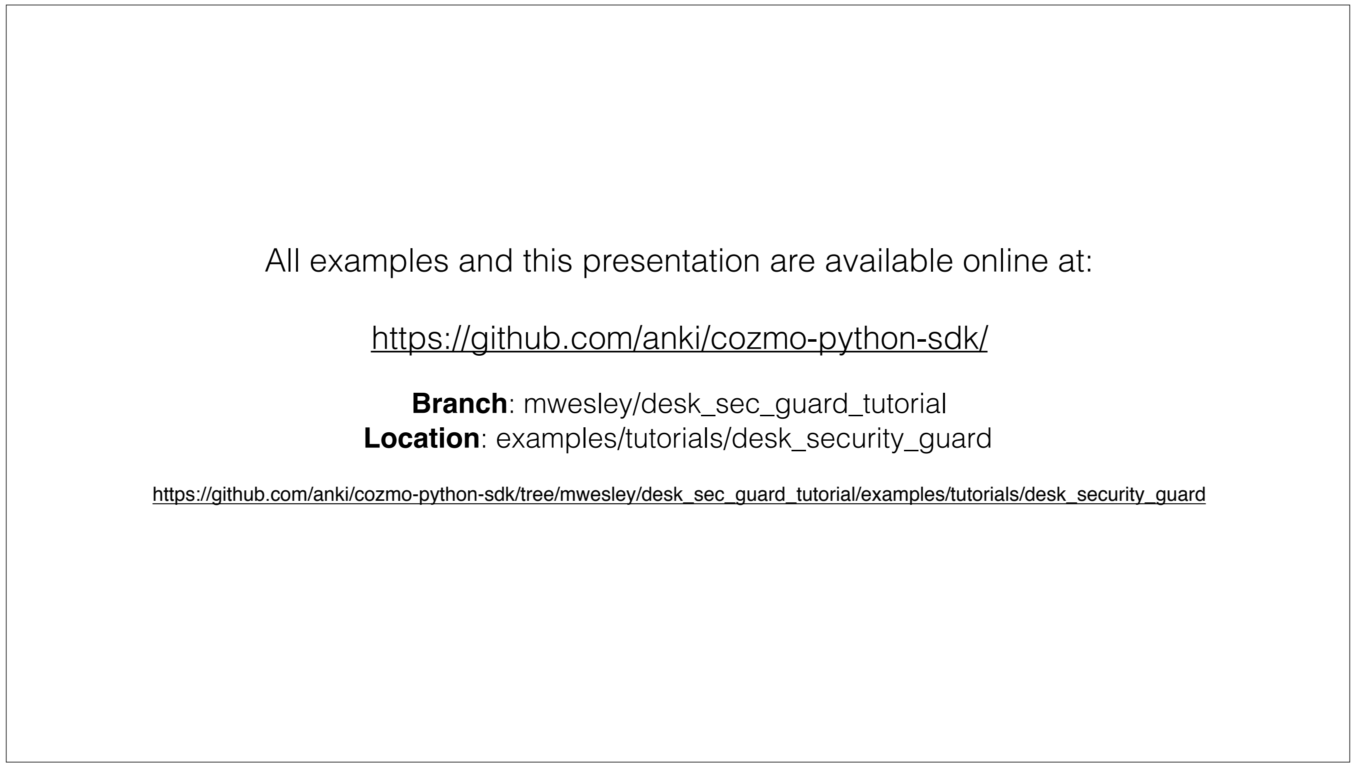
Cozmo SDK

How to build your own Desktop Security Guard

Mark Wesley
Cozmo SDK Lead @ Anki



All examples and this presentation are available online at:

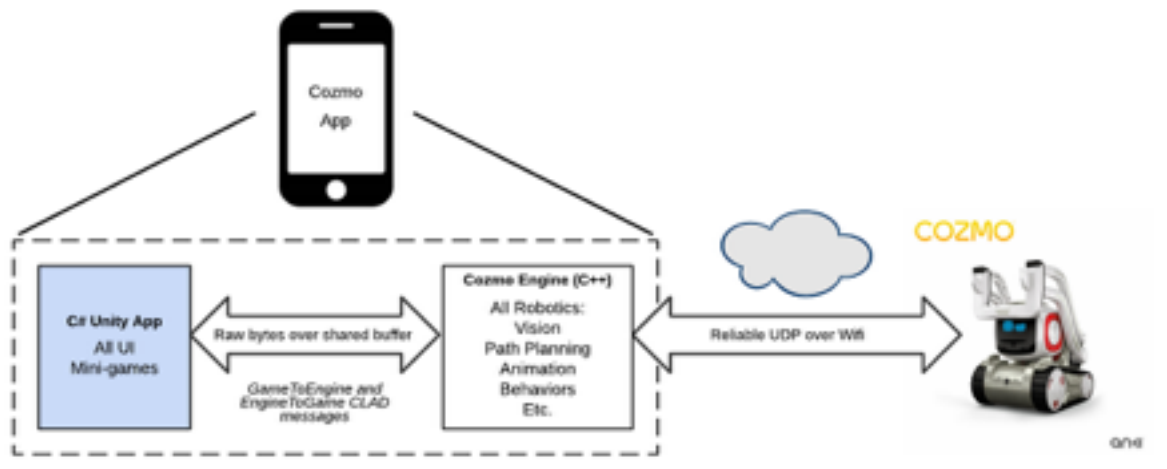
https://github.com/anki/cozmo-python-sdk/

Branch: mwesley/desk_sec_guard_tutorial
Location: examples/tutorials/desk_security_guard

https://github.com/anki/cozmo-python-sdk/tree/mwesley/desk_sec_guard_tutorial/examples/tutorials/desk_security_guard

It's in a branch currently
It will ultimately be merged into master, exact location TBD

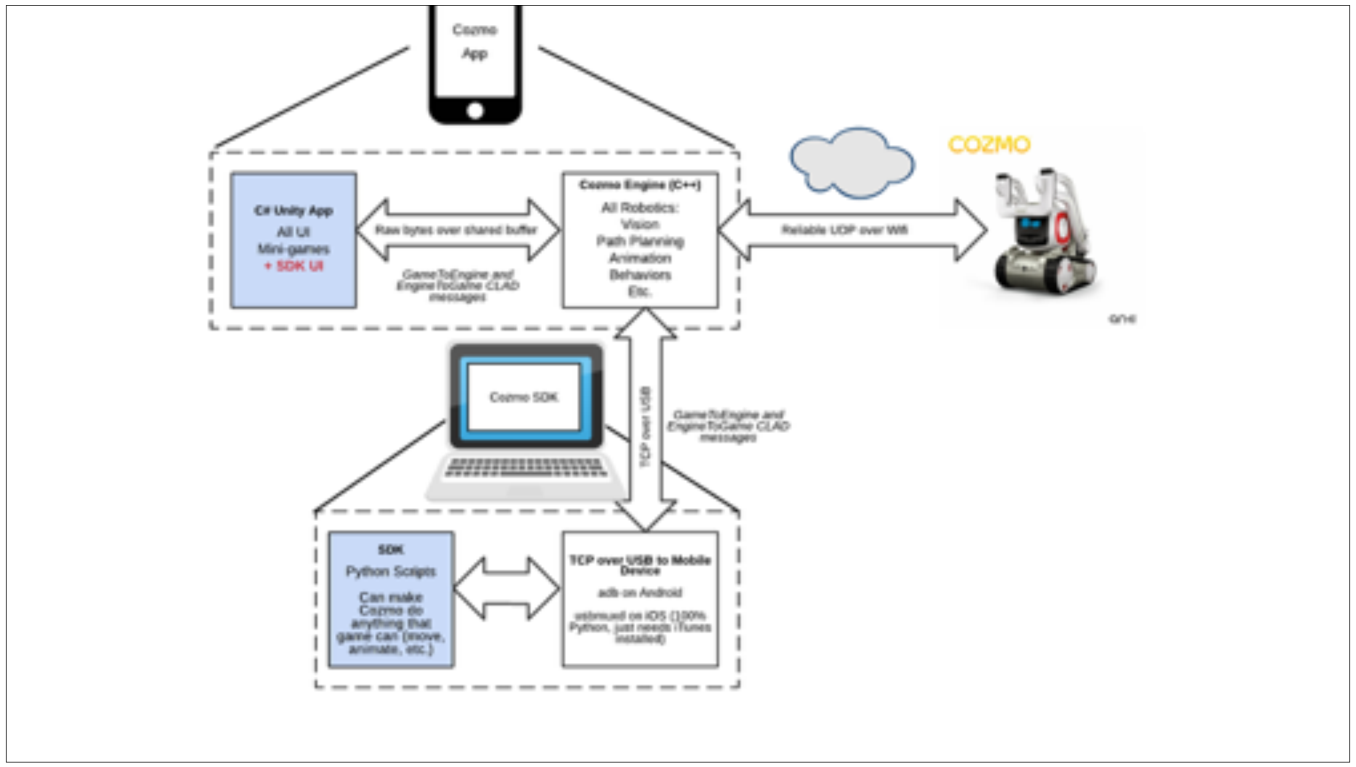
Cozmo App



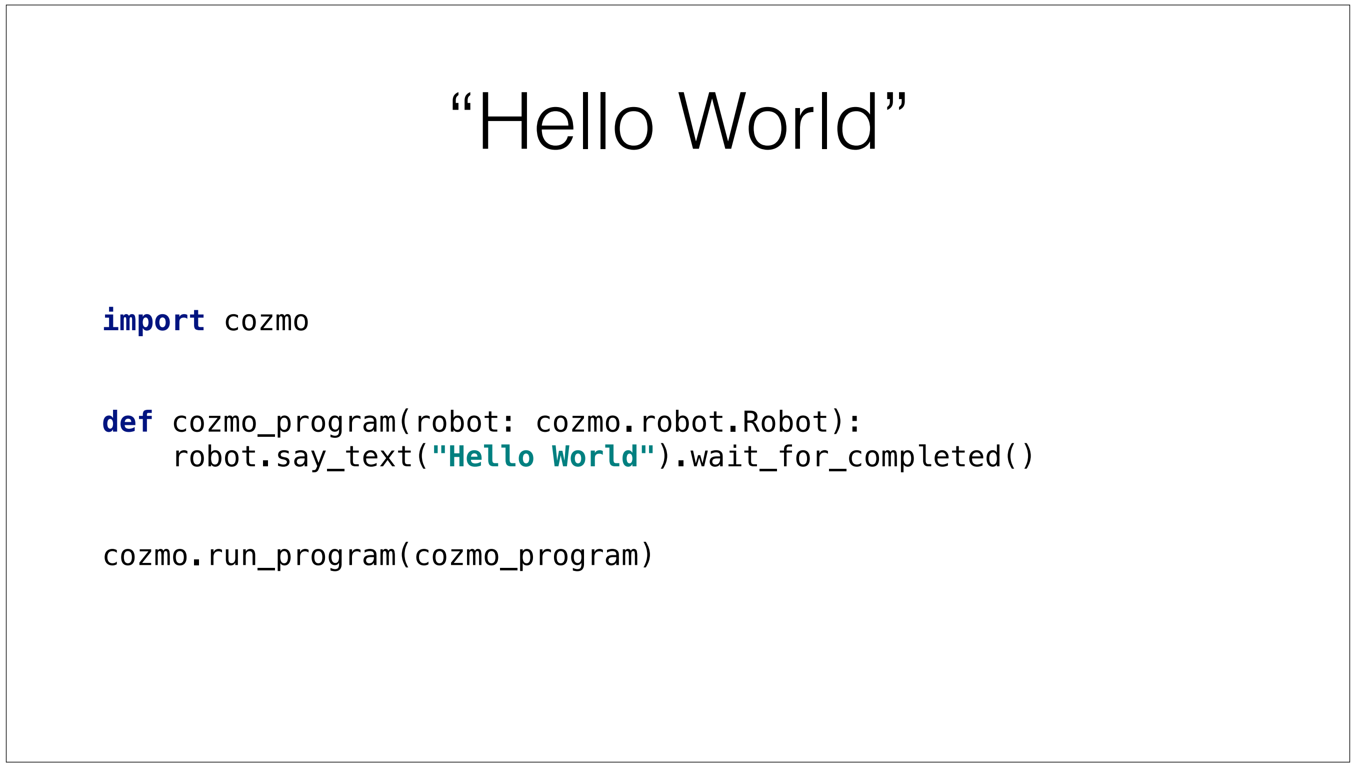
This is the high-level architecture for how the Cozmo app (running on a mobile device) talks to Cozmo.

