Hands-on Python Foundations

Day 3:

Real-world applications

Outline

Today's topics

- Pip
- Virtual environments
- Setting up projects
- Classes
- Dunder methods
- Modules and packages
- Code hosting

Interactive Scenarios – Day 3

7. Virtual Environments and Pip

- https://learning.oreilly.com/scenarios/hands-on-python-foundations/9780137904648X007/

8. Intro to Classes

- https://learning.oreilly.com/scenarios/hands-on-python-foundations/9780137904648X008/

9. Modules and Packages

- https://learning.oreilly.com/scenarios/hands-on-python-foundations/9780137904648X009/

Schedule

0:00 - Review homework, Q&A

0:20 - **Virtual environments and Pip**- Interactive scenario #7

0:45 - Setting up projects

1:15 - *Break* (15 mins)

1:30 - **Intro to Classes**- Interactive scenario #8

2:30 - *Break* (15 mins)

2:45 - **Modules and Packages**- Interactive scenario #9

3:30 - Code hosting + more

3:55 - Wrap up (5 min)

Questions and breaks

- I'll answer attendee chat throughout class
- Q&A widget during the breaks

- 2 Breaks (15 mins each)

- Email more in-depth questions at arianne.dee.studios@gmail.com

Class setup

- Create a PythonAnywhere account
  - https://www.pythonanywhere.com/

Homework review

Automated daily email

Good morning, Arianne 👋

Today's weather ☁️

Today there will be light snow.
High: 5.9 °C (42.6 °F)
Low: -0.9 °C (30.3 °F)

☀️ 7:48 AM
🌙 5:16 PM

Joke of the day 🤣

Little Johnny comes home from his first day of school. His mother asks, "So, what did you learn at school today?" Little Johnny replies, "NOT ENOUGH. They want me to come back tomorrow!"

Quote of the day 💬

When you recover or discover something that nourishes your soul and brings joy, care enough about yourself to make room for it in your life.
~ Jean Shinoda Bolen

Next game 🏀

In 2 days @ 7:30 PM ET
Home: Toronto Raptors
Away: Atlanta Hawks

Project

- Find some free APIs to retrieve data from
  - weather
  - quote/joke of the day
  - calendar, reminders, todos
  - sports, stocks, events

- Send email every morning

- Basic – plain text email
- Advanced – HTML (formatted) email

Week 2 homework

1. Copy your code from **homework_1.py** to **homework_2.py**
2. Accept the name as a script argument instead of input
3. Retrieve weather (and more) data from API
4. Retrieve some data from a file (todo/reminder/other)
5. Create a **new Gmail account** to send your emails from

Review

Covered Keywords – Week 2

False	await	else	import	pass
None	break	except	in	raise
True	class	finally	is	return
and	continue	for	lambda	try
as	def	from	nonlocal	while
assert	del	global	not	with
async	elif	if	or	yield

Data structures

Name	type	New object	Get item	Main features
List	`list`	[1, 2, 3]	a_list[0]	Ordered Duplicates
Dictionary	`dict`	{'a': 1, 'b': 2, 'c': 3}	a_dict['a']	No order No duplicate keys
Tuple	`tuple`	(1, 2, 3)	a_tuple[0]	Ordered Duplicates Immutable
Set	`set`	{1, 2, 3}	a_set.pop()	No duplicates No order

Testing functions

```
def a_function(n):
    pass

assert a_function(1) == 1
```

- Can create function "stubs"
  - Create function signature (first line)
  - Use **pass** in body to create empty function

- Use **assert** keyword with **pytest** to test functions

Covered Keywords – Week 2

False	await	else	import	pass
None	break	except	in	raise
True	class	finally	is	return
and	continue	for	lambda	try
as	def	from	nonlocal	while
assert	del	global	not	with
async	elif	if	or	yield

Exceptions

```
try:
    temp = int(a_str)
except ValueError:
    print('Invalid number')
```

- Can have multiple except cases for different exception types

More exceptions

```
try:
    temp = int(a_str)
except ValueError as e:
    raise CustomError(e)
else:
    print('Valid number')
finally:
    print('Always gets here')
```

- Retrieve error with **as**
- Throw errors with **raise**

Covered Keywords – Exceptions

False	await	else	import	pass
None	break	except	in	raise
True	class	finally	is	return
and	continue	for	lambda	try
as	def	from	nonlocal	while
assert	del	global	not	with
async	elif	if	or	yield

Working with files

```
with open('filename.txt', 'r') as file:
    file.read()
```

- Use **with** so that file is closed when outside code block
- Different modes:
  - r - read
  - w - write
  - a - write by appending to end of file
  - t - text files
  - b - binary files (e.g. images)

Covered Keywords – Files

False	await	else	import	pass
None	break	except	in	raise
True	class	finally	is	return
and	continue	for	lambda	try
as	def	from	nonlocal	while
assert	del	global	not	with
async	elif	if	or	yield

