香港中文大學
The Chinese University of Hong Kong

JS FUNCTIONS AND ARRAYS

CSCI2720 2022-23 Term 1

Building Web Applications

Dr. Chuck-jee Chau
chuckjee@cse.cuhk.edu.hk

OUTLINE

- More on functions
- Function parameters
- Arrow functions
- Invoking functions
- Nested functions
- Callback functions
- Generator functions
- Object methods
- More on arrays
- Creating arrays
- Destructuring arrays
- Modifying arrays
- Iterating arrays
- Searching in arrays
- Transforming arrays

FUNCTIONS

JS FUNCTIONS

- A JS function has the **`function`** keyword, an *optional* function name, *optional* function parameters, and an *optional* return
  - ***Parameters***: the list of input names in function definition
  - ***Arguments***: the actual values being passed at function call

```
function func(para1, para2, …) {
    // function body
    // optional return statement
}
```

FUNCTION PARAMETERS

- ***Default values*** for parameters can be supplied, and ***missing arguments*** will be given the undefined value

```
function f1(x, y=2, z) {
  console.log("x = " + x);
  console.log("y = " + y);
  console.log("z = " + z);
}
```

https://codepen.io/chuckjee/pen/MWObJaE

```
f1(5);  // 2nd and 3rd arguments missing
"x = 5"
"y = 2"
"z = undefined"
```

- The function arguments can also be found in an **`arguments`** object without a parameter list
  - *Note: this is not for arrow functions*

```
function f2() {
  for (i of arguments) {
    console.log(i);
  }
}
```

https://codepen.io/chuckjee/pen/MWObJaE

```
f2(1,2,3); // but f2() has no parameters
"1"
"2"
"3"
```

FUNCTION PARAMETERS

- A new way to obtain an unknown number of arguments: ***rest operator . . .***
  - The rest parameters must be ***the last item*** in the parameter list

```
function f3(x, y, ...more) {
  console.log("x is " + x);
  console.log("y is " + y);
  console.log(more);
  console.log(typeof more);
}
```

https://codepen.io/chuckjee/pen/QWOGdKv

```
f3(2,4,6,8,10);
"x is 2"
"y is 4"
// [object Array] (3)
[6,8,10]
"object"
```

- This can also be used in arrow function syntax

```
let f4 = (a, ...b) =>
console.log(b);
```

https://codepen.io/chuckjee/pen/QWOGdKv

```
f4(1, 3, 5);
// [object Array] (2)
[3,5]
```

- *See: https://dev.to/sagar/three-dots---in-javascript-26ci*

FUNCTION DECLARATION VS. EXPRESSION

- In JavaScript, *function* codes are stored as plain values
  - Function declarations are ***hoisted to the top*** of the scope, *i.e., used before being declared!*
  - Function expressions have the scope defined by the assigned variable on the left

https://codepen.io/chuckjee/pen/podEGXw

```
// function declaration
function f1(text) {
  console.log("This is the f1 input: " + text);
}

// function expression with anonymous function
let f2 = function (text) {
  console.log("This is the f2 input: " + text);
}

// arrow (anonymous) function in expression
let f3 = text => console.log("This is the f3 input: " + text);
```

```
console.log(f1);          // shows f1() code
function f1(text) {
 console.log("This is the f1 input: " + text);
}
console.log(typeof f1);
"function"
f1("a");                  // executes f1()
"This is the f1 input: a"
```

ARROW FUNCTIONS

```
(para1, para2, …) => { statements; }
```

- Brackets `()` for parameter list can be omitted for single parameter
- Single-line: braces `{}` and `return` can be omitted, e.g.,

```
let square = num => num*num; // square(10) is 100
```

- Multi-line: braces `{}` and `return` must be present, like regular functions
- Although similar, arrow functions are ***not the same*** as regular functions
  - No `this`, no `arguments`
  - Not suitable as methods, constructors
- *See: https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Functions/Arrow_functions*

INVOKING FUNCTIONS

- Functions are invoked (executed) using the parentheses **( )** after the function/variable name
  - Without the parentheses, the function code is returned
- Self-invoking anonymous function/ Immediately-invoked function expression (IIFE)
  - Avoiding pollution of the global object
  - Isolating variable declarations

```
(function() {
  console.log("Hello there");
})();
(() => console.log("Hello again"))();
```

INVOKING FUNCTIONS

- Common mistake for DOM events
  - A function should be used for events to invoke later!

https://codepen.io/chuckjee/pen/zYPKXqY

```
<button id="btn1">Button 1</button>
<button id="btn2">Button 2</button>
<button onclick="alert('JS in HTML')">Button 3</button> <!– another method -->

<script>
// without function, the statement is executed when loaded
document.querySelector("#btn1").onclick = alert("Wrong alert: too early");
// function code is executed only at onclick event
// either arrow function or regular function is okay
document.querySelector("#btn2").onclick = () => alert("Correct onclick alert");
</script>
```

