API and Python training

Session 3

This session agenda

- Why Python
- IDE - Integrated Development Environment
- Python virtual environments
- Variables
- Simple data types – integer, float, string, bool
- Data types conversion, output, f-strings
- Complex data types – dictionaries and lists
- Practice

Why Python

We'll be using Python to consume API because:

- Easy to learn
- Multi-platform and portable, you can run it the same code on different platforms
- A lot of excellent libraries and SDKs allowing you to focus on your app development, instead of low-level comms and data handling
- Easy to make requests, parse responses,

manipulate data, implement logic – if …then … else

- Broad community to support, easy to find answer

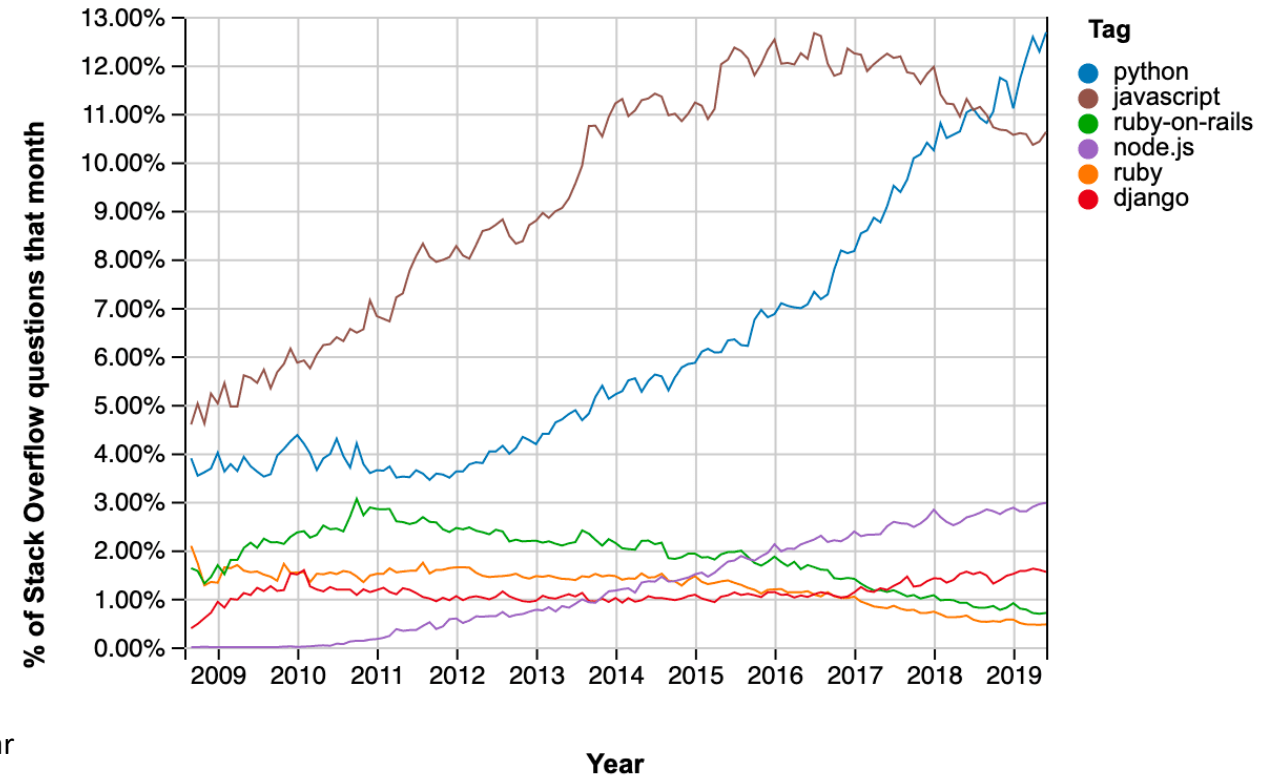
to your question online, for example,

stackoverflow.com

- Many network vendors build modules for Python
- Popular automation tools and frameworks build

