LeapMotion and You

By: Pod 5 -
Mitch Mennelle, Dan Trethaway, Nick Maag

Agenda

Part 1: Leap Motion Controller

Part 2: Getting started with the Leap

Part 3: Working in the SWAMP

Agenda

Part 1: Leap Motion Controller

Part 2: Getting started with the Leap

Part 3: Working in the SWAMP

Leap Motion Controller: Hardware

How the hardware works:

- Stereo Cameras
- Infrared Emitters and Imagers
- Embedded Data Processing

The Leap Motion Controller 2 uses these components to deliver precise hand tracking in a variety of environments

Leap Motion Controller: Software

Ultraleap provides a number of ways for developers to utilize the their services:

- Ultraleap Hyperion
- Ultraleap Gemini
- Unity Plugin
- Unreal Plugin
- Touchfree

All of these allow developers to integrate and utilize the Leap Motion Controller 2

Leap Motion Controller: Software

For the Touchless Kiosk, our winner is Ultraleap Gemini.

Ultraleap Gemini:

- Gemini is the rebuilt tracking engine containing useful tools and SDK for ultraleap.
- Control Panel - Using the control panel allows us as developers to see the camera feeds and adjust settings for your device.
- LeapC - An API used for accessing the Ultraleap tracking data. Ultimately we will use bindings to use LeapC with other higher-level languages.
    - See https://docs.ultraleap.com/api-reference/tracking-api/leapc-guide.html for documentation.

Leap Motion Controller: Uses

The software that Ultraleap provides allows a wide array of uses for the controller including:

- Professional research/training
- Gaming
- Mounted to VR Headsets to provide an AR experience
- Interfacing with your PC without touch (Touchless software)
- Endless possibilities!!!!

Agenda

Part 1: Leap Motion Controller

<u>Part 2: Getting started with the Leap</u>

Part 3: Working in the SWAMP

Getting Started With Leap

Step 1: Download/Install Ultraleap Gemini

Step 2: Pull Python Bindings Repo

Step 3: Setup Venv

Step 4: Build Python Bindings

Step 5: Profit

Step 1: Download/Install Ultraleap Gemini

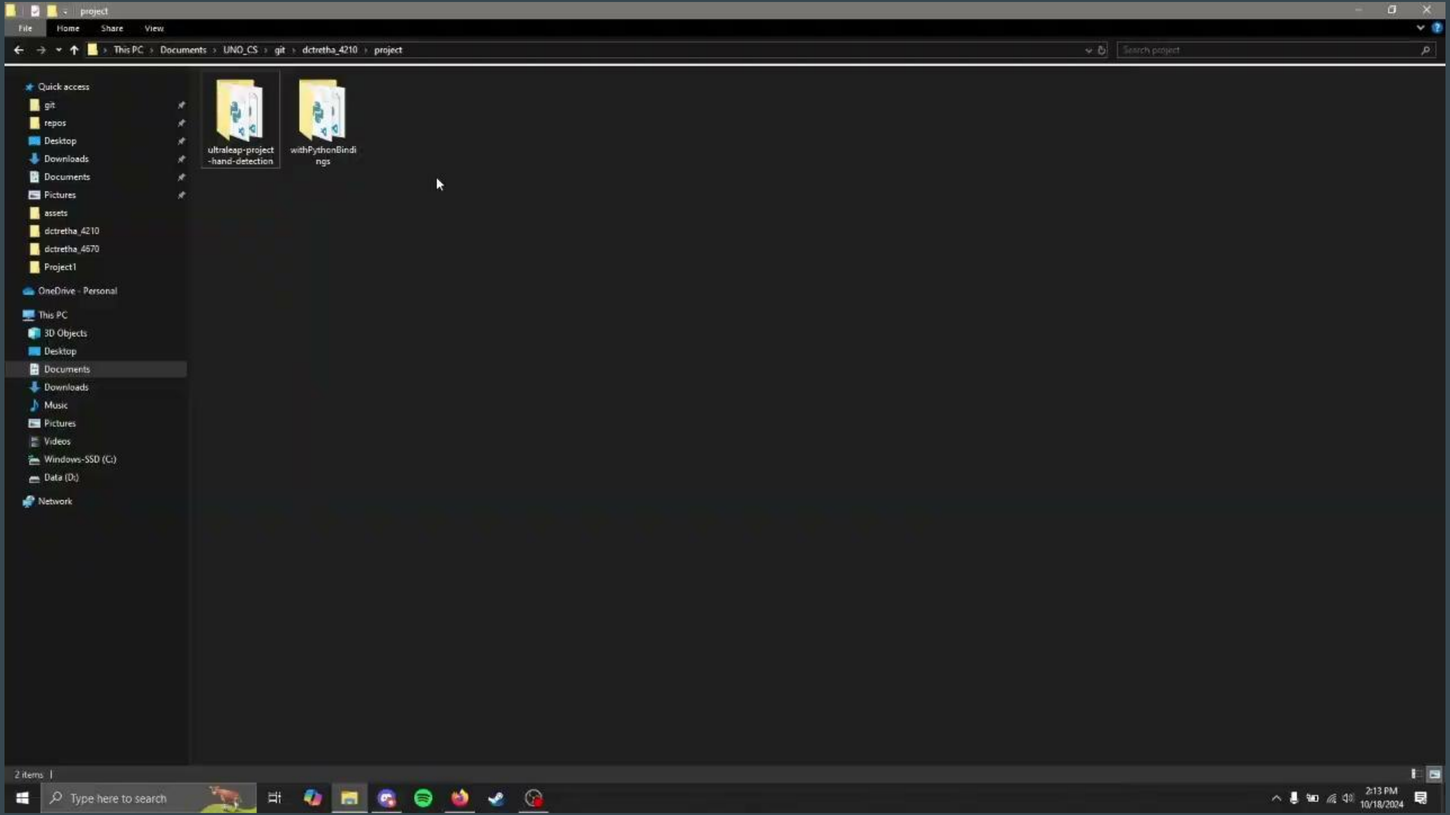
Download:

- Go to: https://leap2.ultraleap.com/downloads/
- Select Ultra Leap Controller 2
- Select the OS your device is currently using -> download Gemini

