

Table of Contents

[1 Home](#)

[2 Python](#)

[2.1 What is Python](#)

[2.2 How to Use](#)

[2.3 Writing Code](#)

[2.4 Building Large Projects](#)

[2.5 Advanced Topics](#)

[2.6 Publishing Projects](#)

[2.7 Important Libraries](#)

[2.8 Documentation Libraries](#)

[2.9 Standard Library Gems](#)

[3 Git & GitHub](#)

[3.1 Git Basics](#)

[3.2 Remotes](#)

[4 Docker](#)

[5 Networking](#)

[5.1 General Networking](#)

[5.2 Servers](#)

[5.3 Caddy](#)

[5.4 DNS](#)

[6 Linux and Other OS](#)

[6.1 Linux](#)

[6.2 SSH](#)

[6.3 Backups](#)

[7 Video Processing](#)

[7.1 FFmpeg](#)

[7.2 Streaming](#)

8 AI

8.1 Ollama

1 My Tech Notes

Click on any of the pages on the left to begin...

```
1 | print('hello world')
```

2 Python

2.1 What is Python?

- Python is a general purpose programming language (its good at a lot of things).
- Its one of the most readable languages that exist. It also has one of the biggest communities which means there are a lot of people and resources to learn.
- Python is an interpreted language, there is no compiling required.
- Python is a dynamic language, no need to declare data types.
- There are several flavors of python, some of the most popular are:
 - CPython - the typical, most common, flavor of python. This is usually what people refer to when they say "python". This is written in the C programming language and maintained by a large group of individuals.
 - Jython - Instead of the C language, this flavor of python is written in Java.
 - IronPython (aka IPython) - Instead of the C language, this flavor of python is written in .NET.
 - Anaconda - This flavor of python comes with many common data analysis packages pre-installed. This is maintained by the Anaconda, Inc. company. Personally I think this is just CPython with bloat.
 - PyPy - A fast, minimal version of python that uses a JIT compiler. A little more difficult to configure.
 - Brython - A version of python that can run in a web browser, translating python code into JavaScript.

2.2 How to use Python

2.2.1 Summary Steps

1. Create a virtual environment (venv).
2. pip install any libraries to the venv.
3. Write code in a .py file
4. Run the .py file with the venv.

Example:

```
1 # create venv
2 Python -m venv venv
3
4 # activate venv
5 venv\Scripts\activate
6
7 # install libraries to venv
8 pip install pandas sqlalchemy
9
10 # run your Python file in your venv (if activated)
11 . python_file.py
```

2.2.2 Running Python

```
1 [Python File]          1. write a .py file
2   |
3   v
4 [Python Interpreter]   2. run .py file through interpreter
5   |
6   v
7 [Result]
```

1. Python files end with `.py` for example `new_file.py`. Notice all lowercase and underscore format. Python heavily uses this **snake case** formatting.
2. In your `.py` file, write python code and then run it through the interpreter by using the terminal.
3. Reference the interpreter and then the .py file like the example below.
4. Example: `C:\username\python\bin\python.exe path\to\your\python\file.py`
5. Note you'll modify this to the correct paths on your computer.

2.2.3 About the package manager (pip)

- The python community has a place where people can create and upload their own python packages.
<https://pypi.org/>
- Python has a built-in manager that allows you to install packages from PyPI directly. This manager is called **pip**
- There are many very popular libraries. Some popular and well utilized ones are:
- For data there is [Pandas](#), [SQLAlchemy](#), [DuckDB](#), [Polars](#)
- API handling - [Requests](#), [aiohttp](#)
- Markup parsing - [BeautifulSoup4](#), [lxml](#)
- Web servers - [Flask](#), [Django](#), [FastAPI](#)
- File transfer - [Paramiko](#) (SFTP)
- Environment variable usage - [python-dotenv](#)
- So much more...
- To make use of these, we need to use **pip** to install them into our python interpreter. But we don't want to install everything to the same python interpreter. That would cause it to become bloated if every project used the same interpreter. For this reason, every python project should have its own interpreter (see [Best practices for using python](#))

2.2.4 Best practices for using python

- Every python project should have its own interpreter, called a virtual environment or shortened to "venv". To do this, we need to create a copy of the python interpreter for each project. PyCharm can manage this for you, or you can do it manually. If using PyCharm, you can skip to the pip commands.
- To do this manually, you should use the terminal to navigate to the folder your project files will exist in and type **python -m venv venv**. This runs a python built-in command (**python**) in module mode (**-m**) to run the module venv (**venv**) and names the new interpreter venv (**venv**).
- After you create the venv folder with the command, you'll need to activate it to install the libraries. To activate, run the command **venv\Scripts\activate** . For linux or mac the command is **venv/bin/activate** .
- From here you should see the terminal change to **(venv) C:\users\username\directory** . In PyCharm, you can open the terminal and this will already be setup.
- Now you can run pip commands like:
- **pip install pandas** - install a package by using their PyPI name.
- **pip install pandas sqlalchemy duckdb** - you can install more than one by chaining package names.
- **pip uninstall pandas** - uninstall package by PyPI name.
- **pip install --upgrade pandas** - upgrade already installed packages when a new version comes out.
- **pip list** - show a list of installed packages.
- **pip freeze > requirements.txt** - save the names and versions of installed packages to a file. Good for saving and distributing dependencies so others can clone and use your project.

- Use the venv the same way `C:\path\to\venv\python\bin\python.exe path\to\your\python\file.py` .
PyCharm will automatically use these commands when you right-click and run a script.

2.3 Writing Python Code

Python uses white-space and tabs, instead of curly-braces and semicolons like other common languages.

The simplest python script. This will print text to your terminal window.

```
1 | print('This is my first Python script')
```

2.3.1 Data Types

Native data types in python include:

- Integer `int` - whole number ie: 5
- Float `float` - decimal number ie: 7.9
- String `str` - text surrounded by single or double quotes ie: 'This is a string' or "This is a string"
- List, Tuple `list`, `tuple` - grouped data combined into a collection that can be iterated over. Lists use square brackets ie: a list of integers `[1, 2, 3, 4]` . Tuples use parenthesis ie: `(1, 2, 3, 4)` . Lists can be mutated and tuples cannot.
- Dictionary `dict` - key value pair collection of data, using curly-braces. ie: `{'key 1': 'value 1', 'key 2': 'value 2'}`

Dictionary methods:

- Get value from dictionary `dict_name['key_name']` or `dict_name.get('key_name')`
- Loop through the dict with:

```
1 | for k, v in dict_name.items():  
2 |     print(k, v)
```

String methods:

- f-strings - inject variables into a string with curly braces `print(f'Variable = {x}')`
- Combine iterable into string `''.join(iterable)`

2.3.2 Variables

Any of the data types can be loaded into a variable that can then be referenced later:

```
1 | x = 14  
2 | variable_01 = 'This is a variable'  
3 | also_a_variable = {'foo': 'bar'}
```

Variables can be named almost anything, you just can't start a variable with a number and no special characters.

You can perform mathematical operations, just like most programming languages.

```
1 x = 5 # set variable
2 x = x + 1 # add 1 to x variable
3 x = x * 10 # multiply x by 10
4 x = x / 4 # divide x by 4
5 x = x ** 2 # raise x to the power of 2
6
7 x = 7 // 3 # floor division (divide and round down to the nearest whole number)
8 # output: 2.333 but rounds down to 2
```

You can also use the math library in the python standard library to perform more complex mathematical functions.

[Math Standard Library Docs](#)

2.3.3 If Statements

If statements are great for checking conditions and running code if certain conditions are met.

```
1 x = 10
2 if x > 8:
3     print('x is greater than 8')
4 elif x <= 1:
5     print('x is less or equal to 1')
6 else:
7     print('x is between 1 and 8')
```

Keep in mind that checking equality should be done with two equal signs `==` because a single `=` is for assigning variables.

```
1 x = 100
2
3 if x == 100:
4     print('x is equal to 100')
```

2.3.4 Loops

Types of loops: - For - iterate over an object. - While - loop until a condition is met. - List comprehension - single line loop over an object.

For loop

```

1 numbers = [2, 4, 6, 8, 10]
2
3 for number in numbers:
4     print(number) # will print each number in the list

```

While loop

```

1 x = 0
2 while x < 10:
3     print(x)
4     x = x + 1 # add 1 to x on each loop
5     # this loop will terminate once x is greater than 10

```

List comprehension - this is like a single line for loop

```

1 numbers = [2, 4, 6, 8, 10]
2
3 # if we want to do a simple change to all the numbers in this list, such as double the
4 amounts, we can use list comprehension
numbers = [number * 2 for number in numbers]

```

2.3.5 Comments

Add comments to your code:

- `#` - single line comments
- `""" """` - multi-line comments

```

1 # single line comment.
2
3 """
4 Multi like comment
5 Use this to describe
6 many things.
7 """

```

2.3.6 Importing other libraries

Python comes with the "Python Standard Library" which has a lot of powerful modules ready to go, no installing required.

[Python Standard Library Docs](#)

To use any of these, or to use any libraries installed with pip, they must be imported. This is typically done at the top of the .py file.

```

1 import os # standard library, lets you interact with objects on your operating system
2 like files.
3 import pandas as pd # third-party library installed with pip. This imports the package
  and renames it as pd
  from sqlalchemy import select, insert # this imports specific parts of the sqlalchemy
  library, avoiding importing everything.

```

Load a csv file into a DataFrame using pandas.

```

1 import pandas as pd
2
3 df = pd.read_csv('path/to/file.csv')

```

Connect to a database with sqlalchemy:

```

1 # import library
2 from sqlalchemy import create_engine, text
3
4 # create an engine object to connect
5 engine =
6 create_engine('postgresql+psycopg2://username:password@servername:port/database')
7
8 # connect to the database and query a table
9 with engine.connect() as conn:
10     query = 'SELECT * FROM table'
    result = conn.execute(text(query))

```

2.3.7 Functions

Use the `def` keyword to define a function. You can use parameters to pass data into the function, process the data, and return a result.

```

1 def example_function():
2     pass
3
4
5 def repeat(message):
6     print(f'You said {message}') # this will print back your message and uses f-
7     strings to inject your message to another string.
8
9
10 def add_numbers(number_1, number_2):
    return number_1 + number_2

```

You can create dynamic parameters using *args* and **kwargs*

```

1  def add_numbers(*args):
2      return_value = 0
3      for value in args:
4          return_value = return_value + value
5
6      return return_value

```

2.3.8 Classes

Classes are good for grouping similar data and functions into one object. It's great for maintaining state of data, and performing certain operations on that data.

```

1  import time
2
3
4  class Car:
5      def __init__(self, year, make, model):
6          # This function runs when the class is first used. Good for loading data right at
7          the beginning.
8          self.year = year
9          self.make = make
10         self.model = model
11         self.current_speed = 0
12
13     def accelerate(self):
14         self.current_speed = 10
15         # make car go
16
17
18     def stop(self):
19         self.current_speed = 0
20         # make care brake
21
22
23     car = Car(2024, 'Tesla', 'Model Y') # create the car object
24     car.accelerate()                    # run the accelerate function (when it's in a
class it's called a method)
time.sleep(5)                            # wait 5 seconds
car.stop()                               # run the stop method

```

2.4 Building Large Projects

Large python projects can be extremely powerful, but they must be organized a certain way to work properly.

