Bash Scripting, Globs, and Regular Expressions

and grep

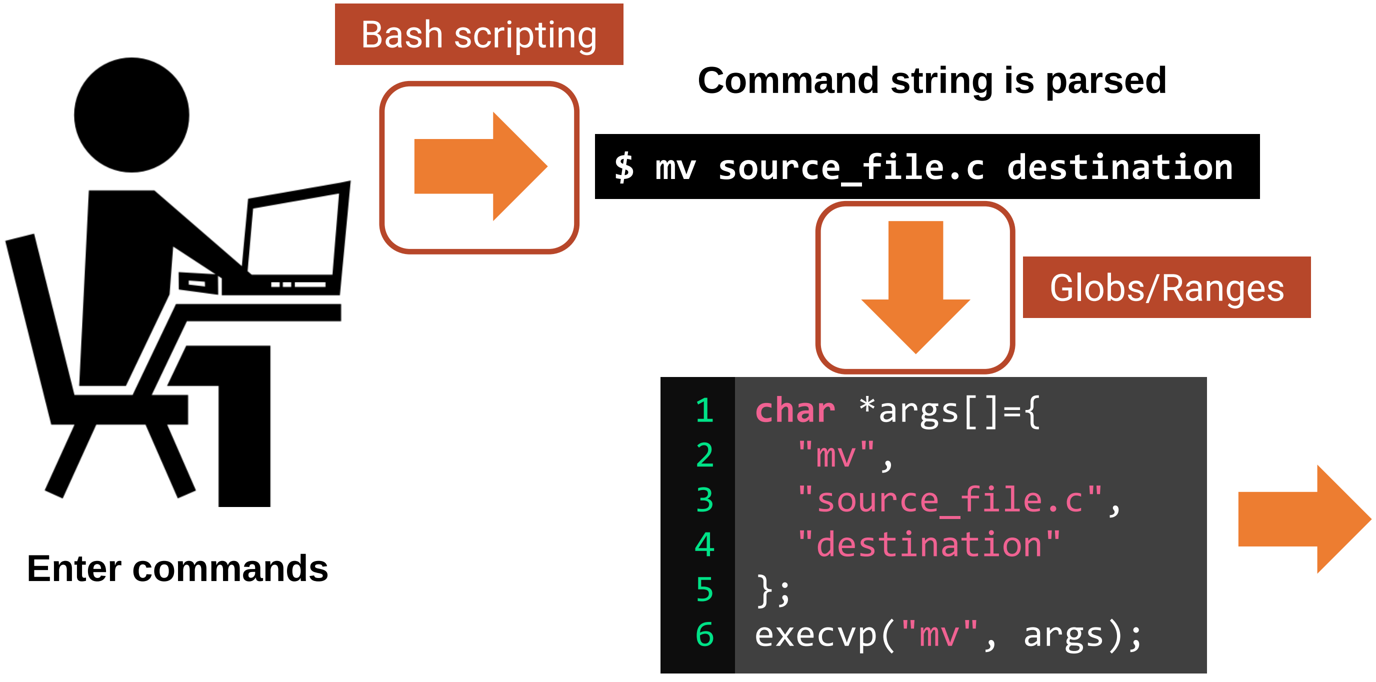
and sed

and more?

Labs

- **Forcelab** and **Zombielab** are out!
- You will have two weeks to do the next three labs
- They cover a lot of material

What does a shell do?



```
$ mv source_file.c destination
```

```
char *args[]={
  "mv",
  "source_file.c",
  "destination"
};
execvp("mv", args);
```

Enter commands

Bash loads the arguments and executes the command

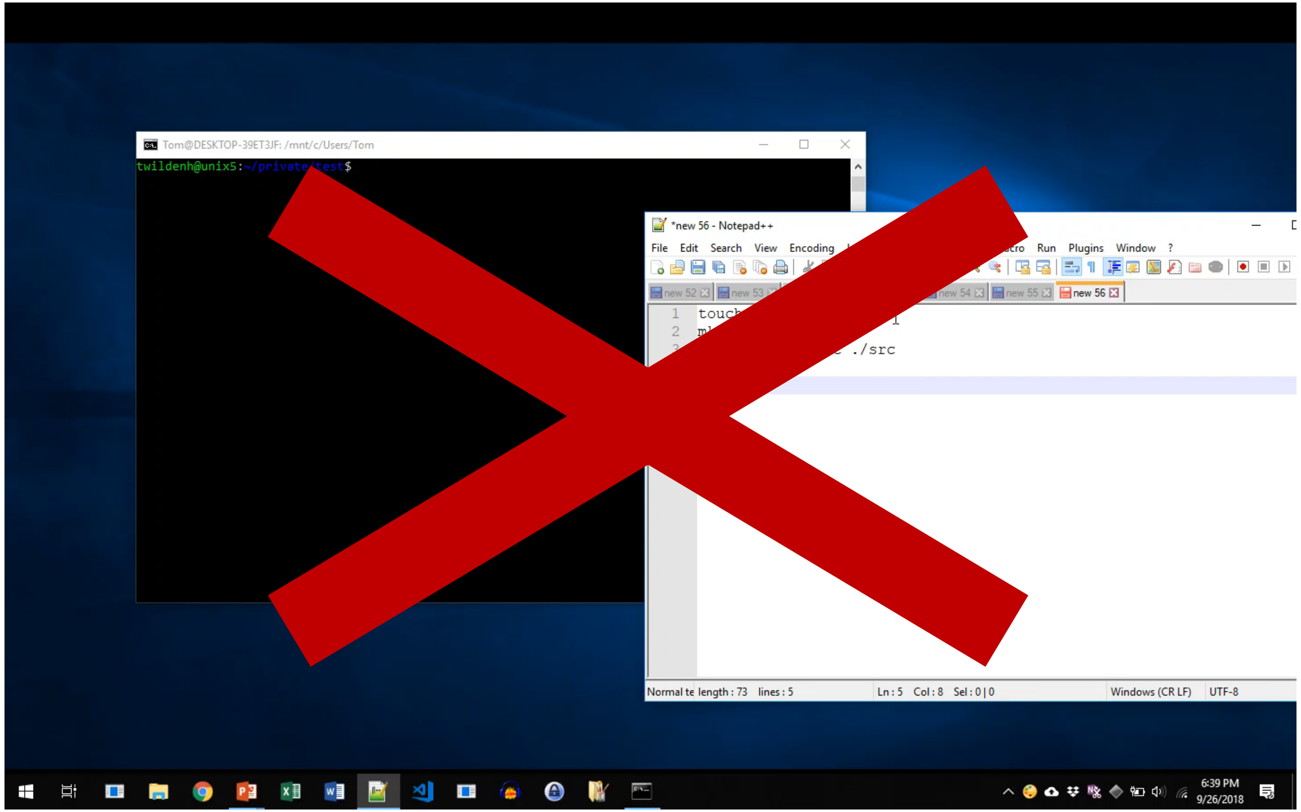
Command Review

Task	Command
Show directory contents	`ls`
Change directory	`cd directory_name`
Move a file	`mv file.txt location`
Copy a file	`cp file.txt copy.txt`
Rename a file	`mv file.txt renamed.txt`
Execute a binary	`./binary_file`
Print something	`echo "Hello World"`

Bash scripting

- Sometimes you run the same set of commands many times.
- Retyping commands isn’t very much fun
  - Unless you like typing
  - I don't like typing
- There’s a simple solution…

Solution?



```
twildenh@unix5:~/private/test$
```

```
touch
ml
./src
```

Normal te length : 73 lines : 5 Ln : 5 Col : 8 Sel : 0 | 0 Windows (CR LF) UTF-8

<-- Don’t do this.

