PDE4430

ROS

Middlesex
University
Dubai

ROS Ecosystem

- Great to see publishers/subscribers in action – But how to make my own stuff? Where do I put it? How do I compile it?

ROS Ecosystem

- What is a workspace?
- What is a package?
- How to create a workspace?
- How to create a package?

ROS Workspace

- A ROS workspace is a folder in your computer where you can organize various ROS project files.
- A ROS Workspace is actually a 'Catkin' workspace
- Catkin is the official build system of ROS
- A Catkin workspace is a ROS workspace that uses catkin as a build tool

The name comes from the flower cluster of willow trees – a reference to the company Willow Garage – The 'founders' of ROS

Catkin Workspace

- What is a build system?
- A build system is responsible for generating 'targets' from raw source code that can be used by an end user.
- These targets may be in the form of libraries, executable programs, generated scripts or anything else that is not static code.
- In ROS terminology, source code is organized into **'packages'** where each package typically consists of one or more targets when built.

Catkin Workspace

- To build targets, the build system needs information such as the source code locations, dependencies, where those dependencies are located, which targets should be built, where targets should be built, and where they should be installed.
- This is typically expressed in some set of configuration files read by the build system.
- With Catkin, it is specified in a file typically called **`CMakeLists.txt`** – We will come back to this soon…
- Catkin utilises this information to process and build source code in the appropriate order to generate targets.

Catkin Workspace

- ROS is installed in **`/opt/ros/noetic`**
- That is also the default workspace
- A workspace contains a **`setup.bash`** that should be "sourced"
- The **`source`** command can be used to load any functions file into the current shell script or a command prompt

Catkin Workspace

- If the `setup.bash` file is not sourced, none of the commands will work
- Each time you open a new terminal, you have to type:

```
source /opt/ros/noetic/setup.bash
```

- Extremely annoying! There has to be an easier way, right?
- `~/.bashrc`
- This file is run each time a new terminal is started
- Add this line at the end of this file and restart the terminal

Catkin Workspace

- We've just seen the default workspace location
- Let's make our own workspace now
- Good practice: Make a folder named ROS in the home directory. Make a workspace inside that directory.

```
mkdir -p ~/ros/catkin_ws/src
cd ~/ros/catkin_ws/
catkin_make
```

Catkin Workspace

- Go to your folder and see what it looks like

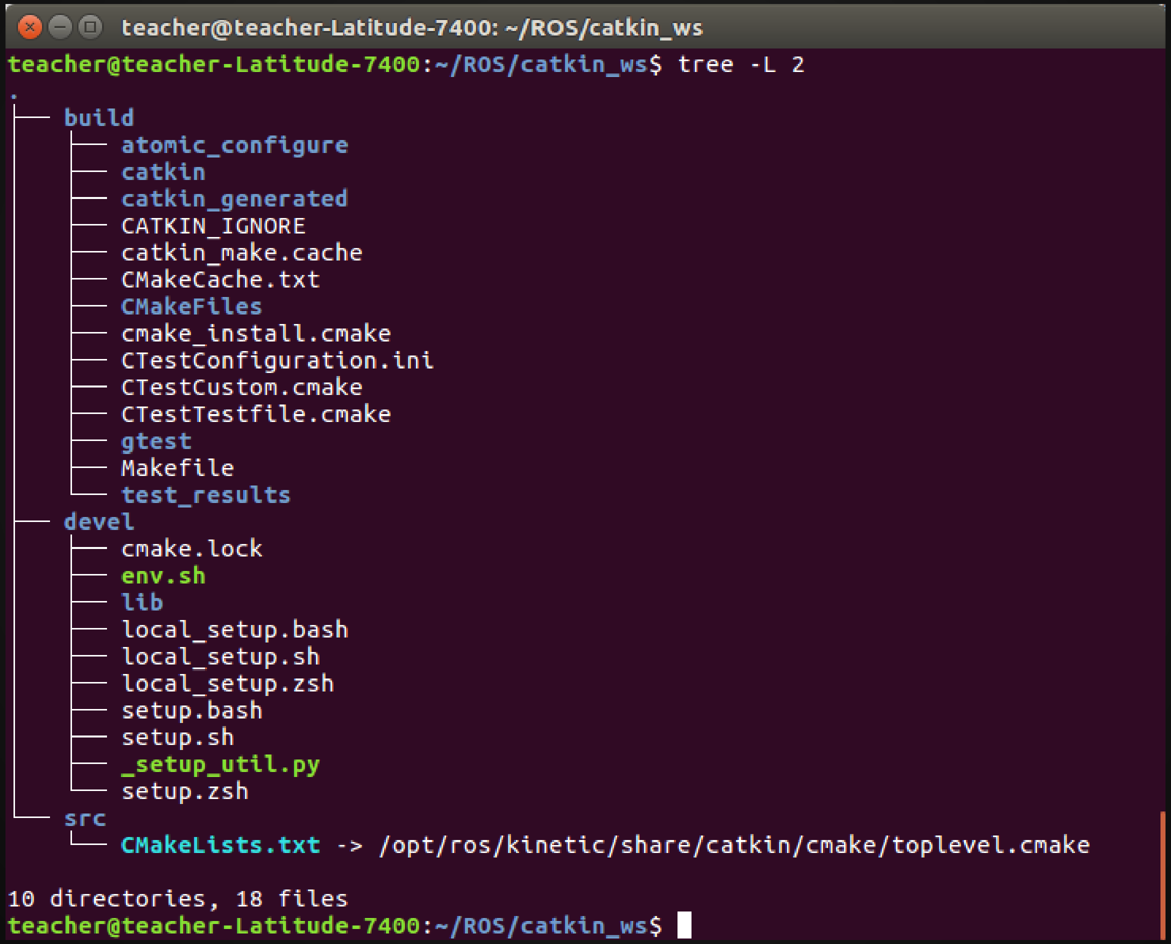
```
ls
tree -L 1
```

If it doesn’t work –

```
sudo apt install tree
```

- Tree is a recursive directory listing command that produces a colourised depth indented listing of files
- -L is the parameter for level of depth

