

# Lab sheet

## Mobile Robot Day 1

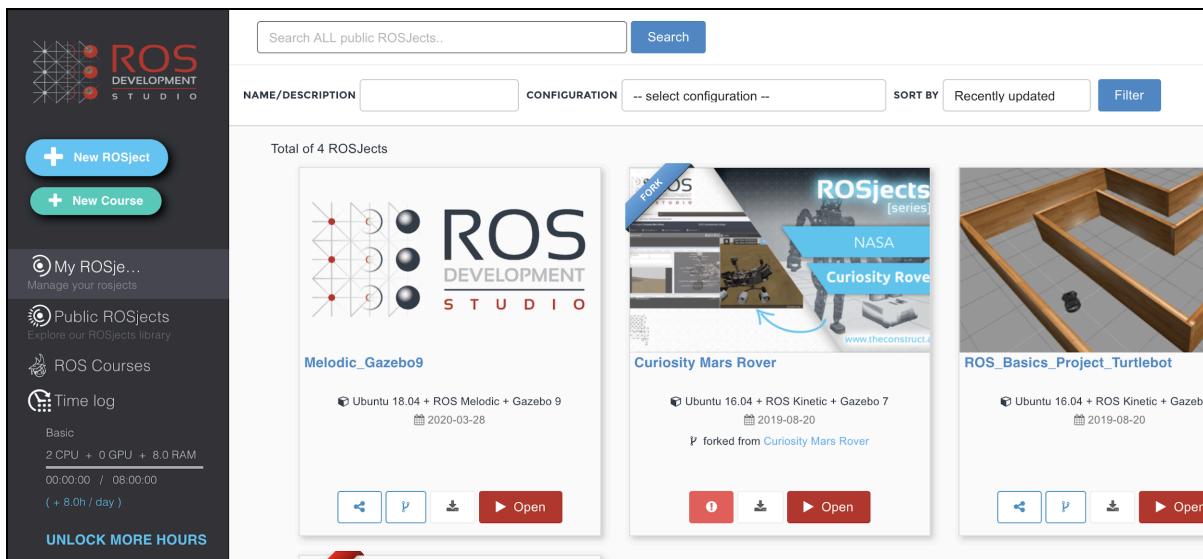
Launch RDS (ROS Development Studio)

Go to <http://rosds.online/>

Registration

[Video: How to register for the robot development studio](#)

My ROS Workspace



New ROSject

- Name
- ROS Configuration = **Ubuntu 18.04 + Melodic + Gazebo 9**
- No robot selected

## Create new ROSJect

Create new ROSJect

NAME

THUMBNAIL  No file chosen  
IMAGE (\*.PNG, \*.JPEG)

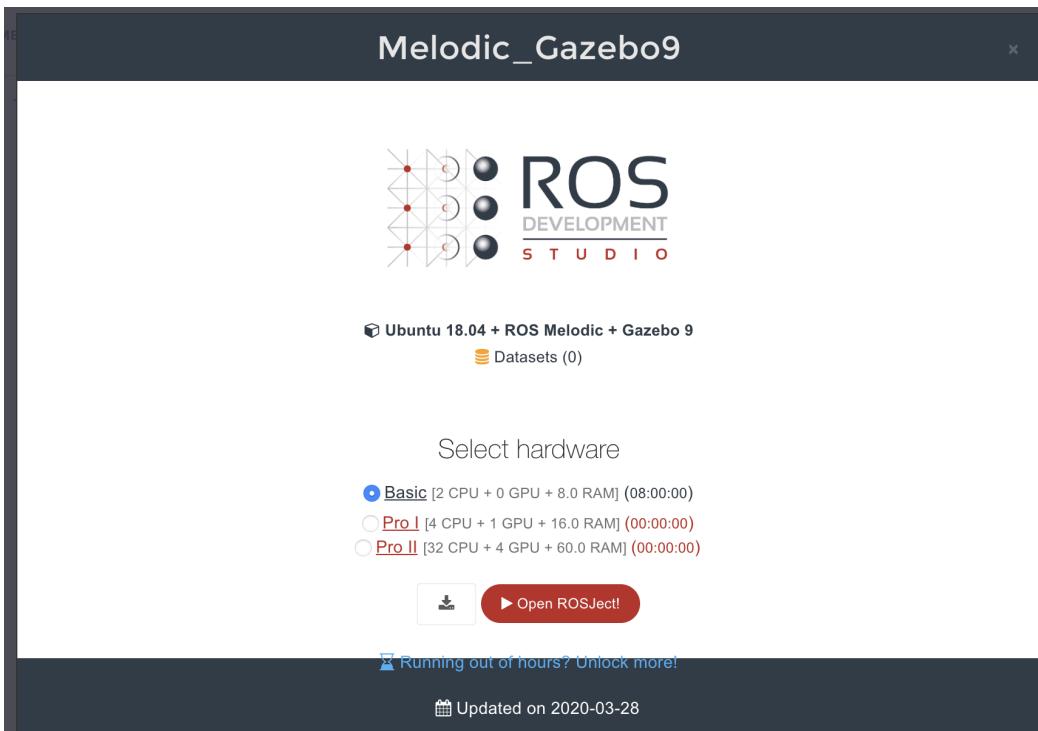
ROS CONFIGURATION

SELECT A ROBOT TO PROGRAM FOR

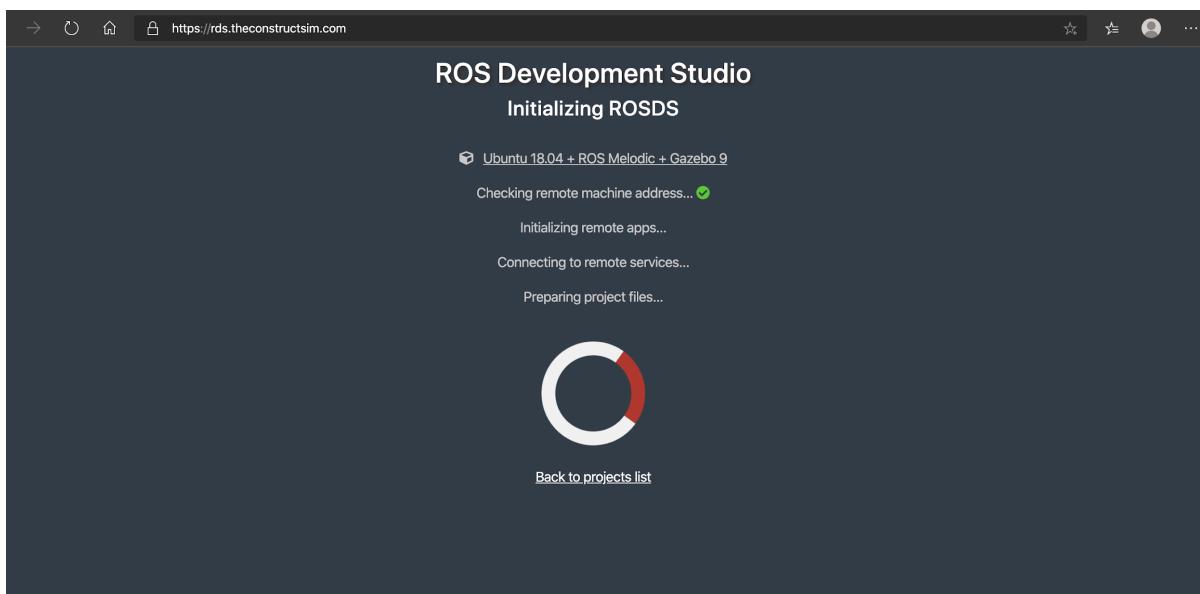
PRIVATE OR PUBLIC?

