# UNIT 2

## ACTION SCRIPT 1&2

ActionScript 1 and 2 are scripting languages primarily used for creating interactive and animated content in Adobe Flash and Adobe Flex. These languages were popular in the early 2000s and have largely been replaced by ActionScript 3 and other technologies. Nonetheless, I can provide a brief overview of ActionScript 1 and 2:

### ActionScript 1 (AS1):

ActionScript 1 was the original scripting language for Flash, introduced with Flash 4.

It was a simple and procedural language.

It used a timeline-based approach, where scripts were often placed directly on the timeline.

ActionScript 1 was based on ECMAScript, the same standard that JavaScript is based on.

It lacked many of the features found in modern programming languages and was limited in terms of object-oriented programming (OOP) capabilities.

### ActionScript 2 (AS2):

ActionScript 2 was introduced with Flash MX 2004 and improved upon the limitations of AS1.

It introduced better support for OOP concepts, such as classes and inheritance.

AS2 still used a timeline-based approach, but it also allowed the creation of external .as files for organizing code.

It had a more comprehensive set of built-in classes and functions.

AS2 included features like event handling and XML support.

Both AS1 and AS2 were popular in their time for creating interactive web animations and simple games. However, they had several shortcomings, including performance issues and security vulnerabilities. As a result, Adobe phased out support for ActionScript 1 and 2 in favor of ActionScript 3, which is a more robust, strongly typed, and object-oriented language.

If you're working on a project that involves ActionScript, I would recommend using ActionScript 3 or considering other modern web technologies, as AS1 and AS2 are considered outdated and not suitable for most contemporary web development needs.

# ACTION SCRIPT FEATURES

ActionScript 1 was the original scripting language used in Adobe Flash, introduced with Flash 4. While it lacks many of the features and capabilities of later versions of ActionScript, it played a crucial role in creating interactive and animated content on the web during its time. Here are some of the key features of ActionScript 1:

**Procedural Language:** ActionScript 1 is primarily a procedural programming language. It lacks the strong object-oriented features and support found in later versions like ActionScript 3 (AS3).

**Timeline-Based:** Flash movies were built on a timeline, and ActionScript 1 was often used directly on the timeline frames to control the animation and interactivity.

**Events:** ActionScript 1 introduced basic event handling, allowing developers to trigger actions in response to user interactions (e.g., mouse clicks).

**Basic Data Types: ActionScript 1 supported basic data types such as numbers, strings, and Booleans.**

**Variables:** You could declare and use variables, but variable typing wasn't as strict as in later versions**.**

**Conditional Statements:** You could use if statements for conditional branching.

**Loops:** ActionScript 1 supported for and while loops for repetitive tasks.

**Built-in Functions:** It included a set of built-in functions to handle basic tasks like manipulating strings and working with numbers.

**Basic Interactivity:** ActionScript 1 was used to create simple interactive elements such as buttons and rollovers in Flash animations.

**Movie Clips:** Developers could use ActionScript 1 to control and animate movie clips, making it possible to create games and interactive content.

**External Data Loading:** Although more limited compared to later versions, ActionScript 1 allowed developers to load external data and display it in Flash applications.

**Custom Functions:** Developers could create their own custom functions to encapsulate a set of actions and reuse them.

**Basic Error Handling:** ActionScript 1 had rudimentary error handling, which was often done using on() and onClipEvent() to respond to errors and exceptions.

**Integration with Flash Authoring Tools:** ActionScript 1 was tightly integrated with the Flash authoring environment, allowing for a visual development approach with timeline-based animations and interactions.

It's important to note that ActionScript 1 was a simpler and less structured language compared to later versions, particularly ActionScript 3 (AS3). AS3 introduced advanced features like strong typing, advanced object-oriented programming, and a more robust event model. As a result, ActionScript 1 is considered outdated, and it's no longer a recommended choice for web development due to security and performance issues. Developers now prefer more modern technologies and languages for building web applications and games.

