

Part 3: Programming in Python

Dr. rer. nat. Teena Hassan

M.Sc. Mihaela Popescu

thassan@uni-bremen.de

Prof. Dr. Dr. h.c. Frank Kirchner

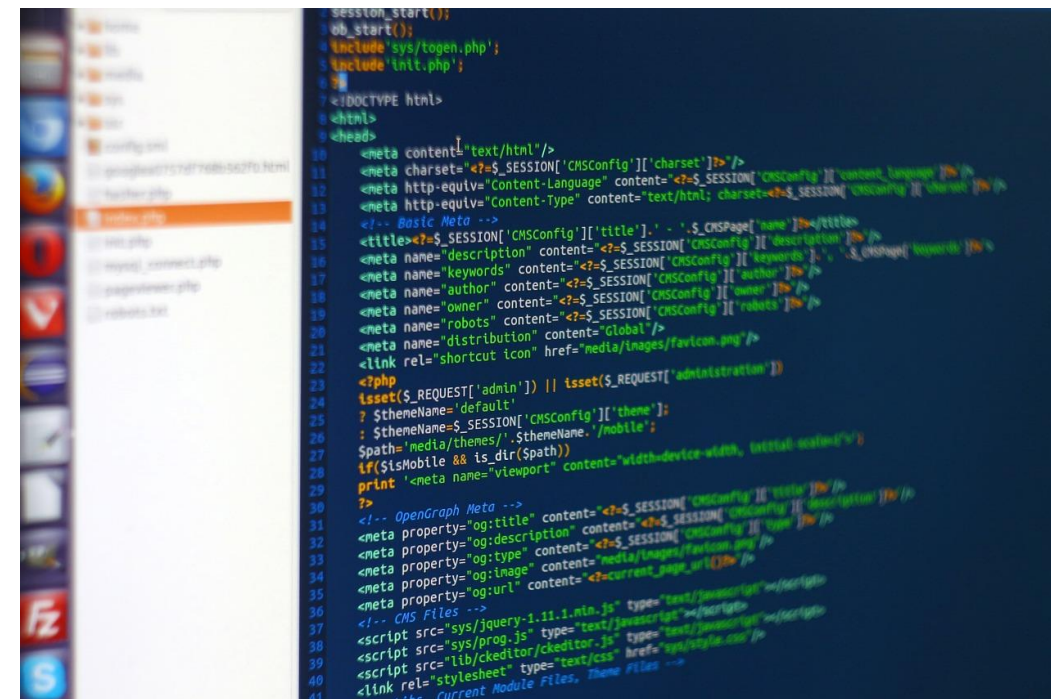
Arbeitsgruppe Robotik, Universität Bremen

Robotics Innovation Center, DFKI

<https://robotik.dfki-bremen.de/>

robotik@dfki.de

October 26, 2022 – Bremen



```
1 session_start();
2 ob_start();
3 include 'sys/togen.php';
4 include 'init.php';
5
6
7 <!DOCTYPE html>
8 <html>
9 <head>
10 <meta content="text/html"/>
11 <meta charset="<?=$SESSION['CMSConfig']['charset']">"/>
12 <meta http-equiv="Content-Language" content="<?=$SESSION['CMSConfig']['content_language']">"/>
13 <meta http-equiv="Content-Type" content="text/html; charset=<?=$SESSION['CMSConfig']['charset']">"/>
14 <!-- Basic Meta -->
15 <title><?=$SESSION['CMSConfig']['title'] . ' - ' . $CMSPage['name']"></title>
16 <meta name="description" content="<?=$SESSION['CMSConfig']['description']">"/>
17 <meta name="keywords" content="<?=$SESSION['CMSConfig']['keywords'] . ', ' . $CMSPage['keywords']">"/>
18 <meta name="author" content="<?=$SESSION['CMSConfig']['author']">"/>
19 <meta name="owner" content="<?=$SESSION['CMSConfig']['owner']">"/>
20 <meta name="robots" content="<?=$SESSION['CMSConfig']['robots']">"/>
21 <meta name="distribution" content="global"/>
22 <link rel="shortcut icon" href="media/images/favicon.png"/>
23 <?php
24 isset($REQUEST['admin']) || isset($REQUEST['administration'])
25 ? $themeName=$SESSION['CMSConfig']['theme'];
26 : $themeName=$SESSION['CMSConfig']['theme'];
27 $path='media/themes/'.$themeName;
28 if($isMobile && is_dir($path))
29 print <meta name="viewport" content="width=device-width, initial-scale=1">
30 ?>
31 <!-- OpenGraph Meta -->
32 <meta property="og:title" content="<?=$SESSION['CMSConfig']['title']">"/>
33 <meta property="og:description" content="<?=$SESSION['CMSConfig']['description']">"/>
34 <meta property="og:type" content="media/images/favicon.png"/>
35 <meta property="og:image" content="<?=$current_page_url">"/>
36 <meta property="og:url" content="<?=$current_page_url">"/>
37 <!-- CMS Files -->
38 <script src="sys/jquery-1.11.1.min.js" type="text/javascript"></script>
39 <script src="sys/prog.js" type="text/javascript"></script>
40 <script src="lib/ckeditor/ckeditor.js" type="text/javascript"></script>
41 <link rel="stylesheet" type="text/css" href="sys/style.css"/>
```