A project should always start with a root folder and other folders and files will be housed inside the root folder.

The file structure of a project might look something like this:

```
1  /project_root
2      /db
3          __init__.py
4          model.py
5          connectors.py
6      /views
7          __init__.py
8          views.py
9      /reports
10         __init__.py
11         report_01.py
12         report_02.py
13     .env
```

Notice other directories have a file called `__init__.py`. This is a python file that tells python "this folder is a python module" which means you can put python files with code inside the folder and use it. With this structure, you have code in the `views.py` file like:

```
1  from db import model
2  from reports import report_01
```

If there is code in the `__init__.py` file, you can just import the module name like `import db`. If there is a function inside the `__init__.py` file called `new_func` you could `from db import new_func`

2.5 Advanced Topics

2.5.1 Dynamic Function Parameters

If you have a function and you want to be able to pass any number of arguments or keyword arguments, you can use args and kwargs.

```
1 def function_name(*args, **kwargs):
2     # in the function you can access args and kwargs
3     for arg in args:
4         print(arg)
5     for kwarg in kwargs:
6         print(kwarg)
```

You can also unpack dictionaries directly into a function call.

```
1 def greet(name, age):
2     print(f"Hello, {name}. You are {age} years old.")
3
4 data = {"name": "Alice", "age": 25}
5 greet(**data) # Equivalent to greet(name="Alice", age=25)
```

2.5.2 Context Managers

In programming there are a lot of instances where you'll need to open something (file, database connection, etc.) and you'll need to make sure to close it to free up resources and to protect the file from being corrupted or altered.

A very safe way to handle this is with context managers, using the `with` keyword.

Read, write, append to files.

```

1 # write to file by opening in write mode 'w'
2 with open('file.txt', 'w') as write_file:
3     write_file.write('Hello there')
4 # file is closed at this point
5
6 with open('file.txt', 'r') as read_only_file:
7     file_data = read_only_file.read()
8
9 # close the file and use the data from it
10 print(file_data)
11
12 with open('file.txt', 'a') as append_file:
13     append_file.write('This is how to append text to file.')
14
15 with open('file.txt', 'rw') as read_write_file:
16     read_write_file.write('info')
17     data = read_write_file.read()

```

Safely manage connections to a database.

```

1 from sqlalchemy import create_engine
2
3 engine = create_engine('connection_string')
4
5 with engine.connect() as conn:
6     conn.execute('query here')

```

You can create your own objects with context managers.

```

1 class NewObject:
2     def __init__(self, vars):
3         self.vars = vars
4
5     def __enter__(self):
6         # open the connection using with statement
7         self.open()
8
9     def __exit__(self):
10        # automatically run this on exiting with statement
11        self.close()

```

2.5.3 Decorators

Decorators are a way to add extra functionality to other functions.


```

1  import functools
2
3  def decorator_name(func):
4      @functools.wraps(func)
5      def wrapper(*args, **kwargs):
6          # do something before the wrapped function
7          wrapper_value = func(*args, **kwargs)
8          # do something after the wrapped function
9          return wrapper_value
10     return wrapper

```

To use this defined decorator, it would look something like this:

```

1  @decorator_name
2  def wrapped_function():
3      # regular function stuff

```

If you've defined the decorator to also take arguments, you can add them:

```

1  import functools
2
3  def decorator_name(arg1, arg2):
4      def decorator_wrap(func):
5          @functools.wraps(func)
6          def wrapper(*args, **kwargs):
7              # do something before the wrapped function
8              wrapper_value = func(*args, **kwargs)
9              # do something after the wrapped function
10             return wrapper_value
11         return wrapper
12     return decorator_name
13
14
15
16 @decorator_name(arg1, arg2)
17 def wrapped_function():
18     # regular function stuff

```

Here is a more in-depth tutorial on decorators <https://realpython.com/primer-on-python-decorators/>

Some favorite decorators:

```

1  def time_it(func):
2      """Prints the time it takes for a function to run"""
3      @functools.wraps(func)
4      def wrapper(*args, **kwargs):
5          start_time = datetime.datetime.now()
6          return_val = func(*args, **kwargs)
7          print(f'{func.__name__} ran successfully in {datetime.datetime.now() -
8 start_time}')
9          return return_val
10     return wrapper

```

```

1  def avoid_day_of_week(day_of_week: List[int]):
2      """
3      Avoid running the decorated function on certain day of week.
4
5      day_of_week:
6          0 -> Mon
7          1 -> Tue
8          2 -> Wed
9          3 -> Thu
10         4 -> Fri
11         5 -> Sat
12         6 -> Sun
13     """
14
15     def decorator(func):
16         @wraps(func)
17         def wrapper(*args, **kwargs):
18             current_dt = utc_now()
19             dow = calendar.weekday(current_dt.year, current_dt.month, current_dt.day)
20             if dow in day_of_week:
21                 logit(f'Avoiding day of week {calendar.day_abbr[dow]} day integer
22                 ({dow})')
23                 return
24             else:
25                 func(*args, **kwargs)
26
27         return wrapper
28     return decorator

```

2.5.4 Generators

Generators can be thought of as iterables that are not fully loaded into memory. This allows you can handle the same data, without worrying about memory consumption.

A generator is defined by a function that uses the keyword `yield`. When a function is used, it is run and then loses all state afterward. A generator will maintain state and can be called again multiple times.

For example, imagine you need to pull API data using every individual for the last 20 years. Instead of loading all dates into one big list, you can create a generator function to calculate it.

```

1  import datetime
2  import requests
3
4  # this is a generator because it uses the keyword yield.
5  def date_generator(start_date):
6      today = datetime.datetime.today()
7      while start_date < today:
8          yield start_date # this gets returned. If this function is called again, it
9  will start here.
10         start_date = start_date + 1
11
12
13
14 url = 'example_url.com/api'
for date in date_generator(datetime.datetime.date(1990, 1, 1)):
    response = requests.get(f'{url}/date_param={date}')

```

If you don't want to loop over a generator, you can also use the `.next()` method.

```

1  gen = generator_function()
2  next_value = gen.next()

```

More detailed info here <https://wiki.python.org/moin/Generators>

2.5.5 Python Import System

Importing libraries can be tricky in certain cases. The typical use is to just import any python package such as

```
import sys
```

The best option 99% of the time is going to be [run scripts in module mode](#). Read below to learn more about how the python import system works.

A python interpreter or virtual environment has a list of directories to look for imports from.

It will start to look for the import in the first item of the list and go through each directory until it finds the import or fails and raises an exception.

If you are running a module that imports another module that you've created, in a directory outside of the current directory, then you might need to add that directory to the `sys.path`.

- Option 1: Add a `pth` file in the virtual environment with the path you want to add.
- Go to `venv/lib/python/site-packages/`
- Add your path such as `/home/tyranderson/snfStudyData`
- Option 2: Hardcode the path directly into the activate file. In a venv you can edit the `bin/activate` file and include:

```
export PYTHONPATH="/the/path/you/want"
```

- Option 3: Add into python script - within your script you can `sys.path.append("/the/path/you/want")` but this is temporary and the path will be dropped once the script is done running.

More info here: <https://help.pythonanywhere.com/pages/DebuggingImportError>

2.5.6 Run scripts in module mode

You can run scripts from a venv 2 different ways: 1. As a standalone script 1. This is the format path/to/venv/bin/python path/to/script 2. If this script has imports to other files, it will have import errors. This is not the recommended way!!!!!! 2. In module mode 1. Path/to/venv/bin/python -m path.to.script

Using module mode is considered best practice because it allows all modules to import from the project root properly. If you have multiple python modules (.py files) you are importing from various directories within the project, you will likely have import errors if you try to run using the standalone method. This is why its considered best practice to run in module mode. (PyCharm does this for you when you run a script).

Linux

```
1 cd path/to/project
2 . venv/bin/activate -m path.to.Python.file
```

Windows

```
1 cd path\to\project
2 venv\Scripts\activate -m path.to.python.file
```

2.5.7 Asyncio

All the code up to this point has been synchronous, meaning everything happens one step at a time. Asynchronous (async) code can be used to handle multiple tasks at the same time.

Be aware, async code is more complicated than synchronous in any language, including python.

When should I use async code?

If your code is...

- CPU Bound - use [Multi Processing](#)
- IO bound, fast IO, limited number of connections - use [Threading](#)
- IO bound, slow IO, many connections - use [Asyncio](#)

In other words....

```

1  if io_bound:
2      if io_very_slow:
3          print('Use asyncio')
4      else:
5          print('Use threads')
6  else:
7      print('Use Multi processing')

```

General Info

The most important keywords: `async` and `await`

The most important functions: `asyncio.run()` and `asyncio.gather()`

- `async` - this keyword is used to define a function that will be capable of running async. This turns the function into what's called a coroutine. If a function is a coroutine (has the `async` keyword), then it has to utilize the `await` keyword as well.
- `await` - is the keyword that is used to let python know that a function is going to take some time to resolve. It will stop trying to run the function and move on to another task while it waits for the task to resolve.
- `asyncio.run()` - this is the most common way to run a coroutine (function with `async`). A coroutine cannot be called like a normal function.
- `asyncio.gather()` - this is not required but is extremely useful and common. `gather` allows you to run multiple coroutines at the same time.

The best way to understand async code is to experiment with it. Let's look at some examples.

```

1  import asyncio
2
3  # this is a coroutine
4  async def async_function(request_data):
5      data = await some_async_task(request_data)
6      return data
7
8  # this is a coroutine
9  async def main():
10     results = await asyncio.gather(
11         async_function('x'),
12         async_function('y'),
13         async_function('z')
14     )
15     print(results)
16
17
18  if __name__ == '__main__':
19     asyncio.run(main()) # this is how you run a coroutine

```

2.5.8 Cython

One of the biggest criticisms of python is its performance. When you compare it to statically typed, compiled languages, it doesn't have near the same speed. One option to improve performance is with something called Cython. Cython is an extension of python that allows statically typed python that can be compiled to C code for performance enhancements.

