GENERAL ASSEMBLY

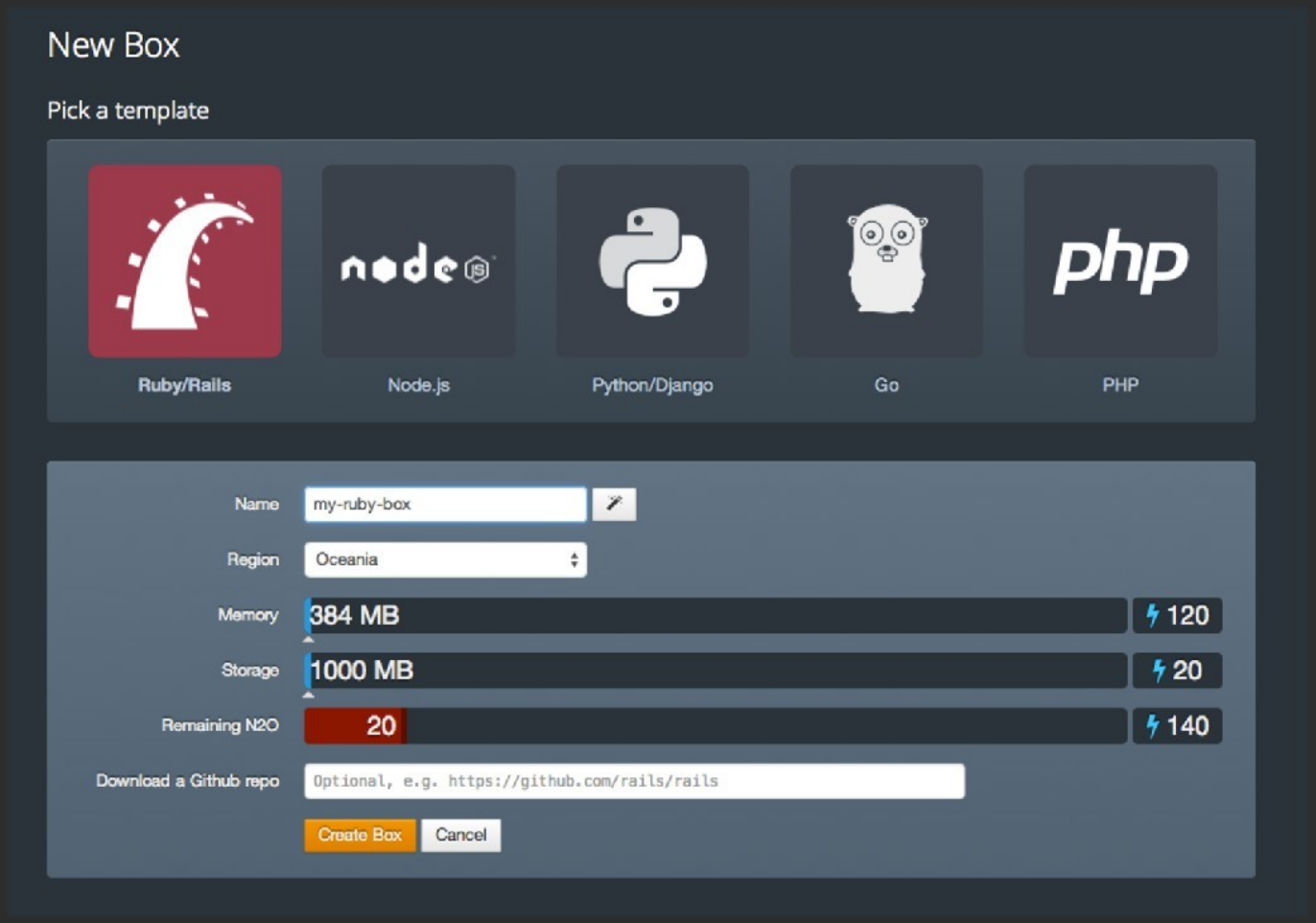
PROGRAMMING FUNDAMENTALS DAY 3

Andrew Buntine
Technical Director, Hardhat

DEVELOPMENT ENVIRONMENT SETUP

Create a Nitrous account (for authoring)
https://nitrous.io/

Create a Github account (for collaborating)
https://github.com/


New Box
Pick a template
Ruby/Rails
Node.js
Python/Django
Go
PHP
Name
my-ruby-box
Region
Oceania
Memory
384 MB
120
Storage
1000 MB
20
Remaining N2O
20
140
Download a Github repo
Optional, e.g. https://github.com/rails/rails
Create Box
Cancel

AGENDA

LEVEL 1

- VOCABULARY
- PROCESS
- BASICS OF CODE

LEVEL 2

- HTML
- CSS
- Javascript

LEVEL 3

- RUBY/RAILS
- CREATE A BASIC WEB APPLICATION

LEVEL 4

- PUTTING IT ALL TOGETHER

TODAY'S AGENDA: BACKEND DEVELOPMENT

Recap

Development Environment

What is backend development?

What is programming?

Programming concepts

- Loops, Variables, Types, Conditionals, Functions, Lists, Arrays
- Libraries, Frameworks

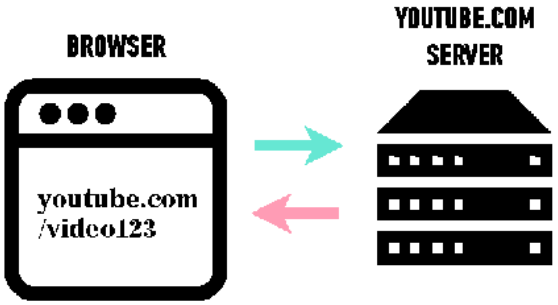
Web Frameworks

Templates, variables, conditionals

Deployment to Heroku

RECAP

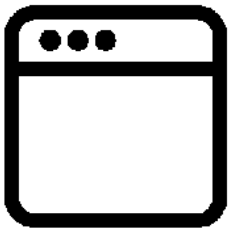
HOW THE WEB WORKS



BROWSER REQUEST: "Connect me to server youtube.com. Great, please send me **/video123**"

SERVER RESPONSE: "Okay. I have found /video123, it is a HTML file. Here is the HTML."