- At the end of this lecture, you will be able to:
 1. Describe some of the basic constructs in the programming language Python.
 2. Explain how simple ROS 2 publisher and subscriber nodes can be written in Python.

- The Python programming language:
 - High-level
 - Interpreted
 - ▶ That is, the code written in Python is executed line by line.
 - Interactive
 - ▶ Using the command "python", you can start the Python interpreter on your CLI.

```
thassan@THASSAN-P15V-U:~$ python3
Python 3.6.9 (default, Jun 29 2022, 11:45:57)
[GCC 8.4.0] on linux
Type "help", "copyright", "credits" or "license" for more information.
>>> 
```

Writing Python Programs

- Python programs are saved as files with **.py** extension.
- Software to write Python programs
 1. Using any text editor
 - ▶ From CLI, use *python <filename>*, to run the Python program.
 2. Using Integrated Development Environments (IDE)
 - ▶ Web-based interactive IDE, e.g. Jupyter Notebook
 - ▶ Desktop IDE, e.g. PyCharm

- Variables are labels for memory locations.
- When we assign values to a variable, they are written in the memory location referred to by the variable.
 - = is the assignment operator.
- The values have a specific data type.
 - For example, 100 is an integer; 'hello' is a string.
- Different data types require different amount of space (bytes) in the memory for storage.
- Python is dynamically typed. That is, memory is allocated dynamically to a variable at runtime depending on the data type of the value assigned to it.

```
thassan@THASSAN-P15V-U:~$ python3
Python 3.6.9 (default, Jun 29 2022, 11:45:57)
[GCC 8.4.0] on linux
Type "help", "copyright", "credits" or "license" for more information.
>>> v1 = 100
>>> v2 = 'hello'
>>> print(v1)
100
>>> print(v2)
hello
>>> v2 = v2 + ' there!'
>>> print(v2)
hello there!
>>> □
```

- List: A collection of items
 - `list_1 = [1, 2, 'weekly', 15.5, 'robots']`
- Dictionary: A collection of key-value pairs
 - `dict_1 = { 'title': 'robotics', 'date': '26/10/2022', 'duration': 1.5}`
- Indexing:
 - First item has in a list has index 0.
 - ▶ `print(list_1[0])` will print 1.
 - Dictionary items are indexed using the keys.
 - ▶ `print(dict_1['date'])` will print 26/10/2022.

```
thassan@THASSAN-P15V-U:~$ python3
Python 3.6.9 (default, Jun 29 2022, 11:45:57)
[GCC 8.4.0] on linux
Type "help", "copyright", "credits" or "license" for more information.
>>>
>>>
>>> list_1 = [1, 2, 'weekly', 15.5, 'robots']
>>>
>>> print(list_1[0])
1
>>>
>>> dict_1 = {'title': 'robotics', 'date': '26/10/2022', 'duration': 1.5}
>>>
>>> print(dict_1['date'])
26/10/2022
>>> □
```

Python: Conditional Statements

- If-then-else statements check on conditions and execute a different block of code, depending on whether the condition is True or False.

```
thassan@THASSAN-P15V-U:~$ python3
Python 3.6.9 (default, Jun 29 2022, 11:45:57)
[GCC 8.4.0] on linux
Type "help", "copyright", "credits" or "license" for more information.
>>>
>>> v = 100
>>>
>>> if v > 49:
...     print( 'Passed' )
... else:
...     print( 'Failed' )
...
Passed
>>> □
```

- Note that the block of code within the if and the else parts should be intended using a tab or a fixed no. of empty spaces.