Requests

```
import requests
response = requests.get('http://test.com')
if response.status_code == 200:
    print(response.text)
```

- Status codes:
  - 2XX – Success
  - 4XX – Client error (you did something wrong)
  - 5XX – Server error (something's wrong but out of your control)
- **response.text** gets content as a string
- **response.json()** gets content as Python dict/list

Week 3 topics

Today's topics

- ❑ Pip
- ❑ Virtual environments
- ❑ Setting up projects
- ❑ Classes
- ❑ Dunder methods
- ❑ Modules and packages
- ❑ Code hosting

Covered Keywords – Week 3

False	await	else	import	pass
None	break	except	in	raise
True	class	finally	is	return
and	continue	for	lambda	try
as	def	from	nonlocal	while
assert	del	global	not	with
async	elif	if	or	yield

Pip and virtual environments

Today's topics

- ❑ **Pip** 
- ❑ Virtual environments
- ❑ Setting up projects
- ❑ Classes
- ❑ Dunder methods
- ❑ Modules and packages
- ❑ Code hosting

References

Interactive scenario

- **Pip and virtual environments**
  - Scenario 7 steps 2 - 4

Video

- **Install external libraries using pip**
  - Next Level Python lesson 3.2

Pip

- PIP – **P**ackage **I**nstaller for **P**ython
  - or **P**ip **I**nstalls **P**ackages

- Installs external packages to a specific Python version
  - `<python_path>/lib/python3.x/site_packages/`

- Looks for packages in PyPI (**Py**thon **P**ackage **I**ndex)
  - Can install from git repository or local directory

Basic commands

- **Install package(s)**
  - `$ pip install <package>`
  - `$ pip install <package1> <package2>`
- **Uninstall package**
  - `$ pip uninstall <package>`
- **List all installed packages**
  - `$ pip list`

Check pip version

- `pip --version`
- `pip3 --version`
- `pip3.10 --version`

From Python module

- `python -m pip –version`

If pip is not installed

- https://pip.pypa.io/en/stable/installation/
- `C:> py get-pip.py`

More advanced **pip** commands

- **Install specific package**
  - `$ pip install <package>` # latest version
  - `$ pip install <package>==1.0.4` # specific version
  - `$ pip install '<package>=1.0.4'` # minimum version
- **Upgrade package to newest version**
  - `$ pip install -U SomePackage`
- **Install with proxy**
  - `pip install --proxy proxy.server:port package`
  - or `--proxy [user:passwd@]proxy.server:port`

Requirements.txt files

- **Create text file with all installed packages + versions**
  - `$ pip freeze > requirements.txt`

- **Install all packages + versions from text file**
  - `$ pip install -r requirements.txt`

Requirements.txt options

- Only include packages directly imported in your project or move the dependencies to a section at bottom
- Can use version ranges, e.g. `requests>=2.20`
- Add inline comments with links to package website
- Split it up into different files for dev, prod, etc...
  - E.g. including `-r base.txt` in a file, like **requirements-dev.txt** will install everything **base.txt** as well

pip vs conda

- **Conda** is a package and environment manager for Anaconda
  - Popular in data science
- Installs from Anaconda or Conda-Forge repos, not PyPI
- Can install non-Python packages (like R or C)
- **Mamba** is a faster alternative to Conda
- https://pythonspeed.com/articles/conda-vs-pip/

Further reading

- Beginner tutorial
  - What is Pip? A Guide for New Pythonistas

- Using `requirements.txt` files to save your list of libraries
  - Why and how to make a requirements.txt

Today's topics

- ☑ Pip
- ❑ **Virtual environments** ←
- ❑ Setting up projects
- ❑ Classes
- ❑ Dunder methods
- ❑ Modules and packages
- ❑ Code hosting

