ELEG4701

Intelligent Interactive Robot Practice

Lab 5: Roslaunch and Service/Client

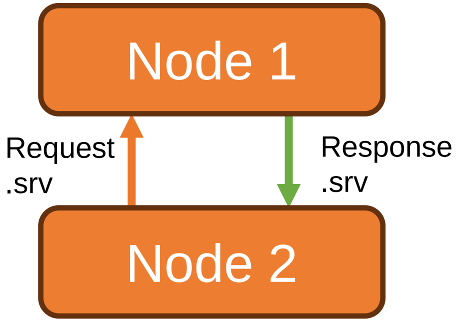
Tutorial

Yameng Zhang
EE, CUHK
zhangyameng@link.cuhk.edu.hk

How to create a ROS srv?

Ref: http://wiki.ros.org/ROS/Tutorials/CreatingMsgAndSrv

Create a simple .srv file



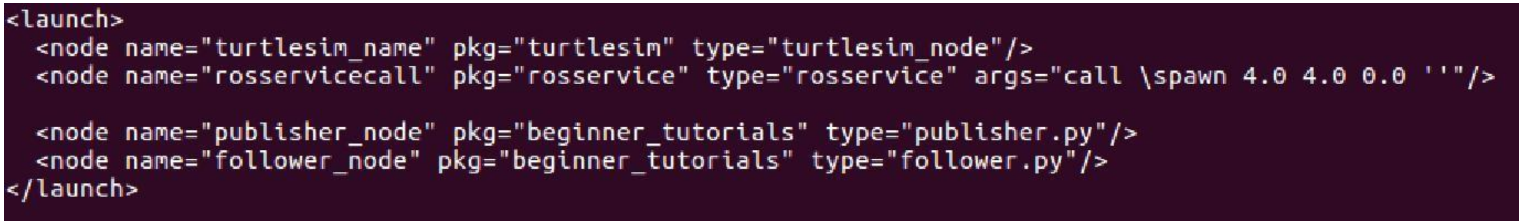
Request/Response interactions

.srv is where we define the data type for ROS service communication

Task 1

- Download the reference answers of `follower.py` and `publisher.py` of Lab 4 (Put them into the correct directory; remove the "_ans")

- Try to write a launch file for these two nodes, launch them using a single `roslaunch` command.
  1. Download the `lab4.launch` from blackboard
  2. The `lab4.launch` already contains the `turtlesim_node` related things
  3. What you need to do is add the `<node>` for the `publisher.py` and `follower.py`
  4. Show the demo to the TA when you can launch everything with one `lab4.launach` file.



```
<launch>
  <node name="turtlesim_name" pkg="turtlesim" type="turtlesim_node"/>
  <node name="rosservicecall" pkg="rosservice" type="rosservice" args="call \spawn 4.0 4.0 0.0 ''"/>

  <node name="publisher_node" pkg="beginner_tutorials" type="publisher.py"/>
  <node name="follower_node" pkg="beginner_tutorials" type="follower.py"/>
</launch>
```

Task 2: Write your first .srv file

- Let’s define a new srv in the package that was created in the previous tutorial

```
$ roscd beginner_tutorials
$ mkdir srv
$ cd srv
$ touch beginner_srv.srv
```

- Open the created .srv file with the editor

```
$ gedit beginner_srv.srv
```

- Write the following content in the .srv file

```
float64 a
float64 b
---
float64 product
```

A service description file consists of a **request** and a **response** msg type, separated by '**---**'. Any two .msg files concatenated together with a '---' are a legal service description.