FRONT-END AND BACK-END



FRONT-END

HTML, CSS
Javascript

BACK-END

Ruby, Python
PHP, Databases

HTML "DOCUMENT"

```
<!doctype html>
<html>
  <head>
    <title>Hipster Coffee Co.</title>
  </head>
  <body>
    <p>Tasty coffee</p>
  </body>
</html>
```

TAGS

Root element	`<html>`
Document metadata	`<head>` `<title>` `<link>` `<meta>`
Scripting	`<script>`
Sections	`<section>` `<nav>` `<article>` `<header>` `<footer>`
Grouping	`<p>` `<ol>` `<ul>` `<li>` `<dl>` `<div>`
Text	`<a>` `<em>` `<strong>` `<span>`
Embedded content	`<img>` `<iframe>` `<video>` `<audio>` `<canvas>` `<svg>`
Tabular data	`<table>` `<tr>` `<th>` `<td>`
Forms	`<form>` `<fieldset>` `<input>` `<label>`

TRAVEL BLOG HTML

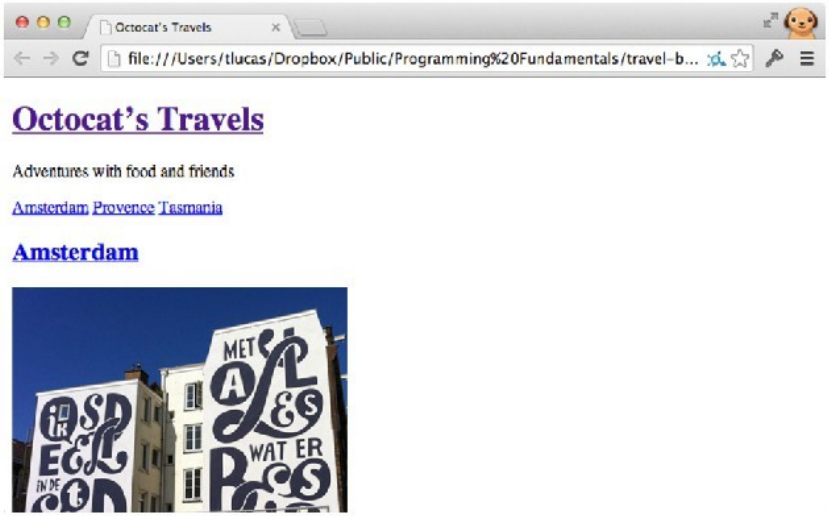
HTML structure

Tags

Attributes

Chrome Dev Tools

Sections (header, footer, section, etc)

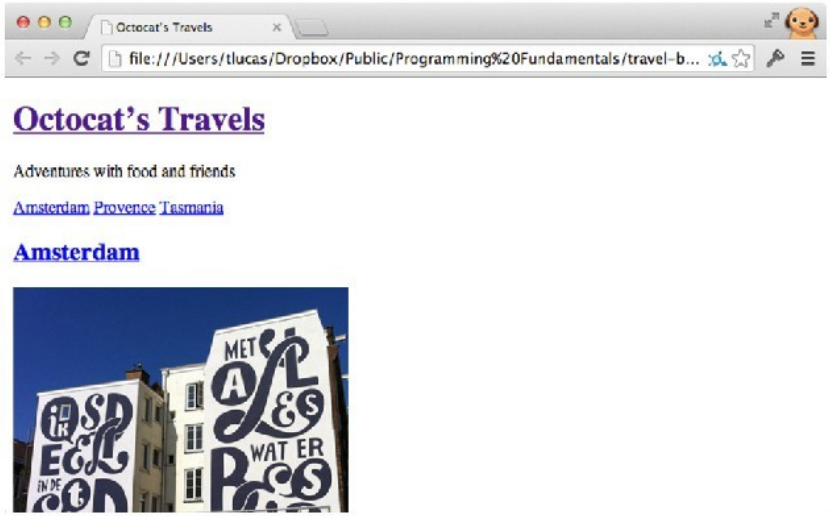


USER AGENT STYLESHEET

Built into every browser

The default stylesheet

See them with Chrome Dev Tools



NORMALIZE VS RESET

normalize.css

- Makes all browsers look the same
- Retains default styles

reset.css

- Removes all the default styles
- Need to make sure you re-add your own styles

CSS SYNTAX

```
selector {
  property: value;
}
```

CSS PROPERTIES

```
.site-footer {
  font: bold 21px “Comic Sans MS”, sans-serif;
  border: 1px solid red;
}
```