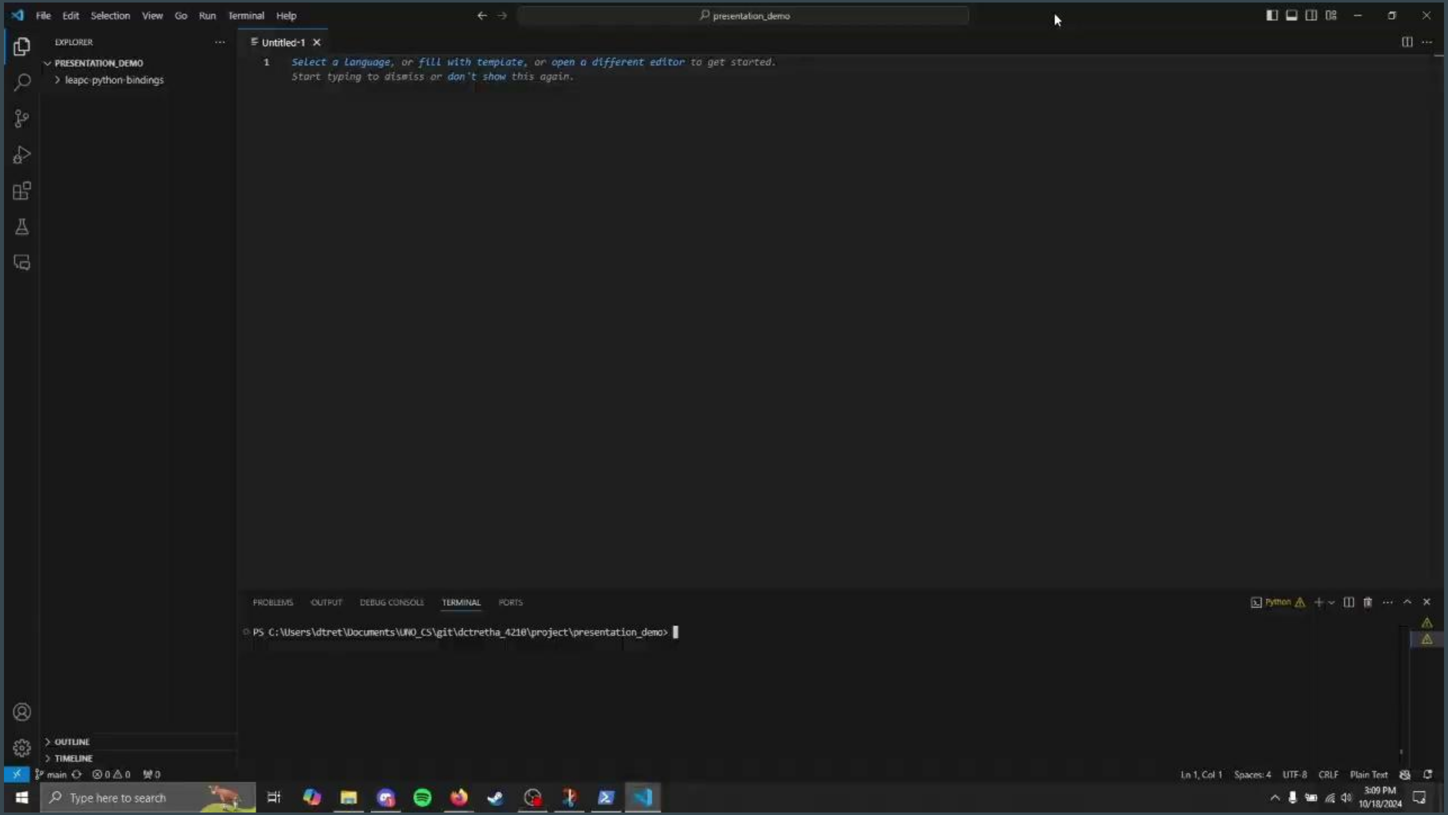
Install:

- Once the download is finished, run the .exe file
- Run through the installation normally, make sure NOT to change default install location! (This could cause much pain later on when building the Python Bindings)

Step 2: Pull Python Bindings Repo



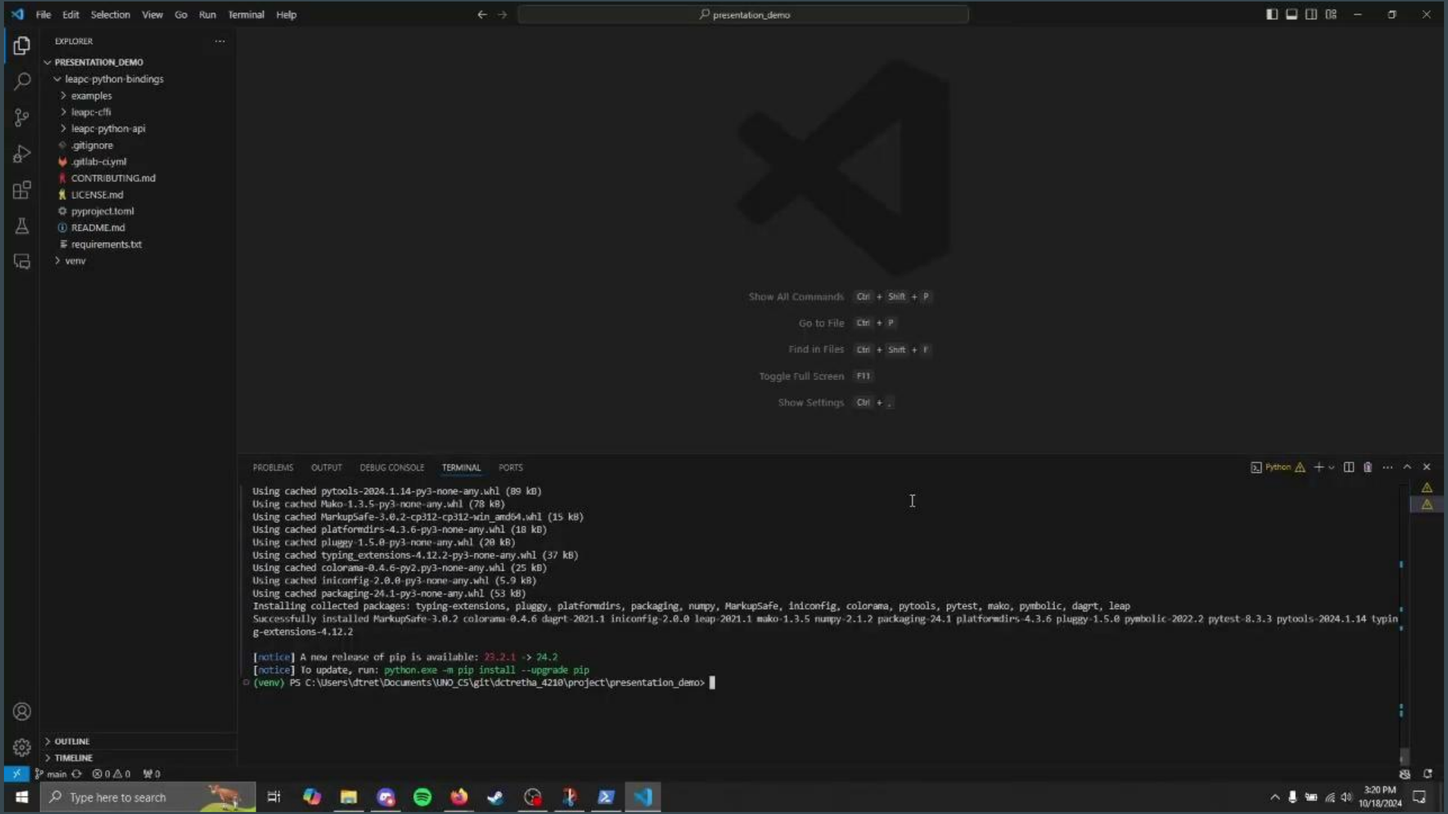
Step 3: Setup Virtual Environment



```
Select a language, or fill with template, or open a different editor to get started.
Start typing to dismiss or don't show this again.
```

```
PS C:\Users\dtret\Documents\UNO_CS\git\dctretha_4210\project\presentation_demo>
```