[Official Cython Docs](#)

Example:

```
1  # example.pyx file
2
3  def sum_integers(int n):
4      cdef int i
5      cdef int total = 0
6      for i in range(n):
7          total += i
8      return total
```

Create setup.py file:

```
1  from setuptools import setup
2  from Cython.Build import cythonize
3
4  setup(
5      ext_modules = cythonize("example.pyx")
6  )
```

run `python setup.py build_ext --inplace`

2.5.9 Global Interpreter Lock (GIL)

To achieve thread safety in python, there is something called the Global Interpreter Lock (GIL). The GIL is a bit of a double-edged sword because it achieves thread safety, but it also makes python slower due to running everything on one thread. Python 3.13 has introduced an experimental mode where the GIL can be deactivated. The most important thing here for now is to know it exists.

2.5.10 Other Helpful Tips

Unpacking Iterables is a useful trick. If you have a list or other iterable that you want to perform an operation on you can unpack it with an asterisk.

```
1 # with unpacking
2 numbers = [1, 2, 3, 4]
3 print(*numbers)
4
5 # this gives the same result, but requires more code
6 numbers = [1, 2, 3, 4]
7 for number in numbers:
8     print(number)
```

The double asterisk ** is for unpacking keyword arguments. This is why they are used in functions with args and kwargs.

```
1 def function_name(*args, **kwargs):
2     # in the function you can access args and kwargs
3     print(args[0])
4     print(kwargs['item_01'])
5
6
7 function_name(1, 2, 3, item_01='value 1', item_02='value 2')
```

2.6 Publishing a Project

You can create your own python library and publish it to PyPI. First you'll need to create an account and download your API keys. Once you have those established, you can create your project and then:

Super summary:

1. Create the pyproject.toml file and fill out the fields
2. run `python -m build`
3. use twine to send to pypi.

More detailed steps:

1. Make sure all of your files are created inside a folder structure
2. Create pyproject.toml

```
1  [build-system]
2  requires = ["setuptools", "wheel"]
3  build-backend = "setuptools.build_meta"
4
5  [project]
6  name = "example_package_YOUR_USERNAME_HERE"
7  version = "0.0.1"
8  authors = [
9    { name="Example Author", email="author@example.com" },
10 ]
11 description = "A small example package"
12 readme = "README.md"
13 requires-python = ">=3.8"
14 classifiers = [
15     "Programming Language :: Python :: 3",
16     "License :: OSI Approved :: MIT License",
17     "Operating System :: OS Independent",
18 ]
19
20
21 [project.urls]
22 Homepage = "https://github.com/pypa/sampleproject"
    Issues = "https://github.com/pypa/sampleproject/issues"
```

3. Make sure all relevant build libs are installed:

```
1  pip install --upgrade build
2  pip install twine
```

4. Run the build - `python -m build`
5. Send to PyPI test - `twine upload --repository testpypi dist/*`
6. Send to PyPI - `twine upload dist/*`

Make sure your build dependencies are not stored in your project dependencies.

Another TOML file might look like this:

```
1  [build-system]
2  requires = ["setuptools", "wheel"]
3  build-backend = "setuptools.build_meta"
4
5  [project]
6  name = "py_simple_sharepoint" # actual name that will get used for pip install.
7  version = "0.1.1" # anytime you submit an update to PyPI, you must change the
8  version.
9  description = "A SharePoint file management tool for python programs."
10 readme = "README.md"
11 requires-python = ">=3.10"
12 dependencies = [ # package dependencies, does install when pip installed
13     "office365-rest-python-client>=2.6.2",
14 ]
15
16 [project.optional-dependencies] # build dependencies (does not install when pip
17 installed)
18 dev = [
19     "build",
20     "twine"
21 ]
```

<https://packaging.python.org/en/latest/tutorials/packaging-projects/#creating-the-package-files>

There are also new tools coming out such as poetry and uv. The community has started to heavily embrace [UV](#) due to its speed and tooling.

2.7 Important Libraries - Getting Started

2.7.1 Pandas

Pandas is a well-known data handling library. It has the ability to extract, manipulate, and load data.

The central component to pandas is the DataFrame. This is where tabular data is loaded and lives in memory.

```
1  import pandas as pd
2
3  # blank dataframe
4  df = pd.DataFrame()
5
6  # load a dict into a dataframe
7  sample_data = {'id', [1, 2, 3], 'value', [200, 300, 400]}
8  df = pd.DataFrame(sample_data)
9
10
11 # load direct from other formats
12 df = pd.read_csv('path/to/csv/file.csv')
13 df = pd.read_parquet('path')
14 df = pd.read_excel('path')
15 df = pd.read_json('path')
16 df = pd.read_sql('sql statement', sqlalchemy_conn)
17 df = pd.read_html('io')
18 df = pd.read_xml('io')
19 df = pd.read_clipboard()
```

More here <https://pandas.pydata.org/docs/reference/io.html>

Transformations

Once you have data loaded into a dataframe, you can perform all kinds of operations on the values. There are generally two ways of performing modifications. Iterating through each value (not recommended) and across columns (called vectorization). A vectorized operation can look like this:

```
1  df = pd.read_csv('path/to/file.csv') # sample data
2
3  # performing math computations across columns. You can overwrite existing columns or
4  create new columns
5  df['int_column_new'] = df['int_column_01'] + df['int_column_02']
6  df['int_column_01'] = df['int_column_01'] - df['int_column_02']
7  df['float_column_new'] = df['float_column_01'] * df['float_column_02']
8  df['float_column_01'] = df['float_column_01'] / df['float_column_02']
9
10 # perform string operations
11 df['string_col'] = df['string_col'].str.replace('-', '')
```

Filtering

```

1 df = pd.read_csv('path/to/file.csv') # sample data
2
3 # filter where column_name is equal to a value
4 df = df[df['column_name'] == 'certain_value']
5 # not equal
6 df = df[df['column_name'] != 'certain_value']
7 # substring
8 df = df[df['column_name'].str.contains('partial_string_match')]
9 # drop rows where column_name equals any of the list values
10 df = df[~df['column_name'].isin(['list', 'of', 'values'])]

```

Change Data Types

```

1 df = pd.read_csv('path/to/file.csv') # sample data
2
3 df['str_num_values'] = df['str_num_values'].astype(int)
4 df['str_num_values'] = df['str_num_values'].astype(str)
5 df['str_num_values'] = df['str_num_values'].astype(float)
6
7 # dates
8 df['date_str_values'] = pd.to_datetime(df['date_str_values']) # gives datetime format
9 df['date_str_values'] = pd.to_datetime(df['date_str_values']).dt.date # gives just
  date, no time

```

Helpful functions:

[Merge](#) (like SQL join) [Melt](#) (convert columns to rows)

2.7.2 SQLAlchemy

SQLAlchemy is a robust database management utility library. There are two main components; Core and ORM. Core is more base level, closer to the database API, while ORM aims to abstract some of the complexity of managing connections with sessions. Generally Core is better for pure database operations while the ORM is geared toward web applications.

Has full compatibility with Postgres, MySQL, SQLite, SQL Server, and Oracle.

Connection strings look like this:

```

1 db_str = 'dialect+driver://username:password@host:port/database'
2
3 postgres = "postgresql+psycopg2://scott:tiger@localhost/public"
4
5 sql_server = 'mssql+pyodbc://host/database?
  trusted_connection=yes&driver=ODBC+Driver+18+for+SQL+Server&TrustServerCertificate=yes'

```

Modeling Tables

```

1  from sqlalchemy import (create_engine, Table, MetaData, Column, String,
2                          Integer, Double, Date, DateTime, Boolean, func)
3
4  engine = create_engine('postgresql+psycopg2://scott:tiger@localhost/public') #
5  engines are what connects to the db
6
7  meta = MetaData() # create a metadata object to attach tables to
8
9  # model out the table
10 table1 = Table('table_name',
11                meta,
12                Column('id', Integer, primary_key=True),
13                Column('column1', String(255), unique=True),
14                Column('column2', Double),
15                Column('column3', Boolean, default=False),
16                Column('column4', Date),
17                Column('created_date', DateTime,
18                      server_default=func.now()),
19                schema='public'
20                )
21
22 meta.create_all(engine) # create all the tables that are connected to the metadata
23                           object, in the database

```

Once you have your model and engine ready, you can connect and start running operations:

```

1  from sqlalchemy import create_engine, text
2
3  engine = create_engine('connection_string_to_db')
4
5  with engine.connect() as conn:
6      query = 'SELECT * FROM table_name'
7      result = conn.execute(text(query)).fetchall()

```

This will return a list of Row objects which contain the values in tuples (even if you only selected one column). To pull these out of the nested tuples, you can use list comprehension.

```

1  result = [r[0] for r in result]

```

Note that to run a raw SQL query we had to put it into a text() function. This function will sanitize the query to make sure there is no malicious injection happening. Sometimes it is difficult to transform data values in python into a raw SQL query. SQLAlchemy has objects that can perform the same database functions, but in a more pythonic way.

```

1  from sqlalchemy import select, insert, update, delete
2
3  df = pd.DataFrame(values)
4  data_dict = df.to_dict(orient='records') # transform pandas dataframe into a
5  dictionary of values
6
7  with engine.connect() as conn:           # connect to database
8      insert_stmt = insert(table_object).values(data_dict) # create the insert
9  statement
10     conn.execute(insert_stmt)             # execute the statement
11     conn.commit()                        # commit the changes
12
13     value = conn.execute(select(table_object)).fetchone()
14     values = conn.execute(select(table_object)).fetchall()
15
16     update_stmt = update(user_table).where(user_table.c.id == 5).values(name="user
17 #5")
18     conn.execute(update_stmt)
19
20     delete_stmt = delete(user_table).where(user_table.c.id == 5)
21     conn.execute(delete_stmt)
22     conn.commit()

```

2.7.3 SQLite

SQLite is an embeddable OLTP database. In other words, it is a database that can be run in memory or from a file. It is the most common database in the world being used in web apps, phone apps, IoT devices, and more.

[SQLite Docs](#)

2.7.4 DuckDB

DuckDB is similar to SQLite, but is an OLAP database. It is incredibly fast and effective and processing large amounts of data in memory. It can connect to other databases and perform direct queries on it, even being able to write queries on multiple separate databases [Mix and Match](#).

