crud operations four notes and also for users

as we read through that so we know our basic rest API is going to need to complete the crud operations for both

notes and for users and it's eventually going to need to support authentication

although we'll do that last we want to get the application working first and then apply the authentication as the

last step so we're now ready to get our server up and running in node.js and to do that I'm going to once again open up

the terminal window with control and backtick then I'm going to type npm init

and dash Y which will help it avoid all of the questions it usually wants to ask and we can see some of that being

completed here and after we did that we get a package Json file now this is

where our dependencies will be listed and I need to install two dependencies today so I just opened up the terminal

window again I had it full screen but inside of the terminal window to install these dependencies I'm going to use npm

and then type I for install and then I want to install Express which is one of

our myrnstack Technologies and then I'll just press enter and I will let that install and we should see it listed here

as a dependency inside of our package Json after that there's a Dev dependence C which is different from a normal

dependency because we'll only use it during development so I'll type npmi for install again and then I want node mon

at the end Dash capital D and that makes this a development dependency only node

mon will let us run our code and as soon as we save the changes it will continue

to run our server with those updated changes so it's very useful okay after

we've applied both of those we should be able to close out the terminal once again I'm going to scroll to the top of

the package Json and you can see it probably named your project whatever you named your folder mine's in lesson one

it's version .that's fine let's go ahead and put a description in here and

say Tech notes myrn project after that we're going to

use a server.js file this is really just kind of preference index.js is what it

defaults to and then for scripts I'm going to go ahead and remove this test script and instead I'm going to put in a

start script and there I'll have node space server a comma after that because

this is an object I'm going to type Dev and then I'm going to have node mon

space server so we'll use node mon during development and that's the script

that will go ahead and let us start nodemon one other thing I like to do right at the beginning so I don't forget

is create a git ignore file we haven't initialized git for our repository yet

but we could at any point and this is dot get ignore and inside of the git

ignore we list the node modules because we do not want to send those to GitHub or wherever we might keep our code

Repository that is this huge folder here that installed when we added some dependencies and there's no reason to

send that up to the code repository so we create this git ignore file so it's ignored and not sent along with the

other code okay now that we've completed those things let's create our

server.js file at the top of our server file we'll Define Express and we'll set

this equal to require and we'll just require Express right inside of there underneath that we need

to define the app so we'll say const app set that equal to Express and we'll call Express and then

I'm going to define a constant called port and this will help set what port we

are running our server on in development but also when we deploy it somewhere so

here we'll get a DOT EnV Port if the place we would deploy it would have a

port number saved in the environment variables then it would grab that otherwise we're going to run it here

locally at Port okay that's the initial Imports now

let's just tell our app to start listening so we'll say app Dot listen

and then we'll pass in the port and then we have a function and in this function we'll say

console.log and we'll go ahead and make a template literal here we'll say server

running on Port and inside of this we can pass in our

value for Port let's go ahead and save our server file let's press Ctrl back

tick to open up the terminal window again and now we can type NP let me get

lowercase npm run Dev and after I press enter

notice our server has started running this is the console for node.js and it

says server running on Port it's not really doing anything if we send a

request there we wouldn't have any luck but the server is up and running with just those few lines of code let's not

stop there though let's go ahead and make our servers serve a few bits of information to us and the first thing we

need to do for that is import path from the node.js system so there we'll say

require path and after we have path now we can

go ahead and use it here in the body of our server file and we're going to say

app dot use and we're going to listen for the route just the slash which would

normally be the root or the index of a web page and here we'll say Express

dot static and then we'll use path dot join which

is a method of path and then we have two underscores and dur name which is a

global variable that node.js understands and it says look inside of the folder

that we're in after that we'll put a comma and then say look inside of the slash public folder we're telling

Express where to find static files like a CSS file or other resources like an

image that we would use on the server now if we save this it won't be able to

find that if something was looking for it because we haven't created it but notice nodemon went ahead and restarted

and we're still running on Port so now let's go ahead and come over here to

the file tree and create that public folder and inside the public folder I'm going

to create a CSS folder now normally with our rest API we're just going to be receiving requests and sending back Json

data that would be requested and will be receiving Json data however a rest API

can still have a splash page it could still also return information about

requests that cannot be fulfilled and so we can at least set that much up as we start now inside the CSS folder you

might guess we'll go ahead and create a style.css file not being a CSS tutorial

I'm just going to paste in some basic Styles here I'll close the terminal quickly so you can see everything on one

screen we're importing in a Google font we've got a basic reset and just some

styles on the HTML embody nothing big but just a few Styles there so we could

have something on the splash page for the rest API or possibly a page when

a resource is not found let's go back to the server over JS now and let's put in

another line of code that says app dot use and we'll once again look at that root route and now we'll say require

and we're going to look for a routes folder and then a root file and now we

crash the app and you can see nodemon said that because we do not have a routes folder and we required it in the

file so let's go ahead and create that routes folder and then inside of routes we're going to

need to create a file named root dot JS inside of root we need to require

Express again so we'll say const Express set this equal to require

fix press now after that we need to define a router so we'll say const router and

