Overview
- Scripting Concepts
- Script Interpreters ("Engines")
- Scripting Languages
  - JavaScript Basics
- Communicating with Scripts
- Using Scripts in Games
- Additional Scripting Engines

Scripting
- Using external code to alter game world structure or game play
- Common "scripting languages":
  - JavaScript, Python, Lua
  - Others (Tcl, Scheme, Ruby, SmallTalk, VB...)
- Scripts must have access to game objects
  - Or at least, to a API
- Requires embedding an interpreter in game/game engine

Script Interpreters
- Rhino/Nashorn (JavaScript)
- Jython (Python)
- LuaJ (Lua)

*Also known as a Script Engine

Using a script engine in Java
- Get the Java script engine manager
- Use it to get the desired script engine
- Use `eval(...)` to run the script interpreter
  - `eval(String, or`  `eval(FileReader)`
- Scripts can also be compiled

JavaScript Basics

Comments
- Same as Java:  `//` or `/*...*/`  (no JavaDoc `/**...*/` form)

Variables
- Declared with `var` (optional)
- Either `global` or `local` (inside a function) – no “class scope”
- Syntax: same as Java (e.g. start with letter or `_`)
  - Cannot use reserved words (most Java reserved words, plus others)
- "weakly typed" – type determined by assigned value

```
var i = 0;  // i is an int
var pi = 3.14159;  // pi is a real
var j = "Hello";  // j is a string
var k = 42 + " is the answer";  // k is also a string
var m = i < 10;  // m is a Boolean = true
```
### JavaScript Basics (cont.)

**Operators**

Same as Java (+, *, /, %, ==, !=, <, >, <=, >=, &&, ||, !, =)

**Control statements**

if (cond) {...} else {...} // same as Java
for (var i=0; i<3; i++) {...} // almost Java
while (cond) {...} // same as Java
try {...} catch(e){...} // same as Java

**Functions**

- Global scope by default
- Defined with keyword: `function`

### Communicating with Scripts

The Java Scripting API: `javax.script.*`

- Allows Java to:
  - Pass data into a script: `engine.put()`
  - Get data back from a script: `engine.get()`

- Scripts can assign values to vars accessible by Java
- Scripts also have a “return value”

- Allows scripts to:
  - Get data from Java
  - Pass data to Java
  - Invoke methods in Java objects

### Java/Script Communication

**Java code:**

```java
int count = 3;
engine.put("count", count);
int[] vals = {10, 20, 30};
engine.put("vals", vals);
FileReader fr = new FileReader("sums.js");
boolean result = (Boolean)engine.eval(fr);
double sum = (Double)engine.get("sum");
System.out.print("Java sum = " + sum);
```

**JavaScript in file "sums.js":**

```javascript
print('count = ' + count);
var sum = 0;
for (var i=0; i<count; i++) {
    sum += vals[i];
}
print("JS sum = " + sum);
```

### Invoking Script Functions

**Define script function**

**Load function into engine (using eval())**

**Cast engine as an “Invocable” object**

**Use `invokeFunction()` to call function**

**Java code:**

```java
FileReader fr = new FileReader("sayHello.js");
engine.eval(fr);
Invocable invocableEngine = (Invocable) engine;
Object[] arg = {"Rufus"};
invocableEngine.invokeFunction("sayHello", arg);
```

**JavaScript in file "sayHello.js":**

```javascript
function sayHello(name)
{
    print("Hello " + name);
}
```

### Uses for scripting

- Gameworld creation and initialization
- Dynamically modifying game details
- Providing user-defined functions that can be called from a Java application
- Modifying player and non-player characters
- Modifying game features
- Testing

### Using Other Script Engines

**Lua**

```lua
local x = 25
print(math.sqrt(x))
```

**Python**

```python
import sys
import java.io as javaio
currentdir = javaio.File(".").getCanonicalPath()
print("Hello Python: Current directory:

currentdir.getCanonicalPath()
```

```java
String lsep = System.getProperty("line.separator");
String a = "import sys" + lsep;
a += "import java.io as javaio" + lsep;
a += "currentdir = javaio.File(".")" + lsep;
a += "print" + lsep;
try {
    eval(a);
} catch (ScriptException e) {
    System.out.println(e);
}
```

---

**Script Engine**

<table>
<thead>
<tr>
<th>Name</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>count</td>
<td>3</td>
</tr>
<tr>
<td>vals</td>
<td>ref</td>
</tr>
<tr>
<td>sum</td>
<td>60</td>
</tr>
</tbody>
</table>

**ScriptEngine**

```
ScriptEngine luaEngine = factory.getEngineByExtension(".lua");
luaEngine.put("x", 25);
luaEngine.eval("y = math.sqrt(x)" );
System.out.println("Hello Lua: y=" + luaEngine.get("y") );
```

```java
String lsep = System.getProperty("line.separator");
String a = "import sys" + lsep;
a += "import java.io as javaio" + lsep;
a += "currentdir = javaio.File(".")" + lsep;
a += "print" + lsep;
try {
    eval(a);
} catch (ScriptException e) {
    System.out.println(e);
}
```
### Additional JavaScript Features

#### Arrays
- Size defined by parentheses at declaration
  ```javascript
  var foo = new Array(10);
  var bar = new Array(5);
  ```
- Indexing from zero and using brackets (like Java)
  ```javascript
  foo[0] = 42;  bar[4] = 99.9;
  ```
- Mixed element types allowed
  ```javascript
  var stuff = new Array("a string", 12, 98.6, true);
  ```
- Dynamically resizeable
  ```javascript
  var colors = new Array();  // colors has no elements
  colors[2] = "red";  // colors now has 3 elements
  ```
- Properties and methods
  ```javascript
  length, indexOf(), concat(), toString(), ...
  ```

#### Built-in Objects
- `var currentTime = new Date();`
- `var month = currentTime.getMonth() + 1;`
- `var day = currentTime.getDate();`
- `var year = currentTime.getFullYear();`

#### User-created Objects
- `var personObj=new Object();`
  ```javascript
  personObj.firstname="John";
  personObj.lastname="Doe";
  personObj.age=50;
  personObj.eyecolor="blue";
  ```
- `var myFather = new person("John","Doe",50,"blue");`
- `var myMother = new person("Sally","Rally",48,"green");`
- `myMother.newLastName("Doe");`