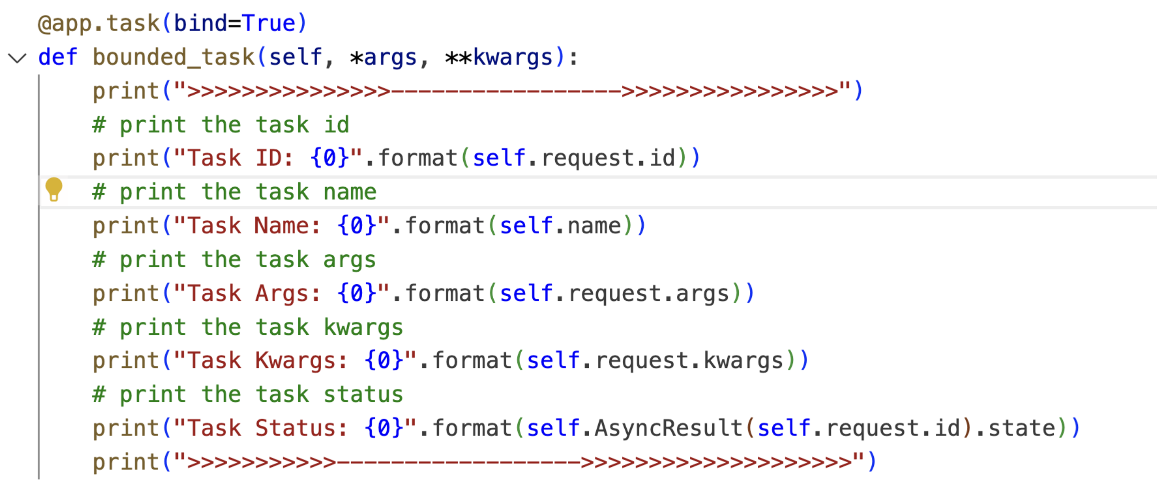
Chapter - 6 : State Management with Celery

In General, any program which stays in the memory after executing a function e.g. `sum` *is a state machine*

1. State Machine Presentations

Bounded Tasks in Celery

A bounded task in celery has access to the task instance using `self`



```
@app.task(bind=True)
def bounded_task(self, *args, **kwargs):
    print(">>>>>>>>>>>>>>>----------------->>>>>>>>>>>>>>>>")
    # print the task id
    print("Task ID: {0}".format(self.request.id))
    # print the task name
    print("Task Name: {0}".format(self.name))
    # print the task args
    print("Task Args: {0}".format(self.request.args))
    # print the task kwargs
    print("Task Kwargs: {0}".format(self.request.kwargs))
    # print the task status
    print("Task Status: {0}".format(self.AsyncResult(self.request.id).state))
    print(">>>>>>>>>>>------------------>>>>>>>>>>>>>>>>>>>>")
```

