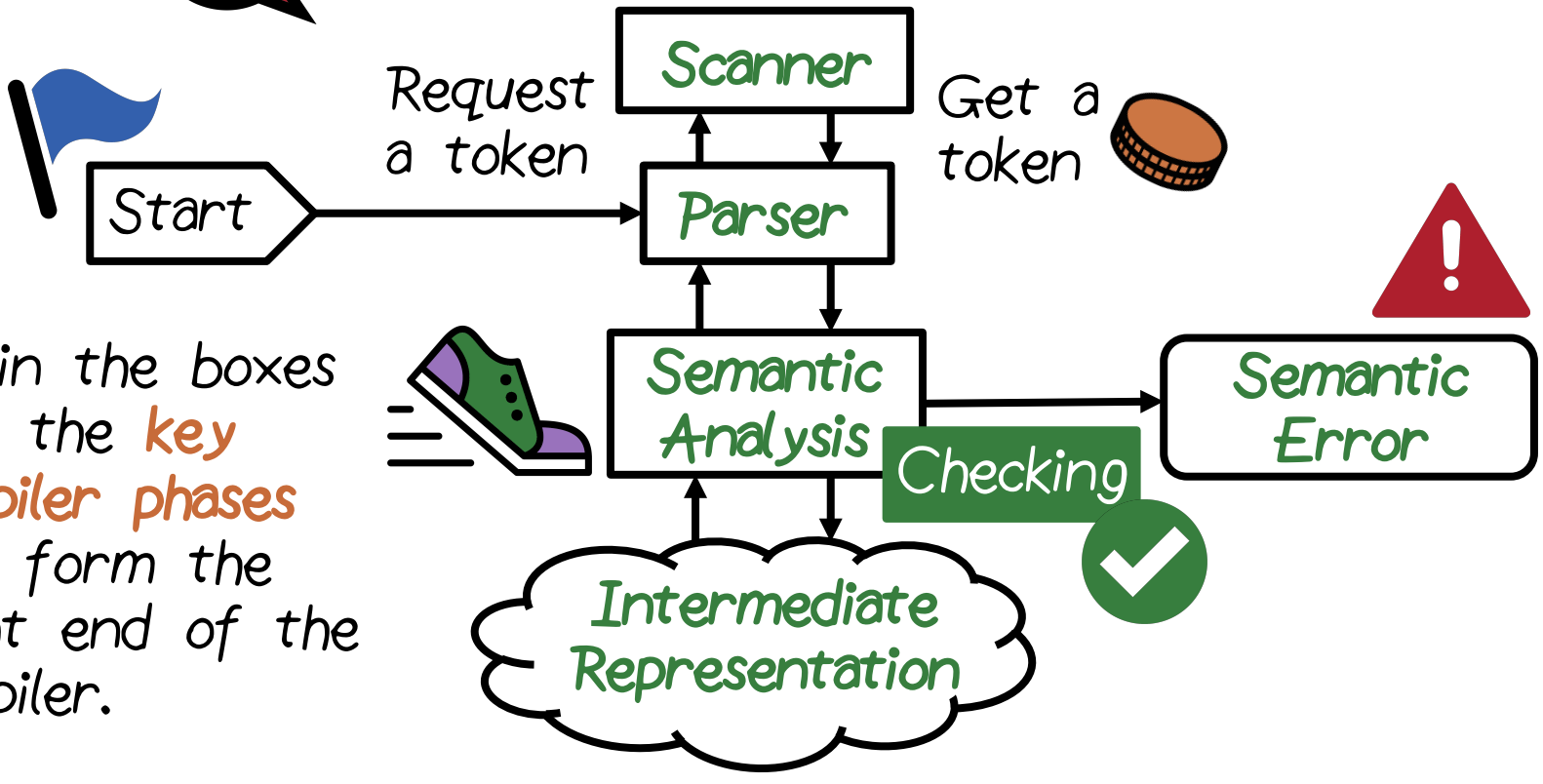
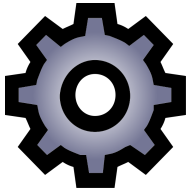
# Review Compilers Quiz

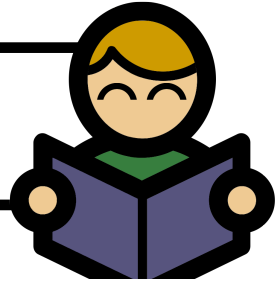



Fill in the boxes with the key compiler phases that form the front end of the compiler.



# Lexical Analysis (Scanner)

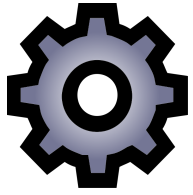
Read input: one character at a time



Group characters into tokens

Remove whitespaces and comments

Encode token types and form tuples
<token-type, value> and return to parser



# Lexical Analysis (Scanner)

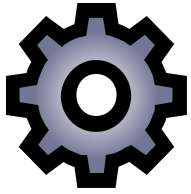
Encode token types (tuples) and return to parser

| | type | value |
|---|---|---|
| 123.45 → | FloatConst | 123.45 |
| DaysofWeek → | VAR | String = DaysOfWeek |
| + → | Operator | Value = + |

# Lexical Analysis (Scanner)


Lexical language → A collection (set) of legal strings

Lexical rules → How to form legal strings

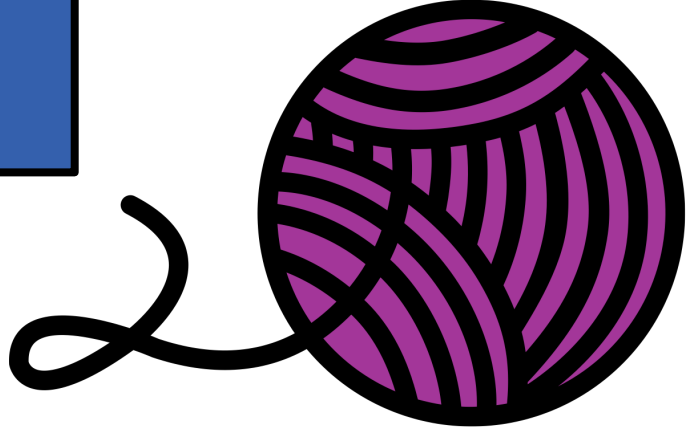


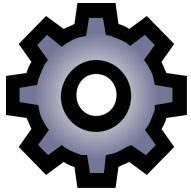
# Representations of Strings

Regular expression r (a pattern of characters) and its language L(r)

Used in many Unix programs (e.g., grep, vi., etc.)