## ACTION SCRIPTING 1 OBJECT ORIENTED

ActionScript 1 (AS1) is the original scripting language used in Adobe Flash, introduced with Flash 4. While it's not as robust or object-oriented as later versions like ActionScript 3 (AS3), AS1 does support basic object-oriented programming (OOP) concepts to some extent. Here's how you can work with object-oriented programming in AS1:

**Creating Custom Objects:**

In AS1, you can create custom objects by defining functions (constructors) that act as object "blueprints." These functions can have properties (variables) and methods (functions).

```
function Person(name, age) {

   this.name = name;

   this.age = age;

}
```

**Instantiating Objects:**

You can create instances of custom objects by invoking the constructor function using the new keyword.

```
var person1 = new Person("John", 25);

var person2 = new Person("Alice", 30);
```

**Adding Methods to Objects:**

Methods can be added to objects by defining functions as part of the object's prototype.

```
Person.prototype.sayHello = function() {
    trace("Hello, my name is " + this.name + " and I'm " + this.age + " years old.");
}
```

**Using Inheritance:**

Inheritance is more limited in AS1, but you can simulate it to some extent by creating "subclasses" that inherit properties and methods from "superclasses."

```
function Student(name, age, studentId) {
    Person.call(this, name, age); // Call the superclass constructor
    this.studentId = studentId;
}
Student.prototype = new Person(); // Inherit methods from Person

Student.prototype.study = function() {
    trace(this.name + " is studying with student ID " + this.studentId);
}
```

**Note that this is not true classical inheritance as seen in more modern OOP languages like AS3.**

**Encapsulation:**

AS1 doesn't provide access modifiers like public or private. All properties and methods are accessible from anywhere within the code, which can make encapsulation more challenging.

**Object Reusability:**

Creating custom objects in AS1 allowed for some level of code reusability, making it possible to create instances with similar functionality.

**Custom Events (Limited):**

You could create custom events and dispatch them to trigger actions based on user interactions or other events.

**Error Handling:**

Basic error handling was available, often involving conditional statements to catch and respond to errors or exceptions.

AS1's approach to object-oriented programming is less structured and lacks the advanced features of later versions like AS3. AS3 introduced strong typing, more advanced class-based inheritance, a proper event model, and other features that made OOP more powerful and maintainable. While AS1 was suitable for its time, it's now considered outdated for modern web and application development due to security and performance concerns. Developers typically use more modern technologies and languages.

## DATATYPE AND TYPECHECKING ACTION SCRIPT1

In ActionScript 1 (AS1), data types and type checking were less strict and explicit compared to ActionScript 3 (AS3). AS1, which was primarily used with older versions of Adobe Flash, was a less structured and loosely typed language. Here's an overview of data types and type checking in AS1:

## Data Types in AS1:

**Number:** In AS1, all numbers, whether integers or floating-point, were typically handled as a single "Number" data type. This data type encompassed both integer and floating-point values.

```
var myNumber:Number = 42; // Integer
```

```
var myFloat:Number = 3.14; // Floating-point
```

String: Strings were used to represent text data.

```
var myString:String = "Hello, AS1!";
```

Boolean: Booleans represented true or false values.

```
var isTrue:Boolean = true;

var isFalse:Boolean = false;
```

**Object:** In AS1, "Object" was a generic data type used for various purposes, including custom objects and associative arrays.

```
var myObject:Object = { name: "John", age: 25 };
```

## Type Checking in AS1:

In AS1, type checking was not as strict as in AS3. It was a loosely typed language, which means that variables were not explicitly declared with data types, and type checking was often performed at runtime rather than at compile-time.

Here's an example of how type checking in AS1 was less strict:

```
var myNumber = 42; // No explicit data type declaration
myNumber = "Hello, AS1!"; // This assignment is allowed, even though it's not a valid operation
```

In the example above, there is no explicit data type declaration for the variable "myNumber," and you can assign a string to it even though it was previously used to store a number.

Due to the loose type checking and dynamic nature of AS1, it was more prone to runtime errors if type inconsistencies were not carefully managed. AS3 introduced stronger type checking, which caught many of these errors at compile-time, making code more robust and easier to maintain. However, AS1 was still a popular choice for web animations and interactive content in its time, and its simplicity made it accessible for designers and developers.

## Authoring an ActionScript 1

Authoring an ActionScript 1 class involves creating a reusable code structure for your Flash-based animations or applications. ActionScript 1 is the scripting language used in older versions of Adobe Flash. Here's a step-by-step guide to authoring an ActionScript 1 class:

## Open Adobe Flash:

Launch Adobe Flash or your preferred ActionScript 1 development environment.