TRAVEL BLOG CSS

Linking to a stylesheet

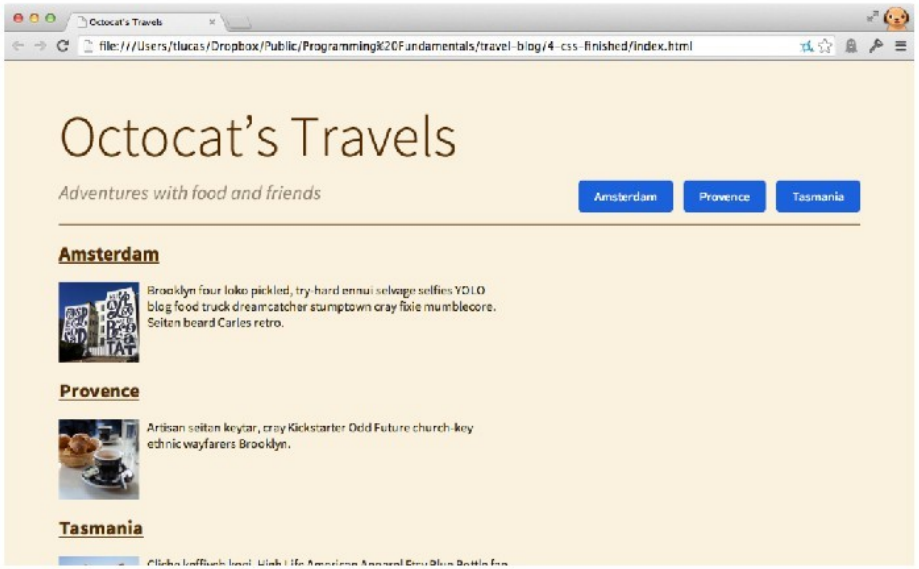
Adding custom fonts

Add classes to your HTML

Adding CSS styles

Colors, fonts, underlines, links, buttons

Positioning



JAVASCRIPT

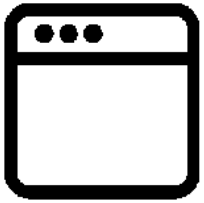
```
var a = document.querySelector("a");

a.addEventListener("click", function() {
  var name = prompt("Hi, I'm T-Rex. Who are you?");
  a.textContent = "Hai " + name + "! :)";
});
```

JQUERY

```
$("a").click(function() {
  var name = prompt("Hi, I'm T-Rex. Who are you?");
  $(this).text("Hai " + name + "! :)");
});
```

WHAT IS A LIBRARY?



FRONT END

Platform:	Browser
Language:	Javascript
Libraries:	**jQuery** **jQuery.UI** **Backbone** **Ember**

BACK END

Platform:	Node.js
Language:	Javascript
Libraries: *(Modules)*	**mocha** **express** **connect** **request**

IMAGE FORMATS

SVG + JPEG COVERS 99% OF NEEDS

SVG for all logos, icons etc

JPEG for all photograph/bitmap images

Future proof + size friendly

BACKEND DEVELOPMENT

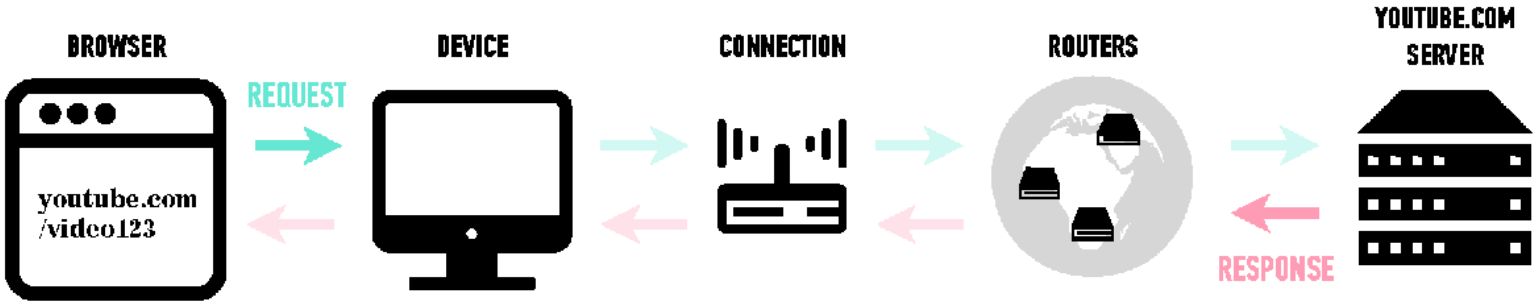
DEVELOPMENT ENVIRONMENT SETUP

Create a Nitrous account (for authoring)
https://nitrous.io/

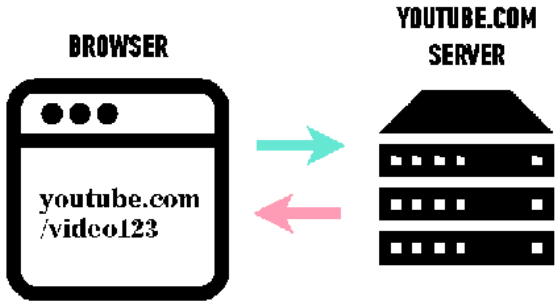
Create a Github account (for collaborating)
https://github.com/

WHAT IS BACKEND DEVELOPMENT?

