

# Programming Fundamentals

## Lecture 04 – Conditional Statements and User Interaction

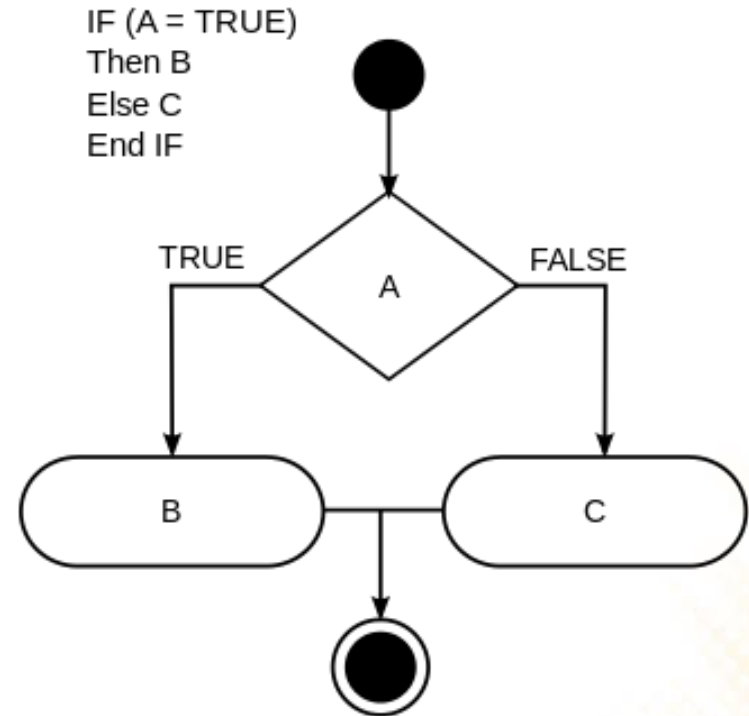
Edirlei Soares de Lima

<edirlei.lima@universidadeeuropeia.pt>



# Conditional Statements

- Conditional statements allow programs to perform different computations or actions depending on whether a boolean condition evaluates to true or false.
- Are used to control the flow of execution and to define logical paths through the code.
- Lua statements: if – elseif – else



# Conditional Statements

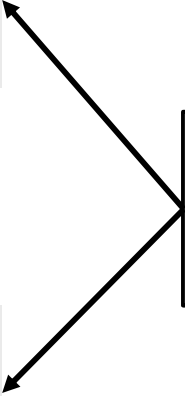
- In Lua, conditional statements are constructed with the *if*:

```
if boolean_condition then
    -- block of code
end
```

- Example:

```
if lives <= 0 then
    io.write("Game Over")
end
```

The lines of code that are in the **block of code** only will be executed if the **boolean condition** is true.



# Conditional Statements

- The **else** statement can also be used to define a block of code to be executed when the boolean condition is not true:

```
if boolean_condition then  
    -- block of code  
else  
    -- block of code  
end
```

## Example:

```
if ammo < 12 then  
    ammo = ammo + 1  
    io.write("Reloaded!")  
else  
    io.write("Ammo is full!")  
end
```

# Conditional Statements

- The **elseif** statement can also be used to create alternatives with conditions:

```
if boolean_condition_1 then
    -- block of code 1
elseif boolean_condition_2 then
    -- block of code 2
elseif boolean_condition_3 then
    -- block of code 3
end
```

If the first condition is *true*, only the first block of code is executed; the other conditions are not even evaluated. **Otherwise, if the second condition is *true***, only the second block of code is executed, and so on.

# Conditional Statements

- **Example:**

```
if enemy_pos_x < player_pos_x then
  -- Go to the right
elseif enemy_pos_x > player_pos_x then
  -- Go to the left
else
  -- Attack!
end
```

# Boolean Conditions

- Boolean conditions are defined with **relational operators**:

## Examples:

X = 10 e Y = 5

Description	Symbol
Equals to	==
Different from	~=
Larger than	>
Smaller than	<
Larger than or equal to	>=
Smaller than or equal to	<=

Description	Symbol
X == Y	False
X ~= Y	True
X > Y	True
X < Y	False
X >= Y	True
X <= Y	False

All the operators are used to compare **two values**, resulting in true or false.