using Python – Ansible, NAPALM, Nornir

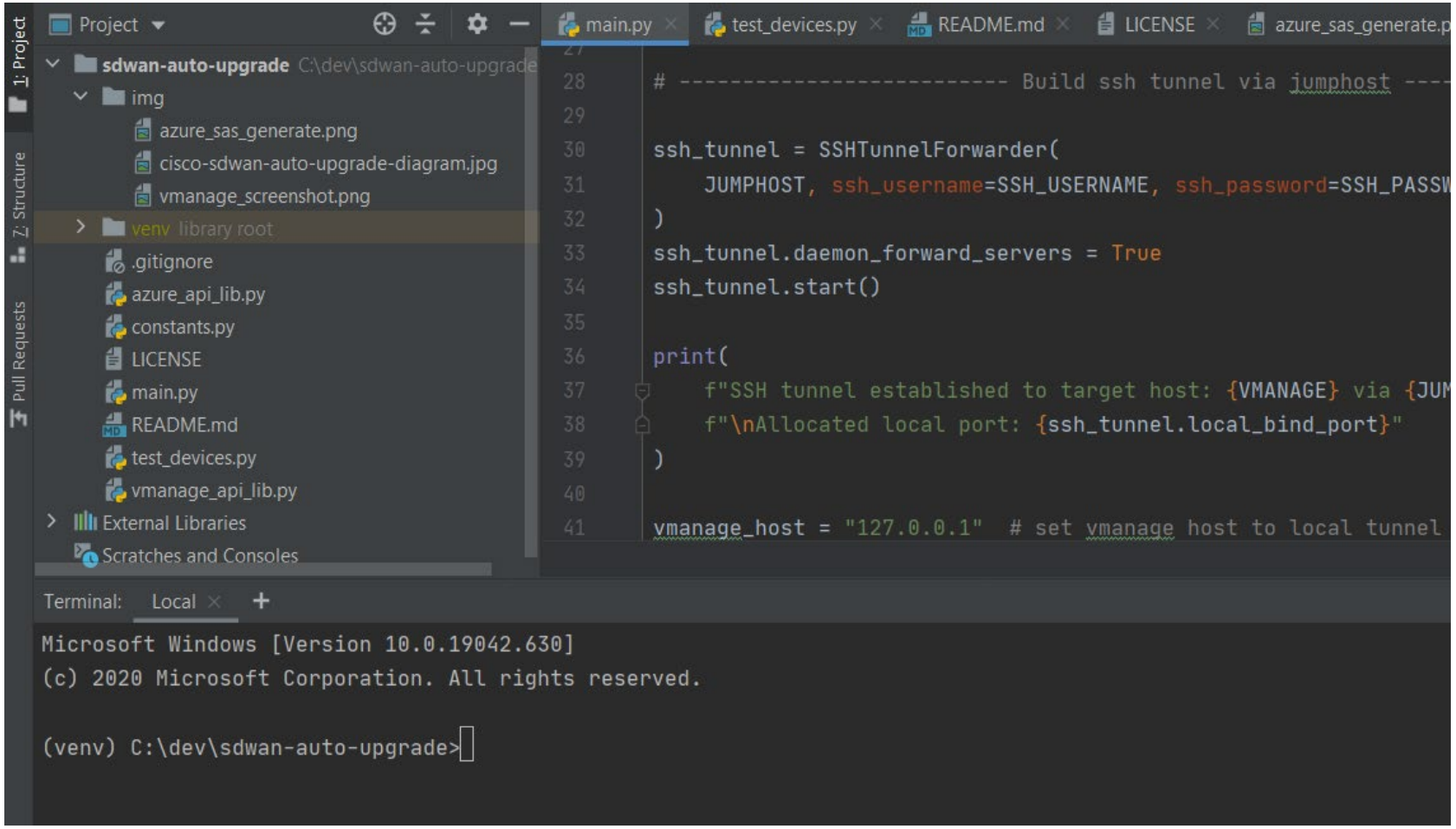
- Most popular programming language in the world



https://www.monterail.com/blog/what-is-python-and-why-is-it-so-popular

IDE

- Multiple vendors – do your research and choose your favorite
- Syntax highlighting
- Auto-completion
- Automatic dependency install
- Error checking, suggestions
- Project and folders navigation
- Plugins, integration with third-party tools, such as Git, Docker, AWS, etc.
- Easier debugging
- Terminal window



Python Virtual Environments (venv)