Communication between the mobile device and the robot is over our own custom reliable UDP protocol - essentially we guarantee that certain messages will arrive, and in-order, but others are dropped (e.g. images) in favor of just getting the latest state over. This provides a number of benefits compared to e.g. TCP where everything has to arrive.

- All of the UI is in C# using Unity. The higher level logic for the mini-games (e.g. the round, the scores, etc.) is also done there.
- All of the "heavy lifting" is in C++ in what we call the Cozmo Engine. This includes all robotics, vision processing, AI planning, the animation system, actions and behaviors.
- The C# layer talks to the C++ layer using CLAD messages - CLAD (C-Like Abstract Data) is an Anki developed messaging layer which allows us to define messages (they look a lot like C structs), and we then auto generate all of the code to pack and unpack those messages for multiple languages, including C++, C# and Python. This lets us easily send data between different processes and languages, as well as over the network, without having to worry about the intricacies of how to pack or unpack them.


Cozmo App
C# Unity App
All UI
Mini-games
+ SDK UI
Raw bytes over shared buffer
GameToEngine and EngineToGame CLAD messages
Cozmo Engine (C++)
All Robotics:
Vision
Path Planning
Animation
Behaviors
Etc.
Reliable UDP over Wifi
COZMO
Cozmo SDK
TCP over USB
GameToEngine and EngineToGame CLAD messages
SDK
Python Scripts
Can make Cozmo do anything that game can (move, animate, etc.)
TCP over USB to Mobile Device
adb on Android
usbmuxd on iOS (100% Python, just needs iTunes installed)

The SDK talks to the engine via the exact same API as the app's Game/UI does
SDK communication uses TCP over USB (as that's the method available on iOS and Android)



“Hello World”

```python
import cozmo


def cozmo_program(robot: cozmo.robot.Robot):
    robot.say_text("Hello World").wait_for_completed()


cozmo.run_program(cozmo_program)
```

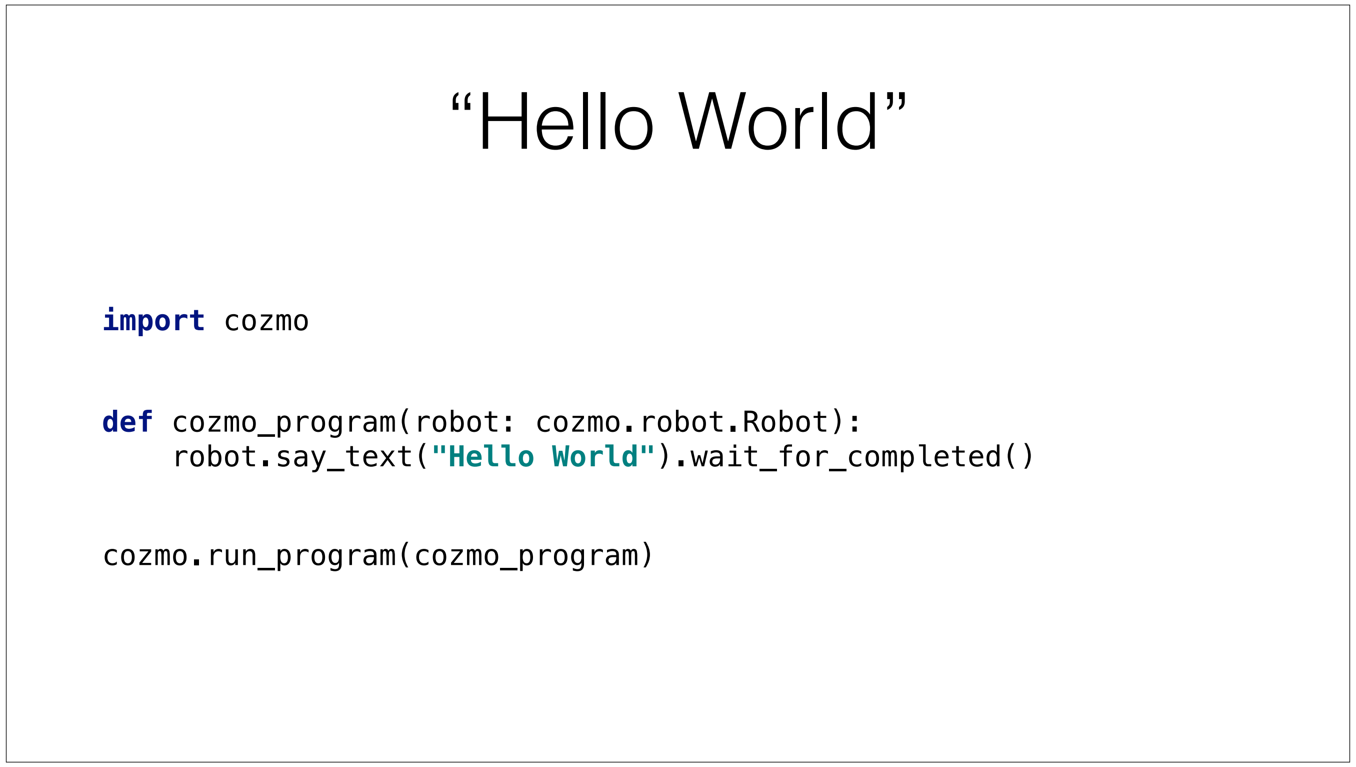
This is the “Hello World” program for the Cozmo SDK, the 1st and simplest example program that we provide.
It’s fairly simple:
We import the cozmo module
We have a method “cozmo_program” that takes in a cozmo Robot (the syntax with
“: cozmo.robot.Robot” is just to clarify to humans, and text editors like PyCharm, so it’s clear what kindof object “robot” is, note that Python doesn’t validate this at runtime, it’s just for documentation.
That method just makes Cozmo say a string, “Hello World”, and waits for that action to complete before exiting.
The final line tells the SDK to run that method - internally SDK will handle connecting to the app/engine, and calling that method whilst maintaining the connection and updating things as needed.
Note that you can call your cozmo_program method anything you like, whatever you pass in to run_program is the method that will be called.

Python asyncio

Asynchronous i/o + coroutine support
Introduced in Python 3.4
Improved in Python 3.5.1
Cozmo SDK requires Python 3.5.1 (or above)

Single threaded
A simpler way to manage asynchronous code than traditional threads

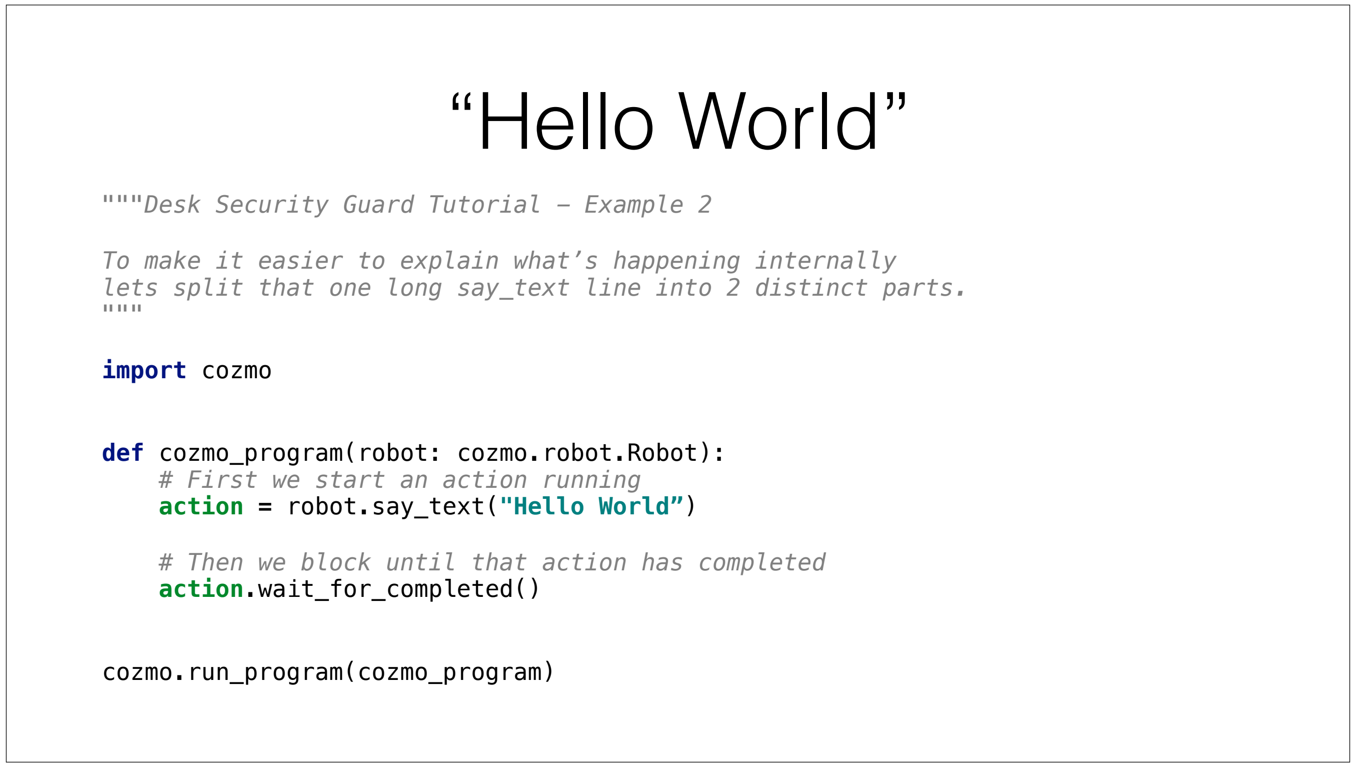
Used internally by the Cozmo SDK



“Hello World”

```python
import cozmo


def cozmo_program(robot: cozmo.robot.Robot):
    robot.say_text("Hello World").wait_for_completed()


cozmo.run_program(cozmo_program)
```

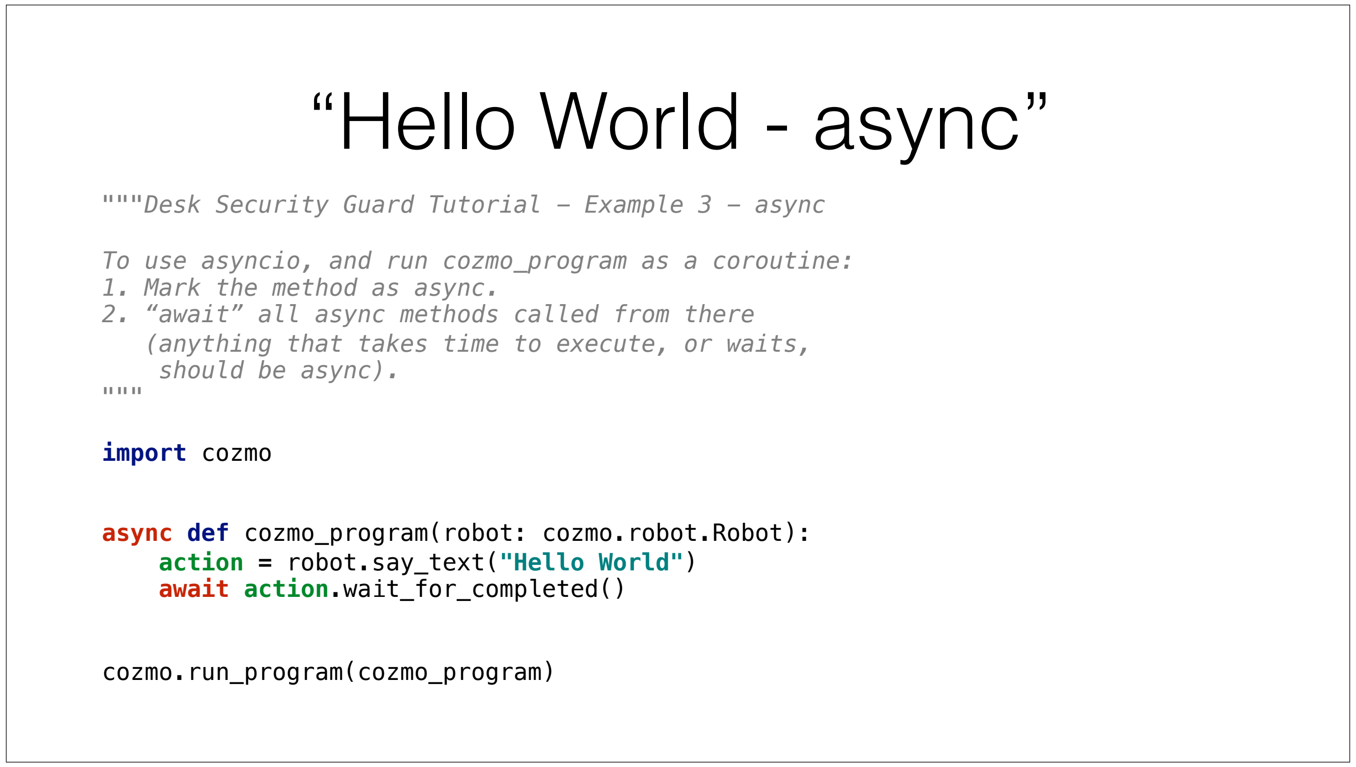
So, lets look back at that Hello World script



