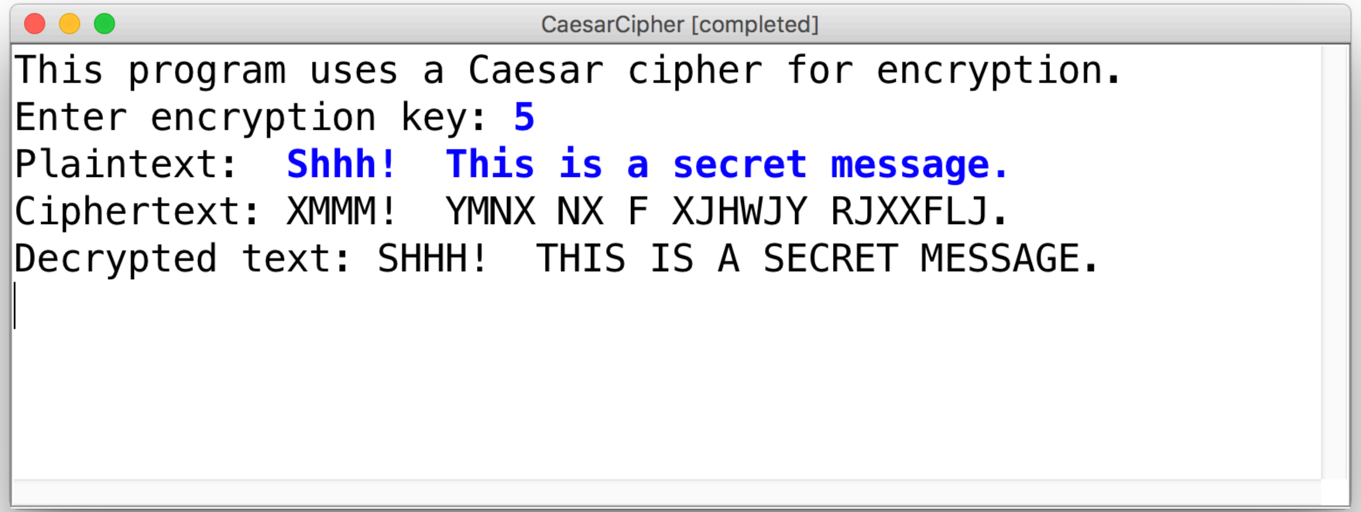
CS 106A, Lecture 9
Problem-Solving with Strings

suggested reading:

Java Ch. 8.5

Learning Goals

- 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: 5
Plaintext:  Shhh!  This is a secret message.
Ciphertext: XMMM!  YMNX NX F XJHWJY RJXXFLJ.
Decrypted text: SHHH!  THIS IS A SECRET MESSAGE.
```

Plan For Today

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

Plan For Today

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

Text Processing

Char

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

```
char letterA = 'A';
char plus = '+';
char zero = '0';
char space = ' ';
char newLine = '\n';
char tab = '\t';
char singleQuote = '\'';
char backSlash = '\\';
```

Char

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

- Uppercase letters are sequentially numbered
- Lowercase letters are sequentially numbered
- Digits are sequentially numbered

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

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;
int diff = 'c' - 'a';                 // 2
int numLettersInAlphabet = 'z' – 'a' + 1;
// or
int numLettersInAlphabet = 'Z' – 'A' + 1;
```

Side Note: 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

Method	Description
`Character.isDigit(`***`ch`***`)`	true if *ch* is `'0'` through `'9'`
`Character.isLetter(`***`ch`***`)`	true if *ch* is `'a'` through `'z'` or `'A'` through `'Z'`
`Character.isLetterOrDigit(`***`ch`***`)`	true if *ch* is `'a'` through `'z'`, `'A'` through `'Z'` or `'0'` through `'9'`
`Character.isLowerCase(`***`ch`***`)`	true if *ch* is `'a'` through `'z'`
`Character.isUpperCase(`***`ch`***`)`	true if *ch* is `'A'` through `'Z'`
`Character.toLowerCase(`***`ch`***`)`	returns lowercase equivalent of a letter
`Character.toUpperCase(`***`ch`***`)`	returns uppercase equivalent of a letter
`Character.isWhitespace(`***`ch`***`)`	true if *ch* is a space, tab, new line, etc.

Remember: these **return** the new char, they cannot modify an existing char!

Strings

A **`String`** is a variable type representing a sequence of characters.

```
String text = "Hi parents!";
```

index	*0*	*1*	*2*	*3*	*4*	*5*	*6*	*7*	*8*	*9*	*10*
character	`'H'`	`'i'`	`' '`	`'p'`	`'a'`	`'r'`	`'e'`	`'n'`	`'t'`	`'s'`	`'!'`

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

Strings vs. Chars

Remember: chars and length-1 strings are different!

`char ch = 'A'` **DIFFERENT FROM** `String str = "A"`

Creating Strings

```
String str = "Hello, world!";
String empty = "";
println(str);

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

// String concatenation (using “+”)
String message = 2 + " cool " + 2 + " handle";
int x = 2;
println("x has the value " + x);
```

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; // ✔
```

String Methods

Method name	Description
`s.length()`	number of characters in this string
`s.charAt(`***`index`***`)`	char at the given index
`s.indexOf(`***`str`***`)`	index where the start of the given string appears in this string (-1 if not found)
`s.substring(`***`index1, index2`***`)` or `s.substring(`***`index1`***`)`	the characters in this string from *index1* (inclusive) to *index2* (exclusive); if *index2* is omitted, goes until end
`s.toLowerCase()`	a new string with all lowercase letters
`s.toUpperCase()`	a new string with all uppercase letters

- These methods are called using **dot notation:**

```
String className = "CS 106A yay!";
println(className.length());   // 12
```

Substrings

A *substring* is a subset of a string.

```
String str = "Hello, world!";
String hello = str.substring(0, 5);
```

0	1	2	3	4	5	6	7	8	9	10	11	12
'H'	'e'	'l'	'l'	'o'	','	' '	'w'	'o'	'r'	'l'	'd'	'!'

Substrings

A *substring* is a subset of a string.

```
String str = "Hello, world!";
String worldExclm = str.substring(7); // to end
```

0	*1*	*2*	*3*	*4*	*5*	*6*	*7*	*8*	*9*	*10*	*11*	*12*
'H'	'e'	'l'	'l'	'o'	','	' '	'w'	'o'	'r'	'l'	'd'	'!'

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**

Plan For Today

- Recap: 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.

```
String str = "Hello!";
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
String str = "Hello!";
for (int i = 0; i < str.length(); i++) {
     char ch = str.charAt(i);
     println(ch);
}
```

Looping Over Strings

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

```
// 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

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

```
// 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!
```