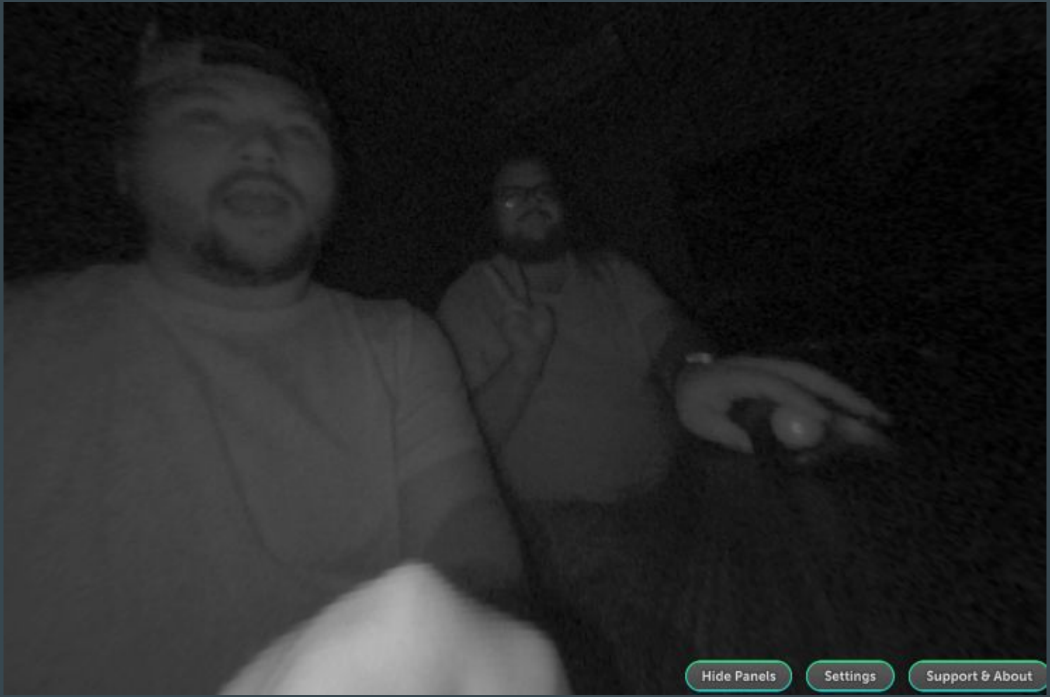
Step 4: Build Python Bindings



```
Using cached pytools-2024.1.14-py3-none-any.whl (89 kB)
Using cached Mako-1.3.5-py3-none-any.whl (78 kB)
Using cached MarkupSafe-3.0.2-cp312-cp312-win_amd64.whl (15 kB)
Using cached platformdirs-4.3.6-py3-none-any.whl (18 kB)
Using cached pluggy-1.5.0-py3-none-any.whl (20 kB)
Using cached typing_extensions-4.12.2-py3-none-any.whl (37 kB)
Using cached colorama-0.4.6-py2.py3-none-any.whl (25 kB)
Using cached iniconfig-2.0.0-py3-none-any.whl (5.9 kB)
Using cached packaging-24.1-py3-none-any.whl (53 kB)
Installing collected packages: typing-extensions, pluggy, platformdirs, packaging, numpy, MarkupSafe, iniconfig, colorama, pytools, pytest, mako, pymbolic, dagrt, leap
Successfully installed MarkupSafe-3.0.2 colorama-0.4.6 dagrt-2021.1 iniconfig-2.0.0 leap-2021.1 mako-1.3.5 numpy-2.1.2 packaging-24.1 platformdirs-4.3.6 pluggy-1.5.0 pymbolic-2022.2 pytest-8.3.3 pytools-2024.1.14 typin
g-extensions-4.12.2

[notice] A new release of pip is available: 23.2.1 -> 24.2
[notice] To update, run: python.exe -m pip install --upgrade pip
(venv) PS C:\Users\dtret\Documents\UNO_CS\git\dctretha_4210\project\presentation_demo>
```

Step 5: Profit!

You are now ready to use the Leap Motion Controller! (locally!)

