# String Processing

Lecture 14

CS106A, Summer 2019
Sarai Gould && Laura Cruz-Albrecht

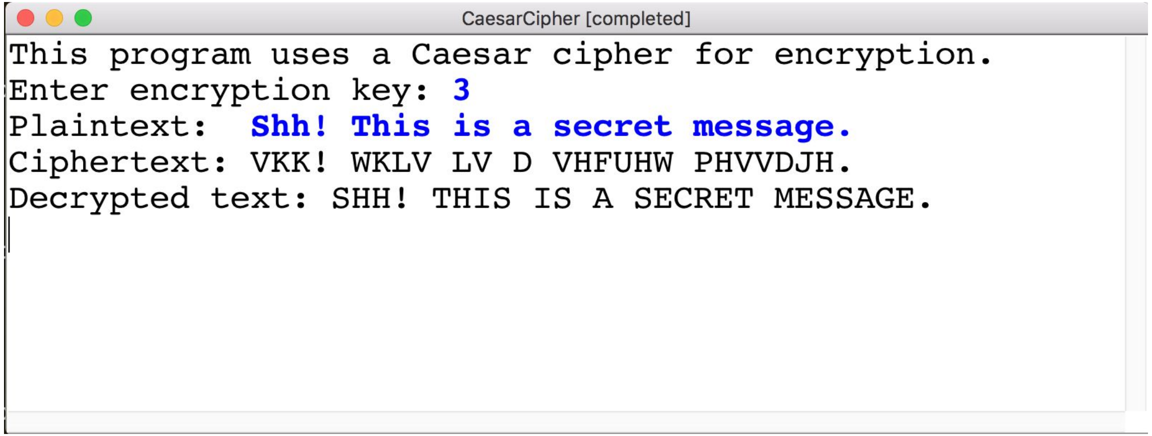
With inspiration from slides created by Keith Schwarz, Mehran Sahami, Eric Roberts, Stuart Reges, Chris Piech and others.

# Announcements

- Assignment 3 due tomorrow at 10AM
- Midterm: check out website page
  - download Bluebook, be sure to have 2 factor authentication with passcodes; see yesterday's lecture for helpful links.
  - Midterm conflicts: was due last night, if for some reason you haven't done it, please do so **now**: http://bit.ly/CS106AMidtermConflicts
- Midterm review session: Friday 10:30AM in Gates B01
- Reminder: Course Schedule has code + suggested readings
  - For today, blank code has been posted so you can code along in lecture if you would like

# Learning Goals Today

- Be able to write string algorithms that operate on each character.
- Be able to build up new strings from existing strings using built-in String methods.



```
This program uses a Caesar cipher for encryption.
Enter encryption key: 3
Plaintext:  Shh! This is a secret message.
Ciphertext: VKK! WKLV LV D VHFUHW PHVVDJH.
Decrypted text: SHH! THIS IS A SECRET MESSAGE.
```

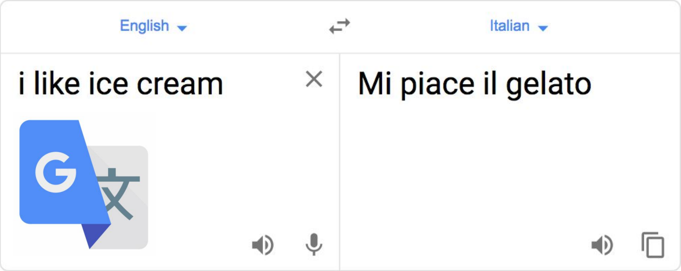
# Plan for Today

- Review: Characters and Strings
- Looping over Strings
- Practice: Reversing a String
- Practice: Palindromes
- Practice: Caesar Cipher

# Plan for Today

- Review: Characters and Strings
- Looping over Strings
- Practice: Reversing a String
- Practice: Palindromes
- Practice: Caesar Cipher

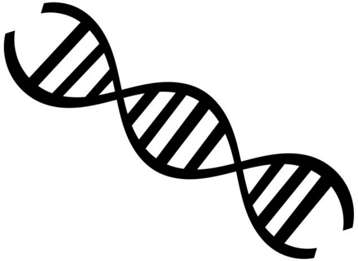
# Text Processing

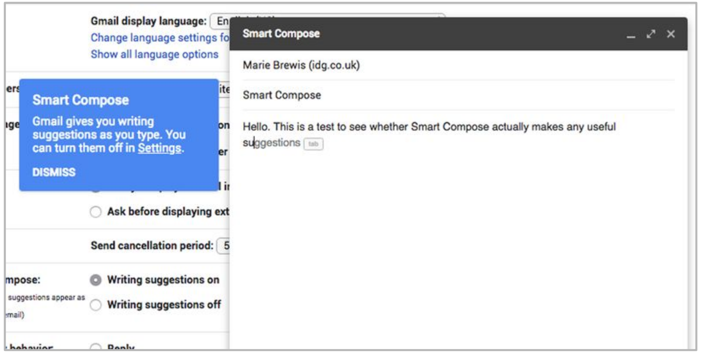


English ▾ ⇄ Italian ▾

i like ice cream

Mi piace il gelato





Gmail display language: En

Change language settings fo

Show all language options

Smart Compose

Gmail gives you writing suggestions as you type. You can turn them off in Settings.

DISMISS

○ Ask before displaying ext

Send cancellation period: 5

mpose: ● Writing suggestions on

suggestions appear as ○ Writing suggestions off

mail)

behavior ○ Reply

Smart Compose

Marie Brewis (idg.co.uk)

Smart Compose

Hello. This is a test to see whether Smart Compose actually makes any useful suggestions tab

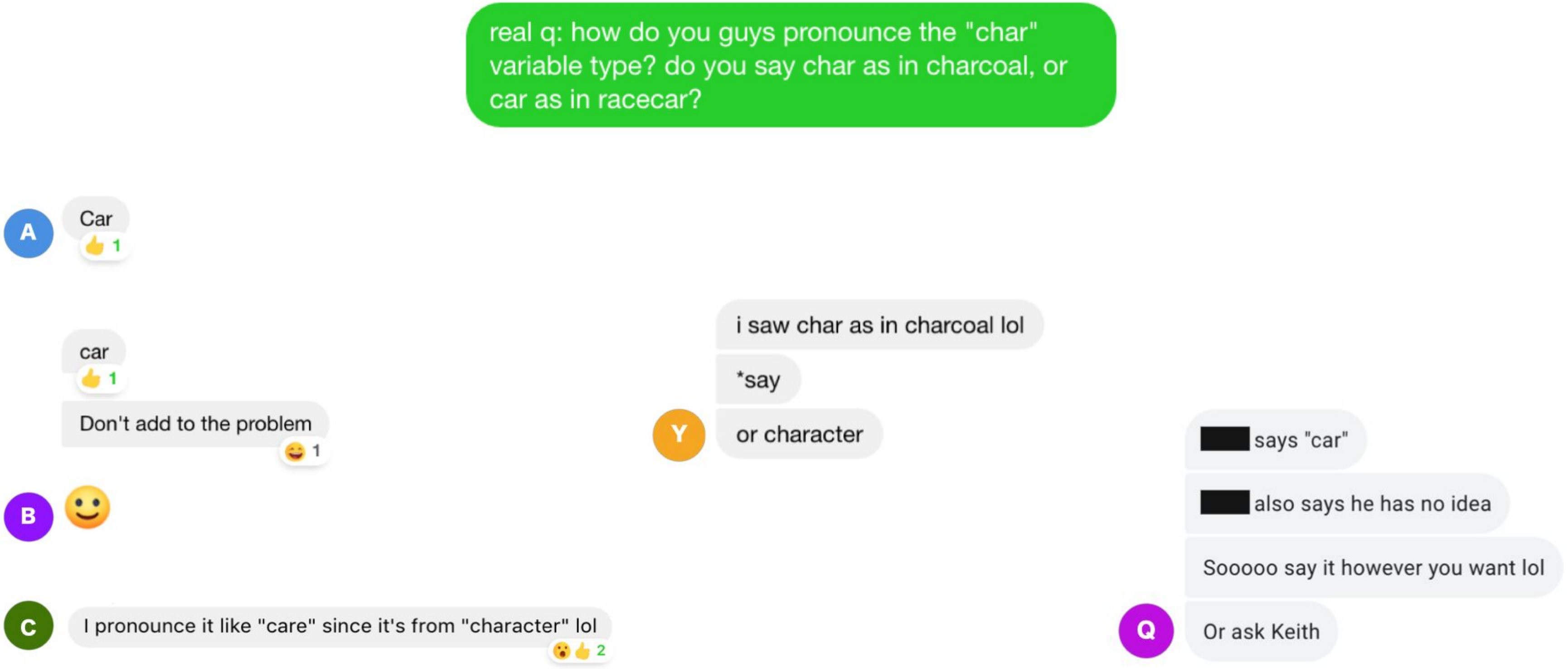
# Characters

A **char** is a variable type that represents a single character or “glyph”.

**Single quotes!**

```
char letterA = 'A';
```

# An Aside



real q: how do you guys pronounce the "char" variable type? do you say char as in charcoal, or car as in racecar?

Car

👍 1

car

👍 1

Don't add to the problem

😄 1

🙂

I pronounce it like "care" since it's from "character" lol

😮👍 2

i saw char as in charcoal lol

*say

or character

[redacted] says "car"

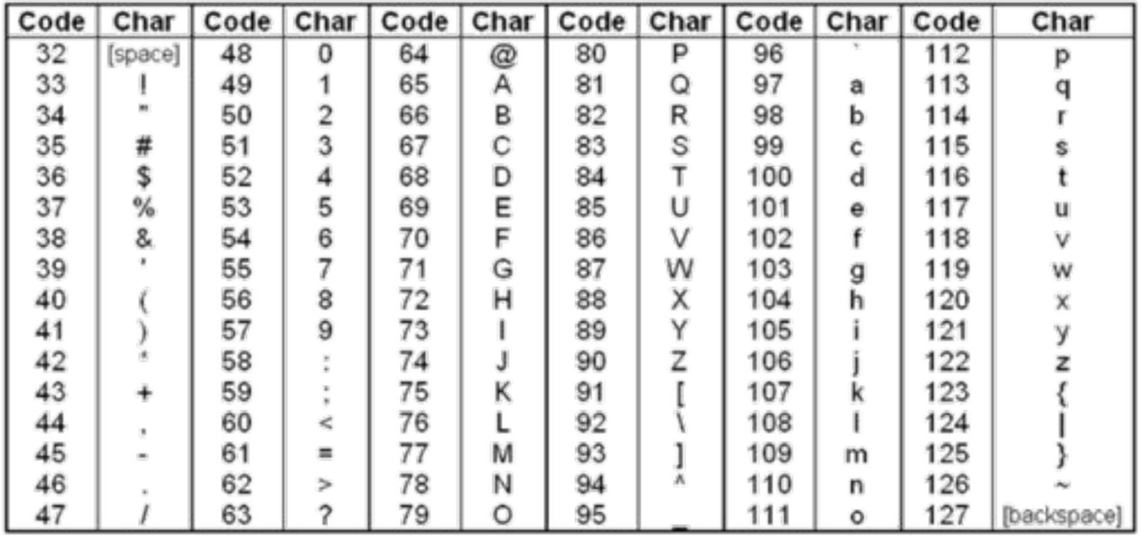
[redacted] also says he has no idea

Sooooo say it however you want lol

Or ask Keith

# Char

Under the hood, Java represents each **char** as an *integer*. This integer is its “ASCII” value.



