Games Technologies

Introduction to C#

Vassilis Markos, Mediterranean College

Week 01

Contents

Key C# Constructs

Hello, World!

```
using System;

namespace School {
  class Program {
    static void Main() {
      Console.WriteLine("Hello, World!");
    }
  }
}
```

The Anatomy of a C# Program

- C# tends to be more verbose than Python in most cases.
- To begin with, everything should be wrapped in `namespace`s, i.e., collections of entities that are conceptually grouped together.
- Then, almost everything lives within a class, much like Java and other object-oriented languages.
- Finally, each project should have a single **main entry point**, which is a function called `Main()` which is typically typed as `static` and `void` (more on that soon...).

Static Typing

```
using System;

namespace Test {
    class TestClass {
        static void Main() {
            // Python: x = 5; x = "High" (Valid)
            // C#:
            int score = 100;
            string grade = "A";
            var autoType = 10.5; // Compiler 'guesses' double, but it's fixed!
            Console.WriteLine("Score: {0}\nGrade: {1}\nAutoType: {2}", score, grade, autoType);
        }
    }
}
```

Static Typing

- In Python, variables are really flexible, being capable of pointing to virtually any language construct.
- In C#, things are a bit more strict, inheriting some C-style semantics.
- Variable types have to be declared explicitly upon variable declaration, determining the size in memory a variable occupies and some other things under the hood.
- One can use the wildcard `var` to let the compiler guess, which, in general, is not a good idea, and should be avoided as much as possible.
    - However, it is quite convenient to have a third party badly documented API endpoint cast this way...

Flow Control

```
using System;

namespace Test {
    class TestClass {
        static void Main() {
            Console.WriteLine("Please, enter an integer: ");
            int n = Int32.Parse(Console.ReadLine());
            string response; // Just a declaration, no assignment
            if (n % 2 == 1) {
                response = "What were the odds?";
            } else if (n % 4 == 2) {
                response = "Four of the best, almost...";
            } else {
                response = "Tough luck!";
            }
            Console.WriteLine("n = {0} means: '{1}'", n, response);
        }
    }
}
```

Flow Control

- Besides, syntax, the same common flow control constructs can be used in C#.
- **Beware:** `else if`, not `elif`!
- Also, deriving from C-style syntax, the logic operators are:
  - `and`: `&&`
  - `or`: `||`
  - `not`: `!`

While Loops

```
using System;

namespace Test {
    class TestClass {
        static void Main() {
            Console.WriteLine("Please, enter an integer: ");
            int n = Int32.Parse(Console.ReadLine());
            string response;
            int d = 2;
            if (n < 2) // You can omit {, } for single line
    blocks
                response = "not";
            else {
                while (n % d != 0) {
                    d++;
                }
            }
            // The so-called ternary operator: condition ?
    true-case : false-case
            // Much like Python's inline if statement.
            response = d == n ? "" : " not";
            Console.WriteLine("{0} is{1} prime!", n, response)
    ;
        }
    }
}
```

For Loops

```
using System;

namespace Test {
    class TestClass {
        static void Main() {
            Console.WriteLine("Please, enter an integer: ");
            int n = Int32.Parse(Console.ReadLine()), a; // You
    can declare multiple variables in the same line
            for (int i = 0; i < 11; i++) {
                a = i * n;
                Console.WriteLine("{0} x {1} = {2}", i, n, a);
            }
        }
    }
}
```

Functions

```
using System;

namespace Test {
    class TestClass {
        static void Main() {
            Console.WriteLine("Please, enter an integer: ");
            int n = Int32.Parse(Console.ReadLine());
            int d = DivSum(n);
            Console.WriteLine("Divisor sum of {0} = {1}.", n,
    d);
        }

        static int DivSum(int n) {
            int s = 0;
            for (int i = 1; i <= n; i++) {
                if (n % i == 0)
                    s += i;
            }
            return s;
        }
    }
}
```

Function Signatures

- In contrast to Python, a function is not identified solely by its name.
- In C#, each function has its unique function signature, which comprises of:
  - its name;
  - its number and type(s) of arguments;
  - its return type.
- So, you can define two functions with the same name, but at least one of the above attributes being different.
- This allow for the so-called **function overloading** (more on that soon).

Common Data Structures

```
using System;
using System.Collections.Generic;

namespace Test {
    class TestClass {
        static void Main() {
            List<int> numbers = new List<int> { 1, 2, 3};
            numbers.Add(5);

            Dictionary<string, int> ages = new Dictionary<
    string, int>();
            ages.Add("Alice", 65);
            ages.Add("Bob", 56);
        }
    }
}
```

Fun Time!

In–class Exercise #001

Make all transcriptions from Python to C# found in:

`lab/transcriptions.pdf`

Use any only resources you might find useful.

Homework

Complete any incomplete lab exercises and then proceed to complete any missing parts of the game lab discussed in class today.

Any Questions?

Do not forget to fill in the questionnaire shown right!

https://forms.gle/dKSrmE1VRVWqxBGZA