Expressions, Statements, Programming Style, and Test Review
Today

- Review last week
- Expressions/Statements
- Programming Style
- Reading/writing IO
- Test review!
Trace Statements

- Purpose is to display values or messages in the Output Panel to see the state or progress of a program or animation
  - Very useful for debugging (when your program isn’t doing what you want it to do and you don’t know why)
- **Only seen by the programmer**, trace statements don’t “do” anything to your program
Examples of Trace Statements

- (try these on your own)

```javascript
trace( 525 );
trace ( "Now starting love scene" );
trace ( "Row " + 5 + "Col " + 7 );
trace ( 5 + 7 );
trace ( "5 + 7 = " + 5 + 7 );
```

**Script**

<table>
<thead>
<tr>
<th>525</th>
</tr>
</thead>
<tbody>
<tr>
<td>Now starting love scene</td>
</tr>
<tr>
<td>Row 5Col 7</td>
</tr>
<tr>
<td>12</td>
</tr>
<tr>
<td>5 + 7 = 57</td>
</tr>
</tbody>
</table>

**Output Window**
Variables!

- Might help to think of variables in programming like your $x$, $y$, $z$ variables in math equations
  - Their purpose is to store values
- Variables in programming are not just numbers (more later)!
Defining Variables - Example

```javascript
var row = 5;
row = 8;
var next = row + 1;
next = next + 1;
var s = "hi";
var val = 3.5;
```

<table>
<thead>
<tr>
<th>next</th>
<th>10</th>
</tr>
</thead>
<tbody>
<tr>
<td>row</td>
<td>8</td>
</tr>
<tr>
<td>val</td>
<td>3.5</td>
</tr>
<tr>
<td>s</td>
<td>hi</td>
</tr>
</tbody>
</table>
DataTyping Variables

- Not required by ActionScript but recommended!
- Purpose is to explicitly specify the data type that a variable holds in order to reduce the potential for errors when using that variable
  - Ex. You might not want to add two variables of different types together on purpose! You would rather get an error
### DataTyping Variables - Example

#### Script

```plaintext
var row: int = 5;
row = 8;
var next: int = row + 1;
next = next + 1;
var s: String = “hi”;s
var val: Number = 3.5;
```

#### Memory

<table>
<thead>
<tr>
<th>next</th>
<th>10</th>
</tr>
</thead>
<tbody>
<tr>
<td>row</td>
<td>8</td>
</tr>
<tr>
<td>val</td>
<td>3.5</td>
</tr>
<tr>
<td>s</td>
<td>“hi”</td>
</tr>
</tbody>
</table>
Operators

- Operators allow for operations to occur in a program
  - Numeric
  - String operators
  - Other (assignment, user defined)
- They are used in an expression to achieve a result
Numeric Operators

Operators:

+ Addition
- Subtraction
* Multiplication
/ Division

% Mod (integer remainder of division)

<table>
<thead>
<tr>
<th>Expression</th>
<th>Result</th>
</tr>
</thead>
<tbody>
<tr>
<td>5 / 2</td>
<td>2.5</td>
</tr>
<tr>
<td>5 % 2</td>
<td>1</td>
</tr>
</tbody>
</table>
String Operators

- (Strings = a “string” of characters)

Operators:

- +       Concatenation
- +=      Append

```javascript
trace("Hey" + "you");
var course = "CISC";
var num = 110;
course += num;
trace("Course is: " + course);
```

Heyyou
Course is: CISC110
Escape Sequences

- Let your format your text
  - Allow you to control what is looks like when it is displayed

\n New Line
\t Tab
\' Single Quote
" Double Quote
\\ Backslash

trace( “Hey\nyou” );
trace( “Hey\tyou” );
trace( “Hey ”Kiddo”!” );

Hey
You
Hey you
Hey “Kiddo”!
Comments

- Allow you to document your code so that when you come back to it later you remember what you did!
- Also useful for other people who might look at your code (i.e. myself or your TAs)
- Comments are only seen by the programmer and are not read/execute by the computer
  - They’re like sticky notes for your code
- Also useful for debugging
  - If you don’t want all your code to run you can “comment it out”
Comments