# Boolean Conditions

- Boolean conditions can also be combined with **logical operators**.

Operator	Meaning	Symbol
Conjunction	and	and
Disjunction	or	or
Negation	not	not

## Examples:

Condition	Result
<code>X &gt; 0 and X == Y</code>	False
<code>X &gt; 0 or X == Y</code>	True
<code>not Y &lt; 10</code>	False

X = 10  
Y = 5



# Boolean Conditions

- Example 1 (and):

```
...  
if var1 >= 5.0 and var2 >= 3.0 and var3 >=3.0 and var4 >= 3.0 then  
    -- block of code  
end  
...
```

- Example 2 (or):

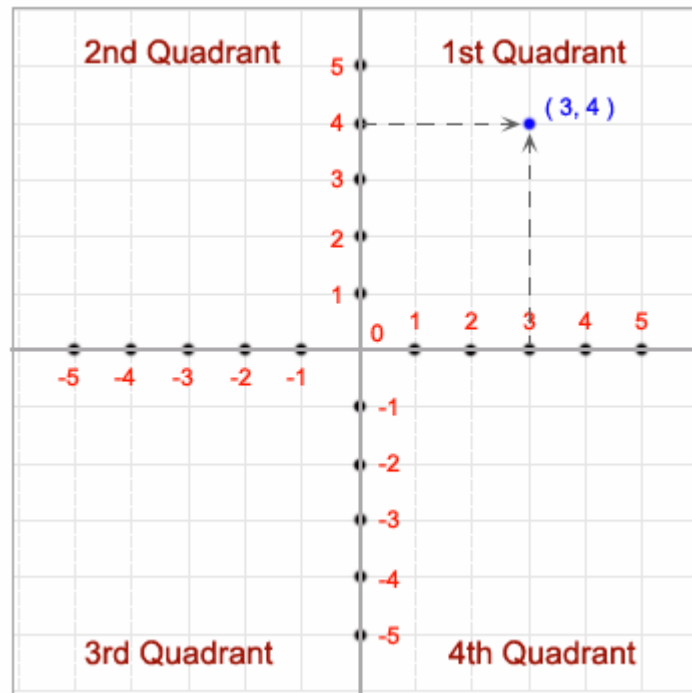
```
...  
if var1 < 5.0 or var2 < 3.0 or var3 < 3.0 or var4 < 3.0 then  
    -- block of code  
end  
...
```

- Example 3 (not):

```
...  
if not (var1 < 5.0 or var2 < 3.0 or var3 < 3.0 or var4 < 3.0) then  
    -- block of code  
end  
...
```

# Conditional Statements – Example

- Write a program to read a coordinate point in a XY coordinate system and determine in which quadrant the coordinate point lies.




# Conditional Statements – Example

```
local coordx, coordy

io.write("Input the X value for the coordinate:")
coordx = tonumber(io.read())
io.write("Input the Y value for the coordinate:")
coordy = tonumber(io.read())

if (coordx > 0) and (coordy > 0) then
    io.write("The coordinate point lies in the First quadrant.\n")
elseif (coordx < 0) and (coordy > 0) then
    io.write("The coordinate point lies in the Second quadrant.\n")
elseif (coordx < 0) and (coordy < 0) then
    io.write("The coordinate point lies in the Third quadrant.\n")
elseif (coordx > 0) and (coordy < 0) then
    io.write("The coordinate point lies in the Fourth quadrant.\n")
elseif (coordx == 0) and (coordy == 0) then
    io.write("The coordinate point lies at the origin.\n")
end
```

# Back to the “Hello World”

- In the last implementation of the “Hello World”, we moved the text through the screen/window.
    - **Problem:** when the text reaches the limit of the screen/window, the text disappears (it keeps moving...)
  - With a conditional statement, we can move the text back when it reaches the limit of screen/window.
  - How can we do that?
- 

