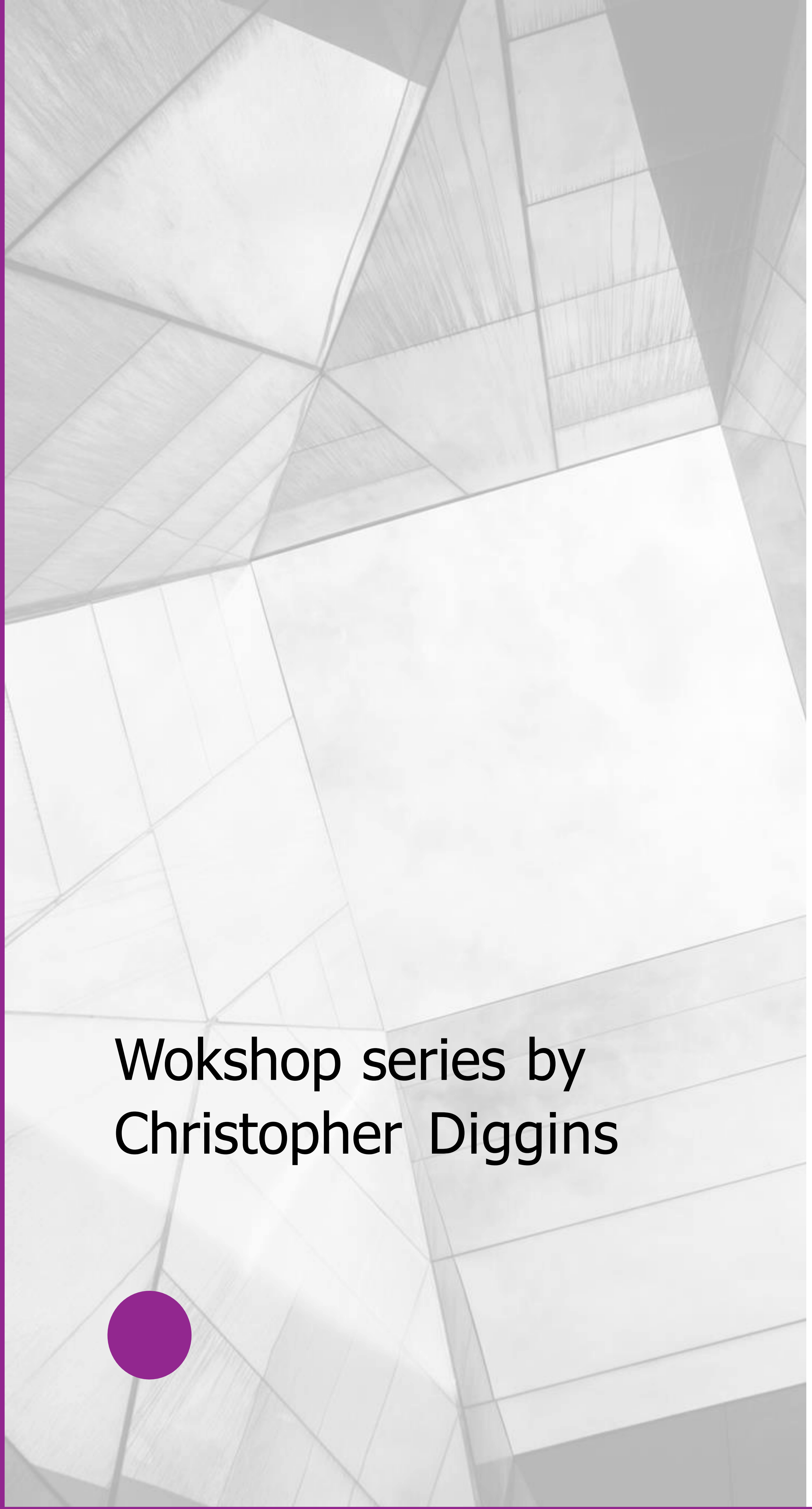


Wokshop series by
Christopher Diggins

Introduction to Coding

Workshop #2
The Elements of Code

Class Objectives

- Key Elements of a Program
- Values and Types
- Control Flow and Statements
- Debugging
- Coding Examples
- Discussion

Key Elements of a Program

- Data types
- Functions
- Statements
- Expressions
- Variables

Data Types

- A data type is a category of data values. As a programmer you can define new types using class, struct, interface, and enum declarations.

