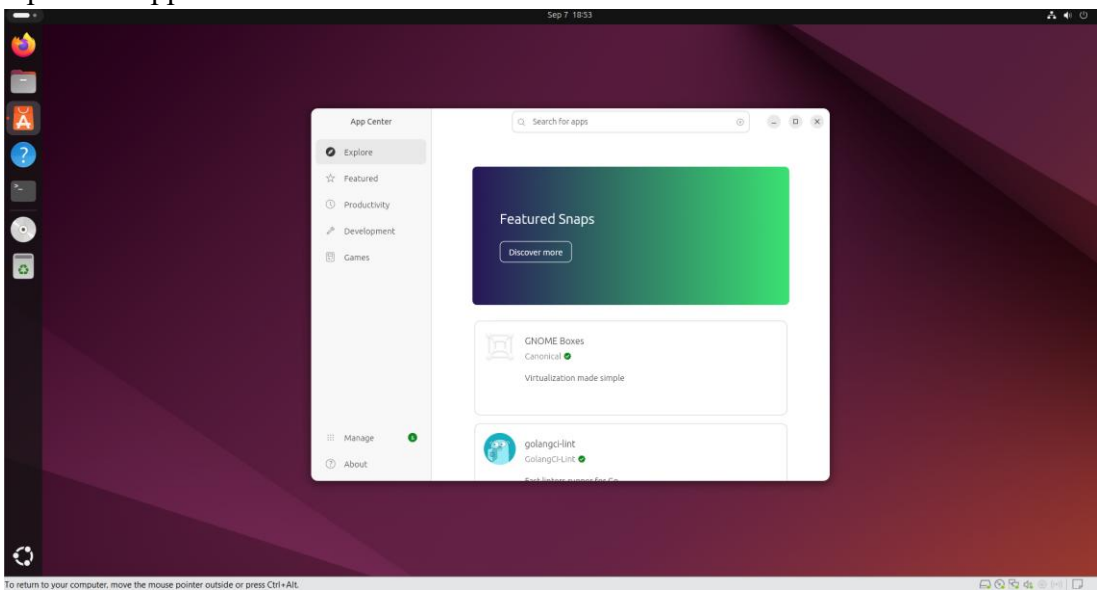
Tutorial 2 - Basic Robotics Programming with ROS

Objective

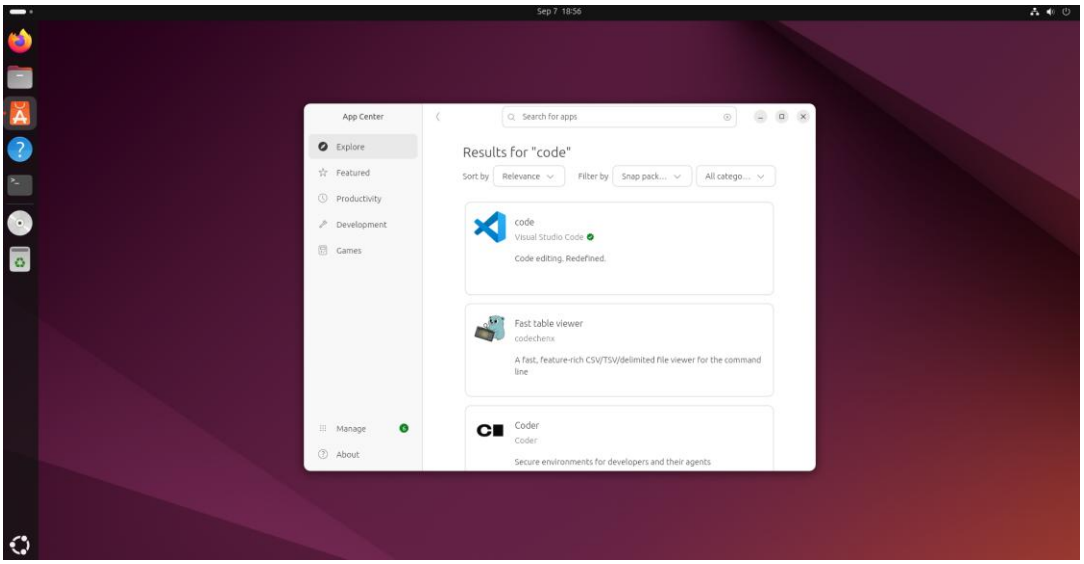
- Introduce the basic robotics programming using ROS 2 Jazzy.
- Cover setting up a ROS 2 workspace with Colcon, creating and managing packages, understanding nodes and topics through the turtlesim demo, and developing simple Python-based publisher and subscriber nodes.
- Learn how to create a ROS 2 package to control a turtle in the turtlesim environment, including modifying the publisher to make the turtle follow complex paths, such as drawing a star.

