Summer School Robotics

Week 2 – Motion Planning in practicle:

Motion Planning Programming in ROS using Python

Haoming Zhang, M.Sc.
Philipp Ennen, M.Sc.

IfU IMA ZLW RWTH AACHEN UNIVERSITY

Outline

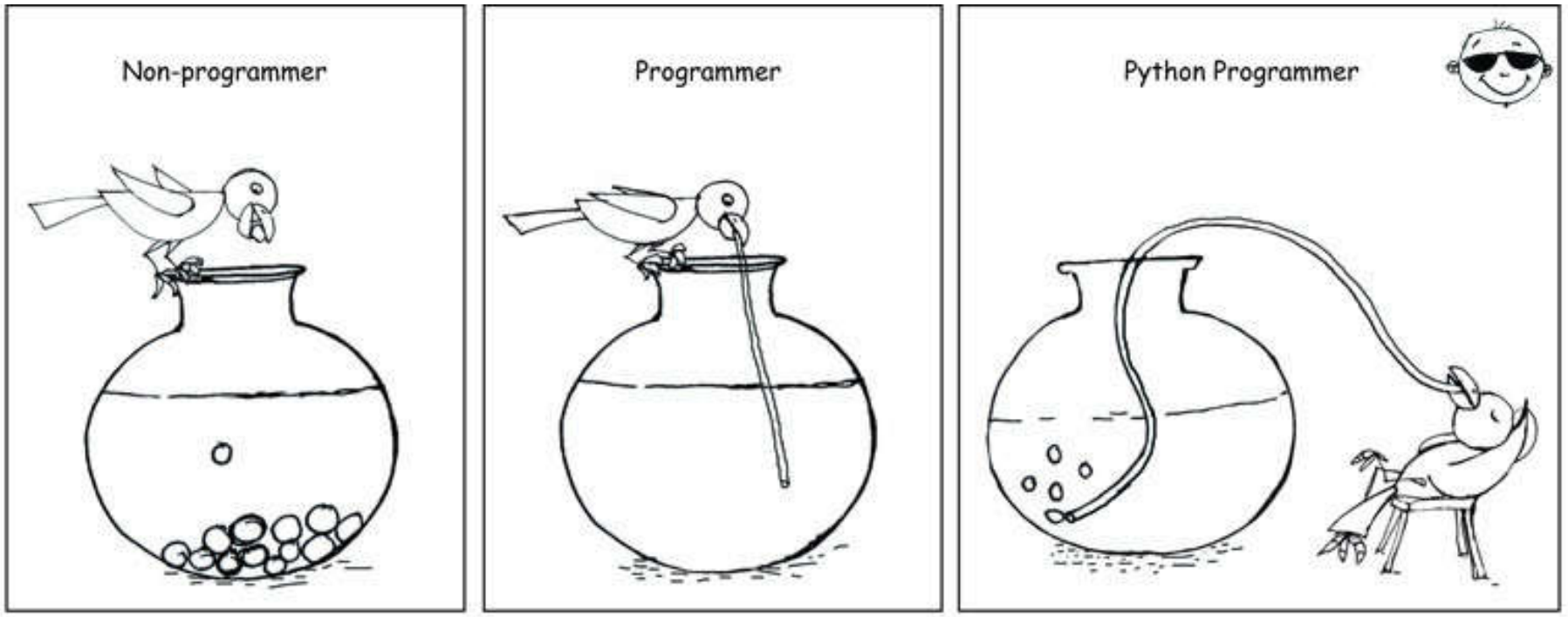
1	Programming using Python	14:00 – 15:25
1.1	The Python Programming Language	14:00 – 14:05
1.2	Python Basic	14:05 – 14:55
1.3	Practicle Unit: Your ROS Code in Python	14:55 – 15:25
2	Programming your Motion Planning for Kinova	15:25 – 17:00

Outline

1	Programming using Python	14:00 – 15:25
1.1	The Python Programming Language	14:00 – 14:05
1.2	Python Basic	14:05 – 14:55
1.3	Practicle Unit: Your ROS Code in Python	14:55 – 15:25
2	Programming your Motion Planning for Kinova	15:25 – 17:00