State machine (finite automata)





# Representations of Strings

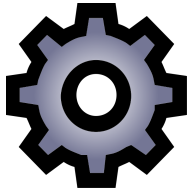
## Basics of Regular Expression

$\Sigma =$ AaBbCcDdEeFfGgHhIiJjKkLlMmNnOoPpQqRrSsTtUuVvWwXxYyZz

1234567890~!@#$%^&*()_+=-{}|[]:";'<>?,./

**Symbols & alphabet**

- A **symbol** is a valid character in a language
- **Alphabet** is set of legal symbols

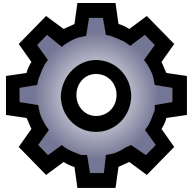


# Representations of Strings

## Basics of Regular Expression

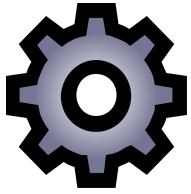
Metacharacters/metasymbols that have special meanings:

- Defining reg-ex operations (e.g. |, (, ), *, +, etc.)
- Escape character (\) to turn off special meanings
- Empty string ε and empty set ϕ={}



# Representations of Strings

| Term | Definition |
|---|---|
| **Basic Regular Expression** | Single characters (a={a} for any character a$\in\Sigma$),<br>Empty string ε={ε},<br>Empty set ϕ |
| **Basic regular expression operations** | Union, denoted by a \| b,<br>Concatenation, denoted by ab<br>Repetition (Kleen closure, 0 or more times) a* |
| **Notation** | Boldface a denotes language {a}. |

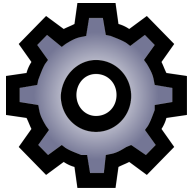


# Representations of Strings

## Precedence of operations

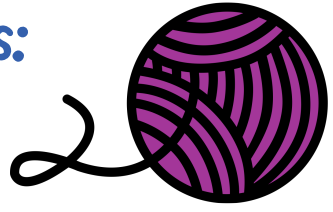

Repetition > Concatenation > Alternation

(Parentheses can be used to change precedence)



# Representations of Strings

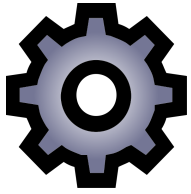
## Examples of regular expressions:



**a | b*** = { a, ε, b, bb, bbb, …}

**(a | b)*** = {ε, a, b, ab, ba, aa, bb, aaa, aab, …} = Any number (including zero) of a's and b's in any order

**(a|c)*b(a|c)*** = Any number of a's and c's (including zero) in any order followed by a single b followed by any number of a's and c's (including zero) in any order

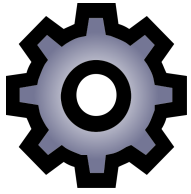


# Unix-Style Regular Expressions

| Symbol | Definition |
|---|---|
| . | denotes any character in alphabet |
| [ ] | character class, allowing range and complement |

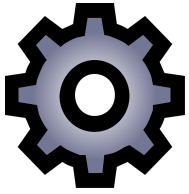
**[a-d] same as [abcd]**

**[^1-3] denotes characters other than 1, 2, or 3**



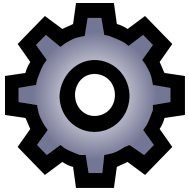
# Unix-Style Regular Expressions

| Symbol | Definition |
|---|---|
| . | denotes any character in alphabet |
| [ ] | character class, allowing range and complement |
| + | repeating one or more times |
| ? | optional (zero or one time) |
| ^ and $ | beginning and end of line |



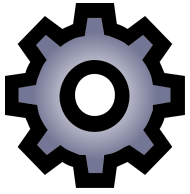
# Regular Expressions: Examples

| Expression | Outcome |
| --- | --- |
| [a-zA-Z_][a-zA-Z_0-9]* | Identifiers starting with lower case or upper case letters or underscore followed by zero or more of upper or lower case letter or underscore or digits |
| [0-9][0-9]* | unsigned integers |
| (+\|-)?[0-9][0-9]* | signed integers with optional sign |



# Regular Expressions: Examples (Continued)

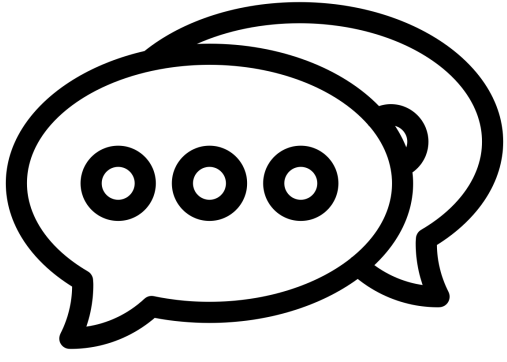
| Expression | Outcome |
|---|---|
| [+-]?[0-9]+ (\.[0-9]*)? ([eE][+-]?[0-9]+)? | floating point numbers - various possibilities - some are as follows:<br>9<br>+8<br>5.8<br>5e-88<br>… |
| [^aeiouAEIOU] | Match any character not a vowel |
| [b-df-hj-np-tv-zB-DF-HJ-NP-TV-Z] | Match any uc or lc consonant |



