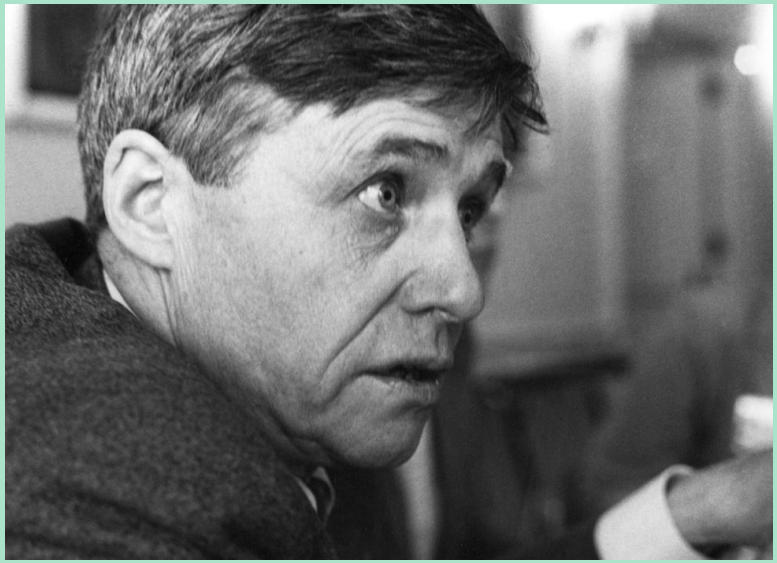
Arrays and Files

Abraham Gebrekidan
EIP 01
Sep 4, 2024

Once upon a time . . .

Ken Iverson and APL

- In the early 1960s, a computer scientist named Ken Iverson invented the APL programming language, which uses arrays as its primary data type. Iverson won the Turing Award in 1979.
- APL required a special keyboard containing Greek letters and a variety of mathematical symbols to support array operations.
- APL's powerful operators made it possible to write enormously complex programs in a cryptic but concise form.



Ken Iverson (1920-2004)

Conway's Life Game in APL

Game of Life [edit]

The following function "life", written in Dyalog APL, takes a boolean matrix and calculates the new generation according to Conway's Game of Life. It demonstrates the power of APL to implement a complex algorithm in very little code, but it is also very hard to follow unless one has advanced knowledge of APL.

```
life←{↑1 ⍵∨.∧3 4=+/,¯1 0 1∘.⊖¯1 0 1∘.⌽⊂⍵}
```

WIKIPEDIA
The Free Encyclopedia

Simple Arrays

Simple Arrays

- An ***array*** is a collection of individual data values in which it is possible to count the values off in order: here is the first, here is the second, and so on.
- The individual values in an array are called ***elements***. The number of elements is called the ***length*** of the array. As with strings, you can determine the length of an array by checking its `length` property.
- Each element is identified by its position number in the array, which is called its ***index***.
- In JavaScript, index numbers always begin with 0 and extend up to one less than the length of the array.

Creating an Array

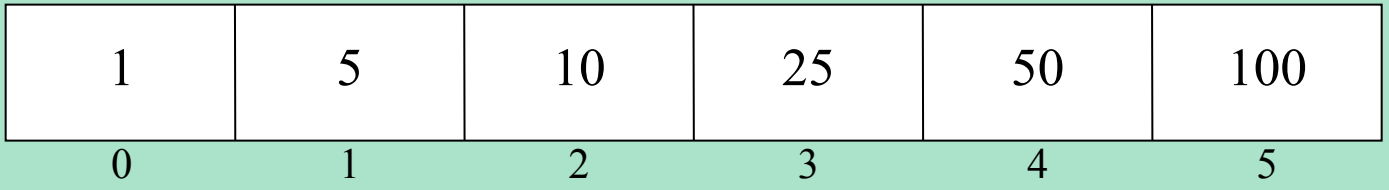
- The simplest way to create an array in JavaScript is to list the elements of the array surrounded by square brackets and separated by commas. For example, the declaration

```
const COIN_VALUES = [ 1, 5, 10, 25, 50, 100 ];
```

  creates a constant array of six elements that correspond to the standard coins available in the United States.