- If we want to execute a block of code several times, until a specific condition is met, we can use control loops.
- ***For*** loop, ***while*** loop

```
thassan@THASSAN-P15V-U:~$ python3
Python 3.6.9 (default, Jun 29 2022, 11:45:57)
[GCC 8.4.0] on linux
Type "help", "copyright", "credits" or "license" for more information.
>>>
>>> robots = ['Mantis', 'RH5', 'Charlie', 'Crex']
>>>
>>> for i in range(0, len(robots)):
...     print(robots[i])
...
Mantis
RH5
Charlie
Crex
>>> █
```

```
thassan@THASSAN-P15V-U:~$ python3
Python 3.6.9 (default, Jun 29 2022, 11:45:57)
[GCC 8.4.0] on linux
Type "help", "copyright", "credits" or "license" for more information.
>>>
>>> robots = ["Mantis", "RH5", "Charlie", "Crex"]
>>>
>>> i=0
>>> while (i < len(robots)):
...     print( robots[i] )
...     i = i + 1
...
Mantis
RH5
Charlie
Crex
>>>
>>> █
```


- A block of code that performs a specific task can be encapsulated into a function.
- Advantages:
 - Enhances readability: We understand better what this block of code is doing.
 - Enables code reuse: It prevents us from repeating the same code every time we want to do this task in the same program.

```
thassan@THASSAN-P15V-U:~$ python3
Python 3.6.9 (default, Jun 29 2022, 11:45:57)
[GCC 8.4.0] on linux
Type "help", "copyright", "credits" or "license" for more information.
>>>
>>> robots = [ 'Mantis', 'RH5', 'Charlie', 'Crex' ]
>>>
>>> def print_item_wise( list_in = None ):
...     if list_in:
...         i = 0
...         while ( i < len(list_in) ):
...             print( list_in[i] )
...             i = i + 1
...     else:
...         print( "List was not provided." )
...
>>>
>>> print_item_wise( robots )
Mantis
RH5
Charlie
Crex
>>>
>>> print_item_wise()
List was not provided.
>>>
>>> □
```

- Objects that have similar characteristics can be grouped into a class.
- ⑩ The class then defines the prototype for all such objects.
- ⑩ E.g. We can define a class Robot that defines the general characteristics of robots.
 - ⑩ Manus, RH5, etc. will then be an object of class Robot.
 - ⑩ Objects are also called instances of a class.

Inside a Python file **class_robot.py**, we have defined the class Robot.

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

```
class Robot:
```

```
    def __init__(self, name, rob_type):  
        self.name = name  
        self.type = rob_type  
        self.state = 'OFF'
```

```
    def print_state(self):  
        print( self.name + ' is in state ' + self.state + '.')
```

```
    def turn_on(self):  
        self.state = 'ON'
```

```
    def turn_off(self):  
        self.state = 'OFF'
```

Constructor (method that initializes attributes when an object is created)

Attributes: self.<variable_name>

Methods
(functions
defined inside
a class)

Python: Classes and Objects

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

```
class Robot:
```

```
    def __init__(self, name, rob_type):
        self.name = name
        self.type = rob_type
        self.state = 'OFF'

    def print_state(self):
        print( self.name + ' is in state ' + self.state + '.')

    def turn_on(self):
        self.state = 'ON'

    def turn_off(self):
        self.state = 'OFF'
```

Import the class definition from the file.

Create the object Mantis as an instance of the class Robot

Access the methods / attributes using the dot operator.

Note: *self* is an internal reference to the object.

