

# EScript

## A Short Introduction

Benjamin Eikel, Claudius Jähn

Version: June 16, 2012



- 1 Introduction
- 2 Data Types and Operators
- 3 Functions and variables
- 4 Control Structures
- 5 Other Features
- 6 Examples

- 1 Introduction
- 2 Data Types and Operators
- 3 Functions and variables
- 4 Control Structures
- 5 Other Features
- 6 Examples

# What is EScript?



EScript . . .

- is an object-oriented scripting language.
- is compiled and executed by a virtual machine.
- has a similar syntax to C.
- was developed to use C++ objects from scripts easily.

# What is EScript?



EScript ...

- is released under a free software license (MIT).
- is available from <https://github.com/EScript>.
- has a command-line interpreter.
- can be built using CMake.
- can be used internally by other C++ projects (e.g. by PADrend <http://PADrend.de>).
- stands for HasE-Script.

## First Example

- A simple script: HelloWorld.escript

```
outln( "Hello, world!" ); // Outputs: Hello World!
```

- Calls the global function `out` with the string "Hello world!" as parameter value.
- The statements ends with a semicolon.
- Comments begin with `//` or are enclosed `/* */`.

- 1 Introduction
- 2 Data Types and Operators**
- 3 Functions and variables
- 4 Control Structures
- 5 Other Features
- 6 Examples

## Simple Types (call-by-value)

### Number

42

27.4

0x1a

-2.7e+10

0b101010

### String

"an"

'example string'

### Bool

**true**

**false**

### Void (empty or no value)

**void**



# Operators

## Some operators

```
outln( 2+40 ); // Output: 42
outln( 2*21 ); // Output: 42
outln( "4" + "2" ); // Output: 42
outln( "foo"+"bar" ); // Output: foobar
outln( "wup " * (6/2) ); // Output: wup wup wup
outln( 1>2 ); // Output: false
outln( !true ); // Output: false
outln( true & true ); // Output: true
outln( false || true ); // Output: true
outln( "foo" == "bar" ); // Output: false
outln( "foo" != "bar" ); // Output: true
```

## Type Conversion

Only **false** and **void** convert to **false**

```
println(false || false); // Output: false
println(false || void); // Output: true
println(false || 0); // Output: true
println(false || ""); // Output: true
```

## Type Conversion

Only **false** and **void** convert to **false**

```
outln(false || false); // Output: false
outln(false || void); // Output: true
outln(false || 0); // Output: true
outln(false || ""); // Output: true
```

Conversion to number (left operand is a number)

```
outln( 12 + "3" ); // Output: 15
outln( 10 * "10" ); // Output: 100
outln( 10 == "10" ); // Output: true
outln( 10 == "10.0" ); // Output: true
```

## Type Conversion

Only **false** and **void** convert to **false**

```
outln(false || false); // Output: false
outln(false || void); // Output: true
outln(false || 0); // Output: true
outln(false || ""); // Output: true
```

Conversion to number (left operand is a number)

```
outln( 12 + "3" ); // Output: 15
outln( 10 * "10" ); // Output: 100
outln( 10 == "10" ); // Output: true
outln( 10 == "10.0" ); // Output: true
```

Conversion to string (left operand is a string)

```
outln("12" + 3); // Output: 123
outln("10" == 10); // Output: true
outln("10.0" == 10); // Output: false
```

## Equality checks

Check equality with conversion ==

Check equality without conversion ===

```
outln( 10 == "10" ); // Output: true
outln( 10 === "10" ); // Output: false
outln( 10 === 10 ); // Output: true
outln( true == "foo" ); // Output: true
outln( true === "true" ); // Output: false
outln( "true" == true ); // Output: true
outln( "true" === true ); // Output: false
```

# Overview

- 1 Introduction
- 2 Data Types and Operators
- 3 Functions and variables**
- 4 Control Structures
- 5 Other Features
- 6 Examples

# Calling functions

## Calling functions with different origins

```
// call global function 'load':  
load( "someScript.escript" );  
  
// call function 'saveTextFile' in namespace 'IO':  
IO.saveTextFile( "foo.txt" , "bar" );  
  
// call member function 'sqrt' of object 9.0:  
out( (9.0).sqrt() ); // Output: 3
```

# Declaring Variables

Declare a variable with **var**:

```
// "foo" is an empty variable (contains void).  
var foo;  
  
// The variable "xPos" contains a number  
var xPos = 500 - 80 / 2;  
  
// The variable "message" will be of type String  
var message = "Please click the button";  
  
// Dynamically change the type to Number  
message = 5;
```



## Declaring simple functions

- Declare functions with **fn**.
- Functions have no names, but they can be stored in a variable:

```
var square = fn(num) {  
    return num * num;  
};  
var a = square(5);  
var b = square(4.2);
```

## Advanced Types (1)

### Array

```
var numbers = [3, 23, 7, 3, 100, 1, 35];  
var colors = ["red", "green", "blue"];  
outln( numbers[4] ); // Outputs: 100  
outln( numbers.count() ); // Outputs: 7  
outln( numbers.empty() ); // Outputs: false
```