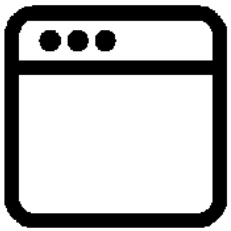
HOW THE WEB WORKS



HOW THE WEB WORKS



HOW THE WEB WORKS

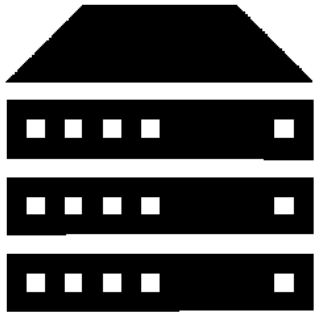


FRONT-END

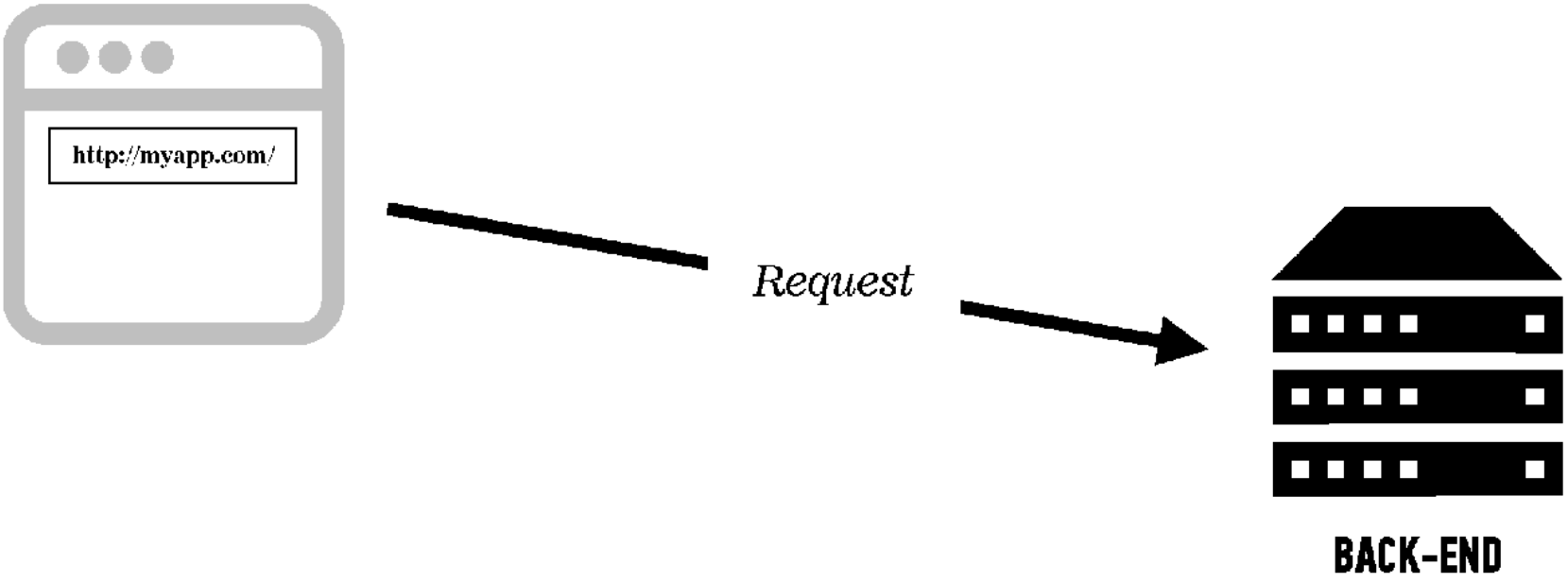
HTML, CSS
Javascript

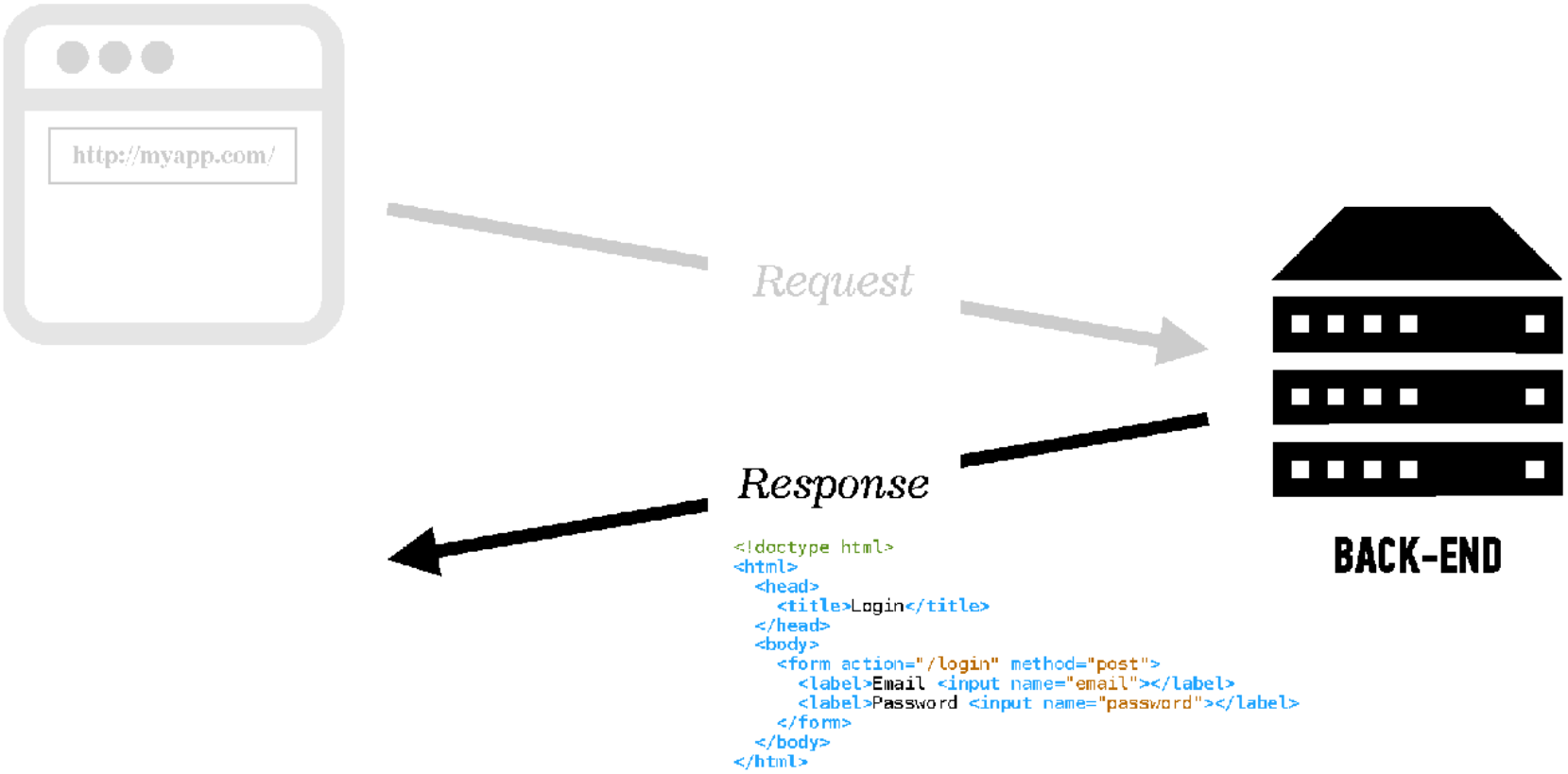
BACK-END

Ruby, Python
PHP, Databases



BACK-END

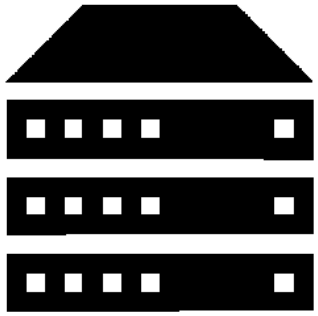

http://myapp.com/
Request
BACK-END


http://myapp.com/
Request
Response
BACK-END
<!doctype html>
<html>
<head>
<title>Login</title>
</head>
<body>
<form action="/login" method="post">
<label>Email <input name="email"></label>
<label>Password <input name="password"></label>
</form>
</body>
</html>

WHAT IS BACKEND DEVELOPMENT?

