State Machine Misconceptions

What *not* to do when writing a state machine program.

Do Not Call SDK Actions Directly

```
class Forward75(StateNode):
    def start(self, event=None):
        self.robot.drive_straight(distance_mm(75))
        self.post_completion()
```

This is bad code. It bypasses all the FSM machinery for keeping track of running actions and generating completion events.

Read the implementation of Forward in nodes.py to see how the Forward node actually works.

Use Subclass To Override Behavior

```
class Forward75(Forward):
    def __init__(self, **kwargs):
        super().__init__(**kwargs)
        self.distance = distance_mm(75)
```

Don’t Call Node Constructors

```
class TriangleLeg(StateNode):
    def start(self, event=None):
        Forward(50)
        Turn(120)
        self.parent.post_completion()
```

Use the State Machine Language

```
class TriangleLeg(StateNode):
  $setup {
      Forward(50) =C=> Turn(120) =C=>
          ParentCompletes()
  }
```

Constructors Are Only Called Once

Node constructors are only called once, when the state machine is being set up, not when the state machine is executing.

```
$setup {
    Turn(robot.pose.rotation.angle_z.degrees/2)
}
```

Put Dynamic Logic in start()

```
class TurnMore(Turn):
    def start(self, event=None):
        heading = self.robot.pose.rotation.angle_z.degrees
        self.angle = degrees(heading/2)
        super().start(event)
```

Bad Style: Spaghetti Code

```
$setup {
    rock: Say(“rock”)
    turn: Turn(180)
    and_roll: Say(“and roll”)

    rock =C=> turn =C=> and_roll
    turn =Tap=> and_roll
}
```

Proper Style: Group Outgoing Transitions

```
$setup {
    rock: Say(“rock”) =C=> turn

    turn: Turn(180)
    turn =C=> and_roll
    turn =Tap=> and_roll

    and_roll: Say(“and roll”)
}
```