The Python Programming Language

How does a Crow get Water



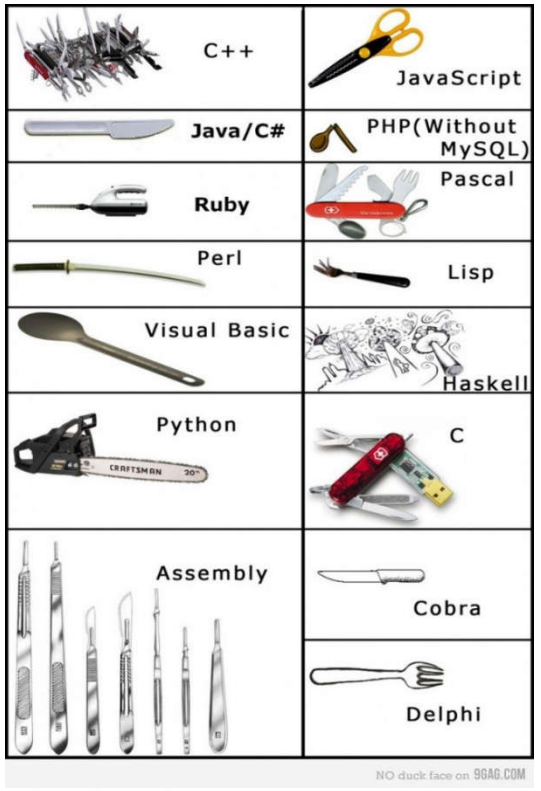
[Source: http://lpycot.appspot.com/]

The Python Programming Language

Introduction

- Created by **Guido van Rossum** in the early 1990s
- Dervied from many other languages: ABC, Modula-3, C, C++, Algol-68, SmallTalk, Unix shell and other **scripting** languages.
- Available unter GNU General Public License.
- Features:
  - Easy-to-learn
  - Easy-to-read
  - Easy-to-maintain
  - A broad standard library
  - Interactive Mode
  - Portable
  - Extendable
  - Databases
  - Graphic User Interface (GUI) Programming
  - Scalable
- Usage:
  - Data Analysis
  - Machine Learning
  - Computer Vision
  - IoT
  - Game Development
  - Web Development
  - GUI Development
  - Rapid Prototyping