# Regular Expressions

Regular Expressions are used in grep, sed, awk, perl, vi, shells, lex, yacc

Each may use slightly different convention



# RegEx Quiz

Write a regular expression consisting of 0's and 1's which may have a 0 but whenever it occurs it must be followed by a 1, empty string ok:

(1 | 01)*

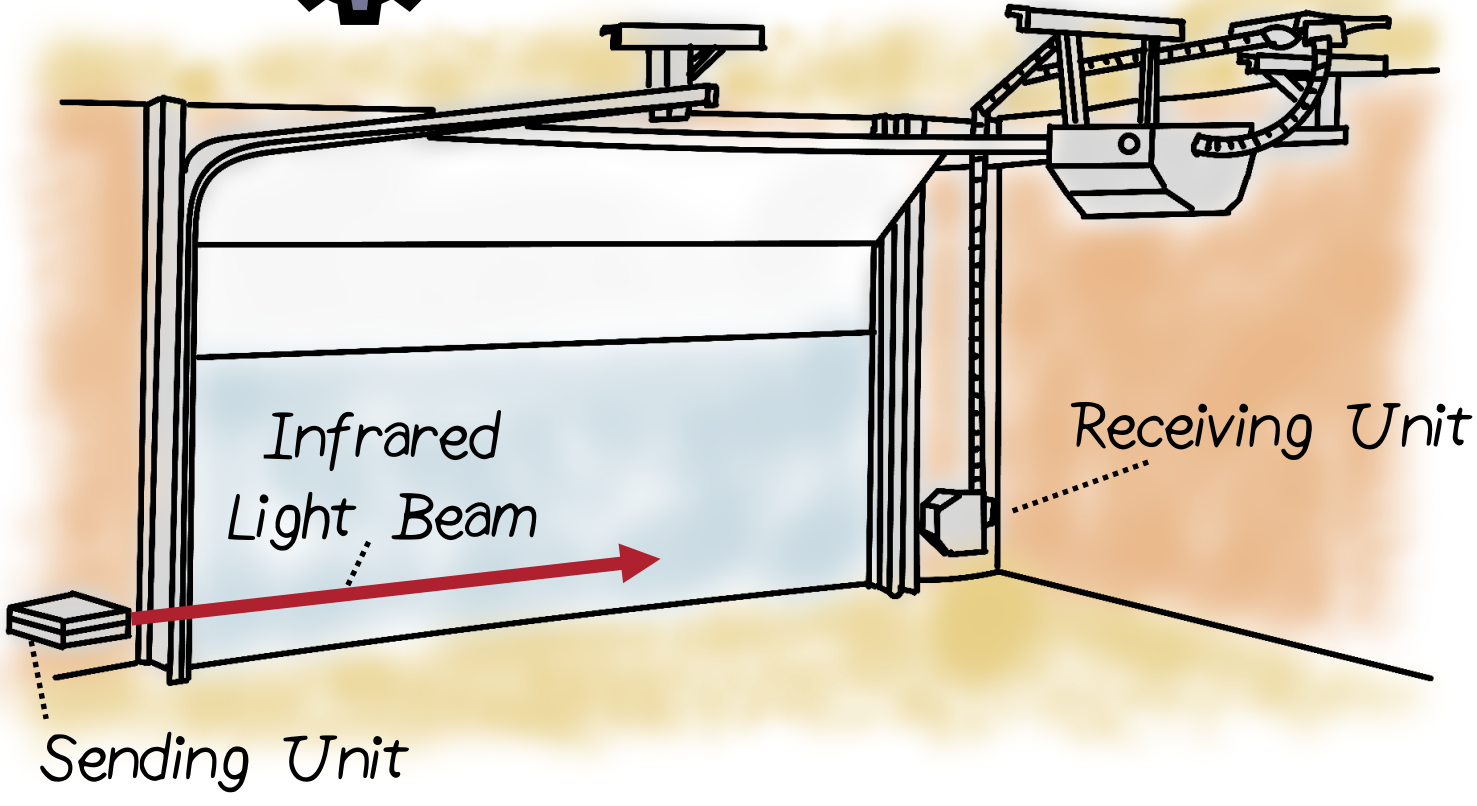
An identifier that starts with a lowercase letter or underscore and followed by one or more of lowercase letters and digits and underscores. A letter or a digit must follow an underscore:

(L | U) (L | D | U (L | D)) +, where L ~ lower case letter, U ~ underscore and D ~ digits

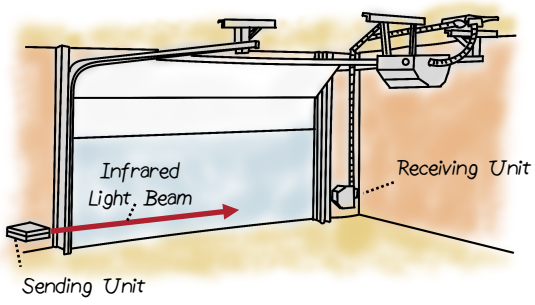
A string that consists of at least 3 consecutive 0's:

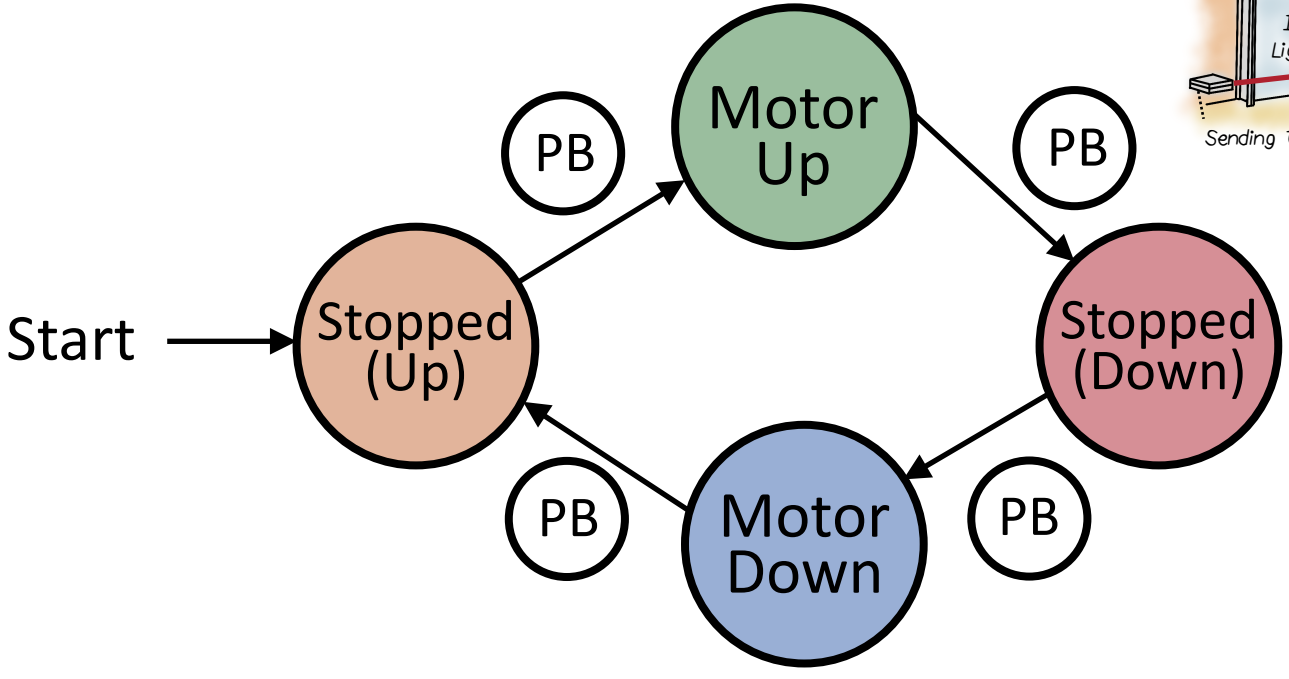
(0 | 1)* 000 (0 | 1)*

# State Machines

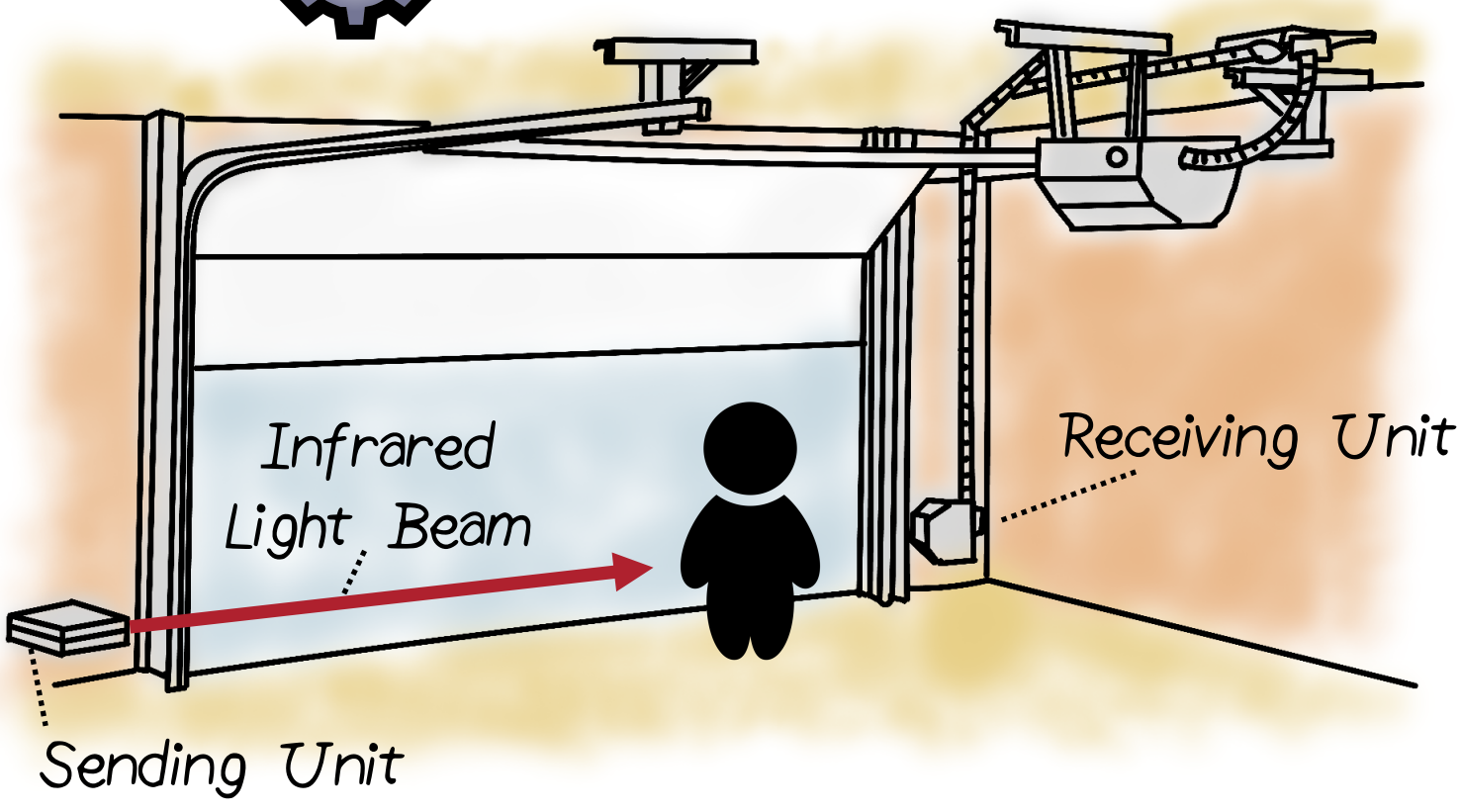


# State Machines

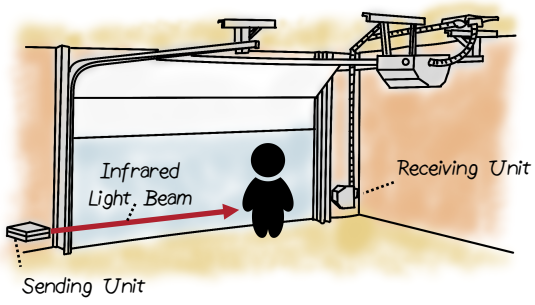


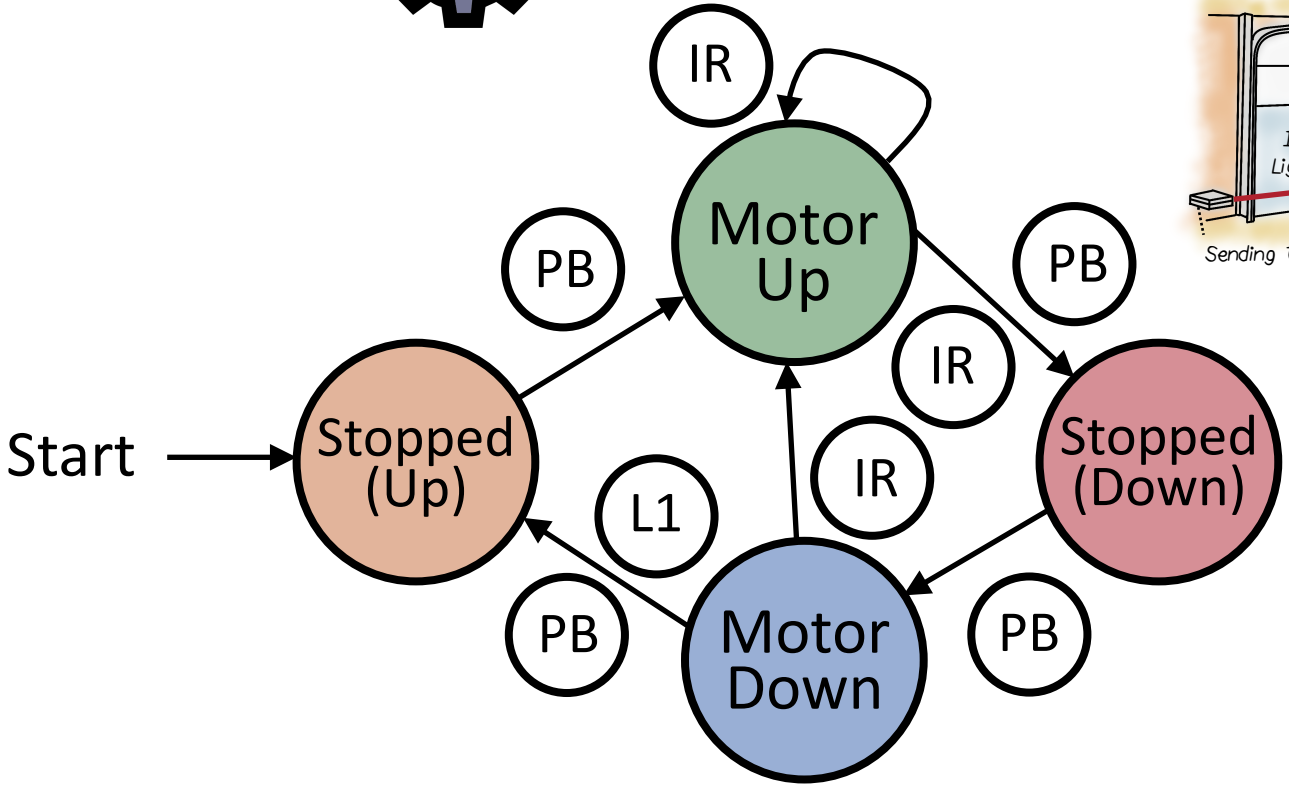



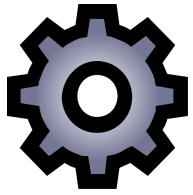
# State Machines



# State Machines





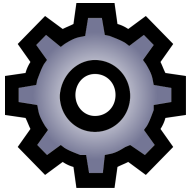
# Deterministic Finite Automata

A simplest model for computing

**Deterministic:** Machine is in a state. Upon receipt of a symbol will go to a unique state.

**Finite:** Have a finite number of states

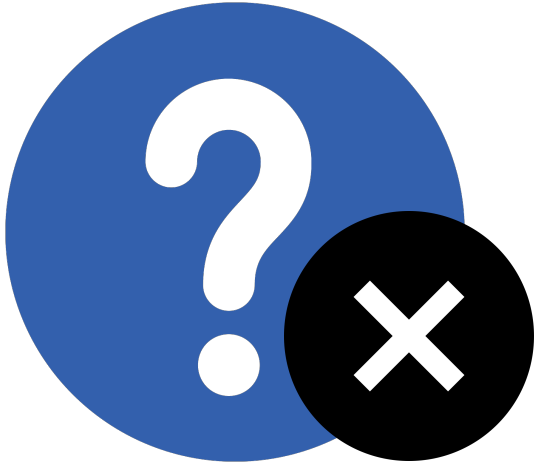
**Automata:** (pl. of automaton) Self-operating machine