DESCRIPTION   
Description of the simulation

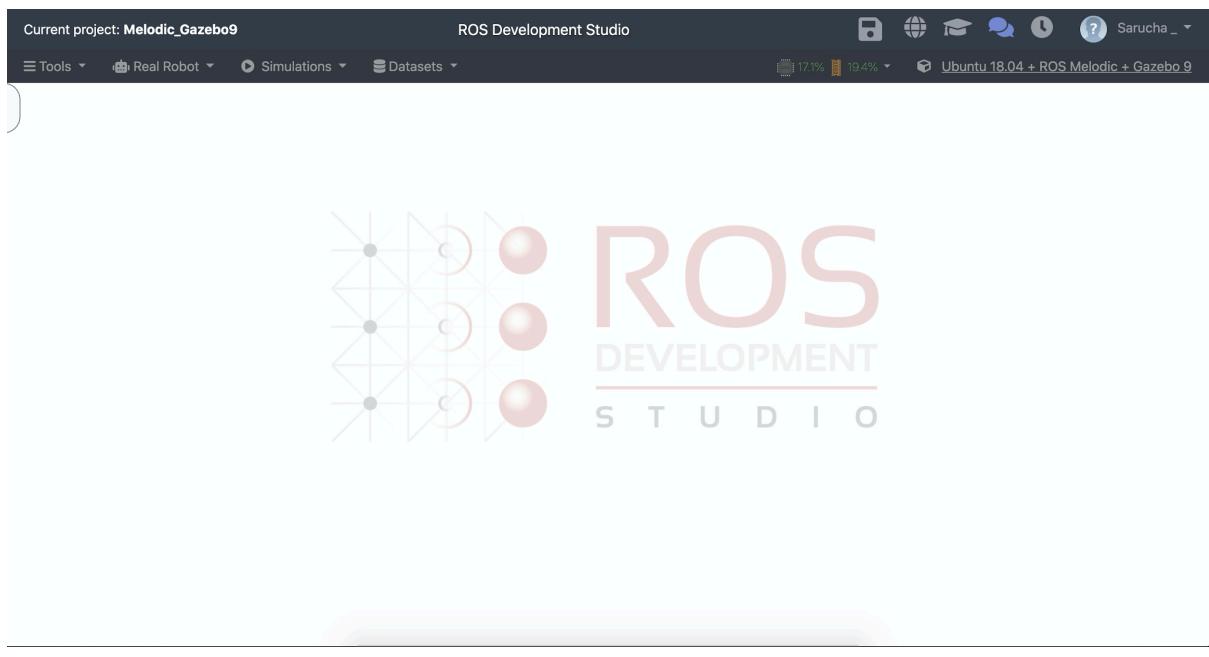
Open project



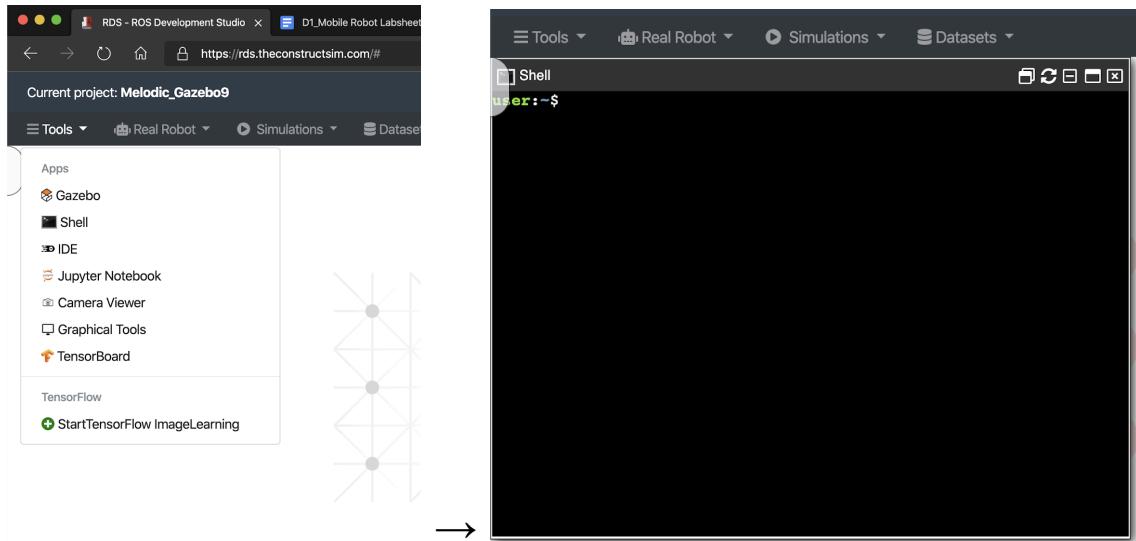
Wait for 3-5 minutes



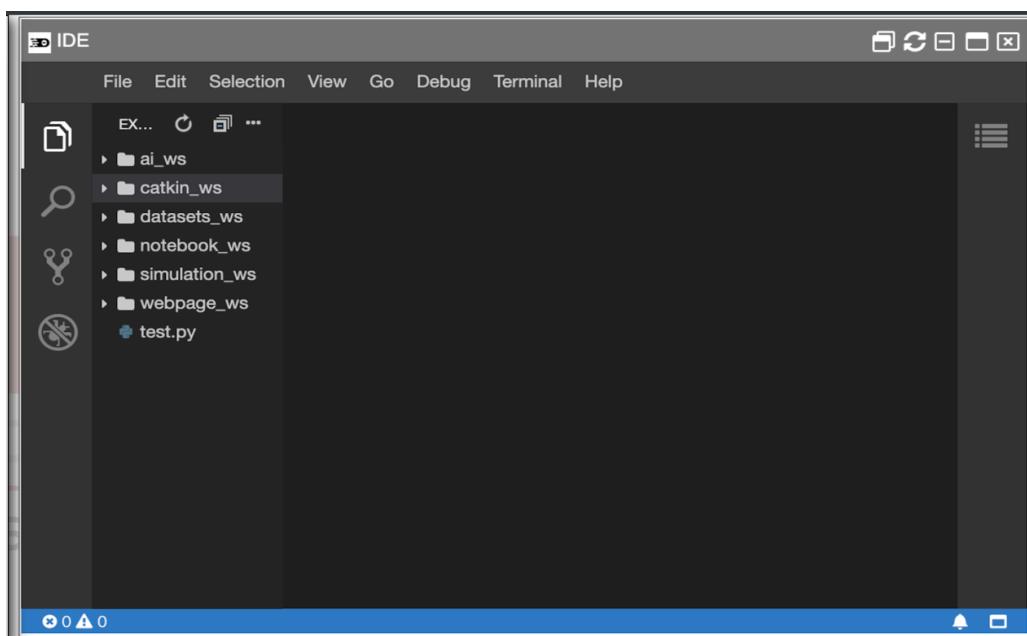




## Tools → Shell



## Tools → IDE



Tools → Graphical tools

## Command-line interface

Commands	Description
ls	Directory listing
ls -la	Directory listing and <b>show hidden files</b>
cd <b>dir</b>	Change current directory to <dir>
cd	Change current directory to home
pwd	Show current directory
mkdir <b>dir</b>	Create a directory <dir>
mkdir -p <b>dir/dir</b>	Create nested directory
rm <b>file</b>	Delete <file>
rm -r <b>dir</b>	Delete directory <dir>
rm -f <b>file</b>	Force remove <file>
cp <b>file1 file2</b>	Copy <b>file1</b> to <b>file2</b>
cp <b>file1 directory2</b>	Copy <b>file1</b> to <b>directory2</b>
mv <b>file1 file2</b>	Rename <b>file1</b> to <b>file2</b>
mv <b>file1 directory2</b>	Move <b>file1</b> to <b>directory2</b>
touch <b>file1</b>	Create empty <b>file1</b>
nano <b>file1</b>	Edit <b>file1</b>
gedit <b>file1</b>	Edit <b>file1</b>
cat <b>file1</b>	Read <b>file1</b>
chmod <b>777 file1</b>	Change permission 777 is read/write from all user  See this website for reference <a href="https://chmod-calculator.com/">https://chmod-calculator.com/</a>
chmod <b>+x file1</b>	Change permission to executable
sudo <b>command</b>	Execute <b>command</b> by Super User