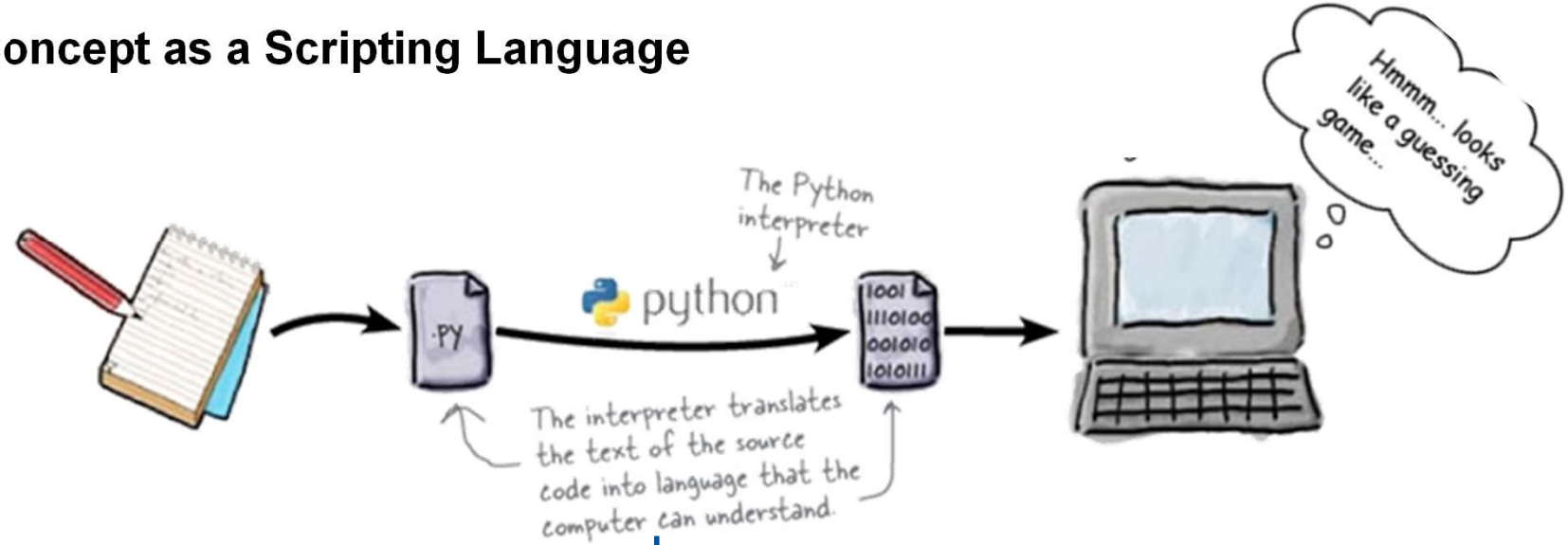
Guido van Rossum

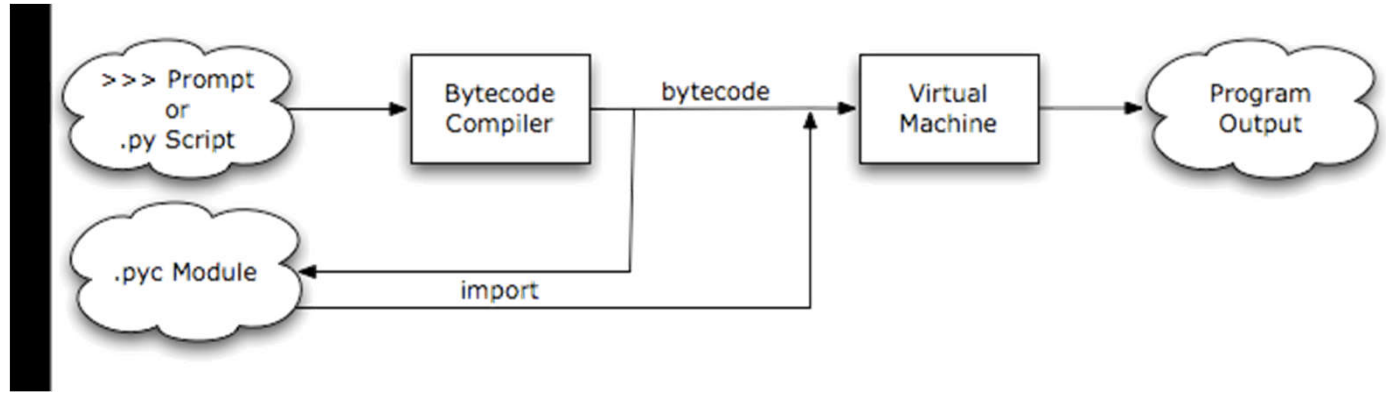


Outline

1	Programming using Python	14:00 – 15:25
1.1	The Python Programming Language	14:00 – 14:05
1.2	Python Basic	14:05 – 14:55
a	Variable and Data Structures	14:05 – 14:15
b	Logic, Condition and Loops	14:15 – 14:25
c	Functions and Object oriented Programming (OOP)	14:25 – 14:45
d	A whole Python Script	14:45 – 14:55
1.3	Practicle Unit: Your ROS Code in Python	14:55 – 15:25
2	Programming your Motion Planning for Kinova	15:25 – 17:00