- Data type declarations define operations, member functions, and member variables.

```
// Represents a 2D pixel coordinate
public class Coordinate
{
    // constructor
    public Coordinate(int x, int y)
    {
        X = x;
        Y = y;
    }

    // member field
    public readonly int X;
    public readonly int Y;

    // member function
    public Coordinate Offset(int offsetX, int offsetY)
    {
        return new Coordinate(X + offsetX, Y + offsetY);
    }
}
```

Functions

- A function in programming is a code block that has zero or more input values (parameters), and an optional output.

```
int Quadratic(int a, int b, int c, int x)
{
    return (a * x * x) + (b * x) + c;
}
```

- Unlike functions in mathematics, they don't have to have a return value and may cause side-effects to happen

```
void WriteHello()
{
    Console.WriteLine("Hello");
}
```

Statements

- A statement is a unit of code that conceptually represents a single action. Statements may be compound, consisting of embedded statements, or are simple and terminated with the ';' character.
- Some common examples of simple statements include:

• Return statement	- return (a * x * x) + (b * x) + c;
• Variable declaration	- var a = 42;
• Assignment statement	- x = Quadratic(1,2,3,4);
• Function call statement	- Debug.Log("hello");

Expressions

An expression is a sequence of symbols in code that can be evaluated at run-time to produce a value. Some examples:

• a literal value	- 42, 3.14, true
• a variable	- x, MyLongVariableName
• a function call	- Quadratic(0, 5, 1, x)
• an operation	- x + 1, 3 <= x
• a parenthesized group	- (x + 2) * 3
• a \`new\` operation	- new Coordinate(3,4)

Common uses of expressions are:

- operation input (operand)
- function input (argument)
- assigned to a variable
- return value of a function
- conditions for loop or branch statements

Literal Expressions

A literal expression is a representation of a value directly in code.

- Integer literal - `-3`, `42`, `0xFF`
- Floating point literal - `3.14f`, `2.13e+45`
- String literal - `“hello”`
- Boolean literal - `true` and `false`
- Character literal - `'a'`, `'\n'`, `'\u263A'`

Values

- A value is a piece of data such as a number or Boolean or object.
- Values are created by a computer program by evaluating expressions.
- Every value has a type.
- Values never change, but variables might change which value they refer to.
- Values are sometimes, but not always, stored in memory.
- Pretend they are, it makes it easier.

Types

A type is a category of values. C# comes with a few types built in (primitives).

