Functions

# Exercise 1. Hello Function!

**Summary.** Write a function that, when called, outputs the message “Hello Function!” to console.

1. Create a new Unity project
2. Creates a new script called **HelloScript**
3. Within the script, create a function called **HelloFunction**.  
   It should take no parameters, and return nothing:

|  |
| --- |
| **void HelloFunction ()** {  … your code here … } |

1. Inside the body of the function, use the Debug.Log to print the desired message to the console
2. Invoke the function from the **Start** method
3. Make sure the script compiles, then attach it to the **Main camera**
4. Test your project to make sure it works

# Exercise 2. Parameters

**Summary.** Modify **HelloFunction** to print a custom message.

1. Modify the **HelloFunction** created so that it takes a single parameter.  
   It should be of type **string**, meaning that it receives a string as an input
2. Modify the body of **HelloFunction** to print your “Hello Function!” followed by the string that was received. In C# you can concatenate two strings by using the symbol **+**.
3. In the Start Method, change the call to the function accordingly, making sure that you get “Hello Function! Custom message!” if you invoke it like this:

|  |
| --- |
| HelloFunction(“Custom message!”); |

# 

# Exercise 3. Move position

**Summary.** In Unity, you cannot change the x, y or z component of your position directly. To move 10 meters right on the x axis, you **cannot** do:

|  |
| --- |
| transform.position.x += 10; // wrong |

Instead, you are forced to do this:

|  |
| --- |
| Vector3 pos = transform.position; // Copy the position  pos.x += 10; // Changes the copied value  transform.position = pos; // Assign the new value |

which is much more verbose and less clear.

Create a method **MovePosition** that takes three values, x, y and z, and moves the position of the object accordingly.

1. Creates a new Script for your new function
2. Create a method called **MovePosition** that takes three parameters of type **float**.  
   **float** stands for “floating point”, and it’s the way C# represents non-integer numbers.
3. Inside the function, copy the green code shown above, with the necessary corrections.

To test your function, use it to move your object by 1 meter per second on the right.

1. In the **Update** function, call **MovePosition**
2. Use **Time.deltaTime** to make sure your object moves 1 meter per second, and not 1 meter per frame.

|  |  |  |  |
| --- | --- | --- | --- |
| **Extra 1: default parameters.** If a function has three parameters, you have to call it providing three parameters. In the case of **MovePosition**, for instance:   |  | | --- | | MovePosition(1, 0, 0); // Move 1 meter right on the x axis |   In C# you can specify a default value for the parameters of your functions, like this:   |  | | --- | | void MyFunction (int a, int b, **int c = 0**) |   When a function is defined in such a way, it can be called in two ways:   |  | | --- | | MyFunction (1, 2, 3); // a=1, b=2, c=3  MyFunction (1, 2); // a=1, b=2, **c=0 (default value)** |   Change **MovePosition** so that **y** and **z** have zero as a default value. |

|  |  |  |
| --- | --- | --- |
| **Extra 3. Function overload.** In C# you can have multiple functions with the same name. This is possible if they take sufficiently different parameters. For instance:   |  | | --- | | void MyFunction (float x, float y) { Debug.Log(“A”); }  void MyFunction (Vector2 v) { Debug.Log(“B”); } |   Depending on the parameters that you have passed, C# will know which one to invoke:   |  | | --- | | MyFunction(1,2); // Prints “A”  MyFunction(new Vector2(1,2)); // Prints “B” |   Creates another version of **MovePosition** that takes a **Vector3** as input. |

|  |
| --- |
| **Extra 3: set & get.** The reason why you cannot change the position change **position** directly is because it’s not a variable, even if it looks like one. See **Setters and Getters** in C# for more information about this. |

# Exercise 4. GetInput

**Summary.** The section “*Moving an object with the keyboard*” of the cheat sheet shows how to use **Input.GetAxis** to get inputs from keyboards and controllers. Creates a function that wraps that piece of logic and returns a **Vector2** with the following components:

* x: Input.GetAxis(“Horizontal”)
* y: Input.GetAxis(“Vertical”)

1. Creates a new scene
2. Creates a new script called **KeyboardController** and attach it to a cube
3. Inside, creates a function called **GetInput**.  
   It should take no parameters, and should return a **Vector2** value.
4. Inside the function, initialises the two components of the Vector2 with the values take from **Input.GetAxis**.  
   Remember that to create a new Vector2, you must do:

|  |
| --- |
| Vector2 v = new Vector2(1,2); |

Test your project by moving a cube with the keyboards. Use the previous **MovePosition** to move your cube.

# Exercise 5. Lighter Colours

**Summary.** Colours in Unity are represented with the type **Color**:

* **Color.r**: red channel (0-1)
* **Color.b**: blue channel (0-1)
* **Color.g**: green channel (0-1)
* **Color.a**: alpha channel (0: fully transparent, 1: fully opaque)

Creates a function **Brighter** that takes two inputs: a colour and a float value between 0 (=0%) and 1 (=100%). It returns a lighter version of the colour. For instance:

|  |
| --- |
| Color c1 = new Color(0.5f,1f,0.5f);  Color c2 = Lighter(c1,**0.5f**); |

Returns a shader of **c2** that is **50%** brighter.