[DuckDB Docs](#)

2.7.5 Pyspark

Spark is a distributed data handling library written in Java. It has a python API that allows users to use Spark with Pyspark. In some ways it is very similar to pandas, but it can directly read and write to Delta Lake tables, and it spreads the data handling tasks across multiple machines (distributed).

```

1  df = spark.createDataFrame(data)
2  current_records = spark.sql("SELECT COALESCE(MAX(created_date), '2000-01-01') FROM
lh_gold_01.fact_cms_star_rating").first()[0]

```

2.7.6 dotenv

dotenv is a lightweight package that allows you to load environment variables into memory from a file.

```
pip install python-dotenv
```

Create a `.env` file in your project like:

```
1 ENV_VAR_NAME='ENV_VAR_VALUE'
2 API_KEY_01='2JHDKFJH3KF'
```

```
1 import os
2 from dotenv import load_dotenv
3
4 load_dotenv()
5
6 env_var = os.getenv('ENV_VAR_NAME')
7 api_key = os.getenv('API_KEY_01')
```

2.7.7 Requests

Requests is a user-friendly way to make HTTP calls. Very good for API's.

```
pip install requests
```

```
1 import requests
2
3 url = 'https://data.cms.gov/provider-data/api/1/metastore/schemas/dataset/items/4pq5-
4 n9py?show-reference-ids=false'
5 response = requests.get(url)
6 print(response.status_code)
7 print(response.headers)
8 print(response.text)
```

Send post requests with auth details and payload.

```
1 import requests
2 from io import BytesIO
3
4 response = requests.get(url,
5                          auth={'user': 'username', 'password': 'password_here'},
6                          data={'payload_json': {'data': 'value'}},
7                          files={'file': BytesIO()})
```

2.7.8 Airflow

Airflow is an orchestration tool to run and monitor jobs. It allows you to program it in a way that it can have dependencies. For example, only run one script after another has run successfully.

[Airflow Docs](#)

2.7.9 Flask

Flask is a popular web app framework. It is very lightweight and has many "plugin" type packages that are built to be pieced together to achieve all desired features.

Web workers rule of thumb:

- Sync workers: ~50 - 100 concurrent users per worker.
- Async workers: ~500 - 1000+ concurrent users per worker.

Running a flask server:

- Standard run Flask: `flask --app <.py file> run` like `flask --app app run`
- Run Flask on different port: `flask --app app run --port 8080`
- Run Flask and expose to network: `flask --app web_app/app run --host 0.0.0.0 --port 8080`
- Run Flask in prod with gunicorn: `gunicorn -w 4 -b 0.0.0.0:8080 web_app.app:app`

Self Hosting

Make sure to activate your python interpreter!

If you want to host your own flask app, gunicorn is a WSGI pure python server used for production. You'll need to `pip install gunicorn` and then run the commands.

Running locally:

```
gunicorn -w 4 'module_name:app_name'
```

`-w` is the number of workers, default is 1.

If you want to be able to access the site on the network, you need to bind to 0.0.0.0:

```
gunicorn -w 4 -b 0.0.0.0:8000 'app:app'
```

`0.0.0.0` binds the app to all available network interfaces, making it accessible on your network. This is a special ip address that tells your application to listen on all available network interfaces of the machine, instead of just localhost.

To make this accessible from the internet, you need to configure your router to forward traffic that goes to your chosen port (8000 in this case) to the machine that is running the app. This is done with port forwarding in your router admin settings. Visually it might look like `public_ip:8000 -> server_local_ip:8000`. BTW this is called port forwarding.

You may need to update firewall config on the server to allow external connections on port 8000 (linux `sudo ufw allow 8000`).

If your ISP changes your public IP, you may need to use a dynamic DNS (DDNS) service like No-IP or DynDNS.

Security Warning: exposing your app to the internet comes with security risks. To mitigate:

1. Use HTTPS: Serve your app behind a reverse proxy like nginx with an SSL certificate.
2. Restrict Access: Limit access to specific IPs or use authentication to secure your app.
3. Monitor Logs: Monitor your server logs for unusual activity.

Stopping the app:

Use `pgrep -fl gunicorn` to show a list of gunicorn processes with their PIDs.

kill by pid: `kill <pid>` (kill by process ID)

OR

kill all gunicorn: `pkill gunicorn` (kill by process name)

For a more stable production environment, a process manager is better.

Create a systemd service file.

Then you can enable/disable/start/stop the app as a service.

```
1 sudo systemctl enable gunicorn # auto start on boot
2 sudo systemctl start gunicorn # prevent auto start on boot
3 sudo systemctl status gunicorn
4 sudo systemctl stop gunicorn
5 sudo systemctl restart <service-name>
```

When you make changes to a service file, you need to reload the service:

```
sudo systemctl daemon-reload
```

[Flask Docs](#)

2.7.10 FastAPI

FastAPI is one of the most performant web frameworks available for python.

2.7.11 Other Interesting Libraries

Locust

Locust is a load testing framework that can send millions of requests.

Reportlab

Creating PDF's can be difficult or require external software like LaTeX or wkhtmltopdf.

If you don't want to have these dependencies, reportlab is a pure python library to create PDFs.

```
pip install reportlab
```

ReportLab includes a low-level API for generating PDF documents directly from Python, and a higher-level template language—similar to HTML and the template systems used in web development—called RML. Generally, the second option is usually more convenient for those who must make exhaustive use of the capabilities of the library when generating documents. For the rest of the cases, the low-level API that we will describe in this article will suffice. However, you can find the official documentation here <https://docs.reportlab.com/>.

```
1  from reportlab.pdfgen import canvas
2
3  c = canvas.Canvas("hello-world.pdf")
4  c.save()
```

```
1  # import the canvas object
2  from reportlab.pdfgen import canvas
3
4  # create a Canvas object with a filename
5  c = canvas.Canvas("rl-hello_again.pdf", pagesize=(595.27, 841.89)) # A4 pagesize
6  # draw a string at x=100, y=800 points
7  # point ~ standard desktop publishing (72 DPI)
8  # coordinate system:
9  #   y
10 #   |
11 #   |   page
12 #   |
13 #   |
14 #   |
15 #   0-----x
16 c.drawString(50, 780, "Hello Again")
17 # finish page
18 c.showPage()
19 # construct and save file to .pdf
   c.save()
```

Use standard page sizes like letter, A4, and more.

```
1  from reportlab.lib.pagesizes import letter
2
3  c = canvas.Canvas("hello-world.pdf", pagesize=letter)
```

Make shapes in PDF.

```
1 from reportlab.lib.pagesizes import A4
2 from reportlab.pdfgen import canvas
3
4 w, h = A4
5 c = canvas.Canvas("shapes.pdf", pagesize=A4)
6 c.drawString(30, h - 50, "Line")
7 x = 120
8 y = h - 45
9 c.line(x, y, x + 100, y)
10 c.drawString(30, h - 100, "Rectangle")
11 c.rect(x, h - 120, 100, 50)
12 c.drawString(30, h - 170, "Circle")
13 c.circle(170, h - 165, 20)
14 c.drawString(30, h - 240, "Ellipse")
15 c.ellipse(x, y - 170, x + 100, y - 220)
16 c.showPage()
17 c.save()
```

You can make tables from a pandas dataframe

```

1 from reportlab.lib.pagesizes import letter, landscape
2 from reportlab.platypus import SimpleDocTemplate, Table, TableStyle
3 from reportlab.lib import colors
4
5 filename = "landscape_table_fit.pdf"
6 doc = SimpleDocTemplate(filename, pagesize=landscape(letter))
7
8 # Example DataFrame
9 import pandas as pd
10 df = pd.DataFrame({
11     "Column 1": ["A", "B", "C"],
12     "Column 2": [1, 2, 3],
13     "Column 3": [4.5, 5.5, 6.5],
14     "Column 4": ["Long Text Example", "Another Example", "More Text"],
15     "Column 5": ["X", "Y", "Z"]
16 })
17
18 # Convert DataFrame to list of lists
19 data = [df.columns.tolist()] + df.values.tolist()
20
21 # Calculate dynamic column widths to fit the page
22 page_width = landscape(letter)[0]
23 available_width = page_width - 40 # Subtract margins
24 num_columns = len(df.columns)
25 col_width = available_width / num_columns
26
27 # Create a table with fixed column widths
28 table = Table(data, colWidths=[col_width] * num_columns)
29 table.setStyle(TableStyle([
30     ('BACKGROUND', (0, 0), (-1, 0), colors.lightgrey),
31     ('TEXTCOLOR', (0, 0), (-1, 0), colors.black),
32     ('ALIGN', (0, 0), (-1, -1), 'CENTER'),
33     ('FONTNAME', (0, 0), (-1, 0), 'Helvetica-Bold'),
34     ('BOTTOMPADDING', (0, 0), (-1, 0), 12),
35     ('GRID', (0, 0), (-1, -1), 1, colors.grey),
36 ]))
37
38 # Add the table to the elements list
39 elements = [table]
40
41 # Build the document
42 doc.build(elements)
43
44 print(f"PDF saved as {filename}")

```

2.8 Documentation Tools

2.8.1 Sphinx

2.8.2 Mermaid

2.8.3 MKDocs

2.8

2.9 Notes on Standard Library

2.9.1 IO package

<https://docs.python.org/3/library/io.html>

The `io` module provides Python's main facilities for dealing with various types of I/O. There are three main types of I/O: text I/O, binary I/O and raw I/O. These are generic categories, and various backing stores can be used for each of them. A concrete object belonging to any of these categories is called a file object. Other common terms are stream and file-like object.

Independent of its category, each concrete stream object will also have various capabilities: it can be read-only, write-only, or read-write. It can also allow arbitrary random access (seeking forwards or backwards to any location), or only sequential access (for example in the case of a socket or pipe).

Text I/O

Text I/O expects and produces `str` objects. This means that whenever the backing store is natively made of bytes (such as in the case of a file), encoding and decoding of data is made transparently as well as optional translation of platform-specific newline characters.

The easiest way to create a text stream is with `open()`, optionally specifying an encoding:


```
f = open("myfile.txt", "r", encoding="utf-8")
```

In-memory text streams are also available as `StringIO` objects:

```
f = io.StringIO("some initial text data")
```

`StringIO` is useful when you need to read or write data to a string buffer as if it were a file, rather than creating an actual file on disk. It can be used to create strings that mimic file objects, allowing you to read and write data to them in the same way you would with a file.

python

 Copy code

```
from io import StringIO

# create a new StringIO object
my_string = StringIO()

# write some data to the StringIO object
my_string.write("Hello, World!")

# move the file pointer to the beginning of the buffer
my_string.seek(0)

# read the data from the StringIO object
data = my_string.read()

print(data) # output: "Hello, World!"
```

Binary I/O

Binary I/O (also called buffered I/O) expects bytes-like objects and produces bytes objects. No encoding, decoding, or newline translation is performed. This category of streams can be used for all kinds of non-text data, and also when manual control over the handling of text data is desired.