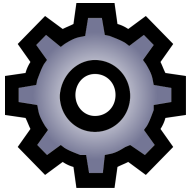


# Deterministic Finite Automata

DFA

Finite-state machine without ambiguity

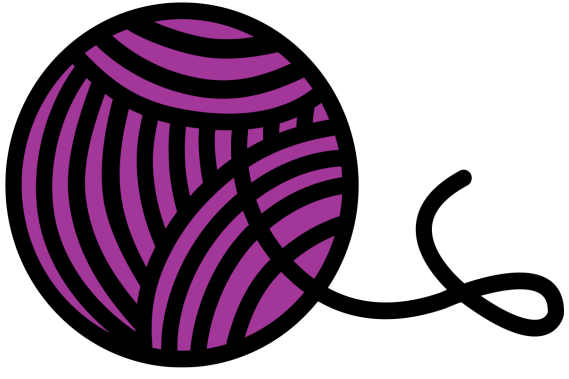




# Deterministic Finite Automata

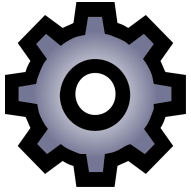
## DFA and Strings

DFA can recognize strings



String is **input**

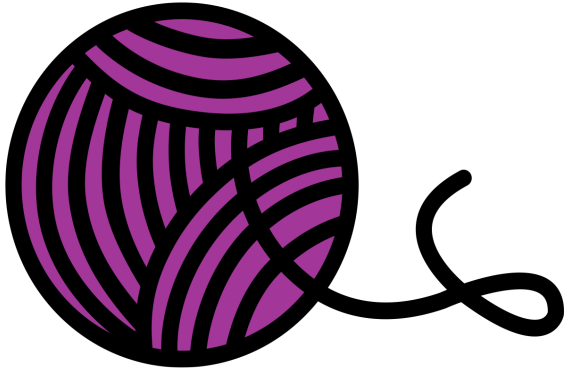
If DFA ends at **accept state**, string is **recognized**



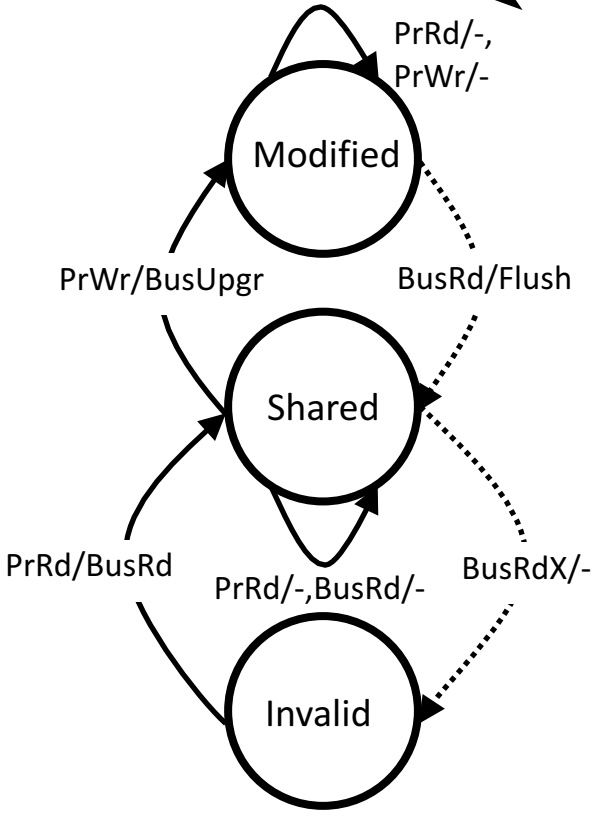
# Deterministic Finite Automata

## DFA and Strings

A language is called a **regular language** if some finite automaton **recognizes** it



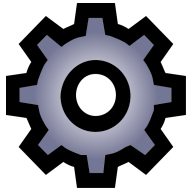
# State Machine Quiz




Fill in the blanks with Modified, Shared, or Invalid.

When in the modified state, if a BusRd command is detected, the machine will go to state: shared

When in the Invalid state, if a BusRd command is detected, the machine will go to state: modified