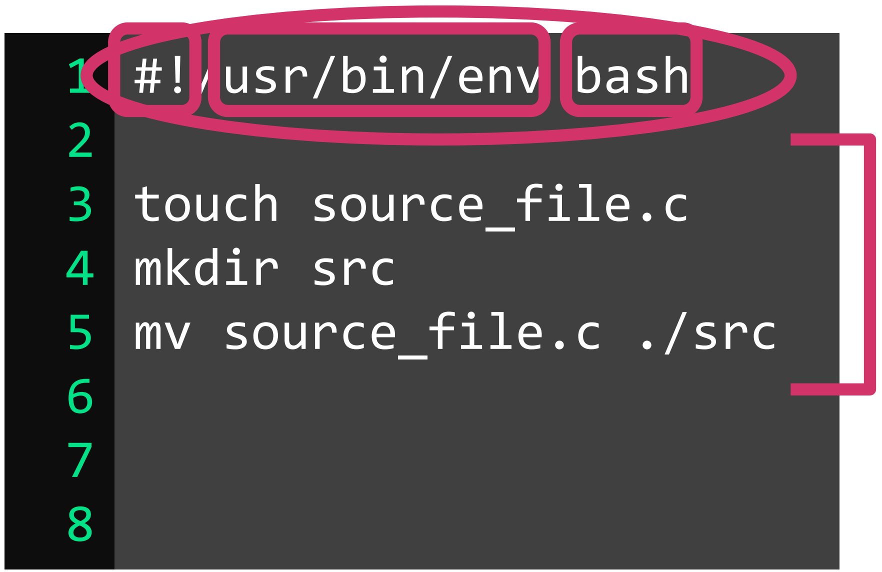
Better solution: Bash scripting

- Lets you run a sequence of commands from a file
- Can be executed like a binary file

```bash
#!/usr/bin/env bash

touch source_file.c
mkdir src
mv source_file.c ./src
```

Example



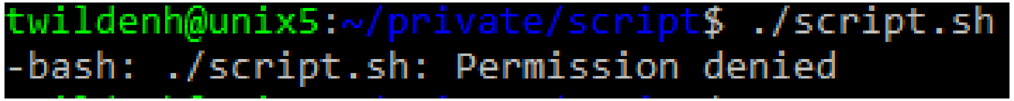
```
#!/usr/bin/env bash

touch source_file.c
mkdir src
mv source_file.c ./src
```

bash_script.sh

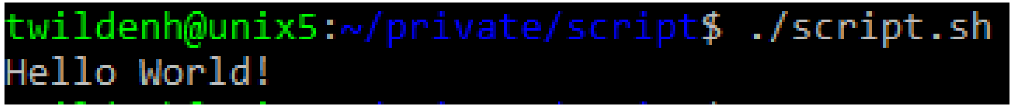
chmod

- Files are not executable by default



```
twildenh@unix5:~/private/script$ ./script.sh
-bash: ./script.sh: Permission denied
```

- Have to add executable permission
    - `chmod +x script.sh`
- Then we can run the script

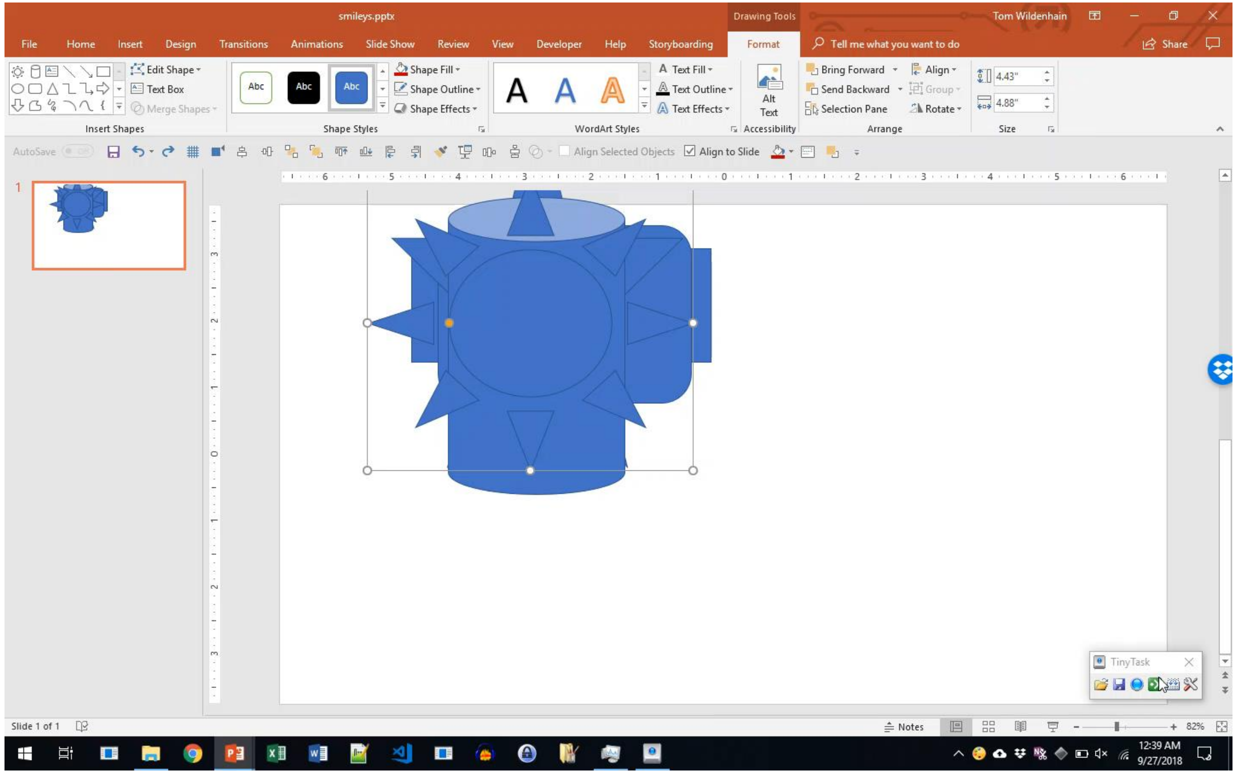


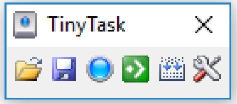
```
twildenh@unix5:~/private/script$ ./script.sh
Hello World!
```

Bash as a programming language

- Bash also supports commands for conditionals, loops, and variables
- Automation is one of the key advantages of using a terminal

Great Impractical Ideas: Automation with TinyTask





Bash scripting summary

- Bash scripts end in a .sh extension
- Always start with a shebang
    - `#!/usr/bin/env bash`
- Add permissions with `chmod +x script.sh`

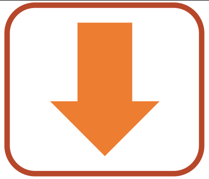
Globs and Ranges

What does a shell do?