| Code | Char | Code | Char | Code | Char | Code | Char | Code | Char | Code | Char |
|---|---|---|---|---|---|---|---|---|---|---|---|
| 32 | [space] | 48 | 0 | 64 | @ | 80 | P | 96 | ` | 112 | p |
| 33 | ! | 49 | 1 | 65 | A | 81 | Q | 97 | a | 113 | q |
| 34 | " | 50 | 2 | 66 | B | 82 | R | 98 | b | 114 | r |
| 35 | # | 51 | 3 | 67 | C | 83 | S | 99 | c | 115 | s |
| 36 | $ | 52 | 4 | 68 | D | 84 | T | 100 | d | 116 | t |
| 37 | % | 53 | 5 | 69 | E | 85 | U | 101 | e | 117 | u |
| 38 | & | 54 | 6 | 70 | F | 86 | V | 102 | f | 118 | v |
| 39 | ' | 55 | 7 | 71 | G | 87 | W | 103 | g | 119 | w |
| 40 | ( | 56 | 8 | 72 | H | 88 | X | 104 | h | 120 | x |
| 41 | ) | 57 | 9 | 73 | I | 89 | Y | 105 | i | 121 | y |
| 42 | * | 58 | : | 74 | J | 90 | Z | 106 | j | 122 | z |
| 43 | + | 59 | ; | 75 | K | 91 | [ | 107 | k | 123 | { |
| 44 | , | 60 | < | 76 | L | 92 | \ | 108 | l | 124 | \| |
| 45 | - | 61 | = | 77 | M | 93 | ] | 109 | m | 125 | } |
| 46 | . | 62 | > | 78 | N | 94 | ^ | 110 | n | 126 | ~ |
| 47 | / | 63 | ? | 79 | O | 95 | _ | 111 | o | 127 | [backspace] |

# Char

Under the hood, Java represents each **char** as an *integer*.
This integer is its “ASCII” value.

```
char uppercaseA = 'A';    // Actually 65
char lowercaseA = 'a';    // Actually 97
char zeroDigit = '0';     // Actually 48
```

# Char

Under the hood, Java represents each **`char`** as an *integer*.
This integer is its “ASCII” value.

```
char uppercaseA = 'A';     // Actually 65
char lowercaseA = 'a';     // Actually 97
char zeroDigit = '0';      // Actually 48
```

- Uppercase letters (`'A' -> 'Z'`) are sequentially numbered
- Lowercase letters (`'a' -> 'z'`) are sequentially numbered
- Digits (`'0' -> '9'`) are sequentially numbered

# Char Math

We can take advantage of Java representing each **char** as an *integer* (its “ASCII” value).

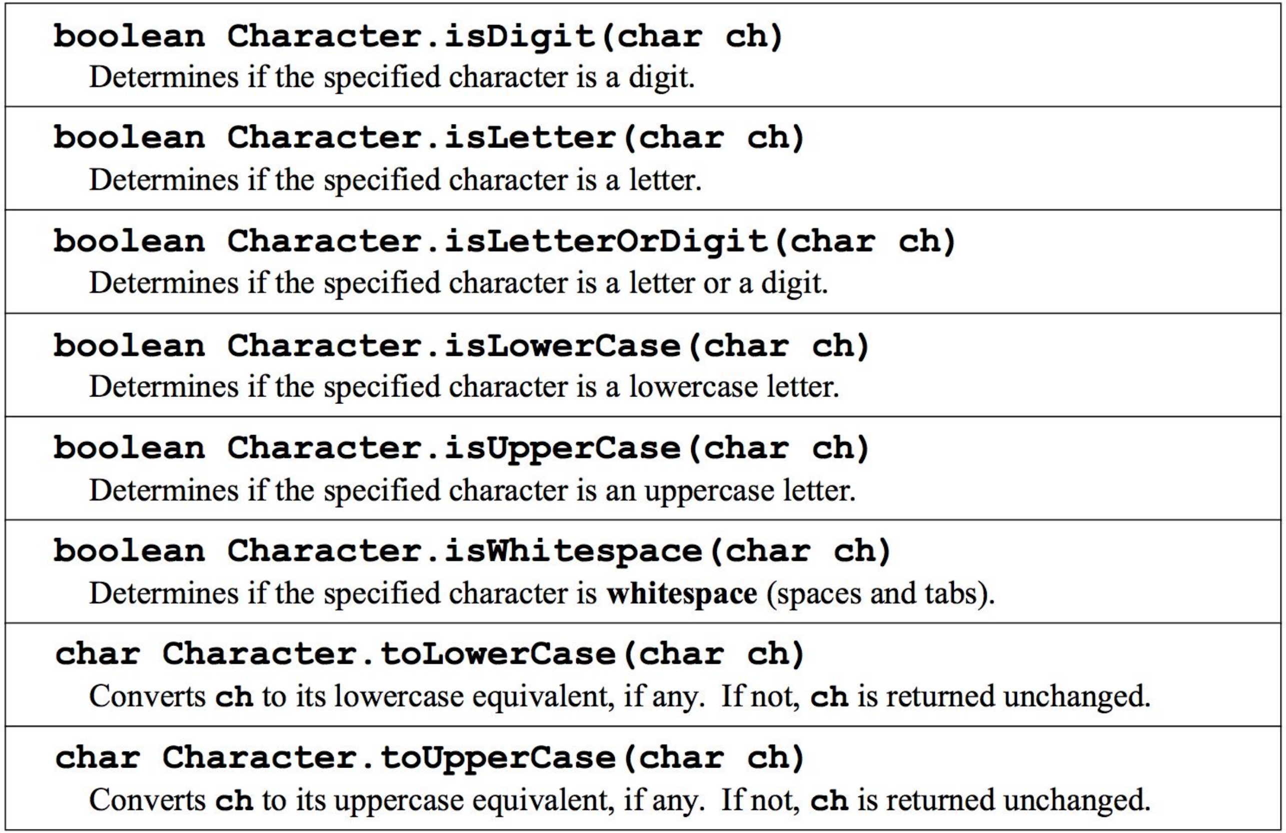
```
boolean areEqual = 'A' == 'A';     // true
boolean earlierLetter = 'f' < 'c'; // false
char uppercaseB = 'A' + 1;         // 'B'
int diff = 'c' - 'a';              // 2

int alphabetSize = 'z' – 'a' + 1;
// or
int alphabetSize = 'Z' – 'A' + 1;
```

# Type-Casting

If we want to force Java to treat an expression as a particular type, we can also *cast it* to that type.

```
'A' + 1                 // evaluates to 66 (int)
(char)('A' + 1)         // evaluates to 'B' (char)

1 / 2                   // evaluates to 0 (int)
(double)1 / 2           // evaluates to 0.5 (double)
1 / (double)2           // evaluates to 0.5 (double)
```

# Character Methods



| |
|---|
| **`boolean Character.isDigit(char ch)`**<br>Determines if the specified character is a digit. |
| **`boolean Character.isLetter(char ch)`**<br>Determines if the specified character is a letter. |
| **`boolean Character.isLetterOrDigit(char ch)`**<br>Determines if the specified character is a letter or a digit. |
| **`boolean Character.isLowerCase(char ch)`**<br>Determines if the specified character is a lowercase letter. |
| **`boolean Character.isUpperCase(char ch)`**<br>Determines if the specified character is an uppercase letter. |
| **`boolean Character.isWhitespace(char ch)`**<br>Determines if the specified character is **whitespace** (spaces and tabs). |
| **`char Character.toLowerCase(char ch)`**<br>Converts **ch** to its lowercase equivalent, if any.  If not, **ch** is returned unchanged. |
| **`char Character.toUpperCase(char ch)`**<br>Converts **ch** to its uppercase equivalent, if any.  If not, **ch** is returned unchanged. |

Remember: these return a new `char`, they cannot modify an existing `char`.

# Strings

Text is stored using the variable type `String`.

A `String` is a sequence of characters!

Double quotes!

```
String text = “Hello!”;
```

# Strings

- Each character is assigned an index, going from 0 to length-1.
- There is a `char` at each index.

| | | | | | |
|---|---|---|---|---|---|
| H | e | l | l | o | ! |
| 0 | 1 | 2 | 3 | 4 | 5 |

```
int strLen = text.length();            // 6
char last = text.charAt(strLen - 1);  // ‘!’
```

# Strings vs. Chars

Remember: `char`s and length-1 `String`s are different!

`char ch = 'A'` DIFFERENT FROM `String str = “A”`

```
'A' + 1    // evaluates to 66 (int)
“A” + 1    // evaluates to “A1” (String)
```

# Creating Strings

```
String str = "Hello, world!";
String empty = "";

// Read in text from the user
String name = readLine("What is your name? ");

// String concatenation (using “+”)
String message = name + " is " + 2 + " cool.";
```

# From Chars to Strings

```
char c1 = 'a';
char c2 = 'b';

// How do we concatenate these characters?

String str = c1 + c2;       // ERROR: this is an int!
```

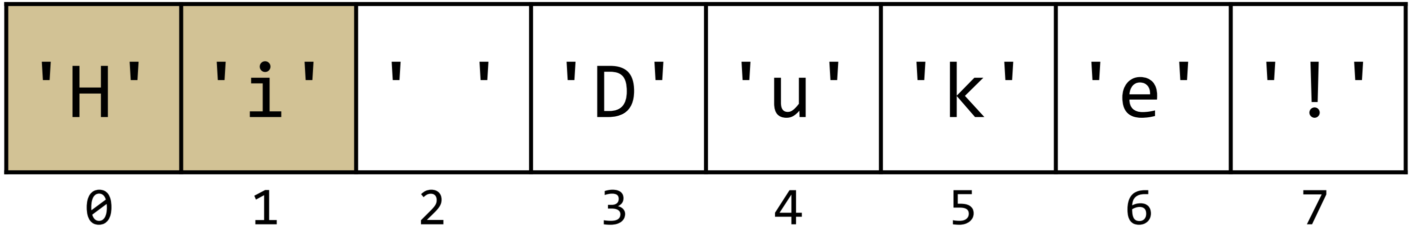
# From Chars to Strings

```
char c1 = 'a';
char c2 = 'b';

// How do we concatenate these characters?

String str = c1 + c2;        // ERROR: this is an int!

String str = "" + c1 + c2;  // ✅
```

# Substrings

A ***substring*** is a subset of a string.

```
String str = “Hi Duke!”;
String hi = str.substring(0, 2);
```

