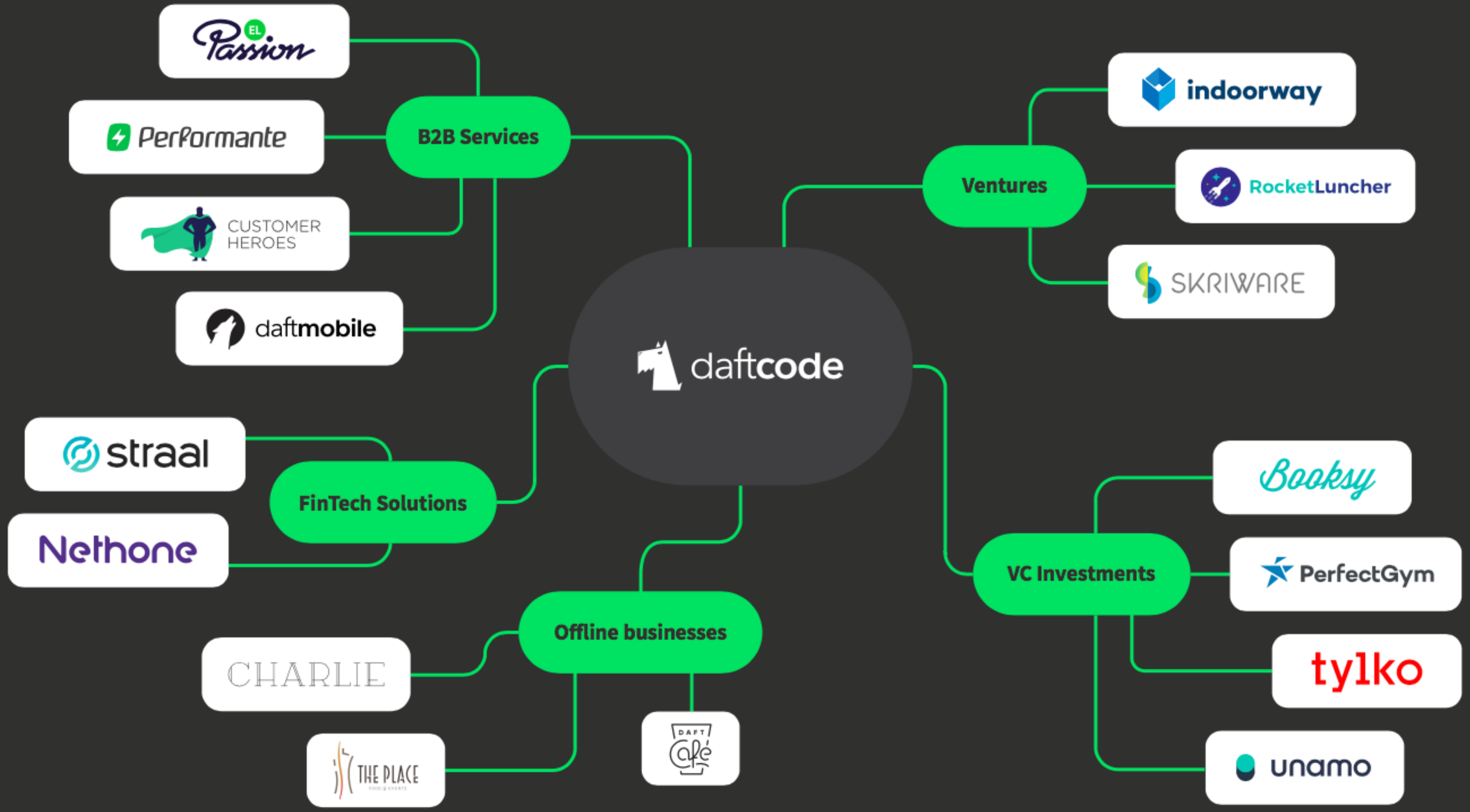
RUBY ON RAILS
BEGINNERS

daftcode


daftcode
B2B Services
El Passion
Performante
CUSTOMER HEROES
daftmobile
Ventures
indoorway
RocketLuncher
SKRIWARE
FinTech Solutions
straal
Nethone
VC Investments
Booksy
PerfectGym
tylko
unamo
Offline businesses
CHARLIE
THE PLACE
DAFT Café