Enter commands

Command string is parsed

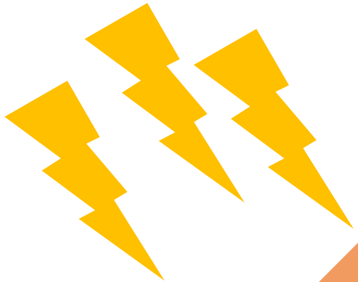
```
$ mv source_file.c destination
```



```
char *args[]={
  "mv",
  "source_file.c",
  "destination"
};
execvp("mv", args);
```

Bash loads the arguments and executes the command

Stuff happens



Globs and Ranges

```
mv file{1..3}.txt dst/
```

$\Downarrow$

```
mv file1.txt file2.txt file3.txt dst/
```

Ranges - { .. }

- Can be used to expand into many strings
  - Given a comma-separated sequence of words, it will expand into every permutation
  - {a,b,c} => a, b, c
  - {1,2,3}plusSome => 1plusSome, 2plusSome, 3plusSome
- You can use multiple ranges in a single line
  - {a,b,c}.{1,2,3} => a.1, a.2, a.3, b.1, b.2, b.3, c.1, c.2, c.3
  - Ranges can also figure out what you want in some cases use ..
  - {1..10} => 1, 2, 3, 4, 5, 6, 7, 8, 9, 10
  - {a..f} => a, b, c, d, e, f

Globs

- Used to match several argument names without typing all of them out
  - `rm *.txt` removes files with the .txt extension, no matter what their names are
- Special Character: ?
  - ? matches to a, b, c, 1, 2, 3, etc…
- Special Character: *
  - Matches to any number of any character
  - * matches to any string
- Can be combined with normal text to limit matches
  - `project*.pdf` matches to any file that starts with project and ends with pdf

Quiz!

Matches	Pattern
`file1 file2 file3`	`file? OR file{1..3}`
`file1 file2 item1 item2`	`{file,item}{1,2}`
`file4.pdf readme.pdf`	`*.pdf`
`file2 file3 file4.pdf`	`file{2..4}*`

Directory Contents

```
file1
file2
file3
file4.pdf
readme.pdf
item1
item2
```

Strings and escaping

- Arguments containing spaces can be written in quotes
  - `echo "Bash scripting is fun" -> Bash scripting is fun`
- They can also be written in single quotes
  - `echo 'Bash scripting is "fun"' -> Bash scripting is "fun"`
- Special characters can also be escaped with backslash
  - `echo "Bash scripting is \"fun\"" -> Bash scripting is "fun"`
- In single quotes, escape characters are ignored.
  - `echo 'Bash scripting is \"fun\"' -> Bash scripting is \"fun\"`

Regular Expressions

Regular Expressions

- Patterns that match against certain strings
- Different from globs
- Compatible with many applications
- But why are they called regular expressions?
    - For interesting theoretical reasons
    - That you will learn later

Example: Phone numbers

- Multiple possible strings
    - 123-456-7890
    - 1234567890
    - 456-789-1234
- But the formats follow a few patterns
    - ###-###-###
    - ##########

Solution: Regular expressions

- Create a pattern that specifies which strings to match
- `(\d{3}-?){2}\d{4}` – matches a phone number

Examples

- `gpi` – matches "gpi"
- `[hjkl]` – matches "h", "j", "k", and "l"
- `07-?131` – matches "07131" and "07-131"
- `item[1-3]` – matches "item1", "item2", "item3"
- `codes*` – matches "code", "codes", "codess", "codesssss", etc.

Parts of a regular expression

- Normal characters
  - `gpi - matches "gpi"`
- Quantifiers
  - repeating* – matches "repeatin", "repeating", " repeatingggg", etc.
  - ab{1,3} – matches "ab", "abb", or "abbb"
- Character classes
  - [hjkl] – matches "h", "j", "k", "l"
  - \d – matches and digit
  - . – matches any character
- Use parentheses for grouping

Quantifiers

Quantifier	Matches
`a?`	Zero or one
`a*`	Zero or more
`a+`	One or more
`a{3}`	Exactly 3
`a{3,}`	3 or more
`a{3,6}`	Between 3 and 6

Character classes

Class	Matches
`[abc]`	a or b or c
`[^abc]`	not any of a, b, c
`[a-z]`	A lowercase letter
`[^A-Za-z]`	Not a letter
`\s`	Whitespace
`\d`	Digit
`.`	Any single character

Example

```
(\d{3}-?){2}\d{4}
```

Matches any digit

Example

```
(\d{3}-?){2}\d{4}
```

Matches any 3 digits

Example

```
(\d{3}-?){2}\d{4}
```

Matches an optional hyphen

Example

```
(\d{3}-?){2}\d{4}
```

Matches 2 groups of 3 digits

Ex:

123-456-

123456-

123456

Example

```
(\d{3}-?){2}\d{4}
```

Matches 2 groups of 3 digits, then 4 more digits

Special sequences

- $ - End of string
- ^ - Start of string
- Parentheses for grouping

Cheat sheet

- a* – Matches zero or more times
- a? – Matches one or zero times
- a{3} – Matches three times
- . – Matches any single character
- [a-z0-9] – Matches a digit or lowercase character
- [^xy] – Matches anything other than x and y.
- ^ - Matches start of string
- $ - Matches end of string

Quiz!

Matches	Regex
ababab or **abab**	`abab(ab)? or (ab){2,3}`
ab any number of times	`(ab)*`
[any letter][any number] ex: A4	`[A-Za-z]\d`
`example.com website.com` etc.	`[a-z]*\.[a-z]*`

Regex vs Globs and ranges

Regex	Glob/Range equivalent
`.`	`?`
`file[1-7]\.txt`	`file{1..7}.txt`
`.*`	`*`
`(ab)*`	`Not possible`

Grep

- Search files and directories using regular expressions!
- Prints lines that include a match
- Name comes from g/re/p command in the UNIX text editor ed
- `$ grep 'evidence' largefile.txt`
    - Searches largefile.txt for "evidence".
- `$ grep -r 'secrets' path/to/directory`
    - Searches recursively for "secrets".

Sed

- Stands for "stream editor"
- Can perform find and replace on a file
- sed 's/find/replace/g' path/to/file
    - Prints result of replacement to the command line, leaving input untouched
- sed -i 's/find/replace/g' path/to/file
    - "In place"
    - Edits the file

How does grep work?