- Arrays are most commonly represented conceptually as a series of numbered boxes, as in the following representation of **`COIN_VALUES`**:

`COIN_VALUES`



Nonnumeric Arrays

- Arrays may contain values of any JavaScript type. For example, the declaration

```
const COIN_NAMES = [
    "penny",
    "nickle",
    "dime",
    "quarter",
    "half-dollar",
    "dollar"
];
```

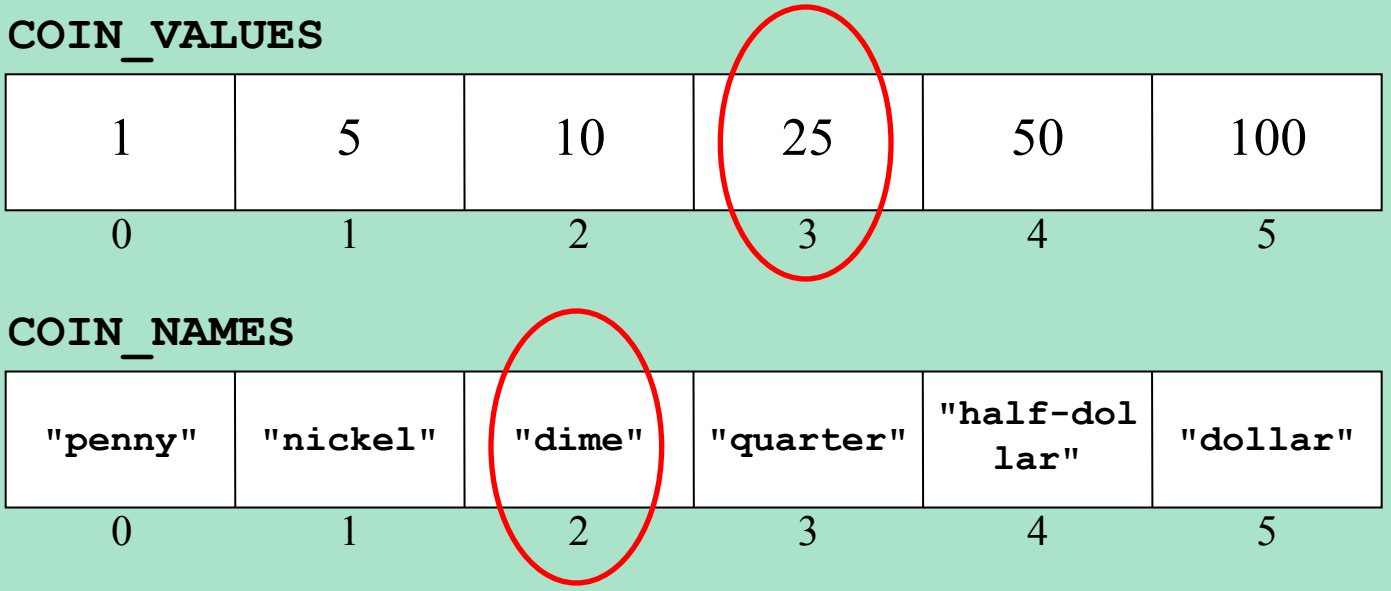
creates the following array:

COIN_NAMES

"penny"	"nickel"	"dime"	"quarter"	"half-dollar"	"dollar"
0	1	2	3	4	5

Array Selection

- Given an array, you can get the value of any element by writing the index of that element in brackets after the array name. This operation is called ***selection***.
- For example, given the declarations on the preceding slides, the value of **`COIN_VALUES[3]`** is 25.
- Similarly, the value of **`COIN_NAMES[2]`** is the string **`"dime"`**.



Cycling through Array Elements

- One of the most useful array idioms is cycling through each of the elements of an array in turn. The standard **`for`** loop pattern for doing so looks like this:

```
for (var i = 0; i < array.length; i++) {
    Operations involving the ith element of the array
}
```

- As an example, the following function computes the sum of the elements in **`array`**:

```
function sumArray(array) {
   var sum = 0;
   for (var i = 0; i < array.length; i++) {
      sum += array[i];
   }
   return sum;
}
```