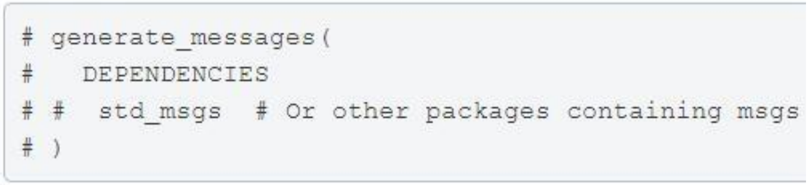
Task 2: Write your first .srv file

- Remove # to uncommented the following lines in the CMakeLists.txt
- Replace the placeholder for your service files:

```
add_service_files(
        FILES
        beginner_srv.srv
)
```

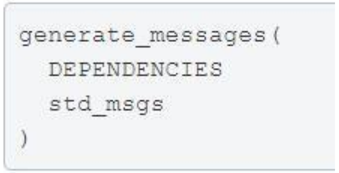
Task 2: Write your first .srv file

- Unless you have already done this in the previous step, change in CmakeLists.txt:



```
# generate_messages(
#   DEPENDENCIES
# #  std_msgs  # Or other packages containing msgs
# )
```

- Uncomment it and add any packages you depend on which contain .msg files that your messages use (in this case std_msgs), such that it looks like this:



```
generate_messages(
  DEPENDENCIES
  std_msgs
)
```

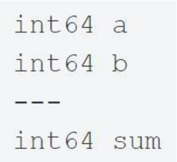
- Now that we have made some new message, we need to Cmake our package again:

```
# in your catkin workspace
$ roscd beginner_tutorials
$ cd ../..
$ catkin_make
$ source ….
```

Create a simple ROS Service and Client

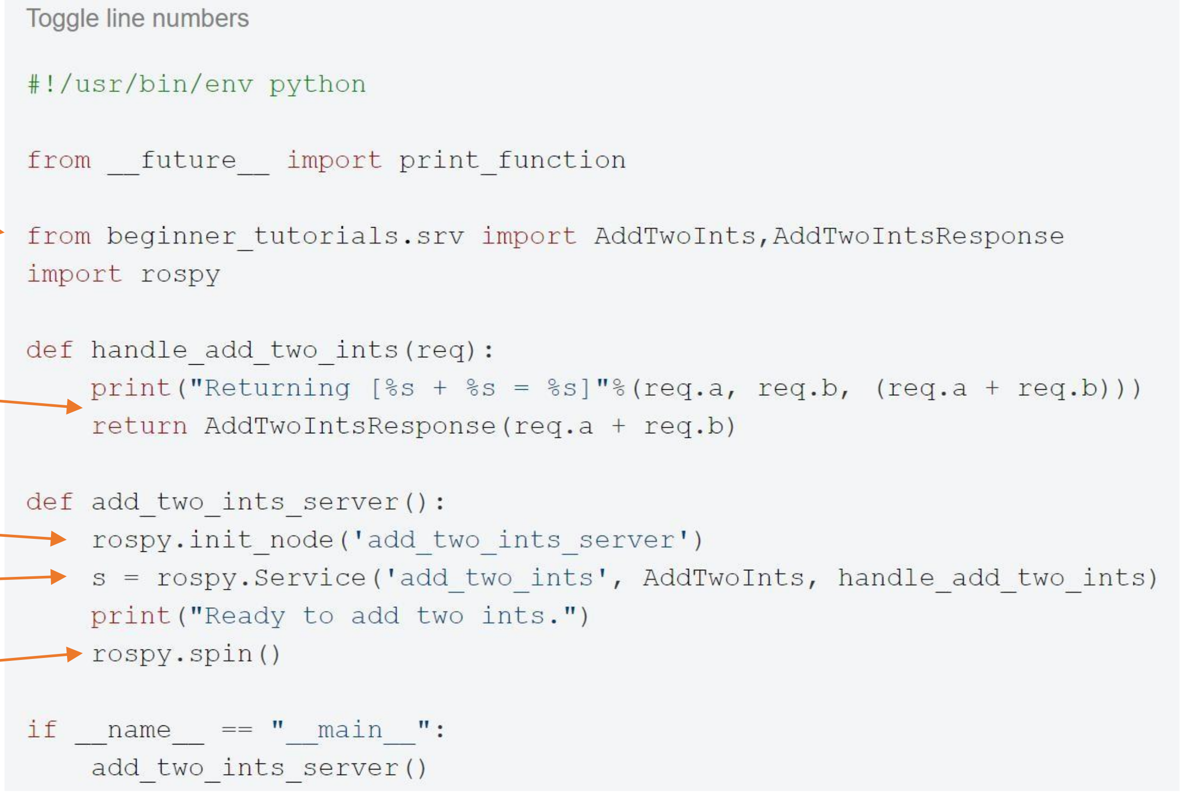
Example:

srv/AddTwoInts.srv



```
int64 a
int64 b
---
int64 sum
```

scripts/add_two_ints_server.py



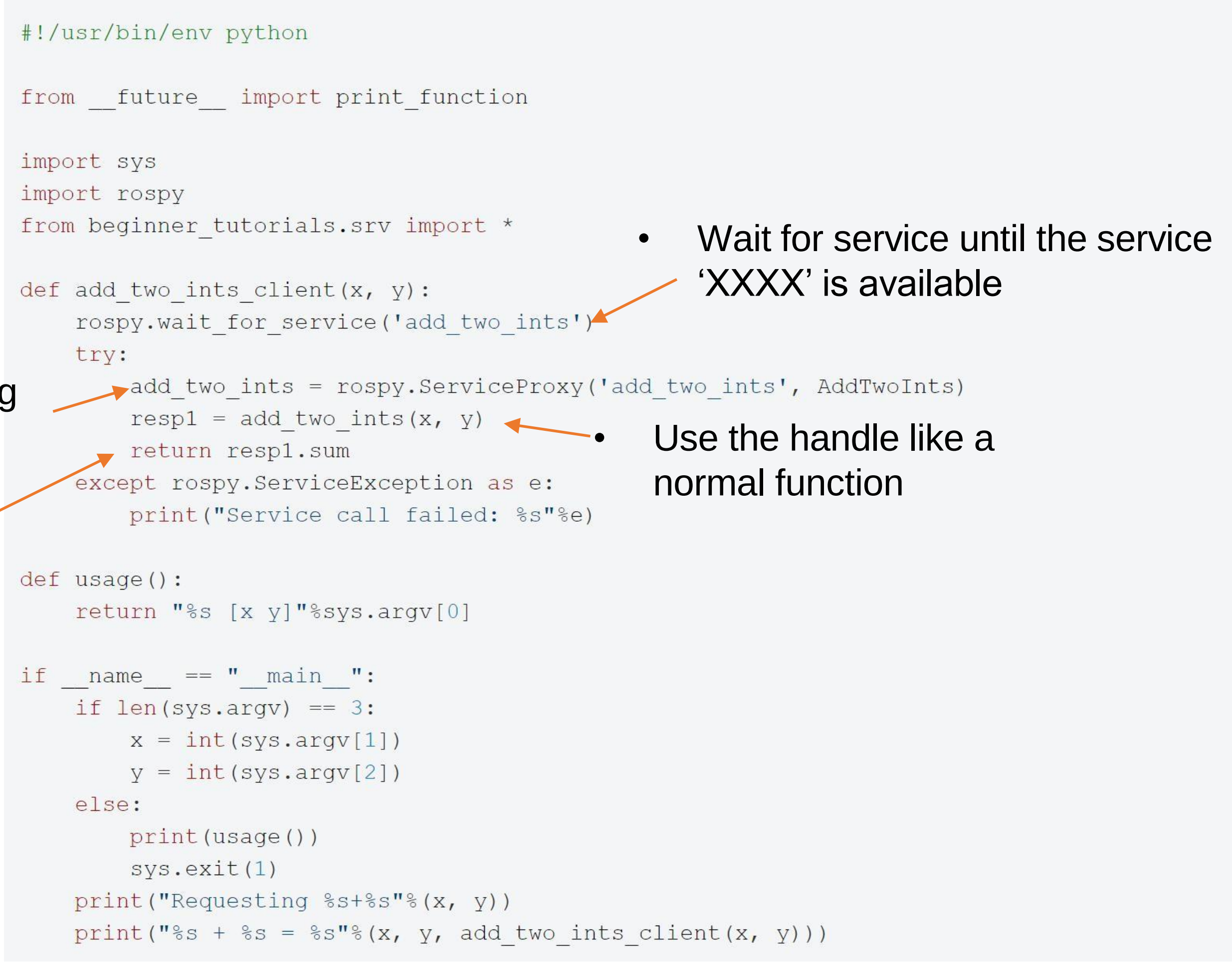
```
#!/usr/bin/env python

from __future__ import print_function

from beginner_tutorials.srv import AddTwoInts,AddTwoIntsResponse
import rospy

def handle_add_two_ints(req):
    print("Returning [%s + %s = %s]"%(req.a, req.b, (req.a + req.b)))
    return AddTwoIntsResponse(req.a + req.b)

def add_two_ints_server():
    rospy.init_node('add_two_ints_server')
    s = rospy.Service('add_two_ints', AddTwoInts, handle_add_two_ints)
    print("Ready to add two ints.")
    rospy.spin()

if __name__ == "__main__":
    add_two_ints_server()
```