References

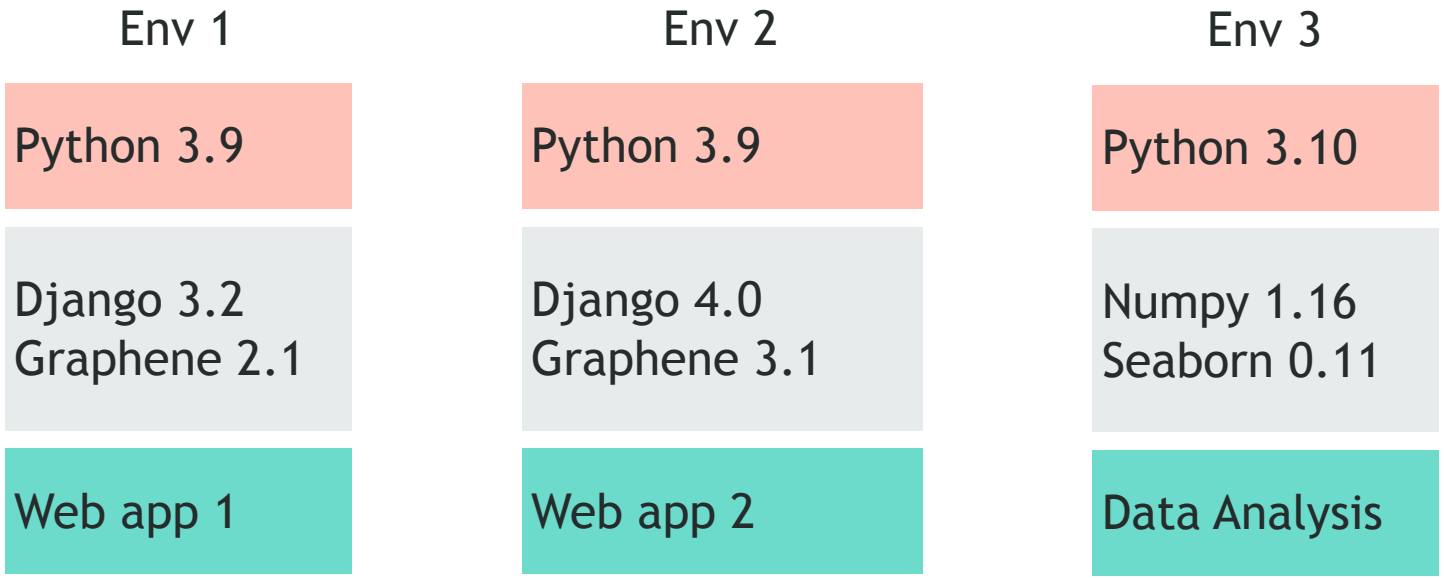
Interactive scenario

- **Pip and virtual environments**
  - Scenario 7 step 5

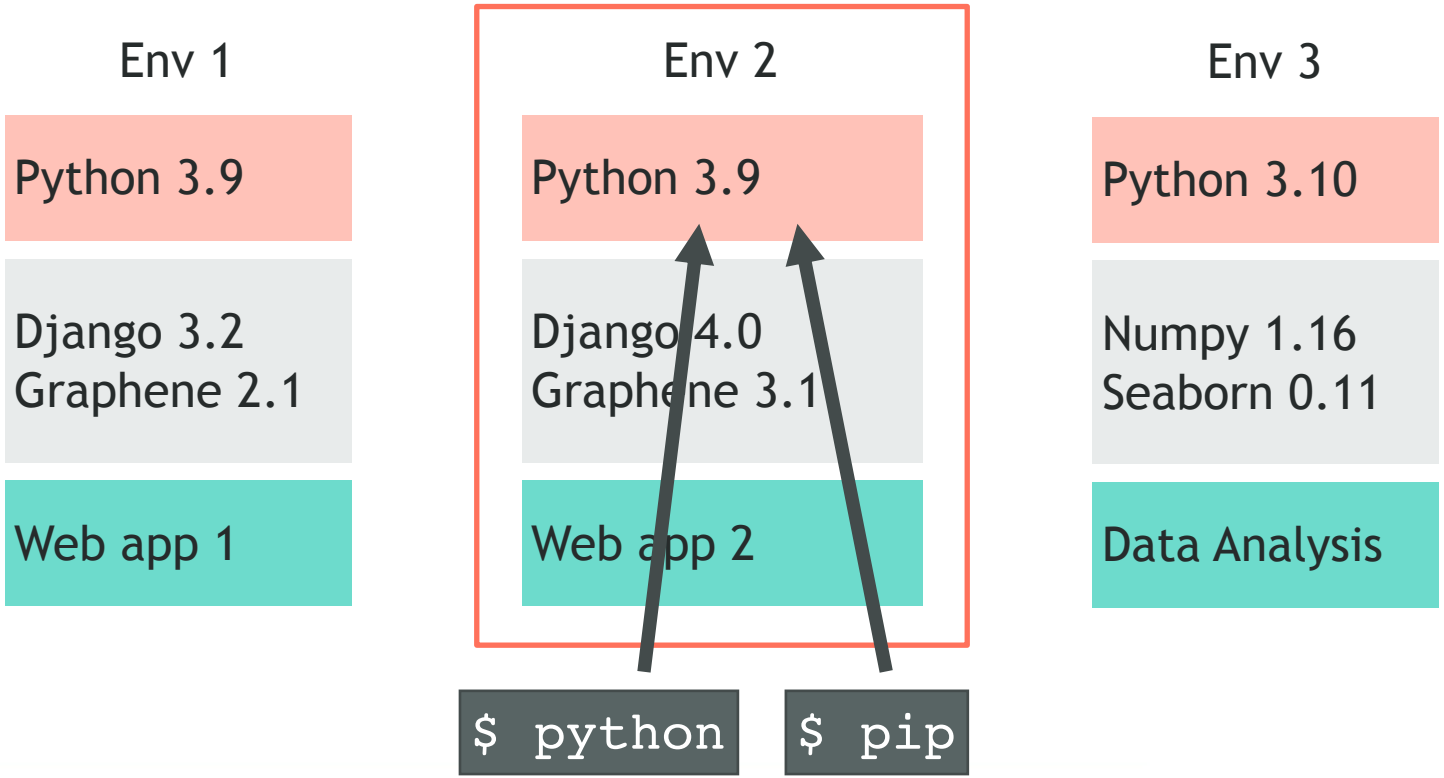
Video

- **Create virtual environments using *venv***
  - Next Level Python lesson 3.3

Virtual Environments



Virtual Environments



Tools for managing environments

- **venv + pip**  **Focus of today**
  - Modules that come with Python
- **pipenv**
  - Combines pip and venv
- **poetry**
  - Improves on pipenv, good for publishing packages to PyPI
- **PyCharm**
  - Can be useful in development

venv

```
$ python3.x -m venv <folder_name>
```

- Choose Python version
- `-m venv` means, run the venv module
- Folder name is usually just "**venv**"

e.g. `$ python3.10 -m venv venv`

Folder contents

- `$ python -m venv <folder_name>`

Creates a new folder with contents:

- **bin/** or **Scripts**
  - A copy of python
  - Script to activate the environment
- **lib/python3.8/site-packages/** or **Lib\site-packages**
  - Stores all installed packages

Mac/Linux

```
$ source venv/bin/activate
```

- Activates virtual env

```
$ which python
```

- Should be **venv/bin/python** now

```
$ pip install <package>
```

- Installs packages into **venv/lib/python/site-packages/**

```
$ deactivate
```

- Deactivates virtual environment

Windows – in PowerShell

```
$ venv\Scripts\Activate.ps1
```

- Activates virtual env

```
$ Get-Command python
```

- Should be **venv\Scripts\python.exe** now

```
$ pip install <package>
```

- Installs packages into **venv\Lib\site-packages**

```
$ deactivate
```

- Deactivates virtual environment

Activate scripts

Platform	Shell	Command to activate virtual environment
POSIX	bash/zsh	$ source <venv>/bin/activate
	fish	$ source <venv>/bin/activate.fish
	csh/tcsh	$ source <venv>/bin/activate.csh
	PowerShell Core	$ <venv>/bin/Activate.ps1
Windows	cmd.exe	C:\> <venv>\Scripts\activate.bat
	PowerShell	PS C:\> <venv>\Scripts\Activate.ps1

How do you link to it in PyCharm?

- Go to PyCharm **settings**
- Open **project interpreter** options
- Click the **gear** icon and **Add...**
- **Virtual environment** should already be selected on the left
- Choose **New environment** or **Existing environment**
- If using existing environment, select **<proj>/venv/bin/python** or **<proj>\venv\Scripts\python.exe**

- Visual instructions in course documentation

venv and source control/git