The easiest way to create a binary stream is with `open()` with 'b' in the mode string:

```
f = open("myfile.jpg", "rb")
```

In-memory binary streams are also available as `BytesIO` objects:

```
f = io.BytesIO(b"some initial binary data: \x00\x01")
```

2.9.2 SMTP - Sending Emails

You can send emails directly through python using the simple mail transmission protocol.

```

1  import smtplib
2  from email.message import EmailMessage
3
4  msg = EmailMessage()
5  msg['From'] = 'email@address.com'
6  msg['To'] = 'recipient@address.com'
7  msg['Subject'] = 'This is an Email'
8
9  msg.set_content('Body of email here.')
10
11  mailserver = smtplib.SMTP('smtp.office365.com', 587) # connect to email service
12  mailserver.starttls() # starts TLS (security protocol that encrypts data sent over
13  the internet
14  mailserver.login(msg['From'], 'password') # login to the email service account
15  mailserver.sendmail(msg['From'], msg['To'], msg.as_string()) # send the email
16  mailserver.quit()

```

Mail with attachment

```

1  import os
2  import smtplib
3  from email.message import EmailMessage
4
5  msg = EmailMessage()
6  msg['From'] = 'email@address.com'
7  msg['To'] = 'recipient@address.com'
8  msg['Subject'] = 'This is an Email'
9
10  msg.set_content('Body of email here.')
11
12  file = 'path/to/file.txt'
13  extension = os.path.basename(file).split('.')[1]
14  with open(file, 'rb') as f:
15      file_data = f.read()
16  msg.add_attachment(file_data, maintype='application', subtype=extension,
17  filename=os.path.basename(file))
18
19
20  mailserver = smtplib.SMTP('smtp.office365.com', 587) # connect to email service
21  mailserver.starttls() # starts TLS (security protocol that encrypts data sent over
22  the internet
23  mailserver.login(msg['From'], 'password') # login to the email service account
24  mailserver.sendmail(msg['From'], msg['To'], msg.as_string()) # send the email
25  mailserver.quit()

```

You can also send nicely formatted HTML emails with MIME

```

1  import smtplib
2  from email.mime.multipart import MIMEMultipart
3  from email.mime.text import MIMEText
4
5  msg = MIMEMultipart('mixed')
6  msg['From'] = 'email@address.com'
7  msg['To'] = 'recipient@address.com'
8  msg['Subject'] = 'Subject Here'
9
10 html_body = MIMEMultipart('alternative')
11
12 html_body_text = MIMEText('<html><div>HTML Content here</div></html>', 'html')
13 html_body.attach(html_body_text)
14
15 # Attach the multipart/alternative part to the multipart/mixed message
16 msg.attach(html_body)
17
18 # Remaining code for sending the email
19 mailserver = smtplib.SMTP('smtp.office365.com', 587)
20 mailserver.starttls()
21 mailserver.login(msg['From'], 'password here')
22 mailserver.send_message(msg)
23 mailserver.quit()

```

2.9.3 Python HTTP Server

Python has an HTTP server built into the standard library.

Serving static files: `python -m http.server 8008` this will look for an index.html file to start serving, and can be accessed on `localhost:8008`.

2.9.4 Python on iOS and Android

This info comes from this article [Medium Article](#)

Python 3.13 has brought official support for iOS and Android as a platform. On iOS, since Apple restricts being able to install system resources or run binaries, developers are forced to run python in embedded mode. In other words, writing native iOS applications and embedding a python interpreter using `libPython`. This bundles the python interpreter and your code into a bundle that gets uploaded to the App store.

Here is the official docs <https://docs.python.org/3/using/ios.html>

3 Git & GitHub

3.1 Git Basics

3.2 Git Remotes

A "remote" in the git environment is another location to store a copy of your git project.

GitHub is the most common remote.

4 Docker

Docker uses images of a software to create containers that run them. They will create their own isolated environment on the host system. They are typically very easy to install and even easier to uninstall. They can enhance security by isolating the host system from the system running the software.

When you find an image you want to use do: `docker pull <image_name>`

To see images downloaded use: `docker images`

See containers: `docker ps`

Run container: `docker run image-name -d` -d means run in detached mode so when the terminal is closed, the container continues to run

Some containers need other info to run, which can be passed after the command: Postgres example: `docker run --name postgres_db -e POSTGRES_PASSWORD=postgres -p 5432:5432 -d postgres`

- --name is what the container will be named
- -p is for port mapping host computer port to image port (first is host, second is container)
- -e is for adding environment variables

Remove an image by name and tag: `docker rmi postgres:latest` or by image id `docker rmi 123456789abc`

4.1 Docker Compose

Docker compose is a handy way to run docker with configuration files. This is nice when you have more complex containers to run, and its a difficult to type all the config every time you want to run it.

Important step: Make sure you have the latest version (currently V2).

Docker compose V1 was built on python, V2 is built in Go. If you run `which docker compose` and it shows the path `/usr/bin/docker-compose` then you still have V1. Remove it with `sudo rm /usr/bin/docker-compose`.

Create a `docker-compose.yaml` file. Here's an example:

```

1  services: # Defines the containers (services)
2  app:
3      image: my_flask_app:latest # Use an existing image
4      build: .. # Or build from Dockerfile in the current directory
5      ports:
6          - "5000:5000" # Map host port 5000 to container port 5000
7      volumes:
8          - ./app:/app # Mount local folder to container folder
9      environment:
10         - FLASK_ENV=development # Set environment variables
11     depends_on:
12         - db # Wait for "db" service before starting
13
14
15     db:
16         image: postgres:15
17         restart: always # Restart if it crashes
18         environment:
19             POSTGRES_USER: user
20             POSTGRES_PASSWORD: password
21             POSTGRES_DB: mydatabase
22         volumes:
23             - pg_data:/var/lib/postgresql/data # Persistent storage for DB
24
25     volumes:
26         pg_data: # Named volume for PostgreSQL data

```

Run container `docker compose up -d`

Stop container `docker compose down`

Upgrade container `docker compose down` `docker compose pull` `docker compose up -d`

Restart container: `docker compose restart`

4.2 Build a Docker Image

A Dockerfile is how to build an image. The contents might look something like this:

Simple python app:

```

1  FROM python:3.12-slim
2  WORKDIR /app
3  COPY requirements.txt .
4  RUN pip install -r requirements.txt
5  COPY . .
6  CMD ["python", "app.py"]

```

Flask app:

```

1  # Use a lightweight Python image
2  FROM python:3.12-slim
3
4  # Set the working directory inside the container
5  WORKDIR /app
6
7  # Copy app files to the container
8  COPY . /app
9
10 # Install dependencies
11 RUN pip install --no-cache-dir -r requirements.txt
12
13 # Set environment variables
14 ENV FLASK_ENV=production
15 ENV PYTHONUNBUFFERED=1
16
17 # Expose the port (optional if using Docker Compose)
18 EXPOSE 8090
19
20 # Run Gunicorn when the container starts
21 CMD ["gunicorn", "-w", "4", "-b", "0.0.0.0:8090", "web_app.app:app"]

```

When your dockerfile is ready, run: `docker build -t my-image-name:tag .`

- Replace my-image-name with the name you want to give your image.
- Replace tag with an optional version (e.g., latest or v1.0).

End example might look like `docker build -t my-flask-app:latest .`

If you are building on a Mac, but will run on Linux, when you build an image, specify to run on amd64: `docker build --platform linux/amd64 -t yt_download:latest .`

Save the image to a tar file: `docker save -o /Users/tyleranderson/Downloads/yt_downloads_250226.tar yt_download:latest`

Load the file as an image: `docker load -i /srv/flask_yt_download/yt_downloads_250226.tar`

4.3 Docker Compose with Custom Images

When you use docker compose with an image, you have two options:

- To build the image with docker compose when its run, use the `build: .` option. This builds the image from the Dockerfile in the directory.

```

1  services:
2    flask_app:
3      build: . # This tells Compose to use the Dockerfile in the current directory
4      container_name: flask_gunicorn
5      ports:
6        - "8090:8090"

```

- If you already built the image with `docker build -t myflaskapp:latest .` then you can tell docker compose what image to use:

```
1  services:
2    flask_app:
3      image: my_flask_app # Use the existing built image
4      container_name: flask_gunicorn
5      ports:
6        - "8090:8090"
```

4.4 Docker Image Files

Save image to file: `docker save -o /path/to/destination/image-name.tar my-image-name:tag`

Load image from file: `docker load -i /path/to/destination/image-name.tar`

4.5 Access Docker Container

Access the container environment `docker exec -it <container-name-or-id> /bin/sh` /bin/sh is the terminal experience (shell in this case) you could do /bin/bash

Access container logs `docker logs <container name or id>`

For continuously showing log output `docker logs -f <container-name-or-id>`

4.6 Docker Volumes

Files and Docker If you need to use files for anything involving docker, you need to configure volumes.

Volumes are a mapping of the docker container file system to the host file system.

```
-v /home/svr/immich:/usr/src/app/upload
```

Just like port mapping, the first is host and second is container.

5 Networking

5.1 Networking

A network has many different things going on.

In networking, devices sit behind routers. Routers have a public IP address that is used in accessing the internet, or a wide area network (WAN). The router also manages local ip address for all devices through DHCP.

5.1.1 DHCP

DHCP - Dynamic Host Configuration Protocol

DHCP is a server responsible for assigning local ip addresses to devices on the network. It can also have local domain names assigned, frequently setup as `home.local` or `local`.

You can use DHCP to reserve certain ip addresses for certain devices. You can also assign names that point to a certain device. This way you can refer to the device on the network by an easy to remember name instead of the ip address. For example 192.168.1.104 -> homeserver.

When a device joins a network, it sends a "DHCP discover" message. A DHCP server responds to the message and assigns an IP address and other configuration information. DHCP also assigns new IP addresses when devices move to new locations.

5.1.2 DNS

DNS - Domain Name System

This translates IP addresses into domain names. Think of a phonebook for the internet. Instead of names to phone numbers it points domain names to ip addresses. There are public DNS servers that route the public domains on the internet.

Local DNS

There are also DNS servers that are run on a router or a local machine. This establishes mappings from names to ip address only on your local network. This makes it so you can setup a device with, say ip `192.168.1.104` to name `mydevice.home`. Keep in mind that your router might have a suffix that attaches to these names, such as `.home`.

5.1.3 IPv4 vs IPv6

IPv4

consists of devices having a 32-bit address that looks like 192.168.1.100. It consists of public and private addresses, where the public addresses are accessible over the internet and private addresses are only accessible over a local network. This means for data to get passed from one device over the internet to another device, the protocol Network Area Translation (NAT) is used. NAT will convert every packet of your private ip to the public which costs compute.

IPv6

- 128-bit addresses, virtually infinite possible addresses.
- Device addresses are globally routable (not hidden behind a public IP).
- Because devices are globally routable, it eliminates the need for NAT as devices can connect end-to-end. Allowing for better connection speeds for video, audio, peer-to-peer, gaming, etc.
- Devices can auto-configure with Stateless Address Autoconfiguration (SLAAC), no need for DHCP.
- Firewalls are more relevant with IPv6 due to addresses being globally routable.

```
ip -6 addr show
```

Local address: **fe80::**

- Scope: Local network segment (cannot be routed on the internet).
- Usage: Used for internal network functions like router discovery and neighbor discovery.
- Common for: Internal device communication.