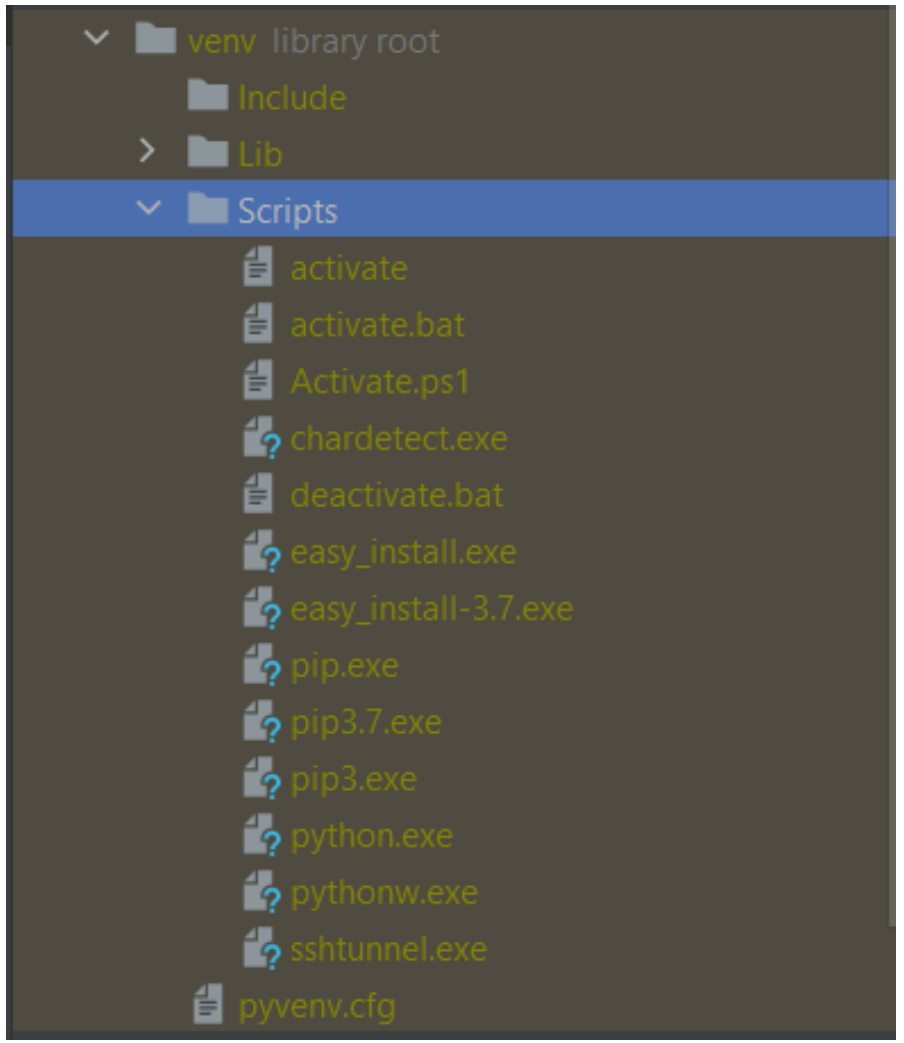
- Isolate different projects from each other
- Avoid dependency conflicts
- When you create venv, a separate folder is created, default system Python version is copied to scripts or bin directory
- All libraries are installed to Lib directory
- To switch between venv, use activate command, the CLI prompt will change to (venv)

```
(venv) C:\dev\sdwan-auto-upgrade>
```

- Create and switch to venv:

Windows
python3 -m venv <env.name>
cd <env.name>
Scripts\activate.bat

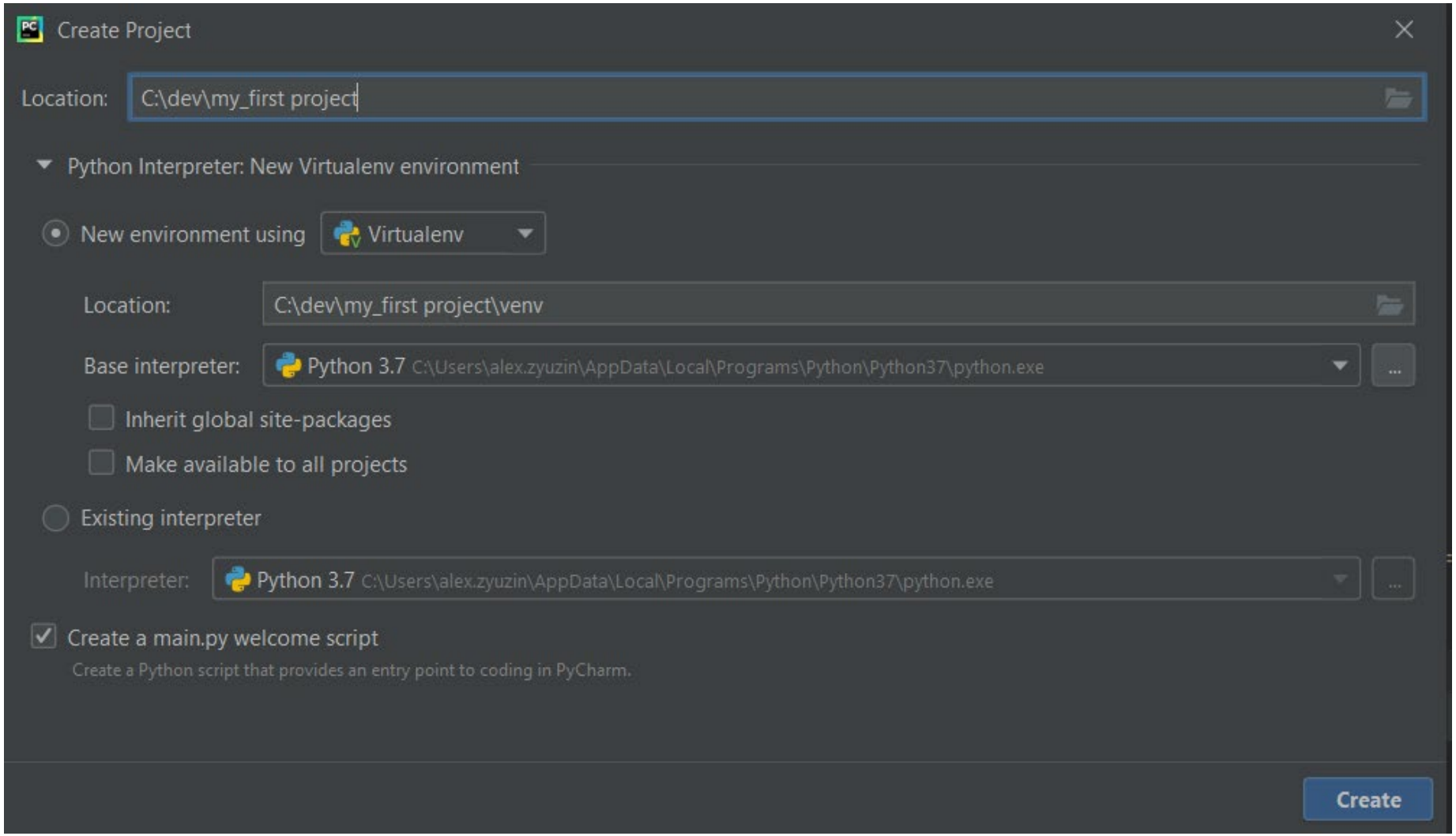
Linux
mkdir <env.name>
cd <env.name>
python3 -m venv .
source bin/activate



Python Virtual Environments (venv)

When you create a project, IDE creates venv automatically for you

---------->



Python variables

- Variables are not declared in advance
- You don't have to define variable type (but you can)
- You simply introduce them in the code, like this:

```
a = 5
b = 10
c = a + b
d = c
```

- Important about **=** sign, this is assignment, not equal
- Case sensitive
- Give variables meaningful names, in Python names usually substitute spaces with underscores e.g. *my_cool_variable*

https://realpython.com/python-pep8/

- All variables can be changed, there is no concept as constants, variable you can't change, so you often can see CAPITAL_LETTER_VARIABLES, which means you be careful and shouldn't change them, but you can.
- There are some data types called immutable which you can't change, such as tuples

Data types

- ALL variables have a type
- Types determine how variables are stored in memory and what operations are allowed
- It's important to know what type it is
- Simple data types – integer, float, string, boolean
- **int** - Integers, like in previous example, can have minus sign
- **float** - 3.14159
- **str** – string- enclosed in single quotes or double quotes
- **bool** - True or False – note the values are Capitalised

- Not a data type, but comments can be single-line starts with #

or multiple lines enclosed in triple quotes or double quotes

```
'''
this is multiple line comment
examples below show different types
'''
# this variable shows integer type
response_code = 200
# this variable shows float type
pi = 3.14
# this variable shows Boolean type
is_correct = True
# this variable shows sting type
string1 = 'my first string'
string2 = "my first string"
```

Output. Type conversion

- Use **print** command to output. Easy:
- You always print string – note it's in quotes

```
print('This will print something to terminal')
```

- Use **+** which is **string concatenation**

```
    pi = 3.14 + string1
TypeError: unsupported operand type(s) for +: 'float' and 'str'
```