## Create a New Flash Document:

Start by creating a new Flash document or open an existing one where you want to add your ActionScript.

## Insert a New Symbol:

If you plan to create a class for a movie clip or button, create a new symbol for it. Right-click on the stage, choose "Insert," and select "New Symbol."

## Define Your Class:

In your new symbol, define your class. For example, let's create a simple class for a movie clip called "MyMovieClip."

```
class MyMovieClip {
    // Class properties
    var xPosition:Number;
    var yPosition:Number;

    // Constructor
    function MyMovieClip() {
        xPosition = 0;
        yPosition = 0;
    }

    // Class methods
    function moveTo(x:Number, y:Number) {
        _x = x;
        _y = y;
        xPosition = x;
        yPosition = y;
    }
}
```

In this example, we've defined a class with properties (xPosition and yPosition), a constructor, and a method (moveTo) to manipulate the movie clip's position.

### Assign the Class to Your Symbol:

To associate the class with your symbol, select the symbol on the stage, open the "Properties" panel, and enter the class name in the "Linkage" section. Make sure the "Export for ActionScript" option is selected.

### Use Your Class:

Now you can create instances of your class and use them in your Flash project. For example:

```
var myClip1 = new MyMovieClip();

myClip1.moveTo(100, 200);


var myClip2 = new MyMovieClip();

myClip2.moveTo(300, 400);
```

This code creates two instances of your class, myClip1 and myClip2, and uses the moveTo method to change their positions.

### Testing Your Flash Document:

To see the results, you can test your Flash document by publishing it or using a test environment within Adobe Flash.

Remember that ActionScript 1 is quite different from more modern programming languages, and it doesn't support advanced object-oriented features. If you're developing for contemporary platforms, consider using ActionScript 3 or other modern languages and technologies.

# ACTION SCRIPT 2 INHERITANCE

In ActionScript 2, inheritance allows you to create new classes that inherit properties and methods from existing classes. This is a fundamental concept in object-oriented programming that promotes code reusability and modularity. Here's how you can implement inheritance in ActionScript 2:

## Define the Base (Parent) Class:

Start by creating a base or parent class that contains the properties and methods you want to share with the derived (child) classes.

```
class Animal {
    var name:String;
    var age:Number;

    function Animal(name:String, age:Number) {
        this.name = name;
        this.age = age;
    }
       function speak():Void {
        trace("Animal speaks!");
    }
}
```

## Define the Derived (Child) Class:

Create a derived class by extending the base class using the extends keyword. This child class will inherit all the properties and methods of the parent class.

```
class Dog extends Animal {
    var breed:String;

    function Dog(name:String, age:Number, breed:String) {
        super(name, age); // Call the parent class constructor
        this.breed = breed;
    }

    // You can override or extend inherited methods
    function speak():Void {
        trace(name + " barks!");
    }
}
```

## Creating Instances:

You can now create instances of both the parent and child classes.

```
var genericAnimal:Animal = new Animal("Generic Animal", 5);
var myDog:Dog = new Dog("Buddy", 3, "Golden Retriever");
```

## Using Inherited Properties and Methods:

Instances of the child class can access both the properties and methods of the parent class.

```
genericAnimal.speak(); // Outputs: "Animal speaks!"
myDog.speak();         // Outputs: "Buddy barks!"
```

## Accessing Properties of the Parent Class:

Instances of the child class can access properties of the parent class directly.

```
var dogName:String = myDog.name; // Accessing the inherited 'name'
property
```

## Overriding Methods:

You can override a method from the parent class by providing a new implementation in the child class. In the example above, the speak method is overridden in the Dog class to provide a different behavior.

## Using the super Keyword:

The super keyword is used to call the constructor of the parent class. It's also used to call methods of the parent class from within the child class.

```
super(name, age); // Call the parent class constructor
```

Inheritance is a powerful concept in ActionScript 2 that allows you to create a hierarchy of classes, reusing and extending functionality, and promoting code organization and maintainability.