1 Install the Visual Studio Code

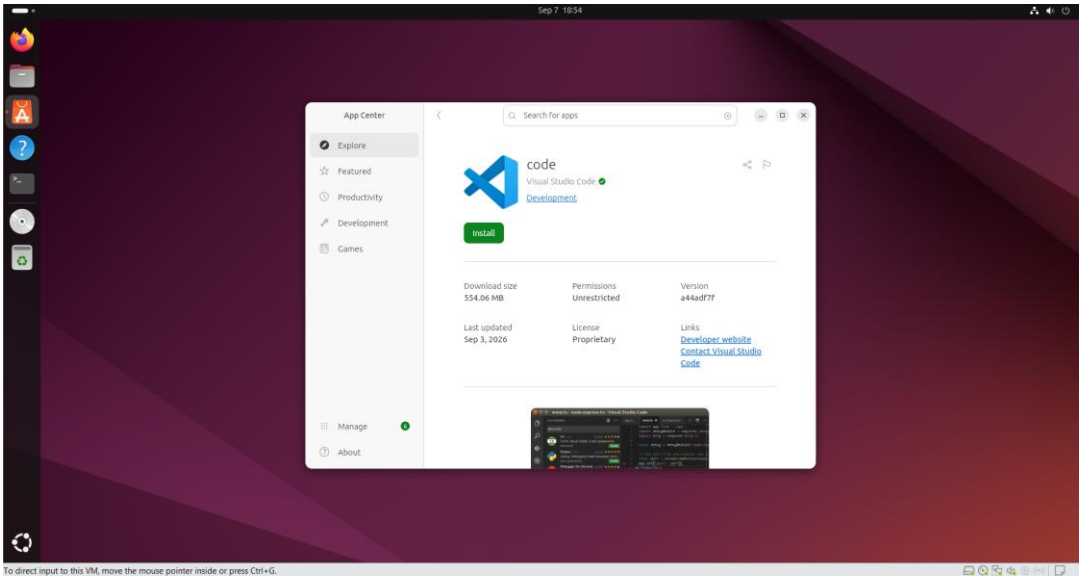
1. Open the virtual machine in vmware workstation.
2. Open the App Centre.



3. Search “Code” in App Centre.



4. Install the visual studio code.



2 Colcon Setup

1. Install Colcon
   Open a new terminal and run

```
$ sudo apt install python3-colcon-common-extensions
```

2. Setup colcon_cd

```
$ echo "source /usr/share/colcon_cd/function/colcon_cd.sh" >> ~/.bashrc

$ echo "export _colcon_cd_root=/opt/ros/jazzy/" >> ~/.bashrc
```

3. Setup colcon mixins

```
$ colcon mixin add default https://raw.githubusercontent.com/colcon/colcon-mixin-repository/master/index.yaml

$ colcon mixin update default
```

3 Create and Build a workspace

ROS 2 is a directory structure where you develop, store, build, and manage your ROS packages.

1. Make a directory for the workspace

```
$ mkdir -p ~/ros2_ws/src
```

2. Clone a sample repo in the "src" directory

```
$ cd ~/ros2_ws/src

$ git clone https://github.com/ros/ros_tutorials.git -b jazzy
```

3. Resolve dependencies

```
$ cd ..

$ sudo rosdep init

$ rosdep update

$ rosdep install -i --from-path src --rosdistro jazzy -y
```

4. Build the workspace

```
$ cd ~/ros2_ws

$ colcon build
```

5. Source the environment

```
$ echo "source ~/ros2_ws/install/setup.bash" >> ~/.bashrc
```

6. Demo of turtlesim

   Open a new terminal and run

```
$ cd ros2_ws

$ ros2 run turtlesim turtlesim_node
```

   Open a new terminal

```
$ ros2 run turtlesim turtle_teleop_key
```

4 Create a Package

1. Head to the src directory and create a new package

```
$ cd ~/ros2_ws/src

$ ros2 pkg create --build-type ament_python --license Apache-2.0 --node-name
my_node my_package
```

2. Return to the root of the workspace and build the package

```
$ cd ~/ros2_ws

$ colcon build --packages-select my_package
```

5 Basic ROS2 Commands for topics and nodes

In ROS 2, nodes are the fundamental building blocks of your robot's software, and they communicate by sending data over topics.