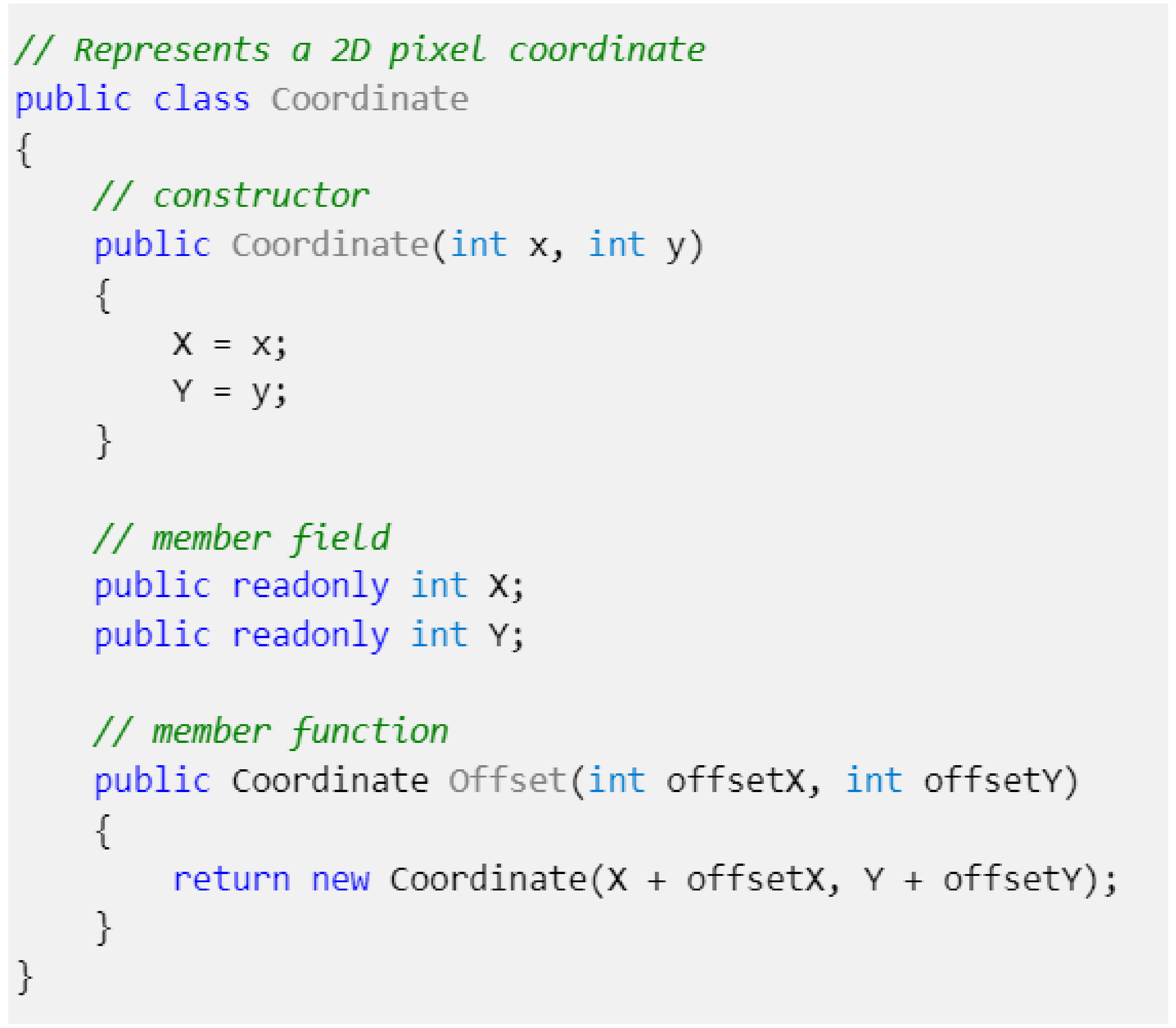
- **int** – a whole number (integer) between approx. -2 billion and +2 billion
- **char** – a text character, such as 'a' or '9' or '/n'
- **string** – a sequence of zero or more text characters.
- **bool** – a Boolean logic value representing true or false
- **double** – a numerical value with a decimal point (e.g., 3.141)
- **byte** – an unsigned whole number between 0 and 255.

User Defined Types

- New types can be defined using class, struct, interface, and enum declarations.
- Types can be placed in a library to be reused.
- Types may contain:
  - Member variable (Fields)
  - Member functions (Method)
  - Operations
  - Constructors
- Types may have per-instance data, or static shared data.
- Types can inherit behavior or data from other types

Class

- The most common kind of user defined type is a class.
- A class is a set of data elements (fields) that can be treated as a single entity along with functions and operations for this type.
- Instances of a class are called objects.
- Class instances are created using the “new” keywords
- Class instances are initialized using the constructor special method.



```
// Represents a 2D pixel coordinate
public class Coordinate
{
    // constructor
    public Coordinate(int x, int y)
    {
        X = x;
        Y = y;
    }

    // member field
    public readonly int X;
    public readonly int Y;

    // member function
    public Coordinate Offset(int offsetX, int offsetY)
    {
        return new Coordinate(X + offsetX, Y + offsetY);
    }
}
```

Control Flow

The order in which the statements, functions, and expressions are executed or evaluated at run-time by a thread of execution.

A program has a single primary line of execution, called a thread. The thread executes statements one after another in a predictable order.

Some statements affect control flow by choosing which statement is executed next, or the number of times a statement is executed.