Building Up New 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 = "";
for (int i = 0; i < 5; i++) {
      str += i;
}
println(str);            // 012345
```

Plan For Today

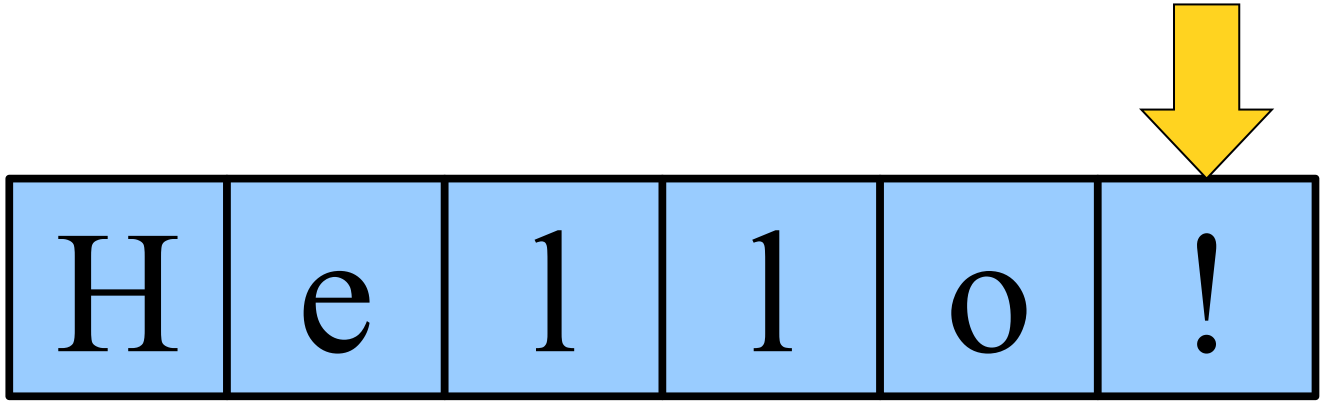
- Recap: Characters and Strings
- Looping over Strings
- Practice: Reversing a String
- Practice: Palindromes
- Practice: Caesar Cipher

Exercise: Reversing a String

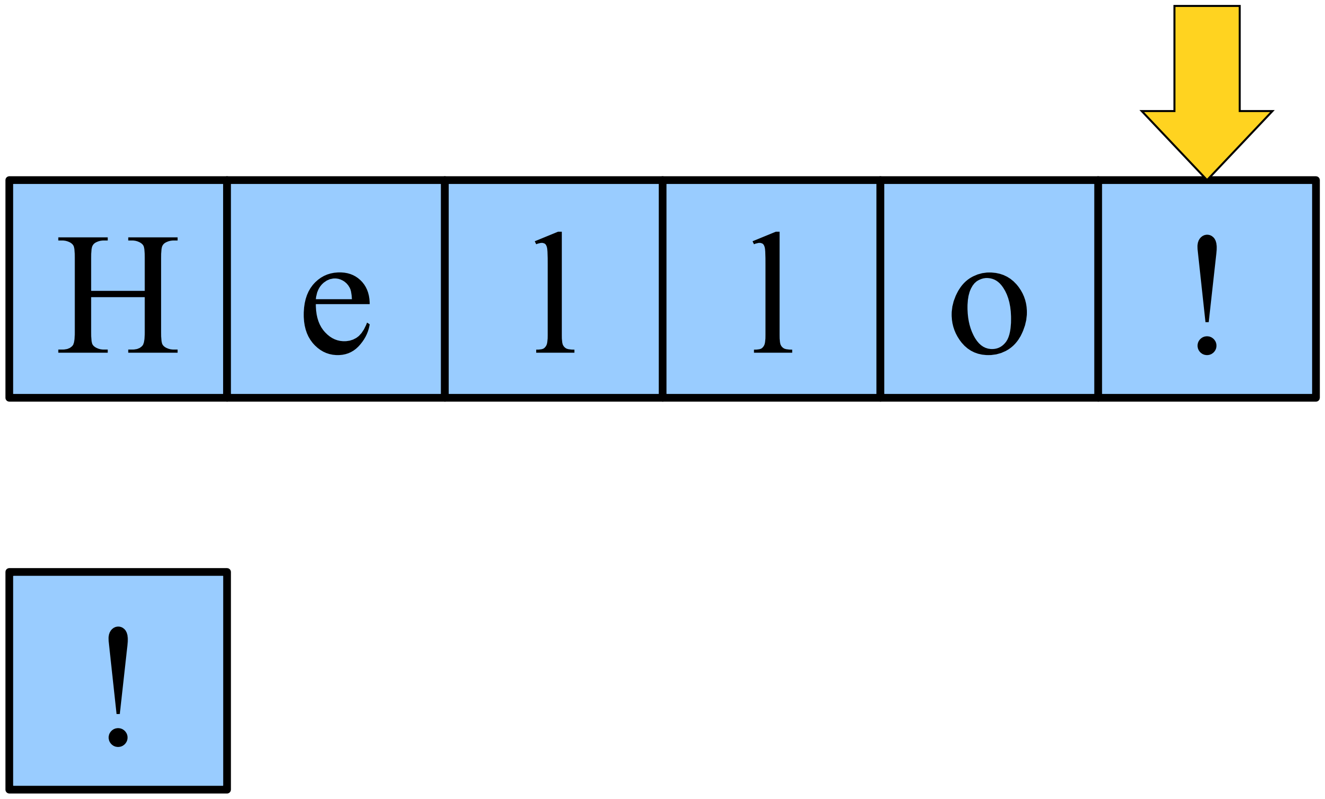
Let’s write a method called **`reverseString`** that takes one String parameter, and returns a new String with the characters in the opposite order.

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

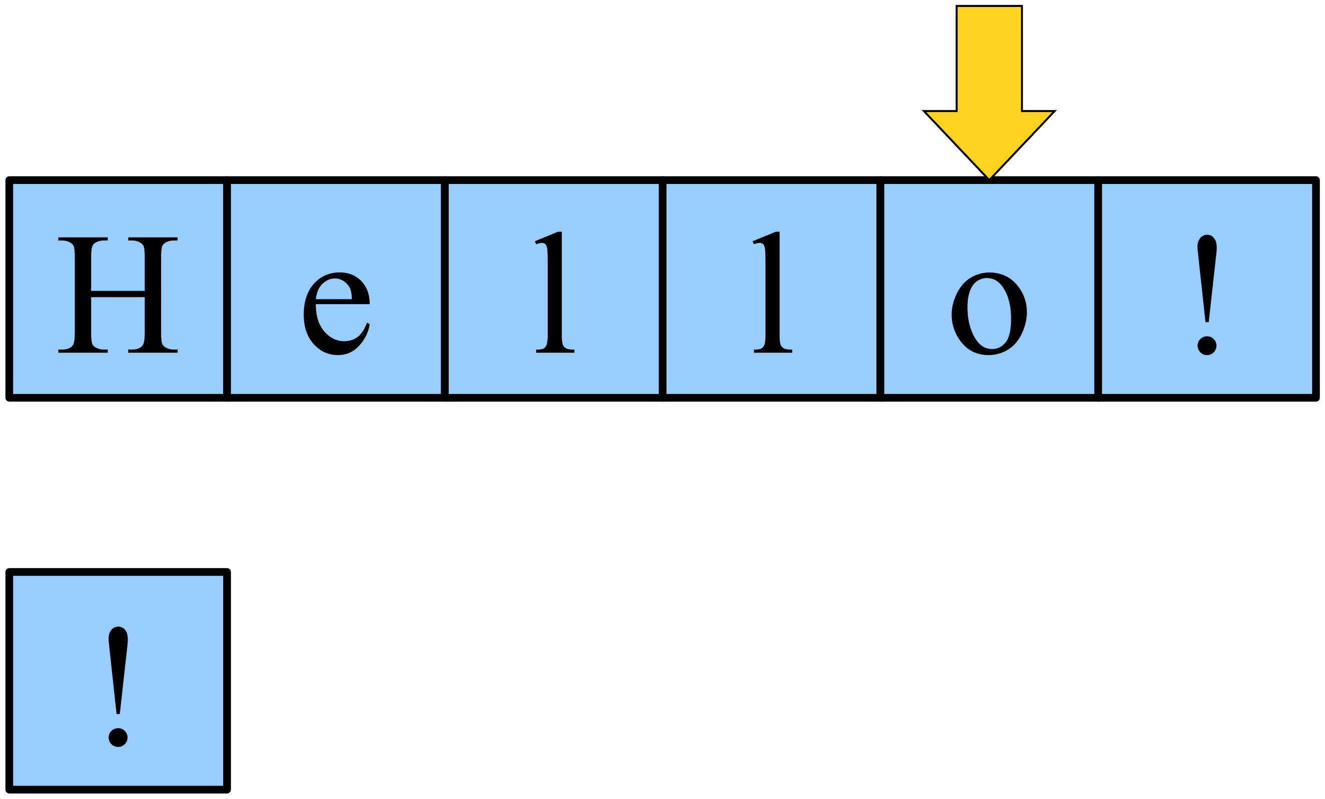
Reversing a String



