Programming for Everybody

5. Methods, blocks and sorting

le wagon

What are methods?

built in methods vs. methods coded by the developer

methods are **reusable** lines of code written to perform a repeatable and specific task

they are mathematical functions that can take one or multiple **parameters** and **arguments** (inputs) to compute calculations using those inputs and then return a result

methods are also known as *functions* in other languages (ex: JavaScript)

Why methods?

they are reusable and dynamic (the output depends on the input)

they help keeping the code organised by separating the different parts of the app: a specific method executes a specific task

this makes the code easier to manage: as it becomes more complex, bigger issues are easier to solve if the whole logic is divided into smaller methods

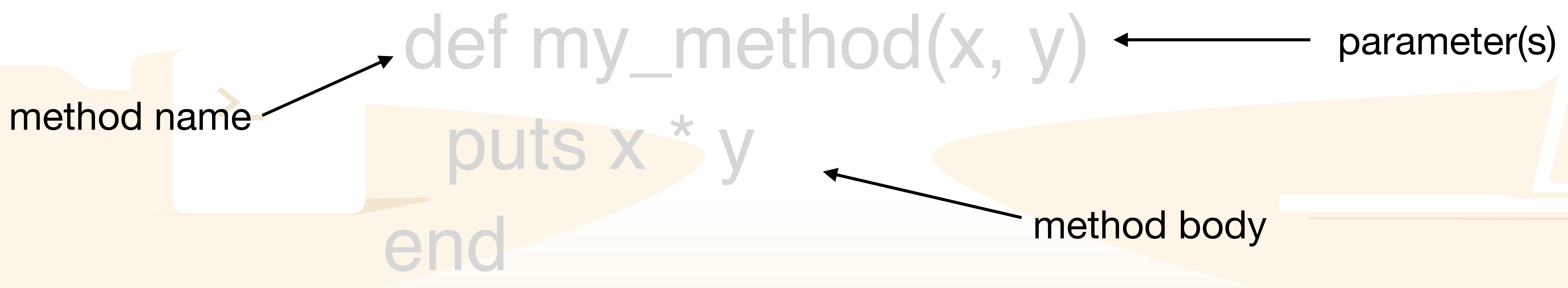
Syntax

methods have 3 parts:

header includes the **def** (short for “define”) keyword, the **name** of the method and any **parameters** the method takes

body includes the lines of code that determine the procedures the method carries out

end a method is closed using with the **end** keyword



```
def my_method(x, y)
  puts x * y
end
```

Calling a method

after defining a method we have to **call it** by **typing its name:** that's what triggers the program to look for a method with that name and then execute the code inside it

```
def my_method(x, y)
    puts x * y
end


my_method (2, 6)


(prints out 12)
```

parameters
the placeholder(s) we put put between the method's parentheses when we **define it**

arguments
the elements/values we put between the method's parentheses when we **call it**

Splat

sometimes methods may not know how many arguments there will be and the solution for that is **splat** -> *

a parameter with the splat operator allows the method to expect one or more arguments

```
def what_up(greeting, *friends)
  friends.each { |friend| puts "#{greeting}, #{friend}!" }
end

what_up("What up", "Ian", “Zoe",
“Zenas", "Eleanor")

#prints out:
What up, Ian!
What up, Zoe!
What up, Zenas!
What up, Eleanor!
```

vs.

```
def what_up(greeting, friends)
  friends.each { |friend| puts "#{greeting}, #{friend}!" }
end

what_up("What up", "Ian", "Zoe",    "Zenas",
“Eleanor")

#prints out:
wrong number of arguments (given 5, expected 2)
```

Returning

sometimes we don't want a method to print something to the console, but we just want it to hand us back a value which we can use afterwards -> that's what the **return** keyword does

when a methods returns, the value we get becomes available within the code and can thus be reused

```
def double(n)
  return n * 2        ← not printing, just giving us back the result
end

output = double(6)
output += 2
puts output     (prints out 14)
```

Blocks

blocks are chunks of code between curly braces **{}** or between the keywords **do** and **end** that we can associate with method invocations

unlike methods, blocks can only be called **once** and in the **specific context** under which they were created

often a method takes a block as a parameter (that’s what .each has been doing this whole time, for instance!)

```
names = [“Zoe”, “John”, “Zack”]

names.each do | name |
  puts reversed_name = name.reverse
end
```

```
names = [“Zoe”, “John”, “Zack”]

names.each { | name | puts
reversed_name = name.reverse }
```

Sorting

the sort method sorts the elements within a collection both from A - Z or from smaller to bigger numbers

```
names = [“Mary”, “John”, “Zack”]

puts names.sort

(prints out [“John”, “Mary”, “Zack”)
```

if we want to reverse the sorting, we just use the reverse method after the sort method!

Thank you!