Two Ways to Comment Code:

// This is a comment

/*
This is a comment spanning multiple lines
*/
## Data Types

<table>
<thead>
<tr>
<th>Data Type</th>
<th>Example</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Number</td>
<td>4.5</td>
<td>Any number</td>
</tr>
<tr>
<td>int</td>
<td>-5</td>
<td>Any integer</td>
</tr>
<tr>
<td>uint</td>
<td>1</td>
<td>Unsigned integer: any non-negative integer</td>
</tr>
<tr>
<td>String</td>
<td>“hello”</td>
<td>Text or a string of characters</td>
</tr>
<tr>
<td>Boolean</td>
<td>true</td>
<td>true or false</td>
</tr>
</tbody>
</table>
Boolean Variables

- Variables that store “true” or “false” values
- Ex:

```javascript
var gameOver: Boolean;  // Creates a Boolean variable

gameOver = false;      // Sets variable to hold value false
                       // (At start of game, it’s not over)

...  // Spend time playing the game and then check score

gameOver = true;       // Sets variable to hold value true
```
### Variable Declarations

<table>
<thead>
<tr>
<th>Data Type</th>
<th>Example</th>
</tr>
</thead>
<tbody>
<tr>
<td>Number</td>
<td>var x: Number;</td>
</tr>
<tr>
<td>int</td>
<td>var y: int;</td>
</tr>
<tr>
<td>uint</td>
<td>var z: uint;</td>
</tr>
<tr>
<td>String</td>
<td>var name: String;</td>
</tr>
<tr>
<td>Boolean</td>
<td>var over: Boolean;</td>
</tr>
</tbody>
</table>

These only find a space in memory and give it a name. The memory space is not given any value.
## Variable Declarations and Assignments

<table>
<thead>
<tr>
<th>Data Type</th>
<th>Example</th>
</tr>
</thead>
<tbody>
<tr>
<td>Number</td>
<td><code>var x: Number = -55.8;</code></td>
</tr>
<tr>
<td>int</td>
<td><code>var y: int = -55;</code></td>
</tr>
<tr>
<td>uint</td>
<td><code>var z: uint = 55;</code></td>
</tr>
<tr>
<td>String</td>
<td><code>var name: String = “MySpace”;</code></td>
</tr>
<tr>
<td>Boolean</td>
<td><code>var over: Boolean = false;</code></td>
</tr>
</tbody>
</table>

These find a space in memory and give it a name, and also assign an initial value.
Variable Assignments

<table>
<thead>
<tr>
<th>Data Type</th>
<th>Example</th>
</tr>
</thead>
<tbody>
<tr>
<td>Number</td>
<td>$x = -33.657;$</td>
</tr>
<tr>
<td>int</td>
<td>$y = -20;$</td>
</tr>
<tr>
<td>uint</td>
<td>$z = 3;$</td>
</tr>
<tr>
<td>String</td>
<td>$name = “Facebook”;</td>
</tr>
<tr>
<td>Boolean</td>
<td>$over = true;$</td>
</tr>
</tbody>
</table>

After a variable has been declared, you can change its value with an assignment statement. However, you must assign the correct type of value for the variable.
You can also define a memory space (like you would define a variable) to hold a constant value

- This value cannot be changed/re-assigned later in a program

Example:

```javascript
const PI : Number = 3.1415;
trace( PI );  // 3.1415 will be displayed
...
PI = 3.8;     // ERROR
```
Expressions and Statements

- An *expression* is a combination of values, variables, and operators that can be evaluated to obtain a value.

  **Examples:**
  
  5 + 3 * 2
  
  x / 2

- A *statement* is a command to carry out an action.

  **Examples:**
  
  x = y + 2      // Action: assign a variable a value
  
  trace( x )     // Action: display value
Variable Assignment Shortcuts

```javascript
var num: Number = 2;

num ++; // same as num = num + 1;
num --; // same as num = num - 1;
num += 3; // same as num = num + 3;
num -= 3; // same as num = num - 3;
num *= 3; // same as num = num * 3;
num /= 3; // same as num = num / 3;
```
Programming Style

- A useful program must be written so that it may be easily understood and modified, as well as be correct and efficient.

