LT 14b Stack and Queue (Part 1)

Data Structure using OOP

What is a Stack?
How we code it in the procedural way?

- A stack is a Last-In-First-Out (LIFO) data structure.
- Using Push and Pop to add or remove element from the Stack

Front **End**

```
stack = ["Eric", "John", "Michael"]


>>> print("< < < Stack < < <")
>>> print(stack)
```

Output:

```
< < < Stack < < <
["Eric", "John", "Michael"]
```

Front **End**

```
stack = ["Eric", "John", "Michael"]
```

```
def pop (s):
    s.pop()

def push(s, value):
    s.append(value)
```

To remove an element from the stack:

```
>>> pop(stack)
```

Show the element removed and mutate the stack.

```
>>> print("< < < Stack < < <")
>>> print(stack)
```

Output:

```
["Michael"]

< < < Stack < < <
["Eric", "John"]
```

Front　　　　End

```
stack = ["Eric", "John"]
```

To add an element into the stack:

```
>>> push(stack, "Olivia")
```

There will be no output, but the stack is mutated.

```
>>> print("< < < Stack < < <")
>>> print(stack)
```

```
def pop (s):
    s.pop()

def push(s, value):
    s.append(value)
```

Output:

```
< < < Stack < < <
["Eric", "John", "Olivia"]
```

Front

End

```
queue = ["Eric", "John", "Michael"]
```

```
def dequeue(q):
    q.remove(q[0])
def enqueue(q, value):
    q.append(value)
```

To remove an element from the queue:

```
>>> dequeue(queue)
```

Show the element removed and mutate the queue.

```
>>> print("< < < Queue < < <")
>>> print(queue)
```

Output:

```
["Eric"]

< < < Queue < < <
["John", "Michael"]
```

Front **End**

```
queue = ["John", "Michael"]
```

```
def dequeue(q):
    q.remove(q[0])
def enqueue(q, value):
    q.append(value)
```

To add an element into the queue:

```
>>> enqueue(queue, "Olivia")
```

There will be no output, but the queue is mutated.

```
>>> print("< < < Queue < < <")
>>> print(queue)
```

Output:

```
< < < Queue < < <
["John", "Michael", "Olivia"]
```

LT 14b Stack and Queue (Part 2)

Data Structure using OOP

Using OOP to implement a Stack

```
class Stack():
    #Constructor
    def __init__(self, seq=[]):
        self.container = []
        for value in seq:
            self.container.append(value)
```

```
>>> emptyStack = Stack()
>>> Stack_from_string = Stack('abcd')
>>> myStack = Stack(["Eric", "John", "Michael"])
```

```
class Stack():
      ...
      def push(self, value):
            print("  Add: " + str(value))
            self.container.append(value)

      def pop(self):
            pass
```

A print statement is included here to indicate the element being added or removed .

```
>>> myStack = Stack(["Eric", "John", "Michael"])
>>> myStack.pop()
>>> myStack.pop()
>>> myStack.push("Olivia")
```

Output:

```
Remove: Michael
Remove: John
Add: Olivia
```

```
class Stack():
    ...
    def is_empty(self):
        return len(self.container)==0
```

```
>>> myStack = Stack(["Eric", "John", "Michael"])
>>> myStack.pop()
>>> myStack.pop()
>>> myStack.push("Olivia")
>>> myStack.is_empty()
```

Output:

```
    False
```

```
class Stack():
    ...
    def output(self):
        if len(self.container) == 0:
            print('Empty Stack')
        else:
            st = ""
            for value in self.container:
                st = st + " < " + str(value)
            print(st)
```

```
>>> myStack.output()
```

Output:

```
 < Eric < Olivia
```

Using OOP to implement a Queue

```
class Queue():
    def __init__(self, seq=[]):
        pass
```

```
>>> myQueue = Queue(["Eric", "John", "Michael"])
```

Using OOP to implement a Queue

```
class Queue():
    ...
    def enqueue(self, value):
        pass
    def dequeue(self):
        pass
    def is_empty(self):
        pass
    def output(self):
        pass
```

```
>>> myQueue.dequeue()
>>> myQueue.dequeue()
>>> myQueue.enqueue("Olivia")
>>> myQueue.is_empty()
>>> myQueue.output()
```

Output:

```
Remove: Eric
Remove: John
Add: Olivia
False
 < Michael < Olivia
```

LT 14b Stack and Queue (Part 3)

Data Structure using OOP

What are the similarities and differences between Stack and Queue?