Before writing your own code, it is essential to understand how to inspect and interact with nodes and topics using the ROS 2 command-line interface.

To show the commands, we are going to use turtlesim as an example.

1. Run a turtlesim_node

```
$ ros2 run turtlesim turtlesim_node
```

2. In another terminal, run "turtle_teleop_key"

```
$ ros2 run turtlesim turtle_teleop_key
```

3. In another terminal, run the "ros2 node list" command will show you the names of all running nodes. It should output:

```
/turtlesim

/teleop_turtle
```

4. Run the command "ros2 node info <node_name> " will give you information about the node. Try running a command "ros2 node info /turtlesim":

```
/turtlesim
```

```
  Subscribers:

    /parameter_events: rcl_interfaces/msg/ParameterEvent

    /turtle1/cmd_vel: geometry_msgs/msg/Twist

  Publishers:

    /parameter_events: rcl_interfaces/msg/ParameterEvent

    /rosout: rcl_interfaces/msg/Log

    /turtle1/color_sensor: turtlesim/msg/Color

    /turtle1/pose: turtlesim/msg/Pose

  Service Servers:

    /clear: std_srvs/srv/Empty

    /kill: turtlesim/srv/Kill

    /reset: std_srvs/srv/Empty

    /spawn: turtlesim/srv/Spawn

    /turtle1/set_pen: turtlesim/srv/SetPen

    /turtle1/teleport_absolute: turtlesim/srv/TeleportAbsolute

    /turtle1/teleport_relative: turtlesim/srv/TeleportRelative

    /turtlesim/describe_parameters: rcl_interfaces/srv/DescribeParameters

    /turtlesim/get_parameter_types: rcl_interfaces/srv/GetParameterTypes

    /turtlesim/get_parameters: rcl_interfaces/srv/GetParameters

    /turtlesim/get_type_description:
type_description_interfaces/srv/GetTypeDescription

    /turtlesim/list_parameters: rcl_interfaces/srv/ListParameters

    /turtlesim/set_parameters: rcl_interfaces/srv/SetParameters

    /turtlesim/set_parameters_atomically:
rcl_interfaces/srv/SetParametersAtomically

  Service Clients:


  Action Servers:

    /turtle1/rotate_absolute: turtlesim/action/RotateAbsolute

  Action Clients:
```

5. The command “ros2 topic list” will return a list of all the topics currently active in the system and ros2 topic list -t will return the same thing along with the topic type which can be useful. It should output:

```
/parameter_events [rcl_interfaces/msg/ParameterEvent]

/rosout [rcl_interfaces/msg/Log]

/turtle1/cmd_vel [geometry_msgs/msg/Twist]

/turtle1/color_sensor [turtlesim/msg/Color]

/turtle1/pose [turtlesim/msg/Pose]
```

6. The command “ros2 topic echo <topic_name>” lets you see the data being published on a topic. Try running the following command:

```
$ ros2 topic echo /turtle1/cmd_vel
```

7. The command “ros2 topic info <topic_name>“ will give you information about the topic. Try running the following command:

```
$ ros2 topic info /turtle1/cmd_vel
```

8. Nodes send data over topics using specific messages of the same type to communicate. Recall in “ros2 topic list -t” that the topic “/turtle1/cmd_vel” has type “geometry_msgs/msg/Twist”. This means the package “geometry_msgs” has a “msg” called Twist.
   “ros2 interface show <msg_type>” will return the details on the structure of the data expected by the nodes. Try running the following command:

```
$ ros2 interface show geometry_msgs/msg/Twist
```

   which will show the structure of the data being published seen in the “ros2 topic echo /turtle1/cmd_vel” command

```
Vector3  linear

            float64 x

            float64 y

            float64 z

Vector3  angular

            float64 x

            float64 y
```

```
            float64 z
```

9. Now that you have the message structure, you can publish data to a topic directly using the command ros2 topic pub <topic_name> <msg_type> '<args>'
   In order to publish data to a topic, you need to pass the data in the form of a YAML string. You can try to make the turtle go in a circle with the command:

```
$ ros2 topic pub /turtle1/cmd_vel geometry_msgs/msg/Twist "{linear: {x: 2.0, y: 0.0, z: 0.0}, angular: {x: 0.0, y: 0.0, z: 1.8}}"
```

