

String Applications

Jerry Cain

CS 106AJ

October 26, 2018

slides courtesy of Eric Roberts and Jerry Cain

String Calisthenics

Let's review some **String** methods you learned about last time:

- ✓ `"AEIOUaeiou".length` 10
- ✓ `"ABCDEFGH".charAt(6)` "G"
- ✓ `"Harry Potter".indexOf("a")` 1
- ✓ `"Harry Potter".indexOf("a", 6)` -1
- ✓ `"Harry Potter".lastIndexOf("tt")` 8
- ✓ `"bumfuzzle".substring(3, 7)` "fuzz"
- ✓ `"cabotage".substring(1, 1)` ""
- ✓ `"agelast".substring(3)` "last"

Generating Acronyms

- An *acronym* is a word formed by taking the first letter of each word in a sequence, as in
 - "North American Free Trade Agreement" → "NAFTA"
 - "not in my back yard" → "nimby"
 - "self-contained underwater breathing apparatus" → "scuba"
- The text describes and implements two versions of a function `acronym(str)` that generates an acronym for `str`:
 - The first version searches for spaces in the string and includes the following character in the acronym. This version, however, fails for acronyms like *scuba*, in which some of the words are separated by hyphens rather than spaces.
 - The second version looks at every character and keeps track of whether the algorithm is scanning a word formed composed of sequential letters. This version correctly handles *scuba* as well as strings that have leading, trailing, or multiple spaces.

acronym , Take I

```
acronym("not in my back yard")  
function acronym(str) {  
  let result = str.charAt(0);  
  let sp = str.indexOf(" ");  
  while (sp !== -1) {  
    result += str.charAt(sp + 1);  
    sp = str.indexOf(" ", sp + 1);  
  }  
  return result;  
}
```

str

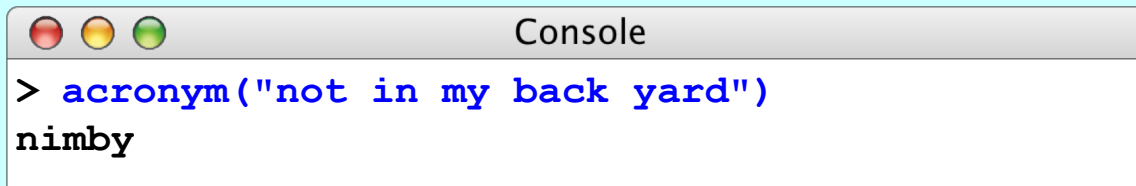
"not in my back yard"

result

"nimby"

sp

-1



```
Console  
> acronym("not in my back yard")  
nimby
```

acronym, Take II

```
acronym("In My Humble Opinion")
```

```
function acronym(str) {
```

```
  const ALPHABET = "ABCDEFGHIJKLMNOPQRSTUVWXYZ";
```

```
  function isLetter(ch) {
```

```
    return ch.length === 1 &&
```

```
    ALPHABET.indexOf(ch.toUpperCase()) !== -1;
```

```
  }
```

-1

"IMHO"

20

true

"n"

```
Console
> acronym("In My Humble Opinion")
IMHO
```

Translating Pig Latin to English

Section 7.4 works through the design and implementation of a program to convert a sentence from English to Pig Latin. In this dialect, the Pig Latin version of a word is formed by applying the following rules:

1. If the word begins with a consonant, the `wordToPigLatin` function moves the initial consonant string to the end of the word and then adds the suffix *ay*, as follows:

scram → *amscray*

2. If the word begins with a vowel, `wordToPigLatin` generates the Pig Latin version simply by adding the suffix *way*, like this:

apple → *appleway*

3. If the word contains no vowels at all, `wordToPigLatin` returns the original word unchanged.

Translating Pig Latin to English

"stout plunder lover"

i		0	1	2	3	4	5	6		12	13	14		18
inWord	F	T	T	T	T	T	F	T		T	F	T		T
start	-1	0	0	0	0	0	-1	6		6	-1	14		14

- **inWord** is **true** if and only if we're in a word, and **start** is the index of the first character of the word we're currently in (or **-1** if we're not in a word).
- **inWord** is now **true** and **start** is set equal to **0**. We assign **start** to be a snapshot of **i** at the same time **inWord** transitions from **false** to **true**, so we can remember where the current word of interest begins.
- This is an interesting transition, since the current word we're in is just now ending. We can isolate the word by calling **str.substring(start, i)**, where **str** is assumed to be the entire sentence or fragment to be translated.
 - Right now, **str.substring(start, i)** produces **"stout"**.
 - And now, **str.substring(start, i)** produces **"plunder"**.

Pseudocode for the Pig Latin Program

```
function toPigLatin(str) {  
    Initialize a variable called result to hold the growing string.  
    for (each character position in str) {  
        if (the current character is a letter) {  
            if (we're not yet scanning a word) Remember the start of this word.  
        } else {  
            if (we were scanning a word) {  
                Call wordToPigLatin to translate the word.  
                Append the translated word to the result variable.  
            }  
            Append the separator character to the result variable.  
        }  
    }  
    if (we're still scanning a word) {  
        Call wordToPigLatin and append the translated word to result.  
    }  
}
```



```
function wordToPigLatin(word) {  
    Find the first vowel in the word.  
    If there are no vowels, return the original word unchanged.  
    If the vowel appears in the first position, return the word concatenated with "way".  
    Divide the string into two parts (head and tail) before the vowel.  
    Return the result of concatenating the tail, the head, and the string "ay".  
}
```


Simulating the PigLatin Program

```
toPigLatin("this is pig latin")
```

```
function toPigLatin(str) {
```

```
function wordToPigLatin(word) {
```

```
let vp = findFirstVowel(word);
```

```
if (vp === -1) {
```

```
return word;
```

```
} else if (vp === 0) {
```

```
return word + "way";
```

```
} else {
```

```
let head = word.substring(0, vp);
```

```
let tail = word.substring(vp);
```

```
return tail + head + "ay";
```

```
}
```

```
}
```

"atinlay"

word

vp

head

tail

"isthay isway igpay atinlay" "

"atin"

```
Console
> toPigLatin("this is pig latin")
isthay isway igpay atinlay
```

The End