“Hello World”

```
"""Desk Security Guard Tutorial - Example 2

To make it easier to explain what’s happening internally
lets split that one long say_text line into 2 distinct parts.
"""

import cozmo


def cozmo_program(robot: cozmo.robot.Robot):
    # First we start an action running
    action = robot.say_text("Hello World”)

    # Then we block until that action has completed
    action.wait_for_completed()


cozmo.run_program(cozmo_program)
```

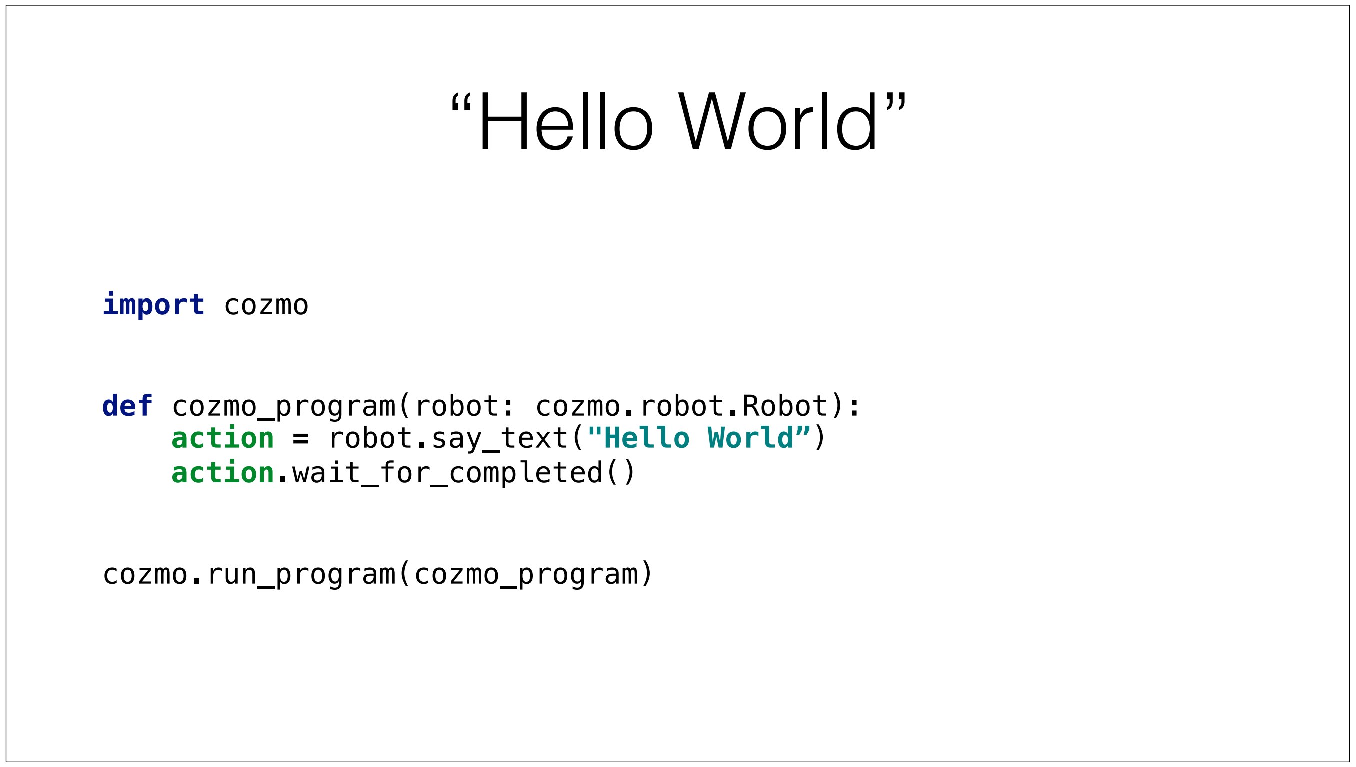
Now, to make it easier to explain what’s happening internally lets split that say_text line into 2 distinct parts.
First we create an action and start it running on Cozmo
Then we block until that action has completed.



“Hello World - async”

```python
"""Desk Security Guard Tutorial - Example 3 - async

To use asyncio, and run cozmo_program as a coroutine:
1. Mark the method as async.
2. “await” all async methods called from there
   (anything that takes time to execute, or waits,
    should be async).
"""

import cozmo


async def cozmo_program(robot: cozmo.robot.Robot):
    action = robot.say_text("Hello World")
    await action.wait_for_completed()


cozmo.run_program(cozmo_program)
```

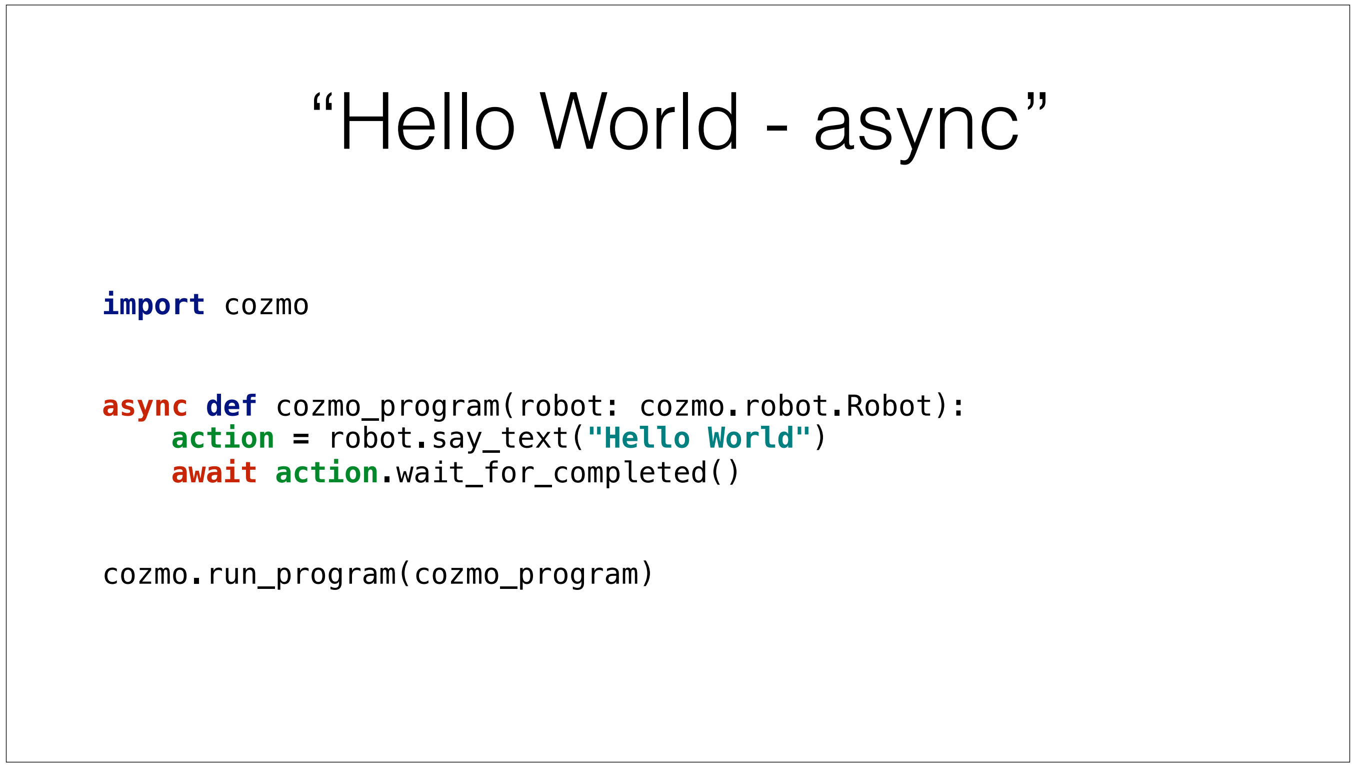
To use asyncio in your programs directly, you need to mark your program as “async” to specify that it is a coroutine.
Any async methods that you call needed to be “await”-ed,
Generally any methods that are slow, or take time to execute, especially if they wait for anything, should be async.
So we mark cozmo_program as async, and we await the “wait_for_completed” method as that is also an async method (it’s waiting for the robot to confirm it completed, which will take a couple of seconds for Cozmo to say that line)
With an async cozmo program we are assured that the underlying SDK and all variables will only be updated whilst you are “await”-ing another async call.
With a sync cozmo_program the SDK is running in the background in another thread - this works fine for the simple example programs, but as you start to build more complex programs you could run into threading bugs where e.g. some part of the robot’s state changes between 2 lines of your program, which you might not be expecting or handling.



“Hello World”

```
import cozmo


def cozmo_program(robot: cozmo.robot.Robot):
    action = robot.say_text("Hello World”)
    action.wait_for_completed()


cozmo.run_program(cozmo_program)
```

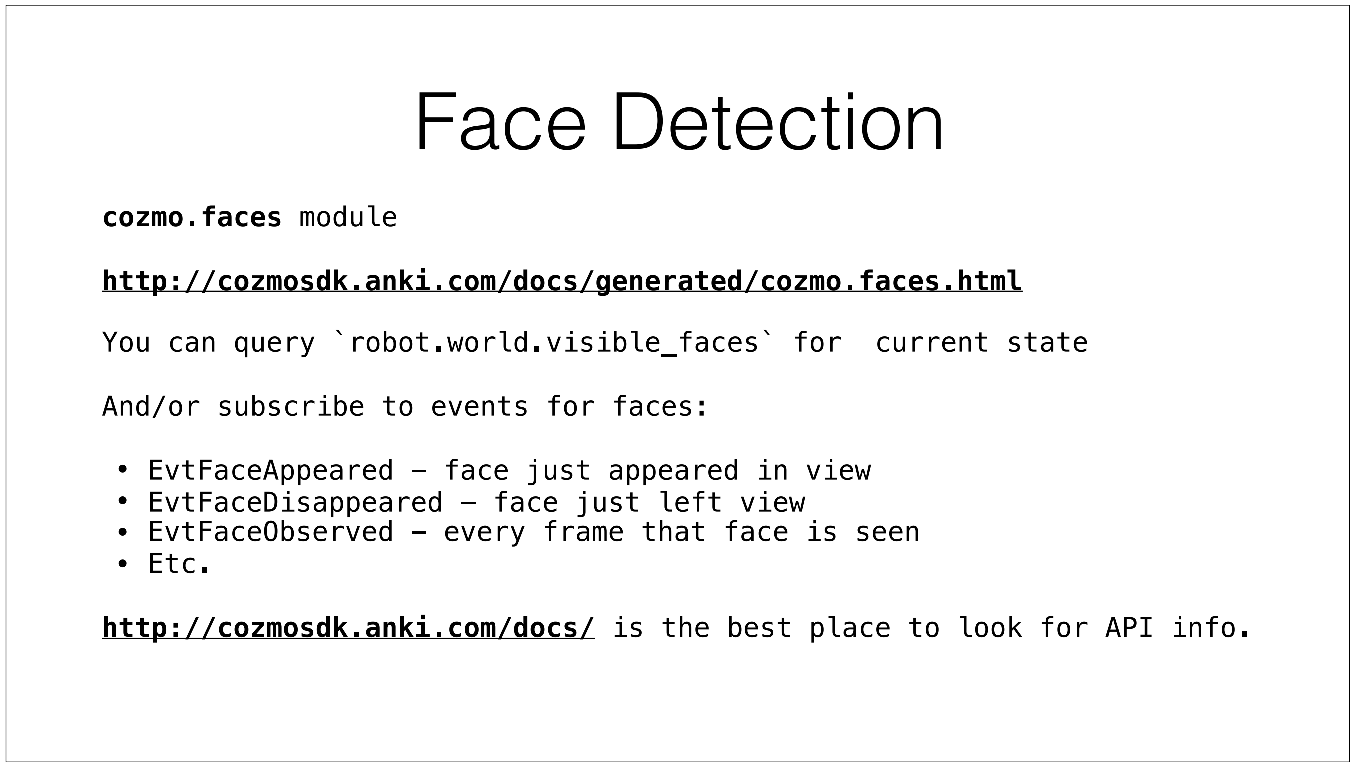
To refresh your memory - this was the “sync” hello world