NESTED FUNCTIONS

- Functions in JS can be nested
  - Separation of variables in different scopes
    - Inner function can access variables of outer functions, but not vice versa
  - Multiple parentheses to invoke functions with function arguments

https://codepen.io/chuckjee/pen/WNXGWRa

```
function f1(a) {
  function f2(b) {
    return a+b;
  }
  return f2;
}
console.log(f1(10));    // code of f2 is returned
console.log(f1(10)(5)); // results of f2(5) is returned
```

CALLBACK FUNCTIONS

- As functions are simply values, they can also be passed as a function argument, and these are ***callbacks***
  - More often used in *asynchronous* JS, where callbacks are called only after some waiting time or events

https://codepen.io/chuckjee/pen/YzEGMar

```
function f0(callback1, callback2) {
  let x = prompt("A number?");
  if (x%2)
    callback1();
  else
    callback2();
}
function f1() { alert("Odd"); }
function f2() { alert("Even"); }
```

```
// Functions can be written right inside
// function calls (modern syntax)

f0(
   ()=>alert("Odd"),
   ()=>alert("Even")
);
```

GENERATOR FUNCTIONS

- Generator functions can return a value and be re-entered later
  - Special keywords **`function*`** and **`yield`**
  - Re-entrance after the previous **`yield`** statement

https://codepen.io/chuckjee/pen/abVmrbW

```
<button onclick="alert(count.next().value)">Click</button>

<script>
function* g(inc) {
  let x = 0;
  while (1)
    yield x += inc;
}
let count = g(2);
</script>
```

- *See: https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Statements/function**

OBJECT METHODS

- Objects can contain functions, and they are called ***object methods***

https://codepen.io/chuckjee/pen/XWzNpRB

```
let human = {
  keyword: "Hello!",
  shout: function() { alert(this.keyword) }
}
let human2 = {
  keyword: "Hello again!",
  shout() { alert(this.keyword) } // alternative shorter syntax for methods
}
human.shout();
human2.shout();
```

- *See: https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Functions/Method_definitions*

ARRAYS

JS ARRAYS

- A JS array is an ordered collection, e.g., `["Hello", "World"]`
  - Data type is not limited, and can be functions, objects, and/or arrays
  - It is a special kind of object, i.e., it can be assessed like an object
  - Optimized with contiguous memory storage
  - Arrays are copied by reference, like objects
    - *See: https://javascript.info/object-copy*
- To verify if a variable/expression is an array, use **`Array.isArray()`**

https://codepen.io/chuckjee/pen/mdqOQLe

```
let x = [1,2,3];
console.log(Array.isArray(x));
// true

let y = x;
console.log(y); // [1,2,3]
y[1] = 0;
console.log(x); // [1,0,3]
console.log(y); // [1,0,3]
```

CREATING ARRAYS

- From an "array-like object": with a *length* and *indexed* elements
  - *See: https://javascript.info/iterable*

- By combining other arrays
  - The ***spread operator ...***
  - This can be done the same way for objects
  - Shallow cloning of arrays/ objects is easy as `x=[...y]`

https://codepen.io/chuckjee/pen/WNXoYaE

```
let s = "Hello World";
let array = Array.from(s);
console.log(array);
// ["H","e","l","l","o"," ","W","o","r","l","d"]
console.log(s.length);              // 11
console.log(array.length);          // 11

let a = [1,3,5];
let b = [2,4,6];
let c = [...a, 0, ...b];
console.log(c);              // [1,3,5,0,2,4,6]
```

DESTRUCTURING ARRAYS

- An array can be destructured into ***separate variables***
- This makes it possible for a function to return multiple values
- The rest operator is also supported

```
let a, b, rest;
[a, b] = [10, 20];

console.log(a);
// 10

console.log(b);
// 20

[a, b, ...rest] = [10, 20, 30, 40, 50];

console.log(rest);
// [30,40,50]
```

- *See: https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Operators/Destructuring_assignment*