- Don't commit the **venv** folder
- Save installed package names to a text file (requirements.txt)
- Add virtual environment folder to a **.gitignore** file
- Add instructions for python environment setup to your README

Don't know git?

- Intro to git and GitHub (lesson 3.4)

Today's topics

- ☑ Pip
- ☑ Virtual environments
- ❑ **Setting up projects** ←
- ❑ Classes
- ❑ Dunder methods
- ❑ Modules and packages
- ❑ Code hosting

References

Video

- **Set up your code to be shared**
  - Next Level Python lesson 3.5
- **Clone a project**
  - Next Level Python lesson 3.6

Project setup from video (from scratch)

- Create a new git repository
- Create a new virtual environment
- Add a README
- Add a .gitignore file
- Add a license
- Add a requirements.txt file
- Make your first commit
- Push your changes

README files

- Usually in Markdown (.md) or ReStructured Text (.rst) format

- Often includes
  - Description
  - Setup instructions
  - How to run tests
  - Links to other documentation/websites

- If you need more documentation pages, put in **`docs`** folder

.gitignore file

- List of file/folders to ignore in git

- Should include, at the very least:
  - Virtual environment
  - Compiled files (e.g. pyc files)
  - IDE configuration files

Example: https://github.com/github/gitignore/blob/master/Python.gitignore

requirements.txt

- **Create text file with all installed packages + versions**
  - `$ pip freeze > requirements.txt`

  - This adds all packages, including dependencies
  - You can trim it down to remove dependencies or reorder them so dependencies are listed after project requirements
  - Add comments for links to documentation

- **Install all packages + versions from text file**
  - `$ pip install -r requirements.txt`

Licenses

- If you want people to be able to modify, share or distribute your code, add an appropriate license inside **LICENSE.txt**
- Find the right license: https://choosealicense.com/
- If you don't want to let anyone use your code, add a copyright notice to your Readme

Set up email project (from template)

1. Fork the project
2. Clone the forked project
3. Create a virtual environment
4. Install dependencies
5. Update **main.py** with your code from **homework_2.py**
6. Update the code to send an email with your content
7. Create a copy of **.env-sample** and rename it to **.env**
8. Update **.env** with your details