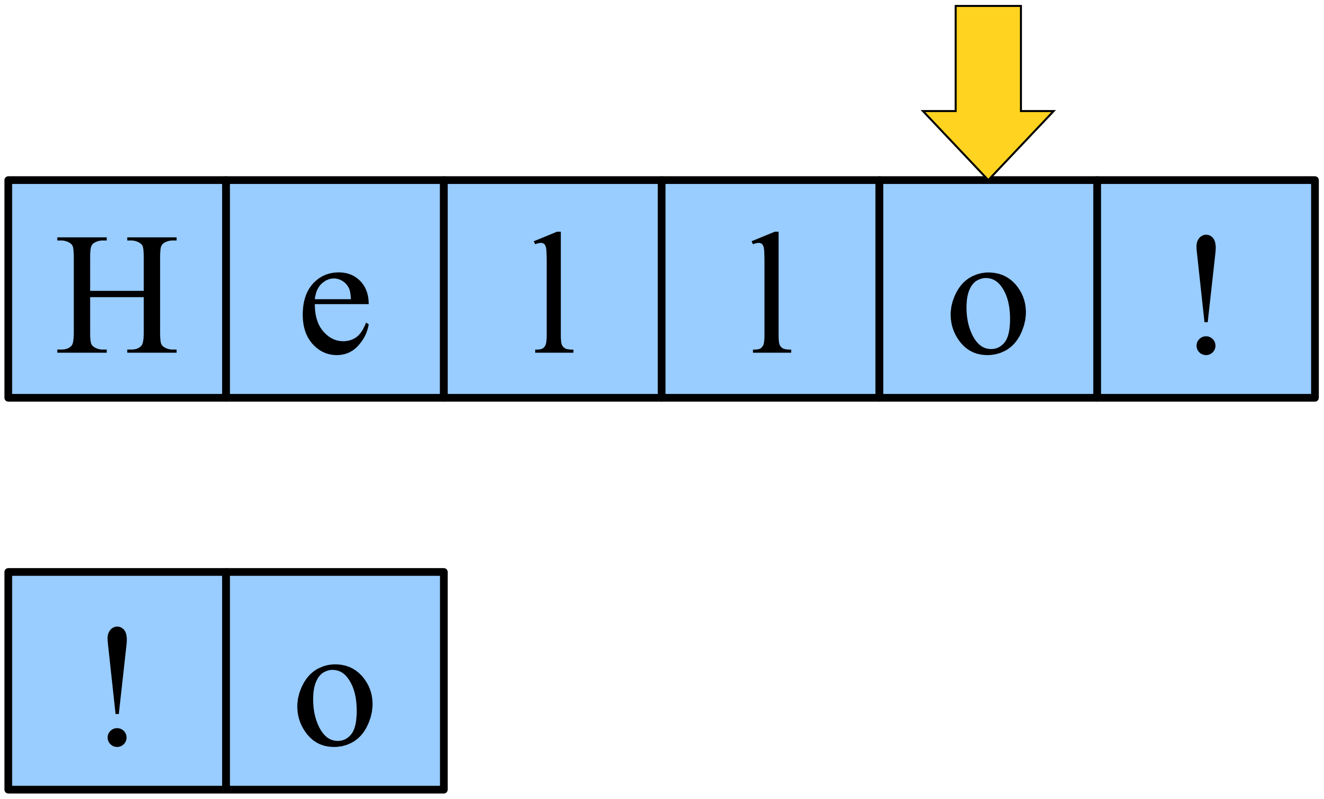
Reversing a String



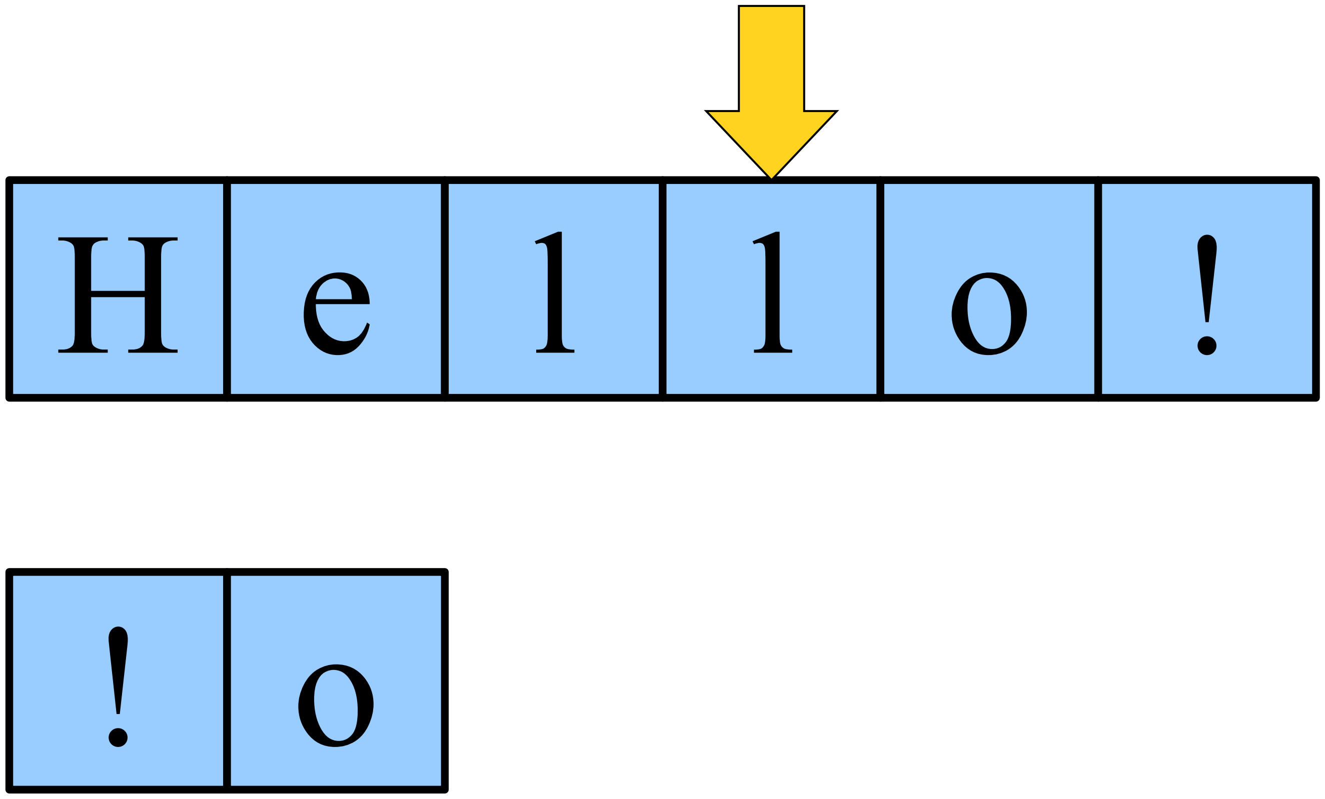
Reversing a String



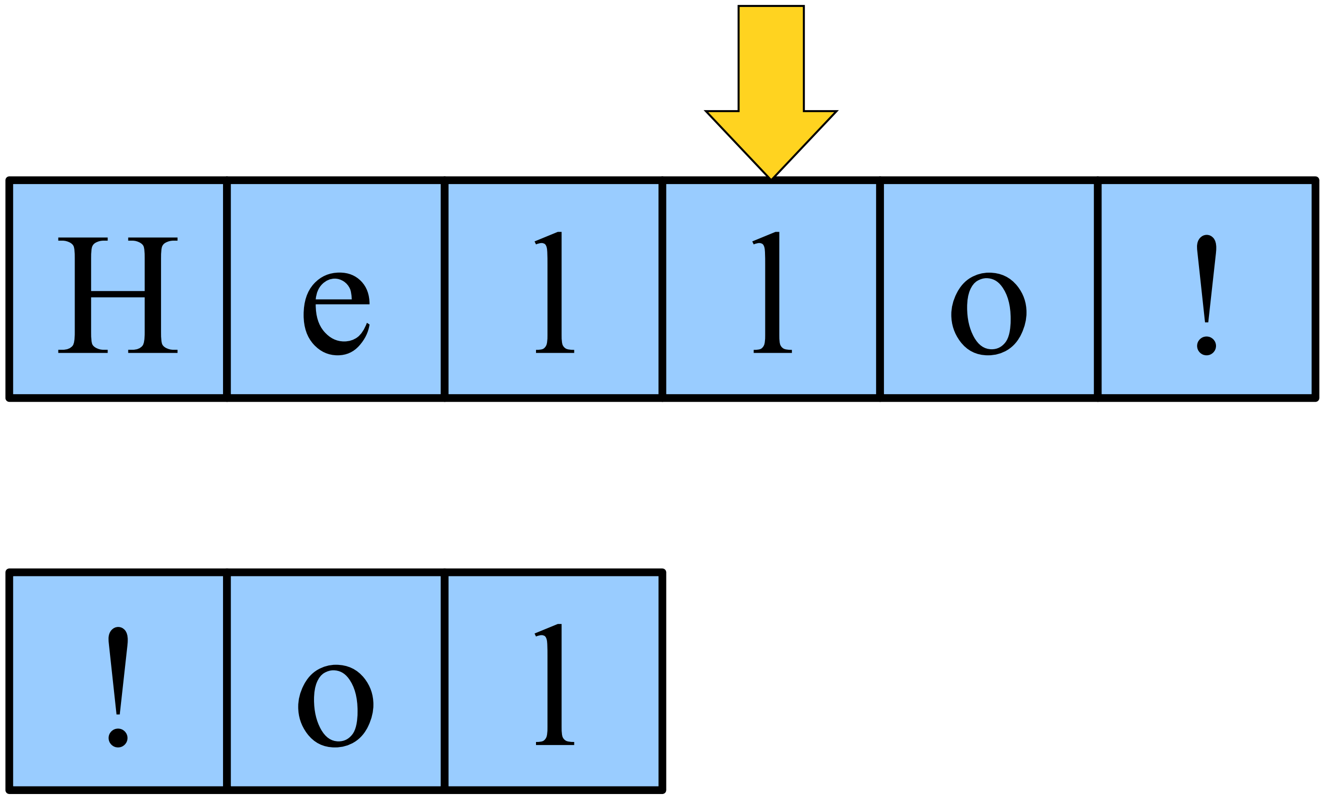
Reversing a String



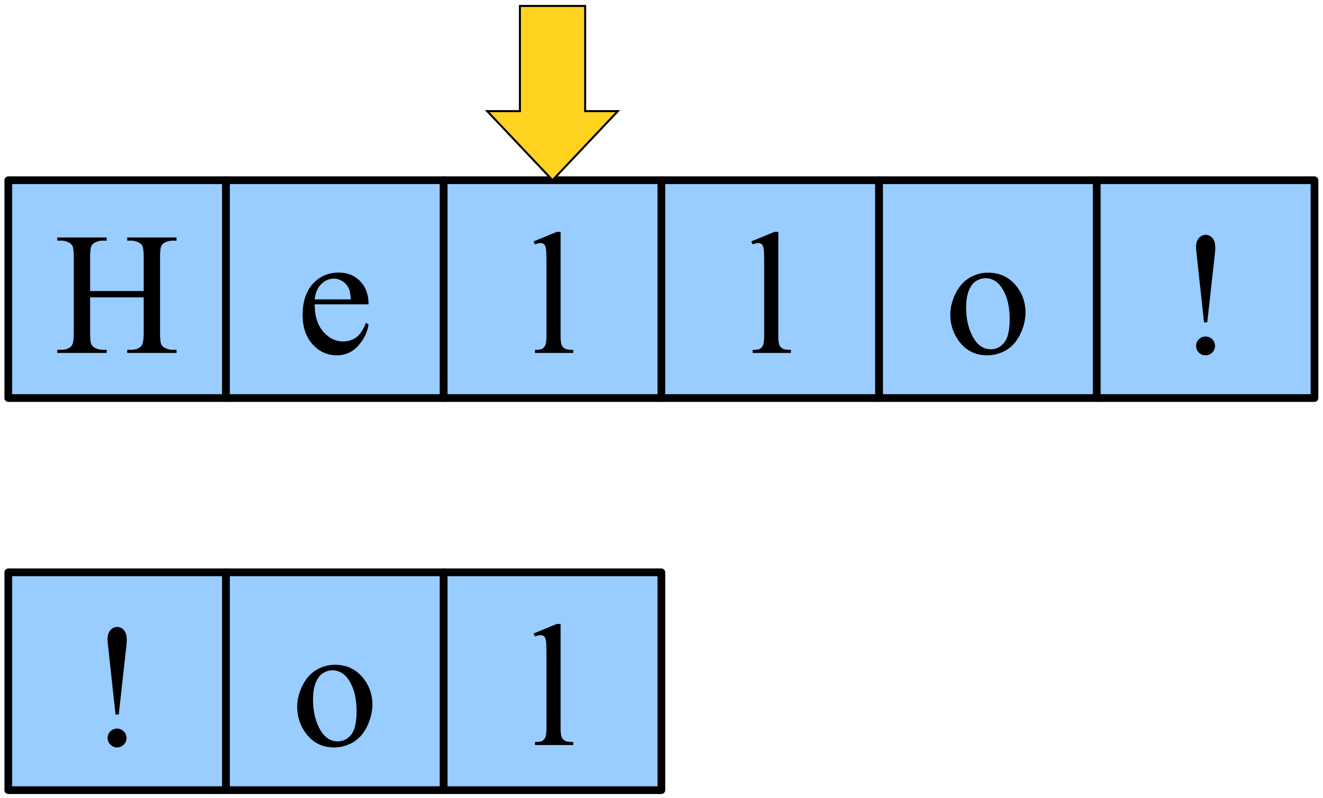
Reversing a String



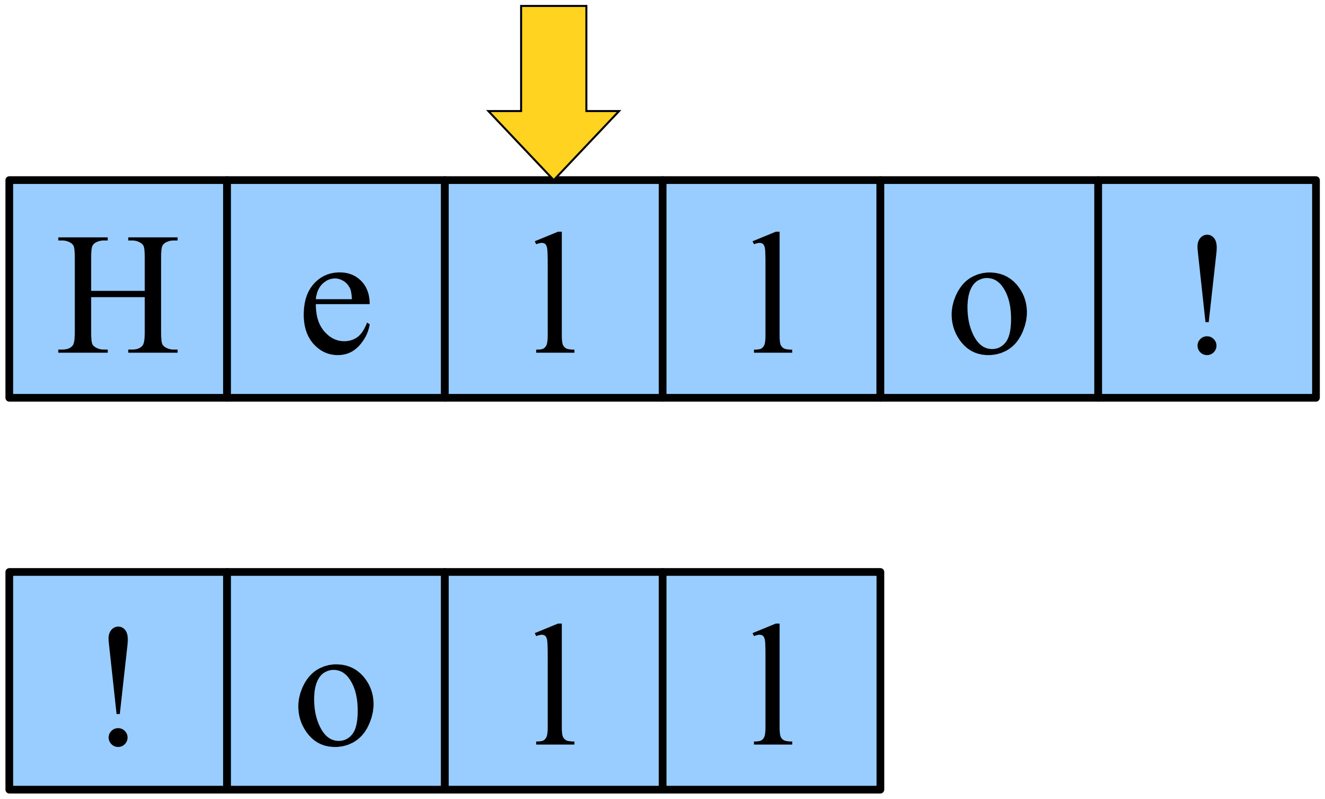
Reversing a String



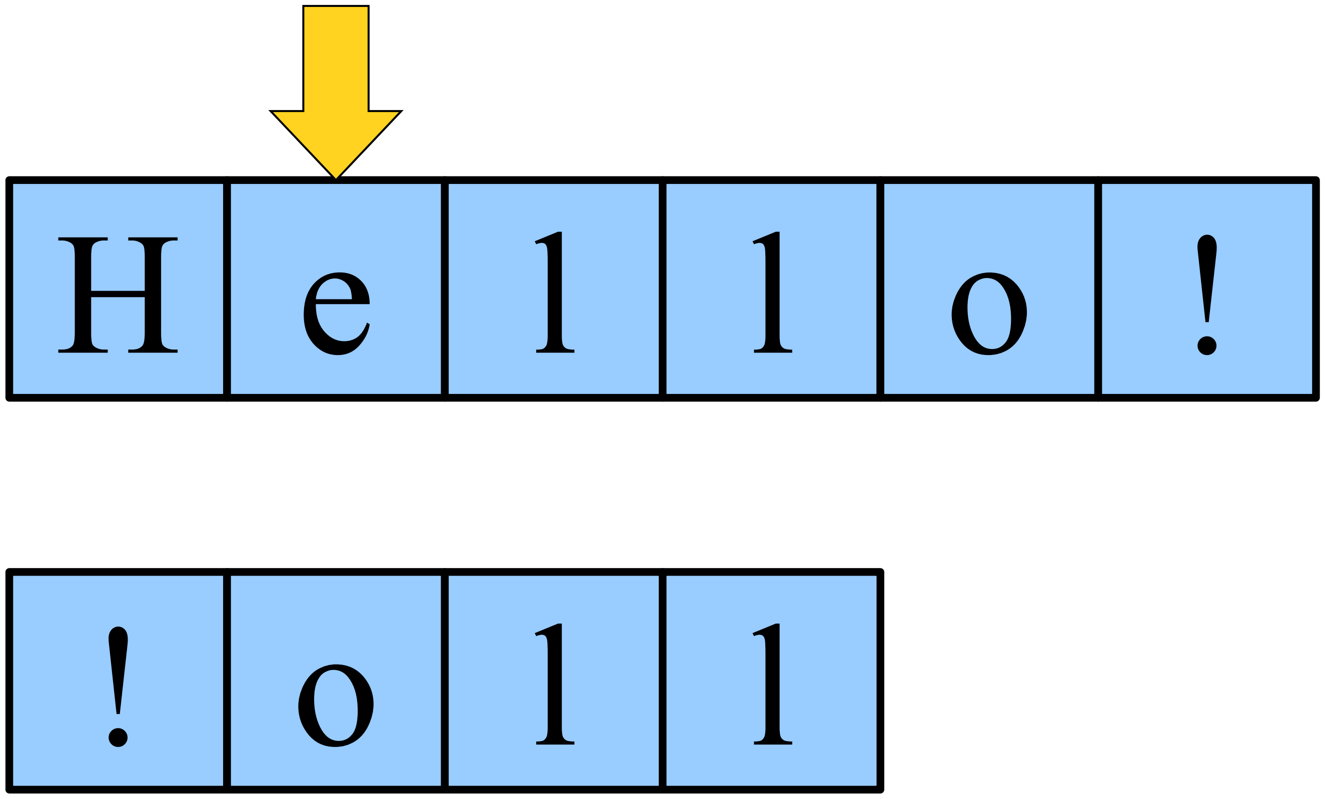
Reversing a String



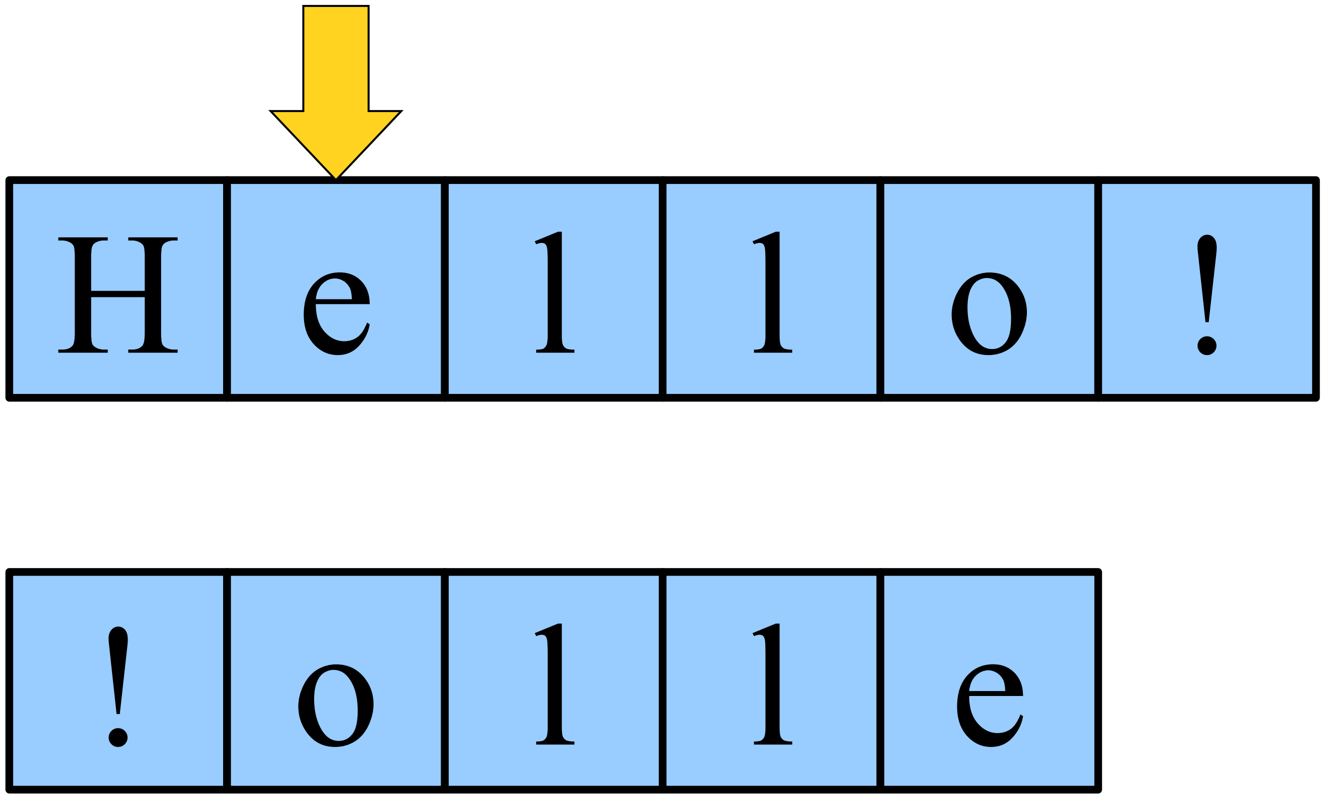
Reversing a String



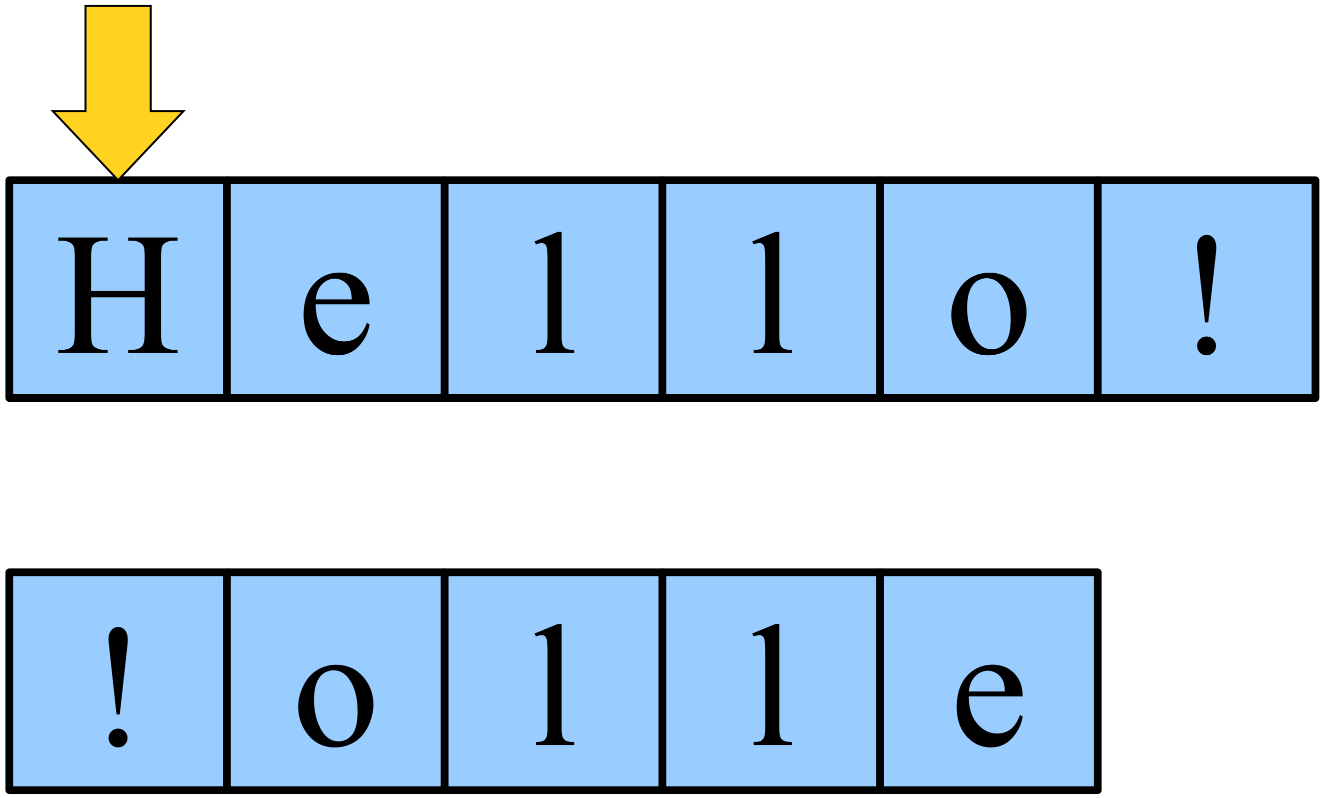
Reversing a String



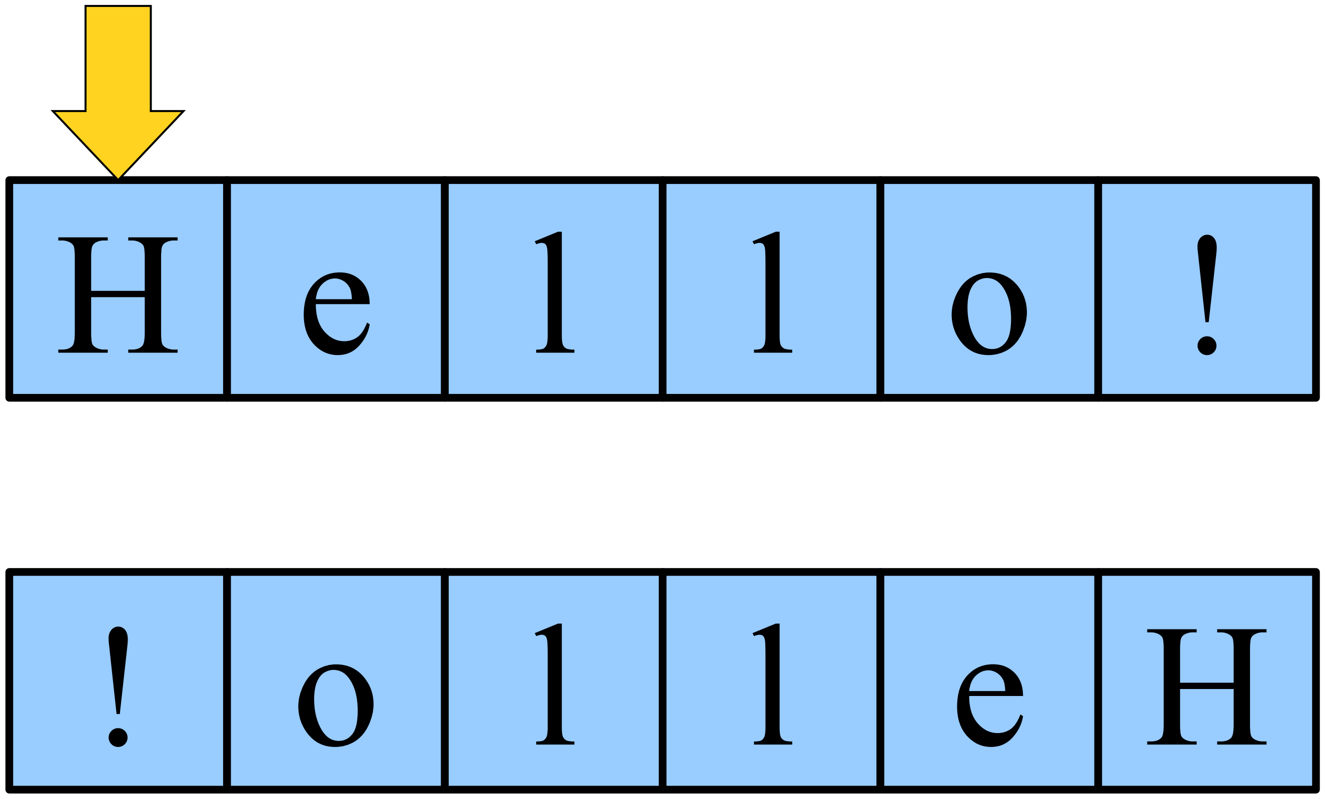
Reversing a String



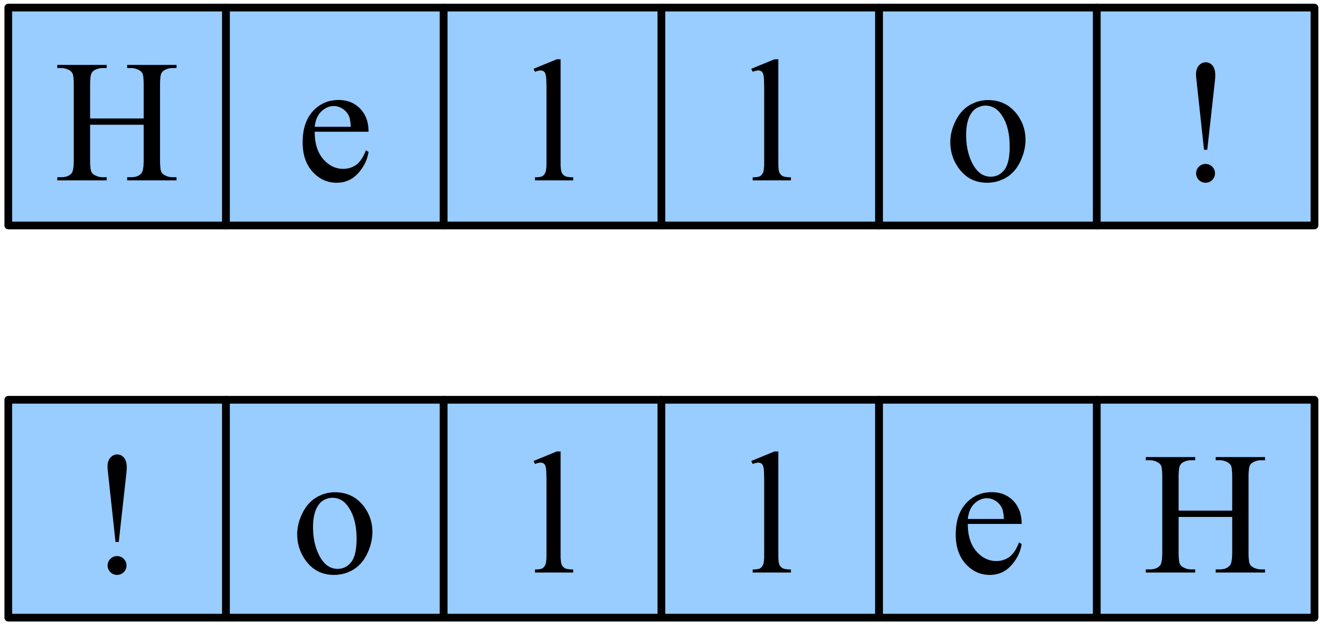
Reversing a String



Reversing a String



Reversing a String

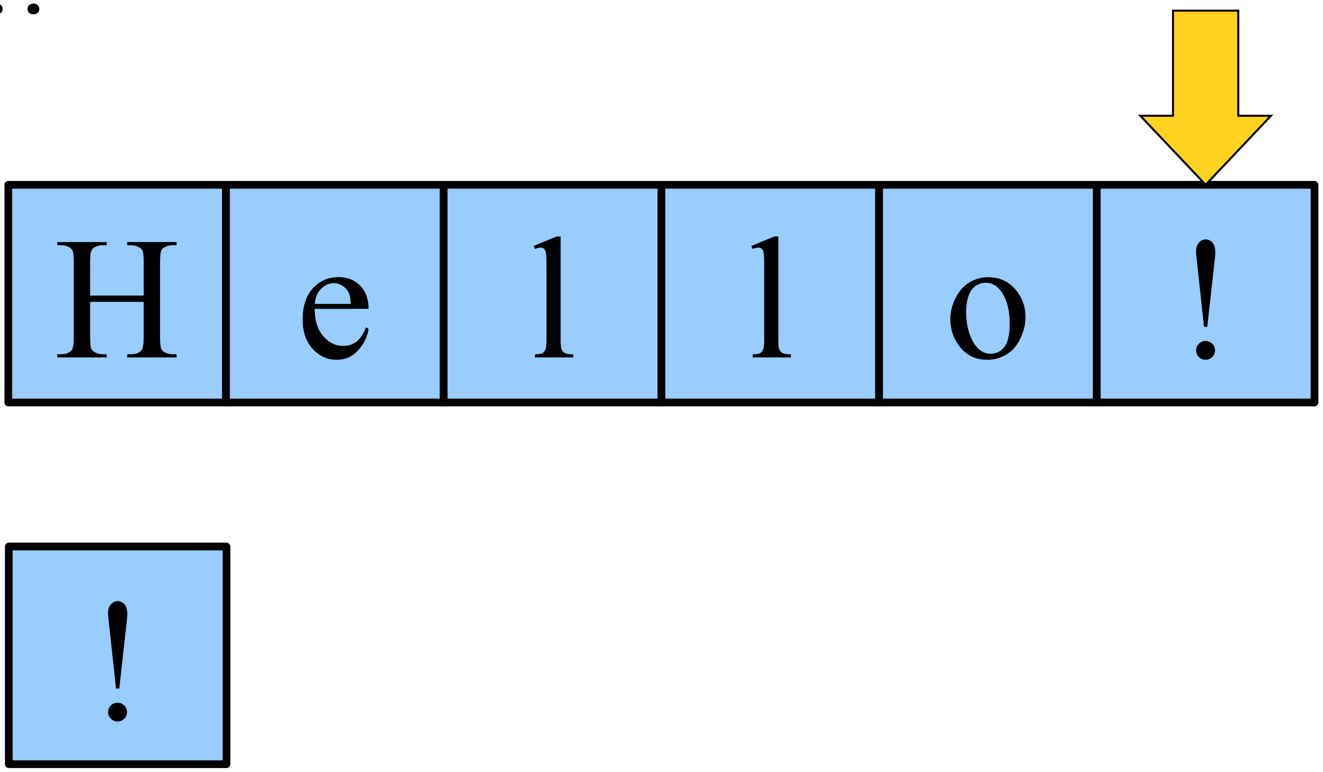


Reversing a String

