

Replace method

★ Every rule should have a unique find string.

Can't have: $[[\text{'F'}, \text{'FF'}], [\text{'F'}, \text{'+F'}]]$
need to be different

★ L System string: 'F+F'

Rules: $[[\text{'F'}, \text{'FF'}]]$

Start 'F+F'
↳ 'FF+F'

now I move on to next character (+).

$\Rightarrow$ if I don't move on:

$F + F$
 $FF + F$
 $FFF + F$
 $FFFF + F$
 $\vdots$

Don't want to
do this

Only apply one rule per character in string,
then I move on.

Cher
↓
Curr string = 'F+F+F'
Self.rules = $\left[\underbrace{[-', 'F+F']}_{\text{rule}}, \underbrace{['F', 'FF']}_{\text{rule}} \right]$

```
def replace(self, currString): 'F++F++F'  
    newString = ''
```

```
    for char in currString:
```

```
        # check to see if char matches
```

```
        # a find string in rules
```

```
        found = False
```

```
        for rule in self.rules:
```

```
            # loop thru rules
```

```
            if char == rule[0]:
```

```
                # do we have a match?
```

```
                newString += rule[1]
```

```
                found = True
```

```
                break
```

```
        if not found:
```

```
            newString += char
```

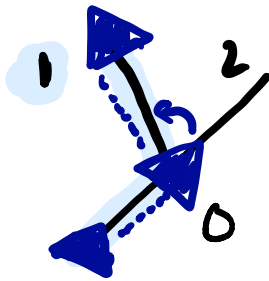
```
    return newString
```

find str

replace str

Branching when drawing trees

Let's say we're drawing a tree branch 1



we're stuck!

how do we get back to the main branch and continue onto branch 2?

goto!

But goto where? At what heading?

Before we leave 0 and go up to 1, we should save our state (position + heading).

Then after drawing 1, we can restore the state, going to 0, before we hit a dead end.

then continue drawing branch 2.

In Summary:

Before embarking down a branch, save position + heading, then restore this state after drawing the branch

Implement with 2 new L-system symbols:

- [Save current state (heading + position)
-] Restore previous saved state (heading + position)

L system string for

"F[+F]F"

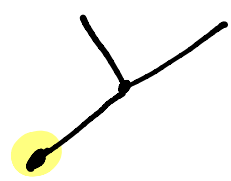
↑ ↑
save restore
go along go along
1 2



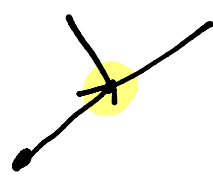
How to implement [and] symbols to save state?

Stack the information in a list:

Stack = []



Stack.append(headng)
Stack.append(position)



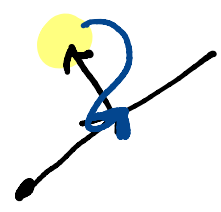
Stack: [headng, position]

prevPos = Stack.pop()

Stack: [headng]

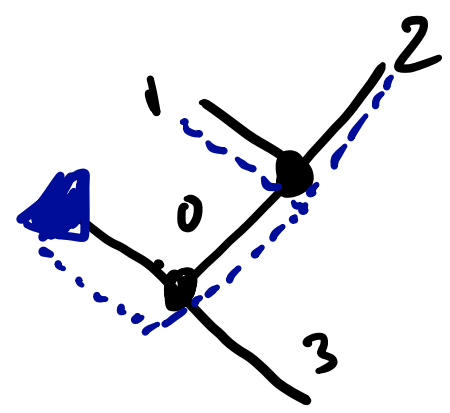
prevHead = Stack.pop()

Stack = []



* If we append a position after headng,
we need to pop position before headng

Practice: What is the L-system string to draw this?



" F [+ F [+ F] F] F
 ↑ ↑ ↑ ↑ ↑
 Saul 0 Saul 1 2 3

L-system project note:

Why do we **loop char-by-char** to apply replacement rules — and not use **replace String method?**

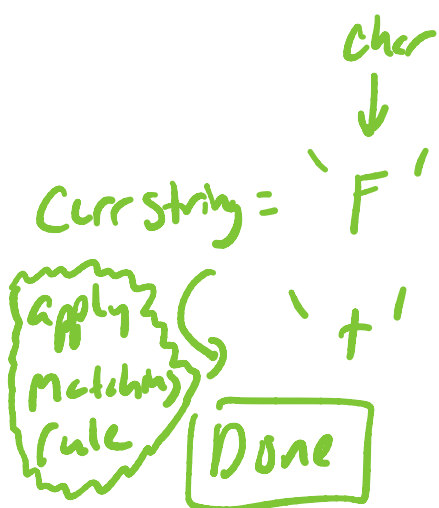
⇒ If there are >1 rule, results will be different [in undesirable way]

Simple example

base = 'F'

self.rules = [['F', '+'], ['+', '-']]

Loop char-by-char



String replace() on rules one-by-one

curr string = 'F'

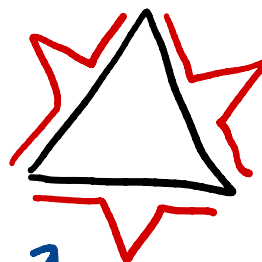
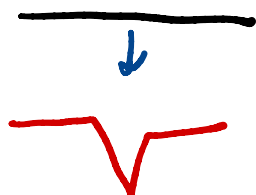
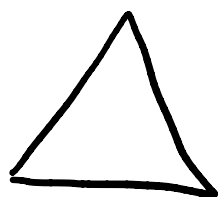
char
↓

replace('F', '+') ('+')

replace('+', '-') ('-')

Done

★ Key: replace may change the result of
previously applied rules mid-iteration
⇒ Not what we want!



Don't want to modify 
mid-iteration — let it be!