Exotic Computing Project: Rotary elements to emulate a reversible FST

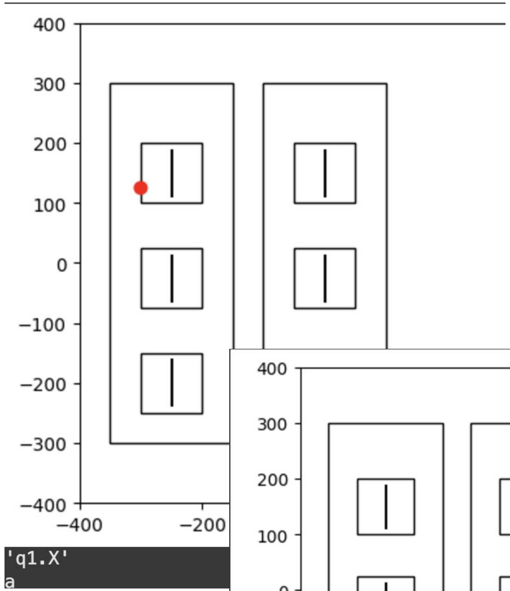
Brendan McGinn and Maggie Dempsey

Goal:

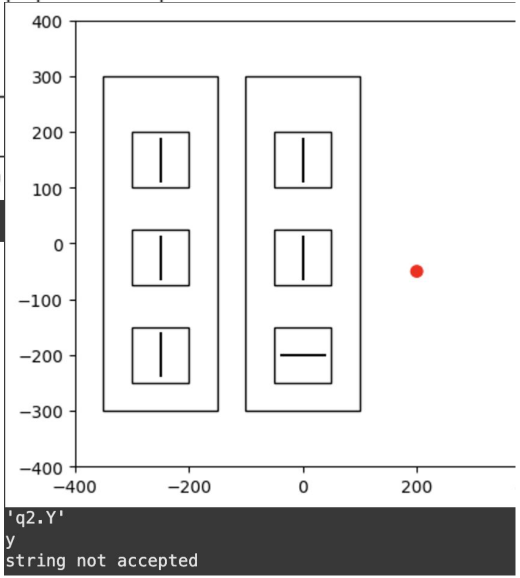
Create an emulation of a 2-state 2-input Reversible FST that:

- Accepts any string of the correct library
- Animates the RE process for the string
- Is entirely reversible

The idea came from the Reversible chapter in class. We wanted to find an interesting way to illustrate how reversibility operates, and REs provided a good tool.

