Gymnasium (essentially gym)

- A project that provides an API for single agent RL environments
- Four key functions: make, Env.reset(), Env.step(), and Env.render()

- Env is a python class representing a markov decision process
- Allows creation of an initial state, transition/move to new states, and being able to visualize the environment.
- Wrapper can be used to augment/modify the environment

To initialize an environment, use the make() function. An example (derived from original documentation)

```
import gymnasium as gym
env = gym.make(‘CartPole-v1’)
```

Now, an env function is returned.
pprint_registry() shows all creatable environments

Code from the original documentation; creates a single-episode run of lunar lander:

```
import gymnasium as gym

env = gym.make("LunarLander-v3", render_mode="human")
observation, info = env.reset()

episode_over = False
while not episode_over:
    action = env.action_space.sample()  # agent policy that uses the observation and info
    observation, reward, terminated, truncated, info = env.step(action)

    episode_over = terminated or truncated

env.close()
```

End code.

An environment is first created with make(). Render mode specifies how to make the environment. The specified LunarLander environment is made.

Using Env.reset() on the environment gets the first environment observation with some additional info.

We want to continue the agent environment loop till the environment ends, which we don't know when that will happen. Episode_over is a variable that determines when to stop the environment.

The agent performs an action in this environment; Env.step() is what executes the specified step. This action becomes random with env.action_space.sample(). This is like having a robot press a specified button on a controller to play a video game. From taking that action, there is a new environment and a reward for taking that action. (positive for doing something good, negative for doing something bad). This whole paragraph details a timestamp.

The environment may end after some steps (if the robot gets to a state where it can't continue, the robot must stop. In this case, step() returns terminated.

If we want the environment to end after X number of steps, the environment issues a truncated signal.

Receiving any end signal like this usually results in restarting the environment

Action_space and observation_space attributes for the environment. This is good for knowing the input and output of the environment. We might want to use env.action_space.sample() instead of an agent policy.

Most importantly, Env.action_space and Env.observation_space are instances of Space

There are a variety of spaces, such as a box, which has upper and lower limits of an n dimensional shape. There is Text, which describes a string space with min/max length, or a dictionary or tuple, which are self-explanatory.

To modify the environment:

Wrappers can do this (without having to alter the underlying code). Will allow code to be more modular and can also be changed

- When using gymnasium.make(), generally will be wrapped with TimeLimit, OrderEnforcing, and PassiveEnvChecker.

First need to make a base environment. Can then pass environment with (possibly optional) parameters to the wrapper's constructor:

When having a wrapped environment and you want to get the unwrapped environment underneath all layers of wrappers, use the unwrapped attribute. Unwrapped attributes on a base environment will just return itself.

There are a few wrappers. These include:

- TimeLimit, which issues a truncated signal if the program exceeds the maximum amount of timestamps.
- ClipAction, which clips any action passed to step such that it lies in the base environment’s action space
- RescaleAction: applies an affine transformation to a particular action to linearly scale for a new low and high bound on the environment
- TimeAwareObservation: adds info about the index of timestep to observation.