- This means for example:
  - Use meaningful names for variables, etc.
  - Use indentation and spacing
  - Use comments to say what code is doing

- A program written with good programming style can be read and interpreted easily by another programmer
Style: Meaningful Variable Names

Good names:
studentNumber
studentNum
studentId
idNumber

Less good names:
studentnumber
student
id
stdNum

Bad names:
s
num
studentIdentificationNumber

Why are the good names better?
Style Conventions: Variables

- Style conventions are used for names in programs, to easily recognize what they represent.
  - The convention for variables is for the first word in the name to start with a lower-case letter
  - Subsequent words in the name start with an upper-case letter
  - (no spaces)

Examples:

```javascript
var score : Number = 5;
var playerName : String = "Billy Bob";
```
Style Conventions

- The convention for constants is for the name to be all capitals
- Words in the same name are separated by an underscore ( _ )

Examples:

```cpp
const PI : Number = 3.1415;
const BULLS_EYE : int = 50;
```
Example: Consider an app that will let a user type in a Celsius temperature and will convert it to Fahrenheit.

Text Fields required:
- *Input Text Field* for input of Celsius temperature
- *Dynamic Text Field* for output of Fahrenheit temperature

Variables required:
- var celsius: Number;
- var fahrenheit: Number;
Reading Input from a Text Field

- Any text read from a text field is of type `String`, even when the user types digits.
- To obtain a String’s numeric value, convert the String to a number.

  For instance, if the user types a Celsius temperature into a text field called `InputTemp`, the following stores its numeric value in the variable `celsius`:

  ```javascript
  var celsius = Number(InputTemp.text);
  ```
Using Methods from the Flash Library

- Remember:
  - To apply a method to an object, specify:
    `<object name>..<method name>(<parameters>)`

- Example:
  ```plaintext
  smallBird.gotoAndPlay(15);
  ```
When you write a number to a Text Field, Flash converts it to a String for you.

You can limit the number of digits displayed after the decimal point with the `toFixed` method.

For instance, you could display the variable called `fahrenheit` in a text field called `OutputTemp` with only two digits after the decimal place with:

```javascript
OutputTemp.text = fahrenheit.toFixed(2);
```
Class Methods

- Not all methods are applied to an object
  - They’re just a useful group of methods, for instance those in the Math class. To use these methods, specify:
    <class name>.<method name>(<parameters>)

- Example:
  ```javascript
  var x: Number = 2.44;
  var y: Number = 3.7
  x = Math.round( x );  // x is now 2
  y = Math.round( y );  // y is now 4
  ```
## More Math Class Methods

<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>abs(val)</td>
<td>Returns absolute value of val</td>
</tr>
<tr>
<td>pow(val1, val2)</td>
<td>Returns val1 to the power val2</td>
</tr>
<tr>
<td>max(val1, val2, …)</td>
<td>Returns highest val of val1, val2, …</td>
</tr>
<tr>
<td>min(val1, val2, …)</td>
<td>Returns lowest val of val1, val2, …</td>
</tr>
<tr>
<td>sqrt(val)</td>
<td>Returns square root of val</td>
</tr>
<tr>
<td>random()</td>
<td>Returns a random number between 0 and .999999999999999999</td>
</tr>
</tbody>
</table>
Random Numbers

- **Purpose:** To simulate randomness in a program (for instance in a game)
- The `random()` method in the Math class returns a random value between 0 and \(0.99999999999999\)

Example:

```javascript
var rand: Number = Math.random();
trace("rand = " + rand);
```
Random Numbers

- To obtain a random integer between 1 and 6, multiply the result by 5 and then round the result, using the round method, also in the Math class, and add 1.

Example:

```javascript
var die1 = Math.round( 5 * Math.random() ) + 1;
var die2 = Math.round( 5 * Math.random() ) + 1;
trace( "Die 1 = " + die1 );
trace( "Die 2 = " + die2 );
```
Practice with the Math Class

- Download both MathClass.fla and MathClass.as files from the website (under this week’s exercises)
Custom Properties for MovieClips

- Custom properties can be added to MovieClip instances.