Create your email project

1. Fork the project
2. Clone the forked project
3. Create a virtual environment
4. Install dependencies
5. Update **main.py** with your code
6. Create a **.env** file to store your secrets

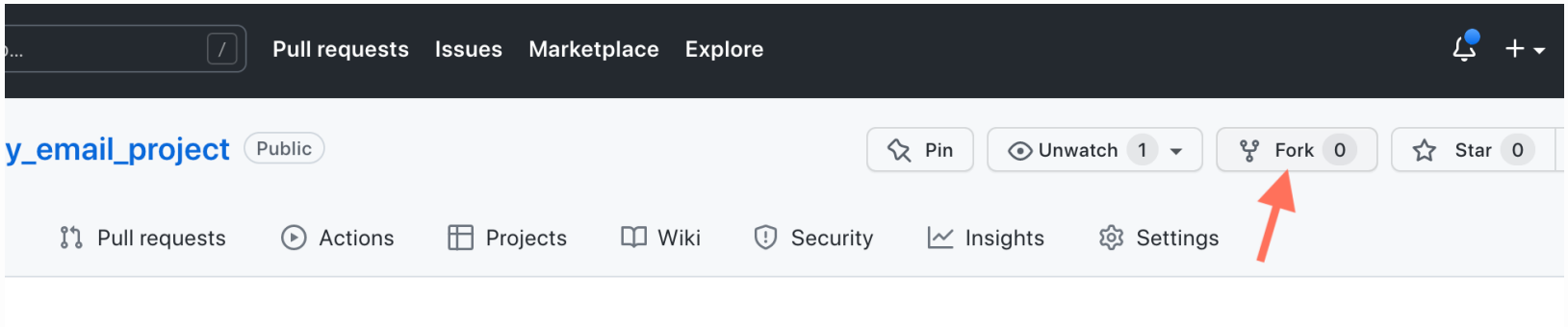
Set up email project (from template)

1. **Fork the project**
2. Clone the forked project
3. Create a virtual environment
4. Install dependencies
5. Update **main.py** with your code
6. Create a **.env** file to store your secrets

1. Fork the project

- Go to https://github.com/ariannedee/daily_email_project

- Fork the project to your account
  - This will create a copy that's owned by you



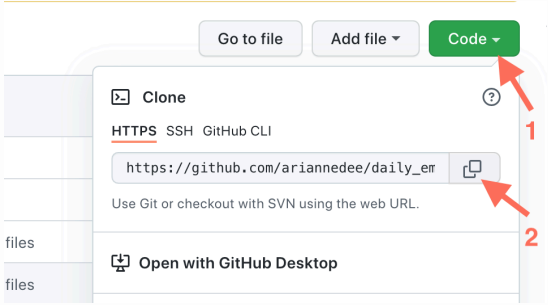
Set up email project (from template)

1. Fork the project
2. **Clone the forked project**
3. Create a virtual environment
4. Install dependencies
5. Update **main.py** with your code
6. Create a **.env** file to store your secrets

2. Clone the forked project

In your new forked project:

1. Click the green **Code** button
2. Copy the url
3. In a terminal, navigate to where you want the code to be
4. Do **`git clone <url>`**
5. Go to the new folder
   `cd daily_email_project`




```
$ cd Code
$   Navigate to where you want your project to be
$
$ git clone https://github.com/ariannedee/daily_email_project.git
                  This should be your GitHub username
```

Set up email project (from template)

1. Fork the project
2. Clone the forked project
3. **Create a virtual environment**
4. Install dependencies
5. Update **main.py** with your code
6. Create a **.env** file to store your secrets

3. Create a virtual environment

- In the **daily_email_project** folder
  - **`python3.10 -m venv venv`**

- Activate the environment
  - Linux/Mac:
    - **`source venv/bin/activate`**
  - PowerShell:
    - **`venv\Scripts\Activate.ps1`**

Set up email project (from template)

1. Fork the project
2. Clone the forked project
3. Create a virtual environment
4. **Install dependencies**
5. Update **main.py** with your code
6. Create a **.env** file to store your secrets

4. Install dependencies

- Project dependencies are in **requirements.txt**

- Install all dependencies:
  - `pip install -r requirements.txt`

Set up email project (from template)

1. Fork the project
2. Clone the forked project
3. Create a virtual environment
4. Install dependencies
5. **Update *main.py* with your code**
6. Create a **.env** file to store your secrets

5. Update **main.py** with your code

- Copy your code from **homework_2.py** into **main.py**

- And send your email content

```
from send_email import send_text_email

subject = 'Daily email'
content = 'Email content'

send_text_email(subject=subject, content=content)
```

Set up email project (from template)

1. Fork the project
2. Clone the forked project
3. Create a virtual environment
4. Install dependencies
5. Update **main.py** with your code
6. **Create a *.env* file to store your secrets**

6. Create a **.env** file to store your secrets

- This project uses the **environs** package to handle private data, like email addresses and passwords