```
String str = "Hello!";
String newStr = "";
for (??? ; ??? ; ???) {
      ...
}
```

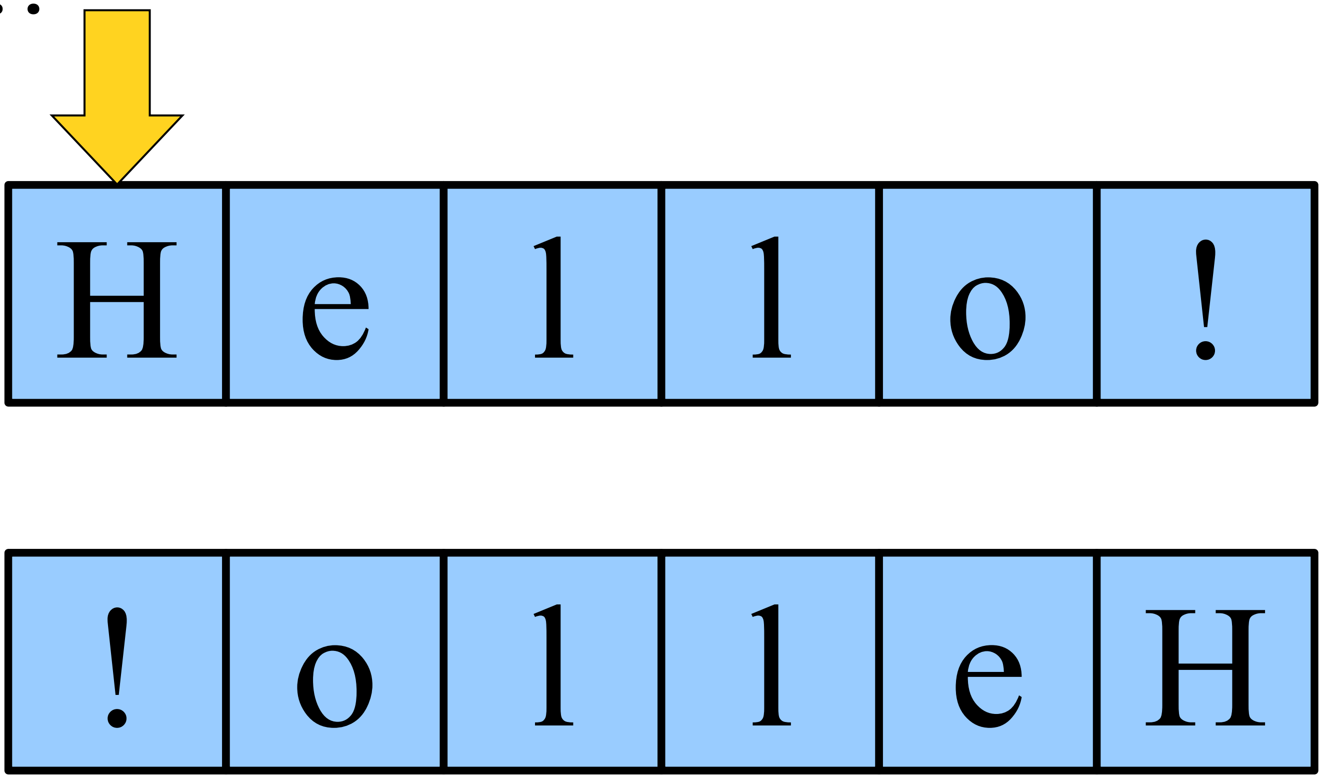
Reversing a String