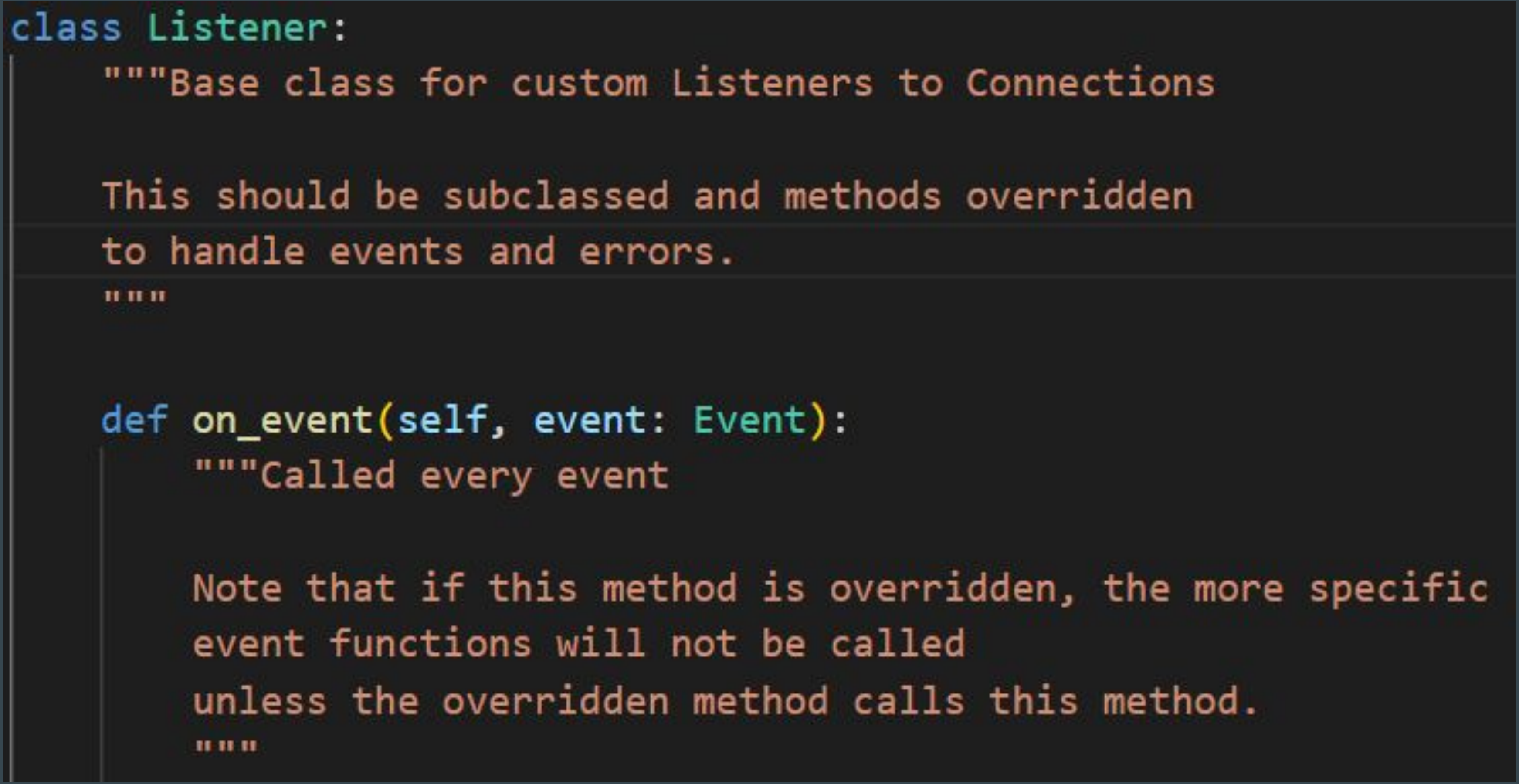


The Part Where I Tell You How to Use Them

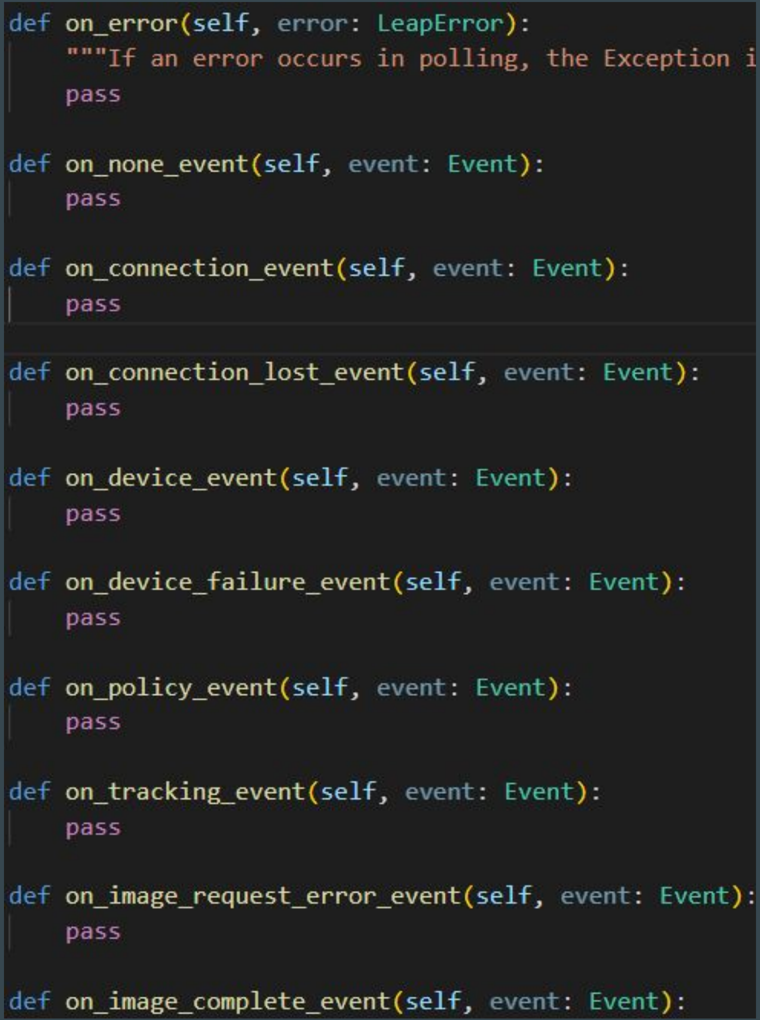
Important classes

Connection:

- Handles Listeners’ connections with the camera.
- ☀ Listener:
  - Handles Events.
  - Event:
    - Handles anything logged by the camera.

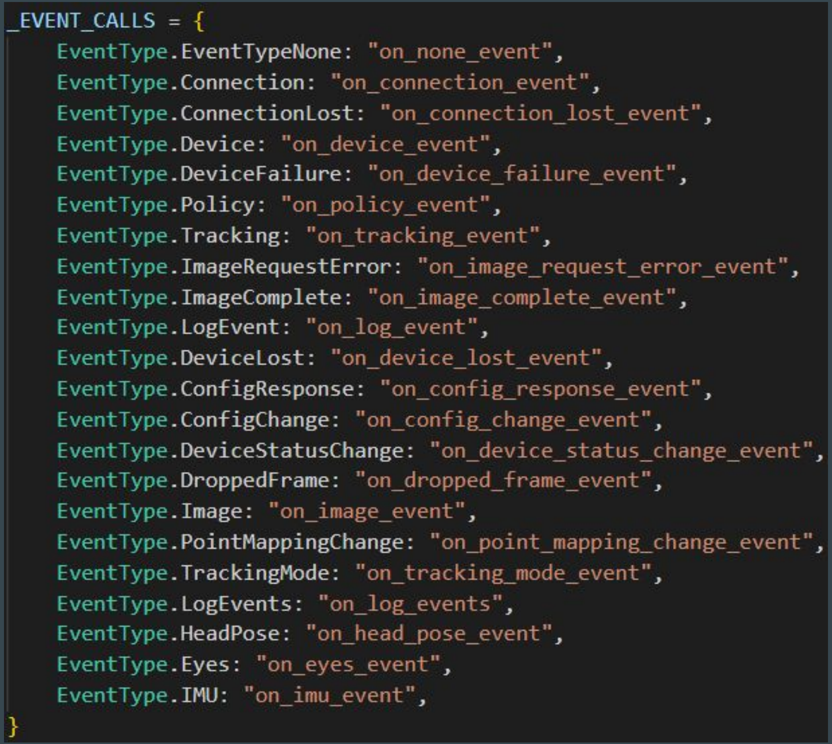


```python
class Listener:
    """Base class for custom Listeners to Connections

    This should be subclassed and methods overridden
    to handle events and errors.
    """

    def on_event(self, event: Event):
        """Called every event

        Note that if this method is overridden, the more specific
        event functions will not be called
        unless the overridden method calls this method.
        """
```