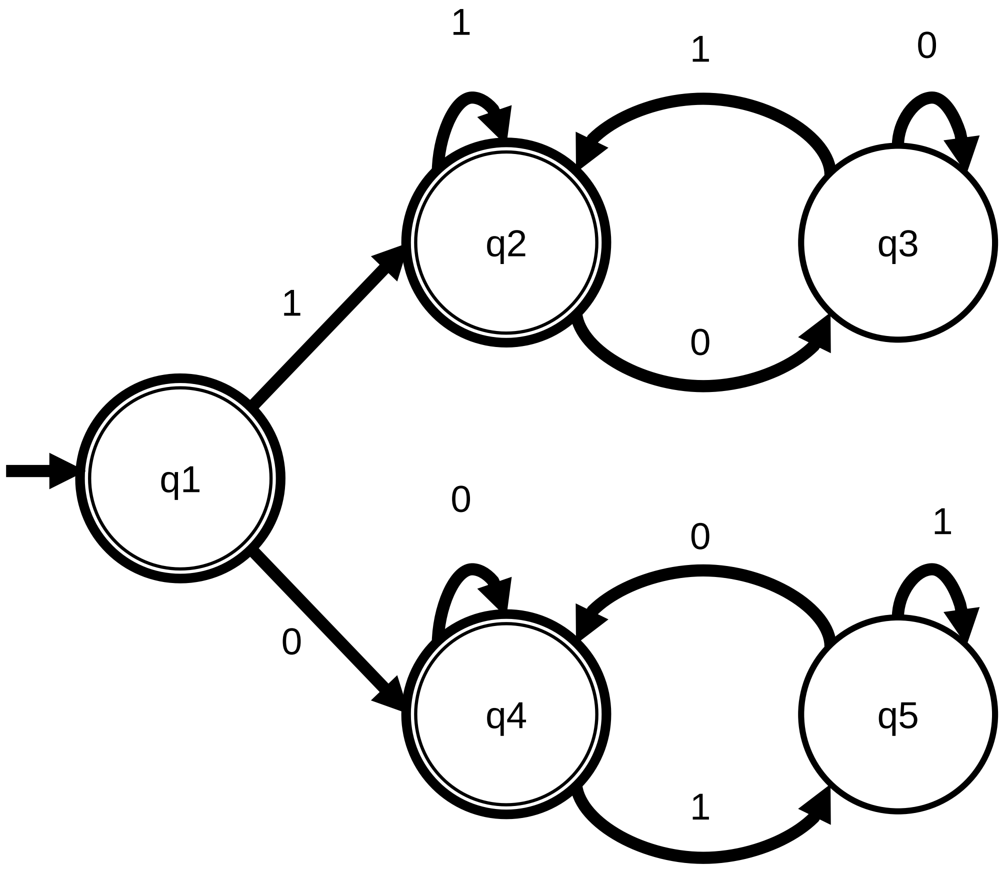
- It seems like some guessing is necessary
  - Imagine matching "abc" against a?b?c?abc
  - Lots of guessing would be exponential time.
- But grep is fast
  - For deep theoretical reasons. Involving finite state machines.

<Extra Content>

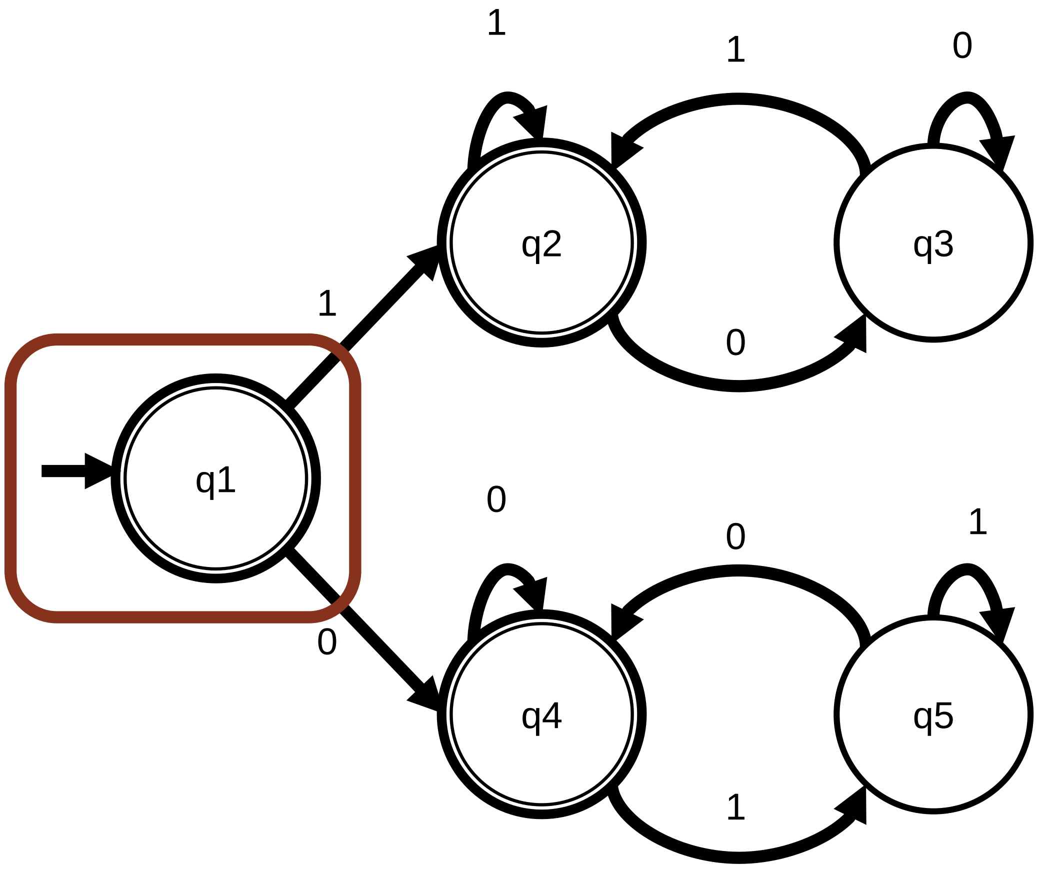
DFA = Deterministic
Finite-state
Automaton