```
String str = "Hello!";
String newStr = "";
for (int i = str.length() - 1; ??? ; ???) {
      ...
}
```



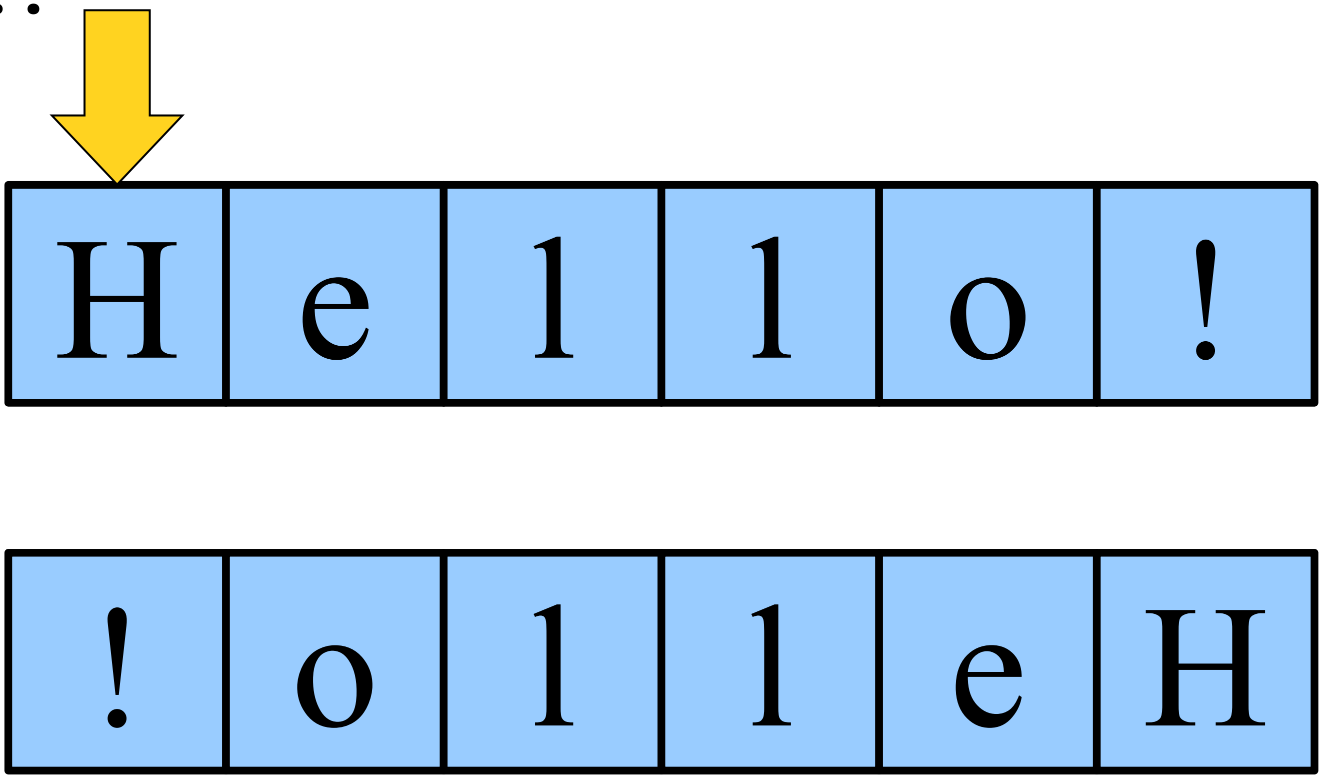
Reversing a String

