LESSON 5

app.rb

```ruby
def square(x) = x * x
double = ->(x) { x * 2 }
[1, 2].map(&double)
```

Ruby

Blocks & Lambdas

Learn it all @MyWebUniversity



A FREE MYWEBUNIVERSITY COURSE

RUBY PROGRAMMING

LESSON 5

Methods, Blocks, Procs and Lambdas

def, default and keyword arguments, implicit return, yield, procs and lambdas

Defining Methods

methods.rb

```ruby
def square(x)
  x * x                    # implicit return
end

def passing?(score)
  score >= 50
end

square(7)        # 49
passing? 42      # false
```

- def starts a method and end closes it. The last expression is returned automatically, so …
- Methods that answer yes or no end with a question mark by convention.

01

Default and Keyword Arguments

args.rb

```ruby
def greet(name, greeting = "Salaam")
  "#{greeting}, #{name}!"
end

def book(title:, author:, year: 2026)
  "#{title} by #{author} (#{year})"
end
book(author: "Ferdowsi", title: "Shahnameh")
def total(*nums) = nums.sum       # endless method
```

- Parameters can have default values. Keyword arguments, with a colon after the name, are …
- A star collects extra arguments into an array, and double star collects extra keywords into …

02

Blocks and yield

yield.rb

```ruby
def twice
  yield 1
  yield 2
end
twice { |n| puts "call #{n}" }

def timed
  start = Time.now
  yield
  puts "took #{Time.now - start}s"
end
timed { sleep 0.5 }
```

- Any method can receive a block. Inside the method, yield runs the block and can pass values …
- block_given? checks whether a block was passed. This is how each, map and times are built.

03

Procs and Lambdas

lambda.rb

```ruby
double = ->(x) { x * 2 }
double.call(5)          # 10
double.(5)              # 10
double[5]               # 10

add_tax = proc { |p| p * 1.1 }
[100, 200].map(&double)        # [200, 400]
%w[a b].map(&:upcase)          # ["A", "B"]
```

- A block is not an object, but you can save one as a Proc or a lambda and call it later.
- The stabby lambda arrow syntax is the most common. Lambdas check their argument count; ...

04

Closures

counter.rb

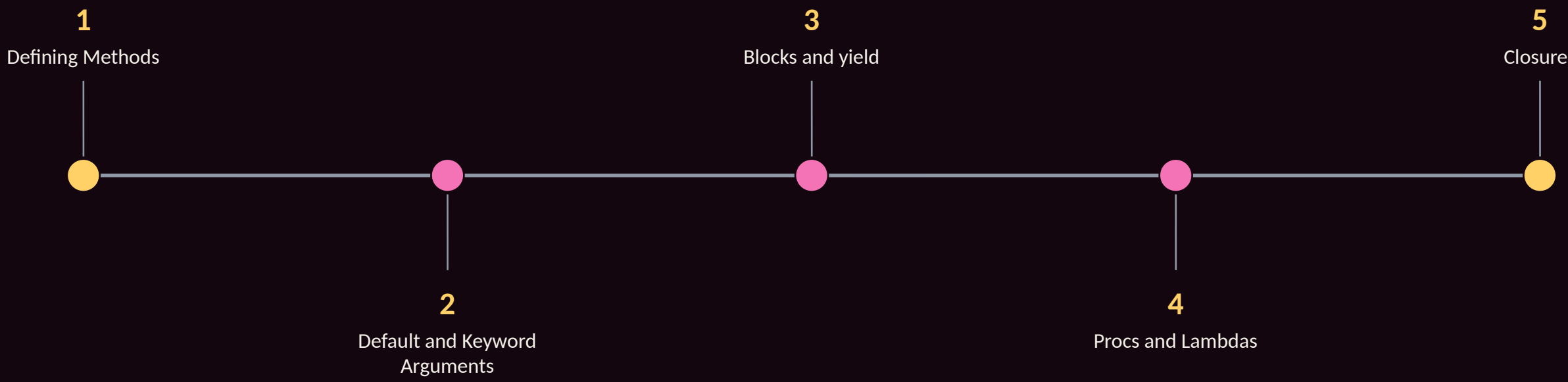
```ruby
def make_counter
  count = 0
  -> { count += 1 }
end

c = make_counter
c.call; c.call
puts c.call         # 3
```

- Blocks, procs and lambdas are closures: they remember the variables around them, even after …
- Here make_counter returns a lambda that keeps its own count.

05

Lesson Map


1
Defining Methods
2
Default and Keyword Arguments
3
Blocks and yield
4
Procs and Lambdas
5
Closures

Sources and Further Reading

methods.rb

```ruby
def square(x)
  x * x                    # implicit return
end

def passing?(score)
  score >= 50
end

square(7)        # 49
passing? 42      # false
```


yield.rb

```ruby
def twice
  yield 1
  yield 2
end
twice { |n| puts "call #{n}" }

def timed
  start = Time.now
  yield
  puts "took #{Time.now - start}s"
end
timed { sleep 0.5 }
```

Thank you for watching

MyWebUniversity.com

Free technical education since 2000 — English and Dari

@MyWebUniversity · @MyWebUniversity-Dari