DFA $= (Q, \Sigma, \delta, q_0, F)$

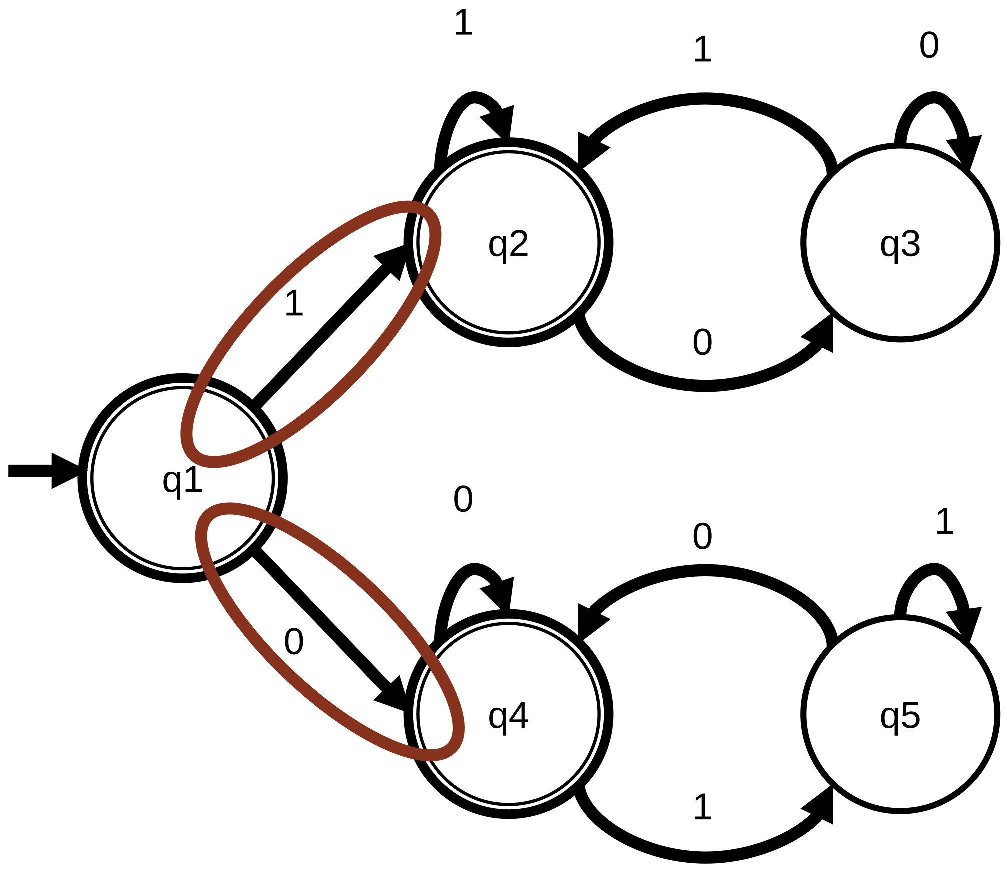
DFA =



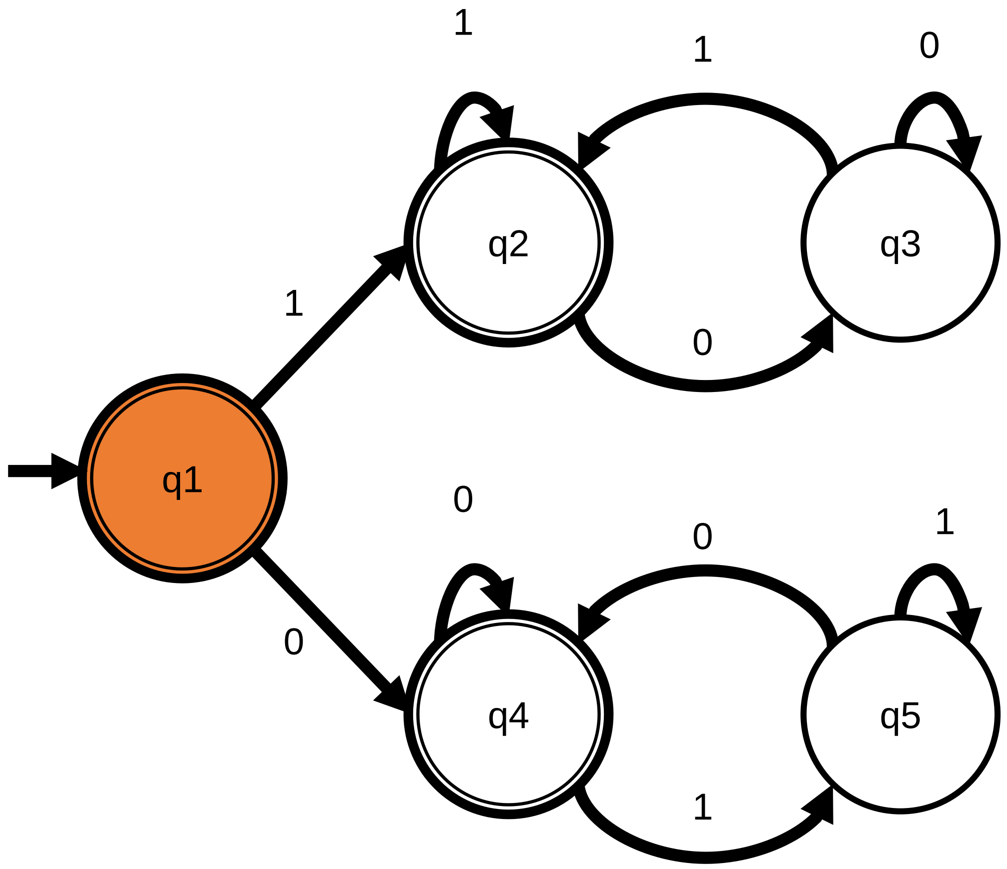
DFA =



DFA =

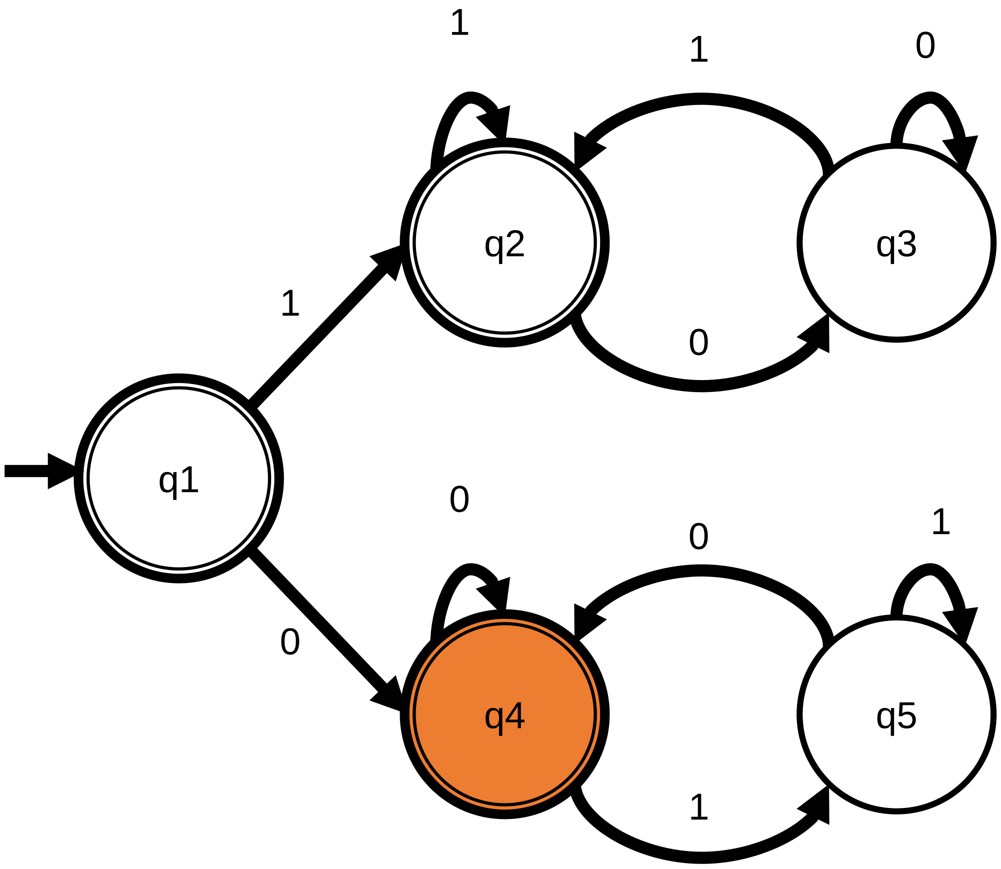


DFA =



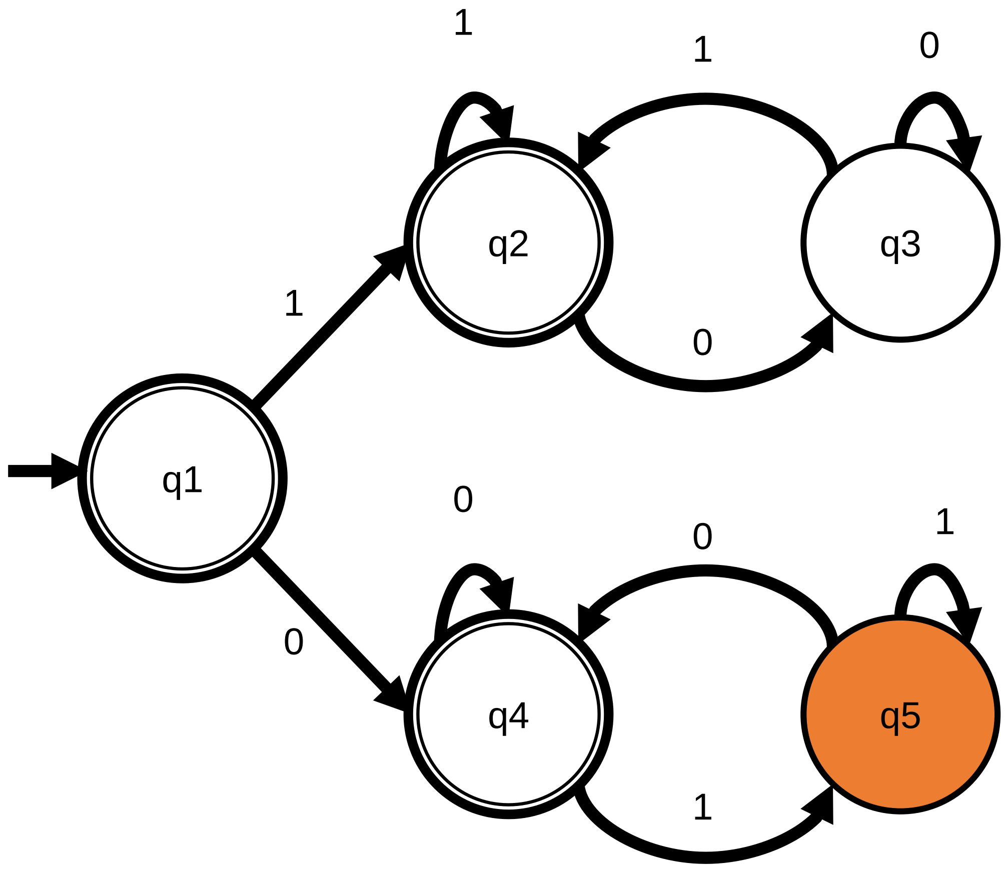
1001

DFA =



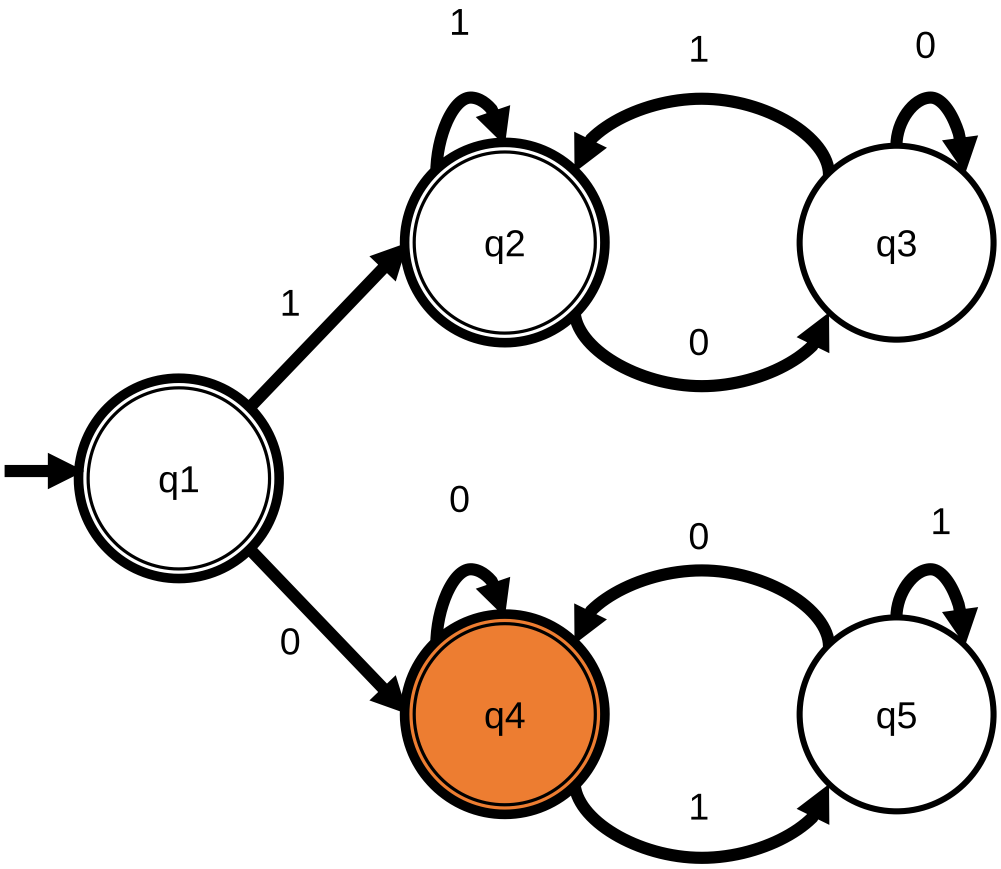
1001

DFA =



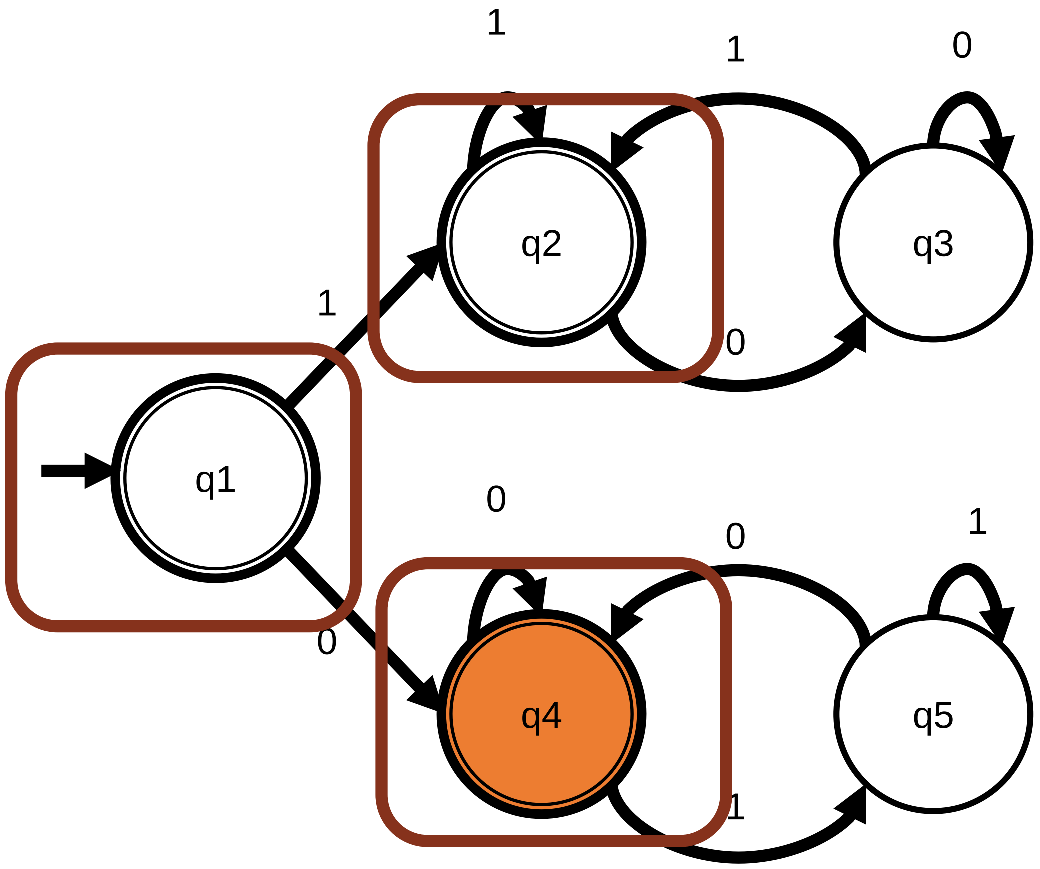
1001

DFA =



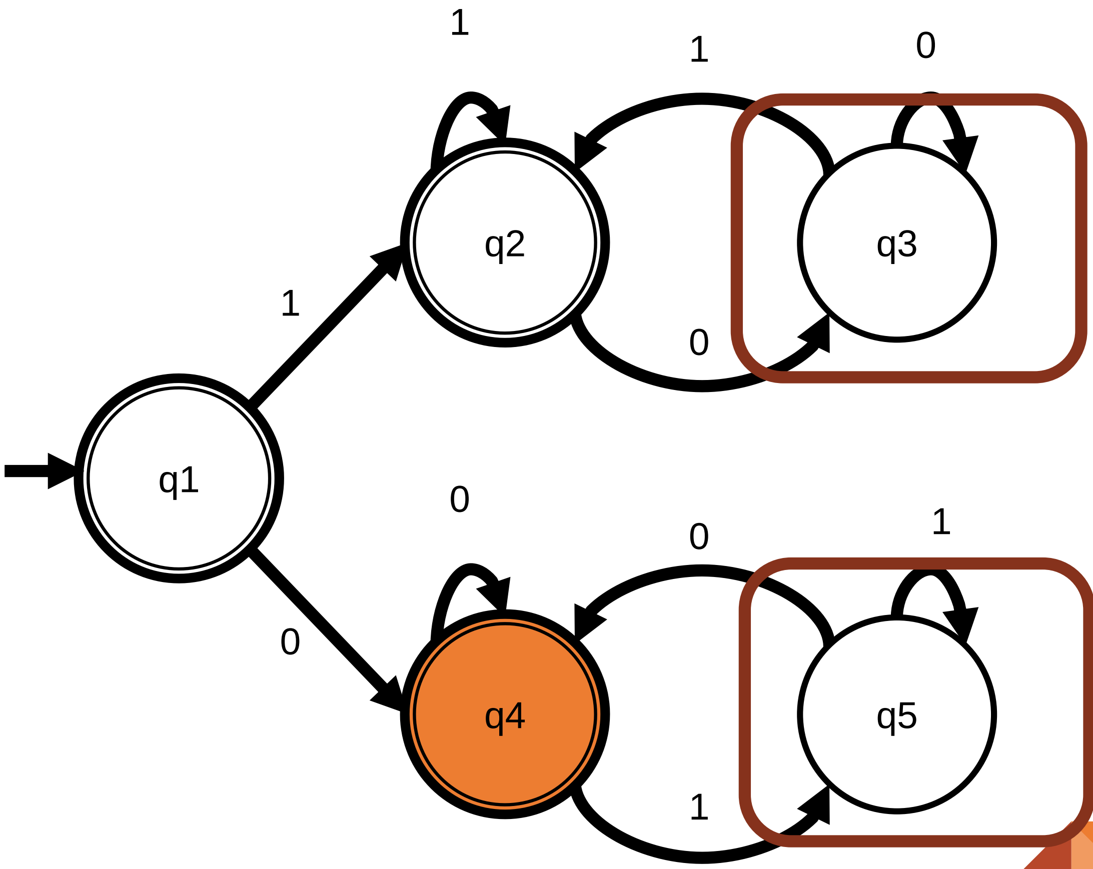
1001

DFA =



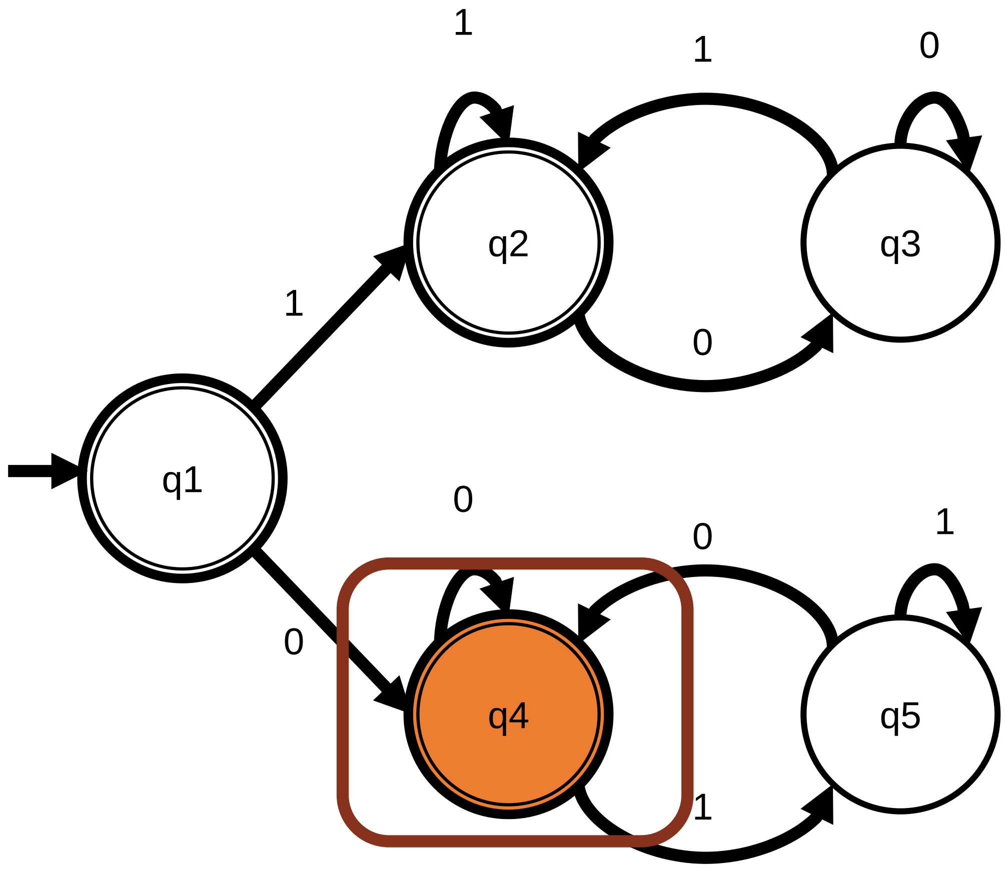
1001

DFA =