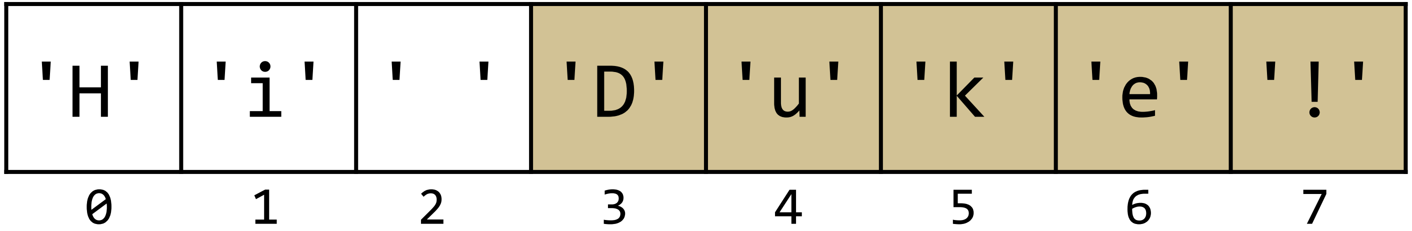


| 'H' | 'i' | ' ' | 'D' | 'u' | 'k' | 'e' | '!' |
|---|---|---|---|---|---|---|---|
| 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 |

# Substrings

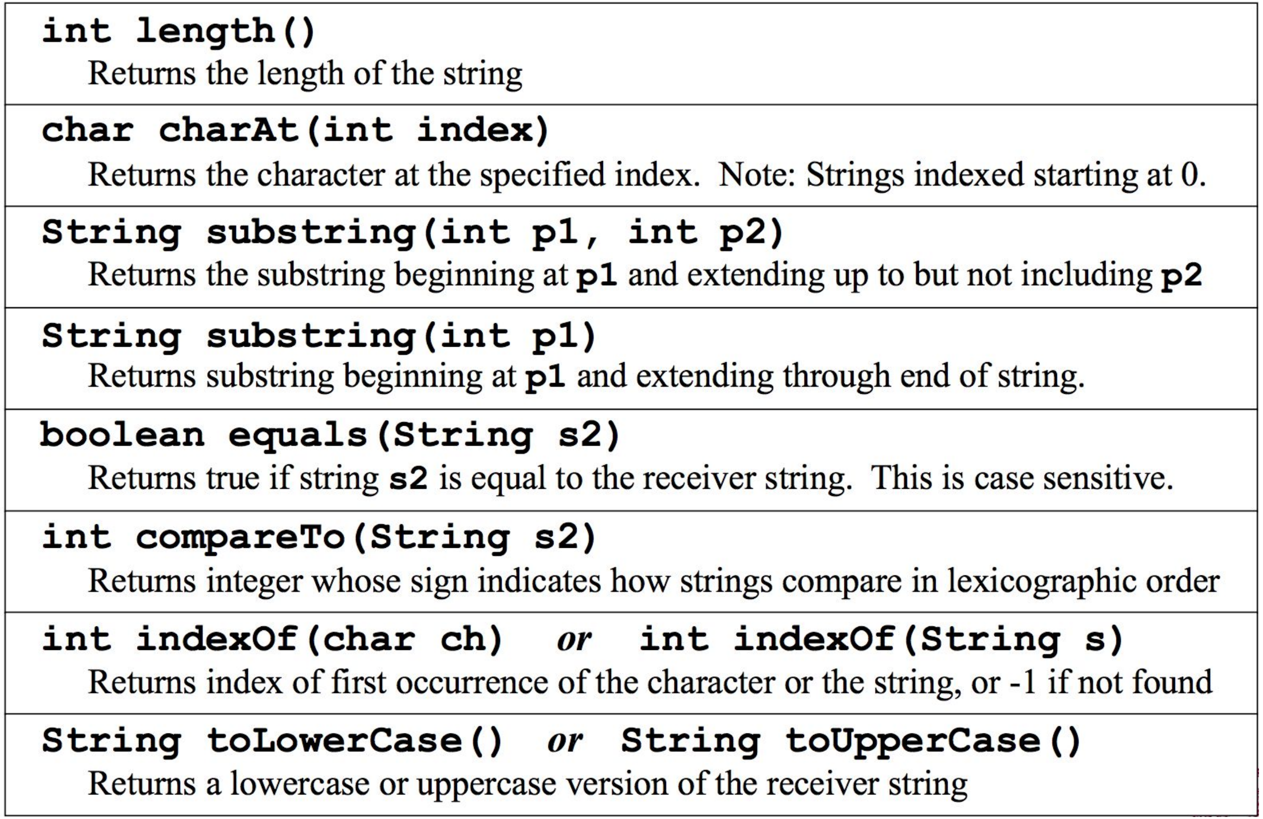
A ***substring*** is a subset of a string.

```
String str = “Hi Duke!”;
String dukeExclm = str.substring(3);  // to end
```



| 'H' | 'i' | ' ' | 'D' | 'u' | 'k' | 'e' | '!' |
|---|---|---|---|---|---|---|---|
| 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 |

# Useful String Methods



**`int length()`**

Returns the length of the string

**`char charAt(int index)`**

Returns the character at the specified index. Note: Strings indexed starting at 0.

**`String substring(int p1, int p2)`**

Returns the substring beginning at **`p1`** and extending up to but not including **`p2`**

**`String substring(int p1)`**

Returns substring beginning at **`p1`** and extending through end of string.

**`boolean equals(String s2)`**

Returns true if string **`s2`** is equal to the receiver string. This is case sensitive.

**`int compareTo(String s2)`**

Returns integer whose sign indicates how strings compare in lexicographic order

**`int indexOf(char ch)`** *or* **`int indexOf(String s)`**

Returns index of first occurrence of the character or the string, or -1 if not found

**`String toLowerCase()`** *or* **`String toUpperCase()`**

Returns a lowercase or uppercase version of the receiver string

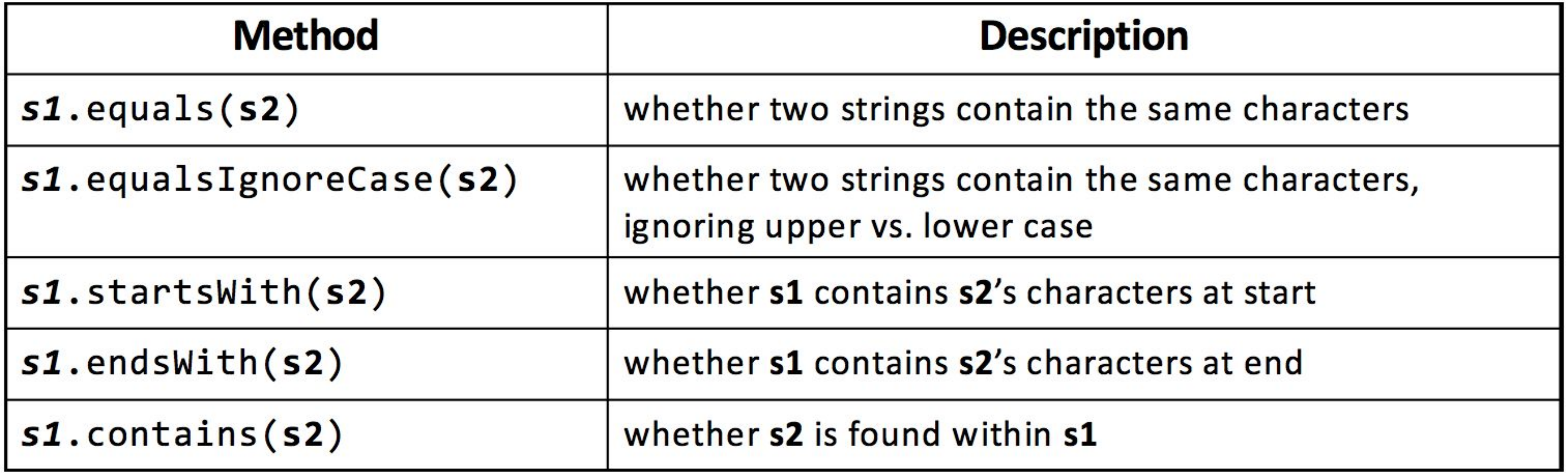
* remember, called using **dot notation**: *`myString`*`.length()`

# Strings are Immutable

- Java strings are **immutable**: once you create a String, its contents cannot be changed.
- To change a String, you must create a *new* String containing the value you want (e.g. using String methods).

```
String typo = "Hello, warld!”;

typo.charAt(8) = ‘o’;   // Error! Will not run.

String corrected = typo.substring(0, 8) +
                     ‘o’ + typo.substring(9);
```

# Comparing Strings



| Method | Description |
|---|---|
| `s1.equals(s2)` | whether two strings contain the same characters |
| `s1.equalsIgnoreCase(s2)` | whether two strings contain the same characters, ignoring upper vs. lower case |
| `s1.startsWith(s2)` | whether **s1** contains **s2**’s characters at start |
| `s1.endsWith(s2)` | whether **s1** contains **s2**’s characters at end |
| `s1.contains(s2)` | whether **s2** is found within **s1** |

*Always* use `.equals()` instead of `==` and `!=`

# Plan for Today

- Review: Characters and Strings
- Looping over Strings
- Practice: Reversing a String
- Practice: Palindromes
- Practice: Caesar Cipher

# Looping over Strings

A common String programming pattern is looping over a String and operating on each character.

```
for (int i = 0; i < str.length(); i++) {
    char ch = str.charAt(i);
    // do something with ch here
}
```

# Looping over Strings

A common String programming pattern is looping over a String and operating on each character.

```
// prints out each letter on a separate line
for (int i = 0; i < str.length(); i++) {
    char ch = str.charAt(i);
    println(ch);
}
```

# Looping over Strings

Another common String programming pattern is **building up a new string** by adding characters to it over time.

```
// Creates a new String in all caps
String str = "Hello!";
String newStr = "";
for (int i = 0; i < str.length(); i++) {
    char ch = str.charAt(i);
    newStr = newStr + Character.toUpperCase(ch);
}
println(newStr);  // HELLO!
```

# Looping over Strings

Another common String programming pattern is **building up a new string** by adding characters to it over time.

```
// Creates a new String in all caps
String str = "Hello!";
String newStr = "";
for (int i = 0; i < str.length(); i++) {
    char ch = str.charAt(i);
    newStr += Character.toUpperCase(ch);
}
println(newStr);  // HELLO!
```

# Looping over Strings

Another common String programming pattern is **building up a new string** by adding characters to it over time.

```
// Creates a new String containing digits 0 through 4
String str = "";
for (int i = 0; i < 5; i++) {
    str += i;
}
println(str);  // 01234
```

# Plan for Today

- Review: Characters and Strings
- Looping over Strings
- Practice: Reversing a String
- Practice: Palindromes
- Practice: Caesar Cipher