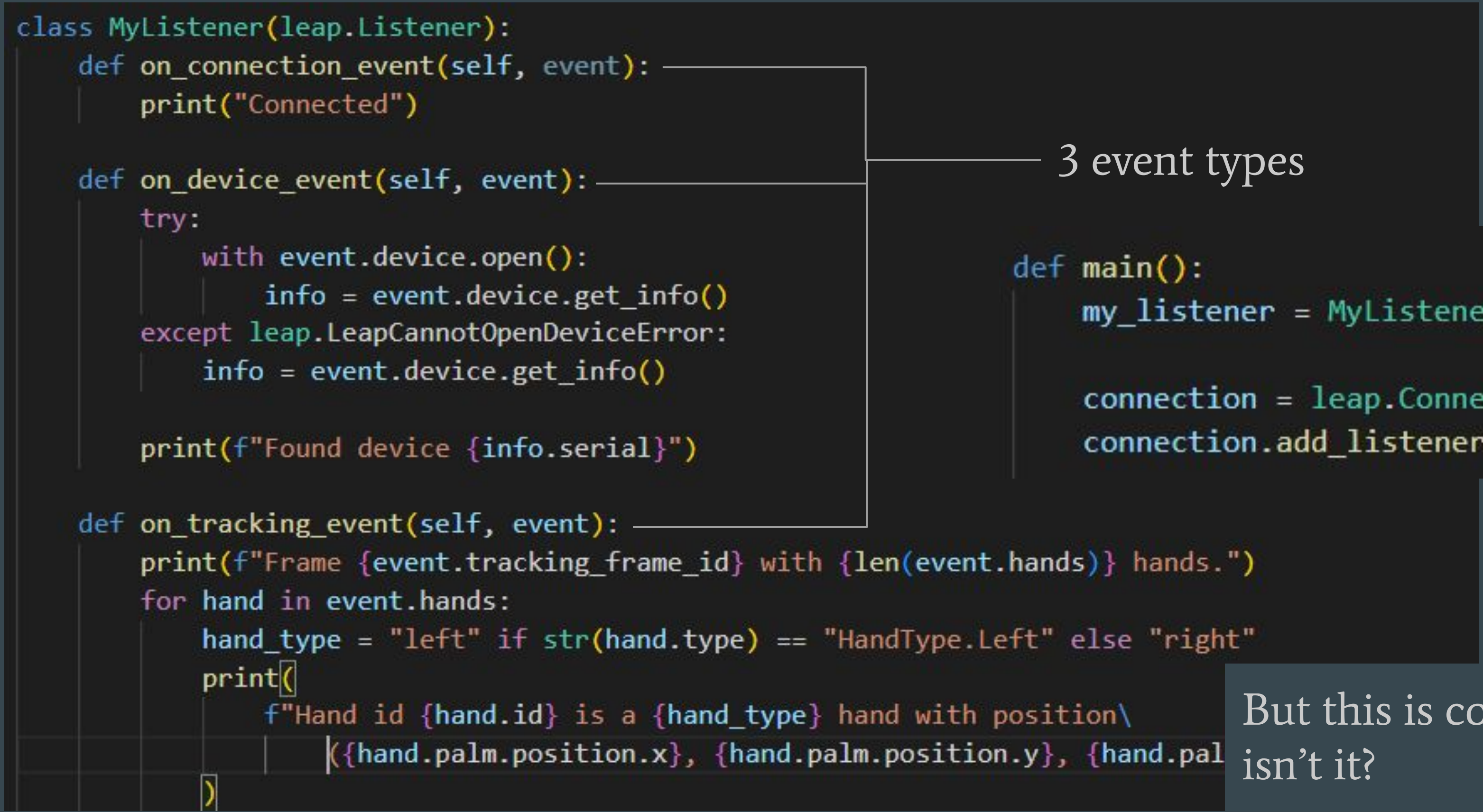
```python
def on_error(self, error: LeapError):
    """If an error occurs in polling, the Exception i
    pass

def on_none_event(self, event: Event):
    pass

def on_connection_event(self, event: Event):
    pass

def on_connection_lost_event(self, event: Event):
    pass

def on_device_event(self, event: Event):
    pass

def on_device_failure_event(self, event: Event):
    pass

def on_policy_event(self, event: Event):
    pass

def on_tracking_event(self, event: Event):
    pass

def on_image_request_error_event(self, event: Event):
    pass

def on_image_complete_event(self, event: Event):
```

most of these don't matter



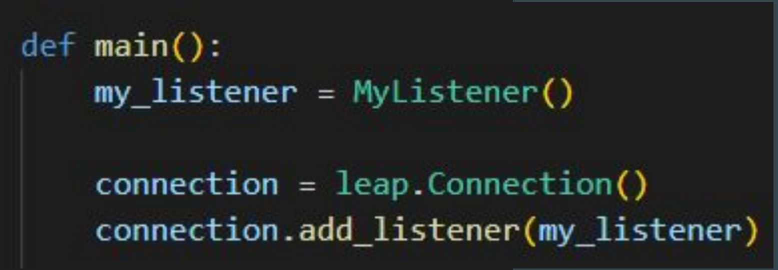
```python
_EVENT_CALLS = {
    EventType.EventTypeNone: "on_none_event",
    EventType.Connection: "on_connection_event",
    EventType.ConnectionLost: "on_connection_lost_event",
    EventType.Device: "on_device_event",
    EventType.DeviceFailure: "on_device_failure_event",
    EventType.Policy: "on_policy_event",
    EventType.Tracking: "on_tracking_event",
    EventType.ImageRequestError: "on_image_request_error_event",
    EventType.ImageComplete: "on_image_complete_event",
    EventType.LogEvent: "on_log_event",
    EventType.DeviceLost: "on_device_lost_event",
    EventType.ConfigResponse: "on_config_response_event",
    EventType.ConfigChange: "on_config_change_event",
    EventType.DeviceStatusChange: "on_device_status_change_event",
    EventType.DroppedFrame: "on_dropped_frame_event",
    EventType.Image: "on_image_event",
    EventType.PointMappingChange: "on_point_mapping_change_event",
    EventType.TrackingMode: "on_tracking_mode_event",
    EventType.LogEvents: "on_log_events",
    EventType.HeadPose: "on_head_pose_event",
    EventType.Eyes: "on_eyes_event",
    EventType.IMU: "on_imu_event",
}
```

track_event_example.py



```python
class MyListener(leap.Listener):
    def on_connection_event(self, event):
        print("Connected")

    def on_device_event(self, event):
        try:
            with event.device.open():
                info = event.device.get_info()
        except leap.LeapCannotOpenDeviceError:
            info = event.device.get_info()

        print(f"Found device {info.serial}")

    def on_tracking_event(self, event):
        print(f"Frame {event.tracking_frame_id} with {len(event.hands)} hands.")
        for hand in event.hands:
            hand_type = "left" if str(hand.type) == "HandType.Left" else "right"
            print(
                f"Hand id {hand.id} is a {hand_type} hand with position\
                    ({hand.palm.position.x}, {hand.palm.position.y}, {hand.pal
            )
```