daftcode

PLAN ZAJĘĆ

23.10.2017

1. **Witaj w świecie Ruby**

   Charakterystyka języka

30.10.2017

2. **Fundamenty Ruby on Rails**

   Wzorzec MVC i struktura frameworku

6.11.2017

3. **Reprezentacja danych w aplikacji**

   REST w teorii i praktyce

13.11.2017

4. **Zbuduj skracacz linków**

   Od prototypu do aplikacji

20.11.2017

5. **Upewnij się, że działa**

   Testowanie przy pomocy RSpec

27.11.2017

6. **Kontrola dostępu i bezpieczeństwa**

   Konta użytkownika i metody autoryzacji

4.12.2017

7. **Dla chcących więcej**

   Kolejne kroki w Ruby

Ruby

Cechy języka

daftcode

daftcode

Obiektowy

```
3.even?              #=> false
5.78.round           #=> 6
'hello'.capitalize   #=> 'Hello'
[1, 2, 3].min        #=> 1
Time.now.friday?     #=> false
```

daftcode

Strukturalny

```
def factorial n
  outcome = 1
  base = 1

  while base <= n
    outcome *= base
    base += 1
  end

  return outcome
end

factorial 5 #=> 120
```

daftcode

Refleksyjny

```
my_method() #=> NoMethodError

define_method(:my_method) do
  'Works now!'
end

my_method() #=> 'Works now!'
```

daftcode

Typowanie

dynamiczne

```
things = [0, 0.5, 'string']

things[0].class #=> Fixnum
things[1].class #=> Float
things[2].class #=> String

things[2] = []
things[2].class #=> Array
```

daftcode

Typowanie

ducktype

```ruby
def get_excited_with thing
  "#{thing.class}s are awesome!"
end

get_excited_with 1.5                #=> Floats are awesome!
get_excited_with 'text'             #=> Strings are awesome!
get_excited_with [:this, 'array'] #=> Arrays are awesome!
```

daftcode

Typowanie

silne

```
'string' + 1          #=> TypeError
'string' * 'another'  #=> TypeError
10 - 'string'         #=> TypeError
```

Podstawowe konstrukcje

daftcode

daftcode

Definicja metody

```ruby
def my_method(param1, param2='default value')
  param1 + param2
end

my_method('this is a ')          #=> 'this is a default value'
my_method('this is ', 'custom') #=> 'this is custom'
```

daftcode

Nil

```
nil.object_id #=> 8
nil.object_id #=> 8

5.nil?        #=> false

array = [0, 1, 2]
array[3]      #=> nil
```

daftcode

Symbol

```
'string'.object_id  #=> ...680
'string'.object_id  #=> ...540
'string' + 's'      #=> 'strings'

:symbol.object_id   #=> 795548
:symbol.object_id   #=> 795548
:symbol + :s        #=> NoMethodError
```

daftcode

Zmienne

```
local_variable = 1

@instance_variable = 2

@@class_variable = 3

$global_variable = 4

CONSTANT = 5
```

daftcode

Definicja klasy

```ruby
class MyClass < ParentClass

  attr_accessor :email, :age

  @@class_variable = 9

  def initialize email, age
    @email = email
    @age = age
  end

  def instance_method
    "#{email}: #{age}"
  end

  def self.class_method
    @@class_variable
  end
end
```

daftcode

Definicja klasy

```
my_object = MyClass.new('me@example.com', 25)
my_object.age                #=> 25
my_object.instance_method    #=> 'me@example.com: 25'
my_object.class_method       #=> NoMethodError

MyClass.instance_method      #=> NoMethodError
MyClass.class_method         #=> 9
```

daftcode

Raise/Rescue

```
begin
  do_something
rescue SomeError => e
  error_action
ensure
  always_do_this
end

do_something rescue error_action

raise SomeError
```

daftcode

If/Else

```
if condition
  condition_true
else
  condition_false
end

unless condition
  condition_false
else
  condition_true
end

do_something if condition
```

daftcode

Case

```ruby
def grade percent
  case percent
  when 100
    5
  when 70...100
    4
  when 50...70
    3
  else
    2
  end
end
```

```ruby
grade 100      #=> 5
grade 66       #=> 3
grade 'error'  #=> 2
```