# Exercise: Reversing a String

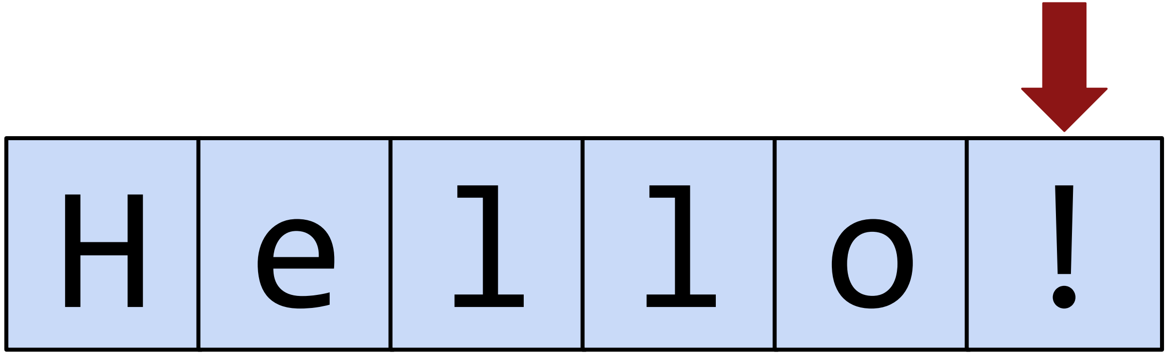
Let's write a method called `reverseString` that takes one String parameter, and returns a new String with the characters in the opposite order.

```
reverseString("Hello!") -> "!olleH"
```

# Reversing a String

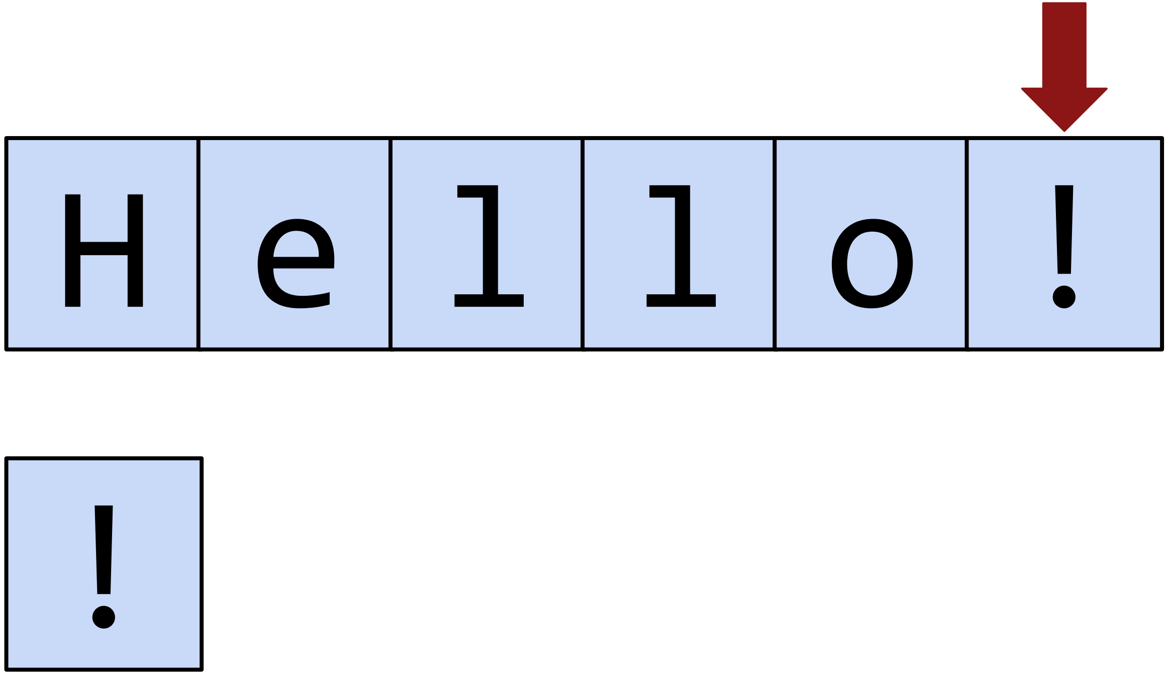
| | | | | | |
|---|---|---|---|---|---|
| H | e | l | l | o | ! |

# Reversing a String



| | | | | | |
|---|---|---|---|---|---|
| H | e | l | l | o | ! |

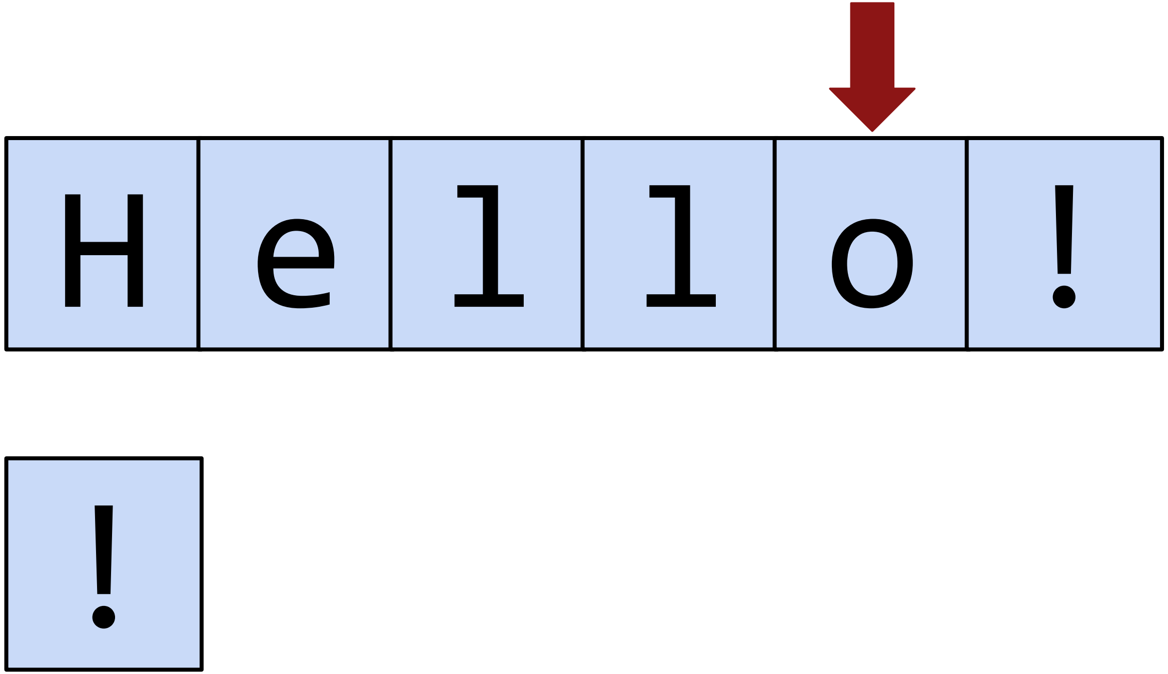
# Reversing a String



H e l l o !

!

