Parameters and Launch Files in ROS2

AND AN INTRO TO TF2

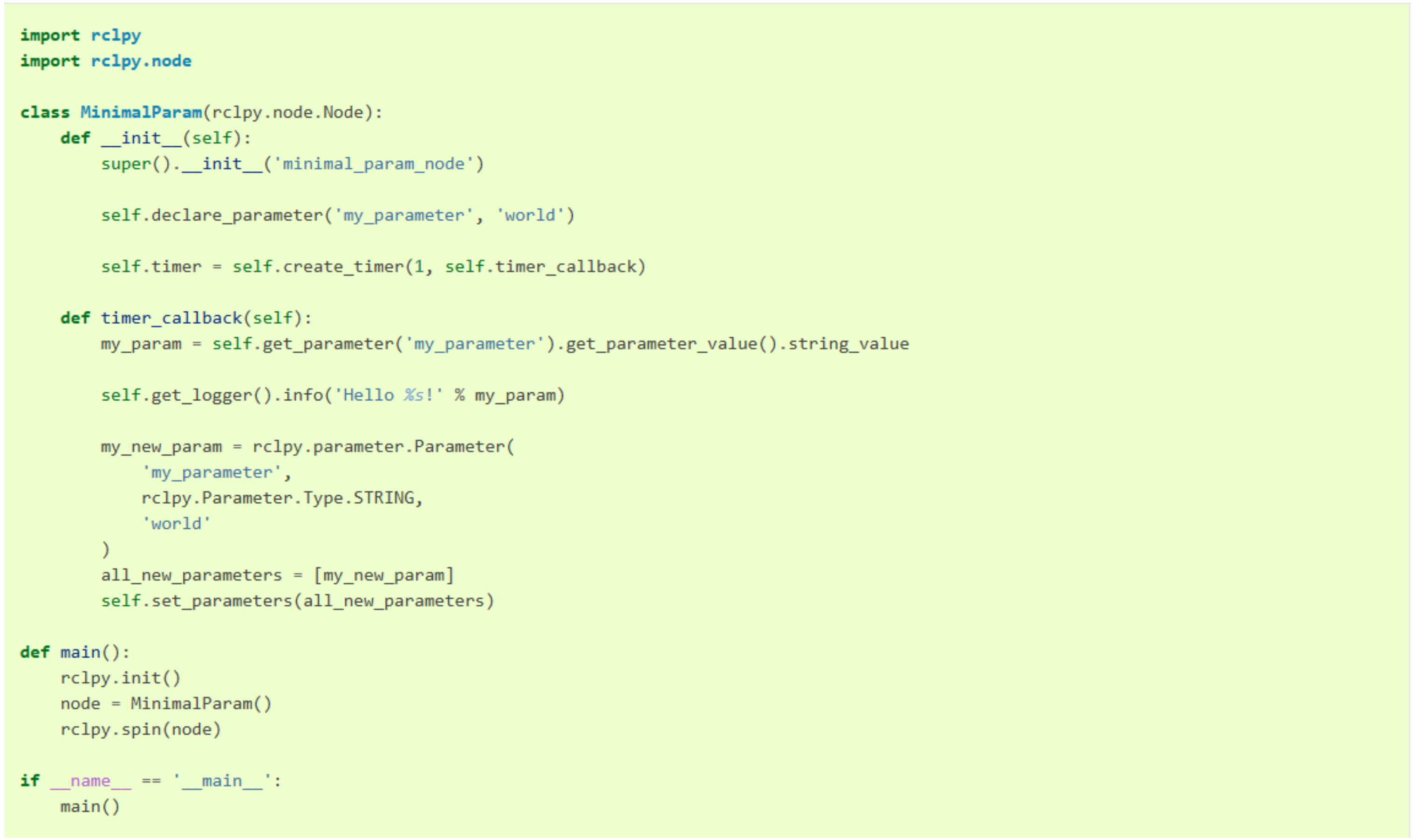
What are Parameters?

In **ROS 2** (Robot Operating System 2), **parameters** are used to configure nodes at runtime, allowing you to modify behavior without changing code. Parameters in ROS 2 are **strongly typed**, and they can be declared and set either programmatically (in code), via launch files, or through the command line.

Why we need Parameters?

The **need for parameters in ROS 2** arises from the requirement to make robot software **modular, configurable, and reusable** without needing to recompile or rewrite code. Parameters allow for **dynamic configuration of nodes**, helping developers tune behavior, adapt to different environments, and manage deployment easily.

Using Parameters



```python
import rclpy
import rclpy.node

class MinimalParam(rclpy.node.Node):
    def __init__(self):
        super().__init__('minimal_param_node')

        self.declare_parameter('my_parameter', 'world')

        self.timer = self.create_timer(1, self.timer_callback)

    def timer_callback(self):
        my_param = self.get_parameter('my_parameter').get_parameter_value().string_value

        self.get_logger().info('Hello %s!' % my_param)

        my_new_param = rclpy.parameter.Parameter(
            'my_parameter',
            rclpy.Parameter.Type.STRING,
            'world'
        )
        all_new_parameters = [my_new_param]
        self.set_parameters(all_new_parameters)

def main():
    rclpy.init()
    node = MinimalParam()
    rclpy.spin(node)

if __name__ == '__main__':
    main()
```

- Add an entry point to the node.
- Build the package
- Run the node
- Experiment with different values of parameters.

Syntax for running and setting parameters

```
ros2 run python_parameters minimal_param_node
ros2 param list
ros2 param set /minimal_param_node my_parameter earth
```

For more details, explore the [ROS2 Parameters Documentation].

What are Launch Files?

Launch files in ROS 2 are used to **start and configure multiple nodes and their environments** in a single, organized way. They are essential for complex robotics systems where many nodes need to be started together with specific parameters, remappings, and namespace configurations.

Why we need Launch Files?

Start multiple nodes, set parameters, remap topics, define namespaces—all in one file. Easily reuse and combine launch files for different parts of a robot (e.g., sensors, planners, navigation stack). Support conditions, substitutions, arguments, and logic (e.g., launch nodes only if a condition is met). Launch files can take arguments, allowing the same launch file to work across different environments or robots. Great for bringing up a simulated environment (Gazebo, RViz, fake sensors) for testing without hardware.

What are TFs?

In ROS 2, TF (short for transform) is a crucial subsystem that allows you to keep track of coordinate frames over time. It is used to transform data (e.g., positions, sensor readings) between different coordinate frames like base_link, camera_link, odom, map, etc. ROS 2 uses tf2, the updated version of the original TF library from ROS 1.

Why we need TFs?

Track the position and orientation of different parts of the robot and the environment. Convert data (e.g., laser scan, image, point cloud) from one frame to another accurately. Transform data across time—critical when dealing with asynchronous sensors. TF allows different parts of the robot (sensors, actuators, planning, etc.) to work in their own frames and still understand each other. Used in SLAM, navigation, and sensor fusion to align sensor data with the robot's current pose.