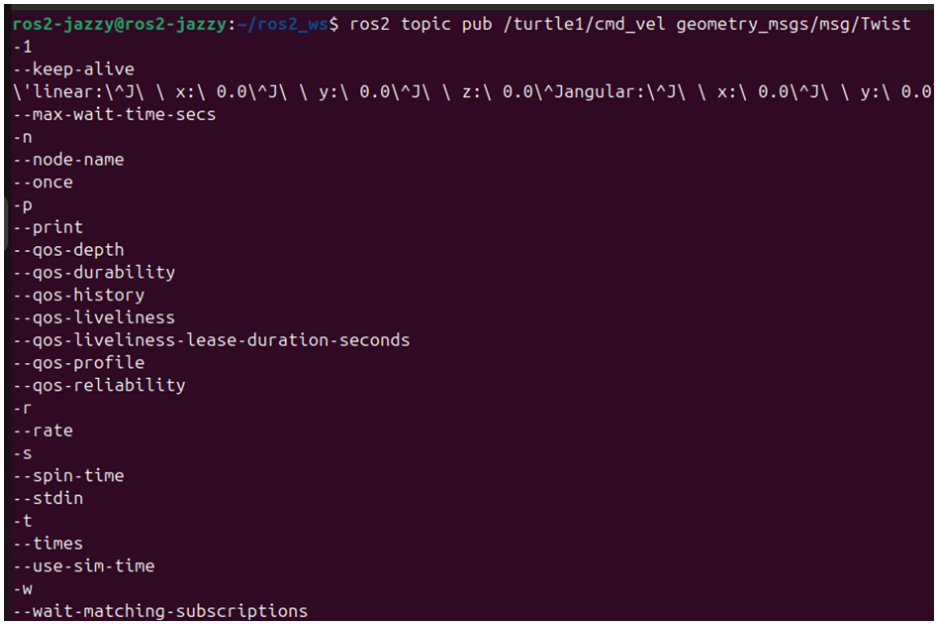
   You can also change only specific parts of that data published

```
ros2 topic pub /turtle1/cmd_vel geometry_msgs/msg/Twist <Press TAB key>
```

   After hitting <TAB> once or twice, the thing to look out for is:

```
\'linear:\^J\ \ x:\ 0.0\^J\ \ y:\ 0.0\^J\ \ z:\ 0.0\^Jangular:\^J\ \ x:\ 0.0\^J\ \ y:\ 0.0\^J\ \ z:\ 0.0\^J\'
```

   If you write \'*<Press TAB key>* then it should autocomplete the structure of the message for you.





6 Create a Simple Publisher and Subscriber

1. Head to src directory and create a new package

```
$ cd ~/ros2_ws/src

$ ros2 pkg create --build-type ament_python --license Apache-2.0 py_pubsub
```

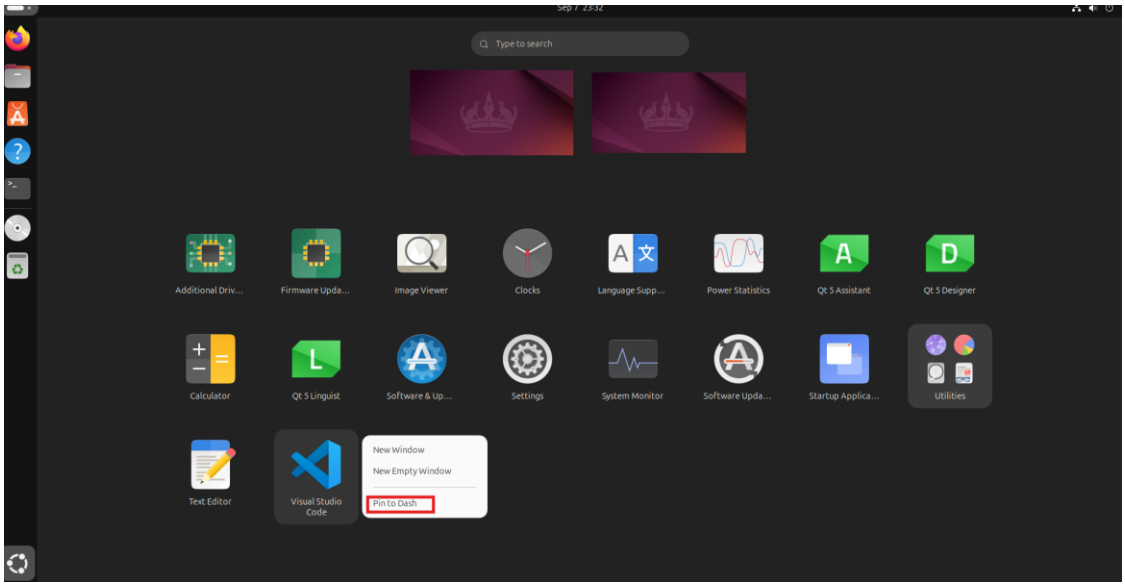
2. Navigate into **ros2_ws/src/py_pubsub/py_pubsub** and download the example publisher code