- Import the srv you need
- What should be returned and printed
- Initialize the node
- rospy.Service
- Check requests

Create a simple ROS Service and Client

scripts/add_two_ints_client.py

Example:



```
#!/usr/bin/env python

from __future__ import print_function

import sys
import rospy
from beginner_tutorials.srv import *

def add_two_ints_client(x, y):
    rospy.wait_for_service('add_two_ints')
    try:
        add_two_ints = rospy.ServiceProxy('add_two_ints', AddTwoInts)
        resp1 = add_two_ints(x, y)
        return resp1.sum
    except rospy.ServiceException as e:
        print("Service call failed: %s"%e)

def usage():
    return "%s [x y]"%sys.argv[0]

if __name__ == "__main__":
    if len(sys.argv) == 3:
        x = int(sys.argv[1])
        y = int(sys.argv[2])
    else:
        print(usage())
        sys.exit(1)
    print("Requesting %s+%s"%(x, y))
    print("%s + %s = %s"%(x, y, add_two_ints_client(x, y)))
```

- Wait for service until the service ‘XXXX’ is available

- Create a handle for calling the service

- Use the handle like a normal function

- Return the function

Task 3: Create a simple ROS Service and Client

The job of Task 3 is to create a service to do the multiplication and use a client to call this service

Task 3: Create a simple ROS Service and Client

- Change dir into beginner_tutorials package you created in the earlier

```
$ roscd beginner_tutorials
$ cd scripts
```

- Download the `lab5_server.py` and `lab5_client.py` from the blackboard to the script directory
- Do the coding job
- Don’t forget to make the node executable (or do it manually throu GUI):

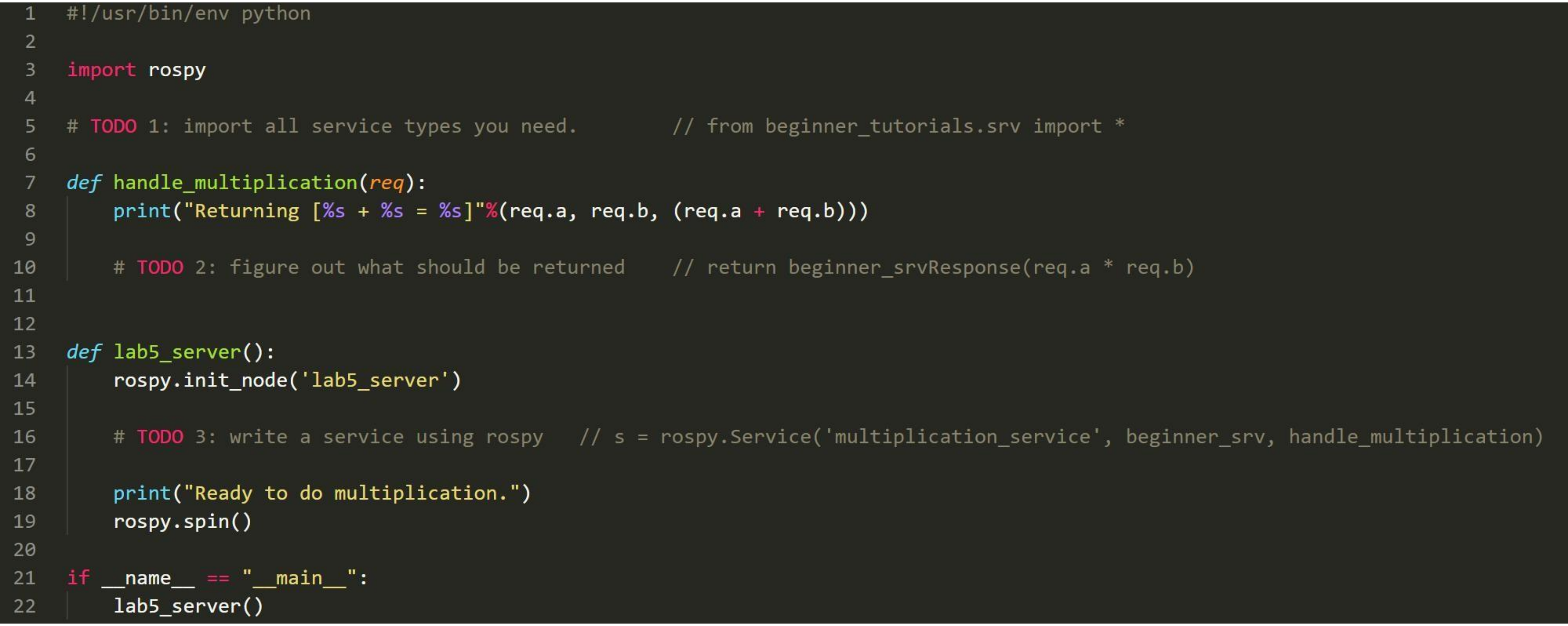
```
$ sudo chmod +x scripts/lab5_server.py scripts/lab5_client.py
```

