COMP120: Creative Computing

1: Tinkering in C#

FALMOUTH UNIVERSITY

Learning Outcomes

- **Outline** the role and basic functions of the IDE
- **Interpret** some basic C# code in Visual Studio
- **Apply** pair programming practices to solve a simple text concatenation problem
- **Explain how** pictures are digitised into raster images by a computer system

Integrated Development Environment (IDE)

Using an IDE

- You *could* just write code in Notepad, but...
- An **Integrated Development Environment (IDE)** is an application providing several useful features for programmers, including:
  - A “run” button
  - Management of multi-file projects
  - Syntax highlighting
  - Autocompletion
  - Navigation
  - Language and API documentation
  - Debugging
  - Profiling
  - Version control

Setting up your own PC

- Programming Language - **C# 8.0** (C sharp)
  `https://docs.microsoft.com/en-us/dotnet/csharp`
- **Visual Studio 9**
  - We use Visual Studio as principle IDE for media computation and game development
  - But you can also use alternative code editors like Sublime Text and Visual Studio Code to write C#
  - Install on your PC here:

  `https://visualstudio.microsoft.com/downloads`

Setting up your own PC

- Install Visual Studio (VS)
  - Register with your `falmouth.ac.uk` email address to obtain VS Professional Edition for free
  - Or, use the free version entitled 'Community Edition'
  - Runs on Windows & Mac

Getting started with Visual Studio

- Create a new project (from the start-up wizard or from the File menu).
- Then choose **"Other $\rightarrow$ Console Project"**
- Create a name for your first project.
- Write some code!

Basic C# programs

Your first C# program

```
using System;

namespace Test
{
    class MainClass
    {
        public static void Main(string[] args)
        {
            Console.WriteLine("Hello World!");
        }
    }
}
```

FALMOUTH UNIVERSITY

C# Terminology

- **Using** The using directive creates an alias for a namespace or import types defined in other namespaces.
- **nameSpace** A namespace is designed to keep one set of names separate from another. Consequently class names declared in one namespace do not conflict with the same class names declared in another.
- **Class** A class defines the kinds of data and the functionality objects will have. A class enables you to create your custom types by grouping variables of other types, methods, and events.
- **public static void Main** It is the first method which gets invoked whenever an application started and it is present in every C# executable file.

Your second C# program

```
Console.WriteLine("This is a very long line of code which
had to be split to fit on the slide, but you should type
it as a single line.")
Console.WriteLine("This is the second line of code.")
```

Assigning to variables

```
int a = 10;
Console.Writeline(a);
```

Variable	Value
a	

Remember!

- A program is a **sequence of instructions**
- The C# interpreter executes the **first line** of your program, then the **second line**, and so on
- When it reaches the end of the file, it **stops**

Socrative - FALCOMPMIKE

Login to Socrative!

`https://b.socrative.com/login/student/`

FALMOUTH UNIVERSITY

Reassigning variables (1)

```
int a = 10;
int b = 20;
b = a;
Console.WriteLine(a);
Console.WriteLine(b);
```

Variable	Value
a	
b	

Reassigning variables (2)

```
int a = 10;
int b = 20;
a = b;
Console.WriteLine(a);
Console.WriteLine(b);
```

Variable	Value
a	
b	

FALMOUTH UNIVERSITY

Reassigning variables (3)

```
int big = 10;
int small = 20;
big = small;
Console.WriteLine(big);
Console.WriteLine(small);
```

Variable	Value
big	
small	

FALMOUTH UNIVERSITY

Reassigning variables (4)

```
int a = 10;
int b = 20;
a = b;
b = a;
Console.WriteLine(a);
Console.WriteLine(b);
```

Variable	Value
a	
b	

Reassigning variables (5)

```
int a = 10;
int b = 20;
int c = 30;

a = b;
b = c;

Console.WriteLine(a);
Console.WriteLine(b);
Console.WriteLine(c);
```

Variable	Value
a	
b	
c	

FALMOUTH UNIVERSITY

Reading Input

```
Console.WriteLine("Enter your name:");
name = Console.ReadLine();

Console.WriteLine("Enter your age:");
age = Int16.Parse(Console.ReadLine());

Console.WriteLine($"Hello {name}");
Console.WriteLine($"On your next birthday, you will be
{age+1} years old");
```

- `Console.ReadLine()` reads a **string** (a sequence of characters—text) from the command line
- `Int16.Parse(...)` parses(converts) a **string** into an **integer** (a number)

FALMOUTH UNIVERSITY

Conditionals (1)

```
int a = Int16.Parse(Console
.ReadLine());

int b = 30;

if (a < 15) {
    b = a;
}

Console.WriteLine(a);
Console.WriteLine(b);
```

Variable	Value
a	
b	

Indentation

- Like many other programming languages, **indentation is not essential but useful** in C#
- C# uses indentation to denote the **block of code** inside a conditional, loop, function etc.
- Microsoft recommends **4 spaces** for indentation
  - Some programmers use a tab character
  - **Never** mix tabs and spaces in the same file!