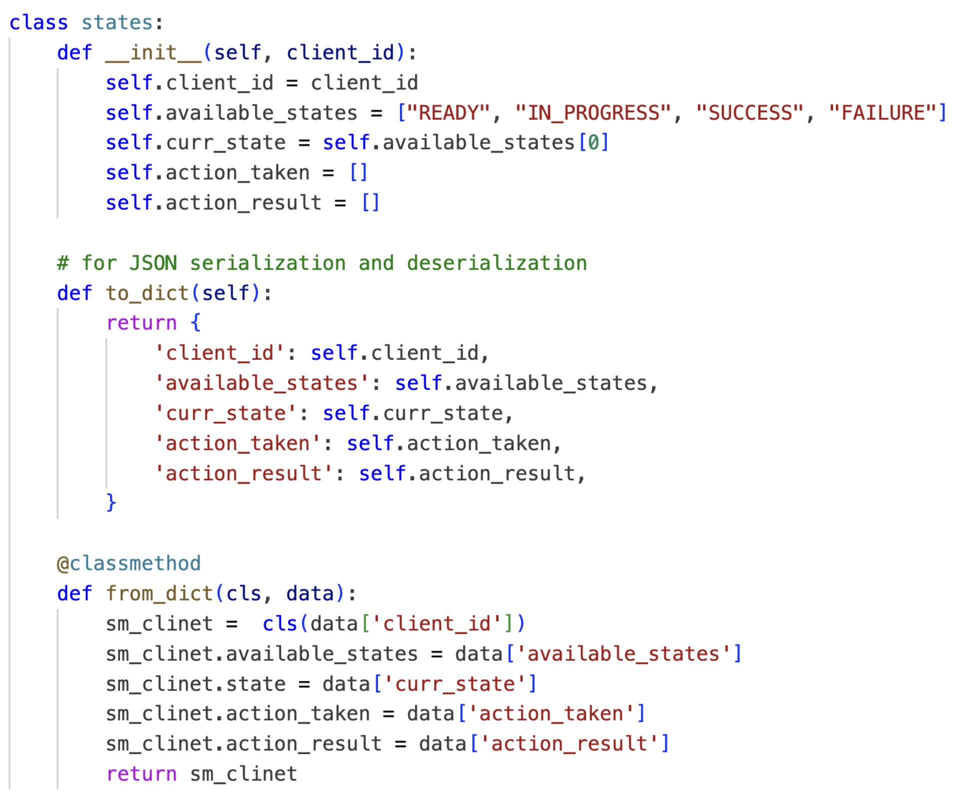
Example:1

Managing State in Celery - Alternative Method

Even though celery generates a client ID and can maintain a state within it, it's generally not suitable for complex state management from the client perspective. However, it's possible to do client state management with a separate state class, but we need to make sure that the class JSON serializable when we're passing it to celery and getting the results back

Note: if you're just doing server side state management, you can skip serialization and deserialization, but just make sure to keep a client ID

Here is how a simple state class with JSON serialization capability will look like



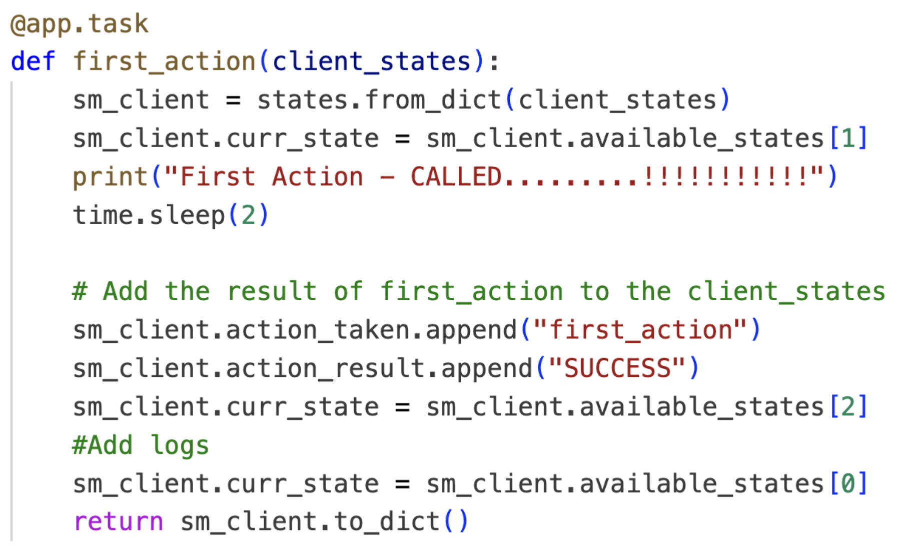
```python
class states:
    def __init__(self, client_id):
        self.client_id = client_id
        self.available_states = ["READY", "IN_PROGRESS", "SUCCESS", "FAILURE"]
        self.curr_state = self.available_states[0]
        self.action_taken = []
        self.action_result = []

    # for JSON serialization and deserialization
    def to_dict(self):
        return {
            'client_id': self.client_id,
            'available_states': self.available_states,
            'curr_state': self.curr_state,
            'action_taken': self.action_taken,
            'action_result': self.action_result,
        }

    @classmethod
    def from_dict(cls, data):
        sm_clinet =  cls(data['client_id'])
        sm_clinet.available_states = data['available_states']
        sm_clinet.state = data['curr_state']
        sm_clinet.action_taken = data['action_taken']
        sm_clinet.action_result = data['action_result']
        return sm_clinet
```

Serializable State Class

Once we have the instance of this class passed as a parameter in the celery task, here is how you can update the information in there after JSON deserialization and sending back the serialized value



```python
@app.task
def first_action(client_states):
    sm_client = states.from_dict(client_states)
    sm_client.curr_state = sm_client.available_states[1]
    print("First Action - CALLED.........!!!!!!!!!!!")
    time.sleep(2)

    # Add the result of first_action to the client_states
    sm_client.action_taken.append("first_action")
    sm_client.action_result.append("SUCCESS")
    sm_client.curr_state = sm_client.available_states[2]
    #Add logs
    sm_client.curr_state = sm_client.available_states[0]
    return sm_client.to_dict()
```

Tasks maintaining states

- **Example : 2**

Classroom Exercise

Manage the states at server side considering the client_ID is transferred in the task as a parameter

- **Example : 3**