Request

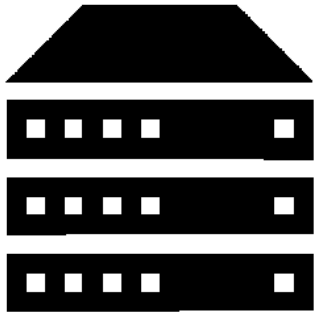
Response

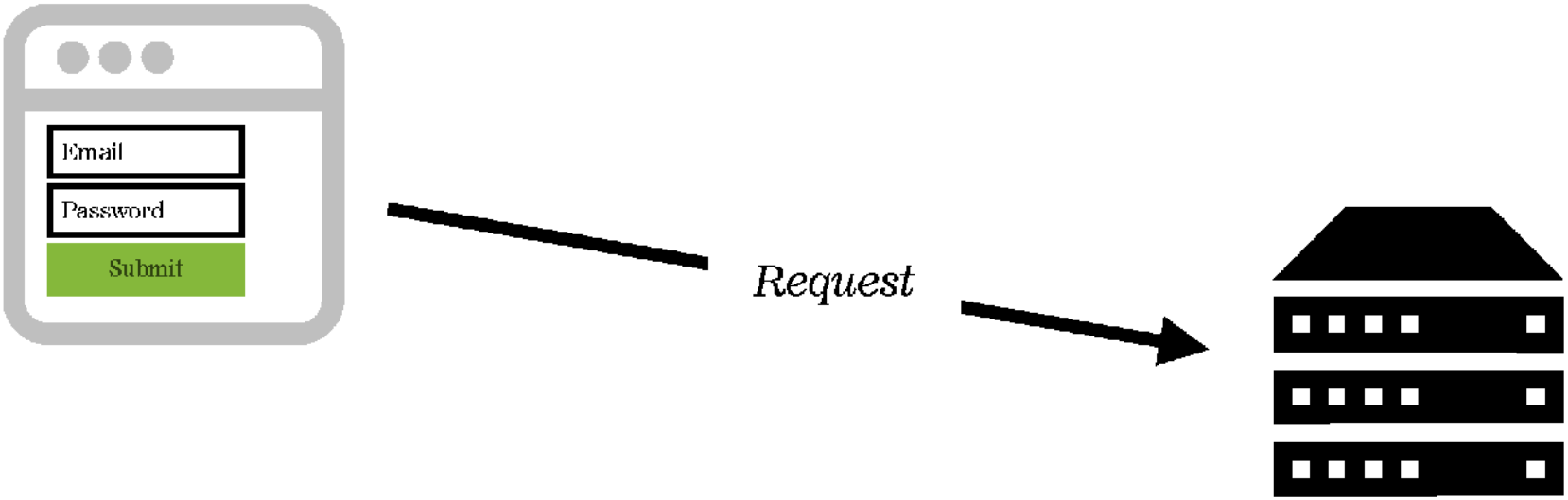


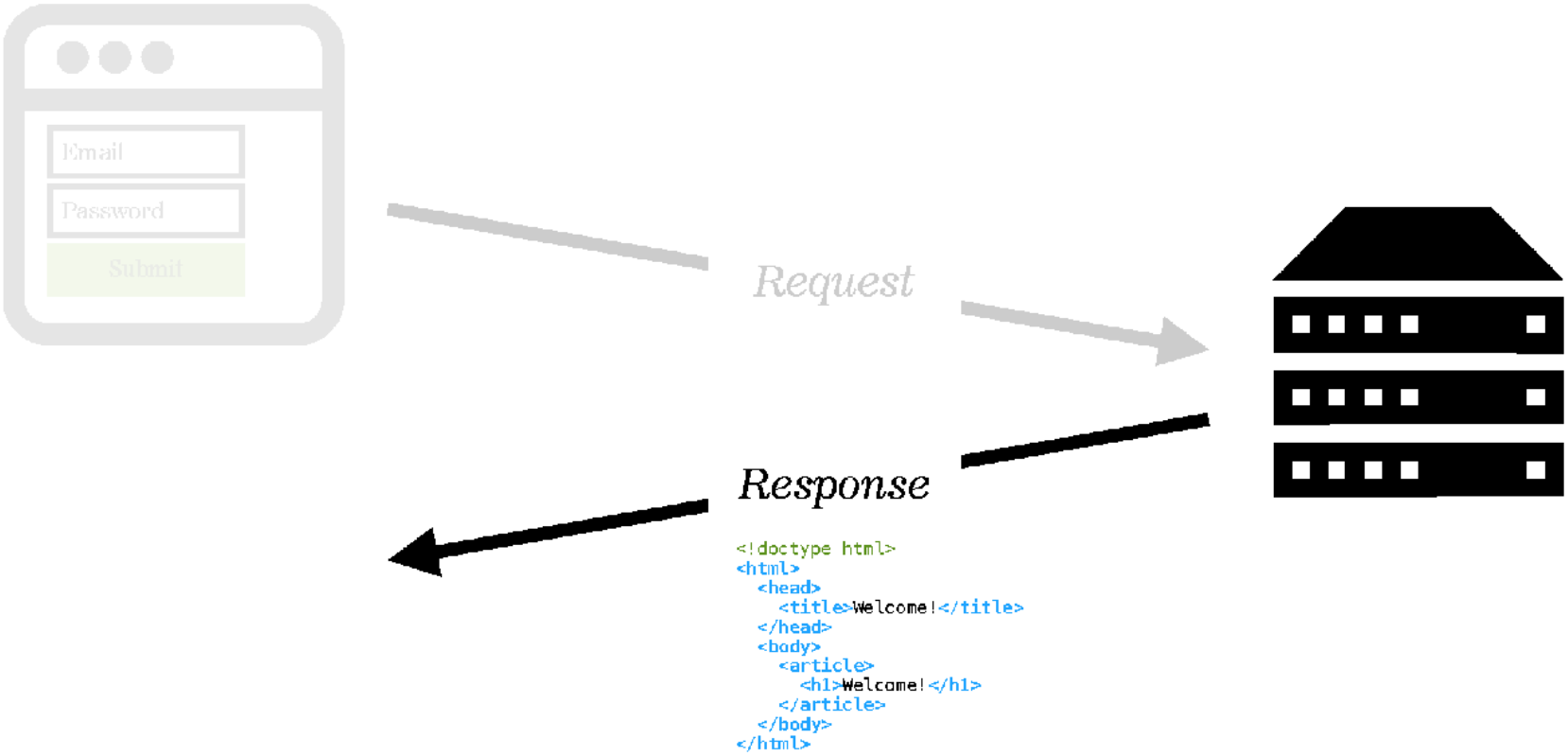
BACK-END

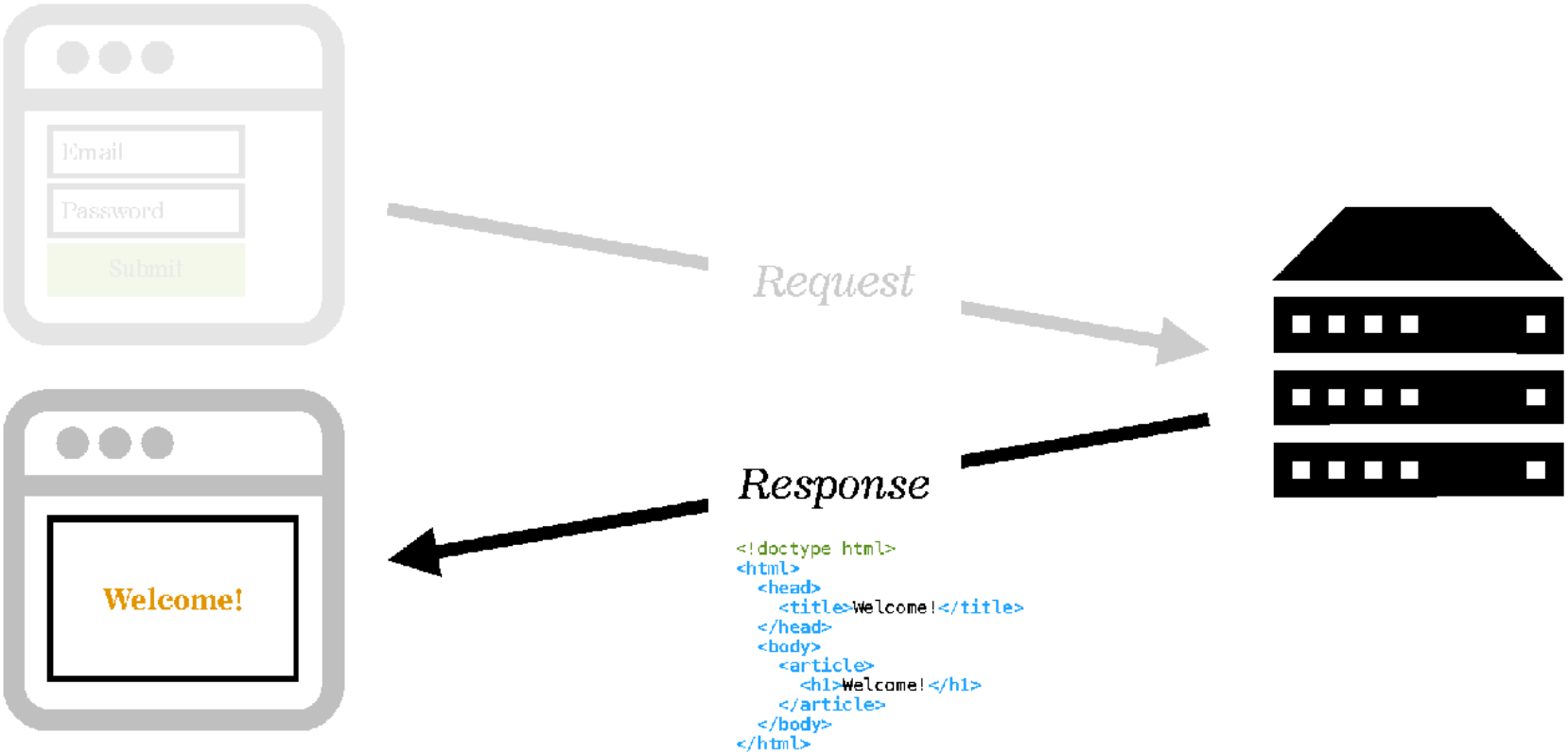
```
<!doctype html>
<html>
  <head>
    <title>Login</title>
  </head>
  <body>
    <form action="/login" method="post">
      <label>Email <input name="email"></label>
      <label>Password <input name="password"></label>
    </form>
  </body>
</html>
```