Common Statement Types

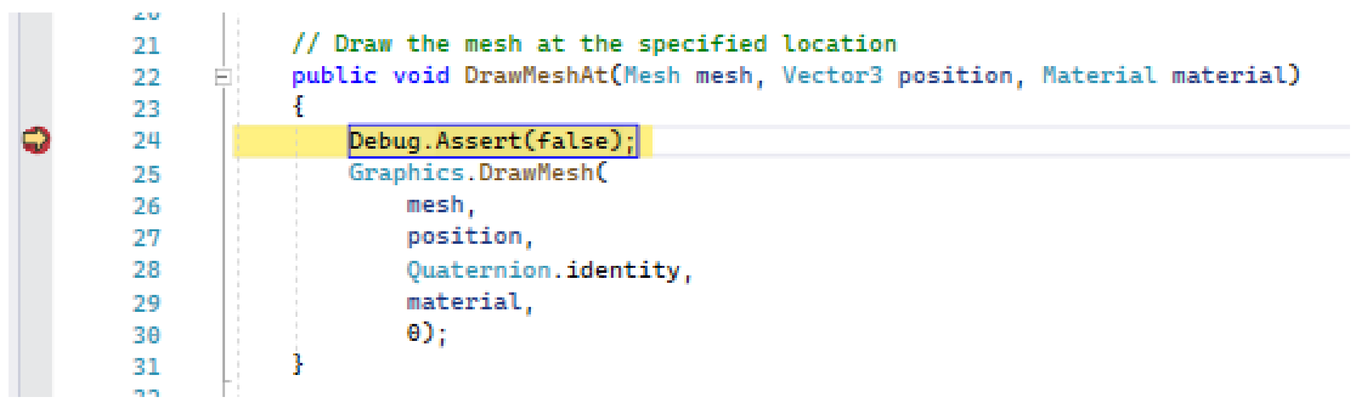
- **Branching statements** - branch statements affect control flow by determining what statement is executed next
- **Loop Statements** – loops affect control flow by executing an embedded statement, repeatedly while a condition evaluates to true.
- **Variable declarations** - declares a new variable and optionally initializes it
- **Expression statements** – either assign a value to a variable or execute a function
- **Block statements** - a set of zero more statements delimited by curly braces {} and that creates a new variable declaration space

Debugging

Bugs are errors, or unexpected behaviors within code. Debugging is the process of trying to figure out how a computer program working and why it is doing what it is doing.

Some basic tools for debugging are:

1. Trace Statements
2. Assertion
3. Breakpoints
4. Watches



```
// Draw the mesh at the specified location
public void DrawMeshAt(Mesh mesh, Vector3 position, Material material)
{
    Debug.Assert(false);
    Graphics.DrawMesh(
        mesh,
        position,
        Quaternion.identity,
        material,
        0);
}
```

Debugger

- A debugger is a computer program that can attach itself to a running process, pause it, step through it line by line, and allow you to view the values associated with variables.

- Debuggers often provide a console window which displays the result from trace statements.

- When running a computer program from within the development environment usually the debugger is already attached.

Debug and Release Mode

- Computer programs are often compiled in one of two modes: debug and release.
- In debug mode a symbol file (.pdb) is generated which contains information that allow the binary executable to be linked to source files.
- Release mode builds are usually optimized, certain instructions (like assertions and trace statements) are removed from the executable, and a PDB is not generated.
- Another name for a “debug” build is a “development” build.

Tracing

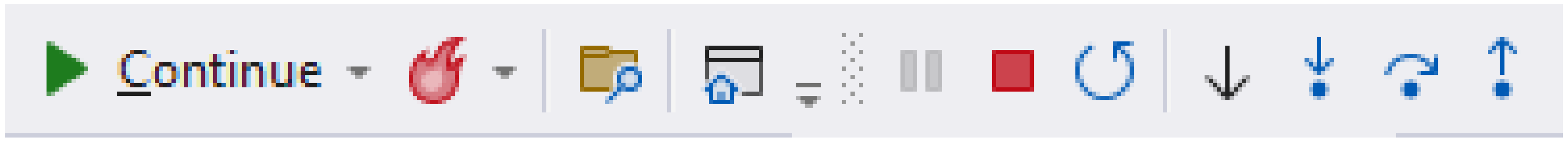
- A function call, which outputs information to a console window, is called a trace statement.
- One example is the Debug.WriteLine() function which is part of the System library.
- Debug.WriteLine() outputs text to the Visual Studio output window.
- When using Unity you can use Debug.Log() which outputs to the Unity console.