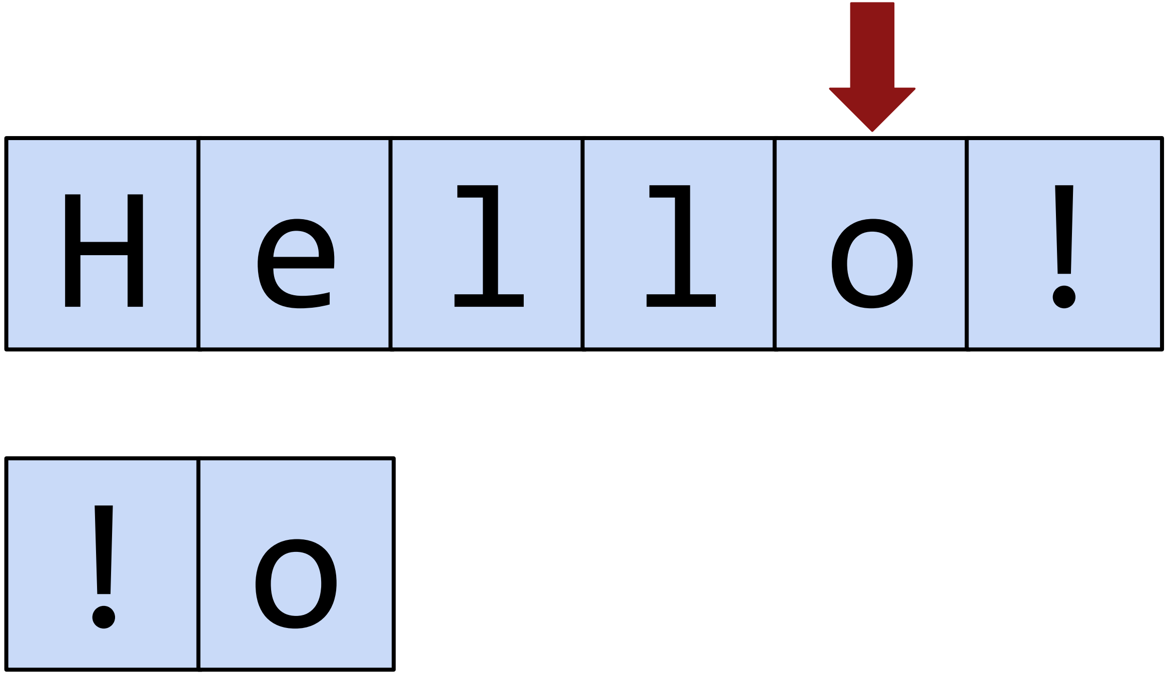
# Reversing a String



| H | e | l | l | o | ! |
|---|---|---|---|---|---|

| ! |
|---|

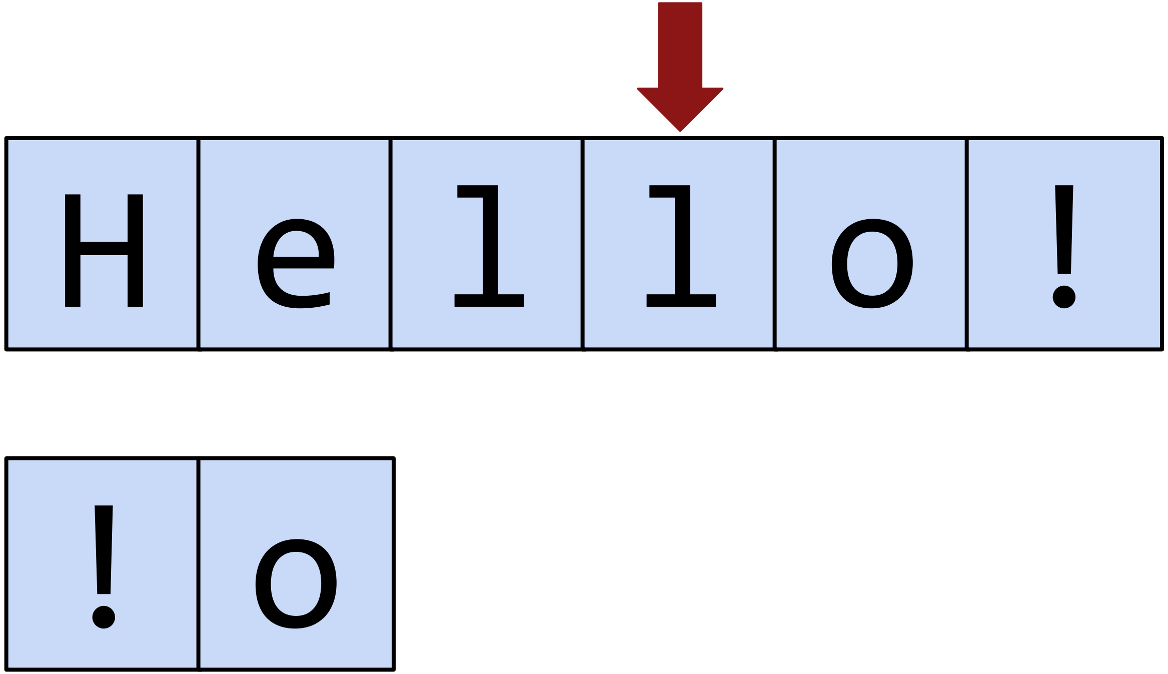
# Reversing a String



| H | e | l | l | o | ! |
|---|---|---|---|---|---|

| ! | o |
|---|---|

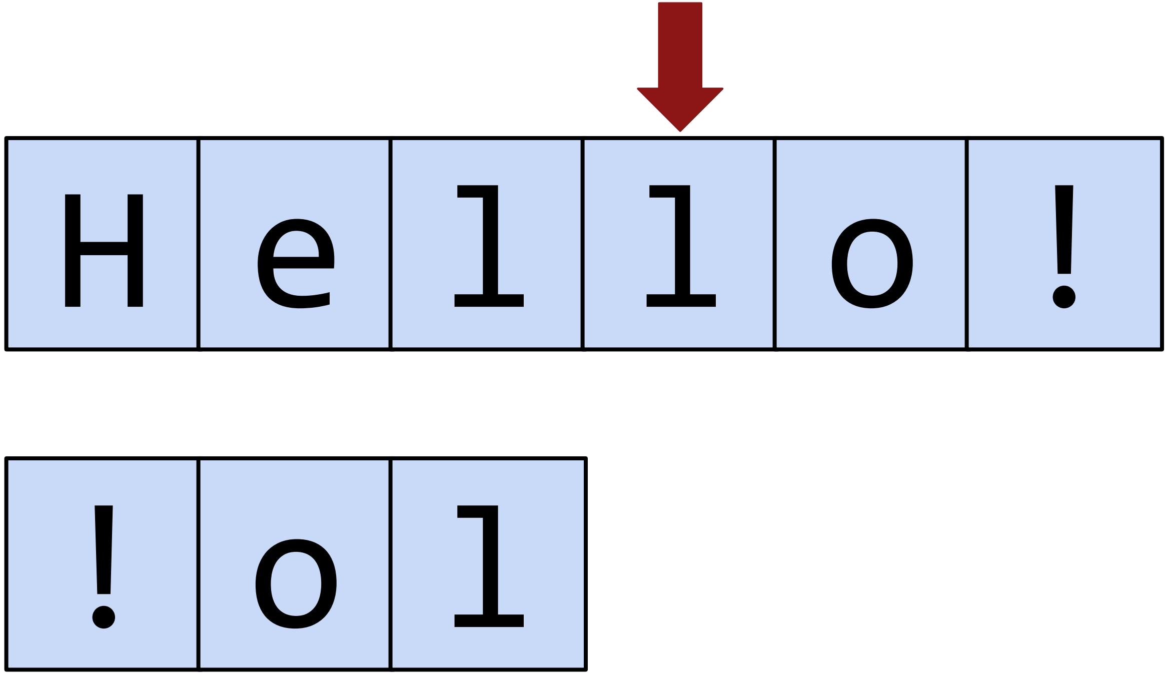
# Reversing a String



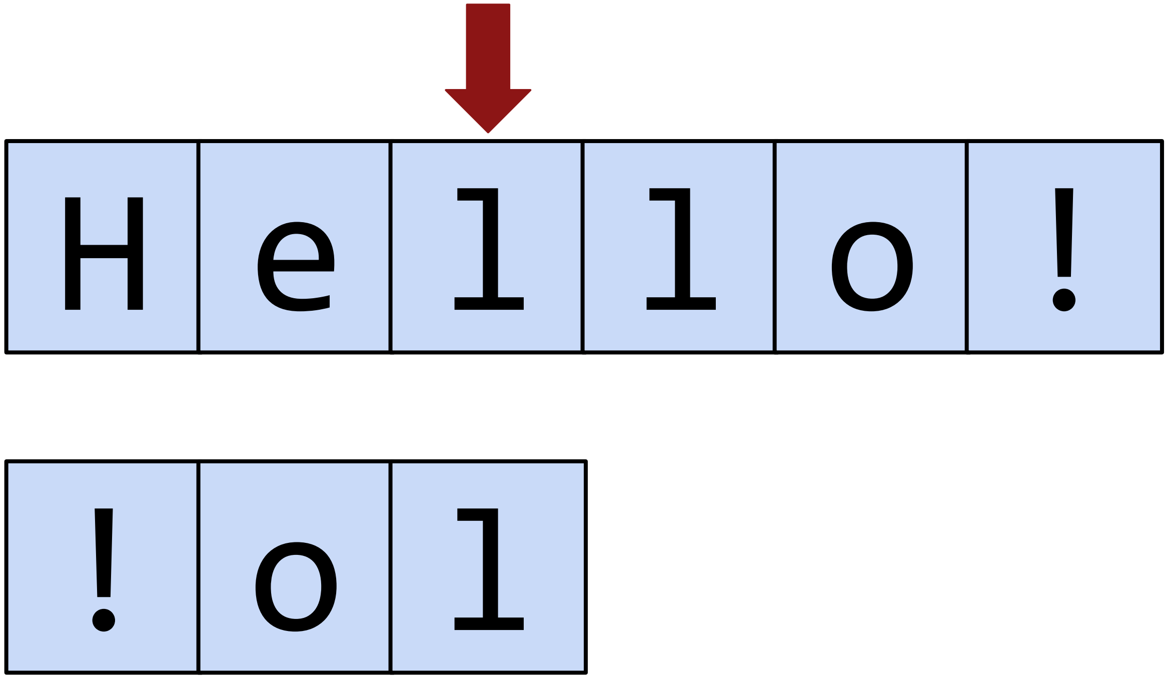
| | | | | | |
|---|---|---|---|---|---|
| H | e | l | l | o | ! |