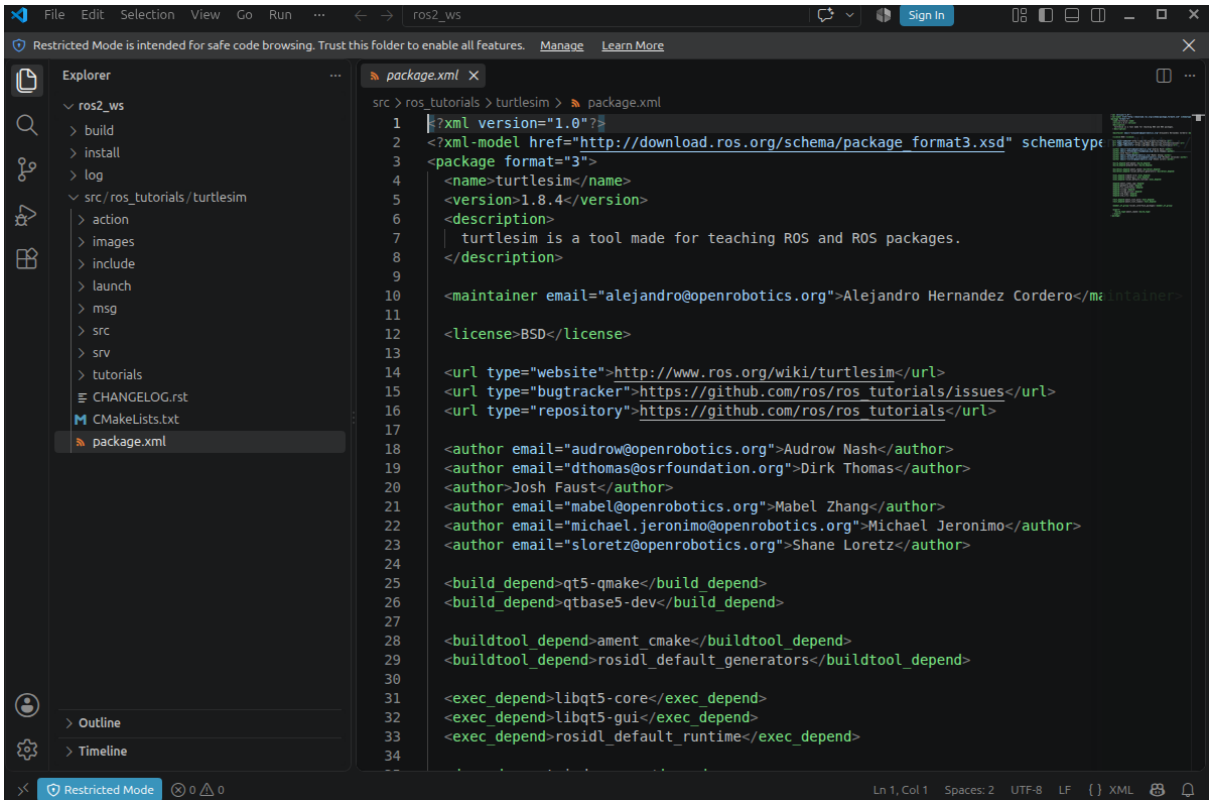
```
$ cd ~/ros2_ws/src/py_pubsub/py_pubsub

$ wget
https://raw.githubusercontent.com/ros2/examples/jazzy/rclpy/topics/minimal_publisher/examples_rclpy_minimal_publisher/publisher_member_function.py
```