Concept as a Scripting Language





Outline

1	Programming using Python	14:00 – 15:25
1.1	The Python Programming Language	14:00 – 14:05
1.2	Python Basic	14:05 – 14:55
a	Variable and Data Structures	14:05 – 14:15
b	Logic, Condition and Loops	14:15 – 14:25
c	Functions and Object oriented Programming (OOP)	14:25 – 14:45
d	A whole Python Script	14:45 – 14:55
1.3	Practicle Unit: Your ROS Code in Python	14:55 – 15:25
2	Programming your Motion Planning for Kinova	15:25 – 17:00

Python Basic

Variable and Data Structures:

- Numbers and String:
  - `int, long, float, complex`
  - `string`
  - `Multiple Assignment`
- Dynamically Typed:

  Variable Typ can be changed dynamically.
- Strong Typed:

  Enforce the Variables after it figures them out.
- Naming Rules:
  - Case sensitive
  - Contains letters, numbers, underscores
  - But can‘t start with numbers
  - Can‘t contain reserved Words

int	long	float	complex
10	51924361L	0.0	3.14j
100	-0x19323L	15.20	45.j
-786	0122L	-21.9	9.322e-36j
080	0xDEFABCECBDAECBFBAEl	32.3+e18	.876j
-0490	535633629843L	-90.	-.6545+0J
-0x260	-052318172735L	-32.54e100	3e+26J
0x69	-4721885298529L	70.2-E12	4.53e-7j

```
str = 'Hello World!'

print str          # Prints complete string
print str[0]       # Prints first character of the string
print str[2:5]     # Prints characters starting from 3rd to 5th
print str[2:]      # Prints string starting from 3rd character
print str * 2      # Prints string two times
print str + "TEST" # Prints concatenated string
```

False	class	finally	is	return
None	continue	for	lambda	try
True	def	from	nonlocal	while
and	del	global	not	with
as	elif	if	or	yield
assert	else	import	pass	
break	except	in	raise	

Reserved Words

Python Basic

Variable and Data Structures:

- Data Structures:
  - **Lists**: Most versatile data types, keeps order `[1, ‘2‘, [3], {4}, (5), {6:‘6‘}, …]`
  - Tuples: similar to list, but a „**read-only**“ list, `(1, ‘2‘, [3], {4}, (5), {6:‘6‘}, …)`
  - **Dictionary**: `{‘int‘: 1, ‘str‘: ‘2‘, ‘list‘: [3], ‘set‘: {4}, ‘tup‘: (5), ‘dict‘: {6:‘6‘}, …}`
    - Associative arrays or hashes
    - Key-Value Pairs
    - Key is unique
  - Sets and fronzenset: no order, unique `{1, ‘2‘, [3], {4}, (5), {6:‘6‘}, …}`

- Others: Date, Time, hex, oct, ……

- Data Type Conversion: `new_type(x[, base])`

Outline

1	Programming using Python	14:00 – 15:25
1.1	The Python Programming Language	14:00 – 14:05
1.2	Python Basic	14:05 – 14:55
a	Variable and Data Structures	14:05 – 14:15
b	Logic, Condition and Loops	14:15 – 14:25
c	Functions and Object oriented Programming (OOP)	14:25 – 14:45
d	A whole Python Script	14:45 – 14:55
1.3	Practicle Unit: Your ROS Code in Python	14:55 – 15:25
2	Programming your Motion Planning for Kinova	15:25 – 17:00

Logic, Condition and Loops

- Logic

Operator	Description	Example
==	If the values of two operands are equal, then the condition becomes true.	$a == b$ is not true.
!=	If values of two operands are not equal, then condition becomes true.	
<>	If values of two operands are not equal, then condition becomes true.	$a <> b$ is true. This is similar to != operator.
>	If the value of left operand is greater than the value of right operand, then condition becomes true.	$a > b$ is not true.
<	If the value of left operand is less than the value of right operand, then condition becomes true.	$a < b$ is true.
>=	If the value of left operand is greater than or equal to the value of right operand, then condition becomes true.	$a >= b$ is not true.
<=	If the value of left operand is less than or equal to the value of right operand, then condition becomes true.	$a <= b$ is true.