| | |
|---|---|
| ! | o |

# Reversing a String



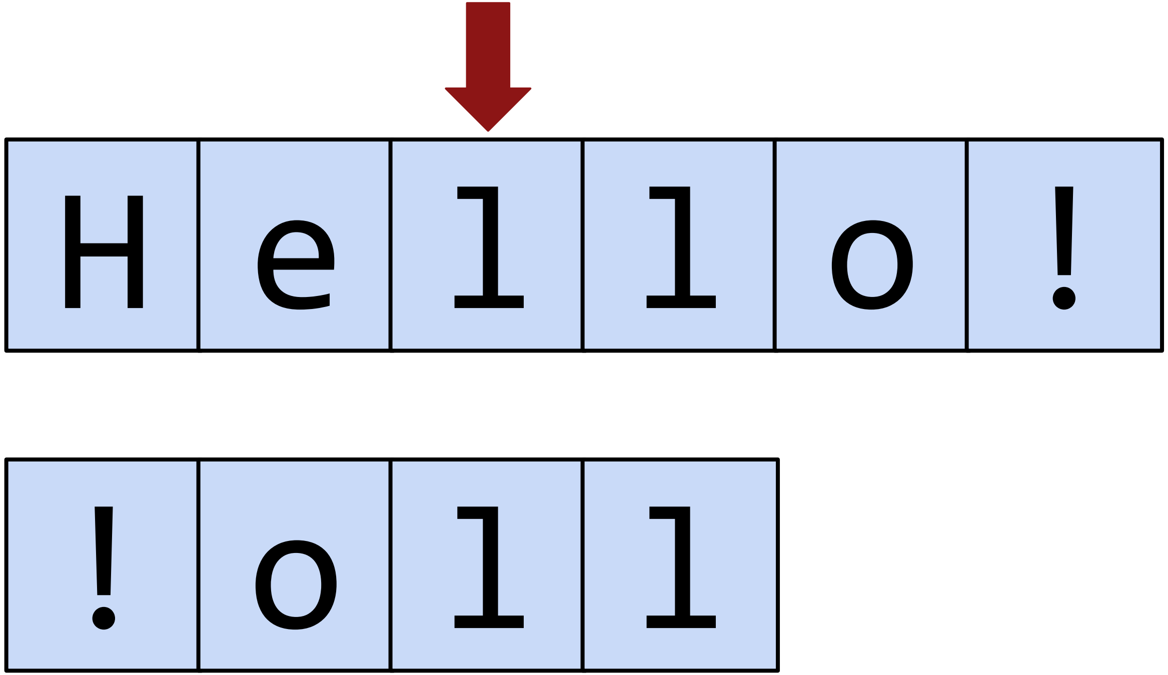
# Reversing a String



| | | | | | |
|---|---|---|---|---|---|
| H | e | l | l | o | ! |

| | | |
|---|---|---|
| ! | o | l |

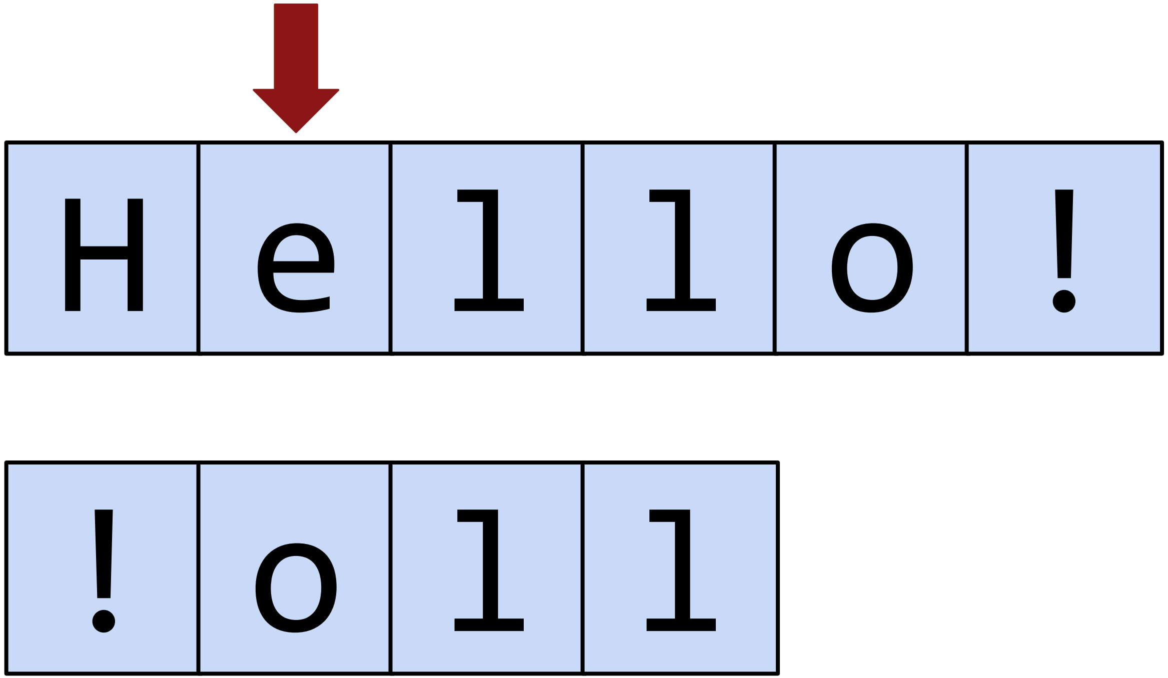
# Reversing a String



|  |  |  |  |  |  |
|---|---|---|---|---|---|
| H | e | l | l | o | ! |

|  |  |  |  |
|---|---|---|---|
| ! | o | l | l |

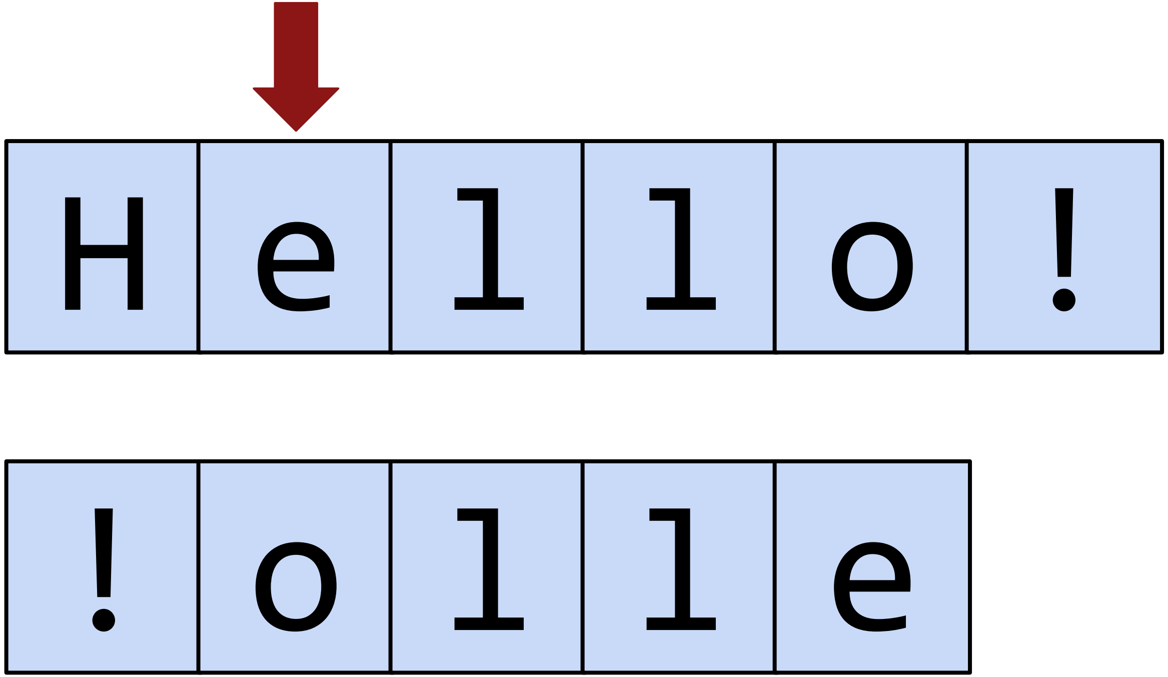
# Reversing a String



| | | | | | |
|---|---|---|---|---|---|
| H | e | l | l | o | ! |

| | | | |
|---|---|---|---|
| ! | o | l | l |

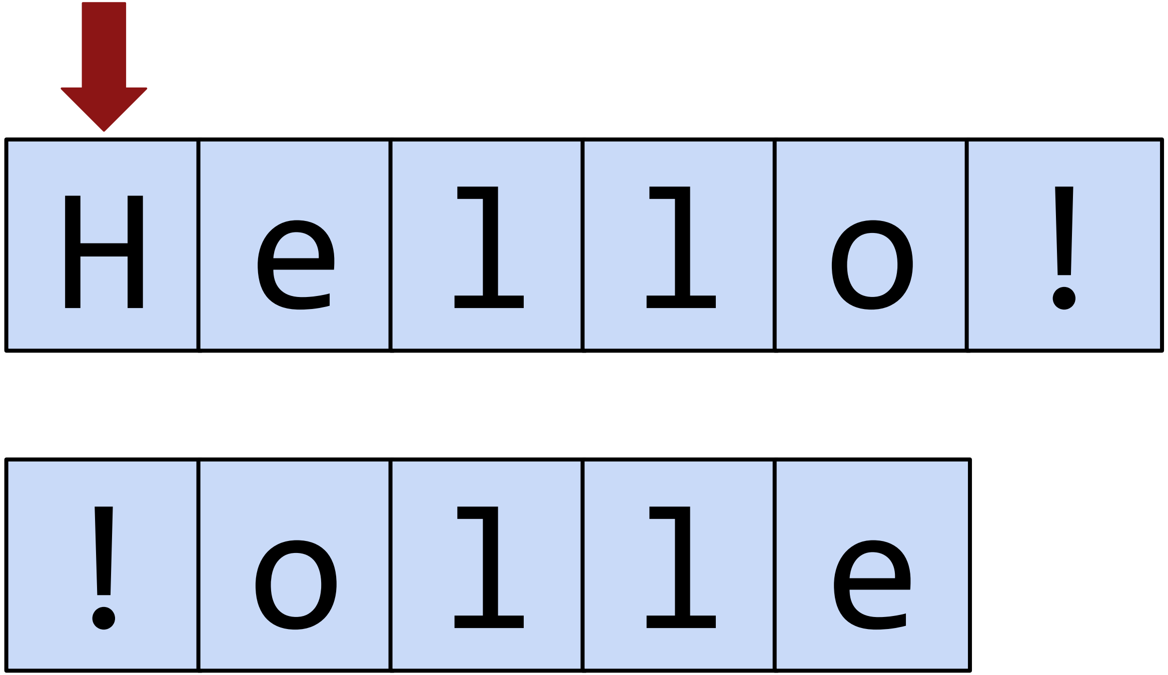
# Reversing a String



| | | | | | |
|---|---|---|---|---|---|
| H | e | l | l | o | ! |

| | | | | |
|---|---|---|---|---|
| ! | o | l | l | e |

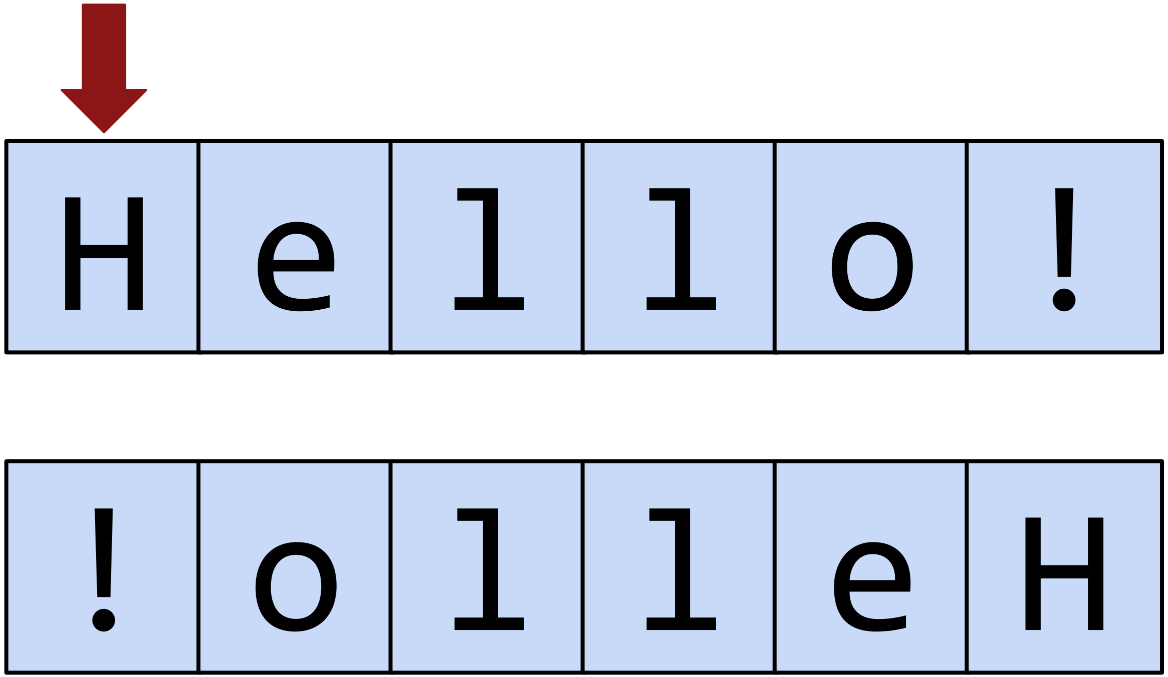
# Reversing a String



| H | e | l | l | o | ! |
|---|---|---|---|---|---|

| ! | o | l | l | e |
|---|---|---|---|---|

# Reversing a String



| | | | | | |
|---|---|---|---|---|---|
| H | e | l | l | o | ! |
| ! | o | l | l | e | H |