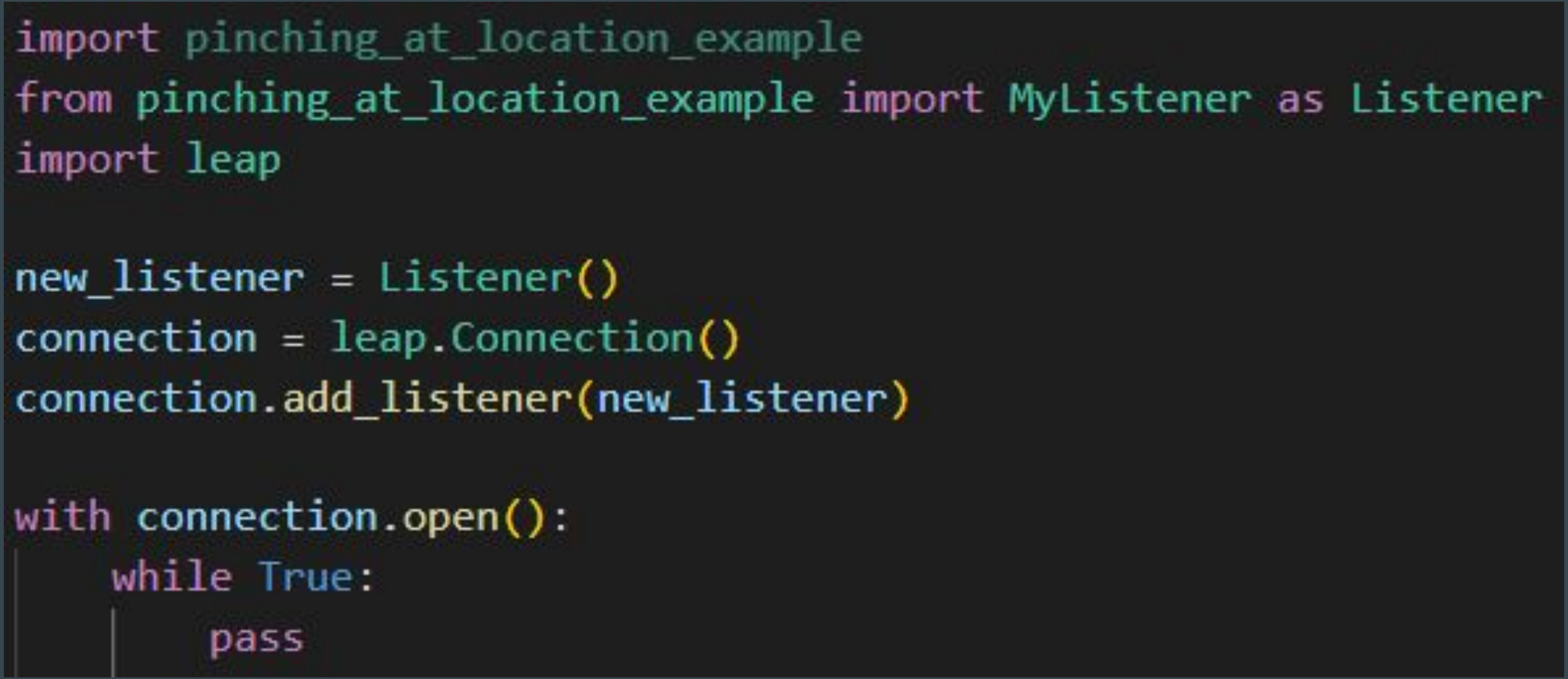
3 event types



```python
def main():
    my_listener = MyListener()

    connection = leap.Connection()
    connection.add_listener(my_listener)
```

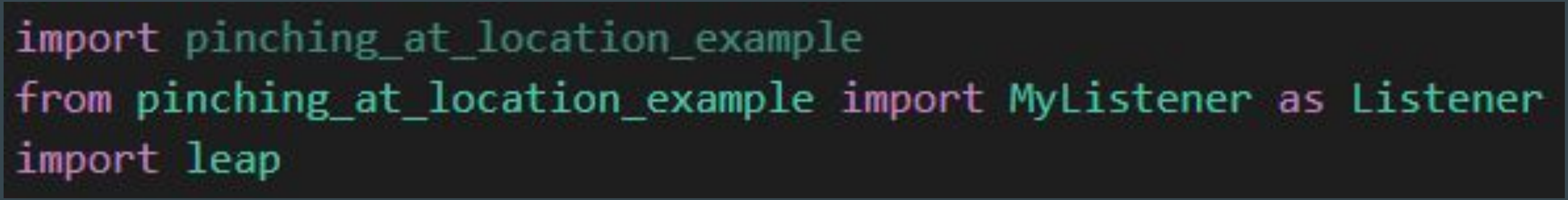
But this is complicated, isn't it?

Maag’s Python Bindings Bindings



```python
import pinching_at_location_example
from pinching_at_location_example import MyListener as Listener
import leap

new_listener = Listener()
connection = leap.Connection()
connection.add_listener(new_listener)

with connection.open():
    while True:
        pass
```

Step 1: Importing



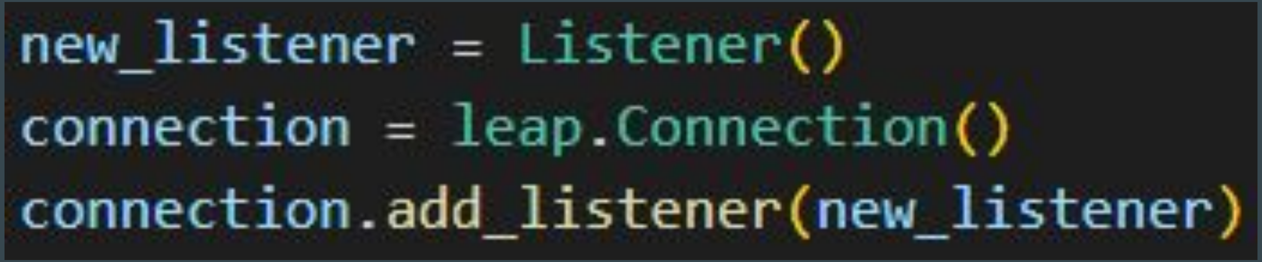
```
import pinching_at_location_example
from pinching_at_location_example import MyListener as Listener
import leap
```

`pinching_at_location_example.py` - I will give you on github (actually `camera_bindings.py`)

`import MyListener as Listener` - Listener class to be instanced

`import leap` - Yeah

Step 2: Setting up Connection & Listener



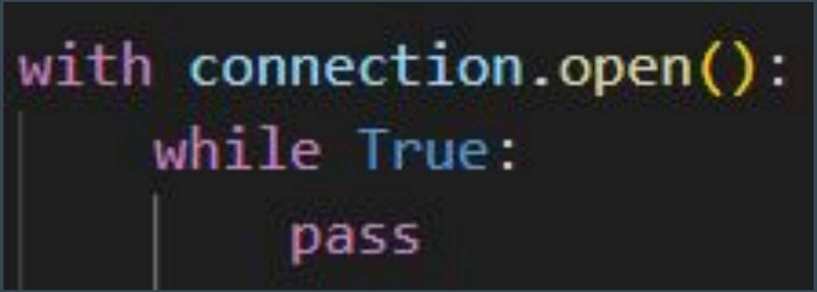
```
new_listener = Listener()
connection = leap.Connection()
connection.add_listener(new_listener)
```