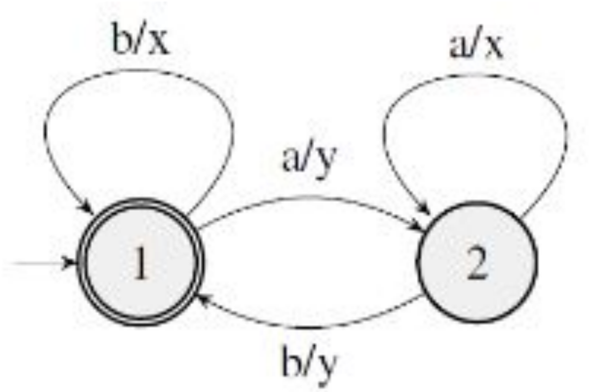


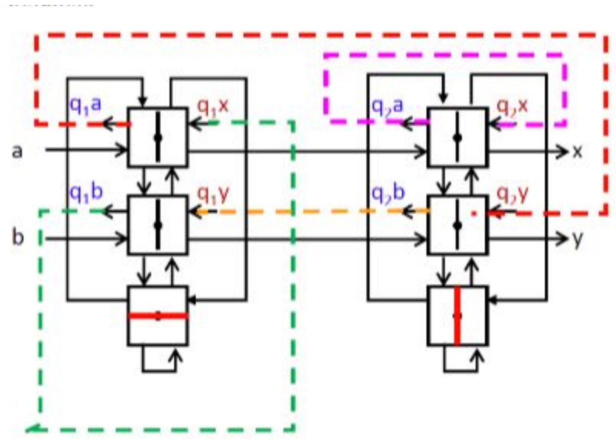
```
'q1.X'
a
```

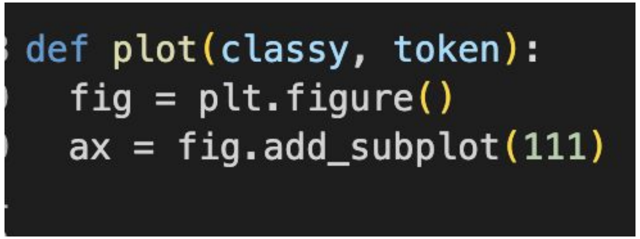


```
'q2.Y'
y
string not accepted
```

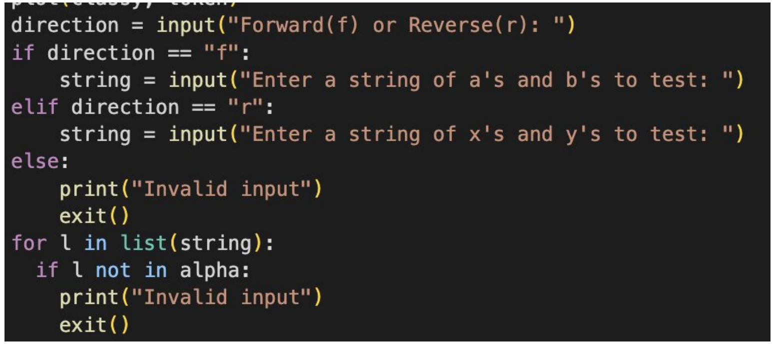
Reversible FST from HW 7



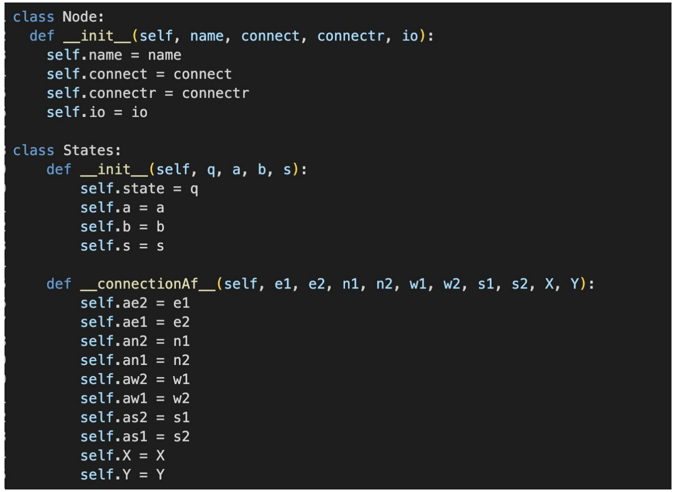




```python
def plot(classy, token):
  fig = plt.figure()
  ax = fig.add_subplot(111)
```



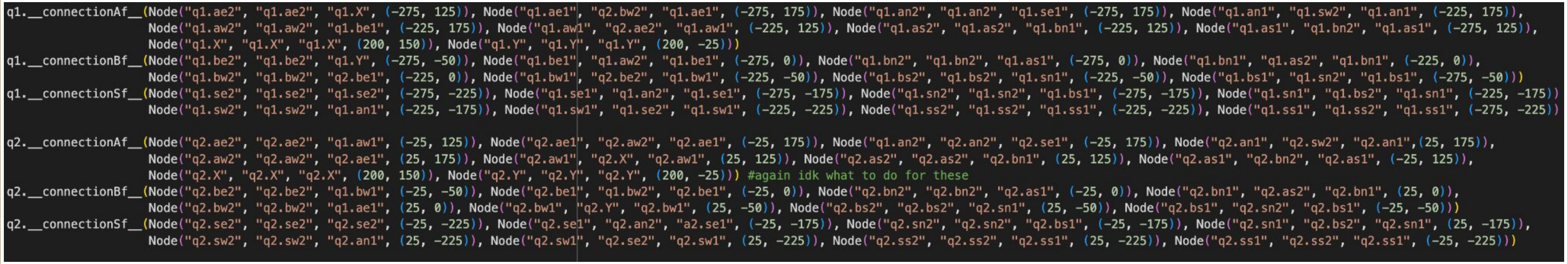
```python
plot(classy, token)
direction = input("Forward(f) or Reverse(r): ")
if direction == "f":
    string = input("Enter a string of a's and b's to test: ")
elif direction == "r":
    string = input("Enter a string of x's and y's to test: ")
else:
    print("Invalid input")
    exit()
for l in list(string):
  if l not in alpha:
    print("Invalid input")
    exit()
```



```python
class Node:
  def __init__(self, name, connect, connectr, io):
    self.name = name
    self.connect = connect
    self.connectr = connectr
    self.io = io

class States:
    def __init__(self, q, a, b, s):
        self.state = q
        self.a = a
        self.b = b
        self.s = s

    def __connectionAf__(self, e1, e2, n1, n2, w1, w2, s1, s2, X, Y):
        self.ae2 = e1
        self.ae1 = e2
        self.an2 = n1
        self.an1 = n2
        self.aw2 = w1
        self.aw1 = w2
        self.as2 = s1
        self.as1 = s2
        self.X = X
        self.Y = Y
```