`https://docs.microsoft.com/en-us/dotnet/csharp/programming-guide/inside-a-program/coding-conventions`

FALMOUTH
UNIVERSITY

Conditionals (2)

```
int a = Int16.Parse(Console
.ReadLine());

int b = 0;

if (a < 20){
    b = a + 1;
} else if (a == 20) {
    b = a * 2;
} else {
    a = 20;
    b = 20;
}

Console.WriteLine(a);
Console.WriteLine(b);
```

Variable	Value
a	
b	

Conditionals

An `if` statement can have:

- **Zero or more** `else if` clauses
- **An optional** `else` clause

In that order!

FALMOUTH UNIVERSITY

Mathematical operators

- + add
- - subtract
- * multiply
- / divide
- ** power

Order of operations: **BIDMAS**

- Brackets first
- Then indices (powers)
- Then division and multiplication (left to right)
- Then addition and subtraction (left to right)

Comparison operators

- `<` less than
- `<=` less than or equal to
- `>` greater than
- `>=` greater than or equal to
- `==` equal to
- `!=` not equal to

Note the difference between `=` and `==`

- `a = b` means "make `a` be equal to `b`"
- `a == b` means "is `a` equal to `b`?"

For loops and ranges

```
for (int i = 0; i < 5; i++)
{
  Console.WriteLine(i);
}
```

- `for` contains 3 statements: **variable, condition** and **increment**
- Initially the **variable** is set to a value and the **incrementer** increases the value until the **condition** is met
- The `for` loop iterates through the items in a sequence **in order.** As the loop iterates the variable is increased each time: 0, 1, 2, 3, 4
- Note: `i < 5` **does not include** 5 as the condition is met at 4 so the loop stops.

FALMOUTH UNIVERSITY

For loops (1)

```
int a = 0;
int b = 0;

for (int i = 0; i < 5; i++)
{
    a = i;
    b = b + i;
}

Console.WriteLine(a);
Console.WriteLine(b);
```

Variable	Value
a	
b	
i	

For loops (2)

```
int a = 0;
int b = 0;

for (int i = 0; i < 5; i++)
{
    if (i < 3 || i > 7)
    {
        a += i;
    }
    else
    {
        b += i;
    }
}
Console.WriteLine(a);
Console.WriteLine(b);
```

Variable	Value
a	
b	
i	

While loops

The `while` loop keeps executing while the condition is **true**

```
int a = 1;

while (a < 100)
{
   a = a * 2;
}

Console.WriteLine(a);
```

Variable	Value
a	

FALMOUTH UNIVERSITY

Looping forever

```
int a = 1;

while (true) {
    a = a * 2;
    Console.WriteLine(a);
}
```

FALMOUTH UNIVERSITY

Summary

We have seen some basic code constructions in Python

- `Console.WriteLine()` and `Console.ReadLine()` for command-line input and output
- Variable assignment using `=`
- **`if`** statements for choosing whether or not to execute a block of code
- **`for`** loops to execute a block of code a specified number of times
- **`while`** loops to execute a block of code until a condition is no longer true

These are enough to write some simple programs, but you will see several more in coming weeks...