`new_listener = Listener()` - Instances Listener class for event handling

`connection = leap.Connection()` - Makes a connection

`connection.add_listener(new_listener)` - Adds listener to connection itself

Step 3: Open the Connection



```
with connection.open():
    while True:
        pass
```

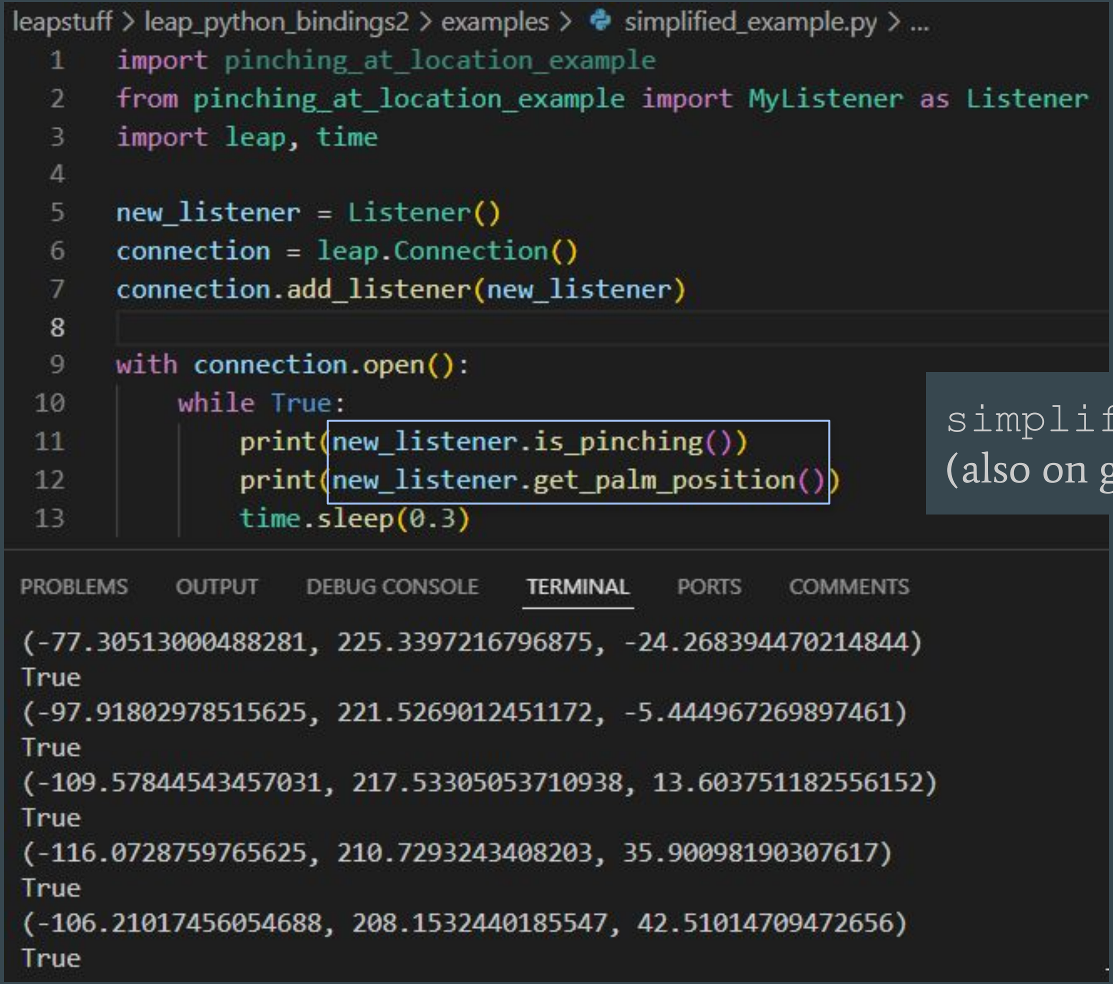
`with connection.open()` - Assures that the connection is open and closes when it needs to

`while True:` - Isn't necessary. Just needs something keeping it running

And that's it!!!

Most Important Functions

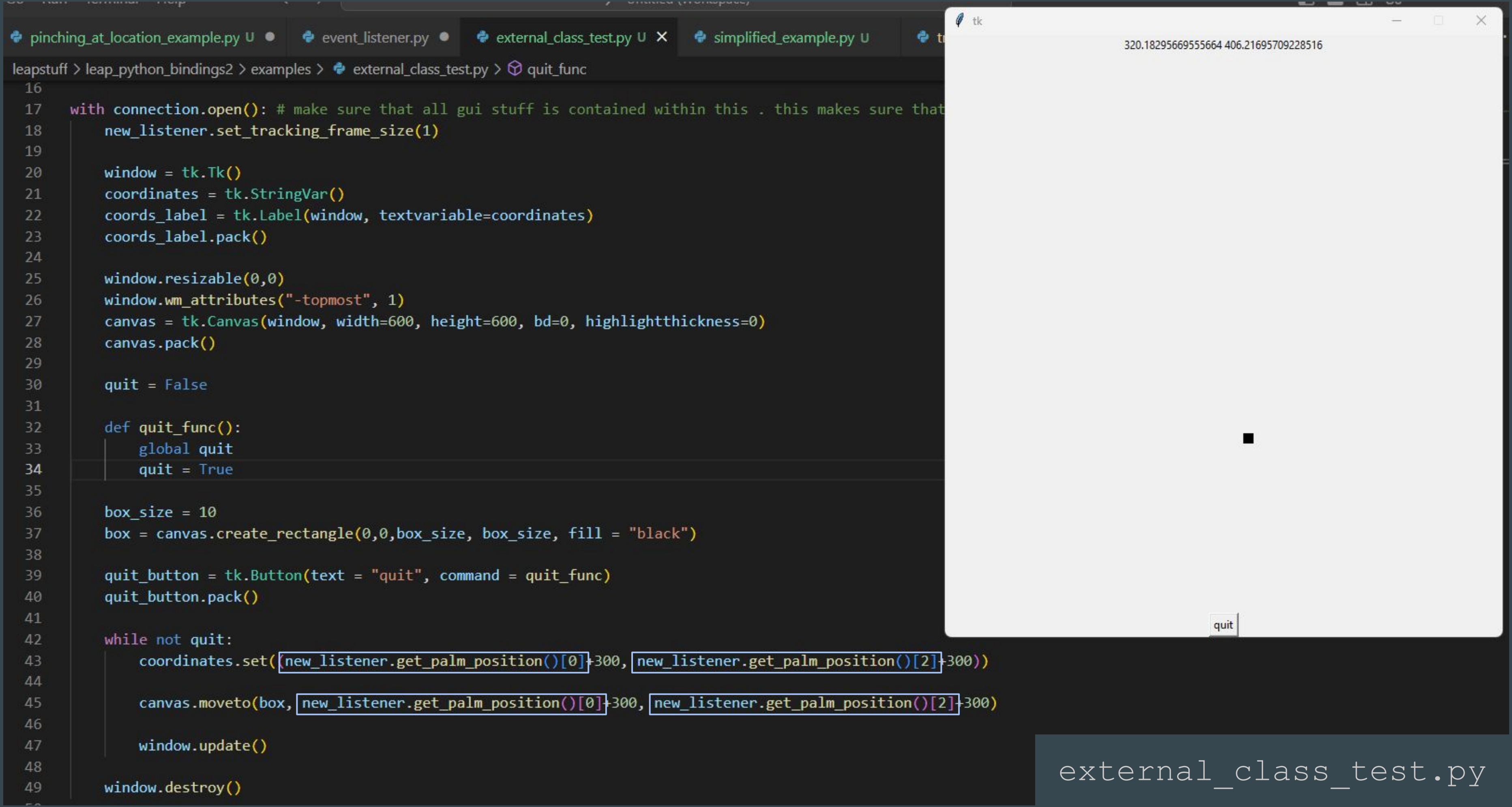
```
get_palm_position() ->
[float, float, float]
is_pinching() ->
bool
```