Global address: Usually starts with **2xxx::** or **3xxx::**

- Scope: Globally routable on the internet.
- Usage: For internet-facing applications or public communication.
- This is often the correct address for external access.

Address types:

- scope global → for internet or external routing.
- scope link → for local network only.

SSH: `ssh user@[fe80::1a2b:3c4d:5e6f%eth0]`

5.1.4 Commands

```
netstat -
```

5.2 Servers

5.2 Web Server

Known for serving over HTTP/HTTPS protocol. Typically websites, API's, web services, anything using HTTP(S).

5.2.1 Proxy Server

Redirects web traffic. A sort of "middleman" server that is responsible for directing requests. For example, Nginx or Caddy can be hosted on a server. The proxy server program, once configured, will push requests to other servers. Good for load balancing, obscuring other servers IP address. Specifically this reroutes web traffic like HTTP(S).

5.2.2 File Server

A file server is a server that is like a central hub to hold files.

Some common options: - Samba <https://hub.docker.com/r/dperson/samba> for your typical file explorer-like experience. - SFTP. - Files over HTTPS aka cloud API's like Google Drive.

5.3 Caddy

Official site: <https://caddyserver.com/>

docker compose:

```
1  services:
2    caddy:
3      image: caddy:latest
4      container_name: caddy
5      ports:
6        - "80:80"          # HTTP
7        - "443:443"
8        - "8009:8009"
9      volumes:
10       - ./Caddyfile:/etc/caddy/Caddyfile # Mount your Caddyfile
11       - ./data:/data                     # Let's Encrypt certificates
12       - ./config:/config                 # Caddy configuration
13       - /srv/web_apps/notes:/srv/web_apps/notes
14      restart: unless-stopped
```

Don't forget to open the port(s) in the firewall.

5.3.1 Reverse-Proxy

Reverse Proxy - software that routes traffic from one endpoint to another, or multiple others.

Caddy is very simple and it comes with built in certificate management through Lets Encrypt.

```
1  <requested-domain> {
2    reverse_proxy <ip and port service is routed to>
3  }
```

Examples using Caddy:

Very simple config. This says using the host machine ip address, using the protocol on port 80 (http) respond with "Hello from Caddy".

```
1  :80 {
2    respond "Hello from Caddy"
3  }
```

Internal and localhost certificates If you configure sites with local or internal addresses, Caddy will serve them over HTTPS using a locally-trusted certificate authority with short-lived, auto-renewing certificates. It even offers to install your unique root into your local trust stores for you.

```

1  localhost {
2      respond "Hello from HTTPS!"
3  }
4
5  192.168.1.10 {
6      respond "Also HTTPS!"
7  }
8
9  http://localhost {
10     respond "Plain HTTP"
11 }

```

When requests go to example.com, it will get routed to this reverse proxy, which then pushes it to localhost:5000. Just be sure the DNS records are updated to route the domain to this IP address.

```

1  example.com {
2      reverse_proxy localhost:5000
3  }

```

Here is Caddy as a reverse proxy doing load balancing, in a round-robin method.

```

1  example.com {
2      reverse_proxy backend1:5000 backend2:5000 backend3:5000
3  }

```

You can do path based proxying to serve different backends based on the url path. In this example `example.com/api` will go to one web server, while `example.com/static` goes to another.

```

1  example.com {
2      reverse_proxy /api backend1:5000
3      reverse_proxy /static backend2:5001
4      reverse_proxy /app backend3:5002
5  }

```

You can setup to route subdomains as well. This will retrieve an SSL certificate for all domains added.

```

1  example.com {
2      reverse_proxy localhost:3000
3  }
4
5  api.example.com {
6      reverse_proxy localhost:4000
7  }

```

You can also proxy to external services.

```

1  example.com {
2      reverse_proxy https://api.example.com
3  }

```

You can also configure domains to redirect to one domain. In this example all requests to `www.example.com` will be rerouted to `example.com`.

```
1 example.com www.example.com {
2     reverse_proxy localhost:3000
3 }
```

To run Caddy:

- Download to computer
- Setup config file `Caddyfile` typically in `/etc/caddy/Caddyfile`
- Start the Caddy server `sudo systemctl start caddy`

To run in Docker:

- Create a `docker-compose.yml` file.

```
1 services:
2   caddy:
3     image: caddy:latest
4     container_name: caddy
5     ports:
6       - "80:80"          # HTTP
7       - "443:443"        # HTTPS
8     volumes:
9       - ./Caddyfile:/etc/caddy/Caddyfile # Mount your Caddyfile
10      - ./data:/data           # Let's Encrypt certificates
11      - ./config:/config       # Caddy configuration
12     restart: unless-stopped
```

- Create your `Caddyfile` (make sure to create it in the same location your volume is pointed to).

```
1 example.com {
2     reverse_proxy backend:3000
3 }
```

- Run `docker compose up -d` to start the server.

Additional considerations to run in Docker:

- If you run Caddy in a docker container, `localhost` will be that container, due to dockers own DNS.
- If you run your web server that you're routing to in a docker container, you can use that container name in the caddy file config.

You can also route ports directly.

```

1  :8443 {
2      reverse_proxy 127.0.0.1:8000
3  }

```

5.3.2 File Server

Caddy has an option to serve static files over HTTP. This is not a file server like sFTP because it serves over http or https.

```

1  healthfin.solutions {
2      root * /srv/website
3      file_server
4  }

```

Serve different sites with different paths of the same domain.

```

1  healthfin.solutions {
2      handle_path /notes* {
3          root * /srv/web_apps/notes
4          file_server
5      }
6
7      reverse_proxy homeserver.home:8000
8  }
9
10 :8009 {
11     root * /srv/web_apps/notes
12     file_server
13 }

```

This config allows for access to the main server from the root domain, but also changes to the static site when you add the /notes path.

Don't forget you can setup a local DNS server on your machine and setup a local domain DNS rewrite.

```

1  anderson.docs {
2      tls internal
3      root * /srv/web_apps/notes
4      file_server
5  }

```

5.4 DNS Server

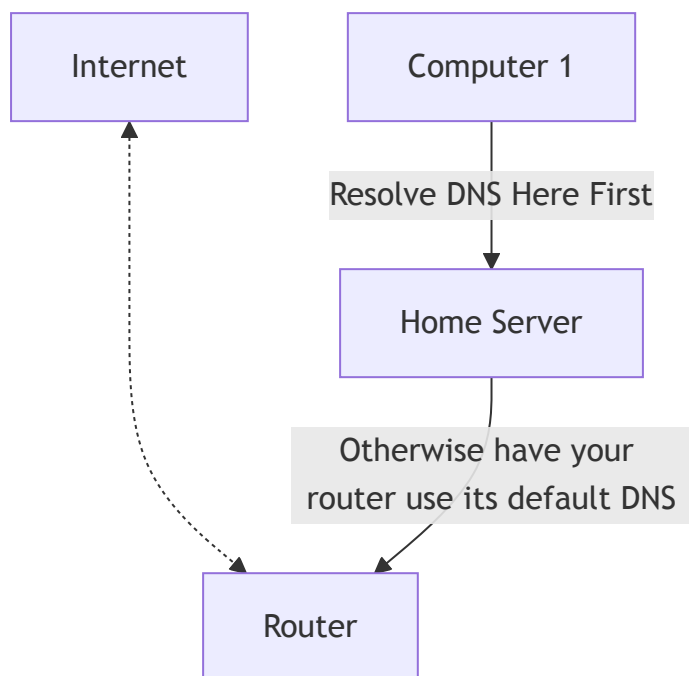
A DNS Server is a server that holds domain name records and point to the corresponding IP addresses.

5.4.1 Adguard

You can setup your own local DNS server on your network. Frequently these allow you to setup your own name resolutions on your network, but also block certain sites from resolving, which can prevent ads.

5.4.2 Concept 1

You can setup your own DNS server on your network. Open source servers include adguard, dnsmasq, and pi-hole.



5.4.3 Options

You have two options when setting up your own DNS server

1. Set up individual devices on your network to use self-hosted DNS server
2. Set up your router to point all devices to the self-hosted DNS server.

5.4.4 Setup Adguard

Docker compose file:


```

1  services:
2      adguardhome:
3          container_name: adguardhome
4          image: adguard/adguardhome
5          restart: unless-stopped
6          ports:
7              - "53:53/udp"
8              - "53:53/tcp"
9              - "3000:80/tcp"    # Web UI
10 #          - "443:443/tcp" # (Optional, for HTTPS UI)
11      volumes:
12          - ./workdir:/opt/adguardhome/work
13          - ./confdir:/opt/adguardhome/conf

```

`sudo docker compose up -d` - you now have a DNS server running.

To access the web UI you can go to `http://<ip address>:3000` .

To add a local domain, go to Filters>DNS rewrites

If you want to do option 1:

Go to the device(s) you want to use your DNS server. Go to System Preferences -> Network -> Wifi (or Ethernet) -> DNS. From there add the IP address of your server running the DNS server.

If you want option 2:

Go into your router options. Find DNS settings (might be under DHCP Settings).

Add your custom server, its a good idea to add a fallback server in case yours goes down. Google is 8.8.8.8 or cloudflare 1.1.1.1.

6 Linux and Other OS

6.1 Linux

6.1.1 Execute files

To execute file: `./path/to/file` or `source /path/to/file`

Example: `./bash_script.sh` or `source /bash_script.sh`

- `./bash_script.sh` executes the script as a standalone process. This method requires execute permissions.
- `source bash_script.sh` executes the script in the existing process (doesn't create standalone process). Allows modifying environment variables in the current shell. Allows `cd` command to change directory.

TLDR: default to using `source` to execute files.

6.1.2 File Permissions

How to check file permissions:

- `ls -l` - view permissions of files in the directory.
- `ls -ld` - view permissions of the directory itself.

You will see something like below:

```
drwxr-xr-x  2 root root 4096 Feb 12 12:34 /srv/web_apps
```

The first section is read, write, execute for owner, group, others.

Section	Meaning	Who it applies to
d	Directory	-
rwX	read(r), write(w), execute(x)	Owner (root)
r-x	read(r), no write(-), execute(x)	Group (root)
r-x	read(r), no write(-), execute(x)	Others (everyone else)

To see if you're root do: `whoami`. If the output is 'root' then you can write. If your user is not root, check if your in the root group `groups`. If your not in 'root' group, you cannot write.

Alter Permissions

Use `chmod`

There are 2 main methods, numeric or symbolic. The numeric mode requires memorizing number codes for altering permissions. Symbolic is more straight forward.