# DFA Examples: Example 1

The **alphabet** is: {0,1}

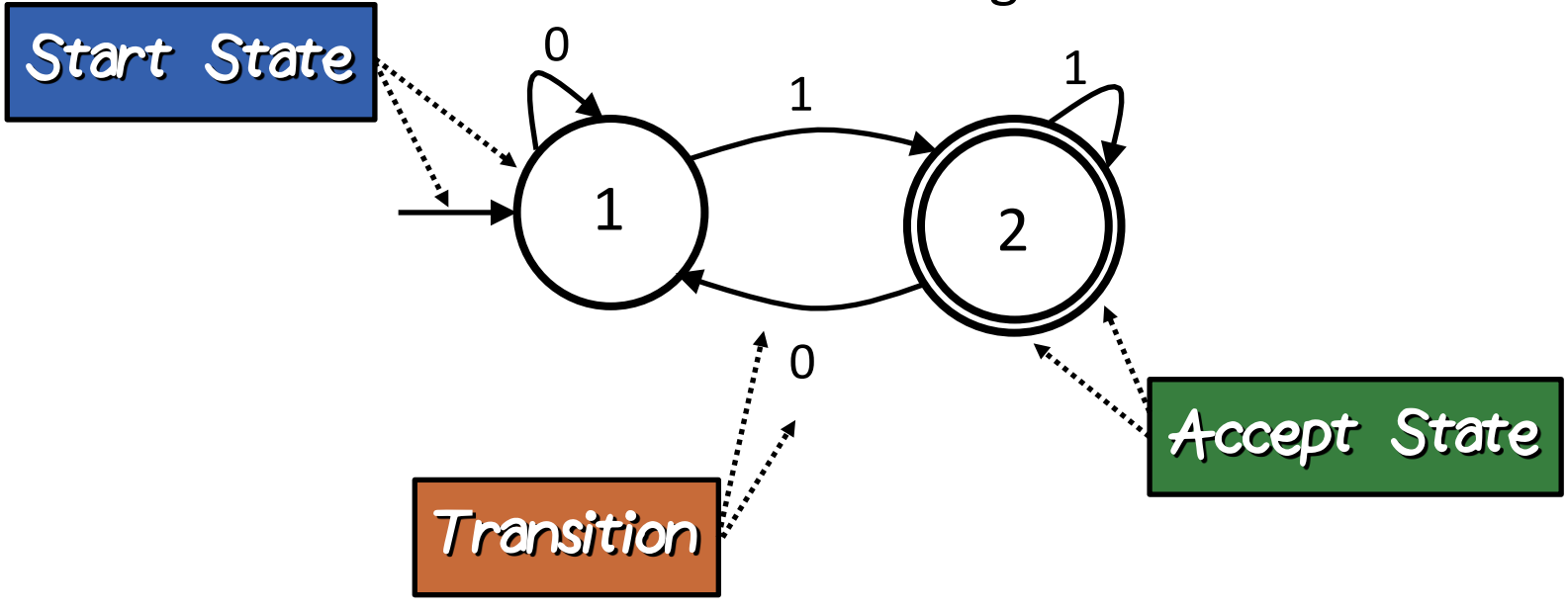
The **mission** is: Accept all strings that end in 1

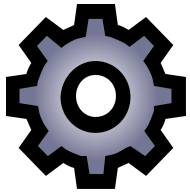
DFA = Deterministic Finite Automata (DFA)

# DFA Examples: Example 1

The **alphabet** is: {0,1}

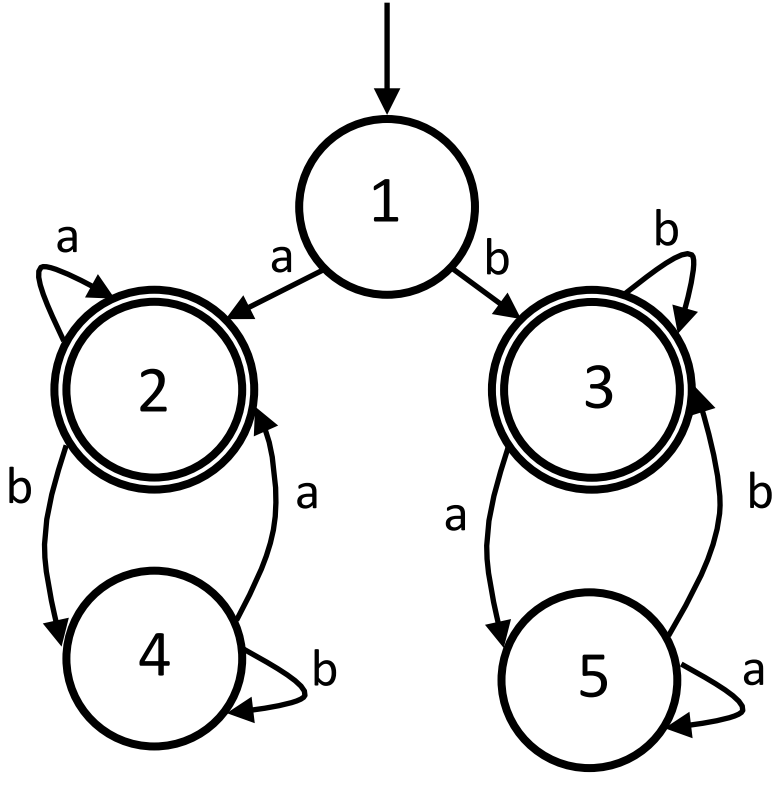
The **mission** is: Accept all strings that end in 1





# DFA Examples: Example 2

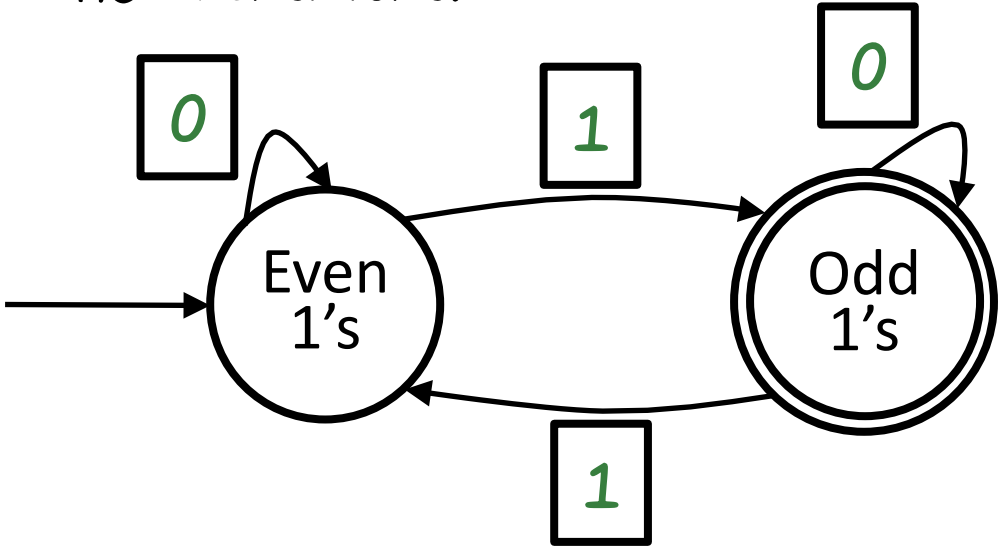
**Mission**: Accept strings of 'a's and 'b's that begin and end with same symbol