daftcode

Warunkowe przypisania

```
variable = get_value rescue get_other_value

variable = get_value if condition_true

variable = if condition_true
  get_value
else
  get_other_value
end

variable = condition_true ? get_value : get_other_value

variable = get_value || get_other_value

variable ||= get_value
```

daftcode

Splat

```ruby
def sentence *args
  args.join(' ').capitalize
end

sentence 'hello', 'world' #=> 'Hello world'

array = ['ala', 'ma', 'kota']

sentence *array           #=> 'Ala ma kota'
```

daftcode

Hash

```ruby
hash = {
  'key' => 'value',
  3 => 8.5,
  symbol: :value
}

hash[3]           #=> 8.5
hash[:symbol]     #=> :value
hash['not_here'] #=> nil
```

daftcode

Range

```
(1..3).to_a           #=> [1, 2, 3]
(1...3).to_a          #=> [1, 2]

('a'..'e').to_a.join #=> 'abcde'

(0.2..1.6).bsearch { |f| Math.log(f) >= 0 } #=> 1.0
```

daftcode

While

```
while condition_true
  do_something
end

until condition_false
  do_something
end

loop do # while true
  do_something
  break if condition_false
end

do_something until condition_false
```

daftcode

Blok

```ruby
[1, 2, 3].each do |number|
  number.to_s
end

[1, 2, 3].each { |number| number.to_s }
```

daftcode

Wywołanie bloku

```ruby
def message_for number
  if number.is_a? Fixnum
    yield(number) if block_given?
  else
    'NaN'
  end
end

message_for(5) { |n| "#{n} is fine!" }      #=> '5 is fine!'
message_for('asd') { |n| "#{n} is not :(" } #=> 'NaN'
```

daftcode

For / foreach

```
10.times { |t| puts "hello #{t}" }

(0..10).each { |t| puts "hello #{t}" }
```

daftcode

Each / map

```
array = [1] * 10

array.each { |el| puts "hello #{el}" }

array.each.with_index { |el, i| puts "hello #{i}" }

array.map { |el| el + 1 }

array.map.with_index { |el, i| el + i }
```

daftcode

Shorthand bloku

```ruby
some_method do |first_argument, *all_the_rest|
  first_argument.do_something(*all_the_rest)
end

some_method(&:do_something)
```

daftcode

Przykład inject

```ruby
[1, 2, 3].inject { |element, sum| sum + element }

[1, 2, 3].inject { |element, sum| element + sum }

[1, 2, 3].inject { |element, sum| element.+(sum) }

[1, 2, 3].inject(&:+)

[1, 2, 3].inject(:+) #=> 6

[1, 2, 3].reduce(:+) #=> 6
```

The Ruby way

daftcode

Refactor

```
def factorial n
  outcome = 1
  base = 1

  while base <= n
    outcome *= base
    base += 1
  end

  return outcome
end

factorial 5 #=> 120
```

daftcode

Refactor

```
def factorial n
  outcome = 1

  (1..n).each do |base|
    outcome *= base
  end

  return outcome
end

factorial 5 #=> 120
```

daftcode

Refactor

```ruby
def factorial n
  outcome = (1..n).inject(1) do |base, acc|
    acc * base
  end

  return outcome
end

factorial 5 #=> 120
```

daftcode

Refactor

```ruby
def factorial n
  (1..n).inject(1) do |base, acc|
    acc * base
  end
end

factorial 5 #=> 120
```

daftcode

Refactor

```ruby
def factorial n
  (1..n).inject(1) { |base, acc| base * acc }
end

factorial 5 #=> 120
```

daftcode

Refactor

```ruby
def factorial n
  (1..n).inject(1, :*)
end

factorial 5 #=> 120
```

daftcode

Refactor

```ruby
def factorial n
  outcome = 1
  base = 1

  while base <= n
    outcome *= base
    base += 1
  end

  return outcome
end

factorial 5 #=> 120
```

```ruby
def factorial n
  (1..n).inject(1, :*)
end

factorial 5 #=> 120
```

daftcode

Dlaczego RoR?

- kompleksowy
- popularny - dużo rozwiązań
- duże możliwości, niski entry level

Pytania?

daftcode