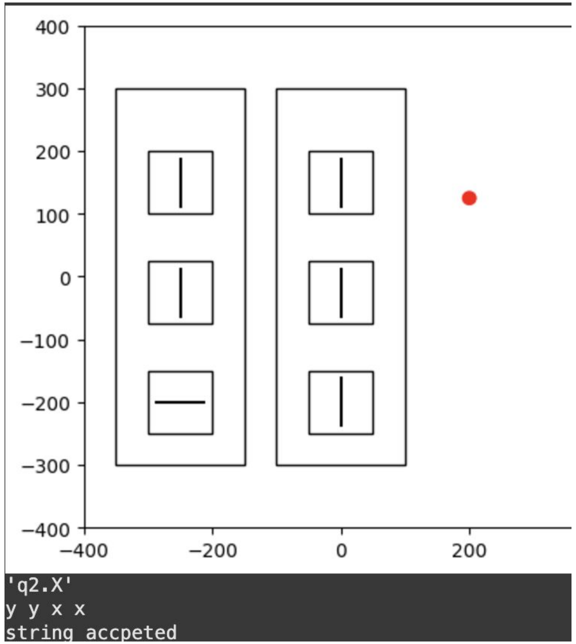
```python
q1.__connectionAf__(Node("q1.ae2", "q1.ae2", "q1.X", (-275, 125)), Node("q1.ae1", "q2.bw2", "q1.ae1", (-275, 175)), Node("q1.an2", "q1.an2", "q1.se1", (-275, 175)), Node("q1.an1", "q1.sw2", "q1.an1", (-225, 175)),
                    Node("q1.aw2", "q1.aw2", "q1.be1", (-225, 175)), Node("q1.aw1", "q2.ae2", "q1.aw1", (-225, 125)), Node("q1.as2", "q1.as2", "q1.bn1", (-225, 125)), Node("q1.as1", "q1.bn2", "q1.as1", (-275, 125)),
                    Node("q1.X", "q1.X", "q1.X", (200, 150)), Node("q1.Y", "q1.Y", "q1.Y", (200, -25)))
q1.__connectionBf__(Node("q1.be2", "q1.be2", "q1.Y", (-275, -50)), Node("q1.be1", "q1.aw2", "q1.be1", (-275, 0)), Node("q1.bn2", "q1.bn2", "q1.as1", (-275, 0)), Node("q1.bn1", "q1.as2", "q1.bn1", (-225, 0)),
                    Node("q1.bw2", "q1.bw2", "q2.be1", (-225, 0)), Node("q1.bw1", "q2.be2", "q1.bw1", (-225, -50)), Node("q1.bs2", "q1.bs2", "q1.sn1", (-225, -50)), Node("q1.bs1", "q1.sn2", "q1.bs1", (-275, -50)))
q1.__connectionSf__(Node("q1.se2", "q1.se2", "q1.se2", (-275, -225)), Node("q1.se1", "q1.an2", "q1.se1", (-275, -175)), Node("q1.sn2", "q1.sn2", "q1.bs1", (-275, -175)), Node("q1.sn1", "q1.bs2", "q1.sn1", (-225, -175))
                    Node("q1.sw2", "q1.sw2", "q1.an1", (-225, -175)), Node("q1.sw1", "q1.se2", "q1.sw1", (-225, -225)), Node("q1.ss2", "q1.ss2", "q1.ss1", (-225, -225)), Node("q1.ss1", "q1.ss2", "q1.ss1", (-275, -225))

q2.__connectionAf__(Node("q2.ae2", "q2.ae2", "q1.aw1", (-25, 125)), Node("q2.ae1", "q2.aw2", "q2.ae1", (-25, 175)), Node("q1.an2", "q2.an2", "q2.se1", (-25, 175)), Node("q2.an1", "q2.sw2", "q2.an1",(25, 175)),
                    Node("q2.aw2", "q2.aw2", "q2.ae1", (25, 175)), Node("q2.aw1", "q2.X", "q2.aw1", (25, 125)), Node("q2.as2", "q2.as2", "q2.bn1", (25, 125)), Node("q2.as1", "q2.bn2", "q2.as1", (-25, 125)),
                    Node("q2.X", "q2.X", "q2.X", (200, 150)), Node("q2.Y", "q2.Y", "q2.Y", (200, -25))) #again idk what to do for these
q2.__connectionBf__(Node("q2.be2", "q2.be2", "q1.bw1", (-25, -50)), Node("q2.be1", "q1.bw2", "q2.be1", (-25, 0)), Node("q2.bn2", "q2.bn2", "q2.as1", (-25, 0)), Node("q2.bn1", "q2.as2", "q2.bn1", (25, 0)),
                    Node("q2.bw2", "q2.bw2", "q1.ae1", (25, 0)), Node("q2.bw1", "q2.Y", "q2.bw1", (25, -50)), Node("q2.bs2", "q2.bs2", "q2.sn1", (25, -50)), Node("q2.bs1", "q2.sn2", "q2.bs1", (-25, -50)))
q2.__connectionSf__(Node("q2.se2", "q2.se2", "q2.se2", (-25, -225)), Node("q2.se1", "q2.an2", "a2.se1", (-25, -175)), Node("q2.sn2", "q2.sn2", "q2.bs1", (-25, -175)), Node("q2.sn1", "q2.bs2", "q2.sn1", (25, -175)),
                    Node("q2.sw2", "q2.sw2", "q2.an1", (25, -225)), Node("q2.sw1", "q2.se2", "q2.sw1", (25, -225)), Node("q2.ss2", "q2.ss2", "q2.ss1", (25, -225)), Node("q2.ss1", "q2.ss2", "q2.ss1", (-25, -225)))
```