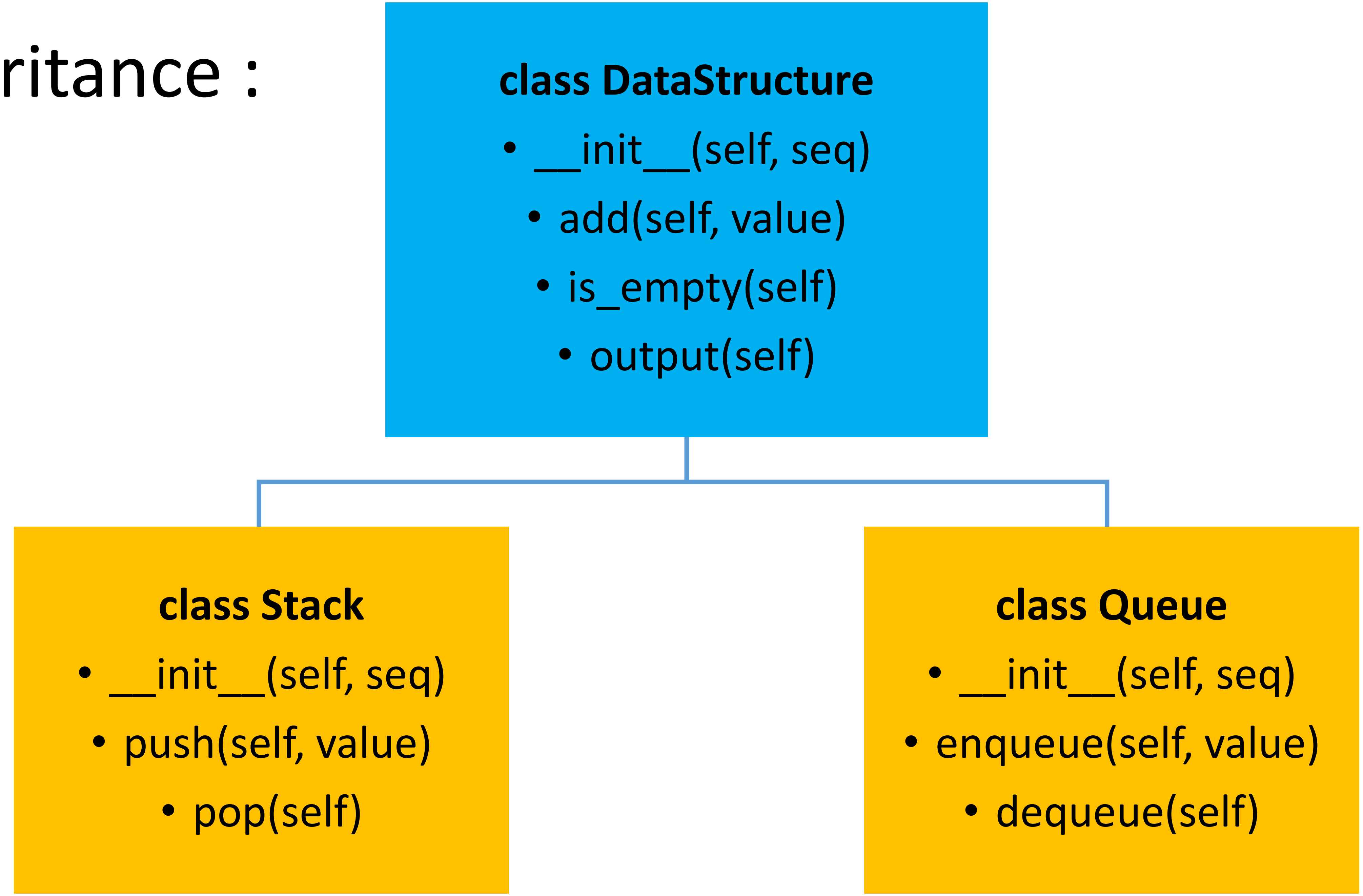
Similarities :

