Workshop 3: Go Go Tello!

In this workshop, you'll continue to learn more about some fundamental programming concepts and have more time to practice creating basic algorithms with Tello drones.

Important Topics

- Basic Programming Concepts Review
- Object-Oriented Programming
- Linking Everything Back Together
- Exit Survey
- More Practice with Algorithms

Object-Oriented Programming Activity

1. As a group, **create a class that describes anything of your choosing. Make sure that your class has at least one attribute and at least one method.** Create two or more instances of it, and then run some methods.
    - Solve the problem as a group, but try to run it on your own individual devices.

Exit Survey

Hey, your satisfaction and success means a lot to us! Could you let us know how we're doing?

https://forms.office.com/r/HsERcHXpj2

More Practice with Algorithms Activity

> algorithm (noun), a step-by-step procedure for solving a problem or accomplishing some end.
>
> - Merriam-Webster

As a group, create the following basic algorithms:

1. `fetch(totalDistance)`
    - Go `totalDistance/2` units forward, then turn 180 degrees and go `totalDistance/2` units back to the original location.
2. `square(totalDistance)`
    - Go `totalDistance/4` units forward, turn 90 degrees either clockwise or counterclockwise (you choose), go `totalDistance/4` units forward, etc. until finally arriving back at the original location, effectively having traveled in a square.

If you would like to try the prompts from last week again, they are listed here:

1. `stepForwards(totalDistance, steps)`
    - Go a total of `totalDistance` units of distance in the forward direction, taking a total of `steps` many steps, waiting for 1 second in-between steps.
    - e.g. `stepForwards(6, 3)` would go forward 2 units, wait a second, go forward 2 units, wait a second, and then go forward 2 units.
2. `stepBackwards(totalDistance, steps)`

- The same as `stepForwards`, but in the backwards direction.

3. `hookLeft(totalDistance)`
   - Go `totalDistance/2` units forward, rotate a full 90 degrees such that your drone turns left, and then go `totalDistance/2` units forward, essentially doing a left hook.
4. `hookRight(totalDistance)`
   - The same as `hookLeft`, but going right instead.

Challenge Activity

5. `arc(totalDistance, turnRadius)`
   - Travel in an arc of `totalDistance`, arcing a certain `turnRadius`.

You may find it useful to refer to the Python and Tello reference sheets.

Python Reference Sheet

```
# Comments look like this. They start with a '#' sign.

# Conditionals look like this.
if condition:
    instructions

# Here are a few examples.

if x < 5:
    print("x is less than 5.")

if x < 5 and x % 2 == 0:
    print("x is less than 5, and x is also an even number.")

if passwordEntered == actualPassword:
    print("You entered the right password!")
    print("Hello user.")

# There are 3 kinds of loops:
# - while loops do something while a condition is true.
# - for x in y loops do something for every item x in a collection of items y.
# - for x in range(N) will do something N many times.

# Here are a few examples.

letter = input("Enter a letter: ")
while letter != 'a':
    print("You didn't enter letter the letter 'a'")
    letter = input("Enter a letter: ")

people = ["John", "Jane", "Jack", "Jill"]
for person in people:
    print("Hello", person)

for i in range(10):
    print("Hi!")

# Functions look like this.
def greet():
    print("Hello!")
```

```python
# That's just the definition. It doesn't actually run the function. In order to
# do that, you need to call it.
greet()

# Functions that take inputs and return values look like this.
def f(x, y, z):
    n = x + y + z
    n *= 5
    return n

# When you have functions that take inputs, you need to pass them in when
# calling them. When you have functions that (optionally) return values, the
# function can directly represent data.
output = f(2, 6, 7)

# Classes are "blueprints" for items.
class Person:

    # These are attributes, or variables inside of classes.
    name = None
    age = None
    pronouns = None

    # This is a special method called a constructor. It is run when an object
    # is first instantiated (made an instance of). It's always called __init__.
    def __init__(self, inputName, inputAge, inputPronouns):
        self.name = inputName
        self.age = inputAge
        self.pronouns = inputPronouns

    # This is a method we defined.
    def grow_up(self):
        self.age += 1

# We instantiate objects like so. If there is a constructor, then we pass
# in arguments to whatever parameters the constructor has.
Shawn = Person("Shawn Kathleen", 20, "he/him")
Caoimhe = Person("Caoimhe Tran", 17, "they/them")

# We can access attributes using a "dot" notation. We access methods the
# same way as well.
print(Shawn.age)     # This will print out 20.
print(Caoimhe.age)   # This will print out 17.
Shawn.grow_up()
print(Shawn.age)     # This will print out 21.
print(Caoimhe.age)   # This will print out 17.
```

Tello Reference Sheet

Remember to install the libraries first before you proceed! You'll need:

- The Solar Energy Association's fork of the easyTello library
    - https://github.com/shawnduong/2021-Fall-SEA-x-ENGRSL-Python-Workshop-Series/blob/main/tello.py
- OpenCV2 (`opencv-python`)

Basics:

```
# Remember to import the fixed library that we need.
# tello.py must be in the same directory (folder) as your main.py.
import tello

# This is how you make drone objects.
# We'll talk more about objects in the future.
drone = tello.Tello()

# This is how you turn the camera on and view the feed.
drone.streamon()

# This is how you turn the camera off and stop viewing the feed.
drone.streamoff()
```

Important note: your program will try to finish as fast as possible. If you're just turning the camera on and off, then this means that it'll turn the camera on, *but then it immediately turns it off.* You might find this code useful:

```
# Import the time library.
import time

# Sleep for 5 minutes. 5 minutes * 60 seconds/minute = 300 seconds.
time.sleep(300)
```

Movement:

```
# Get the drone off the ground.
drone.takeoff()

# Land the drone.
drone.land()

# In the following methods, n is an integer from [20, 500].

# Go forward n cm.
drone.forward(n)

# Go back n cm.
drone.back(n)

# Strafe left n cm.
drone.left(n)

# Strafe right n cm.
drone.right(n)

# Go up n cm.
drone.up(n)

# Go down n cm.
drone.down(n)

# Rotate x degrees clockwise.
drone.cw(x)
```

```
# Rotate x degrees counter-clockwise.
drone.ccw(x)

# This one is a bit more advanced!
# Fly at a curve from a defined (x1, y1, z1) to a defined (x2, y2, z2) at
# s cm/s. The arc radius must be [0.5, 10] m. All coordinates are [-500, 500],
# and s is [10, 60].
drone.curve(x1, y1, z1, x2, y2, z2)
```