- You can't make some operations on different data types

- In some cases you need to convert data types

Need to convert float to string first

Note there is no space between two strings, add leading space manually to make the output pretty

```
14    string1 = 'is my favourite number'
15    pi = str(3.14) + string1
16    print(pi)
```

```
file_1
C:\dev\test_pr1\venv\Scripts\python.exe C:/dev/test_pr1/file_1.py
3.14is my favourite number
```

f-strings

- Powerful feature available from Python 3.6 are format-strings
- **f** and then quotes or double quotes
- Does automatic types conversion
- Supports special characters
- Supports operations

inside the f-string

- Produces just a string
- We'll use it often

```
string1 = 'is my favourite number'
pi = 3.14
circle_radius = 20
tell_the_truth = True

print(f' This will print something to terminal \n {pi} {string1}\n and this is {tell_the_truth}\n '
      f'It can calculate circle circumference like this {2*circle_radius*pi}')
```

file_1

```
C:\dev\test_pr1\venv\Scripts\python.exe C:/dev/test_pr1/file_1.py
 This will print something to terminal
 3.14 is my favourite number
 and this is True
 It can calculate circle circumference like this 125.60000000000001

Process finished with exit code 0
```

https://realpython.com/python-f-strings/

Complex Data Types - List

- Multiple data types, we'll be using **lists** and **dictionaries**, check another two types used often - **tuples** and **sets**

Lists – also called sequence, sometimes one-dimensional array

- Ordered sequence of simple variables
- To **define** variable use square brackets **[ ]**
- Elements can be different types
- You can include variables
- To **access** an element use [<element number>]
- Fist element is **0**, to count from the end, use negative

```
10    some_element = 'another element'
11    my_example_list1 = ['1st element', '2nd element', 50, some_element]
12    print(my_example_list1[0])
13    print(my_example_list1[-1])
```

```
file_1
C:\dev\test_pr1\venv\Scripts\python.exe C:/dev/test_pr1/file_1.py
1st element
another element
```

source: imgur

Complex Data Types - Dictionary

- Unordered key-value pairs, also called hash-tables
- To access **value** you use a **key**
- Keys must be unique
- To **define** the dictionary use {}
- To **access** a value use the key, which is enclosed in [ ] - note it is similar to lists where you also use [ ] to access an element

```
14
15  eth0_interface_definition = {'interface_name': 'eth0', 'ip_addr': '10.10.34.8',
16                               'netmask': 24, 'active': True}
17
18  print('Interface ' + eth0_interface_definition["interface_name"] + ' has IP addresses: ' +
19        eth0_interface_definition["ip_addr"])
```

```
file_1
C:\dev\test_pr1\venv\Scripts\python.exe C:/dev/test_pr1/file_1.py
Interface eth0 has IP addresses: 10.10.34.8
```

Next is the most important slide

Nested complex data types

- Very often you'll see combination of complex data types, this is what **will be returned as API response**
- For example, dictionaries which use lists as values, lists of lists, lists with dictionaries as elements, etc
- It's important to learn how to access them

list_of_interfaces[0]["ip_addr"][1]

1. Using list_of_interfaces[0] you get variable eth0_interface_definition

2. eth0_interface_definition["ip_addr"] retrieves the list ['10.10.34.8', '10.30.34.8']

3. [1] gives you the second element in the List which is the string

```
22  eth0_interface_definition = {'interface_name': 'eth0', 'ip_addr': ['10.10.34.8', '10.30.34.8'],
23                               'netmask': 24, 'active': True}
24  eth1_interface_definition = {'interface_name': 'eth1', 'ip_addr': '10.20.4.80',
25                               'netmask': 25, 'active': False}
26
27  list_of_interfaces = [eth0_interface_definition, eth1_interface_definition]
28
29  print('Interface ' + list_of_interfaces[0]["interface_name"] + ' has IP addresses:\n' +
30        list_of_interfaces[0]["ip_addr"][0] + ' and ' + list_of_interfaces[0]["ip_addr"][1])
31
```

```
file_1
C:\dev\test_pr1\venv\Scripts\python.exe C:/dev/test_pr1/file_1.py
Interface eth0 has IP addresses:
10.10.34.8 and 10.30.34.8
```

Demo

Summary and next steps

- Summary

Python – IDE, venv, variables, simple and complex data types

- Homework

Try simple data types

Try f-strings

Build your own lists and dictionaries

Build list of dictionaries, list of lists, complex dictionaries, the more complex the better

Try to access data, print it using f-strings

- Next time

Control structures – if, for, when

Importing third-party libraries

Requests and JSON libraries

Make first API call with Python