Exercise: Making Change

- Write a function **`makeChange(change)`** that displays the number of coins of each type necessary to produce **`change`** cents using the values in the constant arrays **`COIN_VALUES`** and **`COIN_NAMES`**.
- In writing your program, you may assume that the currency is designed so that the following strategy always produces the correct result:
  - Start with the last element in the array (in this case, dollars) and give as many of those as are possible.
  - Move on to the previous element and give as many of those as possible, continuing this process until you reach the number of cents.
- Assume that someone has written **`createRegularPlural`**

Adding and Removing Elements

`push(`*element, . . .***`)`** Adds one or more elements to the end of the array.
`pop()` Removes and returns the last element of the array.
`shift()` Removes and returns the first element of the array.
`unshift(`*element, . . .***`)`** Adds one or more elements to the front of the array.
`slice(`*start, finish***`)`** Returns a subarray beginning at *start* and ending just before *finish*.
`splice(`*index, count, . . .***`)`** Removes *count* elements starting at *index,* and optionally adds new ones.

Although each of these methods has its uses, the only one that is essential for the programs in this course is **`push`**.

Growing an Array by Accretion

- The **`push`** method makes it possible to create an array by adding one element at a time. This pattern looks like this:

```
var array = [ ];
for (whatever limits are appropriate to the application) {
   array.push(the new element);
}
```

- As an example, the following function creates an array of **`n`** values, each of which is initialized to **`value`**:

```
function createArray(n, value) {
   var array = [ ];
   for (var i = 0; i < n; i++) {
      array.push(value);
   }
   return sum;
}
```

Passing Arrays as Parameters

- When you pass an array as a parameter to a function or return a function as a result, only the *reference* to the array is actually passed between the functions.
- The effect of JavaScript's strategy for representing arrays internally is that the elements of an array are effectively shared between the caller and callee. If a function changes an element of an array passed as a parameter, that change will persist after the function returns.
- The next slide simulates a program that does the following:
    1. Generates an array containing the integers 0 to $N-1$.
    2. Prints out the elements in the array.
    3. Reverses the elements in the array.
    4. Prints out the reversed array on the console.

The `reverseArray` Function

```
function TestReverseArray() {
   function reverseArray(array) {
      for ( var lh = 0 ; lh < array.length / 2 ; lh++ ) {
         var rh = array.length - lh - 1;
         var tmp = array[lh];
         array[lh] = array[rh];
         array[rh] = tmp;
      }
   }
```

array	lh	rh	tmp

Console

```
Forward: [0, 1, 2, 3, 4, 5, 6, 7, 8, 9]
Reverse: [9, 8, 7, 6, 5, 4, 3, 2, 1, 0]
```

Other Array Methods

`concat(`*array, . . .***`)`** Concatenates one or more arrays onto the receiver array.
`indexOf(`*element***`)`** Returns the first index at which *element* appears, or –1 if not found.
`lastIndexOf(`*element***`)`** Returns the last index at which *element* appears, or –1 if not found.
`reverse()` Reverses the elements of the array.
`sort()` Sorts the elements of the array in ascending order.

Reading Data from Files

- In practice, applications often need to work with arrays that are too large to enter by hand. In many cases, it is easier to read the values of a list from a data file.
- A ***file*** is the generic name for any named collection of data maintained on the various types of permanent storage media attached to a computer. In most cases, a file is stored on a hard disk, but it can also be stored on a removable medium, such as a CD or flash memory drive.
- Files can contain information of many different types. The most common type of file, however, is a ***text file***, which contains character data of the sort you find in a string.

Text Files *vs.* Strings

Although text files and strings both contain character data, it is important to keep in mind the following important differences between text files and strings:

1. ***The information stored in a file is permanent.*** The value of a string variable persists only as long as the variable does. Local variables disappear when the function returns, and instance variables disappear when the object goes away, which typically does not occur until the program exits. Information stored in a file exists until the file is deleted.

2. ***Files are usually read sequentially.*** When you read data from a file, you usually start at the beginning and read the characters in order, either individually or in groups that are most commonly individual lines. Once you have read one set of characters, you then move on to the next set of characters until you reach the end of the file.