### Map

```
var fruits = {  
    "lemon" : "yellow",  
    "cherry" : "red"  
};  
fruits["apple"] = "green";
```

# Overview

- 1 Introduction
- 2 Data Types and Operators
- 3 Functions and variables
- 4 Control Structures**
- 5 Other Features
- 6 Examples

## Conditionals (1)

Conditional execution with **if/else**.

```
var result = /* some function */;
if(result) {
    out("Success");
} else {
    out("Failure");
}
```

```
var num = /* some number */;
if(num < 0) {
    out("Too small");
} else if(num >= 0 && num <= 100) {
    out("Range okay");
} else {
    out("Too large");
}
```

## Conditionals (2)

? (conditional operator)

```
var num = /* some number */;  
var positive = (num > 0) ? true : false;
```

## Conditionals (2)

? (conditional operator)

```
var num = /* some number */;  
var positive = (num > 0) ? true : false;
```

## Loops (1)

### Looping with **while**:

```
var tasks = [/* some tasks */];  
while (!tasks.empty()) {  
    var firstTask = tasks.front();  
    tasks.popFront();  
    // do something with first task  
}
```

## Loops (2)

### Looping with **for**:

```
var sum = 0;
for(var i = 0; i < 100; ++i) {
    sum += i;
}
out("Sum of numbers: ", sum, "\n");
```



## Loops (3)

Iterate over a container: **foreach**.

```
var chars = ["a", "c", "k", "b", "d", "x", "j"];  
foreach(chars as var i, var c) {  
    if(c === "x") {  
        out("Character 'x' found at index " + i);  
        break;  
    }  
}
```

Output: Character "x" found at index 5

# Overview

- 1 Introduction
- 2 Data Types and Operators
- 3 Functions and variables
- 4 Control Structures
- 5 Other Features**
- 6 Examples

## Extendable object

Extendable objects: **ExtObject**.

```
var car = new ExtObject;  
car.color := "red"; // create new member  
car.speed := 190;  
car.outputDesc := fn() {  
    out("This is a ", this.color, " car ");  
    out("with top speed ", this.speed, ".\n");  
};  
  
...  
  
car.speed = 185;  
car.outputDesc();
```

Output: This is a red car with top speed 185.

# Inheritance

## Types and inheritance:

```
var Shape = new Type;
Shape.color := "white";

// New type that is derived from Shape
var Polygon = new Type(Shape);
Polygon.numVertices := 3;

// New type that is derived from Shape
var Circle = new Type(Shape);
Circle.radius := 0;

var circle = new Circle;
circle.color = "red";
circle.radius = 5;
```

# Overview

- 1 Introduction
- 2 Data Types and Operators
- 3 Functions and variables
- 4 Control Structures
- 5 Other Features
- 6 Examples**

## Delegation

Call a function on another object.

### Example

```
var printOut = fn() {  
  out("I am a " + this.color + " node.\n");  
};  
  
var nodeRed = new ExtObject();  
nodeRed.color := "red";  
var nodeBlack = new ExtObject();  
nodeBlack.color := "black";  
  
var printOutRed = nodeRed -> printOut;  
var printOutBlack = nodeBlack -> printOut;  
  
printOutRed(); // Output: I am a red node.  
printOutBlack(); // Output: I am a black node.
```

## Example

```
var Polygon = new Type();  
Polygon.vertices @(private, init) := Array;  
Polygon.shapeType @(const) := "Polygon";  
  
Polygon.getNumVertices := fn() {  
    return this.vertices.count();  
};  
  
var polygon = new Polygon();  
polygon.getNumVertices();
```

# Overview

- 1 Introduction
- 2 Data Types and Operators
- 3 Functions and variables
- 4 Control Structures
- 5 Other Features
- 6 Examples



# Factorial

Factorial:  $n! = 1 \cdot 2 \cdot 3 \cdot \dots \cdot n$      $0! = 1$

## Example

```
var factorialRecursive = fn(Number n) {  
  return (n == 0) ? 1 : thisFn(n - 1) * n;  
};  
var factorialIterative = fn(Number n) {  
  var product = 1;  
  for(var i = 2; i <= n; ++i) {  
    product *= i;  
  }  
  return product;  
};  
  
outln(factorialRecursive(6)); // Output: 720  
outln(factorialIterative(7)); // Output: 5040
```

## Example

```
var Player = new Type();
Player.x @(private) := 0;
Player.y @(private) := 0;
Player.move ::= fn(Number dx, Number dy) {
    this.x += dx;
    this.y += dy;
};
Player.printPos ::= fn() {
    outln("Position: (", this.x, ", ", this.y, ")");
};

var playerA = new Player();
playerA.move(5, 7);
playerA.printPos(); // Output: Position: (5, 7)
```

## Further Documentation

You can find additional documentation in  
`EScript/docs/Introduction.html`.