- Build (Cmake) your node:

```
# in your catkin workspace
$ roscd beginner_tutorials
$ cd ~/catkin_ws
$ catkin_make
$ source ....
```

Task 3: Create a simple ROS Service and Client

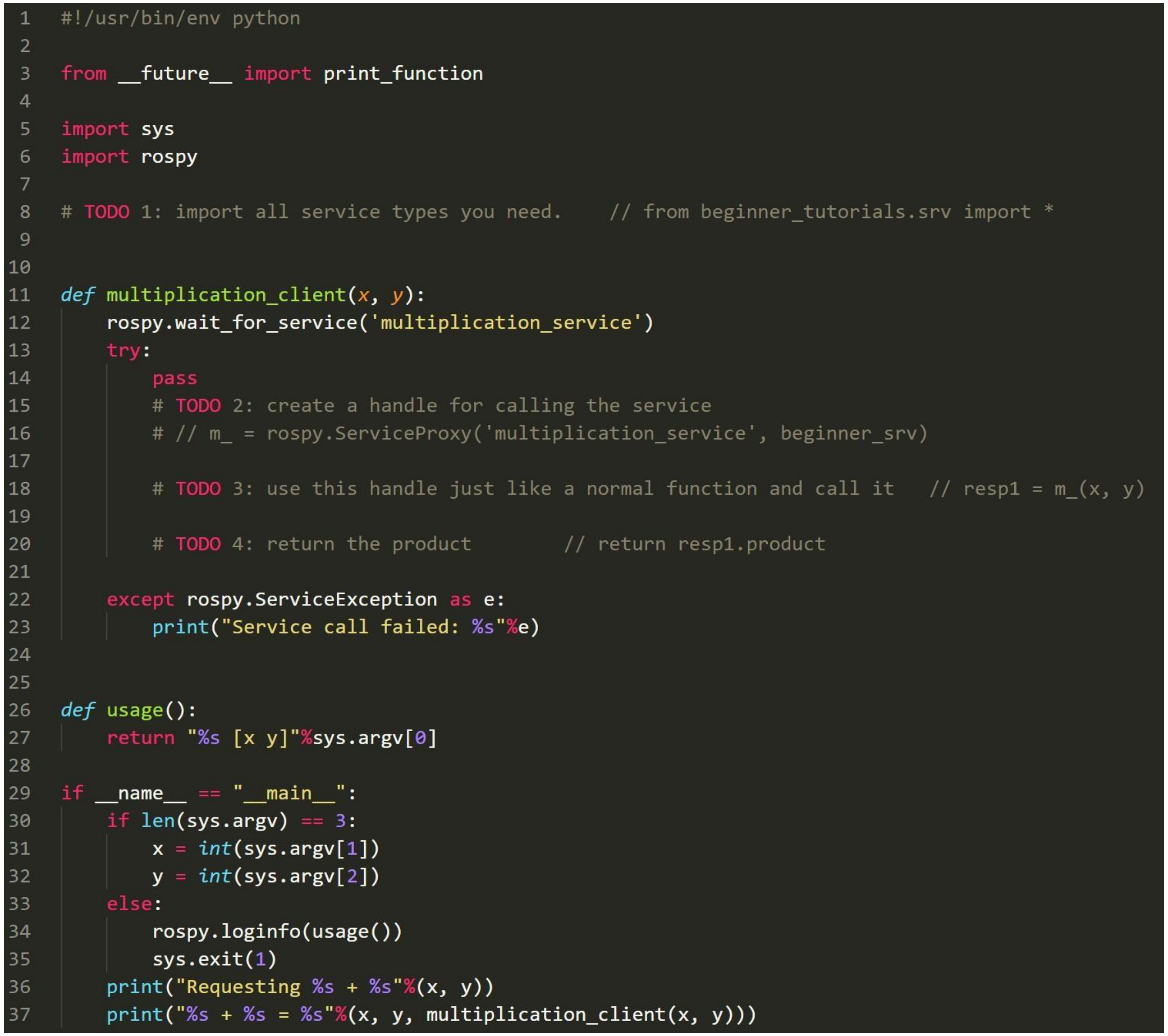
- lab5_server.py



```
#!/usr/bin/env python

import rospy

# TODO 1: import all service types you need.        // from beginner_tutorials.srv import *

def handle_multiplication(req):
    print("Returning [%s + %s = %s]"%(req.a, req.b, (req.a + req.b)))

    # TODO 2: figure out what should be returned    // return beginner_srvResponse(req.a * req.b)


def lab5_server():
    rospy.init_node('lab5_server')

    # TODO 3: write a service using rospy   // s = rospy.Service('multiplication_service', beginner_srv, handle_multiplication)

    print("Ready to do multiplication.")
    rospy.spin()

if __name__ == "__main__":
    lab5_server()
```

Task 3: Create a simple ROS Service and Client

- lab5_client.py



```python
#!/usr/bin/env python

from __future__ import print_function

import sys
import rospy

# TODO 1: import all service types you need.    // from beginner_tutorials.srv import *


def multiplication_client(x, y):
    rospy.wait_for_service('multiplication_service')
    try:
        pass
        # TODO 2: create a handle for calling the service