- For example, all MovieClips have a height, but they don’t have a speed.
  - We could add a speed property to a MovieClip instance called `ship1` and set its value with the following line in the constructor function of the .as file:
    ```javascript
    ship1.speed = 5;
    ```
Custom Properties for MovieClips

- Once we have created a custom property in the constructor function, we could use it anywhere later in our script.

- For instance, each time a button is pressed we could move ship1 to the right by the amount specified by the speed property if we added the following line to the button function:

  ```
  ship1.x = ship1.x + ship1.speed;
  ```
Test 1 Material Ends Here
CISC 110 Test 1

Info and Practice
CISC110 Test 1 Info

- **When:** Tuesday October 6, 6:30-7:30pm (next week!)
- **Where:** Here
- **What:**
  - Tracing code with variables, numeric expressions, assignment statements, trace statements
  - Writing code with variables, numeric expressions, assignment statements, trace statements
  - Use of methods: For example, Number, String, Math.random, Math.round, Math.min, Math.max, Math.sqrt, Math.abs, Math.pow
- No text fields, buttons, or movieclips
- Study custom notes, lecture slides, exercises, labs, assignments, practice questions online
Practice questions: General Instructions

- (These are the same questions found online)

- Try to answer these questions by paper and pen – like you would in a real test situation. You may choose to verify your answers after in Flash/ActionScript

- There may be more than 1 way to answer a question!
**General Instructions**

- For all of the following questions, assume that the input values have already been read from input text boxes and stored in variables. You will be writing small pieces of code that you can imagine will be placed within either the constructor function or a button-handler function in an .as file as part of a larger project.

- Store the results in variables and display the results in trace statements. Include a description of what each result variable is in your trace statement (e.g., label your output). You can imagine that the results might also be displayed in output text fields, but you don’t need to write that code.
Write code for a Temperature Converter App to convert a Celsius temperature to Fahrenheit. Assume the input temperature has been read from an input text field and has been stored in a Number variable called `celsiusTemp`. The formula for converting from Celsius to Fahrenheit is:

\[ °F = ( °C \times \frac{9}{5} ) + 32 \]

For instance, 10 degrees Celsius is 50 degrees Fahrenheit.
Write code for a Gas Mileage Calculator App. To calculate gas mileage, a driver can fill up their tank, drive for 100 kilometers, and then fill up the tank again, looking at how many liters of gas they bought to fill up the tank the second time, which is how many liters they used. That is the number of liters of gas per 100 km their vehicle uses.

What if they want to drive less than 100 km to determine their gas mileage? Then they would first divide 100 by the number of kilometers they drove. Then they would multiply that amount by the number of liters of gas they used. That gives the number of liters of gas per 100 km their vehicle uses.
For instance, if they drove 25 km and used 3 litres of gas, the calculation would be \( \frac{100}{25} \times 3 \), which equals 12 liters per 100 km.

Assume the number of km and the number of liters of gas used have been read from input text fields and are stored in two Number variables called `numberOfKm` and `numberOfLiters`. Write code to calculate the number of liters per 100 km.
Question 3

Write code for a Recipe Helper App to do some useful calculations. Assume the number of milliliters for an ingredient has been read from an input text field and has been stored in an int variable called `numberOfML`.

Calculate the number of cups, the number of tablespoons and the number of teaspoons in that number of milliliters.

- There are 250 ml in one cup.
- There 16 tablespoons in one cup.
- There are 3 teaspoons in one tablespoon.
Question 4

Write code that uses Math.random and Math.round to simulate throwing two dice.

To do so, generate two random numbers in the range 1 ... 6 and store them in two integer variables called \textit{die1} and \textit{die2}. 
Question 5

Write code that uses `Math.random` and `Math.round` to simulate drawing a card from a 52-card deck, which has four different suits (hearts, clubs, spades, diamonds) and 13 different values (Ace, 2, 3, ..., 10, Jack, Queen, King).

Generate one random number in the range 1 ... 4 to represent the suit and one random number in the range 1 ... 13 to represent the value, and store them in integer variables called `cardSuit` and `cardValue`.

You may assume that the numbers 1-4 correspond to hearts, clubs, spades, and diamonds respectively. You do not need to assign them!