Operator	Description	Example
in	Evaluates to true if it finds a variable in the specified sequence and false otherwise.	x in y, here in results in a 1 if x is a member of sequence y.
not in	Evaluates to true if it does not finds a variable in the specified sequence and false otherwise.	x not in y, here not in results in a 1 if x is not a member of sequence y.
is	Evaluates to true if the variables on either side of the operator point to the same object and false otherwise.	x is y, here **is** results in 1 if id$x$ equals id y.
is not	Evaluates to false if the variables on either side of the operator point to the same object and true otherwise.	x is not y, here **is not** results in 1 if id$x$ is not equal to id$y$.

Python Basic

Logic, Condition and Loops

- Condition and Loops: `if, for, while`

```
if expression1:
   statement 1.1 …
elif expression2:
   statement 2.1 …
…
else:
   statement n.1 …
```

```
while condition:
   if exp1:
      continue
   elif exp2:
      pass
   else:
      break
```

```
for expression:
   if exp1:
      continue
   elif exp2:
      pass
   else:
      break
```

- Condition and Loops in Data Structures:
  - `a_list = [word for word in [‘RWTH‘, ‘Summer‘, ‘School‘, 2017] if type(word) is str]`
  - `a_set = {word for word in [‘RWTH‘, ‘Summer‘, ‘School‘, 2017, ‘RWTH‘] if type(word) is str}`
  - `A_turple = (word for word in [‘RWTH‘, ‘Summer‘, ‘School‘, 2017]) Why???`

Outline

1	Programming using Python	14:00 – 15:25
1.1	The Python Programming Language	14:00 – 14:05
1.2	Python Basic	14:05 – 14:55
a	Variable and Data Structures	14:05 – 14:15
b	Logic, Condition and Loops	14:15 – 14:25
c	Functions and Object oriented Programming (OOP)	14:25 – 14:45
d	A whole Python Script	14:45 – 14:55
1.3	Practicle Unit: Your ROS Code in Python	14:55 – 15:25
2	Programming your Motion Planning for Kinova	15:25 – 17:00

Functions and OOP

- Function
  - Regular Functions

```
def function_name(para_a, para_b=None):
    statements……
    return res1, res2, res3
```

  - Others: lambda, yield
- OOP:
  - Object oriented and not object oriented
  - Everything in Python is a child object from `class object`

```
class class_name(faher_class=object):
      a_counter = 0

      def __init__(self, name):
         self.name = name

      def all_can_use(self, para):
         return res

      def _not_all_can_use(self, para):
         return res

      @staticmethod
      def method_for_everyone(para):
         pass

      @classmethod
      def method_within_class_instance(cls, para):
         pass
```

Outline

1	Programming using Python	14:00 – 15:25
1.1	The Python Programming Language	14:00 – 14:05
1.2	Python Basic	14:05 – 14:55
a	Variable and Data Structures	14:05 – 14:15
b	Logic, Condition and Loops	14:15 – 14:25
c	Functions and Object oriented Programming (OOP)	14:25 – 14:45
d	A whole Python Script	14:45 – 14:55
1.3	Practicle Unit: Your ROS Code in Python	14:55 – 15:25
2	Programming your Motion Planning for Kinova	15:25 – 17:00

Python Basic

A whole Python Script

- `import` Modules

```
import a_module
import a_module as a_new_name_for_this_module
from a_module import *
from a_module import something_in_this_module
```

- Do as a Python Programmer: PEP8

- Be smart and always ready for error handling

```
try:
   res = trying_to_run_a_function(para)

except a_General_Error:
   print “Are you kidding me?“

Except a_user_defined_error:
   pinrt “Are you kidding youself?“

Finally:
   print “anyway you will get here.“
```

- Many advanced Features
  - Regular Expressions
  - Networking
  - Multithreading
  - GUI

Outline

1	Programming using Python	14:00 – 15:25
1.1	The Python Programming Language	14:00 – 14:05
1.2	Python Basic	14:05 – 14:55
1.3	Practicle Unit: Your ROS Code in Python	14:55 – 15:25
a	Integrated Development Environment (IDE)	14:55 – 15:05
b	Practicle Unit: Python in ROS, Warming Up	15:05 – 15:25
2	Programming your Motion Planning for Kinova	15:25 – 17:00