3. Customize package.xml and add dependencies

   Navigate one level back to the **ros2_ws/src/py_pubsub** directory. Open your workspace in VS code by "File-Open Folder" and select the "ros2_ws" folder. Find the package.xml.






```
<?xml version="1.0"?>
<?xml-model href="http://download.ros.org/schema/package_format3.xsd" schematype
<package format="3">
  <name>turtlesim</name>
  <version>1.8.4</version>
  <description>
    turtlesim is a tool made for teaching ROS and ROS packages.
  </description>

  <maintainer email="alejandro@openrobotics.org">Alejandro Hernandez Cordero</ma

  <license>BSD</license>

  <url type="website">http://www.ros.org/wiki/turtlesim</url>
  <url type="bugtracker">https://github.com/ros/ros_tutorials/issues</url>
  <url type="repository">https://github.com/ros/ros_tutorials</url>

  <author email="audrow@openrobotics.org">Audrow Nash</author>
  <author email="dthomas@osrfoundation.org">Dirk Thomas</author>
  <author>Josh Faust</author>
  <author email="mabel@openrobotics.org">Mabel Zhang</author>
  <author email="michael.jeronimo@openrobotics.org">Michael Jeronimo</author>
  <author email="sloretz@openrobotics.org">Shane Loretz</author>

  <build_depend>qt5-qmake</build_depend>
  <build_depend>qtbase5-dev</build_depend>

  <buildtool_depend>ament_cmake</buildtool_depend>
  <buildtool_depend>rosidl_default_generators</buildtool_depend>

  <exec_depend>libqt5-core</exec_depend>
  <exec_depend>libqt5-gui</exec_depend>
  <exec_depend>rosidl_default_runtime</exec_depend>
```

After the <license> tag and inside the root <package> element, add the following lines for dependencies

```
<exec_depend>rclpy</exec_depend>

<exec_depend>std_msgs</exec_depend>
```



```
<license>Apache-2.0</license>

<exec_depend>rclpy</exec_depend>
<exec_depend>std_msgs</exec_depend>

<test_depend>ament_copyright</test_depend>
<test_depend>ament_flake8</test_depend>
<test_depend>ament_pep257</test_depend>
```

4. Customize setup.py and add entry point

   Open the setup.py file in VS code.

   Add the following line within the console_scripts brackets of the entry_points field:

```
entry_points={
        'console_scripts': [
                'talker = py_pubsub.publisher_member_function:main',
                'listener = py_pubsub.subscriber_member_function:main',
        ],
},
```

```
    install_requires=['setuptools'],
    zip_safe=True,
    maintainer='bob',
    maintainer_email='bob@gmail.com',
    description='Examples of minimal publisher and subscriber using rclpy',
    license='Apache-2.0',
    tests_require=['pytest'],
    entry_points={
        'console_scripts': [
            'talker = py_pubsub.publisher_member_function:main',
            'listener = py_pubsub.subscriber_member_function:main',
        ],
    },
)
```

5. Navigate into ros2_ws/src/py_pubsub/py_pubsub and Download the example subscriber code

```
$ cd ~/ros2_ws/src/py_pubsub/py_pubsub

$ wget
https://raw.githubusercontent.com/ros2/examples/jazzy/rclpy/topics/minimal_subscri
ber/examples_rclpy_minimal_subscriber/subscriber_member_function.py
```

6. Build and Run

```
$ cd ~/ros2_ws

$ rosdep install -i --from-path src --rosdistro jazzy -y

$ colcon build --packages-select py_pubsub
```

Open a new terminal and run:

```
$ ros2 run py_pubsub talker
```

Open another terminal and run:

```
$ ros2 run py_pubsub listener
```

Press Ctrl+C in each terminal to stop the process in it.

7 Create a publisher to control the turtle using Python

1. Create a package

```
$ cd ~/ros2_ws/src

$ ros2 pkg create --build-type ament_python --license Apache-2.0
turtle_publisher_py
```

2. Create the source code
   Head to the package's Python module directory in VS Code. This is the inner folder named the same as your package, which contains the Python source files.

