Day 05

Object-Oriented Programming and Classes

Classes

What is a class? A class is a blueprint for creating objects (a particular data structure), providing initial values for state (member variables or attributes), and implementations of behavior (member functions or methods). So class is the general scheme for the object, and the object is the specific instance of the class.

Why use classes? Classes allow us to create objects that have their own properties and methods. This allows us to create objects that are more complex than simple data structures.

How to create a class In python, we create a class by using the keyword `class` followed by the name of the class. The class definition is followed by a colon and the body of the class is indented. Let's take a look at an example from the 2024 robot code:

```python
import math

import ntcore

import config
import constants
from toolkit.motors.ctre_motors import TalonFX
from toolkit.subsystem import Subsystem
from units.SI import meters_per_second, radians_per_second

foc_active = False


class Flywheel(Subsystem):
    def __init__(self):
        super().__init__()

        self.motor_1: TalonFX = TalonFX(
            can_id=config.flywheel_id_1,
            foc=foc_active,
            inverted=True,
            config=config.FLYWHEEL_CONFIG,
        )
        self.motor_2: TalonFX = TalonFX(
            can_id=config.flywheel_id_2,
            foc=foc_active,
            inverted=True,
            config=config.FLYWHEEL_CONFIG,
        )

        self.flywheel_top_target = 0
        self.flywheel_bottom_target = 0

        self.ready_to_shoot: bool = False
        self.initialized: bool = False


    def init(self) -> None:
        self.motor_1.init()
        self.motor_2.init()
```

```
def note_shot(self) -> bool:
    return (
        self.get_current(1) > config.flywheel_shot_current_threshold
        or self.get_current(2) > config.flywheel_shot_current_threshold
    )

def set_velocity(self, angular_velocity: radians_per_second, motor=0) -> None:
    if motor == 1:

        self.flywheel_top_target = angular_velocity
        self.motor_1.set_target_velocity(
            self.angular_velocity_to_rps(angular_velocity), 0
        )
    elif motor == 2:

        self.flywheel_bottom_target = angular_velocity
        self.motor_2.set_target_velocity(
            self.angular_velocity_to_rps(angular_velocity), 0
        )
    else:

        self.flywheel_top_target = angular_velocity
        self.flywheel_bottom_target = angular_velocity
        self.motor_1.set_target_velocity(
            self.angular_velocity_to_rps(angular_velocity), 0
        )

        self.motor_2.set_target_velocity(
            self.angular_velocity_to_rps(angular_velocity), 0
        )

def set_velocity_linear(self, linear_velocity: meters_per_second, motor=0) -> None:
    angular_velocity = linear_velocity / constants.flywheel_radius_outer
    self.set_velocity(angular_velocity, motor)

def get_velocity(self, motor=0) -> radians_per_second:
    if motor == 1:
        return self.rps_to_angular_velocity(self.motor_1.get_sensor_velocity())
    elif motor == 2:
        return self.rps_to_angular_velocity(self.motor_2.get_sensor_velocity())
    else:
        return (
            self.rps_to_angular_velocity(self.motor_1.get_sensor_velocity()),
            self.rps_to_angular_velocity(self.motor_2.get_sensor_velocity()),
        )

def get_velocity_linear(self, motor=0) -> meters_per_second:
    if motor == 0:
        return (
            (self.get_velocity(1) + self.get_velocity(2))
            / 2
            * constants.flywheel_radius_outer
        )
    else:
        return self.get_velocity(motor) * constants.flywheel_radius_outer
```

```
def get_current(self, motor=0) -> float:
    if motor == 1:
        return self.motor_1.get_motor_current()
    elif motor == 2:
        return self.motor_2.get_motor_current()
    else:
        return (
            self.motor_1.get_motor_current(),
            self.motor_2.get_motor_current(),
        )

def within_velocity(
    self, velocity: radians_per_second, tolerance: radians_per_second, motor=0
) -> bool:
    """
    Returns True if the flywheel velocity is within the tolerance of the target velocity
    """

    def tol(vel, target, tol):
        return abs(vel - target) < tol

    if motor == 1:
        return tol(self.get_velocity(1), velocity, tolerance)
    elif motor == 2:
        return tol(self.get_velocity(2), velocity, tolerance)
    else:
        return tol(self.get_velocity(1), velocity, tolerance) and tol(
            self.get_velocity(2), velocity, tolerance
        )

def within_velocity_linear(
    self, velocity: meters_per_second, tolerance: meters_per_second, motor=0
) -> bool:
    """
    Returns True if the flywheel velocity is within the tolerance of the target velocity
    """

    def tol(vel, target, tol):
        return abs(vel - target) < tol

    if motor == 1:
        return tol(self.get_velocity_linear(1), velocity, tolerance)
    elif motor == 2:
        return tol(self.get_velocity_linear(2), velocity, tolerance)
    else:
        return tol(self.get_velocity_linear(1), velocity, tolerance) and tol(
            self.get_velocity_linear(2), velocity, tolerance
        )

def periodic(self):

    if self.within_velocity_linear(
        self.angular_velocity_to_linear_velocity(self.flywheel_top_target),
        config.flywheel_shot_tolerance,
    ):
```

```
            self.ready_to_shoot = True
        else:
            self.ready_to_shoot = False

        table = ntcore.NetworkTableInstance.getDefault().getTable("flywheel")
        table.putNumber("flywheel top velocity", self.get_velocity_linear(1))
        table.putNumber("flywheel bottom velocity", self.get_velocity_linear(2))
        table.putBoolean("ready to shoot", self.ready_to_shoot)
        table.putBoolean("note shot", self.note_shot())
        table.putNumber("flywheel top velocity rpm", self.motor_1.get_sensor_velocity())
        table.putNumber(
            "flywheel top target",
            self.angular_velocity_to_linear_velocity(self.flywheel_top_target),
        )
        table.putNumber(
            "flywheel bottom target",
            self.angular_velocity_to_linear_velocity(self.flywheel_bottom_target),
        )
)
```