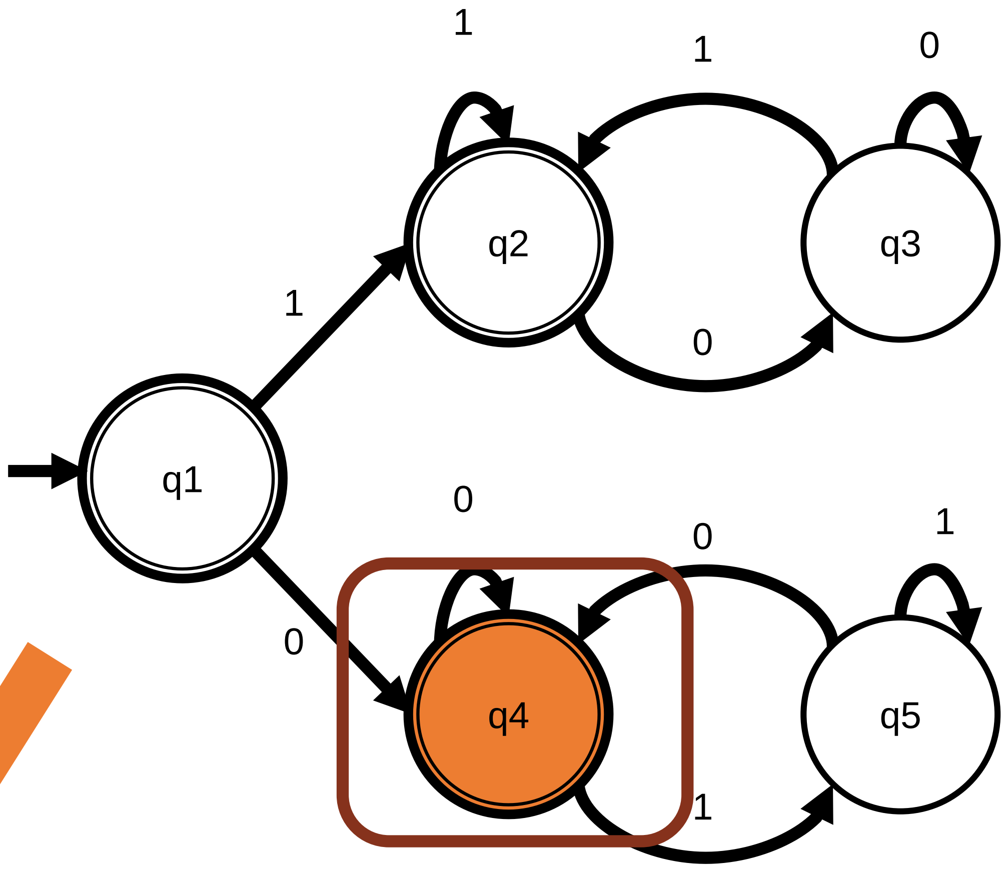
1001

DFA =



1001

DFA =

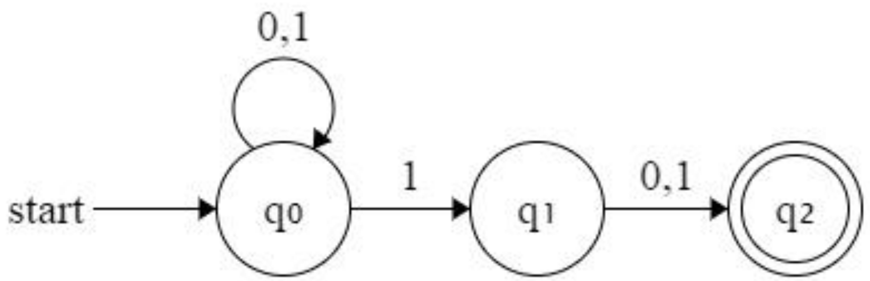


1001 ✓

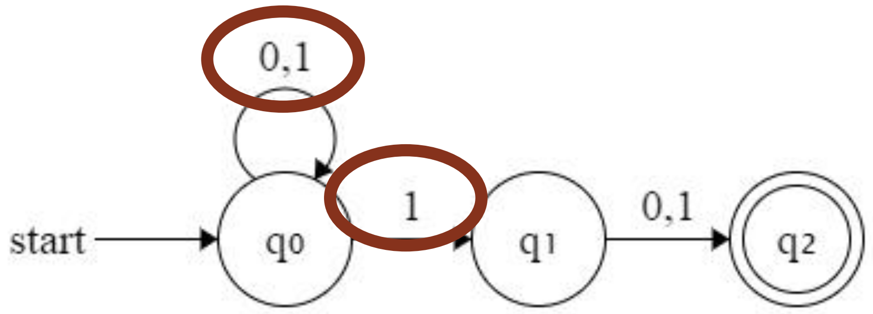
$$\text{Regex} \xrightarrow{\text{Efficiently Converted}} \text{NFA}$$

NFA = Nondeterministic Finite-state Automaton

NFA =



NFA =



- Accept if there exists any path to an accepting state
- Computed by updating the set of reachable states

`grep`

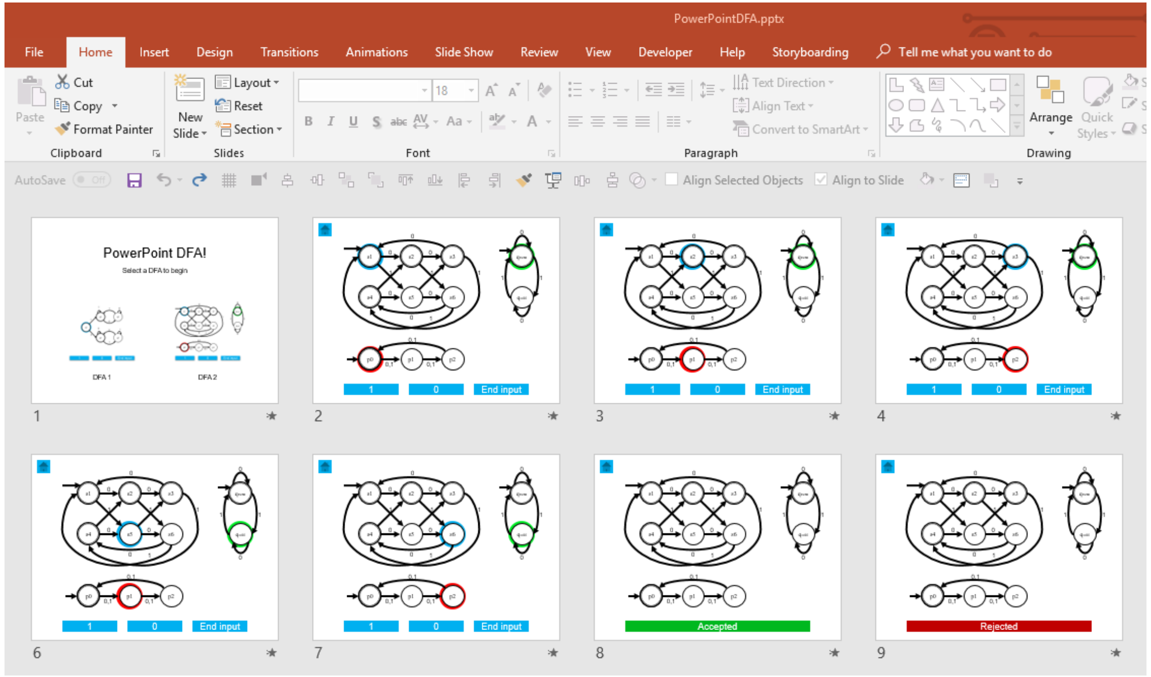
$$\begin{tikzcd}\text{Regex} \arrow[r] & \text{NFA} \arrow[r] & \text{Evaluate NFA on string}\end{tikzcd}$$

</Extra Content>

Just one more thing…

If we call the states “slides” and the transitions “hyperlinks”…

Great Impractical Ideas in CS
PowerPoint Programming



(This week's extratation)

Cheat sheet

- a* – Matches zero or more times
- a? – Matches one or zero times
- a{3} – Matches three times
- . – Matches any single character
- [a-z0-9] – Matches a digit or lowercase character
- [^xy] – Matches anything other than x and y.
- ^ - Matches start of string
- $ - Matches end of string

Bash scripting summary

- Bash scripts end in a .sh extension
- Always start with a shebang
  - `#!/usr/bin/env bash`
- Add permissions with `chmod +x script.sh`

Lab pro tips

- Labs are forcelab and zombielab
- Be careful with escaping correctly. Both bash and regex have characters that must be escaped
- Don't forget to do **chmod +x script.sh** and add **#!/usr/bin/env bash**
- Have to leave at 4:00 today for Jeff Dean lecture