## ROS Filesystem

```
catkin_ws
├── build
├── devel
│   ├── lib
│   ├── _setup_util.py
│   ├── env.sh
│   ├── setup.bash      → 用来设置 workspace
│   ├── setup.sh
│   └── setup.zsh
└── src
    ├── firstgazebo
    ├── my_turtlesim
    │   ├── script
    │   │   └── sub.py
    │   └── src
    │       ├── CMakeLists.txt
    │       └── package.xml
    └── mybot_gazebo
```

→ workspace → 环境 Command

→ devel = development

→ src = 项目 / 包装

→ rosdep command Create Package

→ 生成 Python 脚本

→ Python 脚本

## Create workspace

```
source /opt/ros/melodic/setup.bash  
mkdir -p ~/catkin_ws/src  
cd ~/catkin_ws/  
catkin_make
```

After that, you have to import the workspace environment to OS.

```
source devel/setup.bash
```

## Auto import workspace into OS Environment

1. Open Shell
2. Type

```
echo "source ~/catkin_ws/devel/setup.bash" >> ~/.bashrc
```

3. Close and reopen all Shell windows

## Create a package

The package of ROS must be kept in ~/catkin\_ws/src.

```
cd ~/catkin_ws/src
```

std\_msgs, rospy, and roscpp are the dependency packages that we need for our package.

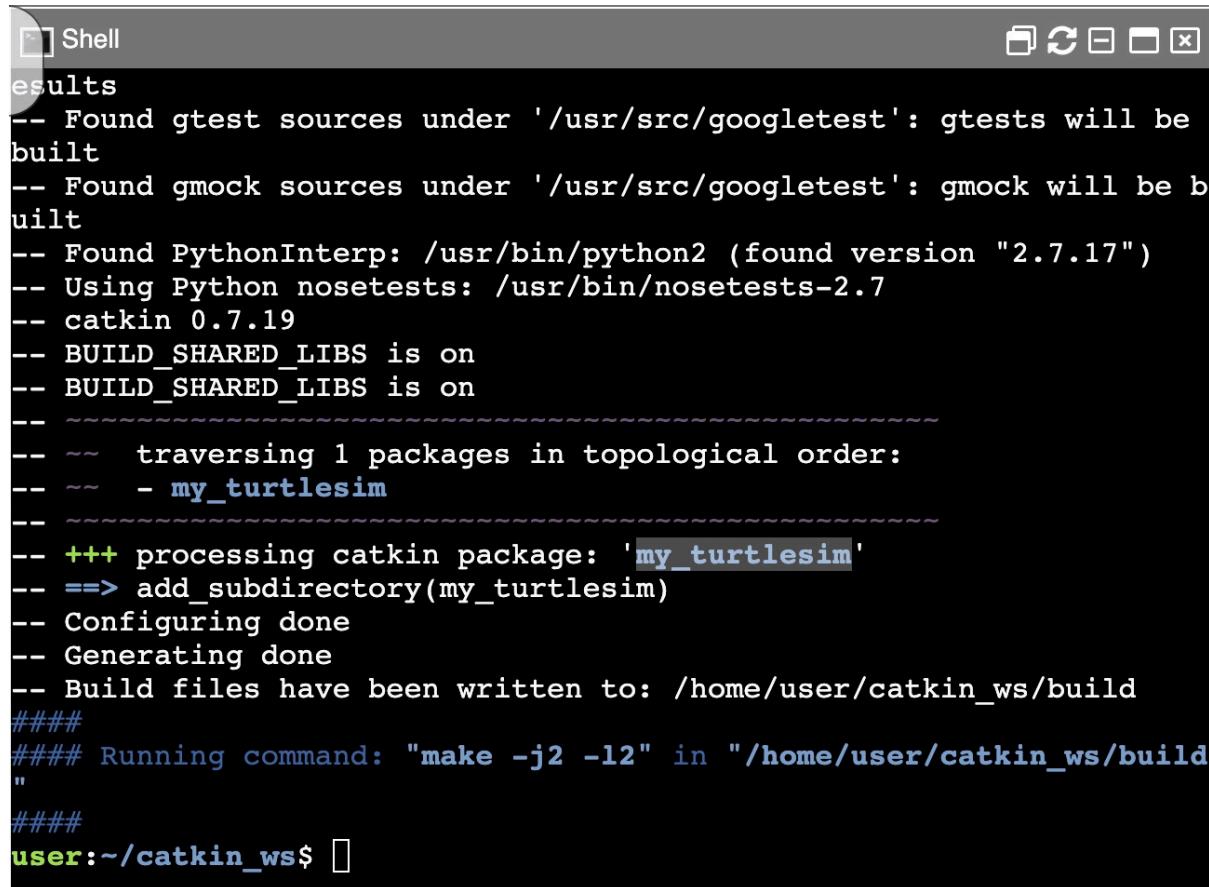
```
catkin_create_pkg my_turtlesim std_msgs rospy roscpp
```

## Build workspace

The workspace must be built after it was changed such as a new package and etc.

```
cd ~/catkin_ws  
catkin_make
```

## Output



A screenshot of a terminal window titled "Shell". The window shows the command "catkin\_make" being run. The output of the command is displayed, including messages about finding gtest and gmock sources, the Python interpreter version, and the catkin version. It also shows the traversal of packages, processing of the 'my\_turtlesim' package, and the final build command being run.

```
results  
-- Found gtest sources under '/usr/src/googletest': gtests will be built  
-- Found gmock sources under '/usr/src/googletest': gmock will be built  
-- Found PythonInterp: /usr/bin/python2 (found version "2.7.17")  
-- Using Python nosetests: /usr/bin/nosetests-2.7  
-- catkin 0.7.19  
-- BUILD_SHARED_LIBS is on  
-- BUILD_SHARED_LIBS is on  
--  
-- ~~ traversing 1 packages in topological order:  
-- ~~ - my_turtlesim  
--  
-- +++ processing catkin package: 'my_turtlesim'  
-- ==> add_subdirectory(my_turtlesim)  
-- Configuring done  
-- Generating done  
-- Build files have been written to: /home/user/catkin_ws/build  
####  
#### Running command: "make -j2 -l2" in "/home/user/catkin_ws/build"  
####  
user:~/catkin_ws$
```

## Create a python script

### Create folder script

```
CMakeLists.txt include package.xml src  
user:~/catkin_ws/src/my_turtlesim$ mkdir script
```

New empty python script

```
user:~/catkin_ws/src/my_turtlesim$ cd script/  
user:~/catkin_ws/src/my_turtlesim/script$ touch pub.py
```

Change mode to executable

```
user:~/catkin_ws/src/my_turtlesim/script$ chmod +x pub.py  
user:~/catkin_ws/src/my_turtlesim/script$
```

## File permission

<https://chmod-calculator.com/>

### File permission changing

```
user:~/catkin_ws/src/my_turtlesim/script$ touch pub.py  
user:~/catkin_ws/src/my_turtlesim/script$ ls -la  
total 8  
drwxr-xr-x 2 user user 4096 Jun 12 08:13 .  
drwxr-xr-x 5 user user 4096 Jun 12 07:56 ..  
-rw-r--r-- 1 user user 0 Jun 12 08:13 pub.py  
user:~/catkin_ws/src/my_turtlesim/script$ chmod +x pub.py  
user:~/catkin_ws/src/my_turtlesim/script$ ls -la  
total 8  
drwxr-xr-x 2 user user 4096 Jun 12 08:13 .  
drwxr-xr-x 5 user user 4096 Jun 12 07:56 ..  
-rwxr-xr-x 1 user user 0 Jun 12 08:13 pub.py  
user:~/catkin_ws/src/my_turtlesim/script$
```

## Python programming

Python programming language is used to program the robot. In this section, we will learn the basics of Python.