Some things about this code that I want you to notice:

- The class is defined with the keyword `class` followed by the name of the class.
- The class definition is followed by a colon and the body of the class is indented.
- The class has a method called `__init__` which is a special method that is called when an object is created from the class.
- The class has other methods that define the behavior of the class.
- The class has properties that define the state of the class.
- Next to the name of the class is a set of parentheses that say what class this class inherits from. In this case, the class inherits from `Subsystem` which is a class that we made to help organize the robot code.

How to create an object from a class To create an object from a class, we use the class name followed by a set of parentheses. Let's take a look at an example:

```
flywheel = Flywheel()
```

If the robot had multiple flywheel we could make multiple objects from the class like this:

```
flywheel_1 = Flywheel()
flywheel_2 = Flywheel()
```

How to access properties and methods of an object To access the properties and methods of an object, we use the dot operator. Let's take a look at an example:

```
flywheel.set_velocity(100)
```

In this example, we are calling the `set_velocity` method of the `flywheel` object and passing in the value `100`.

```
speed = flywheel.get_velocity()
```

In this example, we are calling the `get_velocity` method of the `flywheel` object and storing the return value in the variable `speed`.

```
if flywheel.ready_to_shoot:
    print("Ready to shoot!")
```

In this example, we are accessing the `ready_to_shoot` property of the `flywheel` object and checking if it is `True`.

Summary In this lesson, we learned about classes in python. We learned what a class is, why we use classes, how to create a class, how to create an object from a class, and how to access properties and methods of an object. Classes are a powerful tool in python that allow us to create objects that have their own properties and methods. We will be using classes extensively in our robot code to organize and structure our code.

Practice Let's make sure that all this makes sense.

1. What is a class?
2. Why use classes?
3. How do you create a class in python?
4. How do you create an object from a class?
5. How do you access properties and methods of an object?
6. What is the `__init__` method and what does it do?

Practical Practice Start the assignment at https://classroom.github.com/a/utnqGH_3

Notes on Data Structures in Python Python has the following data structures:

- Lists - which are made using square brackets `[]`
- Tuples - which are made using parentheses `()`. Tuples are immutable, meaning that once they are created, they cannot be changed.
- Sets - which are made using curly braces `{}`. Sets are unordered collections of unique elements.
- Dictionaries - which are made using curly braces `{}`. Dictionaries are collections of key-value pairs.
- Strings - which are made using single or double quotes `''` or `""`.
- Numbers - which can be integers or floats.
- Booleans - which can be `True` or `False`.
- None - which is a special type that represents the absence of a value.
- Objects - which are instances of classes.
- Functions - which are objects that can be called.