# Back to the “Hello World”

```
local px -- position of the text in the x axis

function love.load()
  love.graphics.setColor(0, 0, 0)
  love.graphics.setBackgroundColor(1, 1, 1)
  px = 0
end

function love.update(dt)
  px = px + (100 * dt)
end

function love.draw()
  love.graphics.print("Hello World", px, 300)
end
```

```
local px      -- position of the text in the x axis

function love.load()
    love.graphics.setColor(0, 0, 0)
    love.graphics.setBackgroundColor(1, 1, 1)
    px = 0
end

function love.update(dt)
    px = px + (100 * dt)

    if px > 800 then -- the default width of the window
        px = 0      -- is 800
    end

end

function love.draw()
    love.graphics.print("Hello World", px, 300)
end
```

```
local px      -- position of the text in the x axis
```

```
function love.load()
```

```
    love.graphics.setColor(0, 0, 0)
```

```
    love.graphics.setBackgroundColor(1, 1, 1)
```

```
    px = 0
```

```
end
```

```
function love.update(dt)
```

```
    px = px + (100 * dt)
```

```
    if px > love.graphics.getWidth() then
```

```
        px = 0
```

```
    end
```

```
end
```

```
function love.draw()
```

```
    love.graphics.print("Hello World", px, 300)
```

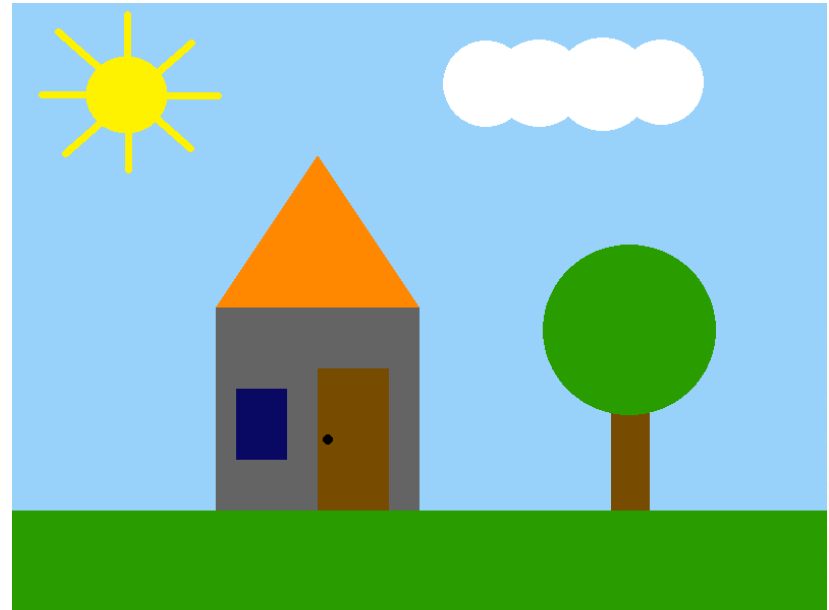
```
end
```

A more general way of  
obtaining the width of the  
window.

# Exercise 1

1) Continue the last exercise and implement an animation to move the cloud (from left to right).

- a) When the cloud disappear on the right side, it should appear on the left side (smoothly).
- b) When the cloud is over the sun, change the color of the background to a darker color.



- Extra Challenge: change the color of the background gradually when the cloud is over the sun.



# Module `love.keyboard`

- The module `love.keyboard` provides an interface to the user's keyboard and contains several functions for user interaction.
- Is possible to check if some key is pressed with the function `love.keyboard.isDown`

```
love.keyboard.isDown(key)
```

- The function returns true if the key (parameter) is pressed.

# Module `love.keyboard`

- A conditional statement is necessary to handle the result of the function.
- **Example:**

```
if love.keyboard.isDown("right") then
  px = px + (100 * dt)
end
```

