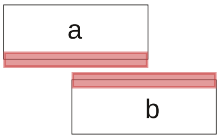
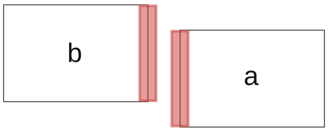
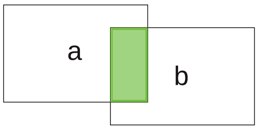
only colliding when all conditions are green: if even one single one is red, the shapes are not colliding.



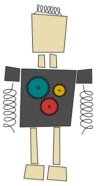
```
a.x < b.x + b.width
b.x < a.x + a.width
a.y < b.y + b.height
b.y < a.y + a.height
```



```
a.x < b.x + b.width
b.x < a.x + a.width
a.y < b.y + b.height
b.y < a.y + a.height
```



```
a.x < b.x + b.width
b.x < a.x + a.width
a.y < b.y + b.height
b.y < a.y + a.height
```



Check collision

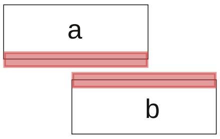
With the right arrow key, you make a star move: when it hits the box, the box "jumps" to the right.

The work is done by the `update(dt)` function:

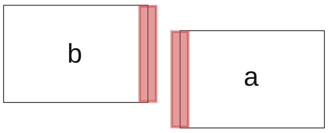
- if the right arrow key is pressed, the star moves to the right at the given speed.
- `if isColliding()` is true, the box gets moved by 100 pixels.

The `isColliding()` function is the core of this small program: it compares the corners of both shapes. The two shapes overlap if all the comparisons "match". In this case the function will return `true`. If any of the comparisons fails the function returns `false`.

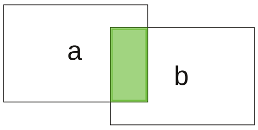
only colliding when all conditions are green: if even one single one is red, the shapes are not colliding.



```
a.x < b.x + b.width
b.x < a.x + a.width
a.y < b.y + b.height
b.y < a.y + a.height
```

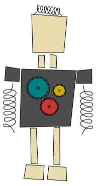


```
a.x < b.x + b.width
b.x < a.x + a.width
a.y < b.y + b.height
b.y < a.y + a.height
```



```
a.x < b.x + b.width
b.x < a.x + a.width
a.y < b.y + b.height
b.y < a.y + a.height
```

Creative Commons License CC-BY-SA,
Coderdojo Zürich, Ale Rimoldi.



Check collision

With the right arrow key, you make a star move: when it hits the box, the box "jumps" to the right.

The work is done by the update(dt) function:

- if the right arrow key is pressed, the star moves to the right at the given speed.
- if isColliding() is true, the box gets moved by 100 pixels.

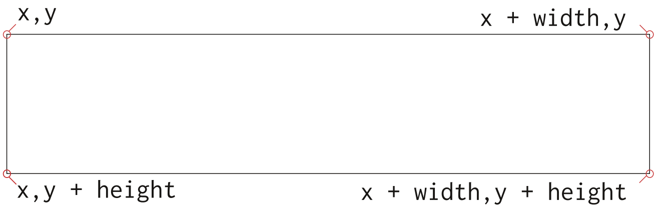
The isColliding() function is the core of this small program: it compares the corners of both shapes. The two shapes overlap if all the comparisons "match". In this case the function will return true. If any of the comparisons fails the function returns false.

```
local star = {x = 10, y = 200, speed = 150,
  char = '*', width = 10, height = 5}
local box = {x = 100, y = 200,
  char = '[ ]', width = 10, height = 5}

function isColliding(a, b)
  return
    a.x < b.x + b.width and
    b.x < a.x + a.width and
    a.y < b.y + b.height and
    b.y < a.y + a.height
end

function love.load(arg)
  love.graphics.setFont(love.graphics.newFont(36))
end

function love.update(dt)
  if isColliding(star, box) then
    box.x = box.x + 100
  end

  if love.keyboard.isDown('right', 'd') then
    star.x = star.x + (star.speed * dt)
  end
end
```

```
function love.draw()
  love.graphics.print(star.char, star.x, star.y)
  love.graphics.print(box.char, box.x, box.y)
end
```

How does `isColliding()` work?

`The isCollding()` function has two arguments a and b that must have four properties: `x`, `y`, `height` and `width`.
The coordinates (`x`, `y`) and the size (`width`, `height`) are need for calculating the corner points of the shapes:



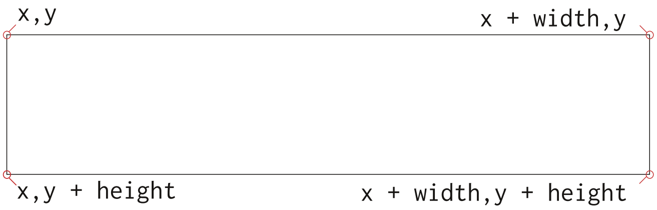
The four equations connected by the `and` conditions compare the corner points of the shapes `a` and `b`. The drawings below show in green the conditions that are met and in red the ones that are not met. The shapes are

```
local star = {x = 10, y = 200, speed = 150,
  char = '*', width = 10, height = 5}
local box = {x = 100, y = 200,
  char = '[ ]', width = 10, height = 5}

function isColliding(a, b)
  return
    a.x < b.x + b.width and
    b.x < a.x + a.width and
    a.y < b.y + b.height and
    b.y < a.y + a.height
end

function love.load(arg)
  love.graphics.setFont(love.graphics.newFont(36))
end

function love.update(dt)
  if isColliding(star, box) then
    box.x = box.x + 100
  end

  if love.keyboard.isDown('right', 'd') then
    star.x = star.x + (star.speed * dt)
  end
end
```

```
function love.draw()
  love.graphics.print(star.char, star.x, star.y)
  love.graphics.print(box.char, box.x, box.y)
end
```

How does `isColliding()` work?

`The isCollding()` function has two arguments a and b that must have four properties: `x`, `y`, `height` and `width`.
The coordinates (`x`, `y`) and the size (`width`, `height`) are need for calculating the corner points of the shapes:



The four equations connected by the `and` conditions compare the corner points of the shapes `a` and `b`. The drawings below show in green the conditions that are met and in red the ones that are not met. The shapes are