   Create a new Python file turtle_controller.py by right click and "New File". Paste the following code below:

```
import rclpy

from rclpy.node import Node

from geometry_msgs.msg import Twist


class TurtleController(Node):

    def __init__(self):

        super().__init__('turtle_controller')

        self.publisher_ = self.create_publisher(Twist, '/turtle1/cmd_vel', 10)

        timer_period = 0.5

        self.timer = self.create_timer(timer_period, self.move_turtle)

    def move_turtle(self):

        msg = Twist()

        msg.linear.x = 2.0

        msg.angular.z = 1.0
```

```
        self.publisher_.publish(msg)

        self.get_logger().info('Moving turtle')


def main(args=None):

    rclpy.init(args=args)

    node = TurtleController()

    rclpy.spin(node)

    node.destroy_node()

    rclpy.shutdown()


if __name__ == '__main__':

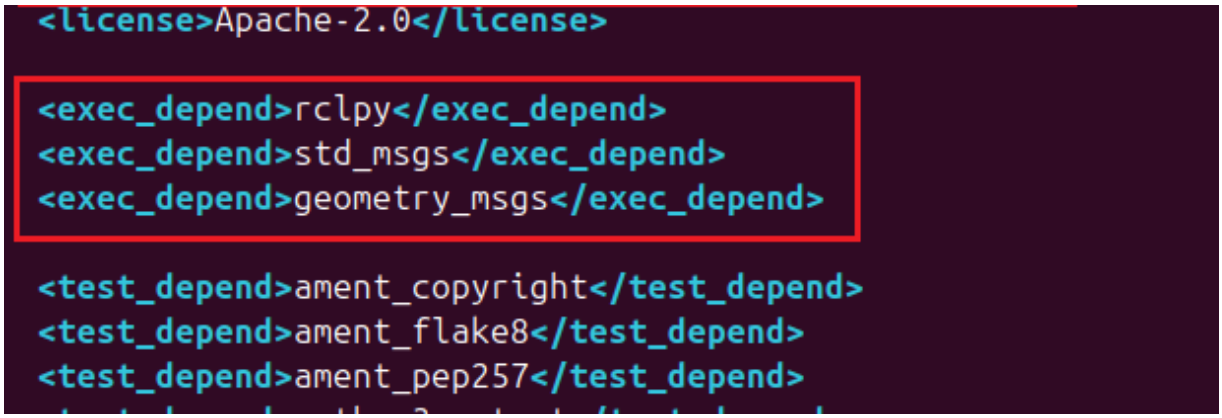
    main()
```

3. Edit package.xml
   Head back to the package directory and edit the package.xml file to edit the description & maintainer section and add the following:

```
<exec_depend>rclpy</exec_depend>

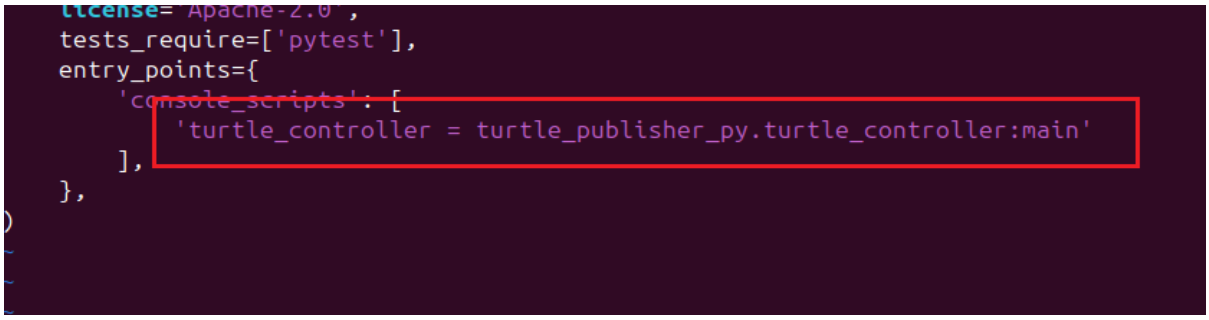
<exec_depend>std_msgs</exec_depend>

<exec_depend>geometry_msgs</exec_depend>
```



4. Edit setup.py
   Edit the setup.py file too by editing the description & maintainer section and add the following line within the console_scripts brackets of the entry_points field:

```
entry_points={
        'console_scripts': [
                'turtle_controller = turtle_publisher_py.turtle_controller:main'
        ],
},
```



5. Run the script
   It is generally good practice to run "rosdep install" before building.

```
$ cd ~/ros2_ws

$ rosdep install -i --from-path src --rosdistro jazzy -y
```