Challenge

- In pairs
- **Implement** the code excerpt
- **Fix** the errors in the code excerpt
- **Modify** the code excerpt to incorporate functions and arguments
- **Post** your solution to the `#comp120` slack channel

You can learn more about functions and arguments at:

`https://docs.python.org/3/tutorial/controlflow.html#defining-functions`

Challenge

The function:

```
public void madlib()
```

Should become:

```
public string madlib(string name, string pet,
string verb, string snack)
```

Challenge

```
public void madlib() {
        string name = "Link";
        string pet = "Spyro";
        string verb = "ate";
        string snack = "doughnuts";
        line1 = "once upon a time," + name + "walked";
        line2 = "with " + pet + ", a trained dragon.";
        line3 = "Suddenly, ' + pet + " announced,";
        line4 = "I really want some " + snack + "!";
        line5 = name + " complained. Where am I going to
        get that?";
        line6 = "Then " + name + "found a wizards wand.";
        line 7 = "With a wave of the wand, ";
        line8 = pet + " got " + snack + ". ";
        line9 = "Perhaps surprisingly, " + pet + " " +
        verb + "  " + snack;
        Console.WriteLine(line1 + line2 + line3 + line4);
        Console.WriteLine(line5 + line6 + line7 + line8
        + line9);
    }
```

Tinkering Graphics

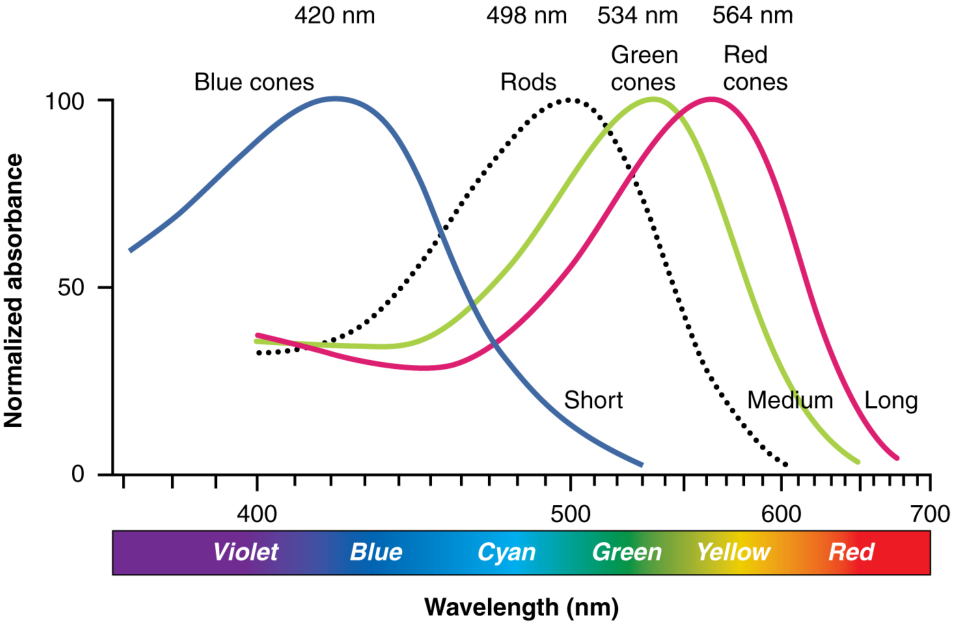
Light Perception

- Colour is continuous:
  - Visible light is in the wavelengths between 370nm and 730nm
  - i.e., 0.00000037 — 0.00000073 meters
- However, we *perceive* light around three particular peaks:
  - Blue peaks around 425nm
  - Green peaks around 550nm
  - Red peaks around 560nm