“Hello World - async”

```python
import cozmo


async def cozmo_program(robot: cozmo.robot.Robot):
    action = robot.say_text("Hello World")
    await action.wait_for_completed()


cozmo.run_program(cozmo_program)
```

And this is the asyncio version - very similar really, and you’ll get used to the asyncio syntax and differences quickly.
Note: If you ever see a runtime error like *“RuntimeWarning: coroutine ‘foo’ was never awaited”* that suggests that you’ve called `foo(…)` where you should have done `await foo(…)`



Face Detection

cozmo.faces module

http://cozmosdk.anki.com/docs/generated/cozmo.faces.html

You can query `robot.world.visible_faces` for current state

And/or subscribe to events for faces:

- EvtFaceAppeared – face just appeared in view
- EvtFaceDisappeared – face just left view
- EvtFaceObserved – every frame that face is seen
- Etc.

http://cozmosdk.anki.com/docs/ is the best place to look for API info.

Cozmo's face detection support is exposed via the **cozmo.faces** module

You can access the faces directly from the world object, and you can also subscribe to a variety of events for whenever faces are seen, and when they appear or disappear from view.

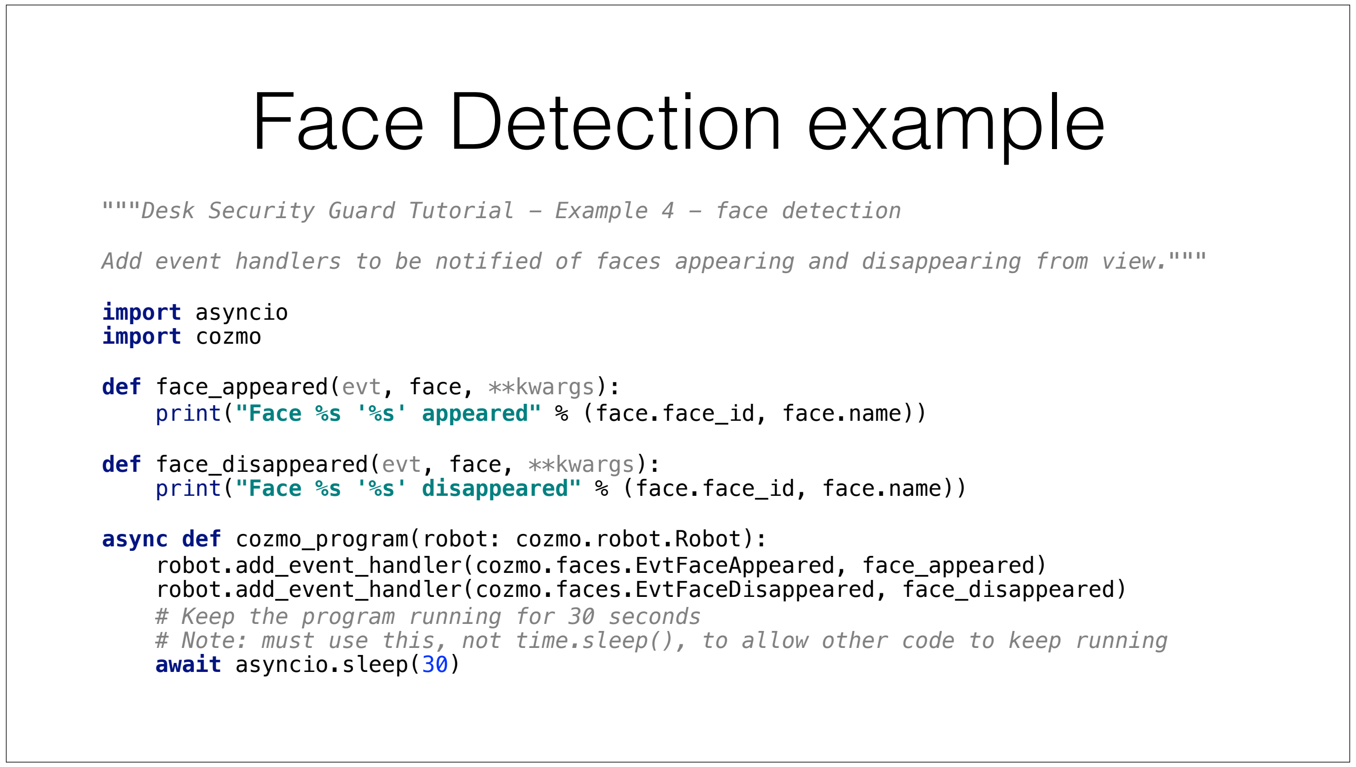
Note: Generally the best place to look for API details is in our online documentation:

http://cozmosdk.anki.com/docs/

There's a search box in the top left, so you can search for anything.

You can also browse the code locally or on Github:

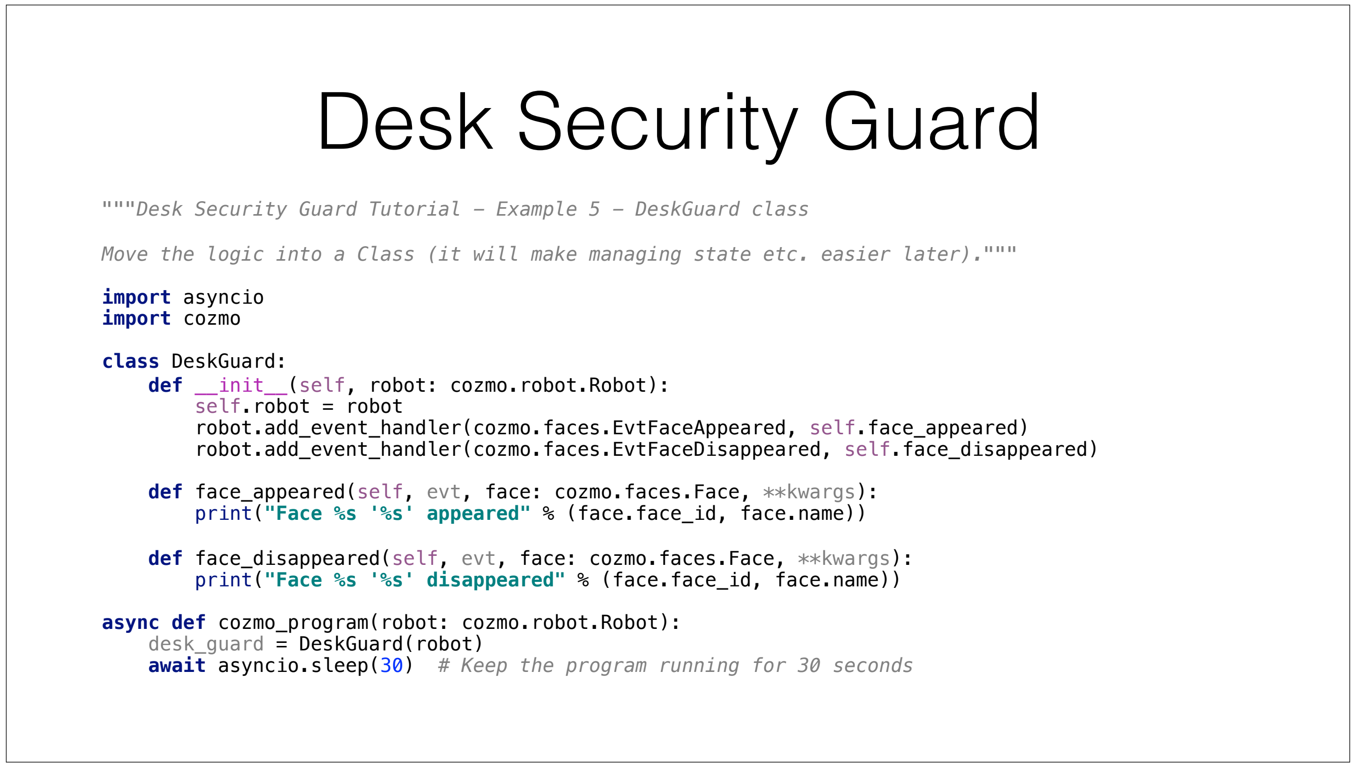
https://github.com/anki/cozmo-python-sdk



Face Detection example

```python
"""Desk Security Guard Tutorial - Example 4 - face detection

Add event handlers to be notified of faces appearing and disappearing from view."""

import asyncio
import cozmo

def face_appeared(evt, face, **kwargs):
    print("Face %s '%s' appeared" % (face.face_id, face.name))

def face_disappeared(evt, face, **kwargs):
    print("Face %s '%s' disappeared" % (face.face_id, face.name))

async def cozmo_program(robot: cozmo.robot.Robot):
    robot.add_event_handler(cozmo.faces.EvtFaceAppeared, face_appeared)
    robot.add_event_handler(cozmo.faces.EvtFaceDisappeared, face_disappeared)
    # Keep the program running for 30 seconds
    # Note: must use this, not time.sleep(), to allow other code to keep running
    await asyncio.sleep(30)
```

Cozmo event methods all take an event object (named “evt” here), followed by a bunch of keyword arguments for that event. See the documentation for that event to see the available arguments.
****kwargs** will capture all subsequent additional arguments as a dictionary of arguments indexed by keyword.
This example will print each time a face comes into view or leaves, so you can play peekaboo with Cozmo.



Desk Security Guard

```python
"""Desk Security Guard Tutorial - Example 5 - DeskGuard class

Move the logic into a Class (it will make managing state etc. easier later)."""

import asyncio
import cozmo

class DeskGuard:
    def __init__(self, robot: cozmo.robot.Robot):
        self.robot = robot
        robot.add_event_handler(cozmo.faces.EvtFaceAppeared, self.face_appeared)
        robot.add_event_handler(cozmo.faces.EvtFaceDisappeared, self.face_disappeared)

    def face_appeared(self, evt, face: cozmo.faces.Face, **kwargs):
        print("Face %s '%s' appeared" % (face.face_id, face.name))

    def face_disappeared(self, evt, face: cozmo.faces.Face, **kwargs):
        print("Face %s '%s' disappeared" % (face.face_id, face.name))

async def cozmo_program(robot: cozmo.robot.Robot):
    desk_guard = DeskGuard(robot)
    await asyncio.sleep(30)  # Keep the program running for 30 seconds
```

Lets start moving this logic into a Class, this will make it easier for us to have a bunch of state that we'll maintain for this program.