1. Make a copy the **.env-sample** file and name it **.env**
2. Fill out **.env** with your details (no quotes necessary)

You can get **.env** values:

```
from environs import Env
env = Env()
env.read_env()
PASSWORD = env('GMAIL_PWD')
```

Environs docs: https://github.com/sloria/environs

More resources

- Python Application Layouts – Real Python
- Environs package
- Environment variables in Python – Twilio
- Sending email and text messages – Automate the Boring Stuff, Chapter 18

Q&A and 15 min break

References

Interactive scenario

- **Intro to classes**
  - Scenario 8

Video

- **None yet**

Live Training

- **Object-Oriented Programming in Python**

Today's topics

- ☑ Pip
- ☑ Virtual environments
- ☑ Setting up projects
- ☐ **Classes** ←
- ☐ Dunder methods
- ☐ Modules and packages
- ☐ Code hosting

Covered Keywords – Classes

False	await	else	import	pass
None	break	except	in	raise
True	class	finally	is	return
and	continue	for	lambda	try
as	def	from	nonlocal	while
assert	del	global	not	with
async	elif	if	or	yield

Classes

```
class Clock:
    def __init__(self, hour=0, min=0):
        self.hour = hour
        self.min = min
```

- Creating a new instance calls __init__

```
c = Clock(hour=5)
```

- **self** references the instance
- Attributes should be set in __init__
- Methods are like functions, with **self** as the first parameter

Today's topics

- ☑ Pip
- ☑ Virtual environments
- ☑ Setting up projects
- ☑ Classes
- ☐ **Dunder methods** ←
- ☐ Modules and packages
- ☐ Code hosting

Dunder methods

- Python's hidden magic revealed

- Meant to be called another way

- **MyClass()** calls **__init__** and __new__
- **+** calls **__add__**
- **==** calls **__eq__**
- **len()** calls **__len__**
- **my_list[0]** calls **__getitem__**

15 min break

Today's topics

- ☑ Pip
- ☑ Virtual environments
- ☑ Setting up projects
- ☑ Classes
- ☑ Dunder methods
- ❑ **Modules and packages** ←
- ❑ Code hosting

References

Interactive scenario

- **Modules and Packages**
  - Scenario 9

Videos

- **Understand Python Modules and Namespaces**
  - Next Level Python lesson 5
    - **Modules and imports** – lesson 5.1
    - **__init__.py files** – lesson 5.2
    - **Namespaces and scope** – lesson 5.3

Covered Keywords – Scope

False	await	else	import	pass
None	break	except	in	raise
True	class	finally	is	return
and	continue	for	lambda	try
as	def	from	nonlocal	while
assert	del	global	not	with
async	elif	if	or	yield

Definitions

- **Module**
  - a python script (.py) with **definitions** / **statements**
  - can be imported and used in other python scripts
  - **Definition** - a variable, function or class
  - **Statement** - the usage of variables, functions or classes (run code)
- **Package**
  - a collection of python files (modules) in a folder
  - Prior to Python 3.3, an __init__.py file was required for a folder to be recognized as a package

Importing from modules and packages

import module	`module.func_1()`
from module **import** func_1, func_2	`func_1()`
from module **import** *	`func_n()` Don't use this
from module **import** function1 **as** f1	`f1()`
import package.submodule	`package.submodule.func_3()`
from package.submodule **import** func_3	`func_3()`
from .module **import** func_1	Relative import

sys.path

```
import sys
from pprint import pprint
pprint(sys.path)
```

- **sys.path** shows which folders Python will look for modules
- You can start absolute imports from these locations

- Includes:
  - Folder that the main script was run from
  - Python install location and site-packages
  - PYTHONPATH environment variable
  - In PyCharm: project directory and folders marked as source root

Relative imports

```
from .sibling import func
from . import sibling
from ..aunt import func
```

- Import relative to the current module
  - No need to know what's on **sys.path**

- Cannot be used in **main** module or a sibling of main
  - Can only be used inside packages

__name__

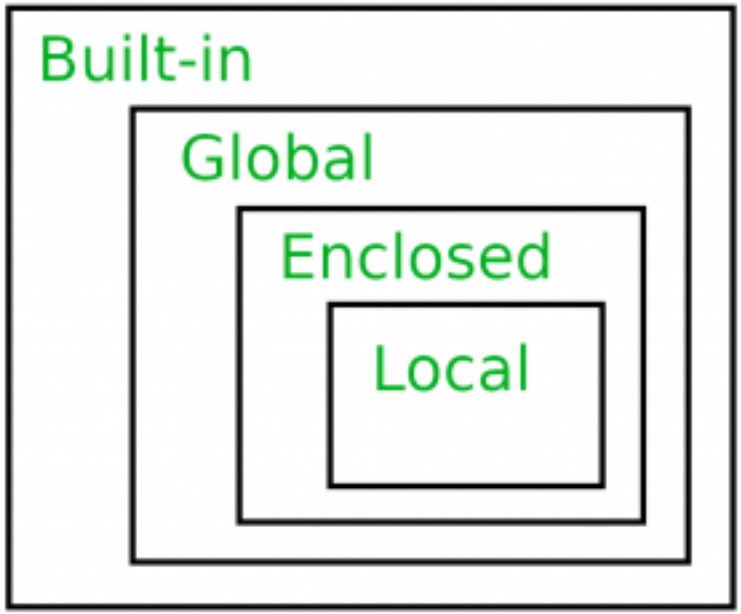
- Special attributes / dunder variables provides metadata