```
String str = "Hello!";
String newStr = "";
for (int i = str.length() - 1; i >= 0; ???) {
      ...
}
```



Reversing a String

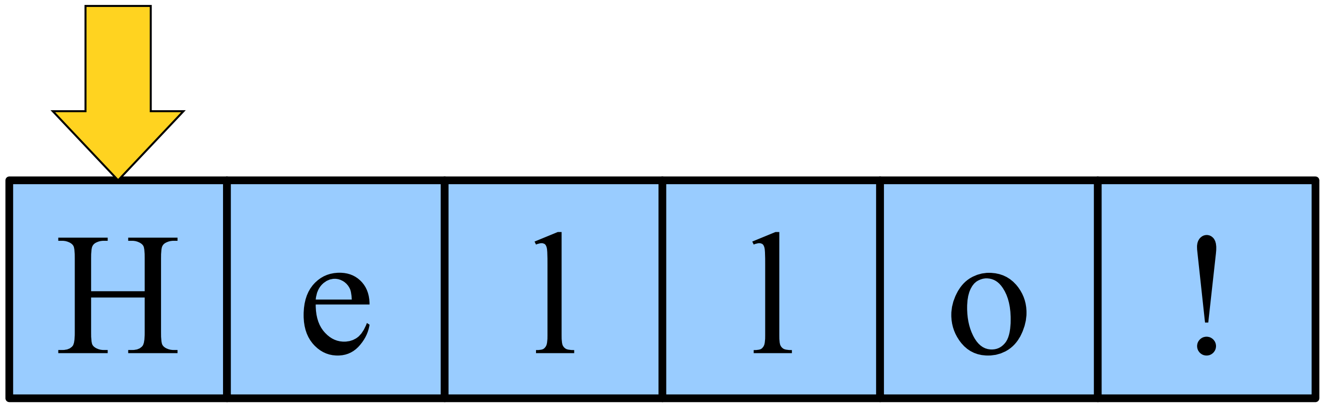
```
String str = "Hello!";
String newStr = "";
for (int i = str.length() - 1; i >= 0; i--) {
      ...
}
```



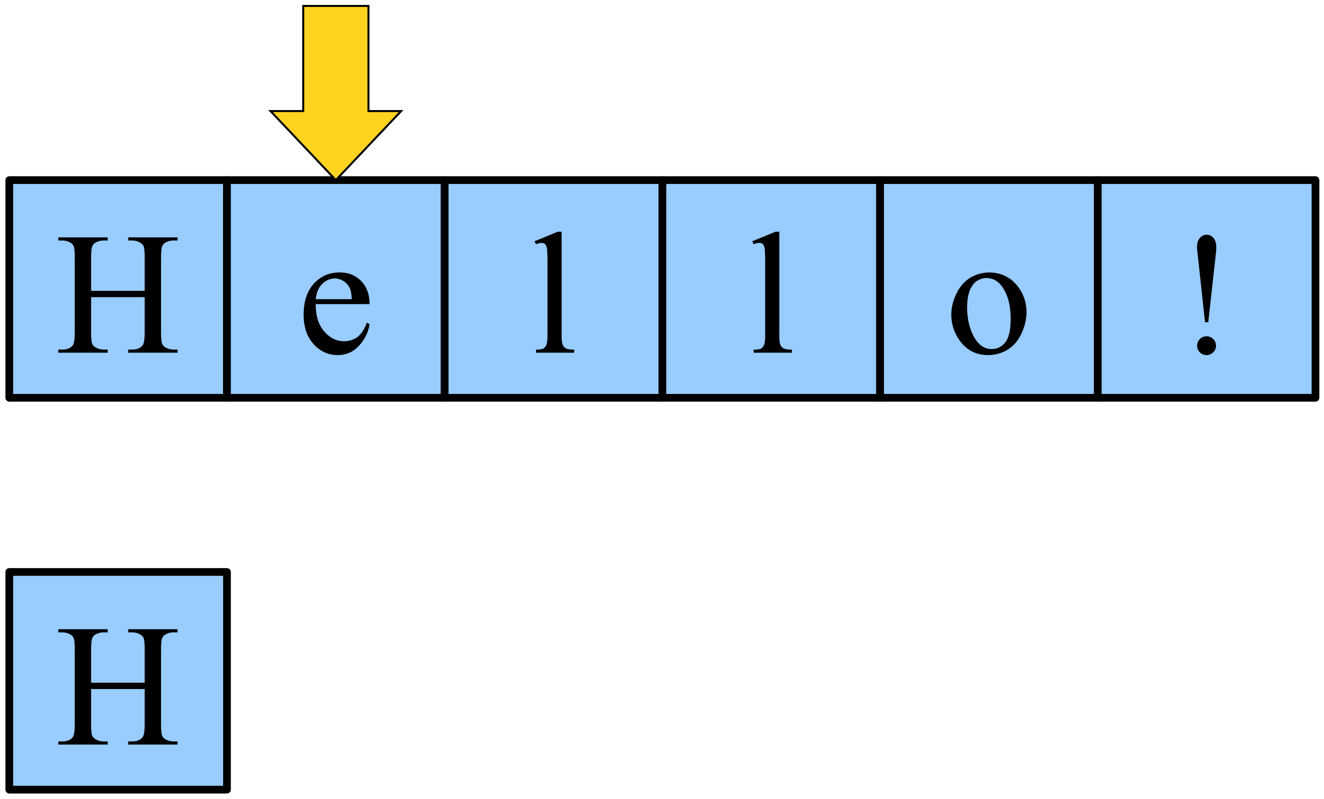
Reversing a String