Note: You can subclass Robot and tell the SDK to create that instance instead by adding:
cozmo.conn.CozmoConnection.robot_factory = MyRobotClass
before calling cozmo.run_program, but unless you're overriding methods or need Polymorphism then it's cleaner to use a “has-a” model instead of “is-a”

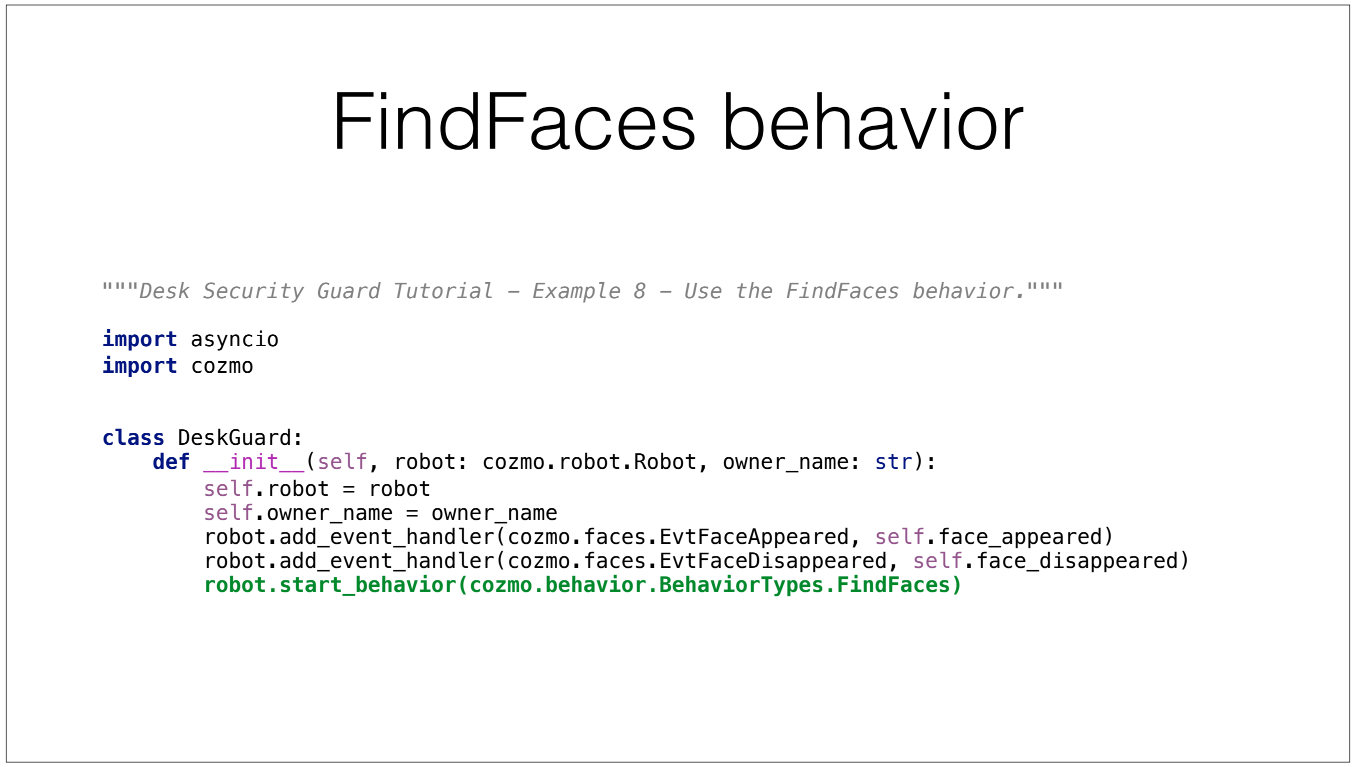
Desk Security Guard

```
"""Desk Security Guard Tutorial - Example 6 - Greet Owner when we see them."""
import asyncio
import cozmo

class DeskGuard:
    def __init__(self, robot: cozmo.robot.Robot, owner_name: str):
        self.robot = robot
        self.owner_name = owner_name
        robot.add_event_handler(cozmo.faces.EvtFaceAppeared, self.face_appeared)
        robot.add_event_handler(cozmo.faces.EvtFaceDisappeared, self.face_disappeared)

    def face_appeared(self, evt, face: cozmo.faces.Face, **kwargs):
        if face.name == self.owner_name:
            self.robot.play_anim_trigger(cozmo.anim.Triggers.NamedFaceInitialGreeting,
                                         in_parallel=True)

    def face_disappeared(self, evt, face: cozmo.faces.Face, **kwargs):
        print("Face %s '%s' disappeared" % (face.face_id, face.name))

async def cozmo_program(robot: cozmo.robot.Robot):
    desk_guard = DeskGuard(robot, "Wez")
    await asyncio.sleep(30)  # Keep the program running for 30 seconds
```

We mark the actions as in_parallel=True so that you can have multiple actions at once without triggering an error.

Desk Security Guard

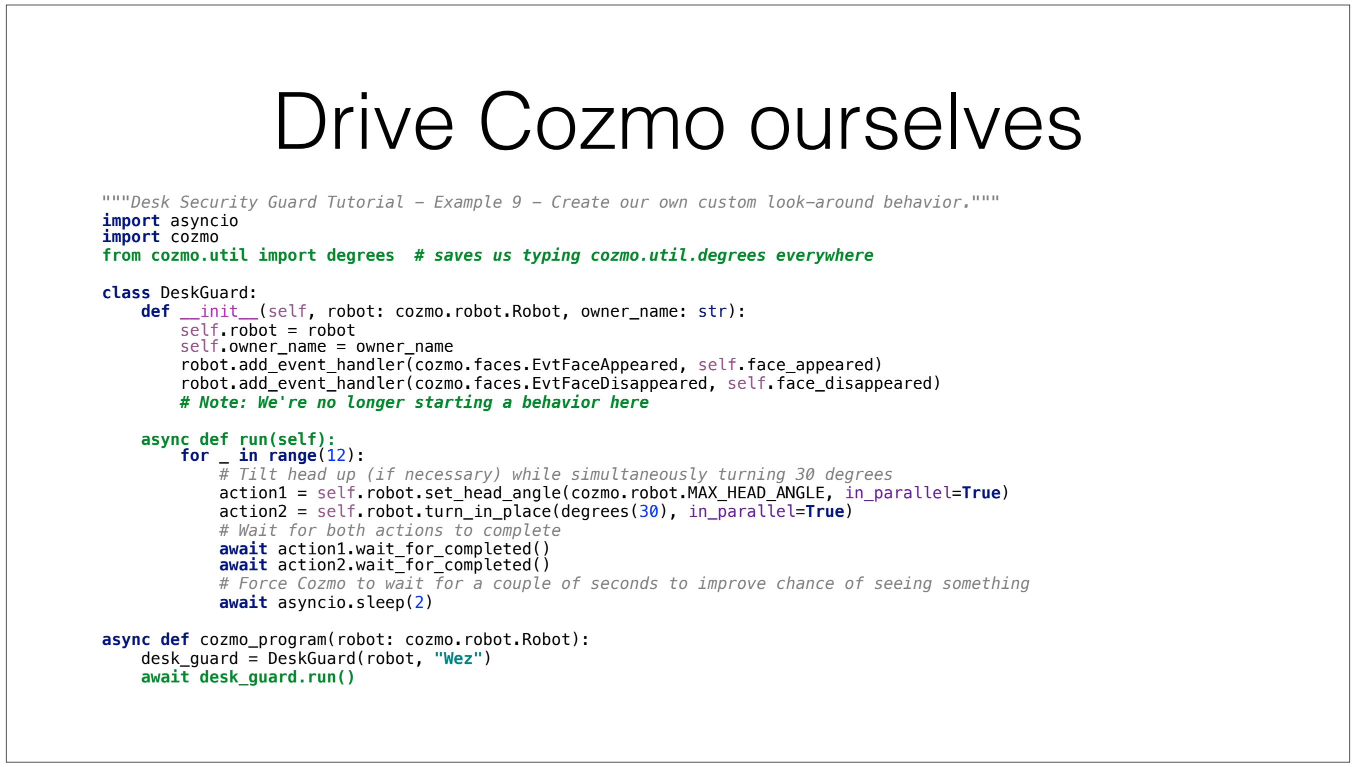
```
"""Desk Security Guard Tutorial - Example 7 - Alert on Intruders when seen."""
import asyncio
import cozmo

class DeskGuard:
    def __init__(self, robot: cozmo.robot.Robot, owner_name: str):
        self.robot = robot
        self.owner_name = owner_name
        robot.add_event_handler(cozmo.faces.EvtFaceAppeared, self.face_appeared)
        robot.add_event_handler(cozmo.faces.EvtFaceDisappeared, self.face_disappeared)

    def face_appeared(self, evt, face: cozmo.faces.Face, **kwargs):
        if face.name == self.owner_name:
            self.robot.play_anim_trigger(cozmo.anim.Triggers.NamedFaceInitialGreeting,
                                         in_parallel=True)
        else:
            self.robot.say_text(“Intruder Alert!”, in_parallel=True)

    def face_disappeared(self, evt, face: cozmo.faces.Face, **kwargs):
        print("Face %s '%s' disappeared" % (face.face_id, face.name))

async def cozmo_program(robot: cozmo.robot.Robot):
    desk_guard = DeskGuard(robot, "Wez")
    await asyncio.sleep(30)  # Keep the program running for 30 seconds
```



FindFaces behavior

```python
"""Desk Security Guard Tutorial - Example 8 - Use the FindFaces behavior."""

import asyncio
import cozmo


class DeskGuard:
    def __init__(self, robot: cozmo.robot.Robot, owner_name: str):
        self.robot = robot
        self.owner_name = owner_name
        robot.add_event_handler(cozmo.faces.EvtFaceAppeared, self.face_appeared)
        robot.add_event_handler(cozmo.faces.EvtFaceDisappeared, self.face_disappeared)
        robot.start_behavior(cozmo.behavior.BehaviorTypes.FindFaces)
```

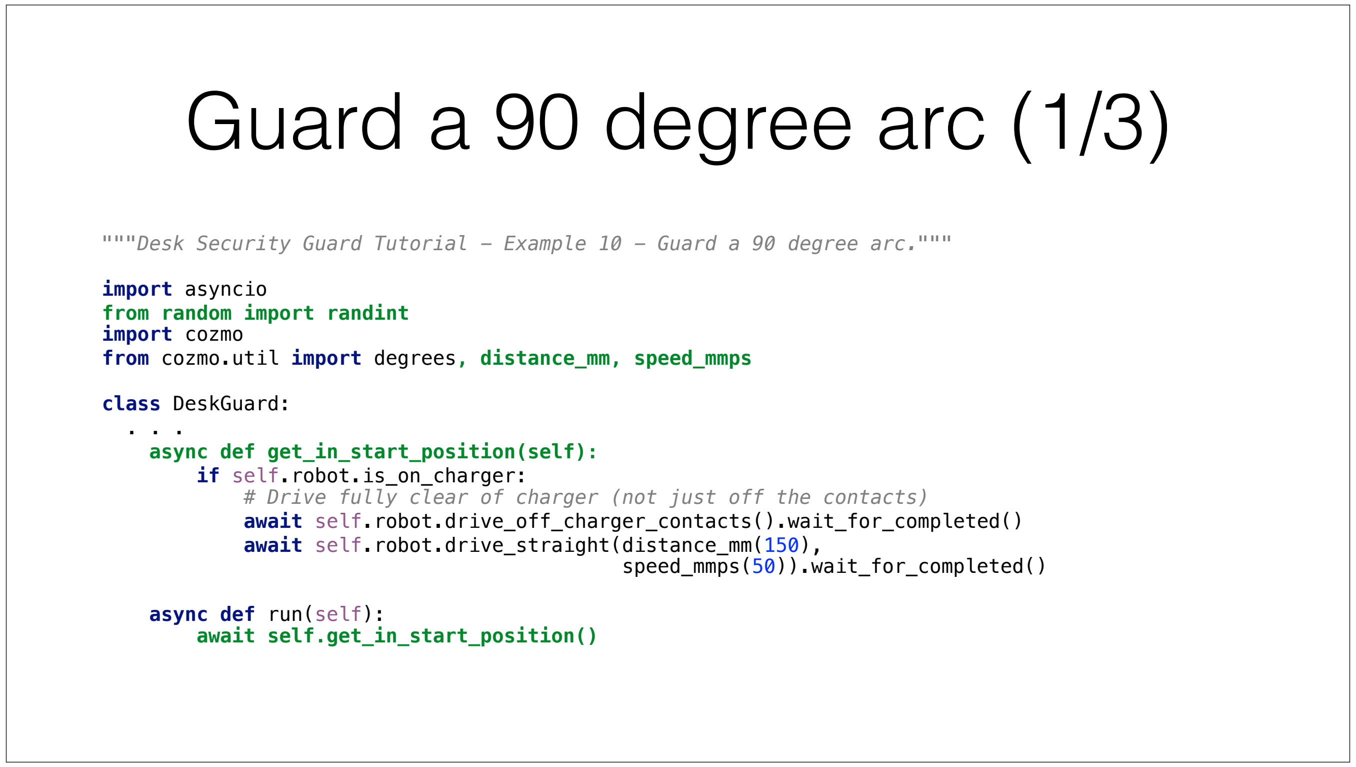
Tell Cozmo to use the FindFaces behavior at the start - this will make Cozmo turn around, whilst looking up, in an effort to see any faces around him.
Behaviors give easy access to high level behavioral functionality
But they take pretty much complete control of the robot, so can fight with anything else that you’re trying to do.



