JavaScript for ABAP Programmers

Functions as 1st Class Citizens

Chris Whealy / The RIG

ABAP	JavaScript
Strongly typed	Weakly typed
Syntax similar to COBOL	Syntax derived from Java
Block Scope	Lexical Scope
No equivalent concept	Functions are 1st class citizens
OO using class based inheritance	OO using referential inheritance
Imperative programming	Imperative or Functional programming

1st Class Functions

Any programming language in which functions are treated as “1st class citizens” is said to implement “1st class functions”.

So, what does that mean – functions have voting rights?

1st Class Functions

Any programming language in which functions are treated as “1st class citizens” is said to implement “1st class functions”.

So, what does that mean – functions have voting rights?

No, a 1st class citizen is any object that can be:

- Created or destroyed dynamically

1st Class Functions

Any programming language in which functions are treated as “1st class citizens” is said to implement “1st class functions”.

So, what does that mean – functions have voting rights?

No, a 1st class citizen is any object that can be:

- Created or destroyed dynamically
- Treated as data, meaning that it can be both:
  - Passed as an argument to a function and
  - Returned as the result of running a function

1st Class Functions

Any programming language in which functions are treated as “1st class citizens” is said to implement “1st class functions”.

So, what does that mean – functions have voting rights?

No, a 1st class citizen is any object that can be:

- Created or destroyed dynamically
- Treated as data, meaning that it can be both:
  - Passed as an argument to a function and
  - Returned as the result of running a function

In JavaScript, all the above statements are applicable to functions because a function is simply an object “with an executable part”. This means a JavaScript function can be treated either as:

- An object like any other data object, or as
- A executable unit of code

1st Class Functions: Dynamic Creation

A JavaScript function can be created dynamically based on the data available at runtime.

```
// List of animals and the noises they make
var animalNoises = ["dog","woof","cat","meow","pig","oink","cow","moo"];

// A function that returns a text string describing an animal noise
function noiseMaker(name) {
  var n     = animalNoises.indexOf(name);
  var noise = (n == -1) ? "a sound I don't know" : animalNoises[n + 1];
  return "A " + name + " says " + noise;
}
```

1st Class Functions: Dynamic Creation

A JavaScript function can be created dynamically based on the data available at runtime.

```
// List of animals and the noises they make
var animalNoises = ["dog","woof","cat","meow","pig","oink","cow","moo"];

// A function that returns a text string describing an animal noise
function noiseMaker(name) {
  var n     = animalNoises.indexOf(name);
  var noise = (n == -1) ? "a sound I don't know" : animalNoises[n + 1];
  return "A " + name + " says " + noise;
}

// Dynamically create some function objects specific to certain animals
var pigSays   = new Function("alert(\"" + noiseMaker("pig") + "\");");
var sheepSays = new Function("alert(\"" + noiseMaker("sheep") + "\");");
```

1st Class Functions: Dynamic Creation

A JavaScript function can be created dynamically based on the data available at runtime.

```
// List of animals and the noises they make
var animalNoises =                                  ,"pig","oink","cow","moo"];

// A function that                                  ribing an animal noise
function noiseMake
  var n     = anim
  var noise = (n =                                  ow" : animalNoises[n + 1];
  return "A " + na
}

// Dynamically create some function objects specific to certain animals
var pigSays   = new Function("alert(\"" + noiseMaker("pig") + "\");");
var sheepSays = new Function("alert(\"" + noiseMaker("sheep") + "\");");

// Call these dynamically created function objects using the invocation operator
pigSays();
```

1st Class Functions: Dynamic Creation

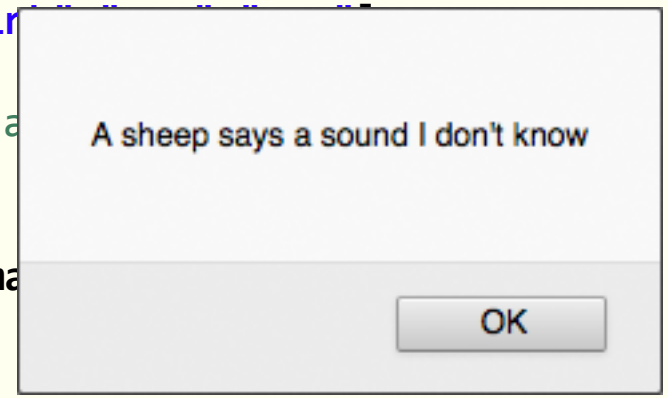
A JavaScript function can be created dynamically based on the data available at runtime.

```
// List of animals and the noises they make
var animalNoises = ["dog","woof","cat","meow","pig","oin

// A function that returns a text string describing an a
function noiseMaker(name) {
  var n     = animalNoises.indexOf(name);
  var noise = (n == -1) ? "a sound I don't know" : anima
  return "A " + name + " says " + noise;
}

// Dynamically create some function objects specific to certain animals
var pigSays   = new Function("alert(\"" + noiseMaker("pig") + "\");");
var sheepSays = new Function("alert(\"" + noiseMaker("sheep") + "\");");

// Call these dynamically created function objects using the invocation operator
pigSays();
sheepSays();
```