The python script can be created or edited by a graphical text editor.

## 1.1. Python Indentation

Most of the programming languages like C, C++, Java use braces {} to define a block of code. Python uses indentation.

A code block (body of a function, loop etc.) starts with indentation and ends with the first unindented line. The amount of indentation is up to you, but it must be consistent throughout that block.

Generally four whitespaces are used for indentation and is preferred over tabs. Here is an example.

```
for i in range(1,11):
    print(i)
    if i == 5:
        break
```

The enforcement of indentation in Python makes the code look neat and clean. This results into Python programs that look similar and consistent.

## 1.2. Python Variables

A variable is a named location used to store data in the memory. It is helpful to think of variables as a container that holds data which can be changed later throughout programming. For example,

```
1. number = 10
```

Here, we have created a named number. We have assigned value 10 to the variable.

You can think variable as a bag to store books in it and those books can be replaced at any time.

```
1. number = 10
2. number = 1.1
3. website = "apple.com"
4.
```

Initially, the value of `number` was 10. Later it's changed to 1.1.

Note: In Python, we don't assign values to the variables, whereas Python gives the reference of the object (value) to the variable.

### 1.3. Python Comments

Comments are very important while writing a program. It describes what's going on inside a program so that a person looking at the source code does not have a hard time figuring it out. You might forget the key details of the program you just wrote in a month's time. So taking time to explain these concepts in form of comments is always fruitful.

In Python, we use the hash (#) symbol to start writing a comment. It extends up to the newline character. Comments are for programmers for better understanding of a program. Python Interpreter ignores comment.

```
5. #This is a comment  
6. #print out Hello  
7. print('Hello')
```

#### Multi-line comments

If we have comments that extend multiple lines, one way of doing it is to use hash (#) in the beginning of each line. For example:

```
#This is a long comment  
#and it extends  
#to multiple lines
```

### 1.4. Python Output Using `print()` function

We use the `print()` function to output data to the standard output device (screen).

We can also [output data to a file](#), but this will be discussed later. An example use is given below.

```
print('This sentence is output to the screen')
# Output: This sentence is output to the screen

a = 5

print('The value of a is', a)
# Output: The value of a is 5
```

## 1.5. Python if Statement Syntax

Decision making is required when we want to execute a code only if a certain condition is satisfied. The if...elif...else statement is used in Python for decision making.

### Syntax of if statement

```
if test expression:
    statement(s)
```

Here, the program evaluates the test expression and will execute statement(s) only if the text expression is True.

If the text expression is False, the statement(s) is not executed.

In Python, the body of the if statement is indicated by the indentation. Body starts with an indentation and the first unindented line marks the end.

Python interprets non-zero values as True. None and 0 are interpreted as False.

### Python if Statement Flowchart

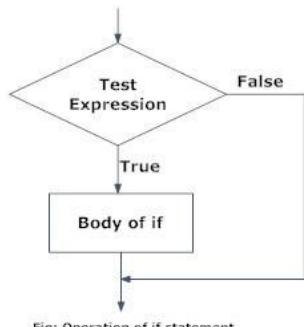


Fig: Operation of if statement.

## Example of if statement

# If the number is positive, we print an appropriate message

```
num = 3
if num > 0:
    print(num, "is a positive number.")
print("This is always printed.")
```

```
num = -1
if num > 0:
    print(num, "is a positive number.")
print("This is also always printed.")
```

## Output

```
3 is a positive number
This is always printed
This is also always printed.
```

In the above example, `num > 0` is the test expression.

The body of if is executed only if this evaluates to True.

When variable `num` is equal to 3, test expression is true and body inside body of if is executed.

If variable `num` is equal to -1, test expression is false and body inside body of if is skipped.

The `print()` statement falls outside of the if block (unindented). Hence, it is executed regardless of the test expression.

## 1.6. Python if...else Statement

### Syntax of if-else statement

```
if test expression:
    Body of if
else:
```

## Body of else

The if..else statement evaluates test expression and will execute body of if only when test condition is True.

If the condition is False, body of else is executed. Indentation is used to separate the blocks.

Python if..else Flowchart

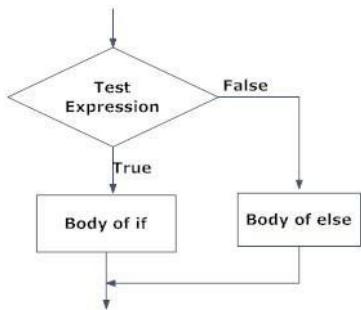


Fig: Operation of if...else statement

## Example of if-else statement

```
# Program checks if the number is positive or negative  
# And displays an appropriate message
```

```
num = 3
```

```
# Try these two variations as well.
```

```
# num = -5  
# num = 0
```

```
if num >= 0:  
    print("Positive or Zero")  
else:  
    print("Negative number")
```

## Output

Positive or Zero

In the above example, when num is equal to 3, the test expression is true and body of if is executed and body of else is skipped.

If num is equal to -5, the test expression is false and body of else is executed and body of if is skipped.

If num is equal to 0, the test expression is true and body of if is executed and body of else is skipped.

## 1.7. Python if...elif...else Statement

### Syntax of if-else statement

```
if test expression:
```

```
    Body of if
```

```
elif test expression:
```

```
    Body of elif
```

```
else:
```

```
    Body of else
```

The elif is short for else if. It allows us to check for multiple expressions.

If the condition for if is False, it checks the condition of the next elif block and so on.

If all the conditions are False, body of else is executed.

Only one block among the several if...elif...else blocks is executed according to the condition.

The if block can have only one else block. But it can have multiple elif blocks.

### Flowchart of if...elif...else

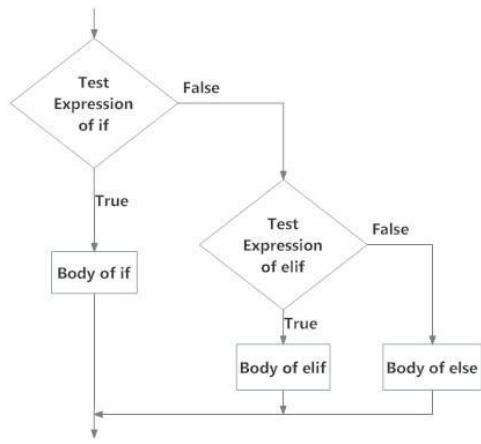


Fig: Operation of if...elif...else statement

### Example of if-elif-else statement

In this program,

```
# we check if the number is positive or  
# negative or zero and  
# display an appropriate message
```

```
num = 3.4
```

```
# Try these two variations as well:
```

```
# num = 0  
# num = -4.5
```

```
if num > 0:  
    print("Positive number")  
elif num == 0:  
    print("Zero")  
else:  
    print("Negative number")
```

### Output

Positive number

When variable num is positive, Positive number is printed.

If num is equal to 0, Zero is printed.

If num is negative, Negative number is printed

## 1.8. Python Nested if statements

We can have a if...elif...else statement inside another if...elif...else statement. This is called nesting in computer programming.

Any number of these statements can be nested inside one another. Indentation is the only way to figure out the level of nesting. This can get confusing, so must be avoided if we can.

#### Example of nested if statement

```
# In this program, we input a number
# check if the number is positive or
# negative or zero and display
# an appropriate message
# This time we use nested if
num = float(input("Enter a number: "))
if num >= 0:
    if num == 0:
        print("Zero")
    else:
        print("Positive number")
else:
    print("Negative number")
```

#### Output 1

Enter a number: 5

Positive number

#### Output 2

Enter a number: -1

Negative number

#### Output 3

Enter a number: 0

Zero

## 1.9. Python for loop

In this article, you'll learn to iterate over a sequence of elements using the different variations of for loop.