- `u+rw` → Add read, write, execute for owner (`u`).
- `g+r` → Add read for group (`g`).
- `o-r` → Remove read for others (`o`).

Examples:

- Change all permissions: `chmod u+rw,g+r,o-r filename`
- Change only owner permissions: `chmod u+rw filename`
- Change only group permissions: `chmod g-w filename`
- Change only other permissions: `chmod o-r filename`
- Give everyone execute permissions: `chmod +x filename`

Change Ownership

Change the owner of the file to a different user: `sudo chown -R user:user /srv/web_apps`

6.1.3 Copy Directory

Two baked-in commands are `scp` or `rsync` .

Here's an example with scp: `scp -r ./site user@server:/srv/web_apps` .

rsync is typically recommended over scp.

Rsync usually comes on Linux and MacOS.

Commands:

- Basic = `rsync -v /source/file/name.txt /dest/file`
- Use literal string (preserve string for special characters) = `rsync -v '/source/file/name.txt' /dest/file`
- Send multiple files = `rsync -v /source/file1.txt /source/file2.txt '/dest'`
- Send over SSH = `rsync -av /source/file/name.txt user@server:/dest/file`
- Send over SSH with custom port `rsync -avz -e "ssh -p 2222" /source/file/name.txt user@server:/dest/file`
- Send over SSH with sudo command = `rsync -v --rsync-path="sudo rsync" yt_download_image.tar user@server:/srv/flask_yt_download`
- Send over entire directory = `rsync -avz /source/dir /dest/dir`
- Send all files in directory = `rsync -avz /source/dir/ /dest/dir`

- `rsync -avz --rsync-path="sudo rsync" -e "ssh -p 2222" /Users/tyleranderson/Bonus tyler@anderson.home:/home/tyler/backup`
- `rsync -avz --rsync-path="sudo rsync" /Users/tyleranderson/Bonus tyler@anderson.home:/home/tyler/backup`

Flags:

- `-v` = verbose mode.
- `-a` = archive mode. saves permissions and timestamps.
- `-z` = compress files to transfer, lossless compression.
- `-e` = execute specific command.
- `--progress` = show transfer progress.
- `--delete` = remove files from the backup if they were removed from the source.
- `--exclude` = Exclude specific files like `--exclude='*.log'` `--exclude='/cache/'`

Copy directory into another: `rsync -av ./site user@server:/srv/web_apps/notes`

Copy directory contents into another: `rsync -av ./site/ user@server:/srv/web_apps/notes/` (trailing `/` on the source. Trailing source on the destination doesn't matter)

6.1.4 Backups

Options: Backup files/folders OR full system backup.

Backup folders: daily `rsync` to a separate disk Full system: weekly `dd` or `timeshift`

Can setup with cron to run regularly.

6.1.5 Tar Files

A tar file `.tar` is an archive file that stores multiple files and directories together in one file, without compression. Very common in Linux and Unix based systems. Common for backups and transfers.

- Create tar file = `tar -cvf archive.tar /path/to/files/`
- Extract tar file = `tar -xvf archive.tar`
- Create compressed tar file = `tar -czvf archive.tar.gz /path/to/files/`
- Extract compressed tar file = `tar -xzvf archive.tar.gz`
- List files in tar file = `tar -tvf archive.tar`
- Extract specific file from tar file = `tar -xvf archive.tar file.txt`

Flags:

- `-c` = Create new archive
- `-v` = Verbose mode (shows progress)
- `-f` = Specifies filename (archive.tar)
- `-x` = Extract
- `-z` = Compress with gzip

Philosophy

Best practice for backing up files is called 3-2-1 backup strategy.

- 3 copies of your data (original and two backups).
- 2 different storage types (external drive, NAS, cloud storage, etc).
- 1 off-site backup, a cloud backup or physical backup at a friends house.

Encryption

`age` is a lightweight focused encryption software for files. It uses modern cryptography making it more secure, but less featured than something like `gpg`.

Simple example, one file: `age -o secretfile.age -p secretfile.txt`

Simple example, directory (use tar): - `tar -czf myfolder.tar.gz myfolder/` - `age -o myfolder.tar.gz.age -p myfolder.tar.gz`

- MacOS Install: `brew install age`
- Linux Ubuntu Install: `sudo apt install age`

Removing Sensitive Data

Hard-Disk Drives:

Securely delete a file with `shred`. This will overwrite the file with random data multiple times, renames the file multiple times to obscure its original name, then deletes the file. This makes it very difficult for data recovery tools to extract meaningful information from the files.

```
shred -u secret.txt
```

- `-u` = truncate and delete the file after shredding.
- `-n` = number of times to overwrite (default is 3) `shred -n 10 secret.txt`

Solid-State Drives:

1. Use `fstrim` (Best for Full SSD)  Best for clearing free space on an SSD

Modern SSDs support TRIM, which tells the SSD to permanently erase deleted data

```
sudo fstrim -v /
```

1. Use srm or wipe (For File-Level Deletion) ✅ Best for deleting a single file securely (better than `shred` on SSDs)

◆ Install srm (Secure Remove)

```
1 | sudo apt install secure-delete # Debian/Ubuntu
2 | brew install srm               # macOS
```

◆ Securely delete a file = `srm -v my_secret_file.txt`

◆ Use wipe for Directories = `wipe -rf my_secret_folder/`

6.2 SSH

To SSH into a server its `ssh username@ipaddress` or `ssh username@servername` .

SSH uses port 22.

Typically when you SSH into a system, it will ask for your password. Instead of using a password everytime, you can create a ssh key. These keys are essentially a file that sits on both machines and grants access without needing a password.

To generate ssh keys:

1. On local machine `ssh-keygen -t rsa -b 4096` . It will ask for a passphrase, but you can leave it empty.
2. Copy the ssh key to the server `ssh-copy-id user@server`

If it worked correctly, you should now be able to login without needing a password.

You can run commands through ssh, without logging in: `ssh user@server "sudo chown -R user:user /srv/web_apps"`

6.3 Backups

Options: Backup files/folders OR full system backup.

Backup folders: daily `rsync` to a separate disk Full system: weekly `dd` or `timeshift`

Can setup with cron to run regularly.

6.3.1 Philosophy

Best practice for backing up files is called 3-2-1 backup strategy.

- 3 copies of your data (original and two backups).
- 2 different storage types (external drive, NAS, cloud storage, etc).
- 1 off-site backup, a cloud backup or physical backup at a friends house.

6.3.2 Copy Directory

Two baked-in commands are `scp` or `rsync` .

Here's an example with scp: `scp -r ./site user@server:/srv/web_apps .`

rsync is typically recommended over scp.

`rsync` usually comes on Linux and MacOS, it is lightweight, fast, can detect changes in files to do an incremental backup.

Commands:

```

1 # Basic save file to another folder
2 rsync -v /source/file/name.txt /dest/file
3
4 # Use literal string (preserve string for special characters)
5 rsync -v '/source/file/na$me.txt' /dest/file
6
7 # Send multiple files
8 rsync -v /source/file1.txt /source/file2.txt '/dest'
9
10
11 # Send over SSH
12 rsync -av /source/file/name.txt user@server:/dest/file
13
14 # Send over SSH with custom port
15 rsync -avz -e "ssh -p 2222" /source/file/name.txt user@server:/dest/file
16
17 # Send over SSH with sudo command
18 rsync -v --rsync-path="sudo rsync" yt_download_image.tar
19 user@server:/srv/flask_yt_download
20
21 # Send over entire directory
22 rsync -avz /source/dir /dest/dir
23
24 # Send all files in directory. Trailing slash.
25 # Trailing slash only matters on source, it doesn't matter on dest
26 rsync -avz /source/dir/ /dest/dir
27
# actual use
rsync -avz --rsync-path="sudo rsync" /Users/tyleranderson/PycharmProjects
tyler@anderson.home:/srv/backup_media

```

Flags:

- **-v** = verbose mode.
- **-a** = archive mode. saves permissions and timestamps.
- **-z** = compress files to transfer, lossless compression.
- **-e** = execute specific command.
- **--progress** = show transfer progress.
- **--delete** = remove files from the backup if they were removed from the source.
- **--exclude** = Exclude specific files like **--exclude='*.log'** **--exclude='/cache/'**

6.3.3 Tar Files

A tar file **.tar** is an archive file that stores multiple files and directories together in one file, without compression. Very common in Linux and Unix based systems. Common for backups and transfers.

```

1 # Create tar file (best to use relative path instead of absolute)
2 sudo tar -cvf archive.tar path/to/files/
3
4 # Extract tar file
5 sudo tar -xvf archive.tar
6
7 # Create compressed tar file
8 tar -czvf archive.tar.gz /path/to/files/
9
10 # Extract compressed tar file
11 tar -xzvf archive.tar.gz
12
13 # List files in tar file
14 tar -tvf archive.tar
15
16 # Extract specific file from tar file
17 tar -xvf archive.tar file.txt

```

Flags:

- **-c** = Create new archive
- **-v** = Verbose mode (shows progress)
- **-f** = Specifies filename (archive.tar)
- **-x** = Extract
- **-z** = Compress with gzip

6.3.4 Encryption

age is a lightweight focused encryption software for files. It uses modern cryptography making it more secure, but less featured than something like **gpg**. Age lets you use passwords or key files.

Install:

```

1 # Install
2 brew install age # MacOS
3 sudo apt install age # Linux

```

How to use - you have 2 options for encryption.

1. Password - best for quick encryption, no need to manage keys. Less secure and harder to automate.

```

1 # encrypt with password, prompt will ask for password
2 age -o myfile.txt.age -p myfile.txt
3
4 # decrypt with password
5 age -d myfile.txt.age > myfile.txt

```

1. Public/Private key Encryption - Most secure and good for automation.

```
1 # generate a key file
2 age-keygen -o ~/.age-key.txt # stores key in ~/.age-key.txt
3 # This file contains a public and private key that you can use to encrypt files.
4
5 # encrypt file using public key (replace PUBLIC_KEY with your public key in the file)
6 age -r PUBLIC_KEY -o myfile.txt.age myfile.txt
7
8 # decrypt using private key
9 age -d -i ~/.age-key.txt myfile.txt.age > myfile.txt
```

To use the file directly instead of copying the public key:

```
1 # use head to extract the public key
2 age -r $(head -n 1 ~/.age-key.txt) -o myfile.txt.age myfile.txt
3
4 # decrypt
5 age -d -i ~/.age-key.txt myfile.txt.age > myfile.txt
```

Encrypt directory by turning into tar file, then encrypting that file.

```
1 # Simple example, directory (use tar).
2 tar -czf myfolder.tar.gz myfolder/
3 age -o myfolder.tar.gz.age -p myfolder.tar.gz
4
5 # convert folder to tar file, encrypt it with age, one command,
6 # no intermediate file created.
7 tar -czf - myfolder/ | age -o myfolder.tar.gz.age -p
8
9 # decrypt and extract in one command.
10 age -d myfolder.tar.gz.age | tar -xz
```

Decrypt:

```
1 # decrypt .age file
2 age -d myfolder.tar.gz.age > myfolder.tar.gz
3
4 # extract the file
5 tar -xzf myfolder.tar.gz
6
7 # delete archive file
8 shred -u myfolder.tar.gz
```

Flags:

- **-o <output>** = output file name.
- **-d** = decrypts an encrypted file.
- **-p** = uses password encryption.