Drive Cozmo ourselves

```python
"""Desk Security Guard Tutorial - Example 9 - Create our own custom look-around behavior."""
import asyncio
import cozmo
from cozmo.util import degrees  # saves us typing cozmo.util.degrees everywhere

class DeskGuard:
    def __init__(self, robot: cozmo.robot.Robot, owner_name: str):
        self.robot = robot
        self.owner_name = owner_name
        robot.add_event_handler(cozmo.faces.EvtFaceAppeared, self.face_appeared)
        robot.add_event_handler(cozmo.faces.EvtFaceDisappeared, self.face_disappeared)
        # Note: We're no longer starting a behavior here

    async def run(self):
        for _ in range(12):
            # Tilt head up (if necessary) while simultaneously turning 30 degrees
            action1 = self.robot.set_head_angle(cozmo.robot.MAX_HEAD_ANGLE, in_parallel=True)
            action2 = self.robot.turn_in_place(degrees(30), in_parallel=True)
            # Wait for both actions to complete
            await action1.wait_for_completed()
            await action2.wait_for_completed()
            # Force Cozmo to wait for a couple of seconds to improve chance of seeing something
            await asyncio.sleep(2)

async def cozmo_program(robot: cozmo.robot.Robot):
    desk_guard = DeskGuard(robot, "Wez")
    await desk_guard.run()
```

We remove the use of the behavior, and implement our own simplified version using parallel actions
Because all of the actions are marked in_parallel they won't interrupt each other, and can run together. If two actions need one animation track (face, lift, head, wheels, audio, etc.) the 2nd one will fail and complete immediately.
Waiting between turns is important to allow vision a chance to recognize the image. (While Cozmo is turning, things like rolling shutter on the camera (even though we compensate for it internally), and other issues, can make it hard for Cozmo to observe something whilst turning.)
Cozmo will perform 12 turns of 30 degrees - so will do one turn. Because the greeting animation also turns Cozmo, if Cozmo sees his owner then he will end up pointing in a completely different direction. This is also why it's necessary to always ensure the head is looking up each time.
Note: the underscore in the for-loop indicates that we're not using the value, we just want to repeat 12 times.



Guard a 90 degree arc (1/3)

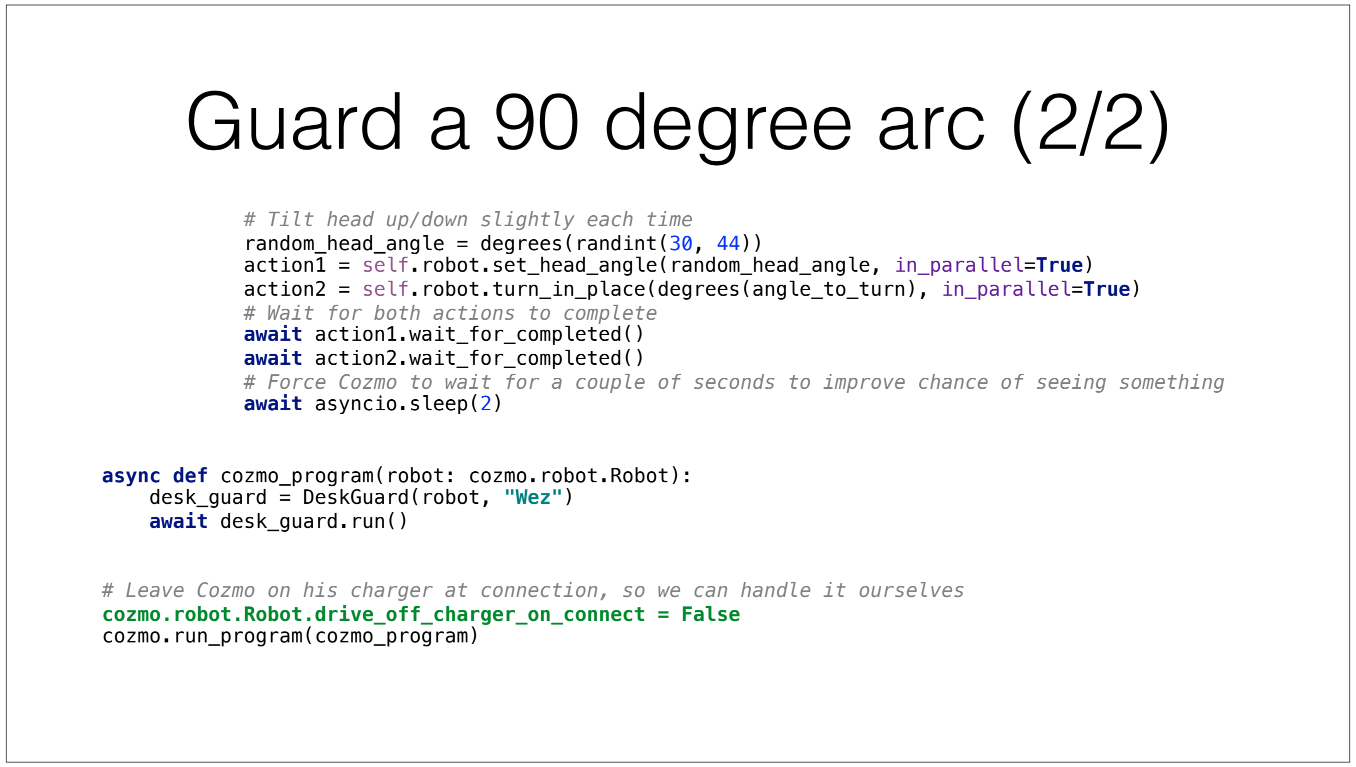
```python
"""Desk Security Guard Tutorial - Example 10 - Guard a 90 degree arc."""

import asyncio
from random import randint
import cozmo
from cozmo.util import degrees, distance_mm, speed_mmps

class DeskGuard:
  . . .
    async def get_in_start_position(self):
        if self.robot.is_on_charger:
            # Drive fully clear of charger (not just off the contacts)
            await self.robot.drive_off_charger_contacts().wait_for_completed()
            await self.robot.drive_straight(distance_mm(150),
                                            speed_mmps(50)).wait_for_completed()

    async def run(self):
        await self.get_in_start_position()
```

- We're importing the randint method from random (you'll see why on one of the following slides)
- We're also importing a couple more things from util - you can see why if you look at the the following new method:
- get_in_start_position - this ensures we're off the charger contacts (to avoid overloading anything, no other motor actions are allowed when Cozmo is drawing current from the charger), and an additional 150mm forward to be completely clear of the charger.
- We call this method at the start now before doing anything else

Guard a 90 degree arc (2/3)

```python
async def run(self):
      await self.get_in_start_position()

      initial_pose_angle = self.robot.pose_angle
      max_offset_angle = 45  # max offset in degrees: Either +ve (left), or -ve (right)
      turn_scalar = 1  # 1 will increase the offset (turn left), -1 will do the opposite

      for _ in range(12):
          # pick a random amount to turn in the current direction
          angle_to_turn = turn_scalar * randint(10,40)

          # Find how far robot is already turned, and calculate the new offset
          current_angle_offset = (self.robot.pose_angle - initial_pose_angle).degrees
          new_angle_offset = current_angle_offset + angle_to_turn

          # Clamp the turn to the desired offsets
          if new_angle_offset > max_offset_angle:
              angle_to_turn = max_offset_angle - current_angle_offset
              turn_scalar = -1  # turn the other direction next time
          elif new_angle_offset < -max_offset_angle:
              angle_to_turn = -max_offset_angle - current_angle_offset
              turn_scalar = 1  # turn the other direction next time
```

- We pick a random angle to turn in the current turn direction
- We find what that new offset would be relative to the start pose
- If this is beyond either the min or the max offset then we clamp it, and reverse the turning direction for next time



Guard a 90 degree arc (2/2)

```python
            # Tilt head up/down slightly each time
            random_head_angle = degrees(randint(30, 44))
            action1 = self.robot.set_head_angle(random_head_angle, in_parallel=True)
            action2 = self.robot.turn_in_place(degrees(angle_to_turn), in_parallel=True)
            # Wait for both actions to complete
            await action1.wait_for_completed()
            await action2.wait_for_completed()
            # Force Cozmo to wait for a couple of seconds to improve chance of seeing something
            await asyncio.sleep(2)


async def cozmo_program(robot: cozmo.robot.Robot):
    desk_guard = DeskGuard(robot, "Wez")
    await desk_guard.run()


# Leave Cozmo on his charger at connection, so we can handle it ourselves
cozmo.robot.Robot.drive_off_charger_on_connect = False
cozmo.run_program(cozmo_program)
```

- We pick a random head angle (but fairly upward) to turn too as well
- Note that we also tell Cozmo to stay on the charger on connecting - this is so that we can handle the behavior for if he’s on the charger at the start, otherwise he’d always be just off the contacts but still too close to turn left/right

Integrating other APIs

There is a Python module available for almost anything you can think of.
Makes integrating with online services easy.
E.g. talk to Twitter, Instagram, IFTTT, IoT, smart light bulbs, etc. etc.

Example - Twitter

There are many Twitter modules
We’re going to use Tweepy

```
pip3 install --user tweepy
```

Look in the lib subdirectory in the examples folder:

twitter_helpers-py – helper methods
cozmo_twitter_keys.py – put your private Twitter keys here

Twitter - setup

Steps are listed in **cozmo_twitter_keys.py:**

1. Go to **https://twitter.com/**
2. Logout of your account if you already have one
3. Create a Twitter account for your Cozmo
4. Make sure you’re logged into that account

Twitter - app setup

Steps are listed in **cozmo_twitter_keys.py:**

Go to **https://apps.twitter.com/app/new** and create your application:

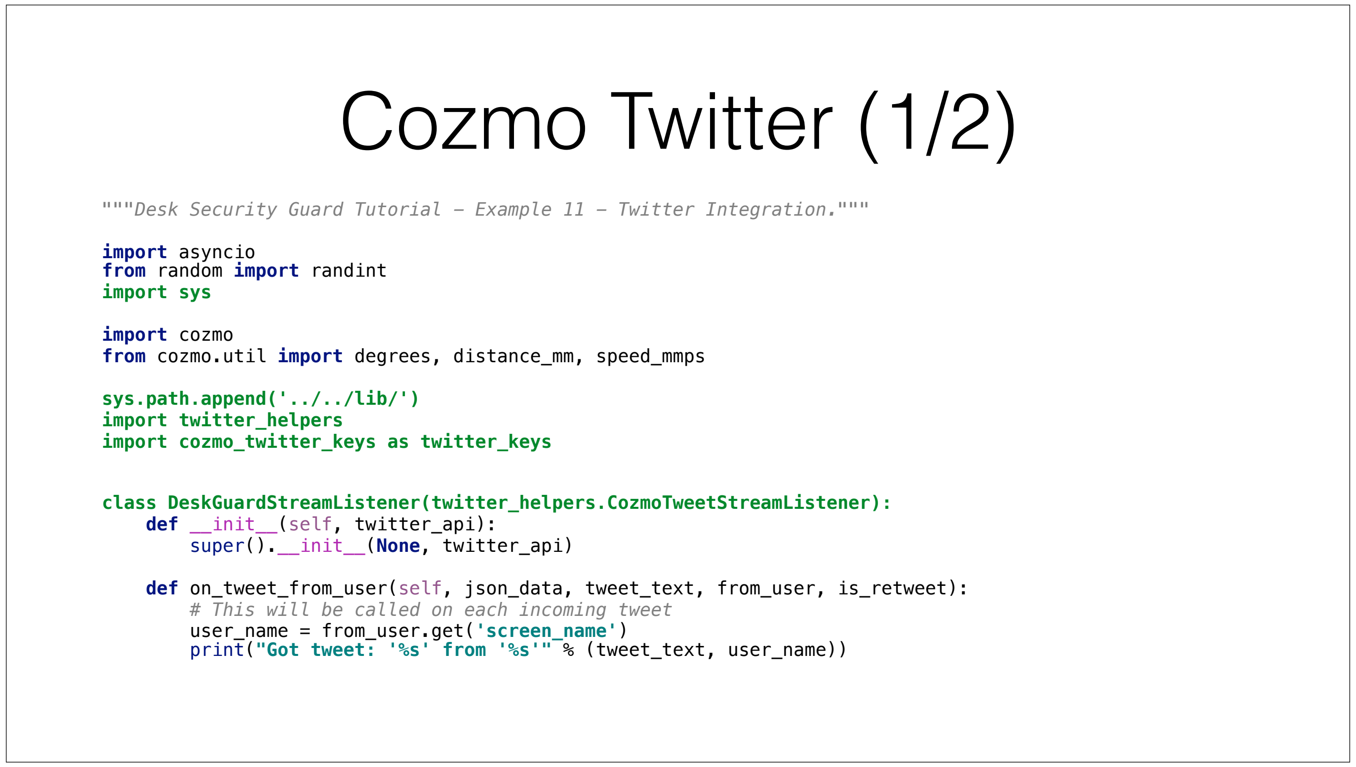
a) Fill in the name and details etc**.** (most are optional)

b) Select "Permissions" tab and set Access to Read and Write (this example needs to read and write tweets)

c) Select **"Keys and Access Tokens"** tab and click **"Generate an Access Token and Secret"**

d) Paste your consumer key + secret**,** and access token + secret into the XXXXXXXXXX fields in **cozmo_twitter_keys.py**