The for loop in Python is used to iterate over a sequence ([list](#), [tuple](#), [string](#)) or other iterable objects. Iterating over a sequence is called traversal.

### Syntax of for loop

```
for val in sequence:
```

```
    Body of for
```

Here, val is the variable that takes the value of the item inside the sequence on each iteration.

Loop continues until we reach the last item in the sequence. The body of for loop is separated from the rest of the code using indentation.

### Flowchart of for Loop

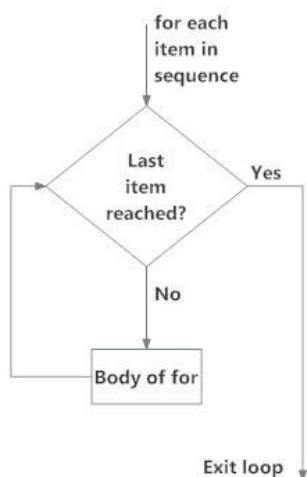


Fig: operation of for loop

## 1.10. Python while Loop

The while loop in Python is used to iterate over a block of code as long as the test expression (condition) is true.

We generally use this loop when we don't know beforehand, the number of times to iterate.

## Syntax of while loop

```
while test_expression:  
    Body of while
```

## Flowchart of while Loop

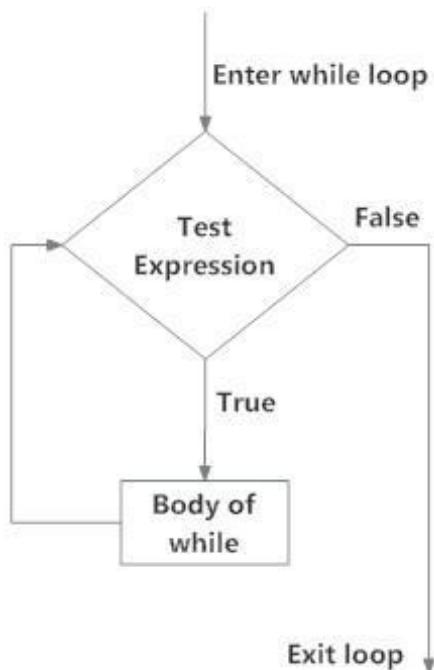


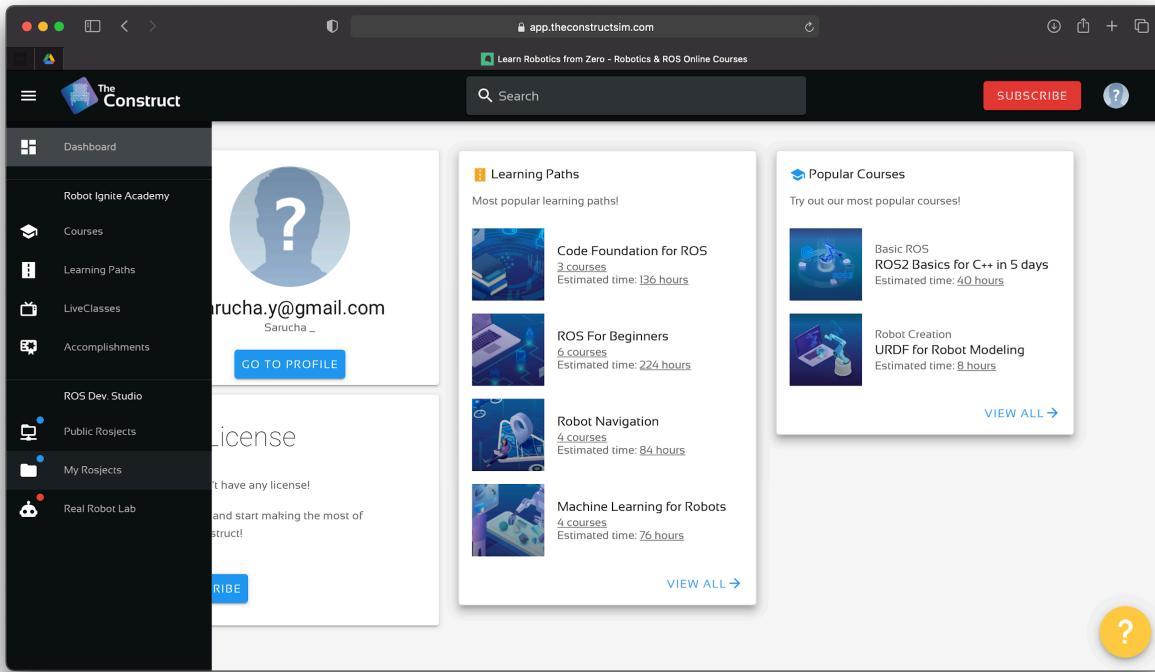
Fig: operation of while loop





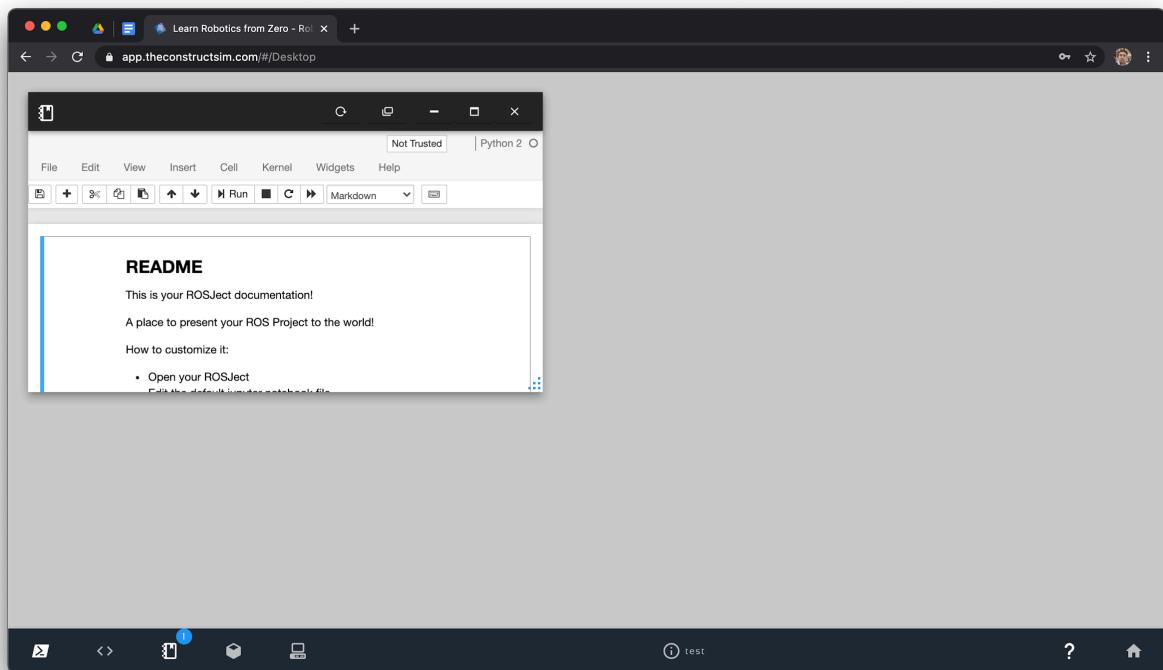
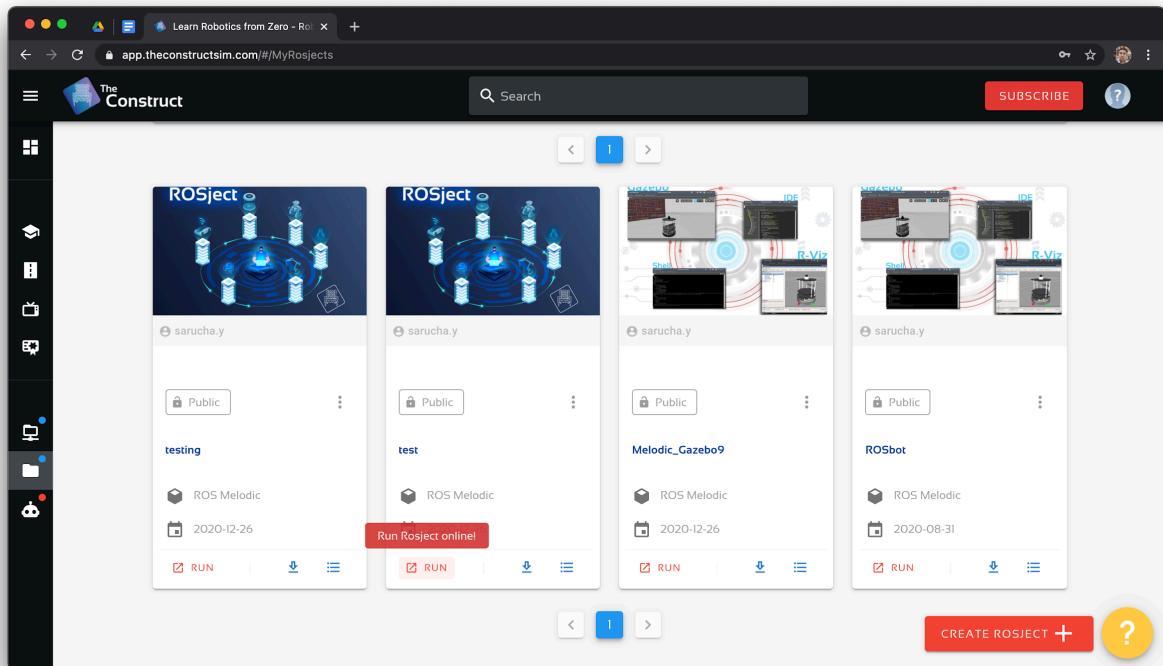
## APPENDIX: New ROS Development Studio

