1. Theoretical background

- Actions are a form of asynchronous communication in ROS 2 based on sevices and topics.
- Their comunication dynamics is:
  - Server:
    1. Wait and receive the action-service request
    2. Execute the action and publish the **feedback**
    3. Return the **request_result**
  - Client:
    1. Wait for the server to available
    2. Send the goal request to the action-server
    3. Wait until the server **acknowledge the request** and get the **request_result**
    4. If the request was accepted,
       - Optionally, read the **feedback**
       - Wait until the **action result** is ready and read it

2. Python actions

2.1 Server

2.1.1 Structure

1. Structure a node
2. Import the action class and the interfaces
3. In the **constructor**:
   - Instanciate an action server defining: **self class**, interface, action_service and callback

4. Create a callback with a parameters: A **goal_handler-object** to receive the goal.
    - Get the goal using the **goal_handler object**
    - Instanciate a **result object** and a **feedback object** using the interface import
    - Execute the action and publish the **feedback object**
    - Indicate that the goal is complete using the **goal_handler object**
    - Assing and return the the **result object**
5. Main function:
    - Spin the server
6. Create an entry point and compile

2.1.2 Methods

- API
- Server instanciation

```
from rclpy.action import ActionServer

Node.server_ = ActionServer(self, <interface>,
<'action_service_name'>, callback)
```

- Read an objective

```
goal = goal_handler.request.<atribute>
```

- Publish feedback

```
feedback.<atribute>.append(<vaue>)
goal_handler.publish_feedback(feedback)
```

2.1.3 Callbacks

- Parameters: attributes defined in the action file
  - goal_handler_object
- Contents:
  - Execute the action
  - Publish the feedback_object
- Returns:
  - result_object

```
def callback(self, goal_handler):
    # Get the request using the "goal handler"
    goal = goal_handle.request.order

    # Create instances of a feedback and a result interfaces
    result = MyAction.Result()
    feedback = MyAction.Feedback()

    for i in range(1, goal):
        # Publish the feedback using the "goal handler"
        feedback.partial_sequence.append(i)
        goal_handle.publish_feedback(feedback)

    # Show that the goal is now complete using the "goal
handler"
    goal_handle.succeed()

    # Return the result
    result.sequence = feedback.partial_sequence
    return result
```

2.2 Client

2.2.1 Structure

1. Structure a node
2. Import the ActionClient class ant the action interfaces
3. In the **constructor**:

    - Instanciate a client defining: **self object**, interface and action_service
4. Create a **request method**:
    - Instanciate and assign the attributes of a **goal object**
    - Request the action-service by sending the **goal object** and get a **request_handler object**
        - Optionally, bind a **feedback_callback**
    - Wait for the request acceptance by binding the **request_handler object** with a **request_callback**
5. request_callback
    - Check if the request was accepted using **request_handler object** received as a parameter of the callback
    - If accepted, instanciate a **result_handler object** and bind it with a **result_callback** for when the result is ready
6. result_callback
   - Read the result values uusing the **result_handler object** received as a parameter of the callback
   - Optionally, shutdown the node
7. feedback_callback
    - Create a **feedback object** as an instance feedback_msg.feedback attribute received as a parameter
    - Read the feedback value
8. Main function:
    - Invoke the request method
    - **Spin** the server
9. Create an entry point and compile

2.2.2 Methods

- API
- Client instanciation:

```
action_client = Node.ActionClient(self, <interface>,
<action_name>)
```

- Instanciate a goal object:

```
goal = <interface>.Goal()
```

- Wait for the server:

```
Node.action_client.wait_for_server()
        ...
```

- Request the action-service: It returns a **request_handler object**

```
Node.request_handler = Node.
<action_client>.send_goal_async(<goal>, feedback_callback =
Node.<feedback_callback>)
```

- Bind the **request_handler object** with a **request_callback**:

```
Node.request_handler.add_done_callback(<Node.request_callback
>)
```

- Check if the request was acceped:

```
if not request_handler.result().accepted:
        return
```

- Request the action-service **result**:

```
Node.result_handler =
request_handler.result().get_result_async()
```

- Bind the **result_handler object** with a **result_callback**:

```
Node.result_handler.add_done_callback(<Node.result_callback>)
```

- Read the result:
  - Instanciate a **result object**
  - Read the **result object** attributes

```
result = <get_result>.result().result
<client>.get_logger().info(f'Result: {result.<attribute>}')
```

2.2.3 Callbacks

acknowledge_request_callback

- Parameters: request_acknoeledgement object
- Contents:
  - Check the request response
  - Resuqest the action-service result
- Returns: Empty

```
def acknowledge_request_callback(self, request_ack):
        request_response = request_ack.result()
        if not request_response.accepted:
            self.get_logger().info('Goal rejected')
            return
        self.get_logger().info('Goal accepted')

        # Request the result
        self.get_result = request_response.get_result_async()

        # Associate the "get_result object" to a callback
method for when the result is ready

self.get_result.add_done_callback(self.get_result_callback)
```

get_result_callback

- Parameters: request_acknoeledgement object

- Contents:
  - Check the request response
  - Request the action-service result
- Returns: Empty

```
def get_result_callback(self, get_result):
        # Get the result
        result = get_result.result().result
        self.get_logger().info('Result: {0}'.format(result.
<attribute>))

        rclpy.shutdown()
```

feedback_callback

- Parameters: feedback_handler object
- Contents:
  - Check the request response
  - Resuqest the action-service result
- Returns: Empty

```
def feedback_callback(self, feedback_handler):
        self.get_logger().info(f'Feedback:
{feedback_handler.feedback.current_position}')
```