Outline

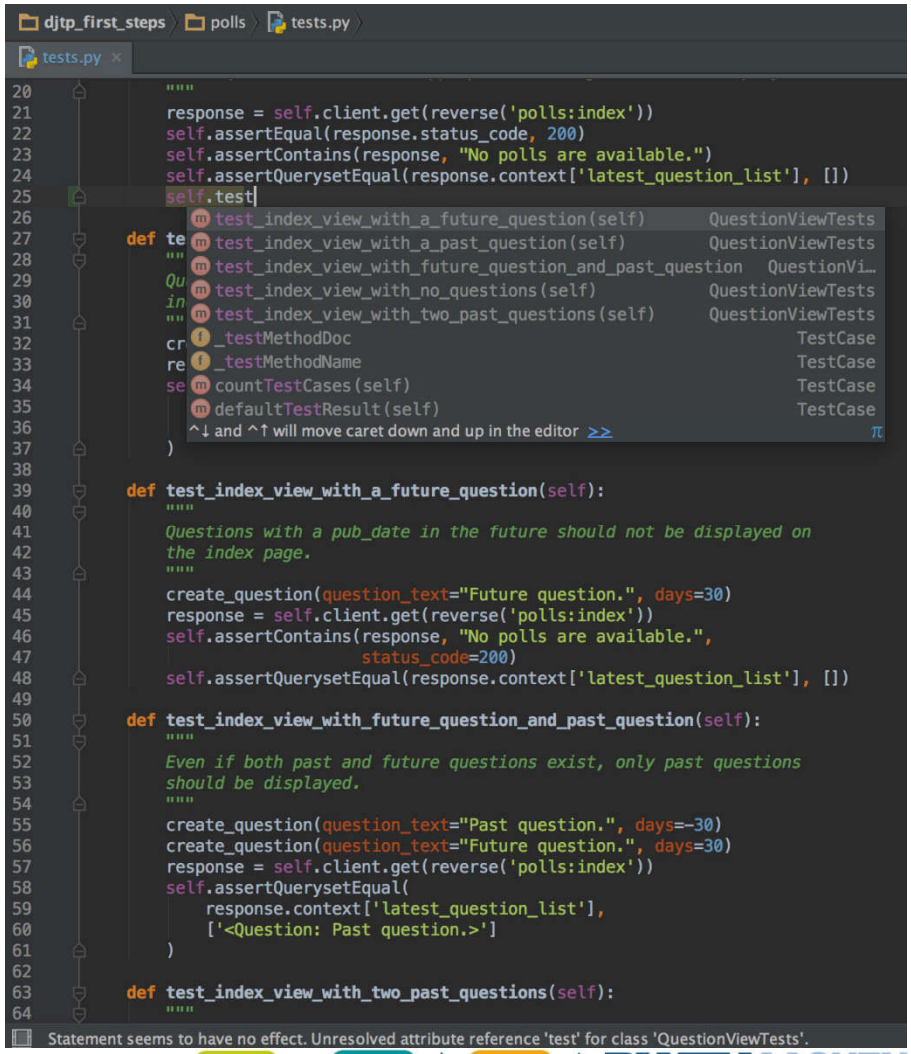
1	Programming using Python	14:00 – 15:25
1.1	The Python Programming Language	14:00 – 14:05
1.2	Python Basic	14:05 – 14:55
1.3	Practicle Unit: Your ROS Code in Python	14:55 – 15:25
a	Integrated Development Environment (IDE)	14:55 – 15:05
b	Practicle Unit: Python in ROS, Warming Up	15:05 – 15:25
2	Programming your Motion Planning for Kinova	15:25 – 17:00

Practicle Unit: Your ROS Code in Python

Integrated Development Environment (IDE)

- An interactive Programming environment:
  - A programming language specified text editor
    - Smart tipping
    - Error checking
    - Keep your code cool
  - Builds automation tools
  - Debugger
- Who needs an IDE?