- List of key codes: <http://www.love2d.org/wiki/KeyConstant>

# Back to the “Hello World”

```
local px      -- position of the text in the x axis

function love.load()
    love.graphics.setColor(0, 0, 0)
    love.graphics.setBackgroundColor(1, 1, 1)
    px = 0
end

function love.update(dt)

    if love.keyboard.isDown("right") then
        px = px + (100 * dt)
    end

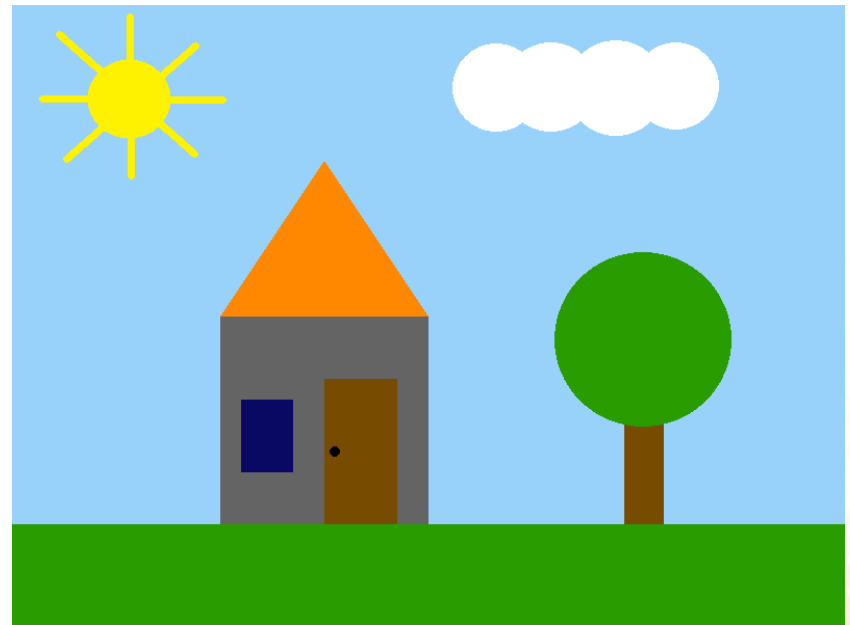
end

function love.draw()
    love.graphics.print("Hello World", px, 300)
end
```

# Exercise 2

2) Continue implement of the last exercise to allow the user to move the sun (in the X and Y axis) using the arrow keys of the keyboard.

- The code that changes the color of the background must still work after moving the sun (considering the new position of the sun).
- If you implemented the challenge of the first exercise (drawing a character with lines), you can move the character instead of moving the sun.



# Module `love.mouse`

- The module `love.mouse` provides an interface to the user's mouse and contains several functions for user interaction with the mouse.
- Is possible to check if a mouse button is pressed with the function `love.mouse.isDown`

```
love.mouse.isDown(button)
```

- The function returns true if the button (parameter) is pressed.

# Module love.mouse

- A conditional statement is necessary to handle the result of the function.
- **Example:**

```
if love.mouse.isDown(2) then
  texto = "Left! :)"
end
```

- 1 is the primary mouse button;
- 2 is the secondary mouse button;
- 3 is the middle button.

# Module `love.mouse`

- The module `love.mouse` also allows access to the mouse position (`love.mouse.getX` and `love.mouse.getY`)


```
love.mouse.getX()
```

```
love.mouse.getY()
```

- The function return the position of the mouse inside the window of the program (X and Y axis).

```
mousex = love.mouse.getX()  
mousey = love.mouse.getY()
```

# Back to the “Hello World”

- Using the mouse functions, we can modify the “Hello World” to allow user to:
    - Move the text with the mouse;
    - Change the content of the text when a mouse button is pressed:
      - Right button: “Right! :)”
      - Left button: “Left! :)”
      - None: “Hello World!”
  - How can we do that?
- 



# Back to the “Hello World”

```
local px      -- position of the text in the x axis
local py      -- position of the text in the y axis
local text = "Hello World!"

function love.update(dt)
    if love.mouse.isDown(2) then
        text = "Left! :)"
    elseif love.mouse.isDown(1) then
        text = "Right! :)"
    else
        text = "Hello World!"
    end
    px = love.mouse.getX()
    py = love.mouse.getY()
end

function love.draw()
    love.graphics.print(text, px, py)
end
```

# Exercise 3

3) Continue implement of the last exercise to allow the user to move the sun (in the X and Y axis) using the mouse.

- The code that changes the color of the background must still work after moving the sun (considering the new position of the sun).