FALMOUTH UNIVERSITY

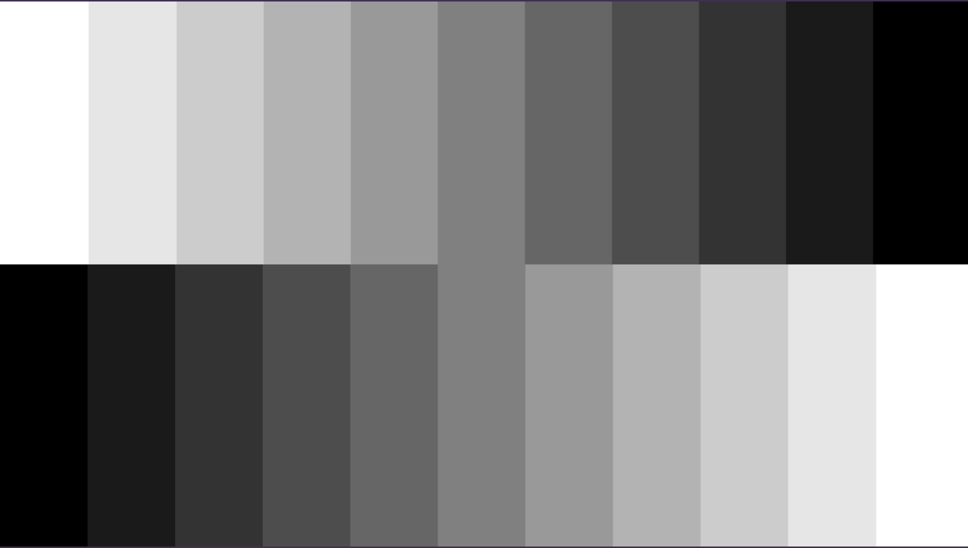
Light Perception

- Our eyes have three types of colour-sensitive photoreceptor cells called `cones' that respond to light wavelengths
- Our perception of colour is based on how much of each kind of sensor is responding
- An implication of this is perception overlap: we see two kinds of `orange' — one that's spectral and one that's combinatorial


420 nm
498 nm
534 nm
564 nm
Blue cones
Rods
Green cones
Red cones
100
50
0
Normalized absorbance
Short
Medium
Long
400
500
600
700
Violet
Blue
Cyan
Green
Yellow
Red
Wavelength (nm)

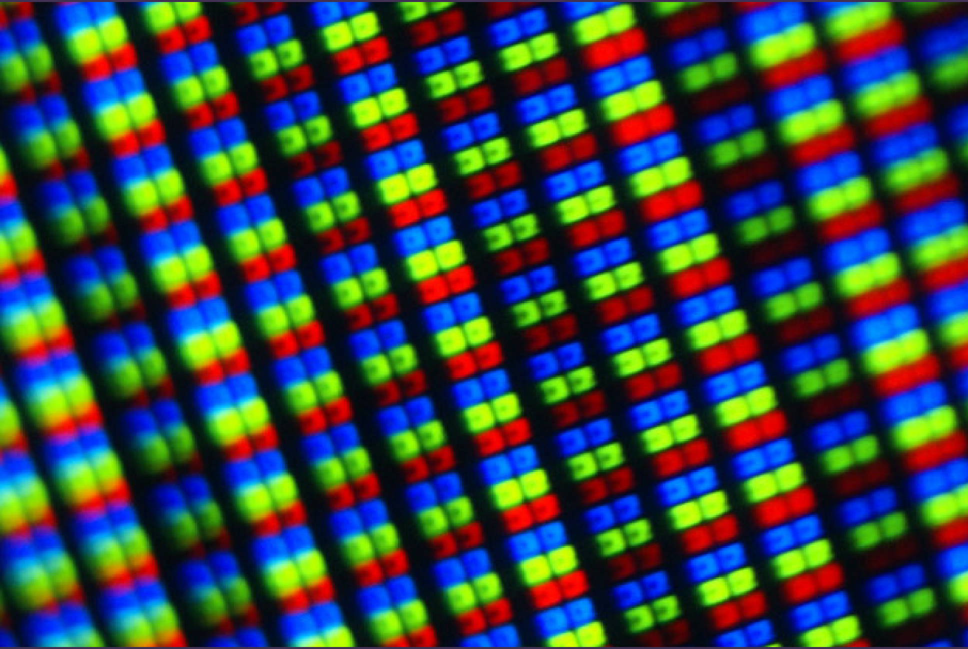
Luminance vs Colour

- Our eyes have another type of photoreceptor cells called `rods' that respond to light intensity
- Our perception, however, is actually luminance: a relativistic contrast of *borders* of things (i.e., motion)
  - Luminance is *not* the amount of light, but our perception of the amount of light
  - Much of our luminance perception is based on comparison to background, not raw values
- An implication of this is perception overlap: we see blue as `darker' than red when the intensity is actually the same



Resolution

- We have a limited number of rods and cones in our eyes
- This means humans perceive vision in a limited resolution — yet, we perceive vision as continuous
- We take advantage of this human characteristic in computer monitors



Pixels

- We digitize pictures into many little dots
- Enough dots and it looks like a continuous whole to our eye
- Each element is referred to as a *pixel*

Pixels

Pixels must have:

- a color
- a position

Pictures and Surfaces

In PyGame, a `Surface` is a *matrix* of pixels

- It is not a continuous line of elements, that is, a one-dimensional *array*
- A picture has two dimensions: width and height
- It's a two-dimensional *array*

Pictures and Surfaces

- (x, y) —or— (horizontal, vertical)
- The origin (0,0) is top-left
- (1,0) = 12
- (0, 2) = 6