- **__name__** in a module refers to:
  - the dotted path to the module from the main file run
  - '**__main__**' if it was the main file run

- Use `if __name__ == '__main__':` to specify functionality that runs when a file is run as a script, but not when imported as a module

Scope

- Follows LEGB rule
- Local > enclosed > global > built-in



Refactor email

- Move things out of **main.py**
- Add an **api/** folder
- Create a module for each API used
- Add a main block to test each API

Automating email

Automating our email project

- You can run your code on your computer

- Scheduled tasks can be done with:
  - Linux/Mac: Cron jobs
  - Windows: Task scheduler

Mac/Linux

1. Create a virtual environment if you don't have one yet
2. Get the absolute path to Python
   - In your active venv, **`which python`** will give you the path
3. Open up crontab
   - **`crontab -e`**
4. Add a listing for our script
   - **`30 6 * * * <venv_python_path> <path/to/main.py> <args>`**
     - Runs your main.py every day at 6:30 am

- Sample tutorial

Mac/Linux

For example, on my computer, I would use:

Python path -

- `/Users/ariannedee/Code/daily_email_project/venv/python`

Script path -

- `/Users/ariannedee/Code/daily_email_project/daily_email/main.py`

Arguments -

- `Arianne`

Windows

1. Create a virtual environment if you don't have one yet
2. Get the absolute path to your **python.exe**
3. Create a new file in Notepad with contents

```
"path\to\python.exe" "path\to\main.py" "args"
pause
```

4. Save as **Daily_Email.bat**
5. Open Task Scheduler
6. Create a basic task that runs **Daily_Email.bat** every morning

Sample tutorial

Problems?

- Might not run if the computer is asleep
  - Schedule wake-up
- Others?

Today's topics

- ☑ Pip
- ☑ Virtual environments
- ☑ Setting up projects
- ☑ Classes
- ☑ Dunder methods
- ☑ Modules and packages
- ❑ **Code hosting** 

Code hosting options

- Online Python
  - https://www.online-python.com/
- Python Anywhere
  - https://www.pythonanywhere.com/
- Replit
  - https://replit.com/
- Anvil (web app)
  - https://anvil.works/

Web-based IDEs – simple scripts

- Online Python
  - https://www.online-python.com/

  - Pros:
    - Perfect for simple scripts that you share with others via a link
    - Supports some more advanced modules, like Pandas and NumPy
  - Cons:
    - Very small feature-set, e.g. no code storage or accounts
    - No control over environment, Python version, or installed modules

Web-based IDEs

- Python Anywhere
  - https://www.pythonanywhere.com/
  - Code, store and run Python scripts online
  - Pros:
    - Easy web-app hosting (1 app allowed in free account)
    - Full control over your environment via bash console (terminal)
  - Cons:
    - No easy package install or git integration
      - Should be comfortable with the command line
    - Limited sharing and collaboration capabilities
    - Jupyter notebook support only for paid accounts

Web-based IDEs

- Repl.it
  - https://repl.it/
  - Code, store and run Python (and more) projects online
  - Pros:
    - Supports many languages, libraries, and frameworks (e.g. PyGame)
    - Can host multiple web apps for free
    - Special interfaces for installing packages and using GitHub
    - Good sharing and forking capabilities
  - Cons:
    - No Jupyter notebook support
    - Can’t run single files easily
    - Environment configuration only in Poetry

Web-based platform

- Anvil
  - https://anvil.works/
  - Create drag & drop web apps with nothing but Python
  - Pros:
    - Don’t need to know HTML, CSS, JavaScript, SQL, command line or web app hosting
    - Unlimited free apps
    - Growing company with new features added regularly
    - Easy forking capability
  - Cons:
    - Paid version is expensive
    - Free version is slow
    - No collaboration with free version
    - IDE navigation/features are still lacking

Host on Python Anywhere

- Clone project
- Install dependencies
- Create .env file
- Create task
- Troubleshoot

Instructions in README

Python Anywhere issues

- Only certain URLs allowed when making requests
  - See allowed URLs
  - You can request additions - instructions

- You can't globally install external packages with pip
  - Use the **`--user`** flag - instructions

- You can only send smtp email through Gmail and you need an app password (not your normal password)

Create a Google app password

1. Enable 2-step authentication - instructions
2. Create an app password for Python Anywhere - instructions
3. Use that password in your **.env** file for GMAIL_PASSWORD