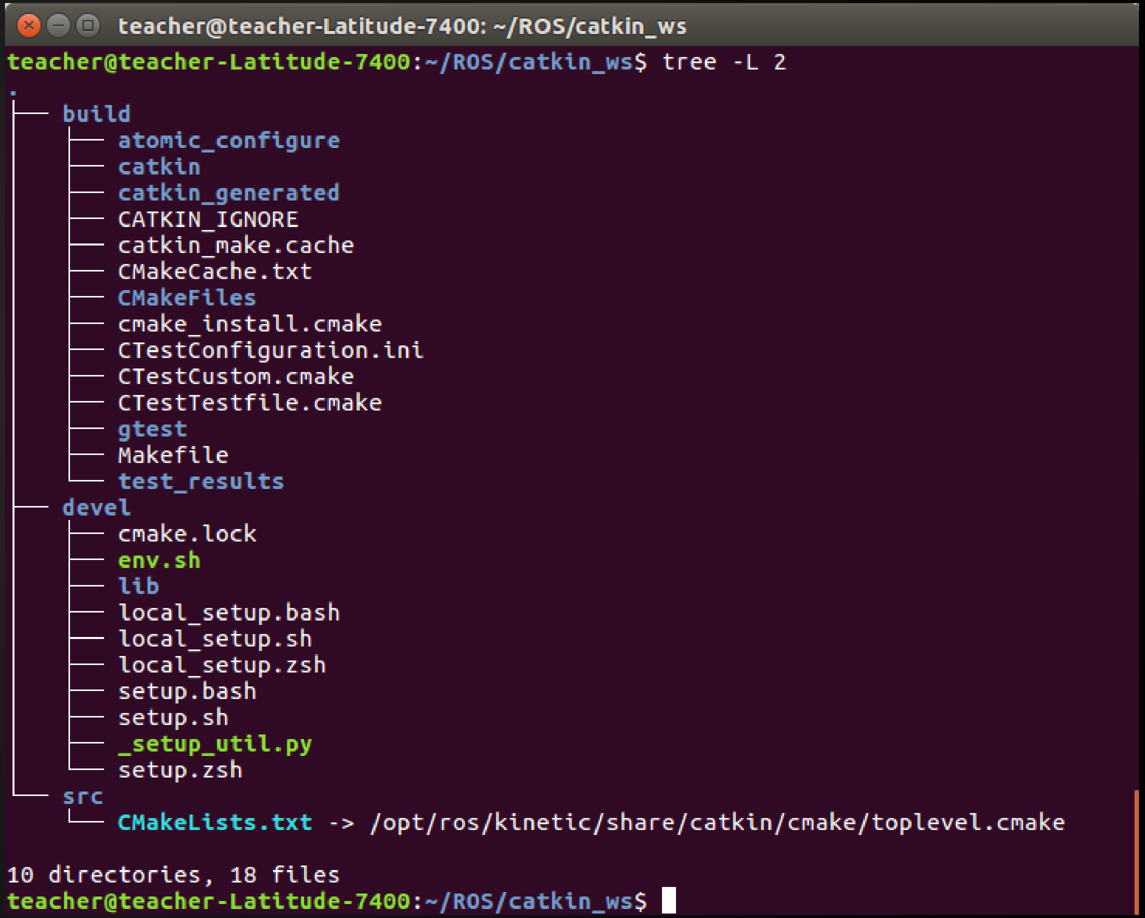
Catkin Workspace



```
teacher@teacher-Latitude-7400:~/ROS/catkin_ws$ tree -L 2
.
├── build
│   ├── atomic_configure
│   ├── catkin
│   ├── catkin_generated
│   ├── CATKIN_IGNORE
│   ├── catkin_make.cache
│   ├── CMakeCache.txt
│   ├── CMakeFiles
│   ├── cmake_install.cmake
│   ├── CTestConfiguration.ini
│   ├── CTestCustom.cmake
│   ├── CTestTestfile.cmake
│   ├── gtest
│   ├── Makefile
│   └── test_results
├── devel
│   ├── cmake.lock
│   ├── env.sh
│   ├── lib
│   ├── local_setup.bash
│   ├── local_setup.sh
│   ├── local_setup.zsh
│   ├── setup.bash
│   ├── setup.sh
│   ├── _setup_util.py
│   └── setup.zsh
└── src
    └── CMakeLists.txt -> /opt/ros/kinetic/share/catkin/cmake/toplevel.cmake

10 directories, 18 files
teacher@teacher-Latitude-7400:~/ROS/catkin_ws$
```

Catkin Workspace

- The **catkin_make** command is a convenience tool for working with catkin workspaces
- **CMakeLists.txt** in your **/src** folder – Remember this?
- You should now have a **'build'** and **'devel'** folder
- Inside the **'devel'** folder you can see that there are now several files. Sourcing the **setup.bash** will overlay this workspace on top of your environment
- Two ways of sourcing – What are they?



```
teacher@teacher-Latitude-7400:~/ROS/catkin_ws$ tree -L 2
.
├── build
│   ├── atomic_configure
│   ├── catkin
│   ├── catkin_generated
│   ├── CATKIN_IGNORE
│   ├── catkin_make.cache
│   ├── CMakeCache.txt
│   ├── CMakeFiles
│   ├── cmake_install.cmake
│   ├── CTestConfiguration.ini
│   ├── CTestCustom.cmake
│   ├── CTestTestfile.cmake
│   ├── gtest
│   ├── Makefile
│   └── test_results
├── devel
│   ├── cmake.lock
│   ├── env.sh
│   ├── lib
│   ├── local_setup.bash
│   ├── local_setup.sh
│   ├── local_setup.zsh
│   ├── setup.bash
│   ├── setup.sh
│   ├── _setup_util.py
│   └── setup.zsh
└── src
    └── CMakeLists.txt -> /opt/ros/kinetic/share/catkin/cmake/toplevel.cmake

10 directories, 18 files
teacher@teacher-Latitude-7400:~/ROS/catkin_ws$ 
```