URL <https://app.theconstructsim.com/>

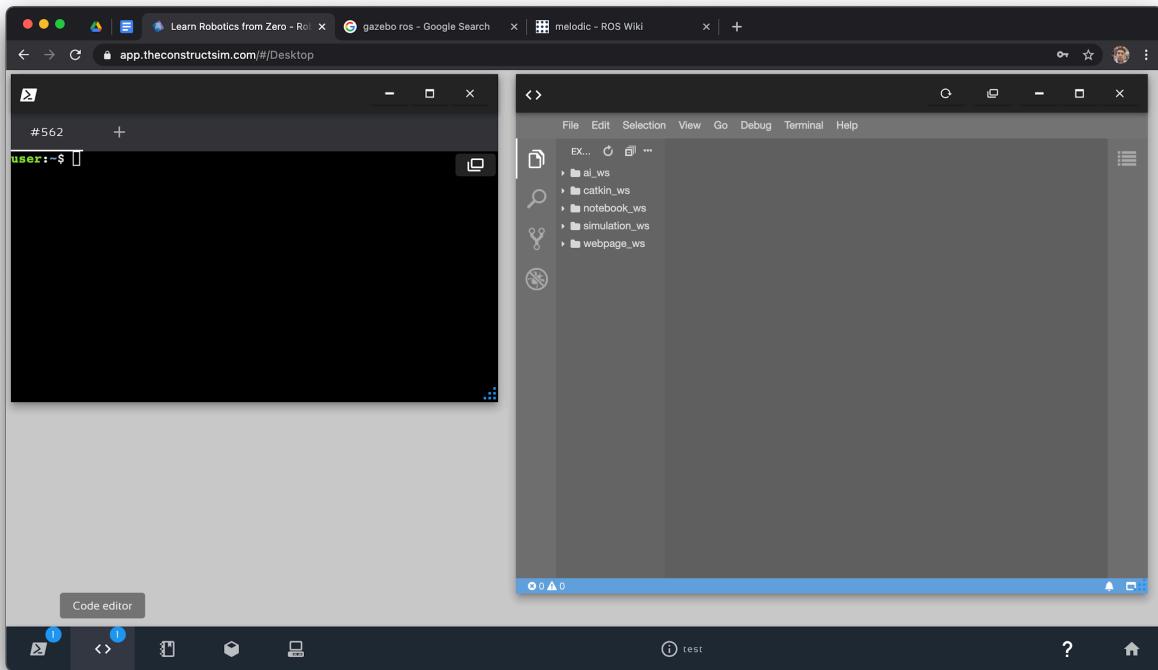


The screenshot shows the 'My projects' page of TheConstruct. At the top, there is a blue banner with the text 'Test the beta version of ROSDS'. Below the banner is a search bar with fields for 'Name/Description', 'ROS Distro', and 'Ordering', followed by a 'SEARCH' button. A sidebar on the left contains various icons for different project types. In the center, there are three project cards. Each card has a thumbnail image, the name 'sarucha.y', and a 'CREATE PROJECT +' button. The first card is titled 'ROSject' and features a circular diagram with various nodes. The second and third cards show a terminal window with code and a visualization window labeled 'R-Viz'. At the bottom left is a URL: <https://app.theconstructsim.com/#/RosjectCreate>. At the bottom right is a yellow circular icon with a question mark.

The screenshot shows the 'Create new project' dialog. The title is 'Create new project'. It includes a dropdown for 'ROS Distro' set to 'ROS Melodic'. The 'Name' field contains 'testing'. There is a checkbox for 'Make it private?' which is unchecked. The 'Description' field contains 'testing'. At the bottom, there is a checkbox for 'Are you creating a course for the Academy?' which is unchecked. A green 'CREATE' button is at the bottom left, and a yellow circular icon with a question mark is at the bottom right.



ໂຫລດ ROSject ເຮືຍບ້ອຍແລ້ວຈະໄດ້ໜ້າຕ່າງນີ້ຄັບ



ໜ້າຈາກຄູກ Web Shell ແລະ Code Editor ຈະໄດ້ກາພນີ້ຄັບ

The screenshot shows a web-based terminal interface. On the left is a terminal window with a black background and white text. It displays the command `ls` and its output, which includes directory names like `ai_ws`, `catkin_ws`, `notebook_ws`, `simulation_ws`, and `webpage_ws`. On the right is a file explorer window with a dark theme, showing the same directory structure. The top of the interface has a navigation bar with tabs for "Learn Robotics from Zero - Ros", "gazebo ros - Google Search", and "melodic - ROS Wiki". The bottom of the interface has a toolbar with icons for file operations and a status bar showing "test".

