Psychological Task Design & Development

A Programming Workshop
Part II*B* – Programming Basics

Wouter Boendermaker, M.Sc.

Johanna Quist, M.Sc.

Soraya Sanchez Maceiras, B.Sc.

University of Amsterdam

EPP Programming Workshop – February 12-13, 2015

4-Step Programme to Programming

Programming: Manipulating stuff through code

I. **Variables** (store values & complex Objects)

II. **Operations** (manipulate variables)

III. **Decisions** (make your program dynamic)

IV. **Repetitions** (i.e., how to avoid them!)

I. Variables - Names

Variables are used to hold all kinds of data.

Naming

- No spaces allowed in variable names: use CamelCase
- Common conventions (not compulsory, but strongly advisable):
    1. Write all words together, starting each new one with an UpperCase:
        - `myVarName`
        - `MyClassName`
    2. Another variant is using underscores (_) instead of spaces:
        - `MY_CONST`
- Please use just one style: have a system, make your code readable!

I. Variables - Syntax

Declaring (making) a variable, data typing and assigning a value:

```
var <varName>:<dataType> = value;
```

- `var` indicates to Flash "I'm declaring a new variable here".
- `<varName>` can be anything, but make sure it makes sense.
- `:<dataType>` tells Flash what type of data is stored.

  NB: Variables in Flash are strongly typed: once assigned,
  you cannot assign a differently typed value to it.
- `= value` use a single = to assign a value to the variable.
- `;` End every command with a semicolon to tell Flash that it ends there.

NB: please replace anything between < >, e.g.:

```
var numberOfTrials:int  = 3;
```

I. Variables – Basic Data Types

```
var <varName>:<dataType> = value;
```

- Numbers:
  int : whole numbers +/- (integer)
  uint : whole numbers + (unsigned integer)
  Number : digit numbers
- Letters/text:
  String : all kinds of text, within "quotes"
- Logical (yes or no):
  Boolean : true or false
- Collections:
  Array : any number of values or other variables

I. Variables – Constants

Constants are a special (not-so-variable) variant.

- Similar to variables, but can be assigned a value only **once**
- To easily see the difference, I use UPPERCASE_NAMES:

```
const <CONST_NAME>:<dataType> = value;
```

- Use constants to define key values, e.g.:

```
const EXP_CONDITION:uint   = 1;
const DEBUG:Boolean        = true;
```

I. Variables – Scope

Remember our first piece of code (Test.as)?

```
package {
    import flash.display.MovieClip;
    public class Test extends MovieClip {
        var myGlobalVar:int = 3;
        public function Test() :void {
            // this is the constructor function
            // put your code between these {curly brackets}
            var myLocalVar:int = 3;
            trace( myLocalVar );
        }
    }
}
```

Scope: Variables have a so called ‘scope’. That means they are accessible only in a certain area, depending on where you **declare** (make) them.

Each area is delimited with {curly brackets}. Use **TABs** whenever you use curly brackets.

Global and **local:** The most important scope distinction to make is between the **Class level** (**global**) and inside one (of many) **functions** inside the **Class** (**local**).

Additional (**scope**) properties you may encounter: `public`, `private`, `internal`, `protected`, `static`, `final`.

Exercise 2 - Variables

1. Download the **epw_ex2.zip** file from www.wouboe.nl.
2. Open **Exercise2.as** and **Exercise2.fla** in Flash.

Hint: Use `trace(…);` in the assignments below to send output to the output box at the bottom of the screen.

1. Make a **global variable** called `varA` (type `int`) with value 1.
2. Inside your **constructor function**, make a **local variable** `varA` (also type `int`), but with value 2. Are they the same?
3. Now make another **local variable** `varB`, also in your **constructor function**. Can you `trace` it from within your second function?
4. Make two **local variables**: `varC1` (`int`) with value 2 and `varC2` (`String`) with value "Hello". Can you combine them and then `trace` them? Can you assign an integer value to `varC2`, after we've given it a type?
5. Challenge: Make two variables: a (`int`) = 3 and b (`int`) = 5. Make a script that **swaps** the values of a and b (so in the end, a is 5 and b is 3).

II. Operations - Simple

Operations are used to modify values and variables.

Simple operators:

- `+` (**addition** with numbers, **concatenation** with strings)
- `-` (**subtraction**)
- `*` (**multiplication**)
- `/` (**division**)
- % (**modulo**: finds remainder after division of one number by another)

Special: To shorten things a bit:

```
a = a + 1;
a += 1;
a ++;
```

`a ++;` (only works with increment 1)

This also works for `-=`, `*=`, `/=` and %=.