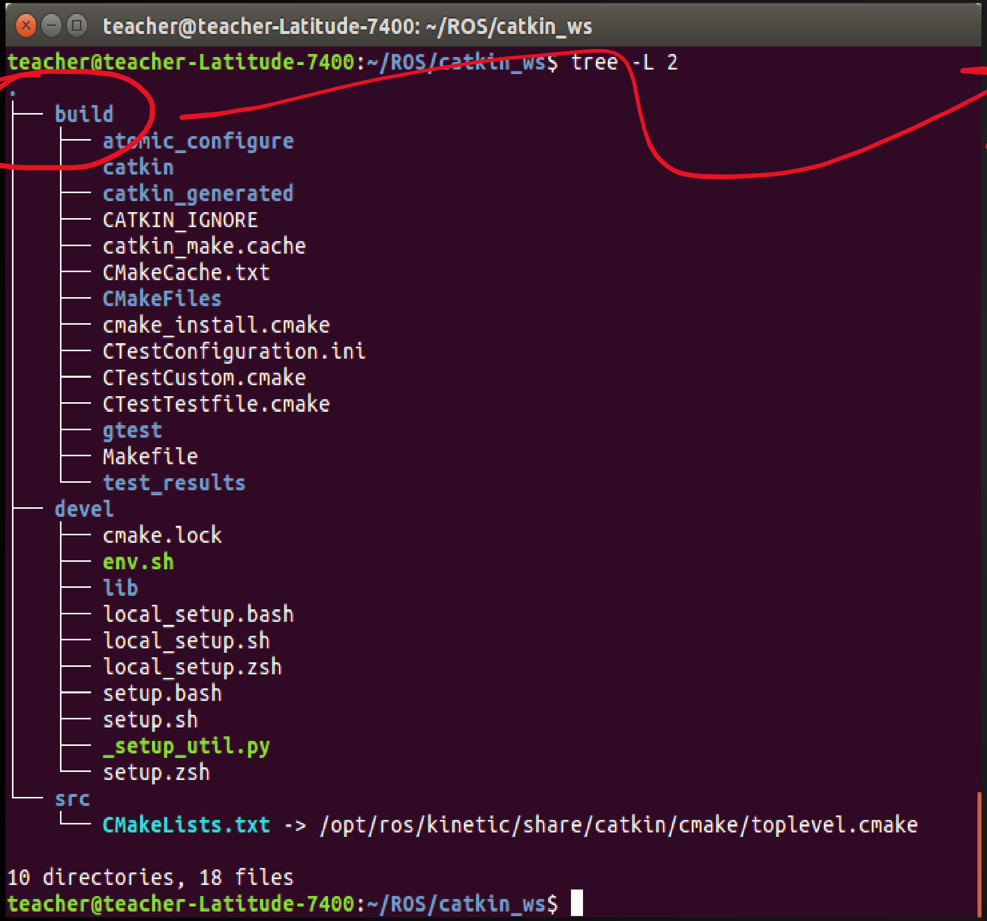
Catkin Workspace

- How to make sure that the workspace is sourced correctly?
- Check the environment variable **ROS_PACKAGE_PATH**

```
echo $ROS_PACKAGE_PATH
```

- Or, use the command: **roscd**
- Try the command before and after sourcing – See the difference

Catkin Workspace



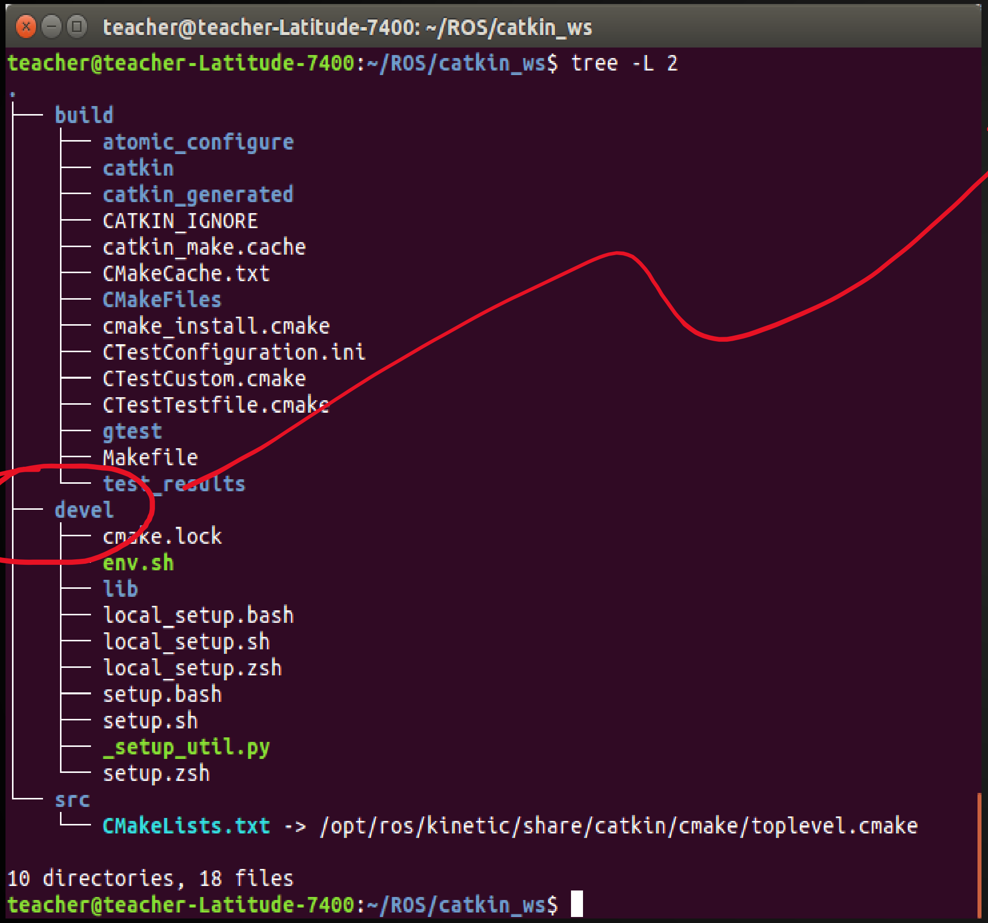
teacher@teacher-Latitude-7400: ~/ROS/catkin_ws

```
teacher@teacher-Latitude-7400:~/ROS/catkin_ws$ tree -L 2
.
├── build
│   ├── atomic_configure
│   ├── catkin
│   ├── catkin_generated
│   ├── CATKIN_IGNORE
│   ├── catkin_make.cache
│   ├── CMakeCache.txt
│   ├── CMakeFiles
│   ├── cmake_install.cmake
│   ├── CTestConfiguration.ini
│   ├── CTestCustom.cmake
│   ├── CTestTestfile.cmake
│   ├── gtest
│   ├── Makefile
│   └── test_results
├── devel
│   ├── cmake.lock
│   ├── env.sh
│   ├── lib
│   ├── local_setup.bash
│   ├── local_setup.sh
│   ├── local_setup.zsh
│   ├── setup.bash
│   ├── setup.sh
│   ├── _setup_util.py
│   └── setup.zsh
└── src
    └── CMakeLists.txt -> /opt/ros/kinetic/share/catkin/cmake/toplevel.cmake

10 directories, 18 files
teacher@teacher-Latitude-7400:~/ROS/catkin_ws$
```