# DFA Odd Ones Quiz

*Fill in the values for the transitions.*

**Alphabet**: {0.1}
**Mission**: Accept strings with an odd number of ones.

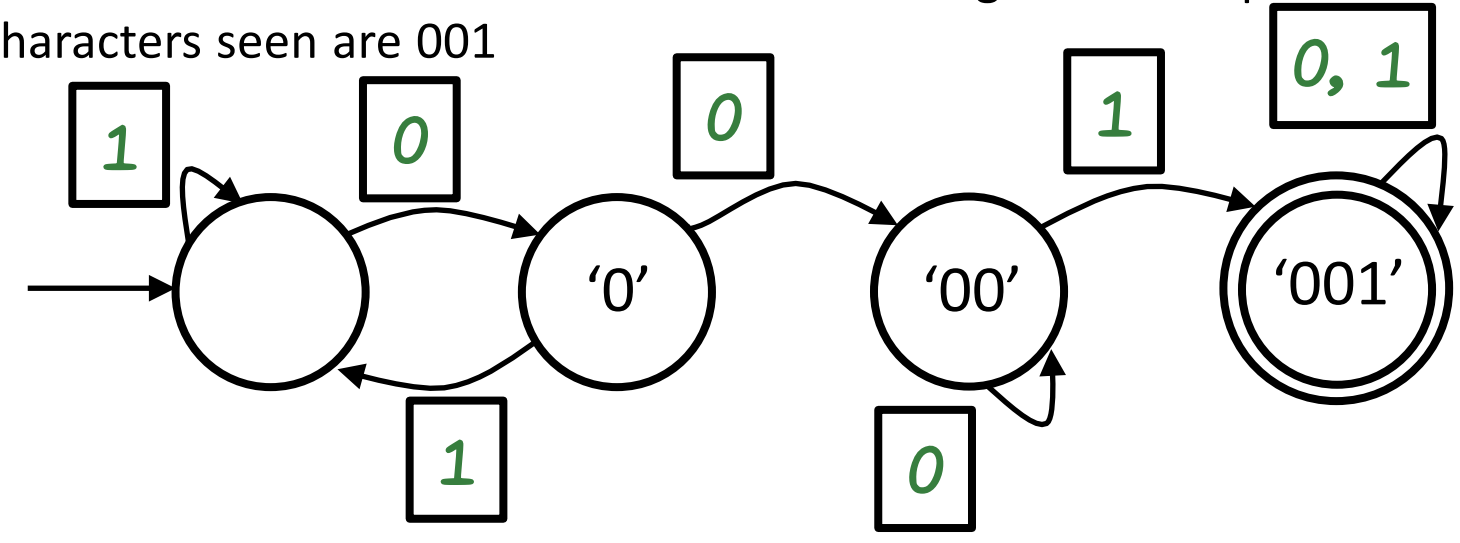


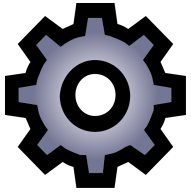
# DFA Substring Quiz

*Fill in the values for the transitions.*

**Alphabet**: {0,1}  **Mission**: Accept strings containing '001'

**Hint**: State '0' designates last character seen is '0', similarly : '00' designates last two characters seen are 00 and '001' designates or captures that last 3 characters seen are 001





# Formal Definition of DFA

A DFA consists of:

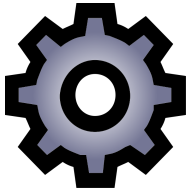
**Alphabet**: $\Sigma$

**A set of states**: $Q$

**A transition function** $\delta$ : $Q \times \Sigma \rightarrow Q$

**One start state**: $q_0$

**One or more accepting states**: $F \subseteq Q$



# Formal Definition of DFA

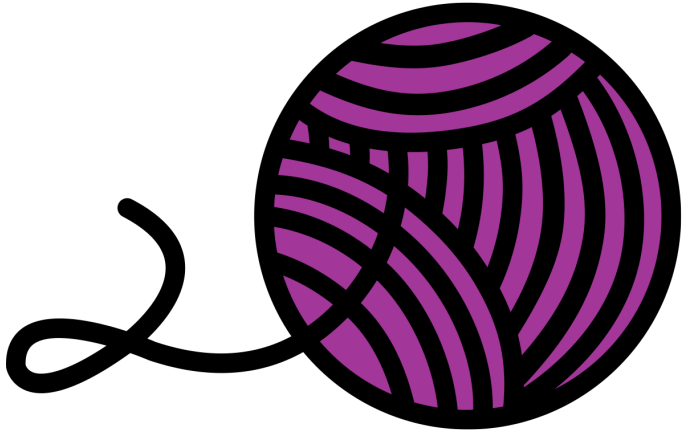
Language accepted by a DFA is the set of strings such that DFA ends at an accepting state

- **Each string** is $c_1c_2...c_n$ with $c_i \in \Sigma$
- **States** are $qi = \delta(q_{i-1},c_i)$ for $i=1...n$
- $q_n$ is an **accepting state**

# DFA Quiz

Can DFA's be designed to accept any string?

☐ Yes ☑ No



# DFA String Recognition Quiz

Select the strings that a DFA be designed to detect.

- [ ] strings that start out with k zeros followed by k ones.
- [ ] strings with an equal number of ones and zeros.
- [x] strings with an equal number of strings "01" and "10".

# DFA String Recognition Quiz