- initialization
- push() and enqueue()
- output()
- is_empty()

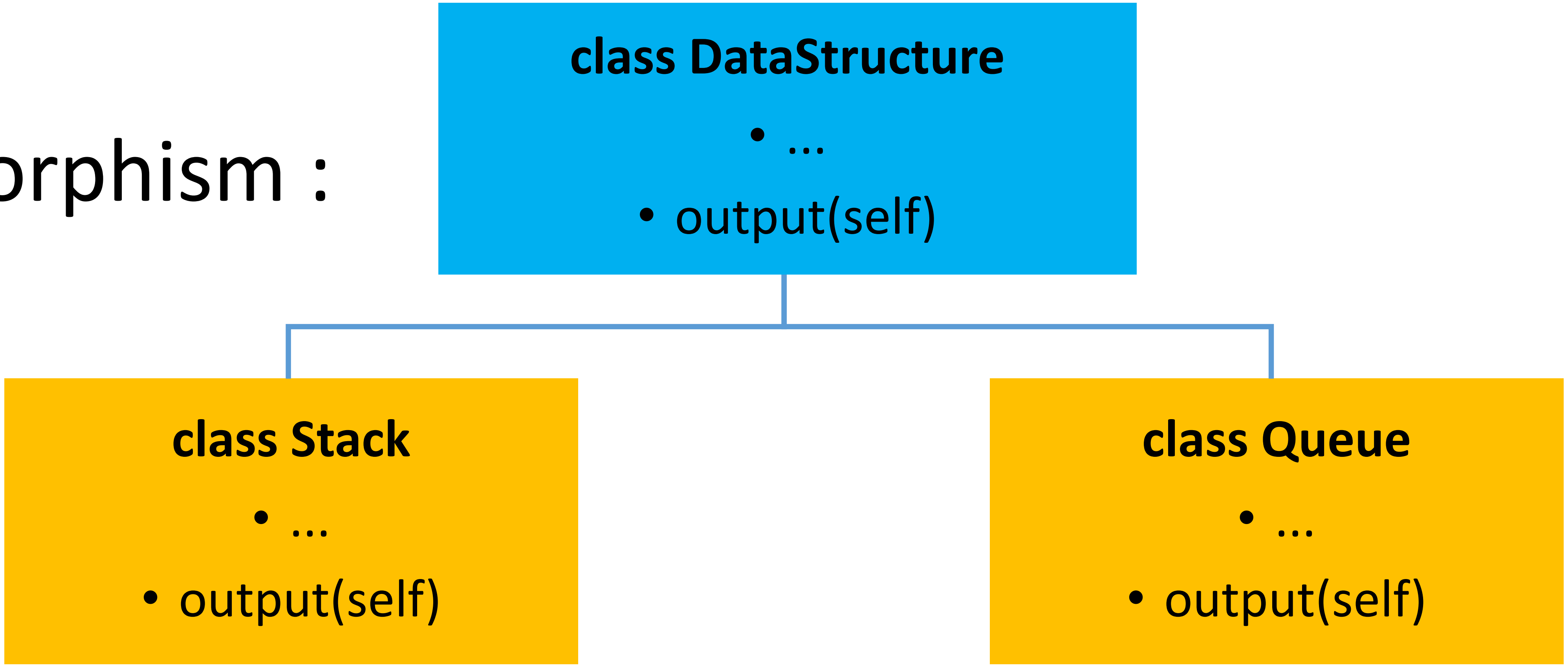
Differences :

- pop() and dequeue()

OOP Inheritance :



OOP Polymorphism :



```
>>> myStack.output()
```

Output:

```
< < < Stack < < <
 < Eric < Olivia
```

```
>>> myQueue.output()
```

Output:

```
< < < Queue < < <
 < Michael < Olivia
```