Build Space

- Used by Catkin to store intermediate files during the build process

Catkin Workspace



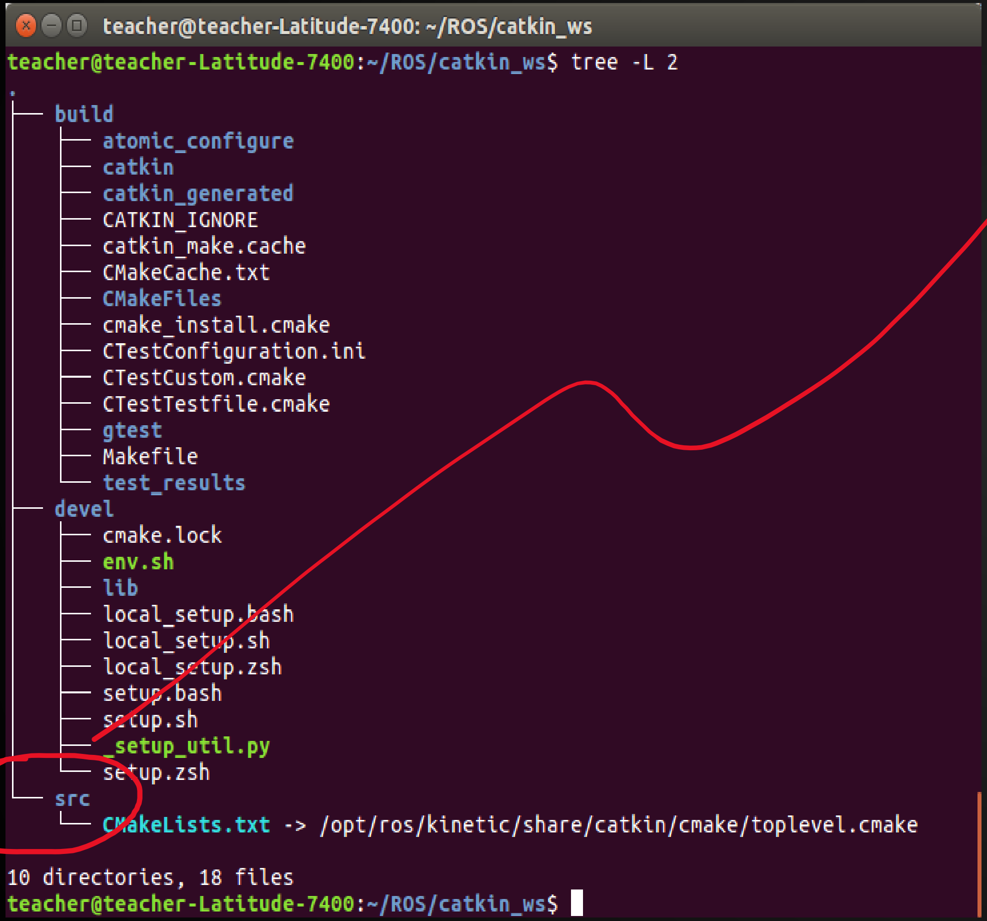
teacher@teacher-Latitude-7400: ~/ROS/catkin_ws

```
teacher@teacher-Latitude-7400:~/ROS/catkin_ws$ tree -L 2
.
├── build
│   ├── atomic_configure
│   ├── catkin
│   ├── catkin_generated
│   ├── CATKIN_IGNORE
│   ├── catkin_make.cache
│   ├── CMakeCache.txt
│   ├── CMakeFiles
│   ├── cmake_install.cmake
│   ├── CTestConfiguration.ini
│   ├── CTestCustom.cmake
│   ├── CTestTestfile.cmake
│   ├── gtest
│   ├── Makefile
│   └── test_results
├── devel
│   ├── cmake.lock
│   ├── env.sh
│   ├── lib
│   ├── local_setup.bash
│   ├── local_setup.sh
│   ├── local_setup.zsh
│   ├── setup.bash
│   ├── setup.sh
│   ├── _setup_util.py
│   └── setup.zsh
└── src
    └── CMakeLists.txt -> /opt/ros/kinetic/share/catkin/cmake/toplevel.cmake

10 directories, 18 files
teacher@teacher-Latitude-7400:~/ROS/catkin_ws$ 
```

Devel Space

- Contains files for setting up a project-specific ROS environment
- Contains executables of source files for development and testing

Catkin Workspace



```
teacher@teacher-Latitude-7400:~/ROS/catkin_ws$ tree -L 2
.
├── build
│   ├── atomic_configure
│   ├── catkin
│   ├── catkin_generated
│   ├── CATKIN_IGNORE
│   ├── catkin_make.cache
│   ├── CMakeCache.txt
│   ├── CMakeFiles
│   ├── cmake_install.cmake
│   ├── CTestConfiguration.ini
│   ├── CTestCustom.cmake
│   ├── CTestTestfile.cmake
│   ├── gtest
│   ├── Makefile
│   └── test_results
├── devel
│   ├── cmake.lock
│   ├── env.sh
│   ├── lib
│   ├── local_setup.bash
│   ├── local_setup.sh
│   ├── local_setup.zsh
│   ├── setup.bash
│   ├── setup.sh
│   ├── _setup_util.py
│   └── setup.zsh
└── src
    └── CMakeLists.txt -> /opt/ros/kinetic/share/catkin/cmake/toplevel.cmake

10 directories, 18 files
teacher@teacher-Latitude-7400:~/ROS/catkin_ws$ 
```

Source Space

- Created by you
- Your playground
- All your source code goes here
- Can also put other people’s source code here for packages you want to build – Although if you’re not modifying the code, simply install the binary

ROS Package

- Software in ROS is organized in packages
- A package might contain a library, a dataset, configuration files or anything else that logically constitutes a useful module
- The goal of these packages it to provide this functionality in an easy-to-consume manner so that software can be easily reused
- In general, ROS packages follow a “Goldilocks” principle: Enough functionality to be useful, but not too much that the package is heavyweight and difficult to use from other software

ROS Package

- Packages go in the **`‘src’`** folder
- Packages may contain multiple nodes
- Put external ROS packages in **`src`** only if you are modifying it
- If you are not modifying it, install the binaries of the package
- Similar to the workspace, we will create a Catkin package
- Catkin package is a type of ROS package

Catkin Package

- Let’s create our Catkin package now:

```
cd ~/ros/catkin_ws/src
catkin_create_pkg packageName dependency1
dependency2 dependency3 etc
```

```
catkin_create_pkg pde4430_session2 std_msgs
rospy
```

Catkin Package

- Requirements for a package to be considered a catkin package:
  - The package must contain a catkin compliant **`package.xml`** file – Provides meta information about the package
  - The package must contain a **`CMakeLists.txt`** which uses catkin
  - Each package must have its own folder – No nested packages/multiple packages sharing the same directory
  - Let’s look at the **`package.xml`** file in more detail…

Package.xml

- Contains lots of information about the package
- Defines properties of the package
- Name, version, description, maintainer and license are required
- 4 different types of dependencies, don’t worry too much now

Catkin Package

- Use **`rospack`** to view the dependencies of a package – Extracted from the **`package.xml`** file
- **`rospack depends1 pde4430_session1`**
- **`rospack depends1 rospy`**
- **`rospack depends pde4430_session1`**
- Run the **`catkin_make`** command again on your workspace