1st Class Functions: Dynamic Destruction

Setting a function object to **`null`** destroys that object.

```
// Destroy the dynamically created function objects
pigSays   = null;
sheepSays = null;
```

Notice here that reference is made to the function object itself, not the result of invoking that function. In other words, the invocation operator ( ) must ***not*** be used.

1st Class Functions: Functions as Arguments

Since JavaScript treats a function is an object, the functions created in the previous example can be passed as parameters in the same way you would pass an object containing only data.

```
// A function that will execute any number of functions it is passed
function noisyFarmyard() {
  // Did we receive any parameters?
  if (arguments.length > 0) {
    // Loop around those parameters checking to see if any of them are of type ‘function’
    for (var i=0; i<arguments.length; i++) {
      if (typeof arguments[i] === 'function') {
        arguments[i]();
      }
    }
  }
}

noisyFarmyard(pigSays, sheepSays);
```

Function objects can be passed as arguments just like any other object

1st Class Functions: Functions as Arguments

Since JavaScript treats a function is an object, the functions created in the previous example can be passed as parameters in the same way you would pass an object containing only data.

```
// A function that will execute any number of functions it is passed
function noisyFarmyard() {
  // Did we receive any parameters?
  if (arguments.length > 0) {
    // Loop around those parameters checking to see if any of them are of type 'function'
    for (var i=0; i<arguments.length; i++) {
      if (typeof arguments[i] === 'function') {
        arguments[i]();
      }
    }
  }
}

noisyFarmyard(pigSays, sheepSays);
```

Because we test the data type of each argument, we can determine whether the value we've been passed is executable or not

1st Class Functions: Functions as Return Values 1/2

Having a function that returns a function is powerful tool for abstraction. Instead of a function returning the required value, it returns a function that must be executed to obtain the required value.

```
// A function that works out how many animal functions have been created
function animalList() {
  var listOfAnimals = [];

  // Check whether a function has been created for each animal
  for (var i=0; i<animalNoises.length; i=i+2) {
    var fName = animalNoises[i]+"Says";

    if (typeof window[fName] === 'function') {
      listOfAnimals.push(fName);
    }
  }

  return function() {
    return listOfAnimals;
  }
}
```

Using the array syntax for accessing an object property, we can check whether the required function not only exists within the Global Object, but is also of type 'function'

1st Class Functions: Functions as Return Values 1/2

Having a function that returns a function is powerful tool for abstraction. Instead of a function returning the required value, it returns a function that must be executed to obtain the required value.

```
// A function that works out how many animal functions have been created
function animalList() {
  var listOfAnimals = [];

  // Check whether a function has been created for each animal
  for (var i=0; i<animalNoises.length; i=i+2) {
    var fName = animalNoises[i]+"Says";

    if (typeof window[fName] === 'function') {
      listOfAnimals.push(fName);
    }
  }

  return function() {
    return listOfAnimals;
  }
}
```

Here, we return a function which when invoked, will return the array called `listOfAnimals`

1st Class Functions: Functions as Return Values 2/2

Once this function is called, it doesn't return the data we require; instead, it returns a function that when called, will return the required data.

```
// A function that works out how many animal functions have been created
function animalList() {
  ... snip ...
}

// The call to function animalList() returns a function object.  So 'animalListFunction' is
// not the data we want, but a function that when executed, will generate the data we want.
var animalListFunction = animalList();
```

1st Class Functions: Functions as Return Values 2/2

Once this function is called, it doesn't return the data we require; instead, it returns a function that when called, will return the required data.

```
// A function that works out how many animal functions have been created
function animalList() {
  ... snip ...
}

// The call to function animalList() returns a function object.  So 'animalListFunction' is
// not the data we want, but a function that when executed, will generate the data we want.
var animalListFunction = animalList();

// 'animalListFunction' must now be executed using the invocation operator.
animalListFunction();  // → ["pigSays","sheepSays"]
```