គណន៍មាន ៥

The screenshot shows a web-based terminal interface. On the left is a terminal window with a black background and white text. It displays the following command-line session:

```
#562
user:~$ ls
ai_ws  catkin_ws  notebook_ws  simulation_ws  webpage_ws
user:~$ pwd
/home/user
user:~$ cd catkin_ws/
user:~/catkin_ws$
```

On the right is a file explorer sidebar with a dark theme. It lists several workspace directories:

- al\_ws
- catkin\_ws
- notebook\_ws
- simulation\_ws
- webpage\_ws

គណនីការណា `cd`

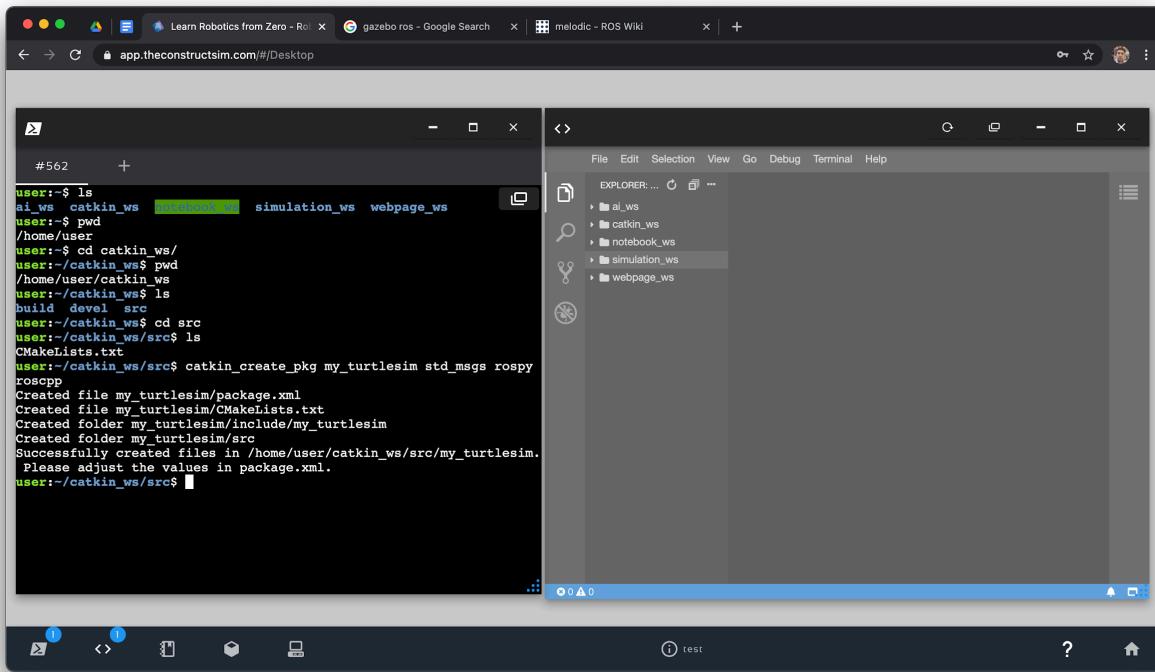
The screenshot shows a web-based interface for a terminal and file explorer. The terminal window on the left displays a command-line session:

```
#562
user:~$ ls
ai_ws  catkin_ws  notebook_ws  simulation_ws  webpage_ws
user:~$ pwd
/home/user
user:~$ cd catkin_ws/
user:~/catkin_ws$ pwd
/home/user/catkin_ws
user:~/catkin_ws$ ls
build  devel  src
user:~/catkin_ws$ cd src
user:~/catkin_ws/src$ ls
CMakeLists.txt
user:~/catkin_ws/src$
```

The file explorer window on the right shows the directory structure:

- al\_ws
- catkin\_ws
- notebook\_ws
- simulation\_ws
- webpage\_ws

ตำแหน่งปัจจุบันก่อนสร้าง Package



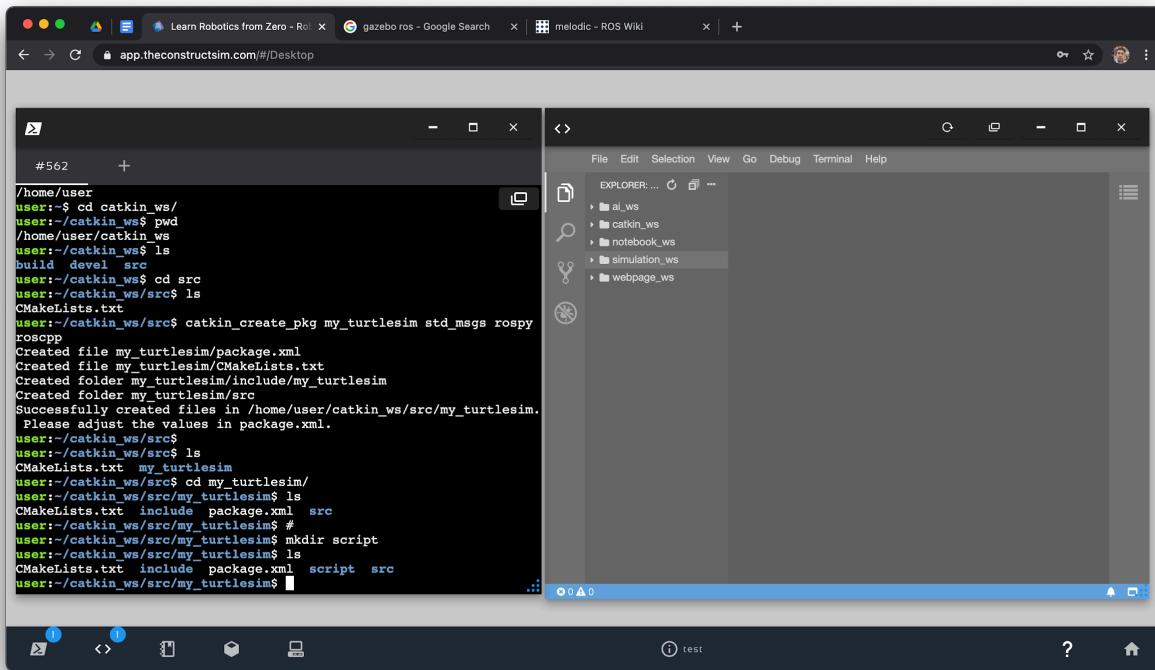
The screenshot shows a terminal window and a file explorer side-by-side. The terminal window displays the following command-line session:

```
#562
user:~$ ls
ai_ws catkin_ws  notebook_ws  simulation_ws  webpage_ws
user:~$ pwd
/home/user
user:~$ cd catkin_ws/
user:~/catkin_ws$ pwd
/home/user/catkin_ws
user:~/catkin_ws$ ls
build  devel  src
user:~/catkin_ws$ cd src
user:~/catkin_ws/src$ ls
CMakeLists.txt
user:~/catkin_ws/src$ catkin_create_pkg my_turtlesim std_msgs rospy
roscpp
Created file my_turtlesim/package.xml
Created file my_turtlesim/CMakeLists.txt
Created folder my_turtlesim/include/my_turtlesim
Created folder my_turtlesim/src
Successfully created files in /home/user/catkin_ws/src/my_turtlesim.
Please adjust the values in package.xml.
user:~/catkin_ws/src$
```

The file explorer on the right shows the directory structure:

- ai\_ws
- catkin\_ws
- notebook\_ws
- simulation\_ws
- webpage\_ws

ภาพหลังสร้าง Package ชื่อ my\_turtlesim



The screenshot shows a terminal window with a dark theme running on a Mac OS X desktop. The terminal is displaying a series of commands for creating a ROS package named 'my\_turtlesim'. The commands include navigating to the workspace, creating a new directory, generating CMakeLists files, and creating a ROS package structure. The terminal window is titled '#562' and has tabs for 'Learn Robotics from Zero - Ro' and 'gazebo ros - Google Search'. To the right of the terminal is a code editor window with an 'EXPLORER' sidebar showing project folders like 'al\_ws', 'catkin\_ws', 'notebook\_ws', 'simulation\_ws', and 'webpage\_ws'.

```
#562
/home/user
user:~ cd catkin_ws/
user:~/catkin_ws$ pwd
/home/user/catkin_ws
user:~/catkin_ws$ ls
build devel src
user:~/catkin_ws$ cd src
user:~/catkin_ws/src$ ls
CMakeLists.txt
user:~/catkin_ws/src$ catkin_create_pkg my_turtlesim std_msgs rospy
roscpp
Created file my_turtlesim/package.xml
Created file my_turtlesim/CMakeLists.txt
Created folder my_turtlesim/include/my_turtlesim
Created folder my_turtlesim/src
Successfully created files in /home/user/catkin_ws/src/my_turtlesim.
Please adjust the values in package.xml.
user:~/catkin_ws/src$ ls
CMakeLists.txt my_turtlesim
user:~/catkin_ws/src$ cd my_turtlesim/
user:~/catkin_ws/src/my_turtlesim$ ls
CMakeLists.txt include package.xml src
user:~/catkin_ws/src/my_turtlesim$ #
user:~/catkin_ws/src/my_turtlesim$ mkdir script
user:~/catkin_ws/src/my_turtlesim$ ls
CMakeLists.txt include package.xml script src
user:~/catkin_ws/src/my_turtlesim$
```