- `-r <public-key>` = Encrypts using public key.
- `-i <identity-file>` = uses private key for decryption.

Other notes:

- The key file contains both public and private keys.
- Encryption uses only the public key.
- Decryption uses both keys

6.3.5 Removing Sensitive Data

Hard-Disk Drives:

Securely delete a file with `shred`. This will overwrite the file with random data multiple times, renames the file multiple times to obscure its original name, then deletes the file. This makes it very difficult for data recovery tools to extract meaningful information from the files.

```
shred -u secret.txt
```

- `-u` = truncate and delete the file after shredding.
- `-n` = number of times to overwrite (default is 3) `shred -n 10 secret.txt`

For entire directories, you can use `wipe`.

`sudo apt install wipe` and then `wipe -r /path/to/directory/` this will remove the directory as well. If you want to keep the directory use `wipe -r /path/to/directory/*`

Solid-State Drives:

1. Use `fstrim` (Best for Full SSD) ✅ Best for clearing free space on an SSD

Modern SSDs support TRIM, which tells the SSD to permanently erase deleted data

```
sudo fstrim -v /
```

1. Use srm or wipe (For File-Level Deletion) ✅ Best for deleting a single file securely (better than `shred` on SSDs)

◆ Install srm (Secure Remove)

```
1 sudo apt install secure-delete # Debian/Ubuntu
2 brew install srm               # macOS
```

◆ Securely delete a file = `srm -v my_secret_file.txt`

◆ Use wipe for Directories = `wipe -rf my_secret_folder/`

7 Video Processing

FFmpeg

ffmpeg is a video editing command tool.

How a Video Works

1. A video file consists of a container (like MP4, MKV) and encoded video/audio streams inside.
2. The codec is what encodes (compresses) the video/audio when saving it and decodes (decompresses) it when playing it.

Converting a Video File (Transcoding)

If you want to convert a video file from one format to another (e.g., MKV to MP4 or H.264 to H.265), you use transcoding software (like HandBrake, FFmpeg, or VLC). The process involves:

1. Decoding (using the current codec).
2. Re-encoding (using a new codec or settings).
3. Packaging the video/audio into a new container format.

For example:

- MKV (H.264 + AAC) → MP4 (H.264 + AAC) → Just a container change (remuxing, no transcoding needed).
- MP4 (H.264) → MP4 (H.265) → A full re-encoding is required.

So while codecs are a part of the conversion process, they don't handle file conversion by themselves—you need transcoding software for that.

- Codec - software that compresses or decompresses media files (audio or video). Determines the encoding. Important for compression, compatibility, and streaming efficiency.
- Video Codecs - These compress video data to reduce file size while maintaining visual quality.
- H.264 (AVC) – Most common, widely supported, good balance of quality and size.
- H.265 (HEVC) – More efficient than H.264 (better quality at smaller sizes), but requires more processing power.
- VP9 – Open-source alternative to H.265, widely used on YouTube.
- AV1 – Newer, even better compression than VP9 and H.265, but requires more CPU power to decode.
- MPEG-2 – Older, used in DVDs and some broadcast TV.
- Audio Codecs - These compress audio data for storage and streaming.
- AAC – Common in MP4 files, better quality than MP3 at the same bitrate.
- MP3 – Universal compatibility, but lower efficiency.
- Opus – High efficiency, used in VoIP and streaming.

- FLAC – Lossless, used for high-quality music storage.
- Dolby Digital (AC-3) / DTS – Used in surround sound systems.

Codec vs. Container (MKV, MP4)

- Container formats (like MKV, MP4, AVI, MOV) hold video, audio, and subtitle tracks.
- Codecs determine how the actual video and audio inside the container are compressed and played.
- Example: An MP4 file can contain H.264 video + AAC audio.

So, when choosing a video format, you need to consider both the container (MKV, MP4, etc.) and the codec (H.264, H.265, etc.).

Streaming

The **"best" streaming protocol** depends on the use case. Here's a breakdown of what's ideal for different scenarios:

Best Streaming Protocols by Use Case

On-Demand Video Streaming (e.g., Netflix, YouTube)

- **Best:** DASH (MPEG-DASH) or HLS
- **Why?** Adaptive bitrate streaming (ABR) ensures smooth playback across different devices and network conditions.
- **Used By:** Netflix, YouTube (DASH), Apple (HLS), Amazon Prime Video (DASH + HLS)

Live Streaming (e.g., Twitch, YouTube Live)

- **Best:** HLS (Low-Latency), DASH (LL-DASH), or WebRTC for real-time interactivity
- **Why?** HLS and DASH offer scalability, while WebRTC provides ultra-low latency for live interaction.
- **Used By:** Twitch (RTMP for ingest, HLS for playback), YouTube Live (DASH + HLS)

Real-Time Communication (e.g., Zoom, Google Meet)

- **Best:** WebRTC
- **Why?** Peer-to-peer, ultra-low latency, no need for external servers.
- **Used By:** Google Meet, Zoom, Microsoft Teams

Professional Broadcast / Contribution (e.g., TV Networks, Sports)

- **Best:** SRT or RTMP (for ingest)
- **Why?** SRT provides reliable, high-quality, low-latency transmission, while RTMP is still widely supported for ingestion.
- **Used By:** ESPN, news networks, live sports broadcasting

What Does Netflix Use?

Netflix uses **MPEG-DASH** for most of its streaming, alongside **HLS for Apple devices**. Their stack includes:

- **DASH (MPEG-DASH)** – Adaptive bitrate streaming for optimal playback across devices.
- **HLS (HTTP Live Streaming)** – For Apple devices, since Apple mandates HLS.
- **CDN Optimization** – Uses its own **Open Connect CDN** to reduce latency and improve video delivery.
- **Per-Title Encoding** – Dynamically adjusts compression based on the content to optimize quality vs. bandwidth.

So HLS can be used to stream an mp4 file to another device?

Yes, **HLS (HTTP Live Streaming)** can be used to stream an **MP4 file** to another device, but it requires some processing first.

How HLS Streams an MP4 File

Segment the MP4 File

- HLS requires the video to be split into small chunks (typically `.ts` files, though newer versions support fragmented MP4 `.m4s`).
- A tool like **FFmpeg** can do this.

Create an M3U8 Playlist

- HLS uses a `.m3u8` file (a playlist) that lists the segmented video files.
- The player downloads and plays the segments in order.

Host the Segments & Playlist on a Server

- You need an **HTTP server** (e.g., **Nginx**, **Apache**, or a cloud service like **S3**) to serve the files.

Play the Stream on Another Device

- Open the `.m3u8` URL in a compatible player (e.g., **Safari**, **VLC**, **ExoPlayer**, or **hls.js in a browser**).

Example: Convert MP4 to HLS Using FFmpeg

```
1 ffmpeg -i input.mp4 -codec: copy -start_number 0 -hls_time 10 -hls_list_size 0 -f hls output.m3u8
```

- `-hls_time 10` → Splits video into **10-second** segments. - `-hls_list_size 0` → Keeps all segments in the playlist. - `output.m3u8` → The playlist file.

How to Serve the Stream

1. Move the `output.m3u8` and `.ts` files to a web server.
2. Access the stream via:

```
1 http://yourserver.com/path/output.m3u8
```

3. Play it in:

Web browser with `hls.js` OR **VLC media player** (Open Network Stream) OR **iOS Safari** (built-in HLS support)

DASH Command with FFmpeg

Run:

```
1 ffmpeg -i vid_file.mp4 -codec: copy -map 0 -f dash output.mpd
```

Explanation of Flags:

- `-i vid_file.mp4` → Input video file.
- `-codec: copy` → Copies the streams without re-encoding.
- `-map 0` → Ensures all streams (audio & video) are included.
- `-f dash` → Specifies DASH output format.
- `output.mpd` → The DASH manifest file.

DASH Output Files

This will generate:

- `output.mpd` → The DASH manifest (index) file.
- **Segmented** `.m4s` video/audio chunks.

Serving DASH with Python HTTP Server

Just like HLS, you can serve DASH using:

```
1 python3 -m http.server 9000
```

Then play in VLC:

```
1 http://localhost:9000/output.mpd
```

✓ **VLC supports DASH**, so this should work without additional configuration.

Play DASH in a Browser

Browsers don't natively support DASH, so you need a **DASH player** like [dash.js](#).

To embed DASH in HTML, use:

```
1 <video id="dashPlayer" controls>
2   <source src="http://localhost:9000/output.mpd" type="application/dash+xml">
3 </video>
4 <script src="https://cdn.dashjs.org/latest/dash.all.min.js"></script>
5 <script>
6   var player = dashjs.MediaPlayer().create();
7   player.initialize(document.querySelector("#dashPlayer"),
8   "http://localhost:9000/output.mpd", true);
</script>
```

Advanced DASH: Multi-Bitrate Encoding

To **support adaptive streaming**, generate multiple bitrates:

```
1 ffmpeg -i vid_file.mp4 \
2   -map 0:v -map 0:a \
3   -b:v:0 2000k -b:v:1 1000k \
4   -s:v:0 1920x1080 -s:v:1 1280x720 \
5   -c:v libx264 -c:a aac \
6   -f dash -seg_duration 4 -use_timeline 1 -use_template 1 \
7   output.mpd
```

✓ This creates multiple video quality options that DASH players can switch between dynamically.

Summary

- Basic DASH: `ffmpeg -i vid_file.mp4 -codec: copy -map 0 -f dash output.mpd`
- Serve DASH: `python3 -m http.server 9000`
- Play in VLC: `http://localhost:9000/output.mpd`
- Play in Browser: Use `dash.js`
- Adaptive Streaming: Encode multiple bitrates with `-b:v` and `-s:v`

8 AI

8.1 Ollama

Ollama is an open-source LLM runtime environment.

Simply put:

1. Run Ollama
2. Download the model you'd like to use (mistral, llama3, etc.)
3. Go to the WebUI
4. Select the model you want to use.
5. Chat away!

Download a model: `docker exec -it ollama ollama pull mistral` or `docker exec -it ollama ollama pull llama3`