we'll set this equal to Express dot router with a capital R to start the

word router and then we need the path once again so we'll say const path and

we'll require path from node.js and then after we have

required those three things we're ready to say router dot get so this would be a

get request that relates directly back to our HTTP methods and the nice thing

about Express and these routes is they recognize reg X so we can use a reg X

here and the first thing I'm going to type is the carrot and that says at the beginning of the string only and then

we'll put a slash and then the dollar sign which says at the end of the string only so that means this will only match

if the requested route is only a slash and that would be for the root but then

I'm going to put in a pipe which is an or for regex then I'll have slash index

because maybe they would request more than just the slash as they put that in and after that I'm going to have oh I

put in a quote that's the reason we're seeing red right now I want to remove that quote because all of this is going to be inside this string so the quote

comes at the end but then after the index here I'm going to make the HTML

optional so they could request just the slash or maybe just the slash index

without the dot HTML or the user could request the full index.html which would

also work I got an extra parenthesis in there let's put the comma there then we'll have a request and a response for

our function and then inside the function we will send the file back so it'll be response dot send file

then we'll say path.join and we'll use that dur name directory name variable

that nodejs recognizes and now we need to tell it where to find the file and the file is going to be up out of the

routes folder so that's what the two dots indicate and then we're going to tell it to look into a views folder and

then we're going to have it look for the index.html file and that's quite a bit

there but that should wrap it up and I need that parentheses there at the end so there I've got the syntax correct and

we've got our router.get and it will get the index.html file if it matches any of

those three we just now need to create the views folder and the index.html file

but before we leave this file we need to say module.exports and set that equal to

router and now this file is complete also notice our server is once again running nodemon restarted it as soon as

the errors were fixed and it's not crashing even though we're telling them we have a file that we don't

because nobody's requesting it right now so we can go ahead and do that this is not a require the require is actually

what crashed before when the required thing was not available but now this

does not of course crash the application okay now we'll go over here to our file tree Once Again create a new folder

we'll call this views and then inside of the views folder we'll create a new file and call it

index.html I'm going to type an exclamation mark which is an emit abbreviation in vs code then I can press

Tab and we instantly get the foundation of an HTML page here I'm going to

replace the title where it says document with Tech notes API and then inside of the

body I'm just going to use an Helement and type Tech notes and then we need to

bring in the CSS so that is a link Rel is a style sheet

and then the href is going to be set to CSS slash

Style dot CSS and then we close out that link

tag and actually I think I need the slash close there we go and after we save that now it can find this CSS

because that is a static public file that we set up earlier with our route

back here in the server so we didn't have to put in the full file path for

that it's just one of those things that's available inside of this public route our server is still running let's

drag vs code to the left I'll go to my new tab we want to go to localhost colon

and press enter and there we get our Tech notes home page at the root

however we could request something that doesn't exist and we might not get the best results if I put in slash Dave we

just get the basic Express response cannot get Dave so let's fix that by

going ahead and taking care of basically errors and those are for resources

that are not found let's start by creating in a page inside of the views since we're already there so

another new file I'll type html again with the emit

abbreviation to get the foundation here and we'll just say error or we could

say not found I think that's a little bit better than error but then inside of an

Hwe'll just say sorry and then we can put a paragraph as

well and inside the paragraph we'll say the page or let's say the resource

you have requested does not exist and of course we once

again need to bring in the link and that would be Rel equals

style sheet and href equals CSS slash style dot CSS

we'll close that out and save let's go back to the server and let's handle anything that's not found we want to put

this after all the other routes and of course I'll put it before the app listen for this server but here we'll just say

app.all instead of app.use and now

we'll listen for this asterisk which essentially means all everything that reaches it to app.all will be put

through this instead of being routed to anything that's above so that it's the catch-all that goes at the end so then

we'll once again have our request and our response and we'll have our function now and the

first thing we know is the status is a so we can set that right away although we're not sending the response

yet and now we can look at the headers from the requests that come in and

determine what type of response to send so we can say if the request has an accepts header

that is HTML then we can base our response on that so

here inside of this if we'll say res dot send file and now this will basically be what we

did before to send our index.html but we're going to send our and of

course we have to route to it correctly so now we're at the server level so we don't need to go up out of a folder

we're just going to go down into the views folder and from there we're going to get the

html file after that we can put an else if

here we could say request dot accepts

and let's look for Json which would be very common sent to a rest API so if there's a Json request

that wasn't routed properly and didn't get stopped by any of the expected routes this would be the response we'll

say response.json and now inside of this we'll have a message

and then we'll say not found very basic generic message I

guess I was using single quotes before Ctrl D to select both of those switch those to single just to stay consistent

and finally we'll have our last else that will be sent no matter what if HTML

or Json was not matched in the accepts header and here we'll say response DOT

type text is fairly safe just about everything can receive text and we will

send once again our not found now we'll test this out in the

browser and as you might expect we should get HTML back so if I'll

highlight this and just press enter again now we get sorry the resource you have requested does not exist so we got

our page when it didn't match any of the other routes we are well on our way

to having our rest API up and running for our full stack mernstack application

