OOP TERMINOLOGY

OOP Object Oriented Programming a way of thinking and arranging data in programming that groups types of data ("properties") with functions ("methods") and calls the entire thing a "class"

Class also known as an object's "type", classes form "blueprints" for creating new object instances, defining methods and properties

Object instance A specific occurrence of a class. Creating one is called instantiation or constructrion.

Property Data stored by the class, can be accessed with a "." character

Method A function defined in a class declaration that gets attached ("bound") to the object instances, also accessed with "."

Constructor A special method that is run when you instantiate a class

Extend Classes (the subclass) can *inherit* or *extend* another class (the base class) which effectively copies over the methods and properties

Overriding When a subclass replaces a base-class method we say it is overridden

Super Super is a keyword to allow subclasses to access an overridden method on a base class

Interface The outwardly facing methods and properties of a class

PYTHON CLASS SYNTAX

```
# User class with properties name
# and logged_in, and method login
class User:
    def __init__(self, name):
        self.name = name
        self.logged_in = False
    def login(self):
        self.logged_in = True
# StudentUser class extends base
# class User overriding login
class StudentUser(User):
    def login(self):
        super(self).login()
        self.attended = True
# Construct object instances
# of User and StudentUser
jo_user = User("Jo")
felix_user = StudentUser("Felix")
```

Pipenv

Creating a new virtualenv

pipenv --python 3.6

Enter current virtualenv

pipenv shell

Install a new package from PyPI

pipenv install jinja2

Install all packages listed in Pipfile

pipenv install

PYTHON IMPORT SYNTAX

```
# Import from my_helper_code.py
import my_helper_code
my_helper_code.example_func()
print(my_helper_code.ex_data)

# Get only specific functions,
# variables, or submodules
from my_helper_code import (
        ex_data,
        example_func,
)
example_func()
print(ex_data)
```

Module Terminology

module A file (or directory) that can be imported into other files, allowing you to use data and functions from one file in another

standard library Built-in modules that always can be imported in Python

virtualenv An "environment" that stores downloaded Python packages - each project gets one. Can be "entered" (pipenv shell) and "exited" (deactivate) to gain access to downloaded packages.

PyPI Python Package Index - Site with free Python packages.

pipenv Tool for downloading packages from PyPI into a *virtualenv*

Pipfile Keeps a log of what you download with pipenv, so you or teammates can get set-up again

TEMPLATES

```
variables Use "." to access dict keys
     <h2>Hi {{ user.username }}!</h2>
if
     {% if age > 17 %}
         You may continue.
     {% else %}
         Too young.
     {% endif %}
for
     {% for post in blog_posts %}
         <h2>{{ post.title }}</h2>
     {% endfor %}
filters
     Hi {{ name | upper }}
include
     {% include "user_snippet.html" %}
extends & blocks Allows
                          template
     variations: replace "block" place-
    holder in a base.html
     {% extends "base.html" %}
     {% block main_content %}
```

<h1>User</h1> {{ user.name }}

{% endblock main_content %}

Using Jinja

TEMPLATING TERMINOLOGY

Template HTML file or string containing "placeholder" spots for variable data to be inserted, and limited use of logic (if, for, etc)

Context Collection of context variables to be inserted in various places in a template, and used in logic

Render When a template is invoked with a context, outputs a string