Reading Text Files in JavaScript

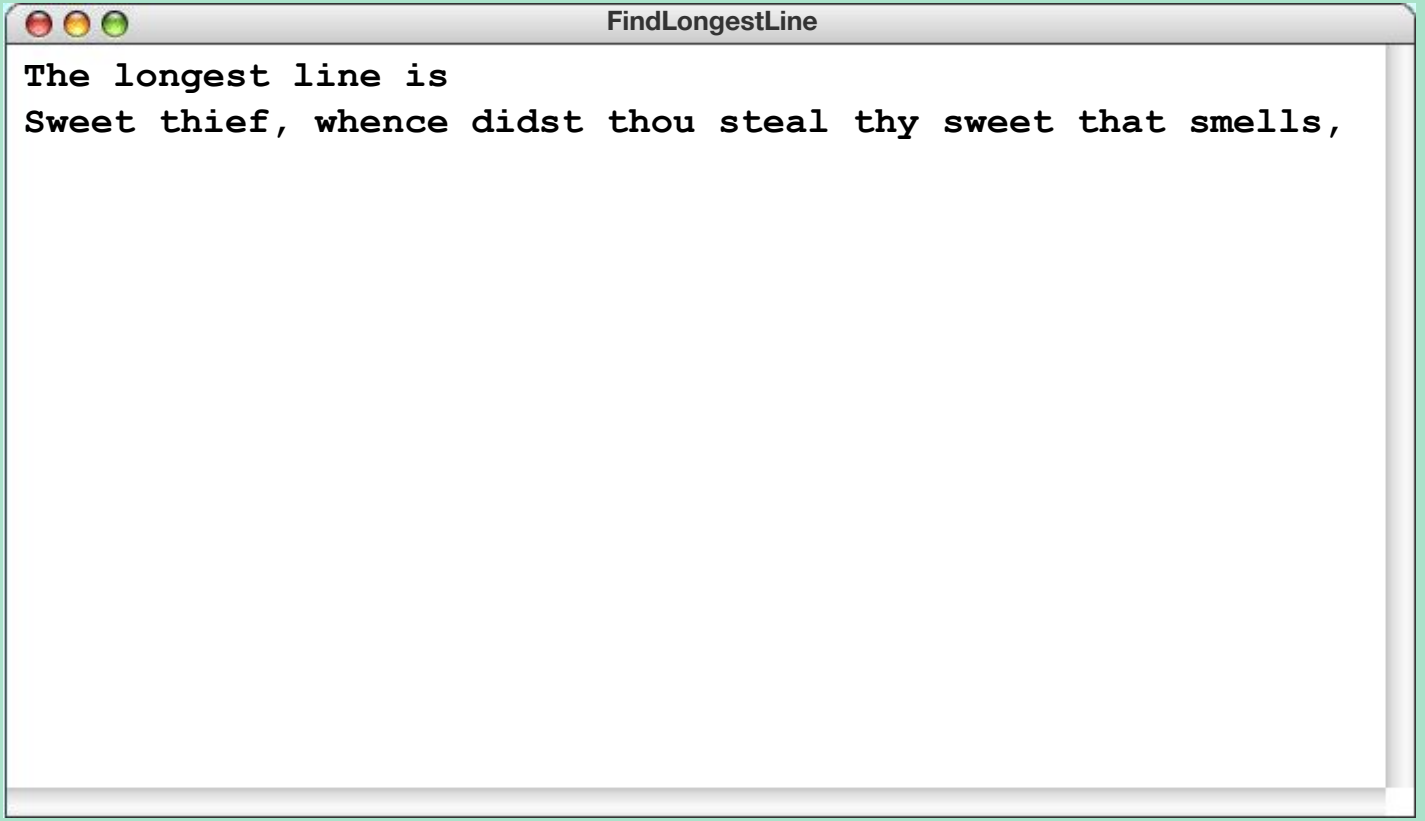
- Because JavaScript is designed for use in web pages that are not permitted to read files on the user's computer, JavaScript has no built-in facilities for reading files.
- To get around this restriction, SJS defines a **`"file"`** library that makes it possible to read files from the SJS application. The **`File`** class exports the following class methods:

`File.read``(`*filename*`)` Reads the entire file as a string.
`File.readLines``(`*filename*`)` Reads the entire file as an array of lines.
`File.chooseFile``(`*callback*`)` Lets the user select a file and then calls *callback* with the file name.

- The HangKarel starter code uses **`File.readLines`** to read the lexicon file.

Exercise: Find Longest Line

- Write a program that uses a file chooser to select an input file and that then finds and prints the longest line in the file.



The End