II. Operations - Equations

Equation operators:

== (equal)
>= (greater than or equal to)
> (greater than)
< (less than)
<= (less than or equal to)
! = (**not** equal)

Note:

- with Strings we only use == and ! =
- = sign always on the right
- = is the sign for assigning a value to a variable
- == is the equation sign, where two values are compared
- (=== also exists: strict equality; not important now)

II. Operations - Logical

Logical operators:

&& (**AND**)
| | (AND/**OR**)
! (**NOT**, converts whatever its next to the opposite Boolean value:

```
!false == true
!true  == false
```

Given x, y:

x	y	:	x && y	x || y
false	false	:	false	false
false	true	:	false	true
true	false	:	false	true
true	true	:	true	true

II. Operations - Example

Given:

```
var a:int = 8;
var b:Number    = 2.5;
var c:String    = "hello";
var d:Boolean   = false;

( ( a > b ) || ( c == "HELLO" && !d ) )

∴   ( true  || ( false && (! false) ) )
∴   ( true  || ( false && true ) )
∴   ( true  || ( false ) )

∴   true
```

III. Decisions - if / else (1)

Decisions are used to make choices, to make code dynamic.

When deciding if a value or a variable conforms to a certain condition, we can use the **if / else** statement:

```
if( <conditionA> == true ) {
  // execute commandA;
} else {
  // execute another command;
}
```

Note:

- The curly brackets `{...}` These denote a **section** of code to be executed, with its own **scope**.
- `//` means the rest of the line is **comment** (skipped by Flash)

III. Decisions - if / else (2)

More elaborately, one can make several levels of (**nested**) `if` / `else` trees:

```
if( <conditionA> == true ) {
  if( <conditionB> == true ) {
    // execute commandAB;
  } else {
    // execute commandA;
  }
} else {
  if( <conditionB> == true ) { // So A is false; B is true
    // execute commandB;
  } else {
    // don’t execute any command;
  }
}
```

Note:

- To denote nesting, use **TABs** whenever you use curly brackets.
- The `else` condition automatically runs when the corresponding `if` conditions are `false`.
- `if` can occur without a consecutive `else`. Then just nothing happens.

III. Decisions - if / else (3)

Another use of consecutive `if` / `else` statements is the following:

```
if( myAge <= 22 ) {
  // execute commandA;
} else if( myAge == 23 ){
  // execute commandB;
} else if( myAge == 24 ){
  // execute commandC;
} else if( myAge == 25 ){
  // execute commandD;
} else {
  // execute commandZ;
}
```

Note:

- Running from top to bottom, once one of them is true, the {…} code is executed and we exit the `if/else` tree.

III. Decisions - Switch!

For this last `if`-variant, a nice alternative exists: The `switch`-statement:

```
switch( myCondition ) {
   case PLACEBO_CONDITION:
        //execute commandA;
        break;
   case EXPERIMENTAL_CONIDITION:
        //execute commandB;
        break;
   default:
        // execute commandZ;
}
```

Note:

- The `switch`-statement is useful for readability (use with constants)
- Must use `break` command to exit the `switch`.
- The `default` command equals the general `else`-statement

Exercise 3 - Decisions

1. Download **epw_ex3.zip** from www.wouboe.nl.
2. Open **Exercise3.as** and **Exercise3.fla** in Flash.

3. Make a decision tree that determines whether the randomly generated value of variable `rand` falls within certain categories.
4. Make a `switch` statement that determines in which condition we have been put.
5. What happens in a `switch` if you leave out the `break` statements? (try it!)

IV. Repetitions – Loops

Repetitions are used to make multiple use of the same code.

Loops execute some code repeatedly until a certain condition is met.

```
for( var <counterName>:<dataType> = startingValue;
     <counterName> < maximumValue;
     <counterName> ++ ) {
  // do something;
}
```

e.g.:

```
for( var i:uint = 0; i < 5; i ++ ) {
  trace( i );
}
```

Note:

- The `trace()` function allows you to output any value to the debug panel.

Exercise 4 - Loops

1. Download **epw_ex4.zip** from www.wouboe.nl.
2. Open **Exercise4.as** and **Exercise4.fla** in Flash.

3. Create a `for` **loop** that repeats exactly **4** times. Use the loop to `trace` the numbers 3 5 7 9, consecutively.
   - Challenge: instead, call the trace function only **once**; make it as short / elegant as possible.
4. Now have it `trace` the numbers 9 7 5 3.
5. Write some code that counts from 1 to 10 and decides for each number whether it's odd or even. Use:
   - `for`
   - `if/else`
   - `trace()`
6. Challenge: Create an `Array` containing the numbers 11 to 30. Loop through this array and calculate the factorial (11*12*13*…) of only the numbers that are dividable by 3.