```
String str = "Hello!";
String newStr = "";
for (int i = str.length() - 1; i >= 0; i--) {
     newStr += str.charAt(i);
}
```

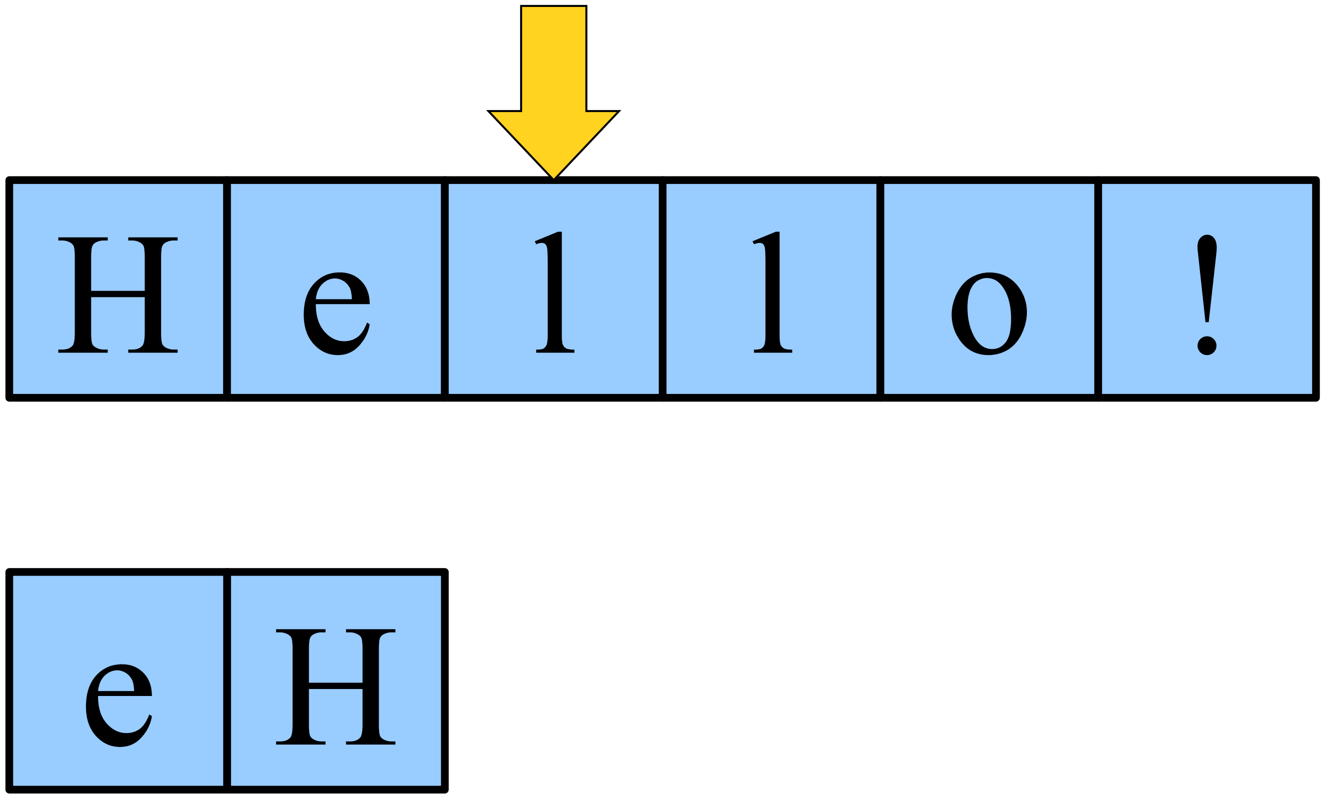
Reversing a String



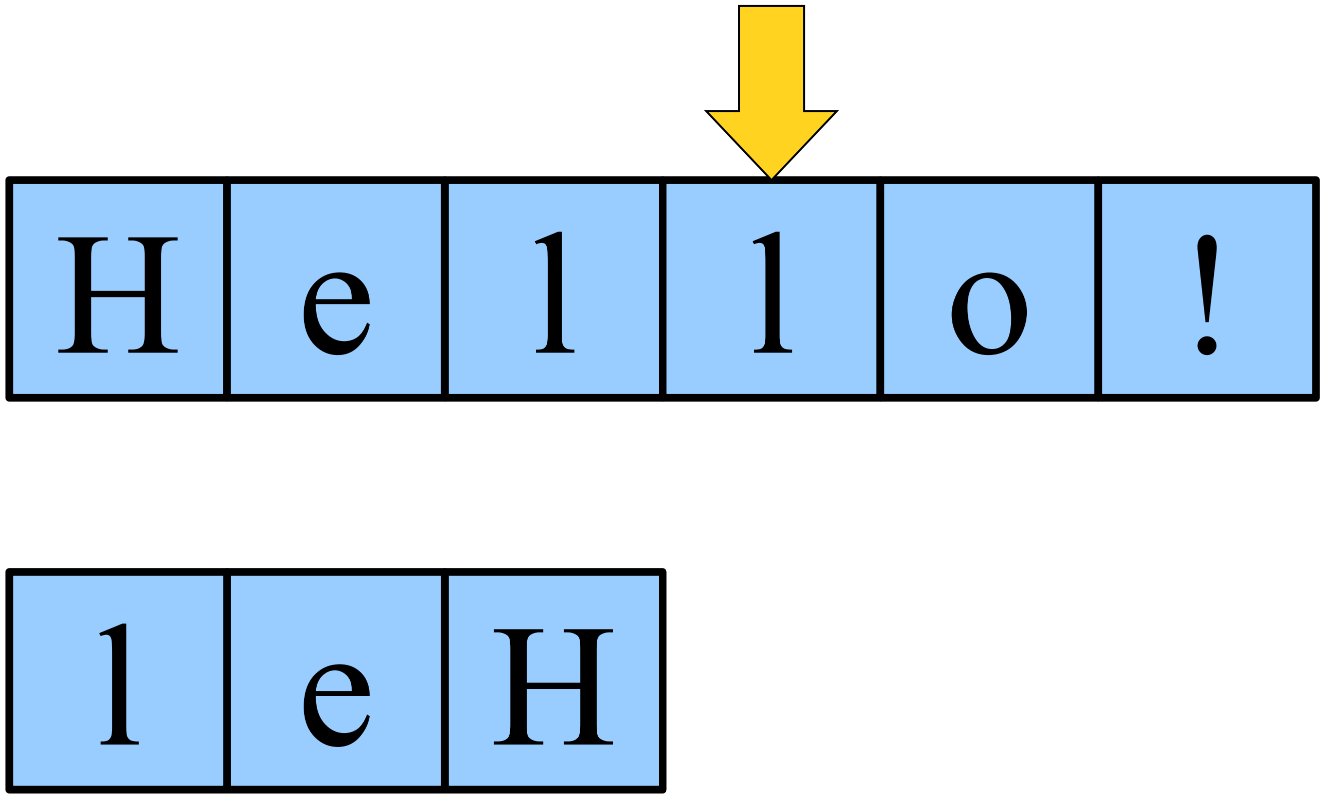
Reversing a String



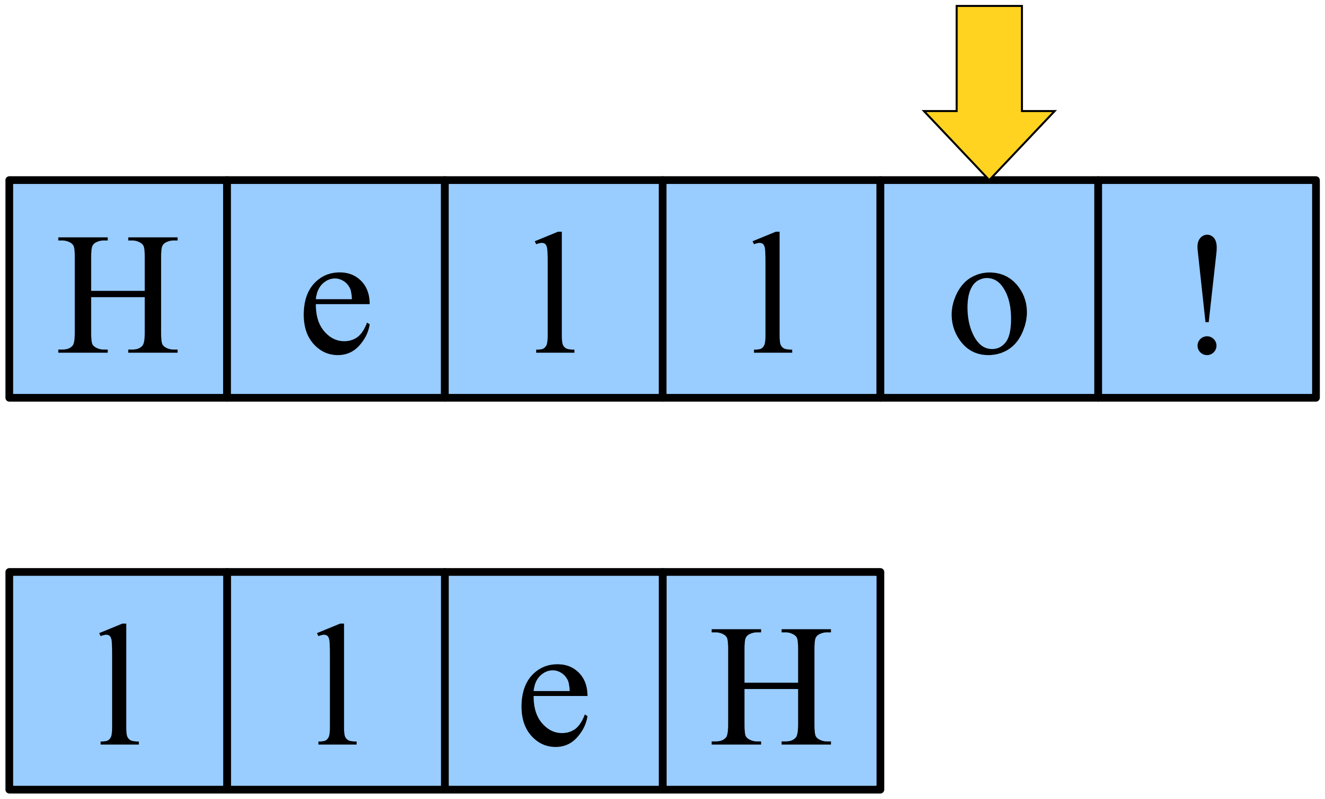
Reversing a String



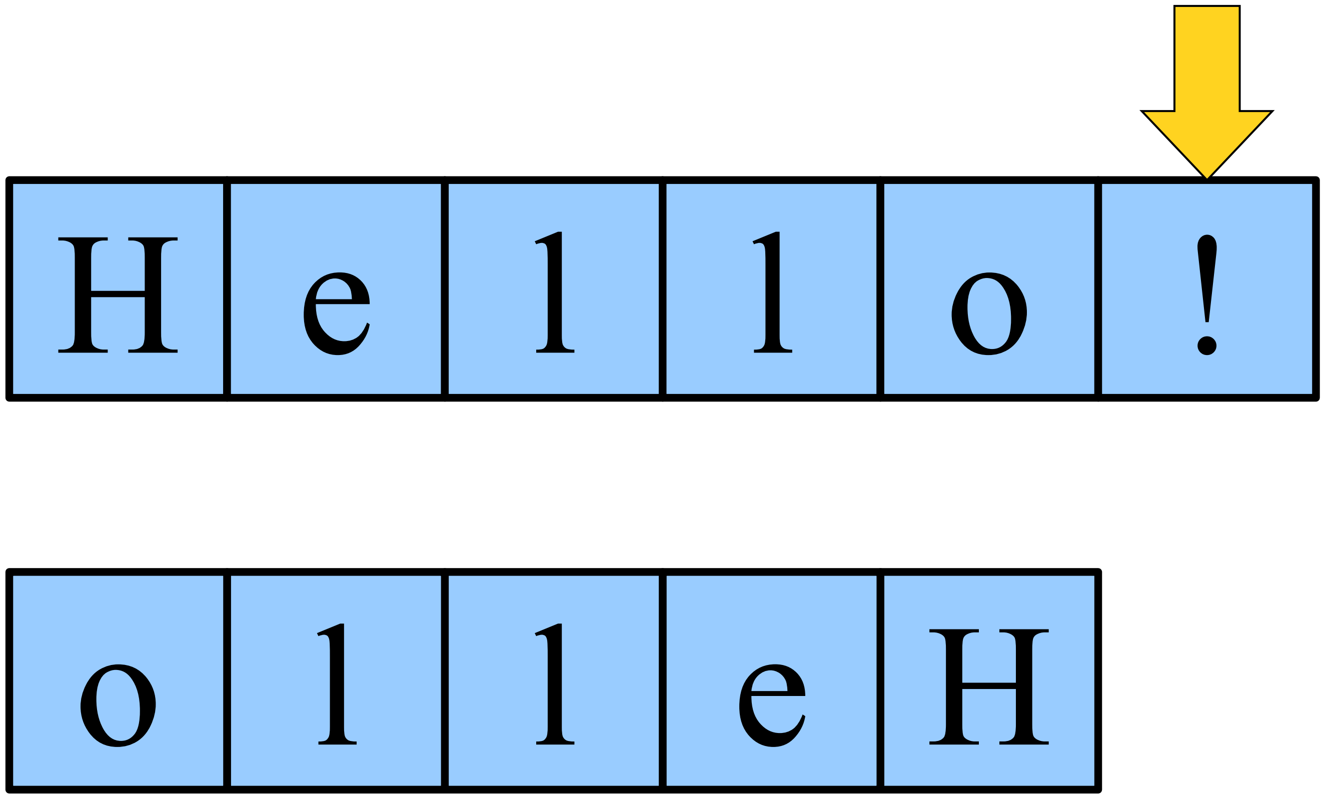
Reversing a String



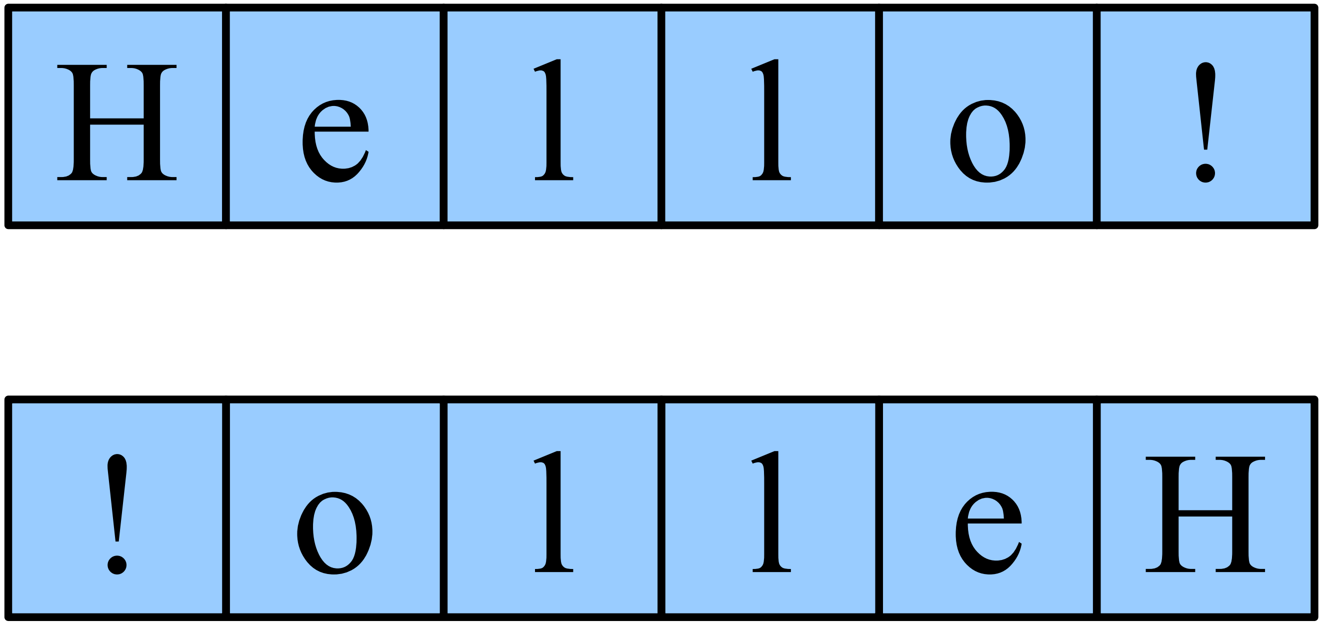
Reversing a String



Reversing a String



Reversing a String

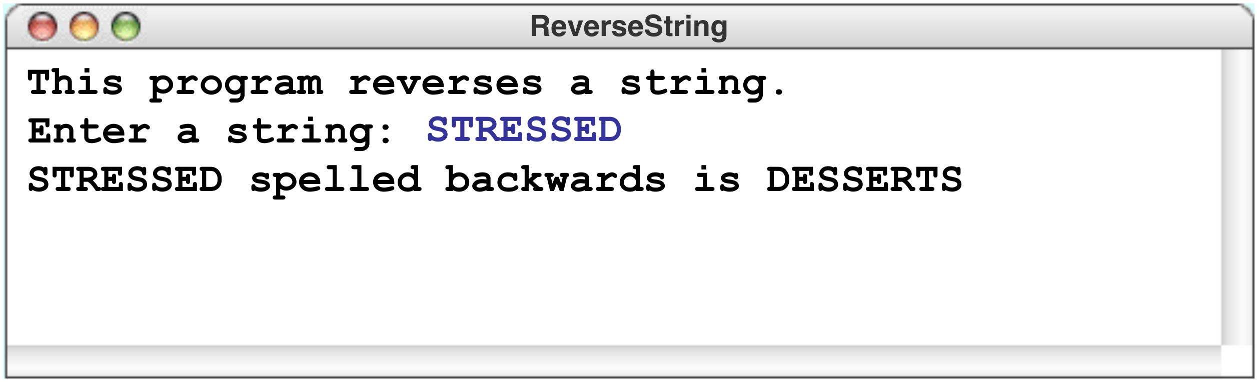


Reversing a String