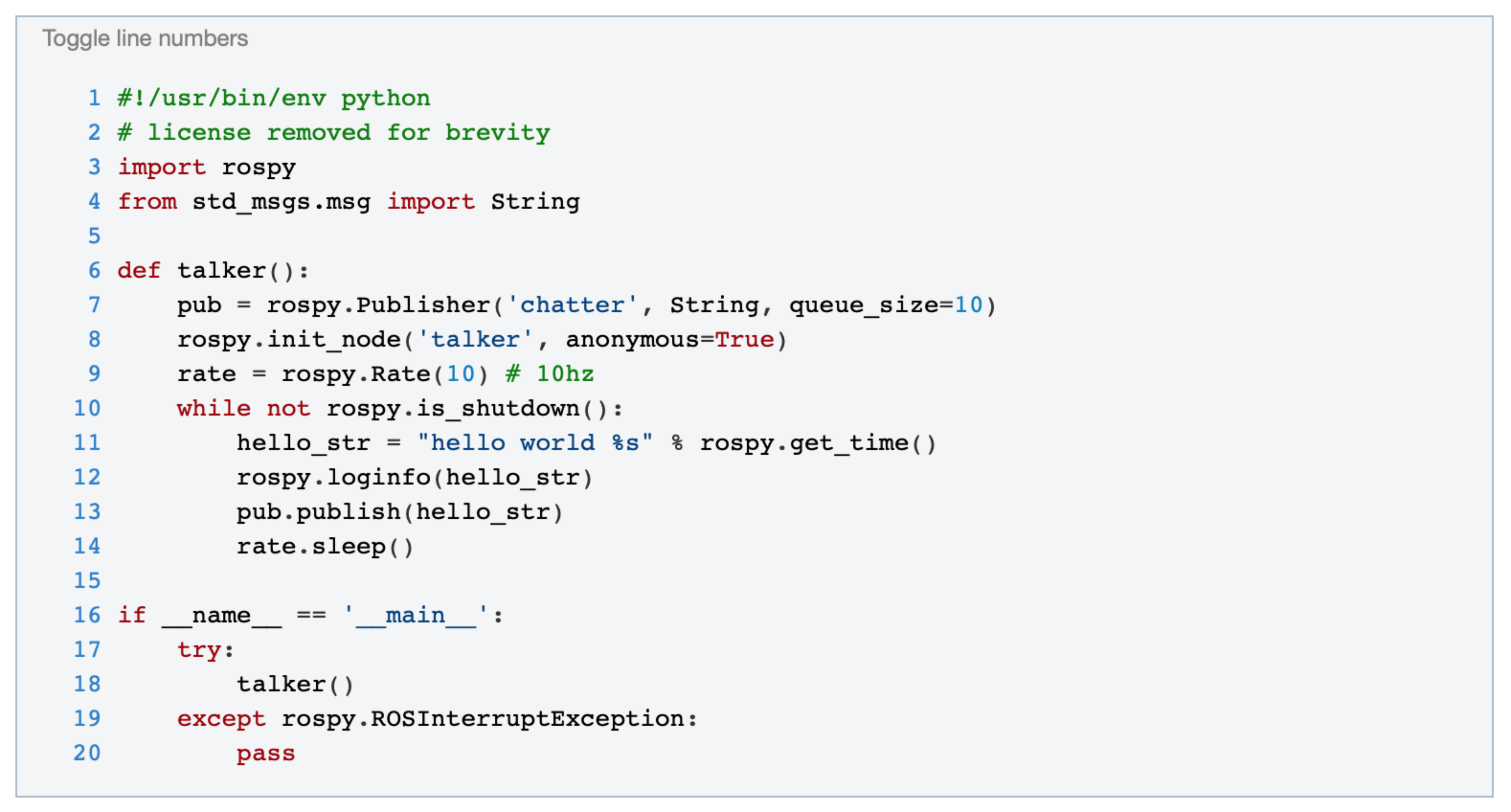
Creating a Publisher and Subscriber

- Simple publisher that broadcasts a message

```
roscd pde4430_session2
mkdir scripts
cd scripts
```

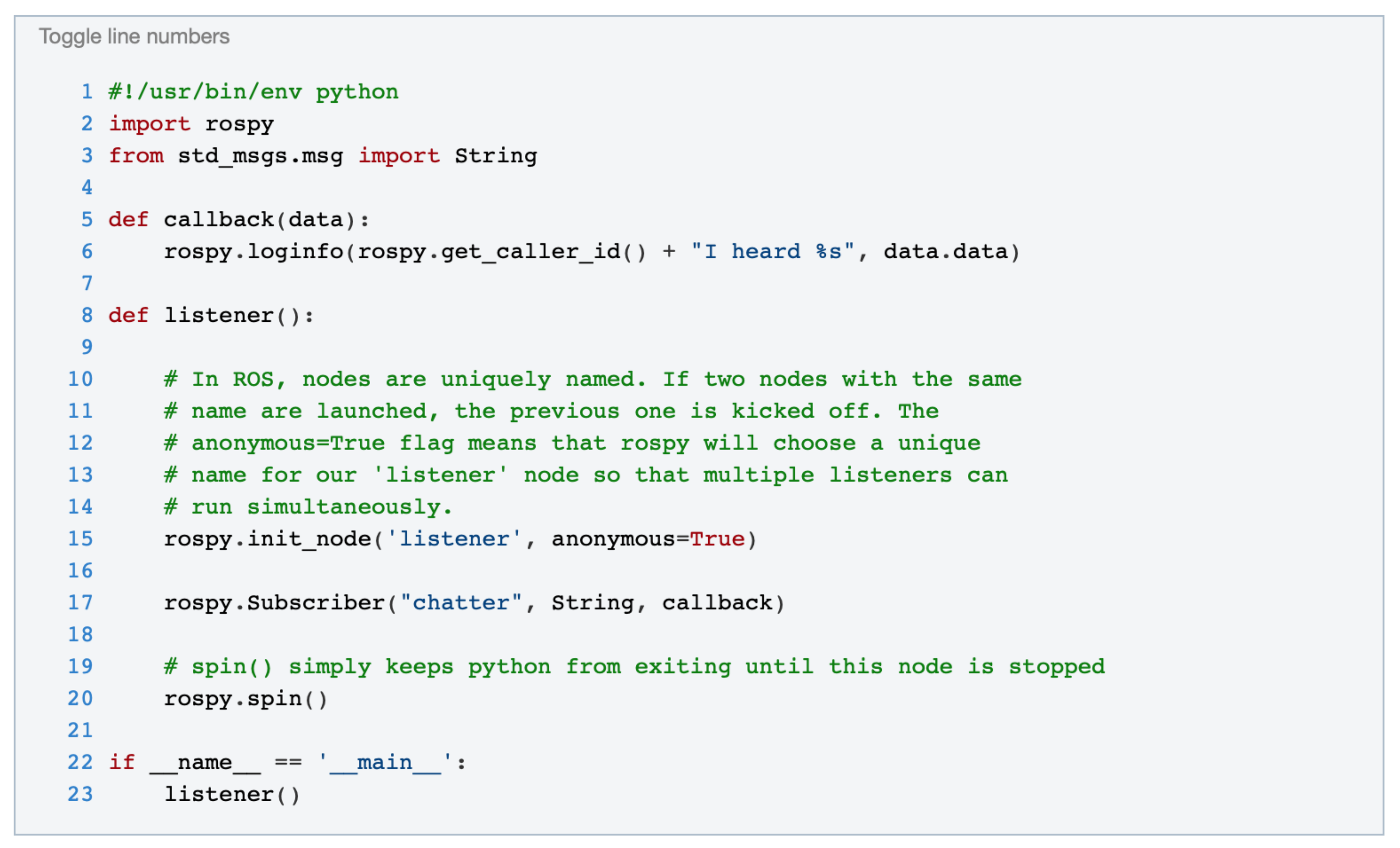
- Open VS Code and create 2 new files – **`talker.py`**; **`listener.py`**