Head back to the root of the workspace (~/ros2_ws) and rebuild

```
$ cd ~/ros2_ws

$ colcon build --packages-select turtle_publisher_py

$ source install/setup.bash

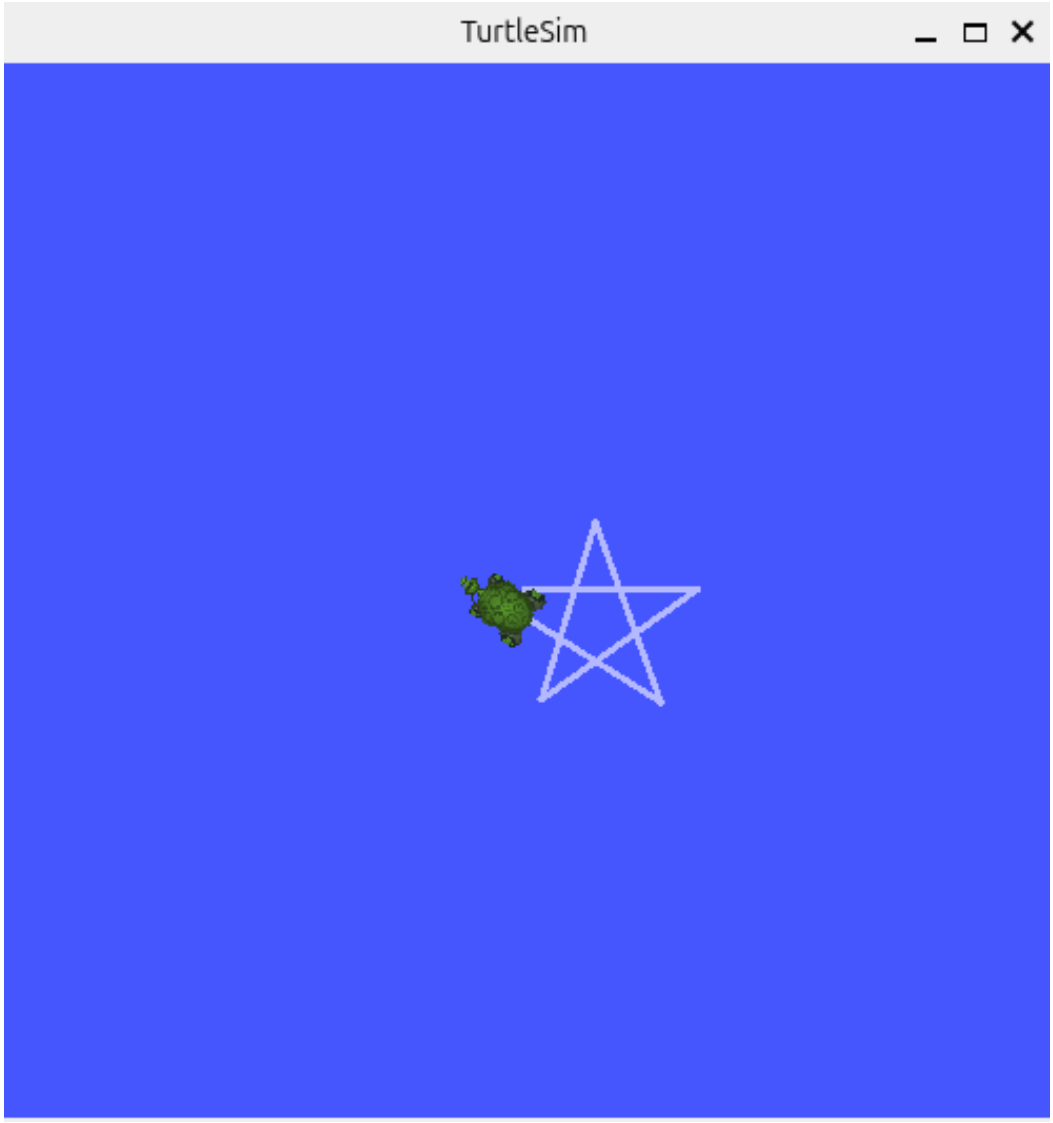
$ ros2 run turtlesim turtlesim_node
```

Open a new terminal to run the turtlesim:

```
$ ros2 run turtle_publisher_py turtle_controller
```

Then you can see the turtle constantly moving in a circular path.

6. Have a trial?
   How to modify your publisher to let the turtle move in a more complex path? Let your turtle draw a star?


TurtleSim