Testing

Here are a few of the example inputs we tested:

1. ab -> yy accepted
2. aaaa -> yxxx, not accepted
3. abbb -> yyxx, accepted
4. ababababababa -> yyyyyyyyyyyyyy, not accepted
5. yyxx -> baaa
6. xxxxxxxxxyyyy -> bbbbbbbbbabab



```
'q2.X'
y y x x
string accpeted
```

DEMO

https://colab.research.google.com/drive/1SjJFHVrh3y-bC4QJfrQtwWXK19e2PCFM?usp=sharing

Lessons

1. Plan out farther in advance - we wanted to get started right away, but we rushed into it a little. We should have planned what libraries to use a little better, as well as plan our architecture. We had to go back and redo parts of this because they were not planned well, which added unnecessary time.
2. Communication - we had slightly different visions for what the project was going to be at the beginning, so at first our code did not match up very well.
3. Encompassing design - we tried to first design how we were going to get the forward part of the FST working without thinking too much about reverse. This led us to doing extra work to get reverse, before realizing we could just add an attribute to the nodes and use a similar if statement.
4. How REs work - both of us were interested in exactly how the rotary elements worked with FSTs, which is why we chose the project, but did not completely grasp the concept. Working with them showed us how they can be used for entire strings rather than only individual characters.

Future improvements:

- Smoother graphics
- Faster graphing time
- Add States and Inputs

Github

https://github.com/bmcginn2/exoticwork/tree/main/project