Assertions

- An assertion statement is a function call to which takes a conditional expression that is expected to be true.
- If the condition evaluates to false, and a debugger is attached the program will be paused.
- All assert calls will be conditionally included only in a development/debug builds
- Assert statements are great ways to document and test your assumptions about code.

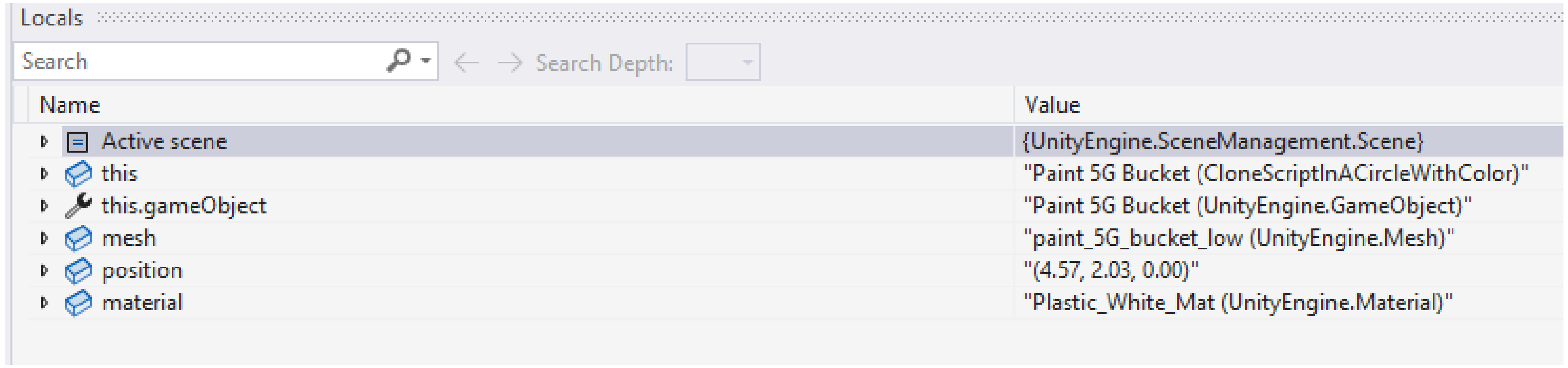
Breakpoints

- A breakpoint is a marker assigned to an instruction within a computer program that triggers the debugger to pause.
- In C# you can also use the Debugger.Break() to programmatically pause any attached debugger.
- If no debugger is attached, you can also use the Debugger.Launch() function to launch and attach a debugger.
- Once you hit a breakpoint, you can step through code one line at a time.



Watches

- A watch is a window that shows the value associated with a variable or expression.
- We can view all local variables, or specific expressions.
- For more information on using watches see this article.



Name	Value
Active scene	{UnityEngine.SceneManagement.Scene}
this	"Paint 5G Bucket (CloneScriptInACircleWithColor)"
this.gameObject	"Paint 5G Bucket (UnityEngine.GameObject)"
mesh	"paint_5G_bucket_low (UnityEngine.Mesh)"
position	"(4.57, 2.03, 0.00)"
material	"Plastic_White_Mat (UnityEngine.Material)"

Debugging Advice

- Computers and compilers rarely make mistakes.
- Virtually of the time, problems arise from an incomplete understanding of what we have asked the computer to do.
- When you are stuck, the problem often boils down to unidentified and incorrect assumptions.
- You have assumed something about how either the compiler or program works, and you are not aware of what assumptions you have made.
- Write out your assumptions as assertion statements or comments

Programming Advice

- The most important thing is the ability to understand the code, and what is being asked of the computer in the context of the language.
- Program defensively.
- Minimize the chance that something can go wrong.
- Minimize the number of assumptions or requirements for using your code.
- Use assertions to test and document your assumptions.
- Write lots of small functions.