        # // m_ = rospy.ServiceProxy('multiplication_service', beginner_srv)

        # TODO 3: use this handle just like a normal function and call it   // resp1 = m_(x, y)

        # TODO 4: return the product        // return resp1.product

    except rospy.ServiceException as e:
        print("Service call failed: %s"%e)


def usage():
    return "%s [x y]"%sys.argv[0]

if __name__ == "__main__":
    if len(sys.argv) == 3:
        x = int(sys.argv[1])
        y = int(sys.argv[2])
    else:
        rospy.loginfo(usage())
        sys.exit(1)
    print("Requesting %s + %s"%(x, y))
    print("%s + %s = %s"%(x, y, multiplication_client(x, y)))
```

Task 3: Create a simple ROS Service and Client

Something you might need for your Python scripts:

Import the .srv:

```
from beginner_tutorials.srv import*
```

Create a service:

```
s = rospy.Service(<service_name>, <service_type>,
                  <function_for_handling_request>)
```

Create a handle for calling the service:

```
m_ = rospy.ServiceProxy(<service_name>,<service_type>)
```

Task 4: Using the launch file to launch created server and client

BONUS

```
$ roscd beginner_tutorials
$ cd launch
$ touch launch_server_client.launch
$ gedit launch_server_client.launch
```

Note: how to pass two float number to the ‘lab5_client’ in the launch file.