Email
Password
Submit




Email
Password
Submit
Request


Email
Password
Submit
Request
Response
<!doctype html>
<html>
<head>
<title>Welcome!</title>
</head>
<body>
<article>
<h1>Welcome!</h1>
</article>
</body>
</html>


Email
Password
Submit
Request
Response
Welcome!
<!doctype html>
<html>
<head>
<title>Welcome!</title>
</head>
<body>
<article>
<h1>Welcome!</h1>
</article>
</body>
</html>

DEMO

http://login-form.bunts.io

Source code:

http://github.com/buntine/login-form

WHAT ELSE CAN YOU DO?

Use APIs

- *Grab a picture from Reddit.com*
  using their API
- *Email that picture to a friend*
  using the Mandrill API

DEMO

http://picster.bunts.io

Source code:

http://github.com/buntine/picster

WHAT ELSE CAN YOU DO?

Retrieve, store and update information from a database

Read, store and update files

Render templates

Set HTTP cookies

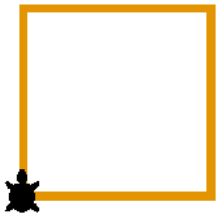
PROGRAMMING CONCEPTS

EXERCISE: TURTLE

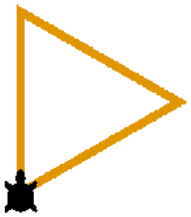
Imagine there was a turtle that understood the instructions (functions) on the right.

Break up into *groups of 2*

Write down a recipe (routine) for each of the pictures below, using the functions available.



Square



Triangle

Double Square

Documentation

Available turtle functions

- `Forward` x `metres`
- `Turn right` x `degrees`
- `Turn left` x `degrees`
- `Repeat` x `times [`
  - *instructions*

  `]`

```
Forward 5 metres
Turn right 90 degrees
Forward 5 metres
Turn right 90 degrees
Forward 5 metres
Turn right 90 degrees
Forward 5 metres
Turn right 90 degrees
```

```
Forward 5 metres
Turn right 90 degrees
Forward 5 metres
Turn right 90 degrees
Forward 5 metres
Turn right 90 degrees
Forward 5 metres
Turn right 90 degrees
```

or

```
Repeat 4 times [
  Forward 5 metres
  Turn right 90 degrees
]
```

```
Forward 5 metres
Turn right 90 degrees
Forward 5 metres
Turn right 90 degrees
Forward 5 metres
Turn right 90 degrees
Forward 5 metres
Turn right 90 degrees
```

or

```
Repeat 4 times [
  Forward 5 metres
  Turn right 90 degrees
]
```

```
Forward 5 metres
Turn right 90 degrees
Forward 5 metres
Turn right 90 degrees
Forward 5 metres
Turn right 90 degrees
Forward 5 metres
Turn right 90 degrees
```

or

```
Repeat 4 times [
  Forward 5 metres
  Turn right 90 degrees
]
```

```
Forward 5 metres
Turn right 120 degrees
Forward 5 metres
Turn right 120 degrees
Forward 5 metres
Turn right 120 degrees
```

```
Forward 5 metres
Turn right 90 degrees
Forward 5 metres
Turn right 90 degrees
Forward 5 metres
Turn right 90 degrees
Forward 5 metres
Turn right 90 degrees
```

or

```
Repeat 4 times [
  Forward 5 metres
  Turn right 90 degrees
]
```

```
Forward 5 metres
Turn right 120 degrees
Forward 5 metres
Turn right 120 degrees
Forward 5 metres
Turn right 120 degrees
```

or

?

```
Forward 5 metres
Turn right 90 degrees
Forward 5 metres
Turn right 90 degrees
Forward 5 metres
Turn right 90 degrees
Forward 5 metres
Turn right 90 degrees
```

or

```
Repeat 4 times [
  Forward 5 metres
  Turn right 90 degrees
]
```

```
Forward 5 metres
Turn right 120 degrees
Forward 5 metres
Turn right 120 degrees
Forward 5 metres
Turn right 120 degrees
```

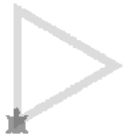
or

```
Repeat 3 times [
  Forward 5 metres
  Turn right 120 degrees
]
```

```
Forward 5 metres
Turn right 90 degrees
Forward 5 metres
Turn right 90 degrees
Forward 5 metres
Turn right 90 degrees
Forward 5 metres
Turn right 90 degrees
```

or

```
Repeat 4 times [
  Forward 5 metres
  Turn right 90 degrees
]
```



```
Forward 5 metres
Turn right 120 degrees
Forward 5 metres
Turn right 120 degrees
Forward 5 metres
Turn right 120 degrees
```