To make a color brighter by X%, take its RGB components and multiply each component by **(1f+x/100f)**.

To test it, create two public **Color** variables that can be seen in the inspector. Then, use the **Start** function to change one of them.

|  |
| --- |
| **Extra 1.** The RGB colour space does not correctly captures the concept of “brightness”. The equation presented in the exercise will lead to often unnatural shades. For better results, one should use the HSV colour space.   * Convert a colour from RGB to to the HSV space. * Increase its V value * Convert the colour back to RGB space and return it   Refers to the Scripting API for the type **Color** ( <https://docs.unity3d.com/ScriptReference/Color.html> ) to learn how to convert colours. |

|  |
| --- |
| **Extra 2.** The function **Color.HSVToRGB** does not return a value; it alters its input parameters using the **out** value.  Creates a new version of **Lighter** that uses out of change the colour that has been passed as a parameter, rather than returning a new colour.  Note that:   * **out**: takes a non initialised parameters, and change it * **ref**: takes a parameter which has already a value, and change it |

Solutions

# Exercise 1. Hello Function!

|  |
| --- |
| using UnityEngine;  public class HelloScript : MonoBehaviour  {  void Start ()  {  HelloFunction ();  }  void HelloFunction ()  {  Debug.Log(“Hello Function!”);  }  } |

# Exercise 2. Parameters

|  |
| --- |
| using UnityEngine;  public class HelloScript : MonoBehaviour  {  void Start ()  {  HelloFunction (“Custom message!”);  }  void HelloFunction (string s)  {  Debug.Log(“Hello Function! ” + s);  }  } |

# 

# 

# Exercise 3. Move position

|  |
| --- |
| using UnityEngine;  public class Exercise3 : MonoBehaviour  {  void Update ()  {  MovePosition (1 \* Time.deltaTime, 0, 0);  }  void MovePosition (float x, float y, float z)  {  Vector3 pos = transform.position;  pos.x = x;  pos.y = y;  pos.z = z;  transform.position = pos;  }  } |

|  |
| --- |
| using UnityEngine;  public class Exercise3 : MonoBehaviour  {  void Update ()  {  MovePosition (1 \* Time.deltaTime, 0, 0);  }  // Extra 1  void MovePosition (float x, float y = 0, float z = 0)  {  Vector3 pos = transform.position;  pos.x = x;  pos.y = y;  pos.z = z;  transform.position = pos;  }  // Extra 2  void MovePosition (Vector3 v)  {  MovePosition (v.x, v.y, v.z);  }  } |

# Exercise 4. GetInput

|  |
| --- |
| using UnityEngine;  public class KeyboardController : MonoBehaviour  {  public Vector2 speed = new Vector2(1,2);  void Update ()  {  Vector2 input = GetInput();  MovePosition  ( input.x \* speed.x \* Time.deltaTime,  input.y \* speed.y \* Time.deltaTime,  0  );  }  Vector2 GetInput ()  {  Vector2 v = new Vector2  ( Input.GetAxis(“Horizontal”),  Input.GetAxis(“Vertical”)  );  return v;  }  void MovePosition (float x, float y, float z)  {  Vector3 pos = transform.position;  pos.x = x;  pos.y = y;  pos.z = z;  transform.position = pos;  }  } |

# Exercise 5. Lighter colours

|  |
| --- |
| using UnityEngine;  public class Exercise5 : MonoBehaviour  {  public Color c1;  public Color c2;  [Range(0,1)]  public float b = 0.5f;  void Start ()  {  c2 = Lighter (c1, b);  }  Color Lighter (Color color, float b)  {  Color newColor = new Color  ( color.r \* (1f + b),  color.g \* (1f + b),  color.b \* (1f + b),  color.a  );  return newColor;  }  } |

|  |
| --- |
| using UnityEngine;  public class Exercise5 : MonoBehaviour  {  public Color c1;  public Color c2;  [Range(0,1)]  public float b = 0.5f;  void Start ()  {  c2 = Lighter (c1, b);  }  Color Lighter (Color rgb, float b)  {  float h, s, v;  Color.RGBToHSV(rgb, out h, out s, out v);  Color newColor = Color.HSVToRGB  ( h,  s,  v \* (1f + b)  );  newColor.a = rgb.a;  return newColor;  }  } |

|  |
| --- |
| using UnityEngine;  public class Exercise5 : MonoBehaviour  {  public Color c1;  [Range(0,1)]  public float b = 0.5f;  void Start ()  {  Lighter (ref c1, b);  }  void Lighter (ref Color rgb, float b)  {  float h, s, v;  Color.RGBToHSV(rgb, out h, out s, out v);  Color newColor = Color.HSVToRGB  ( h,  s,  v \* (1f + b)  );  rgb.r = newColor.r;  rgb.g = newColor.g;  rgb.b = newColor.b;  }  } |