Talker.py



```python
#!/usr/bin/env python
# license removed for brevity
import rospy
from std_msgs.msg import String

def talker():
    pub = rospy.Publisher('chatter', String, queue_size=10)
    rospy.init_node('talker', anonymous=True)
    rate = rospy.Rate(10) # 10hz
    while not rospy.is_shutdown():
        hello_str = "hello world %s" % rospy.get_time()
        rospy.loginfo(hello_str)
        pub.publish(hello_str)
        rate.sleep()

if __name__ == '__main__':
    try:
        talker()
    except rospy.ROSInterruptException:
        pass
```

Listener.py



```python
#!/usr/bin/env python
import rospy
from std_msgs.msg import String

def callback(data):
    rospy.loginfo(rospy.get_caller_id() + "I heard %s", data.data)

def listener():

    # In ROS, nodes are uniquely named. If two nodes with the same
    # name are launched, the previous one is kicked off. The
    # anonymous=True flag means that rospy will choose a unique
    # name for our 'listener' node so that multiple listeners can
    # run simultaneously.
    rospy.init_node('listener', anonymous=True)

    rospy.Subscriber("chatter", String, callback)

    # spin() simply keeps python from exiting until this node is stopped
    rospy.spin()

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

Building the Workspace

- **`chmod +x talker.py`**
- **`chmod +x listener.py`**

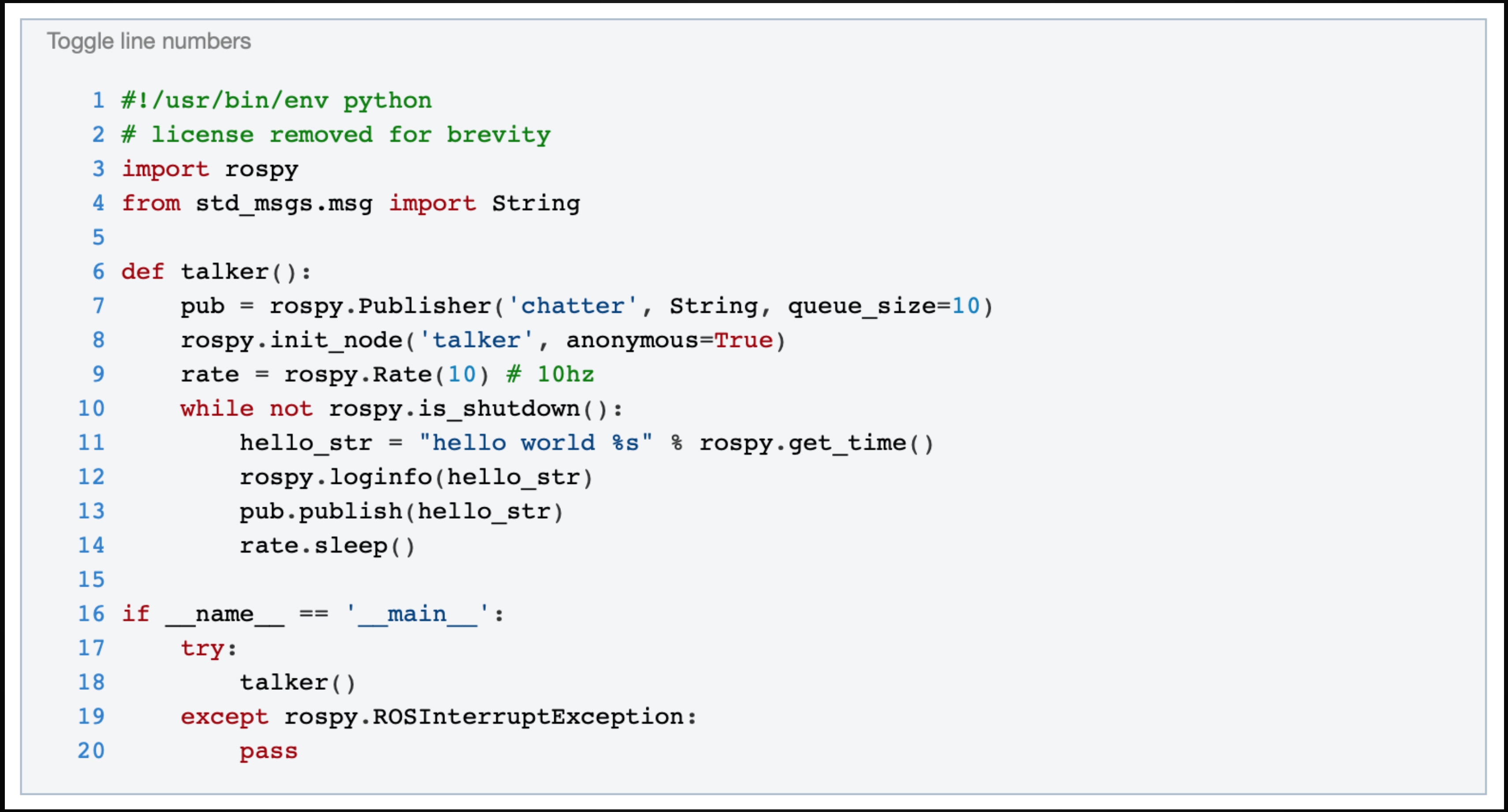
- Run your first ROS app! **`(roscore, rosrun, etc.)`**

- With C++, you have to do **`catkin_make`** each time you make any changes
- With Python, you just have to make them executable once

Running the ROS Application

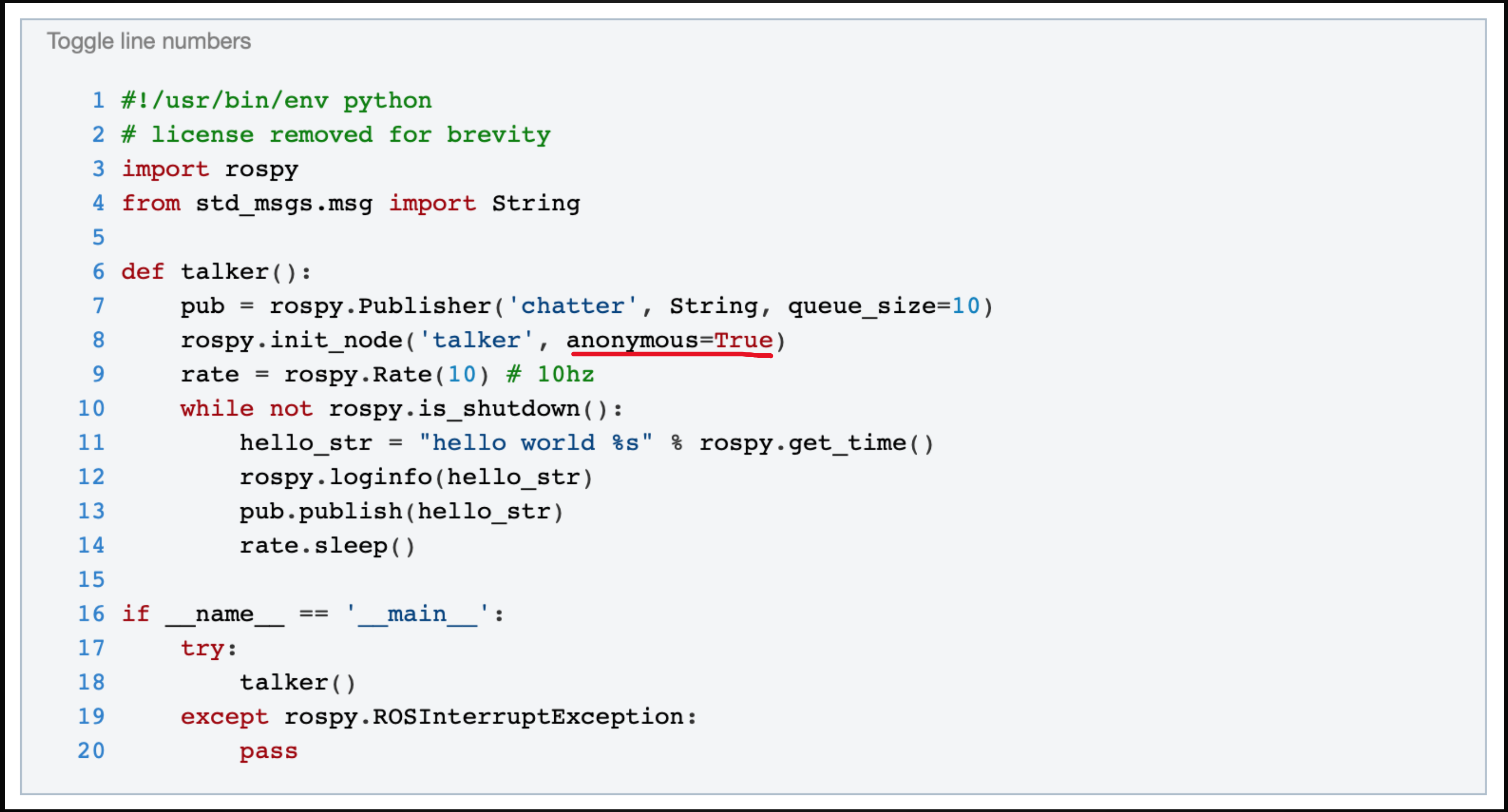
- Remember to check the active nodes using **`rosnode list`**
- Check the topics
- Echo the chatter topic to see what’s going on

ROS Application – Publisher Breakdown



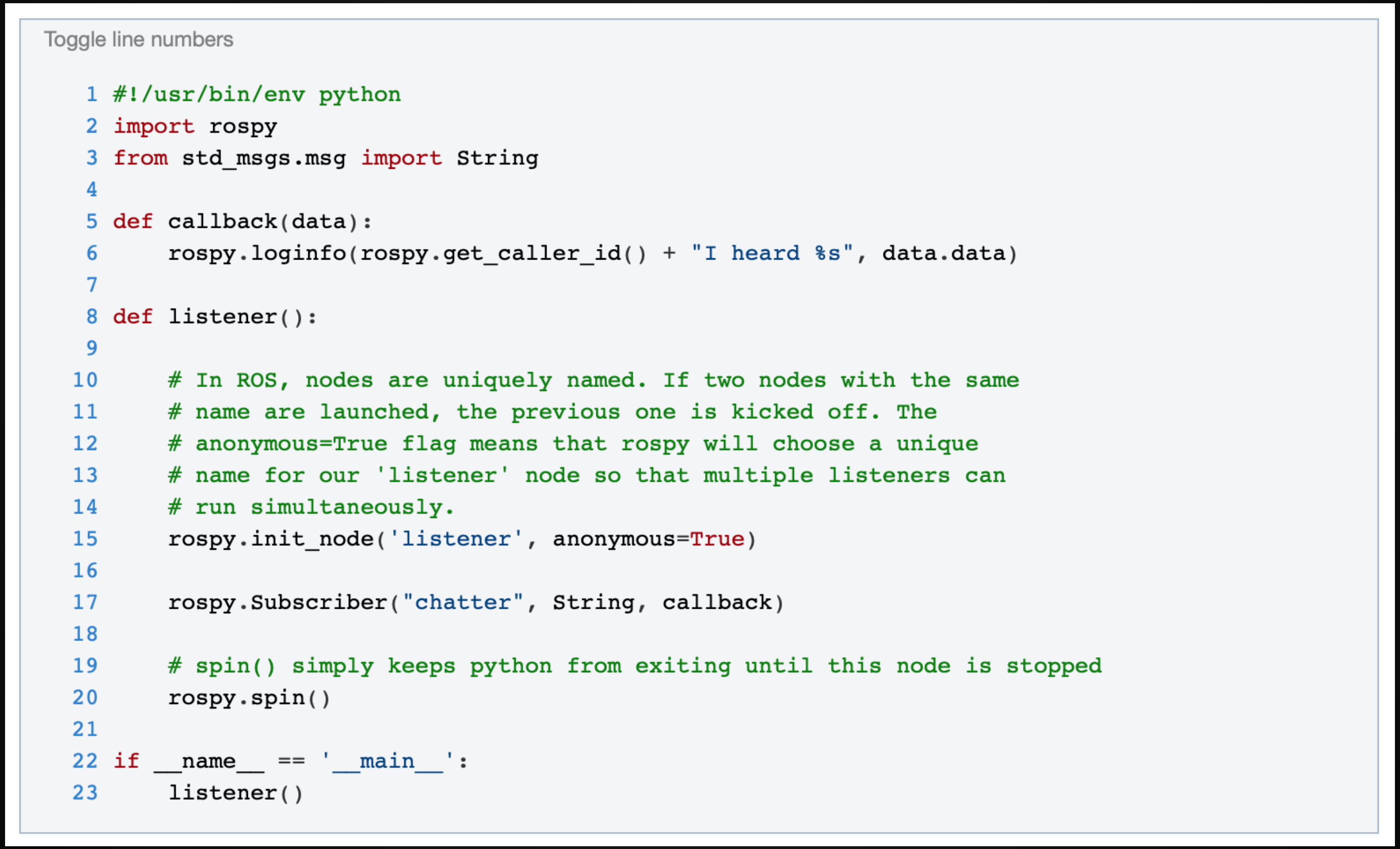
```python
#!/usr/bin/env python
# license removed for brevity
import rospy
from std_msgs.msg import String