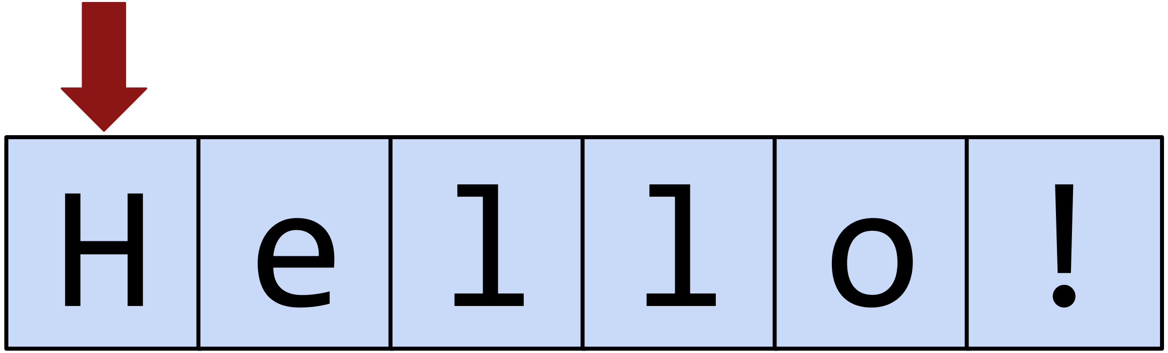
# Reversing a String

| | | | | | |
|---|---|---|---|---|---|
| H | e | l | l | o | ! |

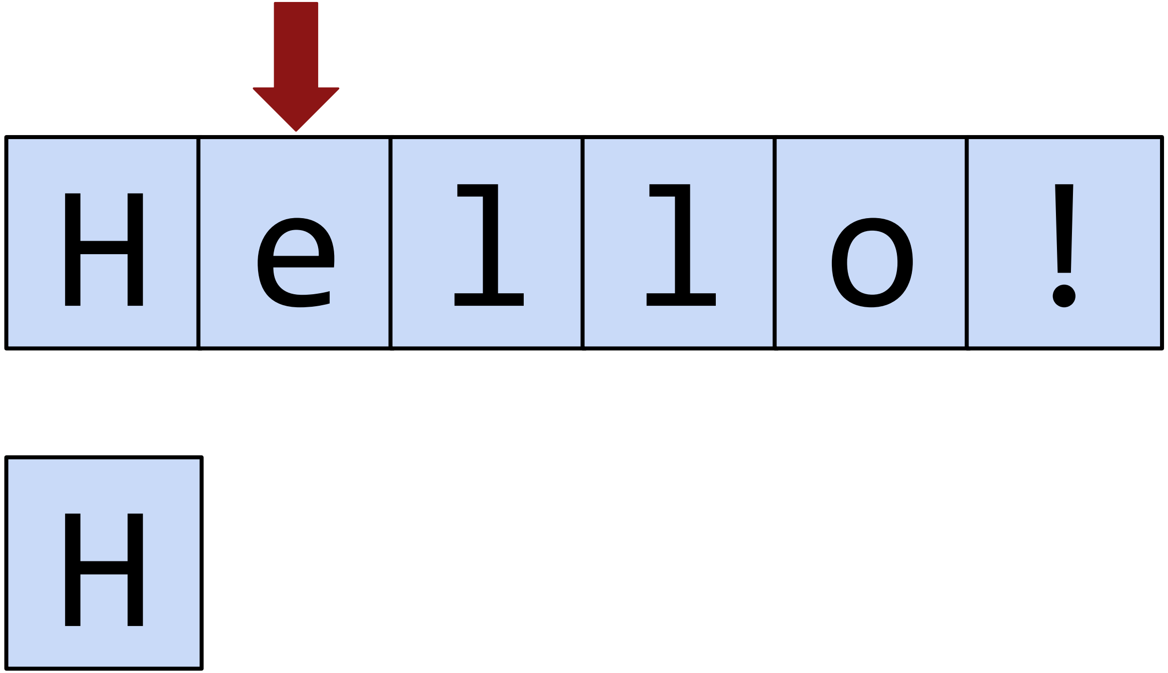
| | | | | | |
|---|---|---|---|---|---|
| ! | o | l | l | e | H |