หลังพิมพ์ script

The screenshot shows a terminal window on the left and a code editor window on the right. The terminal window displays a series of commands being run in a Catkin workspace, specifically creating a package named 'my\_turtlesim' with std\_msgs and rospy dependencies. The code editor window shows the file system structure of the workspace, with the 'script' folder highlighted.

```
#562
build devel src
user:~/catkin_ws$ cd src
user:~/catkin_ws/src$ ls
CMakeLists.txt
user:~/catkin_ws/src$ catkin_create_pkg my_turtlesim std_msgs rospy
roscpp
Created file my_turtlesim/package.xml
Created file my_turtlesim/CMakeLists.txt
Created folder my_turtlesim/include/my_turtlesim
Created folder my_turtlesim/src
Successfully created files in /home/user/catkin_ws/src/my_turtlesim.
Please adjust the values in package.xml.
user:~/catkin_ws/src$ user:~/catkin_ws/src$ ls
CMakeLists.txt my_turtlesim
user:~/catkin_ws/src$ cd my_turtlesim/
user:~/catkin_ws/src/my_turtlesim$ ls
CMakeLists.txt include package.xml src
user:~/catkin_ws/src/my_turtlesim$ #
user:~/catkin_ws/src/my_turtlesim$ mkdir script
user:~/catkin_ws/src/my_turtlesim$ ls
CMakeLists.txt include package.xml script src
user:~/catkin_ws/src/my_turtlesim$ user:~/catkin_ws/src/my_turtlesim$ ls
CMakeLists.txt include package.xml script src
user:~/catkin_ws/src/my_turtlesim$ cd script/
user:~/catkin_ws/src/my_turtlesim/script$ ls
user:~/catkin_ws/src/my_turtlesim/script$
```

เปิดไฟล์จาก Code Editor

The screenshot shows a terminal window and a file explorer interface. The terminal output is as follows:

```
#562
build devel src
user:~/catkin_ws$ cd src
user:~/catkin_ws/src$ ls
CMakeLists.txt
user:~/catkin_ws/src$ catkin_create_pkg my_turtlesim std_msgs rospy
roscpp
Created file my_turtlesim/package.xml
Created file my_turtlesim/CMakeLists.txt
Created folder my_turtlesim/include/my_turtlesim
Created folder my_turtlesim/src
Successfully created files in /home/user/catkin_ws/src/my_turtlesim.
Please adjust the values in package.xml.
user:~/catkin_ws/src$ user:~/catkin_ws/src$ ls
CMakeLists.txt my_turtlesim
user:~/catkin_ws/src$ cd my_turtlesim/
user:~/catkin_ws/src/my_turtlesim$ ls
CMakeLists.txt include package.xml src
user:~/catkin_ws/src/my_turtlesim$ #
user:~/catkin_ws/src/my_turtlesim$ mkdir script
user:~/catkin_ws/src/my_turtlesim$ ls
CMakeLists.txt include package.xml script src
user:~/catkin_ws/src/my_turtlesim$ user:~/catkin_ws/src/my_turtlesim$ ls
CMakeLists.txt include package.xml script src
user:~/catkin_ws/src/my_turtlesim$ cd script/
user:~/catkin_ws/src/my_turtlesim/script$ ls
user:~/catkin_ws/src/my_turtlesim/script$ []
```

The file explorer sidebar shows the directory structure:

- al\_ws
- catkin\_ws
  - build
  - devel
  - src
    - my\_turtlesim
      - script
    - include
    - CMakeLists.txt
    - package.xml
  - notebook\_ws
  - simulation\_ws
  - webpage\_ws

A context menu is open over the 'script' folder in the 'my\_turtlesim' directory, with the 'New File' option highlighted. Other options in the menu include Open, Open in Terminal, Select for Compare, Find in Folder, Copy, Paste, Copy Download Link, Upload Files..., Download, Delete, Duplicate, Rename, and Generate .editorconfig.

The screenshot shows a terminal window on the left and a code editor window on the right.

**Terminal Output:**

```
#562
Created file my_turtlesim/package.xml
Created file my_turtlesim/CMakeLists.txt
Created folder my_turtlesim/include/my_turtlesim
Created folder my_turtlesim/src
Successfully created files in /home/user/catkin_ws/src/my_turtlesim.
Please adjust the values in package.xml.
user:~/catkin_ws/src$ 
user:~/catkin_ws/src$ ls
CMakeLists.txt  my_turtlesim
user:~/catkin_ws/src$ cd my_turtlesim/
user:~/catkin_ws/src/my_turtlesim$ ls
CMakeLists.txt  include  package.xml  src
user:~/catkin_ws/src/my_turtlesim$ #
user:~/catkin_ws/src/my_turtlesim$ mkdir script
user:~/catkin_ws/src/my_turtlesim$ ls
CMakeLists.txt  include  package.xml  script  src
user:~/catkin_ws/src/my_turtlesim$ user:~/catkin_ws/src/my_turtlesim$ ls
CMakeLists.txt  include  package.xml  script  src
user:~/catkin_ws/src/my_turtlesim$ cd script/
user:~/catkin_ws/src/my_turtlesim/script$ ls
user:~/catkin_ws/src/my_turtlesim/script$ 
user:~/catkin_ws/src/my_turtlesim/script$ ls
pub.py
user:~/catkin_ws/src/my_turtlesim/script$ chmod +x pub.py
user:~/catkin_ws/src/my_turtlesim/script$ ls
pub.py
user:~/catkin_ws/src/my_turtlesim/script$
```

**Code Editor:**

The code editor shows a file named `pub.py` with the following content:

```
1
```

The code editor interface includes tabs for File, Edit, Selection, View, Go, Debug, Terminal, Help, and a search bar. The status bar at the bottom indicates "Ln 1, Col 1 LF UTF-8 Spaces: 4 Python".

# Lesson learned

```
Shell
user:~$ echo "source ~/catkin_ws/devel/setup.bash"
source ~/catkin_ws/devel/setup.bash
user:~$ tail ~/.bashrc
fi
fi
#
# -----
# If we from The Construct disar
# gain.
# -----
# -----
if [[ "${tc_exit_on_error}" = "true" ]]; then
    unset tc_exit_on_error
# -----
# -----
source ~/catkin_ws/devel/setup.bash
user:~$ roscore
Traceback (most recent call last):
  File "/opt/ros/melodic/bin/roscore", line 1, in <module>
    from rosmaster.master_api import RosmasterMasterAPI
ImportError: No module named rosmaster
user:~$ 
```

INF

Edit Select

```
Shell
user:~$ rosnode
Contact master at [master:11311]. Retrying...
rosnode
^C
user:~$ rosnode
rosnode is a command-line tool for printing information about ROS Nodes.

Commands:
  rosnode ping      test connectivity to a node
  rosnode list       list active nodes
  rosnode info       print information about a node
  rosnode machine list nodes running on a machine
  rosnode list machines
  rosnode kill       kill a running node
  rosnode cleanup    purge registration information
Type rosnode <command> -h for more detailed usage information
  e ping -h'

user:~$ rosnode list
ERROR: Unable to communicate with master!
user:~$ 
```

- ## 1. ERROR: Unable to communicate with master!

### Solution:

1. roscore must be running.

2. No module named rosmaster.master\_api

**Solution:**

- #### 1. Add the workspace to OS Environment variable by

```
source ~/catkin_ws/devel/setup.bash
```

3.