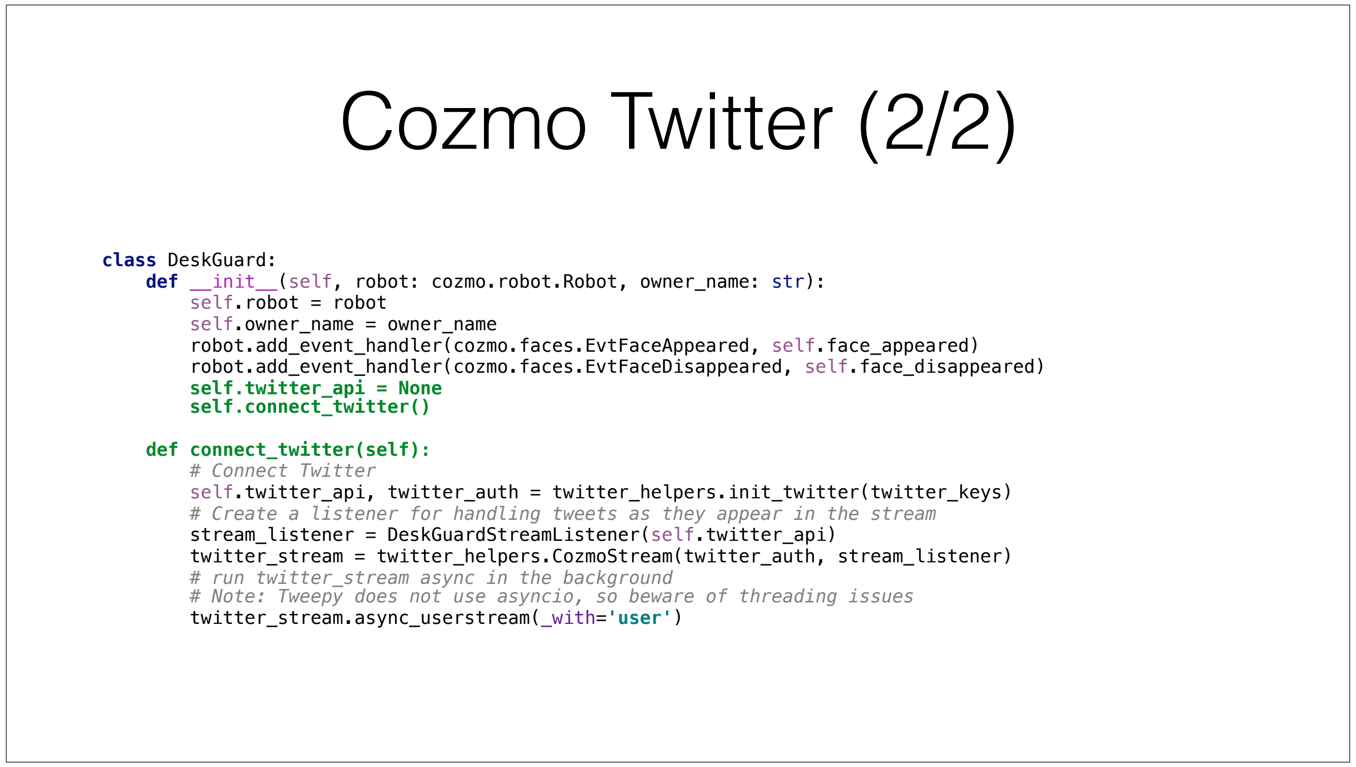
Keep that file + the keys private and safe!



Cozmo Twitter (1/2)

```python
"""Desk Security Guard Tutorial - Example 11 - Twitter Integration."""

import asyncio
from random import randint
import sys

import cozmo
from cozmo.util import degrees, distance_mm, speed_mmps

sys.path.append('../../lib/')
import twitter_helpers
import cozmo_twitter_keys as twitter_keys


class DeskGuardStreamListener(twitter_helpers.CozmoTweetStreamListener):
    def __init__(self, twitter_api):
        super().__init__(None, twitter_api)

    def on_tweet_from_user(self, json_data, tweet_text, from_user, is_retweet):
        # This will be called on each incoming tweet
        user_name = from_user.get('screen_name')
        print("Got tweet: '%s' from '%s'" % (tweet_text, user_name))
```

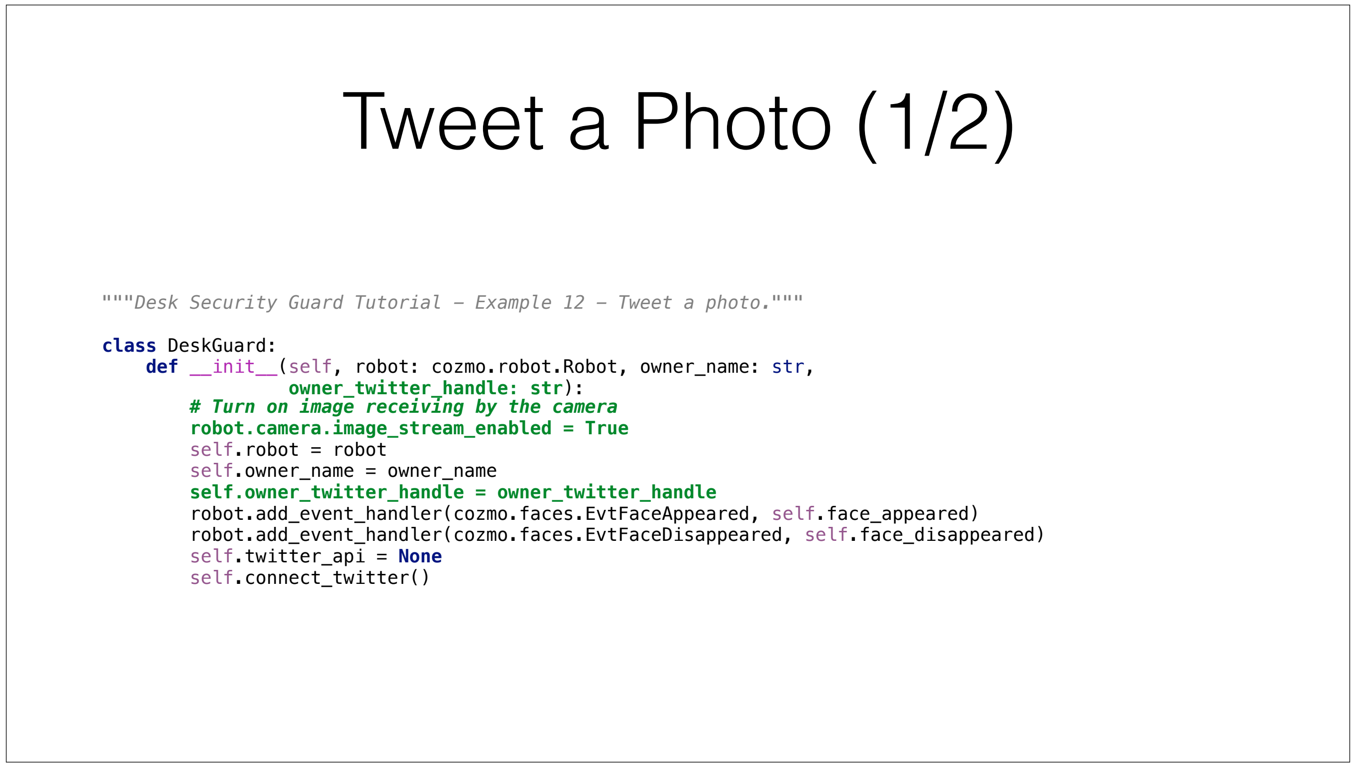
- We import sys and add an additional search path for imports - this is a bit ugly, but we want to share these two files across multiple examples in different folders.
- We add a new class for listening for tweets, sub-classed from the twitter_helpers file



Cozmo Twitter (2/2)

```
class DeskGuard:
    def __init__(self, robot: cozmo.robot.Robot, owner_name: str):
        self.robot = robot
        self.owner_name = owner_name
        robot.add_event_handler(cozmo.faces.EvtFaceAppeared, self.face_appeared)
        robot.add_event_handler(cozmo.faces.EvtFaceDisappeared, self.face_disappeared)
        self.twitter_api = None
        self.connect_twitter()

    def connect_twitter(self):
        # Connect Twitter
        self.twitter_api, twitter_auth = twitter_helpers.init_twitter(twitter_keys)
        # Create a listener for handling tweets as they appear in the stream
        stream_listener = DeskGuardStreamListener(self.twitter_api)
        twitter_stream = twitter_helpers.CozmoStream(twitter_auth, stream_listener)
        # run twitter_stream async in the background
        # Note: Tweepy does not use asyncio, so beware of threading issues
        twitter_stream.async_userstream(_with='user')
```

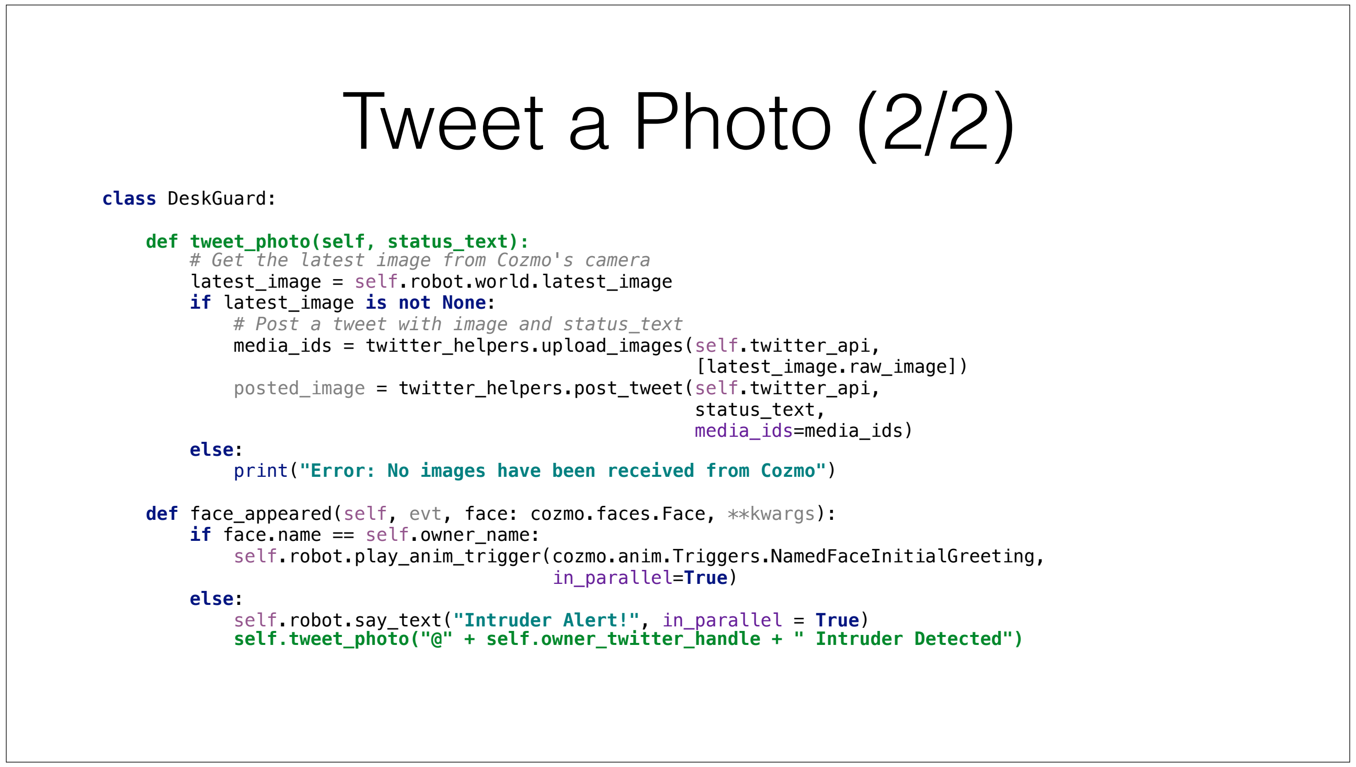
- DeskGuard now has a twitter_api member, and we connect to twitter on startup and run our stream in the background asynchronously - note that Tweepy doesn’t support asyncio so that’s actually in a different thread, so we need to be careful how we communicate between the listener and our program.