# Another Take

# Reversing a String



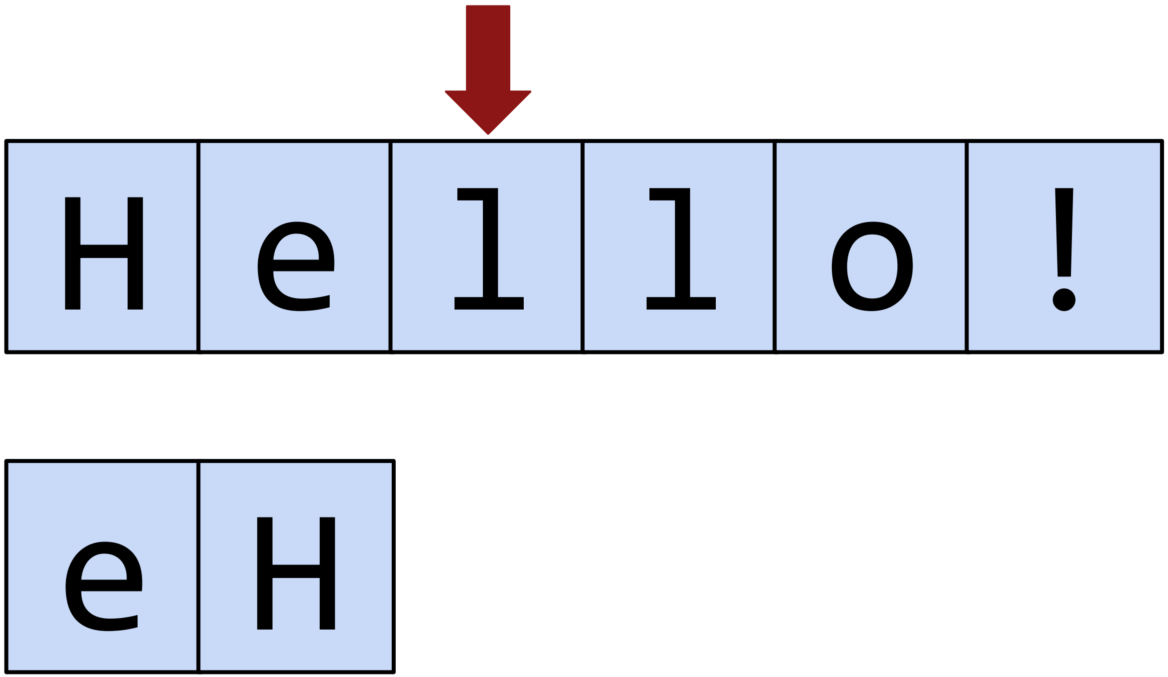
# Reversing a String



| H | e | l | l | o | ! |
|---|---|---|---|---|---|

| H |
|---|

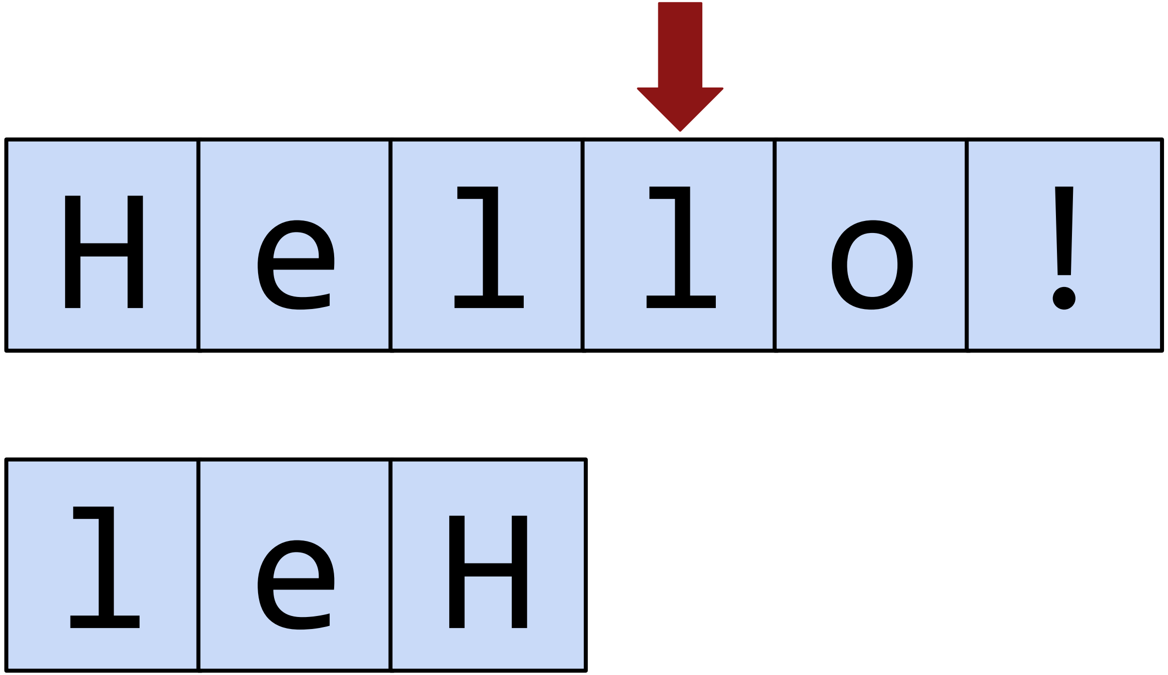
# Reversing a String



| | | | | | |
|---|---|---|---|---|---|
| H | e | l | l | o | ! |

| | |
|---|---|
| e | H |

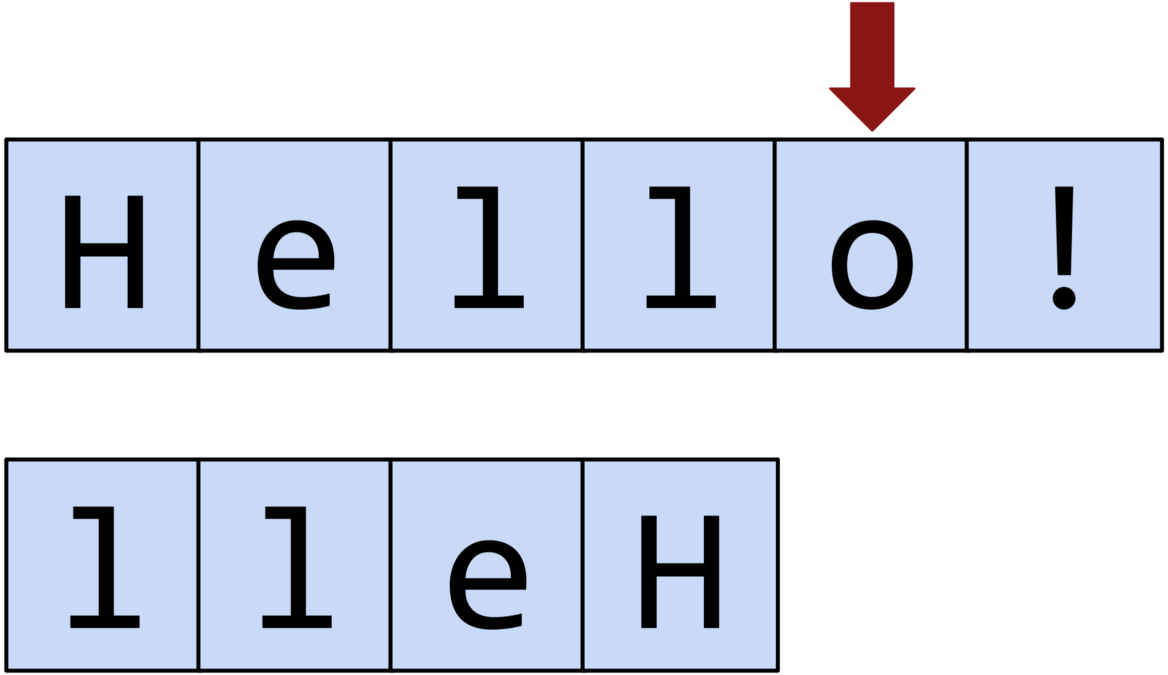
# Reversing a String



| | | | | | |
|---|---|---|---|---|---|
| H | e | l | l | o | ! |

| | | |
|---|---|---|
| l | e | H |

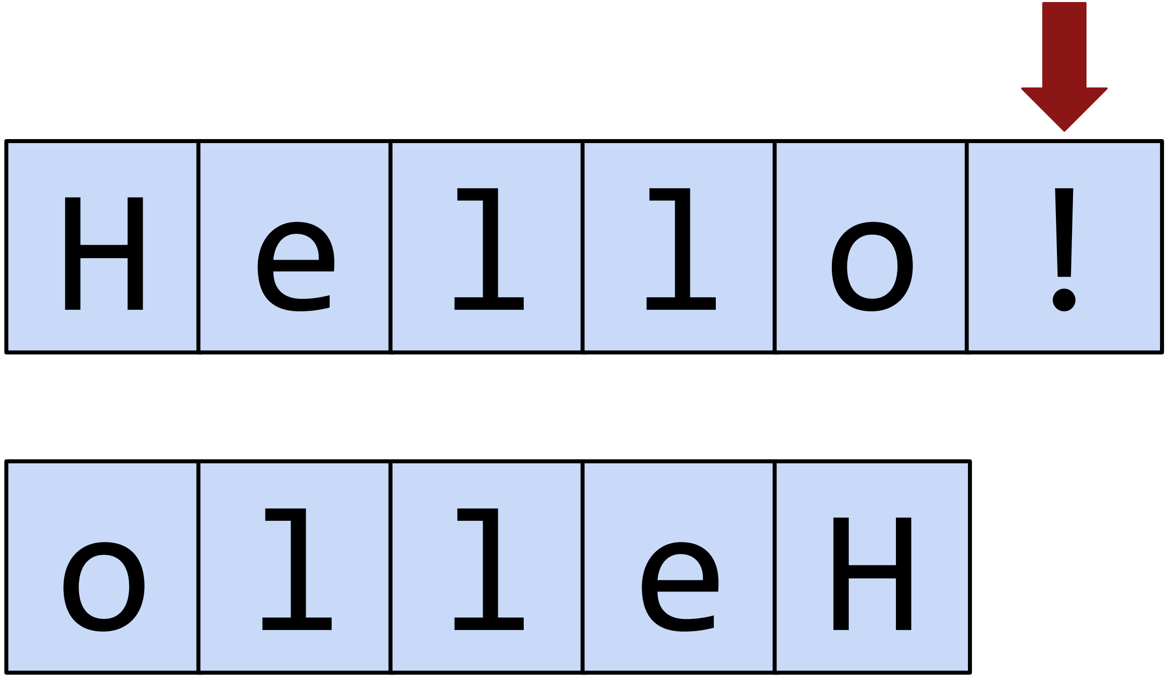
# Reversing a String



| | | | | | |
|---|---|---|---|---|---|
| H | e | l | l | o | ! |

| | | | |
|---|---|---|---|
| l | l | e | H |

# Reversing a String



| | | | | | |
|---|---|---|---|---|---|
| H | e | l | l | o | ! |

| | | | | |
|---|---|---|---|---|
| o | l | l | e | H |

# Reversing a String

| | | | | | |
|---|---|---|---|---|---|
| H | e | l | l | o | ! |

| | | | | | |
|---|---|---|---|---|---|
| ! | o | l | l | e | H |

# Plan for Today

- Review: Characters and Strings
- Looping over Strings
- Practice: Reversing a String
- Practice: Palindromes
- Practice: Caesar Cipher

# Exercise: Palindromes

Let's write a method called `isPalindrome` that takes one String parameter, and returns whether or not that String is a palindrome (the same forwards and backwards)

```
isPalindrome("racecar")   -> true
isPalindrome("hi there") -> false
isPalindrome("kayak")     -> true
```

# Let's Code It!

# More Palindromes

Here are some palindromes in other languages:

- ب قلعة تحت تعلق بلح (Dates hang underneath a castle in Halab)
- 여보 , 안경 안보여 (Honey, I can't see my glasses)
- कड़क (a loud thunderous sound)
- 上海自來水來自海上 (Shanghai tap water originates from "above" the ocean)

Do you know a palindrome in another language?

# Stress Test

A man, a plan, a caret, a ban, a myriad, a sum, a lac, a liar, a hoop, a pint, a catalpa, a gas, an oil, a bird, a yell, a vat, a caw, a pax, a wag, a tax, a nay, a ram, a cap, a yam, a gay, a tsar, a wall, a car, a luger, a ward, a bin, a woman, a vassal, a wolf, a tuna, a nit, a pall, a fret, a watt, a bay, a daub, a tan, a cab, a datum, a gall, a hat, a tag, a zap, a say, a jaw, a lay, a wet, a gallop, a tug, a trot, a trap, a tram, a torr, a caper, a top, a tonk, a toll, a ball, a fair, a sax, a minim, a tenor, a bass, a passer, a capital, a rut, an amen, a ted, a cabal, a tang, a sun, an ass, a maw, a sag, a jam, a dam, a sub, a salt, an axon, a sail, an ad, a wadi, a radian, a room, a rood, a rip, a tad, a pariah, a revel, a reel, a reed, a pool, a plug, a pin, a peek, a parabola, a dog, a pat, a cud, a nu, a fan, a pal, a rum, a nod, an eta, a lag, an eel, a batik, a mug, a mot, a nap, a maxim, a mood, a leek, a grub, a gob, a gel, a drab, a citadel, a total, a cedar, a tap, a gag, a rat, a manor, a bar, a gal, a cola, a pap, a yaw, a tab, a raj, a gab, a nag, a pagan, a bag, a jar, a bat, a way, a papa, a local, a gar, a baron, a mat, a rag, a gap, a tar, a decal, a tot, a led, a tic, a bard, a leg, a bog, a burg, a keel, a doom, a mix, a map, an atom, a gum, a kit, a baleen, a gala, a ten, a don, a mural, a pan, a faun, a ducat, a pagoda, a lob, a rap, a keep, a nip, a gulp, a loop, a deer, a leer, a lever, a hair, a pad, a tapir, a door, a moor, an aid, a raid, a wad, an alias, an ox, an atlas, a bus, a madam, a jag, a saw, a mass, an anus, a gnat, a lab, a cadet, an em, a natural, a tip, a caress, a pass, a baronet, a minimax, a sari, a fall, a ballot, a knot, a pot, a rep, a carrot, a mart, a part, a tort, a gut, a poll, a gateway, a law, a jay, a sap, a zag, a tat, a hall, a gamut, a dab, a can, a tabu, a day, a batt, a waterfall, a patina, a nut, a flow, a lass, a van, a mow, a nib, a draw, a regular, a call, a war, a stay, a gam, a yap, a cam, a ray, an ax, a tag, a wax, a paw, a cat, a valley, a drib, a lion, a saga, a plat, a catnip, a pooh, a rail, a calamus, a dairyman, a bater, a canal – Panama!

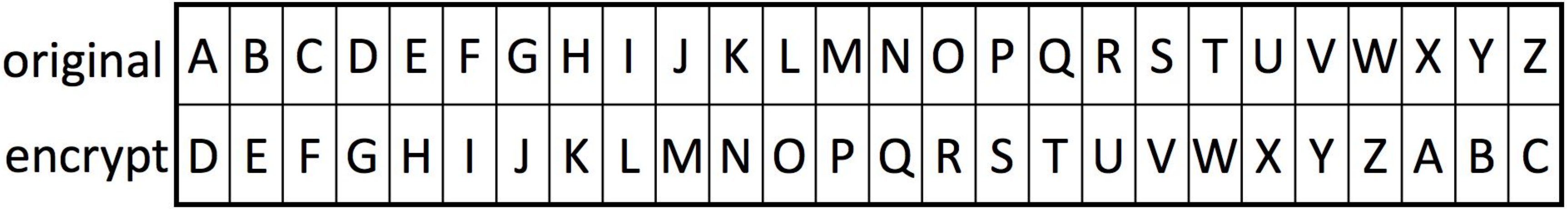
# Plan for Today

- Review: Characters and Strings
- Looping over Strings
- Practice: Reversing a String
- Practice: Palindromes
- Practice: Caesar Cipher

# Exercise: Caesar Cipher

Let’s write a program that encrypts text using a Caesar Cipher! In a Caesar Cipher:

- Rotate text by n letters → this is the key (n=3 below)
- Wrap-around at the end
- Substitute letters based on this mapping

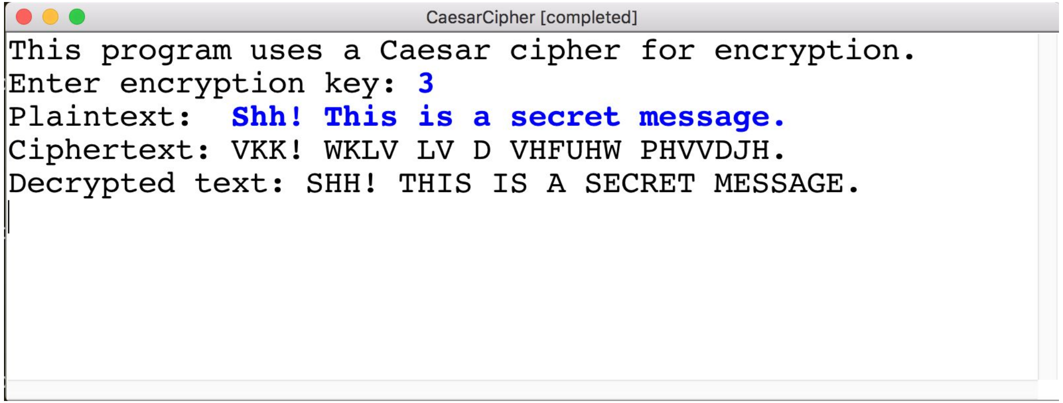


| | | | | | | | | | | | | | | | | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| original | A | B | C | D | E | F | G | H | I | J | K | L | M | N | O | P | Q | R | S | T | U | V | W | X | Y | Z |
| encrypt | D | E | F | G | H | I | J | K | L | M | N | O | P | Q | R | S | T | U | V | W | X | Y | Z | A | B | C |

# Exercise: Caesar Cipher

- rotate the alphabet by a certain key, with wrapping.

| | | | | | | | | | | | | | | | | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| original | A | B | C | D | E | F | G | H | I | J | K | L | M | N | O | P | Q | R | S | T | U | V | W | X | Y | Z |
| encrypt | D | E | F | G | H | I | J | K | L | M | N | O | P | Q | R | S | T | U | V | W | X | Y | Z | A | B | C |



```
This program uses a Caesar cipher for encryption.
Enter encryption key: 3
Plaintext:  Shh! This is a secret message.
Ciphertext: VKK! WKLV LV D VHFUHW PHVVDJH.
Decrypted text: SHH! THIS IS A SECRET MESSAGE.
```

# Let's Code It!

# Extra Practice: Recall Char Loops

```
// prints the characters a to z
for (char ch = 'a'; ch <= 'z'; ch++) {
    println(ch);
}
```

# Extra Practice: Passcodes

- So, Duke forgot his password
- It’s 3 characters long
- Each character is between a and e
- How can we use **char loops** to generate all possible passwords for Duke to try?

aaa
aab
aac
...
eed
eee

# Plan for Today

- Review: Characters and Strings
- Looping over Strings
- Practice: Reversing a String
- Practice: Palindromes
- Practice: Caesar Cipher

**Next Time:** How can we read data from a file?