Encoding Colour

- Each element in the matrix is a pixel, with the matrix defining its position and the value defining its colour
- Computer memory stores numbers, so colour must be encoded into a number:
  - CMYK = cyan, magenta, yellow, black
  - HSB = hue, saturation, brightness
  - RGBA = red, green, blue, alpha (transparency)
- By default, Visual Studio and C# uses RGBA

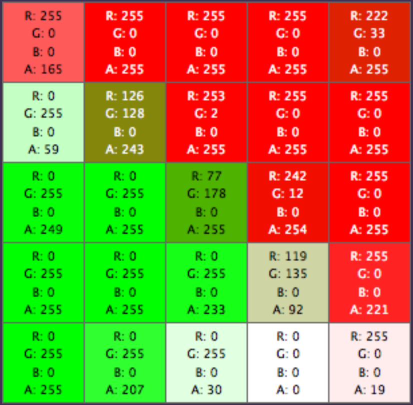
Encoding RGB

- Each component color (red, green, and blue) is encoded as a single byte
- Colors go from
  - If all three components are the same, the colour is in grey-scale
  - `(0,0,0)` is black
  - `(255, 255, 255)` is white

Encoding Bits

Why `255`?

- If we have one bit, we can represent **TWO** patterns:
  - 0
  - 1
- If we have two bits, we can represent **FOUR** patterns:
  - 00
  - 01
  - 10
  - 11
- With n bits, we can have 2^n patterns
- With 8 bits, there will be 256 patterns
- One of these patterns will be 0, so the highest value we can represent with 8 bits is: $2^8 - 1$, or 255



R: 255 G: 0 B: 0 A: 165	R: 255 G: 0 B: 0 A: 255	R: 255 G: 0 B: 0 A: 255	R: 255 G: 0 B: 0 A: 255	R: 222 G: 33 B: 0 A: 255
R: 0 G: 255 B: 0 A: 59	R: 126 G: 128 B: 0 A: 243	R: 253 G: 2 B: 0 A: 255	R: 255 G: 0 B: 0 A: 255	R: 255 G: 0 B: 0 A: 255
R: 0 G: 255 B: 0 A: 249	R: 0 G: 255 B: 0 A: 255	R: 77 G: 178 B: 0 A: 255	R: 242 G: 12 B: 0 A: 254	R: 255 G: 0 B: 0 A: 255
R: 0 G: 255 B: 0 A: 255	R: 0 G: 255 B: 0 A: 255	R: 0 G: 255 B: 0 A: 233	R: 119 G: 135 B: 0 A: 92	R: 255 G: 0 B: 0 A: 221
R: 0 G: 255 B: 0 A: 255	R: 0 G: 255 B: 0 A: 207	R: 0 G: 255 B: 0 A: 30	R: 0 G: 0 B: 0 A: 0	R: 255 G: 0 B: 0 A: 19

Encoding Bits

- RGB uses 24-bit color (i.e., $3 * 8 = 24$)
  - That's 16,777,216 possible colours
  - Our eyes cannot discern many colours beyond this
  - A challenge is display technology: monitors and projectors can`t reliably reproduce 16 million colours
- RGBA uses 32-bit colour
  - No additional colour, but offers support for transparency
  - This transparency channel is called `alpha`
  - The alpha channel also requires 8 bits
- Assuming `1 byte` == `8 bits`
- We can use this information to estimate the size of a bitmap:
  - 320x240x24 = 230,400 bytes
  - 640x480x32 = 1,228,800 bytes
  - 1024x768x32 = 3,145,728 bytes

FALMOUTH UNIVERSITY

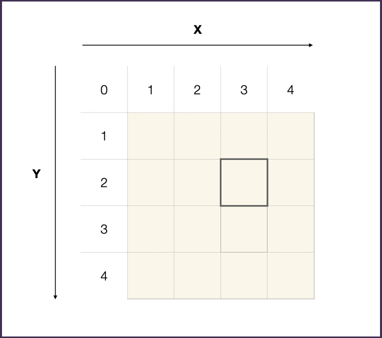
Manipulating Bitmap Pixels

- Images are controlled and manipulated using the `Bitmap` class in C#
- We can use `GetPixel` and `SetPixel` methods to both find and change pixels.

```
myImage.GetPixel(x, y);
myImage.SetPixel(x, y, newColor);
```

- Both methods use cartesian coordinates (x and y) to define the position of a specific pixel

Manipulating Bitmap Pixels



```
myImage.GetPixel(3, 2);
```

We can use the method to discover the ARGB values at the above position