ROS2 CORE

ros2 pkg executables pkg_name: lists all execs in the package
ros2 run pkg_name node_name: to run executable

I initially started off with turtlesim node and teleop key as always

ros2 run turtlesim turtlesim_node: runs turtlesim
ros2 run teleop_twist_keyboard teleop_twist_keyboard --remap cmd_vel:=/turtle1/cmd_vel: instead of using turtlesim teleop, I remapped it

rqt, rqt_graph: GUI to visualize and use various topics, services etc
ros2 run rqt_console rqt_console: view logs

Nodes: executables used to send and receive data
ros2 node list, info

Topics: an intermediary between two nodes where some publisher nodes can connect to some subscriber nodes
ros2 topic list, list -t (type), echo, info, pub –once or pub msg, hz

ros2 interface show msg_type/service_type/action_type: shows the message params

Services: a request-response model used to communicate between nodes
ros2 service list, list -t, type service_name, call service_name service_type (params)

ros2 find service_type

Parameters: values that can be changed inside a node
ros2 param list, ros2 param get node_name parameter value, ros2 param dump node_name

Actions: node communication which includes a goal, feedback and result
ros2 action list, list -t, info action_name, send_goal action_name action_type goal

Launch files: run multiple nodes using one command
ros2 launch package_name launch_arguments

ros2 pkg create –build-type ament_cmake pkg_name –dependencies list_of_dependencies: create a pkg

colcon build, colcon build –packages-select

Custom Pub and Sub in Python
You can find the nodes in the nodes directory. I used an ament_cmake build-type. Add the following code in your CmakeLists.txt

```
install(PROGRAMS
nodes/pub_python.py
nodes/sub_python.py
DESTINATION lib/${PROJECT_NAME}
```

)

and the following code in your package.xml

```
<exec_depend>rclpy</exec_depend>
<depend>std_msgs</depend>
```

colcon build the package, source install/setup.bash and then ros2 run pkg_name publisher_node_name and ros2 run pkg_name subscriber_node_name

Custom Server and Client in Python
The nodes are present in the nodes directory. ros2 run ros2 run pubsub serv_python.py and in another terminal ros2 run pubsub client_python.py integer_val_1 integer_val_2

add the following snippet in your in CmakeLists.txt within intall PROGRAMS

```
nodes/serv_python.py
nodes/client_python.py
```

and add this line to the package.xml:
```
<depend>example_interfaces</depend>
```

Custom Interface
2 .msg files and 1 .srv file

Add this part to CmakeLists.txt:

```
find_package(geometry_msgs REQUIRED)
find_package(rosidl_default_generators REQUIRED)

rosidl_generate_interfaces($(PROJECT_NAME)
"msg/num.msg"
"msg/sphere.msg"
"srv/addthreeints.srv"
DEPENDENCIES geometry_msgs
)
```

and in package.xml:

```
<buildtool_depend>rosidl_default_generators</buildtool_depend>
<exec_depend>rosidl_default_runtime</exec_depend>
<member_of_group>rosidl_interface_packages</member_of_group>
```

message filenames must start with a capital letter

ros2 interface show pubsub/msg/Num: displays the msg info

Now to run this using pubsub and srv:
<depend>pubsub</depend> in package.xml and find_package(pubsub REQUIRED) in Cmakelists

I used num msg to send a number and sphere msg to calculate equation of the sphere

ROS2 Doctor: checks if dependencies and commands are upto date and checks nodes
ros2 doctor, ros2 doctor –report

ROS2 Plugins: loads new functionality to your code without having to recompile code dynamically

Rosdep: tool to manage dependencies

You can run a launch file from within another launch file

from the repository u can use
ros2 launch pubsub main_launch.launch.py

Now for the final project in using various launch files,
ros2 launch pubsub launch_turtlesim.launch.py