Send an HTML email

HTML email

- **<h1>Heading</h1>** - largest heading
- **<h2>Weather</h2>** - smaller heading (goes to h6)
- **
** - line break
- ** some text ** - a paragraph
- **Click me** - a link
- **<img src="url_to_img">** - an image
- **Bolded text** - bold text
- **Italicized text** - italicized text

HTML email - CSS

- You can add custom styling via CSS to any HTML tag
- <h1 **style="color: #FF2233; font-family: Helvetica;"**>Hi</h1>
- Basic styling works, like font and margin/padding
- More advanced styling might not work

In the code

```
from send_email import send_html_email

send_html_email(subject=subject, content=content)
```

- Send HTML content as string
- Email sends as both HTML and plaintext (in case the client doesn't support HTML emails)

More resources

- Send email tutorial
  - https://realpython.com/python-send-email/
- Intro to HTML:
  - https://www.w3schools.com/html/html_intro.asp
- Intro to CSS
  - https://www.w3schools.com/Css/css_intro.asp

Today's topics

- ☑ Pip
- ☑ Virtual environments
- ☑ Setting up projects
- ☑ Classes
- ☑ Dunder methods
- ☑ Modules and packages
- ☑ Code hosting

Covered Keywords – Week 3

False	await	else	import	pass
None	break	except	in	raise
True	class	finally	is	return
and	continue	for	lambda	try
as	def	from	nonlocal	while
assert	del	global	not	with
async	elif	if	or	yield

Week 3 homework

- Create a virtual environment for the project
- Send a text email (automated)
- Refactor into modules and functions (and classes)
- Optional: Send an HTML formatted email
- Optional: Host code on Python Anywhere

Video links

Video references for week 1

- **Running Python** – Intro to Python lesson 1.1
- **First code** – Intro to Python lesson 1.2
- **Variables** – Intro to Python lesson 2.2
- **Types** – Intro to Python lesson 2.1
- **Strings** – Intro to Python lesson 2.7
- **Functions** – Intro to Python lesson 2.6
- **Conditions** – Intro to Python lesson 3.2
- **Conditionals (if else)** – Intro to Python lesson 3.3
- **While loops** – Intro to Python lesson 4.1
- **For loops** – Intro to Python lesson 4.4

Video references for week 2

- **Lists** – Intro to Python lesson 4.3
- **Dictionaries** – Next Level Python lesson 1.3
- **Sets, tuples** – Intro to Python lesson 5.2
- **Exceptions** – Next Level Python lesson 1.4
- **Reading files** – Next Level Python lesson 2.1
- **Writing files** – Next Level Python lesson 2.2
- **CSV files** – Next Level Python lesson 2.3
- **Requests** – Next Level Python lesson 7.1
- **API requests** – Next Level Python lesson 7.5
- **Command line** – Next Level Python lesson 3.1

Video references for week 3

- **Pip** – Next Level Python lesson 3.2
- **Virtual environments** – Next Level Python lesson 3.3
- **Set up project** – Next Level Python lesson 3.5
- **Clone a project** – Next Level Python lesson 3.6
- **Modules and import** – Next Level Python lesson 5.1
- **__init__.py files** – Next Level Python lesson 5.2
- **Namespaces and scope** – Next Level Python lesson 5.3
- **Run code in the cloud** – Next Level Python lesson 9.1
- **Classes** – Live training – Object-Oriented Programming in Python

Supplemental videos – concepts

- **Dates and times** – Next Level Python lesson 1.5
- **Regular expressions** – Next Level Python lesson 1.6
- **HTML overview** – Next Level Python lesson 7.2
- **Scraping websites** – Next Level Python lesson 7.3
- **Git and GitHub**– Next Level Python lesson 3.4
- **Debugging** – Next Level Python lesson 6.1
- **Testing** – Next Level Python lesson 6.2

Supplemental videos – projects

- **Basics review** – Next Level Python lesson 1.2
- **Web scraper project** – Next Level Python lesson 8
- **Intro to data analysis** – Intro to Python lesson 6
- **Intro to web apps** – Intro to Python lesson 7
- **Create an Anvil web app** – Next Level Python lesson 9.2

Poll #1 (multi-answer)

- What aspects did you like about the course?
  - Interactive scenarios
  - Example code in GitHub
  - Email project
  - Problems
  - Practice
  - Quiz

Poll #2 (multi-answer)

- What aspects do you think I can remove?
  - Interactive scenarios
  - Example code in GitHub
  - Email project
  - Problems
  - Practice
  - Quiz

Poll #3 (multi-answer)

- How can I improve for next time?
  - Incorporate interactive scenarios more
  - Cover topics in less detail
  - Cover fewer topics
  - Cover more topics
  - Don't bother with in-class coding problems
  - Focus more on in-class coding problems
  - Don't bother with homework
  - Answer fewer questions from group chat
  - Other (say in chat)