  People who are not programming **FREAKs**.



Practicle Unit: Your ROS Code in Python

Integrated Development Environment (IDE) : PyCharm

Outline

1	Programming using Python	14:00 – 15:25
1.1	The Python Programming Language	14:00 – 14:05
1.2	Python Basic	14:05 – 14:55
1.3	Practicle Unit: Your ROS Code in Python	14:55 – 15:25
a	Integrated Development Environment (IDE)	14:55 – 15:05
b	Practicle Unit: Python in ROS, Warming Up	15:05 – 15:25
2	Programming your Motion Planning for Kinova	15:25 – 17:00

Practicle Unit: Your ROS Code in Python

Python in ROS, Warming Up

- Build your own Package in ROS
- Visit: http://wiki.ros.org/ROS/Tutorials/CreatingPackage
- Workflow:
  1. Set your Directory in `/catkin_ws/src` using `cd ~/catkin_ws/src`
  2. Using the Command `catkin_create_pkg your_pacakge_name dependencies`

e.g: `catkin_create_pkg move_kinova std_msgs rospy moveit_commander geometry_msgs`

-> If you missed one dependency package, you can add them later in:

- CMakeFile.txt: find_package()
- Package.xml: <build_depend> and <run_depend>

1. Go Back to `/catkin_ws/` and run `catkin_make` to build your new Package
2. Don‘t forget to add your new-build workspace to ROS Environment using

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

Tipp: if you want to be smart, you can put this command in ~/.bashrc

```
source /home/your_accound_name/catkin_ws/devel/setup.bash
```

Practicle Unit: Your ROS Code in Python

Python in ROS, Warming Up

- A Simple Publisher and Subscriber in Python

- Visit: http://wiki.ros.org/ROS/Tutorials/WritingPublisherSubscriber%28python%29

- Task:
  - A Publisher: send a message with Time from your Input:

```
msg = str(input("Waitting for a Message…"))
```

  - A Subscriber: print the message

Outline

1	Programming using Python	14:00 – 15:25
2	Programming your Motion Planning for Kinova	15:25 – 17:00
2.1	Motion Planning Interface for Python	15:25 – 15:30
2.2	Mission Statement	15:30 – 15:40
2.3	Practicle Unit: Get into the task	15:40 – 17:00

Outline

1	Programming using Python	14:00 – 15:25
2	Programming your Motion Planning for Kinova	15:25 – 17:00
	2.1 Motion Planning Interface for Python	15:25 – 15:30
	2.2 Mission Statement	15:30 – 15:40
	2.3 Practicle Unit: Get into the task	15:40 – 17:00

Programming your Motion Planning for Kinova

Motion Planning Interface for Python

- Visit:
  http://docs.ros.org/kinetic/api/moveit_tutorials/html/doc/pr2_tutorials/planning/scripts/doc/move_group_python_interface_tutorial.html

Outline

1	Programming using Python	14:00 – 15:25
2	Programming your Motion Planning for Kinova	15:25 – 17:00
	2.1 Motion Planning Interface for Python	15:25 – 15:30
	2.2 Mission Statement	15:30 – 15:40
	2.3 Practicle Unit: Get into the task	15:40 – 17:00

Programming your Motion Planning for Kinova

Mission Statement

- Pick up and Place in the Hole
  - Using your URDF or XACRO and MoveIt! Configuration from last Section
  - Programming a Python Script for the Task
  - Write a Launch to run your Node and RVIZ Visualisation

Outline

1	Programming using Python	14:00 – 15:25
2	Programming your Motion Planning for Kinova	15:25 – 17:00
	2.1 Motion Planning Interface for Python	15:25 – 15:30
	2.2 Mission Statement	15:30 – 15:40
	2.3 Practicle Unit: Get into the task	15:40 – 17:00

Programming your Motion Planning for Kinova

Practicle Unit: Get into the task

Backup: New Agenda