def talker():
    pub = rospy.Publisher('chatter', String, queue_size=10)
    rospy.init_node('talker', anonymous=True)
    rate = rospy.Rate(10) # 10hz
    while not rospy.is_shutdown():
        hello_str = "hello world %s" % rospy.get_time()
        rospy.loginfo(hello_str)
        pub.publish(hello_str)
        rate.sleep()

if __name__ == '__main__':
    try:
        talker()
    except rospy.ROSInterruptException:
        pass
```

ROS Application – Publisher Breakdown



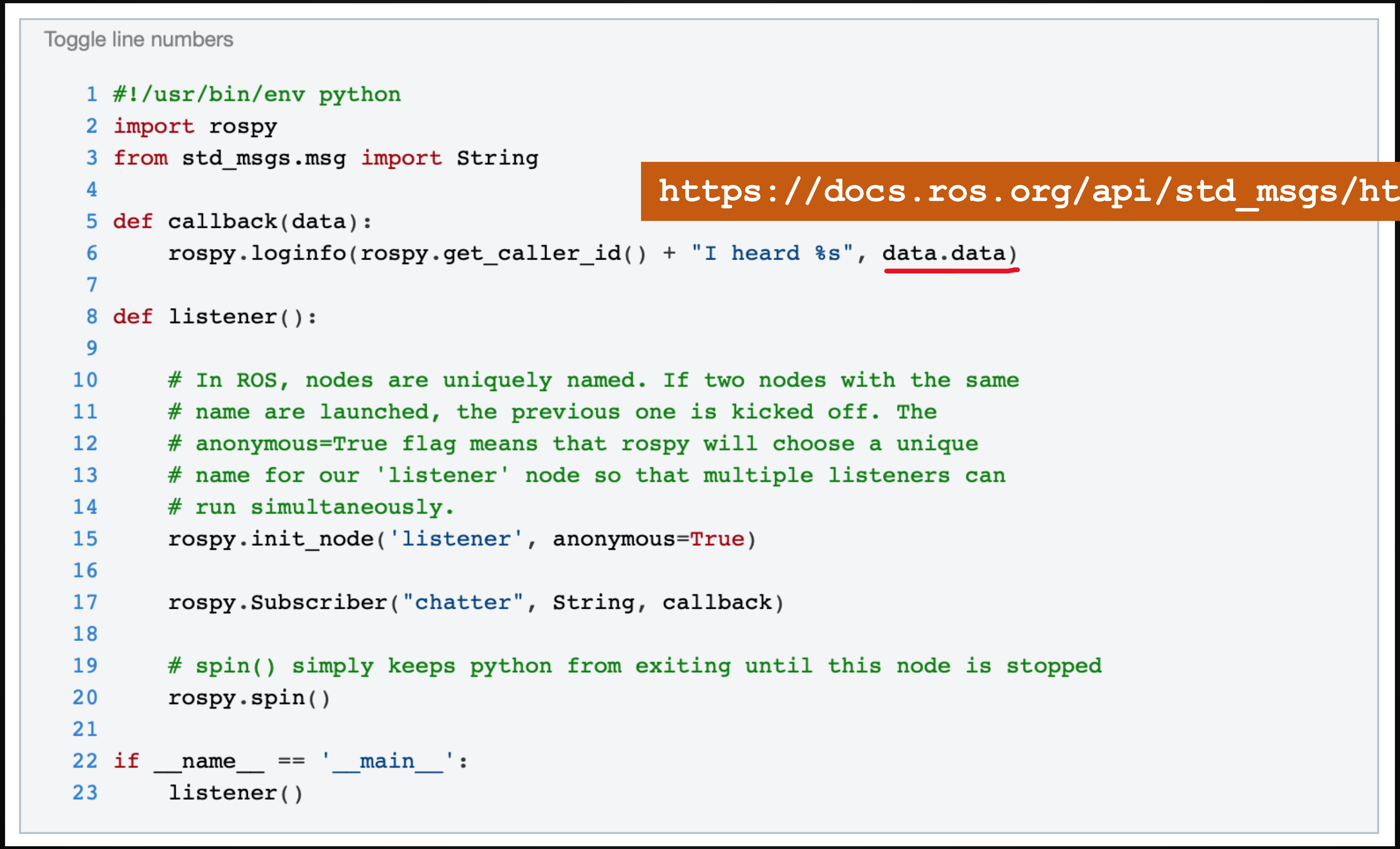
```python
#!/usr/bin/env python
# license removed for brevity
import rospy
from std_msgs.msg import String

def talker():
    pub = rospy.Publisher('chatter', String, queue_size=10)
    rospy.init_node('talker', anonymous=True)
    rate = rospy.Rate(10) # 10hz
    while not rospy.is_shutdown():
        hello_str = "hello world %s" % rospy.get_time()
        rospy.loginfo(hello_str)
        pub.publish(hello_str)
        rate.sleep()

if __name__ == '__main__':
    try:
        talker()
    except rospy.ROSInterruptException:
        pass
```

ROS Application – Subscriber Breakdown



```python
#!/usr/bin/env python
import rospy
from std_msgs.msg import String

def callback(data):
    rospy.loginfo(rospy.get_caller_id() + "I heard %s", data.data)

def listener():

    # In ROS, nodes are uniquely named. If two nodes with the same
    # name are launched, the previous one is kicked off. The
    # anonymous=True flag means that rospy will choose a unique
    # name for our 'listener' node so that multiple listeners can
    # run simultaneously.
    rospy.init_node('listener', anonymous=True)

    rospy.Subscriber("chatter", String, callback)

    # spin() simply keeps python from exiting until this node is stopped
    rospy.spin()

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

ROS Application – Subscriber Breakdown



Toggle line numbers

```python
#!/usr/bin/env python
import rospy
from std_msgs.msg import String

def callback(data):
    rospy.loginfo(rospy.get_caller_id() + "I heard %s", data.data)

def listener():

    # In ROS, nodes are uniquely named. If two nodes with the same
    # name are launched, the previous one is kicked off. The
    # anonymous=True flag means that rospy will choose a unique
    # name for our 'listener' node so that multiple listeners can
    # run simultaneously.
    rospy.init_node('listener', anonymous=True)

    rospy.Subscriber("chatter", String, callback)

    # spin() simply keeps python from exiting until this node is stopped
    rospy.spin()

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

https://docs.ros.org/api/std_msgs/html/msg/String.html

ROS Publisher – Tips

1. Create a **name** for the topic to publish
2. Determine the **type** of the messages that the topic will publish
3. Fix the **frequency** of topic publication (how many messages per second)
4. **Create** a publisher object with parameters chosen
5. Keep **publishing** the topic message at the selected frequency

ROS Subscriber – Tips

1. Identify the **name** of the topic to listen to
2. Know the **message type** to be received
3. Create a **callback** function that is triggered each time a new value is received
4. **Start listening** to the topic
5. Keep listening **forever** (until the node is shutdown)

Exercise

- Modify the code to print a counter instead of the time.
- 10 minutes only.