MODIFYING ARRAYS

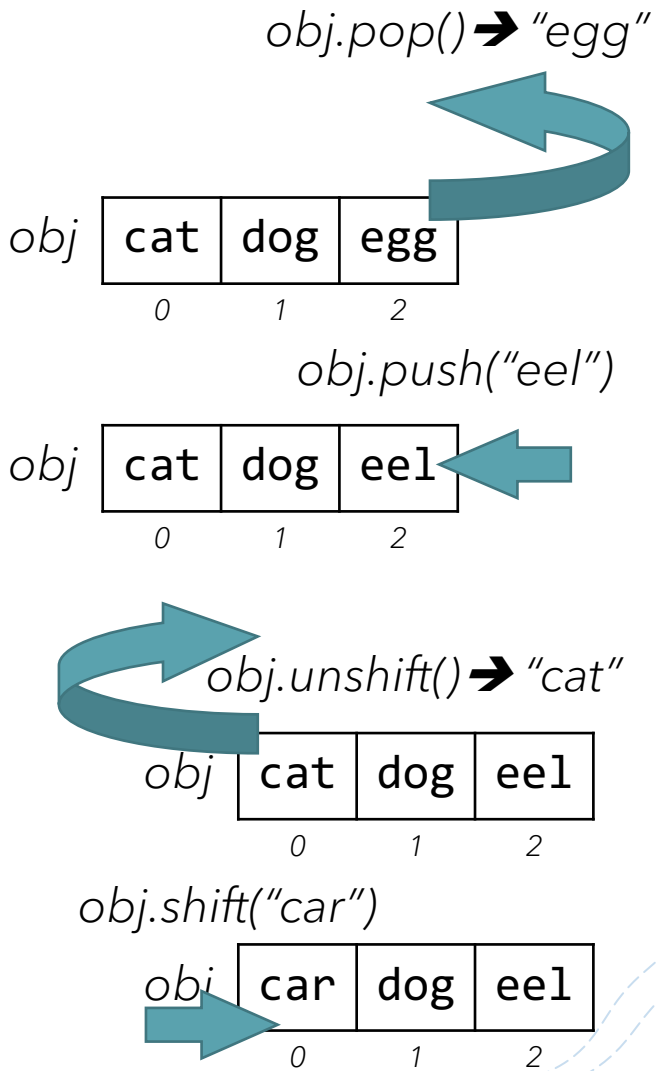
- *array*.**slice**(*start*, [*end*])
  - Returns the new sliced array from ***start*** inclusive to ***end*** exclusive
  - Not modifying the original array
- *array*.**splice**(*start*, [*deleteCount*, [*itemsToAdd*…])
  - Changes original, and returns an array with deleted elements: ***deleteCount*** from ***start*** index
  - Items are added to the original array from ***start*** index
- Negative indices are accepted for ***start*** or ***end***
  - -1 denotes last item, -2 second last, and so on

```
let c = ["cyan","magenta","yellow","black"];
let c1 = c.slice(1,2);
console.log(c);
// ["cyan","magenta","yellow","black"]
console.log(c1);
// ["magenta"]
let c2 = c.splice(2,1,"red","green","blue");
console.log(c);
// ["cyan","magenta","red","green","blue","black"]
console.log(c2);
// ["yellow"]
```

https://codepen.io/chuckjee/pen/PoObXPj

MODIFYING ARRAYS

- Array as a stack (Last-in-first-out LIFO)
  - ***array*.pop()** removes the last element and returns it
  - ***array*.push(*items*)** adds items to the end of array
- Array as a queue (First-in-first-out FIFO)
  - ***array*.shift()** removes the first element and returns it
  - ***array*.unshift(*items*)** adds items to the start of array
  - **push()** can be used to add items to the end of queue
- Original array is always modified
- Stack processes are faster since the array index is not affected
  - *See: https://javascript.info/array#performance*



ITERATING ARRAYS

- The traditional **`for`** loop allows flexible changes, and is ***fastest***
- The **`for…of`** loop is handy for obtaining only a copy of the array elements (e.g., for displaying)
- The **`forEach`** loop takes a function as input with different levels of flexibility
  - Callback functions can also be used

```
let a = [1,3,5];
for (let i=0; i<a.length; i++) {
  console.log(a[i]);
  a[i] = a[i]+1;
}
console.log(a); // [2,4,6]

let b = [1,3,5];
for (let item of b) {
  console.log(item);
  item = item + 1;
}
console.log(b); // [1,3,5] not modified

let c = [1,3,5];
c.forEach(item=>item+1);
console.log(c); // [1,3,5] not modified

let d = [1,3,5];
d.forEach((item,i,d) => d[i]+=1);
console.log(d); // [2,4,6]
```

SEARCHING IN ARRAYS

- *array*.indexOf(*item, start*) / *array*.lastIndexOf(*item, start*)
  - Return the index if found (with === comparison) from start, or -1 if not found
- *array*.includes(*value*)
  - True if the array has the value
- *array*.find(*function*(*item,index,array*))
  - The way to match can be ***defined in the function***
  - The first item returning *true* in function will be returned
- *array*.filter(*function*(*item,index,array*))
  - An array of matching items will be returned

```
let num = [10, 12, 13, 15, 20];
console.log( num.find(n => n%5) )
// 12
console.log( num.filter(n => n%5) )
// [12,13]
```

TRANSFORM ARRAYS

- *array*`.reverse()`
  - Reversing order of elements in array
- *array*`.split()` / *array*`.join()`
  - Converting a string to character array, or vice versa
- *array*`.map(` *function(item,index,array)* `)`
  - A new array is returned with the transformation ***defined in function***

```
let num = [10, 12, 13, 15, 20];
console.log( num.map(n => n*2) )
// [20,24,26,30,40]
```

- `sort(` *[function(a,b)]* `)`
  - Without the function, default sorting is comparing as strings (e.g., 2 > 1000)
  - The function can decide how comparison should be done by returning greater/smaller numbers

READ FURTHER…

MDN: Functions

https://developer.mozilla.org/en-US/docs/Web/JavaScript/Guide/Functions

JavaScript.info Array Methods

https://javascript.info/array-methods