leapstuff > leap_python_bindings2 > examples > simplified_example.py > ...

```python
import pinching_at_location_example
from pinching_at_location_example import MyListener as Listener
import leap, time

new_listener = Listener()
connection = leap.Connection()
connection.add_listener(new_listener)

with connection.open():
    while True:
        print(new_listener.is_pinching())
        print(new_listener.get_palm_position())
        time.sleep(0.3)
```

```
(-77.30513000488281, 225.3397216796875, -24.268394470214844)
True
(-97.91802978515625, 221.5269012451172, -5.444967269897461)
True
(-109.57844543457031, 217.53305053710938, 13.603751182556152)
True
(-116.0728759765625, 210.7293243408203, 35.90098190307617)
True
(-106.21017456054688, 208.1532440185547, 42.51014709472656)
True
```

simplified_example.py
(also on github)



pinching_at_location_example.py U
event_listener.py
external_class_test.py U
simplified_example.py U
leapstuff > leap_python_bindings2 > examples > external_class_test.py > quit_func
tk
quit

```python
with connection.open(): # make sure that all gui stuff is contained within this . this makes sure that
    new_listener.set_tracking_frame_size(1)

    window = tk.Tk()
    coordinates = tk.StringVar()
    coords_label = tk.Label(window, textvariable=coordinates)
    coords_label.pack()

    window.resizable(0,0)
    window.wm_attributes("-topmost", 1)
    canvas = tk.Canvas(window, width=600, height=600, bd=0, highlightthickness=0)
    canvas.pack()

    quit = False

    def quit_func():
        global quit
        quit = True

    box_size = 10
    box = canvas.create_rectangle(0,0,box_size, box_size, fill = "black")

    quit_button = tk.Button(text = "quit", command = quit_func)
    quit_button.pack()

    while not quit:
        coordinates.set((new_listener.get_palm_position()[0]+300, new_listener.get_palm_position()[2]+300))

        canvas.moveto(box, new_listener.get_palm_position()[0]+300, new_listener.get_palm_position()[2]+300)

        window.update()

    window.destroy()
```

```
320.18295669555664 406.21695709228516
```

external_class_test.py

Other Things Exposed:

`get_hand_type() -> ‘left’ || ‘right’`

`get_pinching_vectors() -> [float, float, float]`

- returns x, y, z axes difference between index & thumb positions ([0,0,0] if not tracking)

`get_tracking_frame_id(_synced)() -> int`

- returns current frame id (synced to current tracking event, recommended)

`set/get_tracking_frame_size() -> void/int = 10`

- sets/gets interval of frames between each tracking event (does nothing if < 1)

`test_func()` - prints “`tested!!! \n\n\n`”

If you need any help or have suggestions/needs:

nrmaag@uno.edu

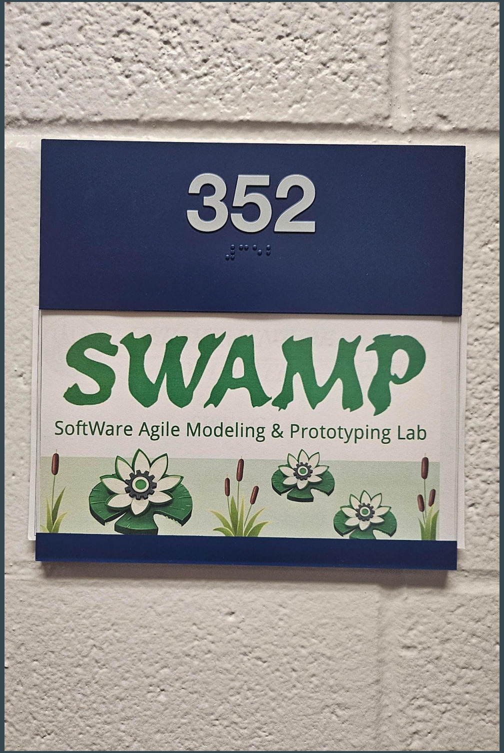
Agenda

Part 1: Leap Motion Controller

Part 2: Getting started with the Leap

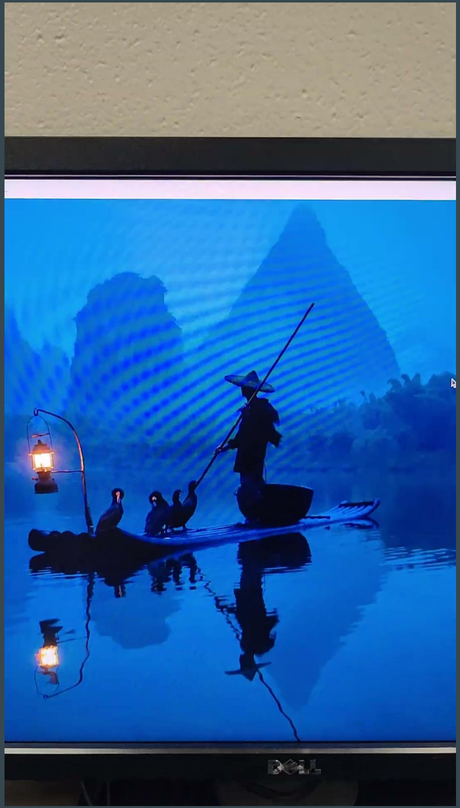
Part 3: Working in the SWAMP

Welcome to the SWAMP!



SWAMP Tour!

Using Leap on Raspberry Pi!



Deliverable

- Using either a personal PC or the SWAMP Lab workstations:
  - Submit a screenshot of you running one of the examples:

- BONUS: If you do go into the lab, upload a pic of your pod in there!