Tweet a Photo (1/2)

```python
"""Desk Security Guard Tutorial - Example 12 - Tweet a photo."""

class DeskGuard:
    def __init__(self, robot: cozmo.robot.Robot, owner_name: str,
                 owner_twitter_handle: str):
        # Turn on image receiving by the camera
        robot.camera.image_stream_enabled = True
        self.robot = robot
        self.owner_name = owner_name
        self.owner_twitter_handle = owner_twitter_handle
        robot.add_event_handler(cozmo.faces.EvtFaceAppeared, self.face_appeared)
        robot.add_event_handler(cozmo.faces.EvtFaceDisappeared, self.face_disappeared)
        self.twitter_api = None
        self.connect_twitter()
```

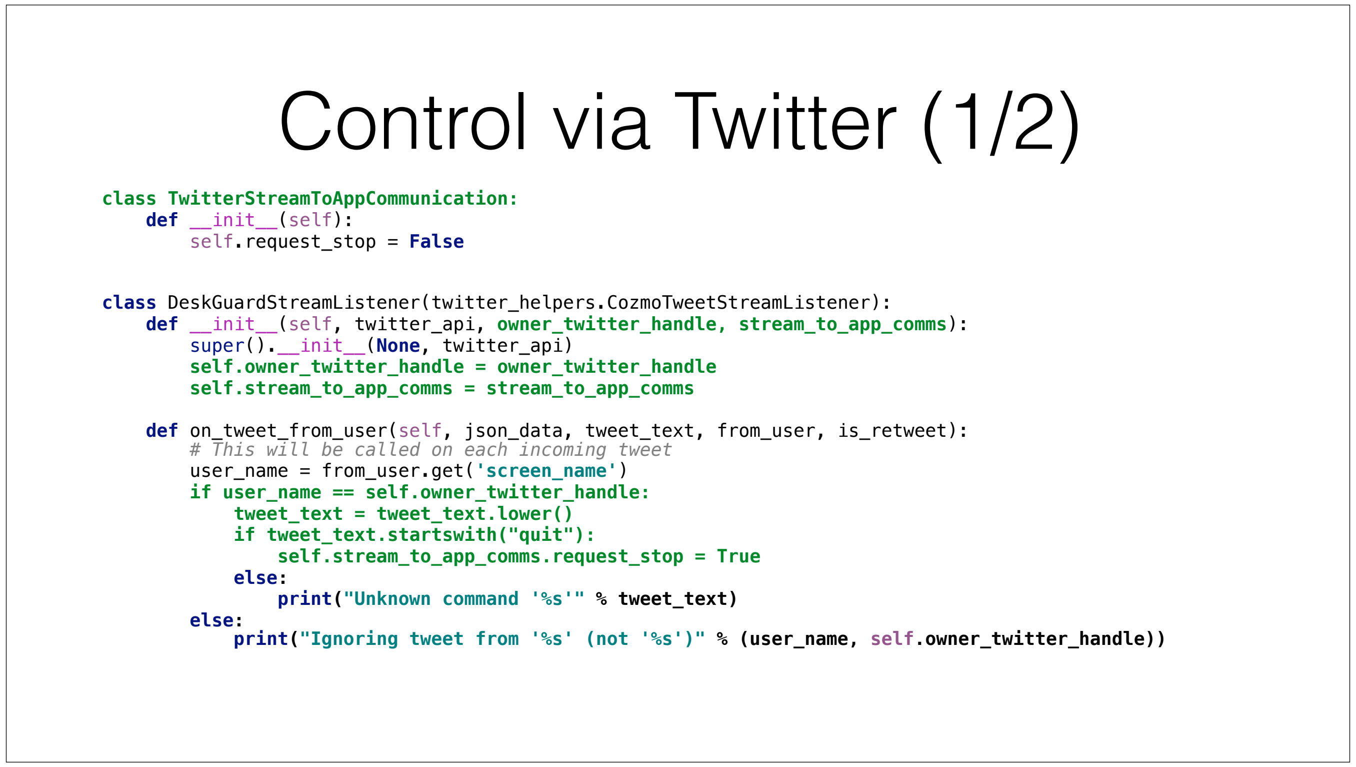
- We need to know the Twitter handle for the owner so that we can tweet at them
- We also want to ensure that we’re streaming the images from Cozmo’s camera over to the SDK



Tweet a Photo (2/2)

```
class DeskGuard:

    def tweet_photo(self, status_text):
        # Get the latest image from Cozmo's camera
        latest_image = self.robot.world.latest_image
        if latest_image is not None:
            # Post a tweet with image and status_text
            media_ids = twitter_helpers.upload_images(self.twitter_api,
                                                      [latest_image.raw_image])
            posted_image = twitter_helpers.post_tweet(self.twitter_api,
                                                      status_text,
                                                      media_ids=media_ids)
        else:
            print("Error: No images have been received from Cozmo")

    def face_appeared(self, evt, face: cozmo.faces.Face, **kwargs):
        if face.name == self.owner_name:
            self.robot.play_anim_trigger(cozmo.anim.Triggers.NamedFaceInitialGreeting,
                                         in_parallel=True)
        else:
            self.robot.say_text("Intruder Alert!", in_parallel = True)
            self.tweet_photo("@" + self.owner_twitter_handle + " Intruder Detected")
```

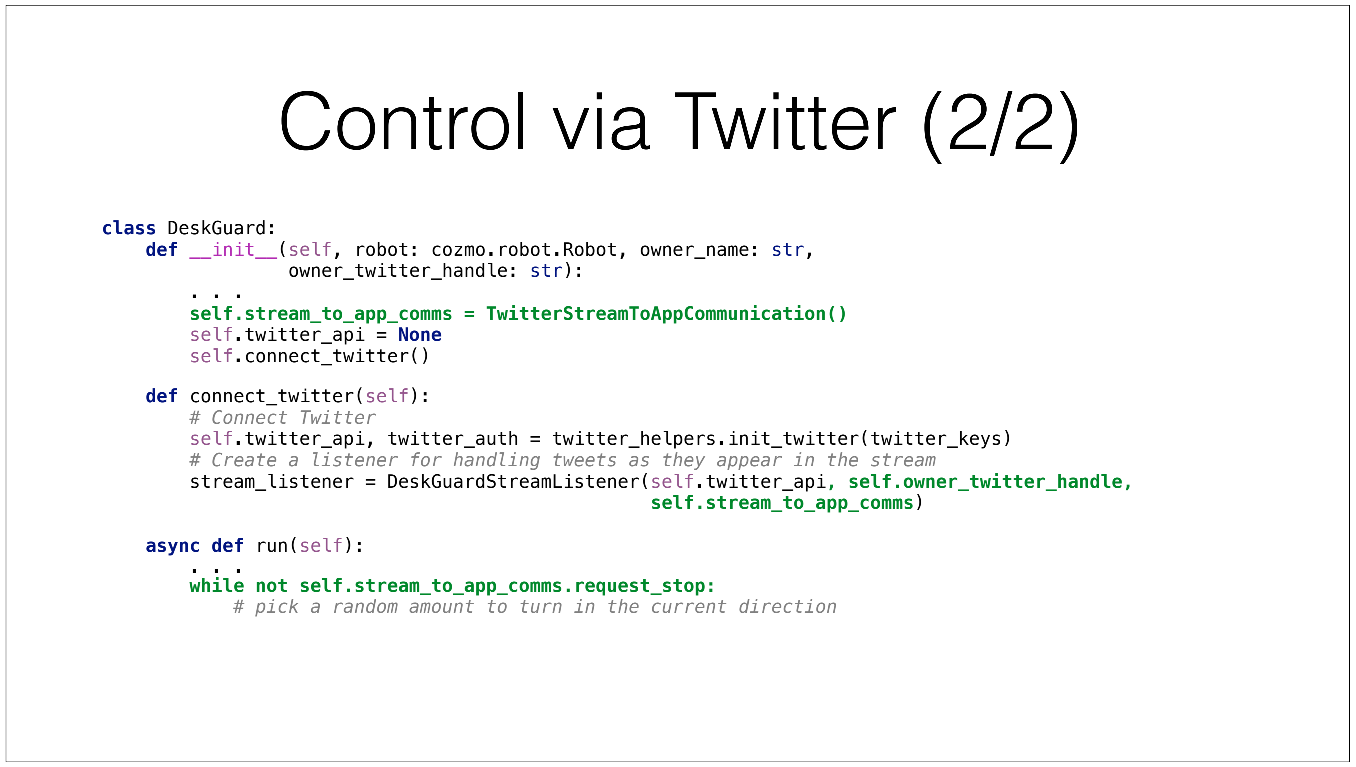
- We add a new method for tweeting Cozmo’s camera image
- And we call it when we detect an intruder, with status text to tweet at our owner



Control via Twitter (1/2)

```
class TwitterStreamToAppCommunication:
    def __init__(self):
        self.request_stop = False


class DeskGuardStreamListener(twitter_helpers.CozmoTweetStreamListener):
    def __init__(self, twitter_api, owner_twitter_handle, stream_to_app_comms):
        super().__init__(None, twitter_api)
        self.owner_twitter_handle = owner_twitter_handle
        self.stream_to_app_comms = stream_to_app_comms

    def on_tweet_from_user(self, json_data, tweet_text, from_user, is_retweet):
        # This will be called on each incoming tweet
        user_name = from_user.get('screen_name')
        if user_name == self.owner_twitter_handle:
            tweet_text = tweet_text.lower()
            if tweet_text.startswith("quit"):
                self.stream_to_app_comms.request_stop = True
            else:
                print("Unknown command '%s'" % tweet_text)
        else:
            print("Ignoring tweet from '%s' (not '%s')" % (user_name, self.owner_twitter_handle))
```

- We add a small class for passing data between the stream (on one thread) and our program, this makes it easier to enforce that we don’t access anything in an unsafe manner
- The owner’s twitter handle is used to ignore tweets from others, and we use the tweet text as a command for Cozmo
- See some of the other Cozmo twitter examples for how you could scale this to many commands instead of having a giant if-else block.



Control via Twitter (2/2)

```
class DeskGuard:
    def __init__(self, robot: cozmo.robot.Robot, owner_name: str,
                 owner_twitter_handle: str):
        . . .
        self.stream_to_app_comms = TwitterStreamToAppCommunication()
        self.twitter_api = None
        self.connect_twitter()

    def connect_twitter(self):
        # Connect Twitter
        self.twitter_api, twitter_auth = twitter_helpers.init_twitter(twitter_keys)
        # Create a listener for handling tweets as they appear in the stream
        stream_listener = DeskGuardStreamListener(self.twitter_api, self.owner_twitter_handle,
                                                  self.stream_to_app_comms)

    async def run(self):
        . . .
        while not self.stream_to_app_comms.request_stop:
            # pick a random amount to turn in the current direction
```

- We now keep the guard looking-around behavior running until we receive that stop (or you press CTRL-C to quit)

The End.
Any Questions?

https://developer.anki.com/
https://forums.anki.com/