In e.g. below, *self* will automatically refer to Mantis when the attributes and methods are accessed using the dot operator.

```
thassan@THASSAN-P15V-U:~/Downloads$ python3
Python 3.6.9 (default, Jun 29 2022, 11:45:57)
[GCC 8.4.0] on linux
Type "help", "copyright", "credits" or "license" for more information.
>>>
>>> from class_robot import Robot
>>>
>>> Mantis = Robot('Mantis01', 'zoomorphic')
>>>
>>> Mantis.turn_on()
>>>
>>> Mantis.print_state()
Mantis01 is in state ON.
>>>
>>> █
```

```
#!/usr/bin/python

class Robot:

    def __init__(self, name, rob_type):
        self.name = name
        self.type = rob_type
        self.state = 'OFF'

    def print_state(self):
        print( self.name + ' is in state ' + self.state + '.')

    def turn_on(self):
        self.state = 'ON'

    def turn_off(self):
        self.state = 'OFF'

class Humanoid(Robot):

    def __init__(self, name, n_joints):
        super().__init__(name, 'anthropomorphic')
        self.joints = {'total': n_joints, 'names': []}

    def set_joints(self, joint_names):
        if len(joint_names) == self.joints['total']:
            self.joints['names'] = joint_names.copy()
        else:
            print('Fewer or more joint names than joints available.')

    def print_joints(self):
        print( self.joints['names'] )
```

`super().__init__`
calls the
constructor of the
parent class.

Class *Humanoid* inherits from the class *Robot*.

Thereby, *Humanoid* gets the attributes and methods of *Robot* class and can change or add new methods and attributes.

Robot is the parent or base class.
Humanoid is the child or derived class.

ROS 2 Publisher -- Example

```
import rclpy
from rclpy.node import Node

from std_msgs.msg import String
```

```
class MinimalPublisher(Node):
```

```
    def __init__(self):
        super().__init__('minimal_publisher')
        self.publisher_ = self.create_publisher(String, 'topic', 10)
        timer_period = 0.5 # seconds
        self.timer = self.create_timer(timer_period, self.timer_callback)
        self.i = 0
```

```
    def timer_callback(self):
        msg = String()
        msg.data = 'Hello World: %d' % self.i
        self.publisher_.publish(msg)
        self.get_logger().info('Publishing: "%s"' % msg.data)
        self.i += 1
```

```
def main(args=None):
```

```
    rclpy.init(args=args)
```

```
    minimal_publisher = MinimalPublisher()
```

```
    rclpy.spin(minimal_publisher)
```

```
    # Destroy the node explicitly
    # (optional - otherwise it will be done automatically
    # when the garbage collector destroys the node object)
    minimal_publisher.destroy_node()
    rclpy.shutdown()
```

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

Definition of class *MinimalPublisher*. This inherits from class *Node*.

Constructor initializing attributes of a publisher node.

Creates the publisher

1. Mention the type of data that will be published.
2. Mention the name of the topic where data will be published.
3. Mention the size of the queue that buffers published messages.

Creates a timer to publish a message at a specific frequency.
• Note: This is not always needed.

Publishes the message on the topic.

Initializes ROS 2 Python context.

Creates an object of class *MinimalPublisher*. That is, creates a node.

Spins the node. That is, executes the callbacks until the node is shutdown.

ROS 2 Subscriber -- Example

```
import rclpy
from rclpy.node import Node

from std_msgs.msg import String

class MinimalSubscriber(Node):

    def __init__(self):
        super().__init__('minimal_subscriber')
        self.subscription = self.create_subscription(
            String,
            'topic',
            self.listener_callback,
            10)
        self.subscription # prevent unused variable warning

    def listener_callback(self, msg):
        self.get_logger().info('I heard: "%s"' % msg.data)

def main(args=None):
    rclpy.init(args=args)

    minimal_subscriber = MinimalSubscriber()

    rclpy.spin(minimal_subscriber)

    # Destroy the node explicitly
    # (optional - otherwise it will be done automatically
    # when the garbage collector destroys the node object)
    minimal_subscriber.destroy_node()
    rclpy.shutdown()

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

Creates the subscriber

1. Mention the type of data that will be received.
2. Mention the name of the topic where data will be received.
3. Mention the function that should be called when a message with the data is received.
4. Mention the size of the queue that buffers received messages.

The received ROS 2 message is stored in *msg*.
msg.data gives the data in the message.

Creates the node of type *MinimalSubscriber*.

Spins the node.

Conclusion

- In this lecture, you learnt to:
 1. Describe some of the basic constructs in the programming language Python.
 2. Explain how simple ROS 2 publisher and subscriber nodes can be written in Python.

- Python crash course from Robin Horn on YouTube [[link](#)].
 - Some videos have been linked in the PDF uploaded in Stud.IP under Tutorial-02.

⑩ Python3 Tutorial: <https://www.tutorialspoint.com/python3/index.htm>

⑩ Python Tutorial with interactive examples:
<https://www.w3schools.com/python/default.asp>

Next Part: Battery Handling