## AUTHORING AN ACTIONSCRIPT 2.0 SUBCLASS

Authoring an ActionScript 2.0 subclass involves creating a new class that inherits properties and methods from an existing class, known as the superclass or parent class. Subclassing allows you to reuse and extend the functionality of the parent class. Here are the steps to create an ActionScript 2.0 subclass:

**Create the Parent Class (Superclass):**

Start by creating the parent class (superclass). This class will contain properties and methods that you want to inherit in the subclass.

```
class Animal {
    var name:String;
    var age:Number;

    function Animal(name:String, age:Number) {
        this.name = name;
        this.age = age;
    }

    function speak():Void {
        trace("Animal speaks!");
    }
}
```

## Create the Subclass:

To create a subclass, you use the extends keyword followed by the name of the parent class. This indicates that your new class inherits properties and methods from the parent class.

```
class Dog extends Animal {
    var breed:String;
    function Dog(name:String, age:Number, breed:String) {
        super(name, age); // Call the parent class constructor
        this.breed = breed;
    }

    function bark():Void {
        trace(name + " barks!");
    }
}
```

### Instantiating the Subclass:

You can now create instances of both the parent and child classes. When you create an instance of the subclass, it inherits properties and methods from the parent class.

```
var genericAnimal:Animal = new Animal("Generic Animal", 5);
var myDog:Dog = new Dog("Buddy", 3, "Golden Retriever");
```

### Using Inherited Properties and Methods:

Instances of the subclass can access properties and methods inherited from the parent class.

```
genericAnimal.speak(); // Outputs: "Animal speaks!"
myDog.speak();         // Outputs: "Buddy speaks!"
```

### Using Subclass-Specific Properties and Methods:

Subclasses can also have properties and methods specific to them, in addition to what they inherit.

```
myDog.bark(); // Outputs: "Buddy barks!"
```

Accessing Properties of the Parent Class:

Instances of the subclass can directly access properties inherited from the parent class.

```
var dogName:String = myDog.name; // Accessing the inherited 'name' property
```

**Calling the Parent Class Constructor:**

You can use the super keyword in the subclass's constructor to call the constructor of the parent class. This is necessary to initialize the inherited properties of the parent class.

```
function Dog(name:String, age:Number, breed:String) {
   super(name, age); // Call the parent class constructor
   this.breed = breed;
}
```

Creating a subclass in ActionScript 2.0 allows you to extend and customize the functionality of existing classes, promoting code reusability and maintainability.

## ACTIONSCRIPT 2 INTEFACES, PACKAGES, EXCEPTIONS

ActionScript 2 (AS2) lacks some of the more advanced features and concepts found in later versions, such as AS3. In AS2, there are no explicit support for interfaces, packages, and exceptions like you'd find in modern programming languages. However, you can work with AS2 in a way that simulates some of these concepts:

### 1. Interfaces (Simulation):

In AS2, you can simulate interfaces by creating a class with abstract methods. Objects of this class can't be instantiated, but other classes can extend this class and provide implementations for the abstract methods.

```
class MyInterface {
    function doSomething():Void;
}

class MyClass extends MyInterface {
    function doSomething():Void {
        trace("Implementation of doSomething in MyClass.");
    }
}

var myObject:MyInterface = new MyClass();
myObject.doSomething();
```

In this example, MyInterface serves as a pseudo-interface, and MyClass provides an implementation for the doSomething method.

## 2. Packages (Simulation):

AS2 doesn't have native support for packages like AS3 or other programming languages. However, you can create a folder structure to organize your classes in a way that resembles packages. For example, you can create a directory structure like this:

```
com/
   example/
      MyClass1.as
      MyClass2.as
```

In this structure, you can use the package name com.example as a way to organize your classes. This is a common practice to avoid naming conflicts.

## 3. Exceptions (Simulation):

AS2 doesn't support structured exception handling like try-catch blocks in AS3 or other languages. To handle errors or exceptions in AS2, you typically use conditional statements and error codes. For example:

```
var result:Number;
var error:Boolean = false;

if (error) {
   trace("An error occurred.");
} else {
   result = 10 / 0; // Simulating a potential error
   trace("Result: " + result);
}
```

In the above example, we use a Boolean variable to simulate an error condition. When an error is detected, we handle it with a conditional statement. You might use error codes or flags to indicate different types of errors.

Keep in mind that AS2 is an older technology, and these approaches are workarounds to mimic some of the features found in more modern languages and AS3. If you have the option, it's recommended to use AS3 or other modern languages that provide better support for interfaces, packages, and structured exception handling.