```
public void run() {
```

```
private String reverseString(String str) {
    String result = "";
    for ( int i = 0; i < str.length(); i++ ) {
        result = str.charAt(i) + result;
    }
    return result;
}
```

```
result        str         i
DESSERTS      STRESSED    8
```



ReverseString

```
This program reverses a string.
Enter a string: STRESSED
STRESSED spelled backwards is DESSERTS
```

Using portions of slides by Eric Roberts

Plan For Today

- Recap: 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).

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

Plan For Today

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

Exercise: Caesar Cipher

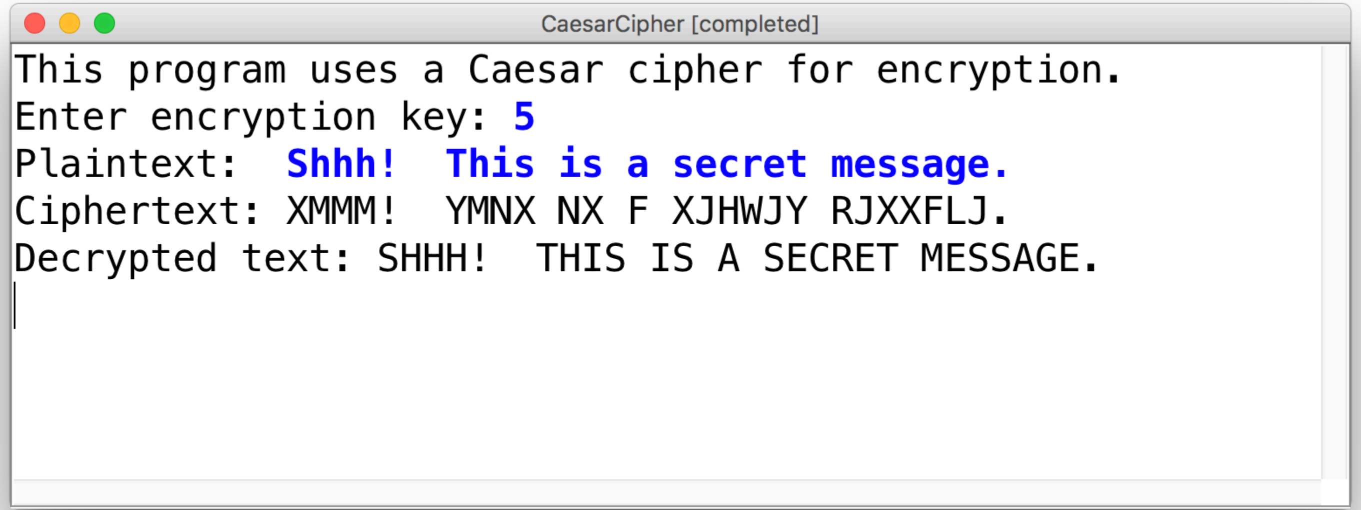
- Rotate alphabet by *n* letters (*n* = 3 in below)
  - *n* is called the **key**
- Wrap-around at the end
- Substitute letters based on this mapping

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 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: 5
Plaintext:  Shhh!  This is a secret message.
Ciphertext: XMMM!  YMNX NX F XJHWJY RJXXFLJ.
Decrypted text: SHHH!  THIS IS A SECRET MESSAGE.
```

Recap

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

Next time: reading text files