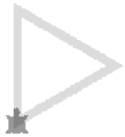
or

```
Repeat 3 times [
  Forward 5 metres
  Turn right 120 degrees
]
```

```
Forward 5 metres
Turn right 90 degrees
Forward 5 metres
Turn right 90 degrees
Forward 5 metres
Turn right 90 degrees
Forward 5 metres
Turn right 90 degrees
```

or

```
Repeat 4 times [
  Forward 5 metres
  Turn right 90 degrees
]
```



```
Forward 5 metres
Turn right 120 degrees
Forward 5 metres
Turn right 120 degrees
Forward 5 metres
Turn right 120 degrees
```

or

```
Repeat 3 times [
  Forward 5 metres
  Turn right 120 degrees
]
```

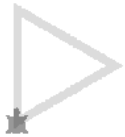
```
Repeat 4 times [
  Forward 5 metres
  Turn right 90 degrees
]
```

PROGRAMMING CONCEPTS: TURTLE SOLUTIONS

```
Forward 5 metres
Turn right 90 degrees
Forward 5 metres
Turn right 90 degrees
Forward 5 metres
Turn right 90 degrees
Forward 5 metres
Turn right 90 degrees
```

or

```
Repeat 4 times [
  Forward 5 metres
  Turn right 90 degrees
]
```



```
Forward 5 metres
Turn right 120 degrees
Forward 5 metres
Turn right 120 degrees
Forward 5 metres
Turn right 120 degrees
```

or

```
Repeat 3 times [
  Forward 5 metres
  Turn right 120 degrees
]
```

```
Repeat 4 times [
  Forward 5 metres
  Turn right 90 degrees
]
Turn right 45 degrees
```

```
Forward 5 metres
Turn right 90 degrees
Forward 5 metres
Turn right 90 degrees
Forward 5 metres
Turn right 90 degrees
Forward 5 metres
Turn right 90 degrees
```

or

```
Repeat 4 times [
  Forward 5 metres
  Turn right 90 degrees
]
```

```
Forward 5 metres
Turn right 120 degrees
Forward 5 metres
Turn right 120 degrees
Forward 5 metres
Turn right 120 degrees
```

or

```
Repeat 3 times [
  Forward 5 metres
  Turn right 120 degrees
]
```

```
Repeat 4 times [
  Forward 5 metres
  Turn right 90 degrees
]
Turn right 45 degrees
Repeat 4 times [
  Forward 5 metres
  Turn right 90 degrees
]
```

```
Forward 5 metres
Turn right 90 degrees
Forward 5 metres
Turn right 90 degrees
Forward 5 metres
Turn right 90 degrees
Forward 5 metres
Turn right 90 degrees
```

or

```
Repeat 4 times [
  Forward 5 metres
  Turn right 90 degrees
]
```

```
Forward 5 metres
Turn right 120 degrees
Forward 5 metres
Turn right 120 degrees
Forward 5 metres
Turn right 120 degrees
```

or

```
Repeat 3 times [
  Forward 5 metres
  Turn right 120 degrees
]
```

```
Repeat 4 times [
  Forward 5 metres
  Turn right 90 degrees
]
Turn right 45 degrees
Repeat 4 times [
  Forward 5 metres
  Turn right 90 degrees
]
```

or

```
Repeat 2 times [
  Repeat 4 times [
    Forward 5 metres
    Turn right 90 degrees
  ]
  Turn right 45 degrees
]
```

ALGORITHMS

AKA FUNCTIONS, METHODS, ROUTINES

Square

```
Repeat 4 times [
  Forward 5 metres
  Turn right 90 degrees
]
```

Triangle

```
Repeat 3 times [
  Forward 5 metres
  Turn right 120 degrees
]
```

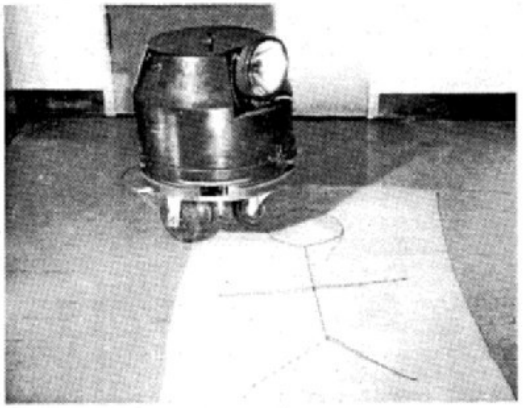
Double Square

```
Repeat 2 times [
  Repeat 4 times [
    Forward 5 metres
    Turn right 90 degrees
  ]
  Turn right 45 degrees
]
```

YOU JUST WROTE LOGO

http://www.calormen.com/jslogo/

1. Make a Turtle



The picture shows one of our turtles . . . so-called in honor of a famous species of cybernetic animal made by Grey Walter, an English neurophysiologist. Grey Walter's turtles had life-like behavior patterns built into its wiring diagram. Ours have no behavior except the ability to obey a few simple commands from a computer to which they are attached by a wire that plugs into a control-box that connects to a telephone line that speaks to the computer, which thinks it is talking to a

PERSON

FROM "TWENTY THINGS TO DO WITH A COMPUTER" 1971, SEYMOUR PAPERY & CYNTHIA SOLOMON

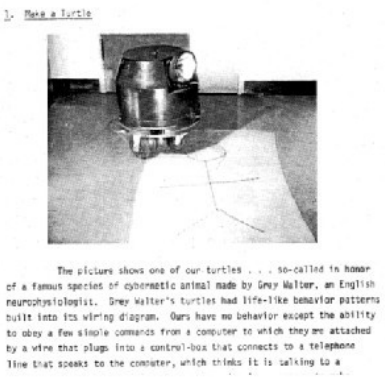
```
TO draw :distance
  forward :distance
  back :distance
END

TO vee :size
  left 50
  draw :size
  right 100
  draw :size
  left 50
END

TO circle :diameter
  arc 360 :diameter/2
END
```

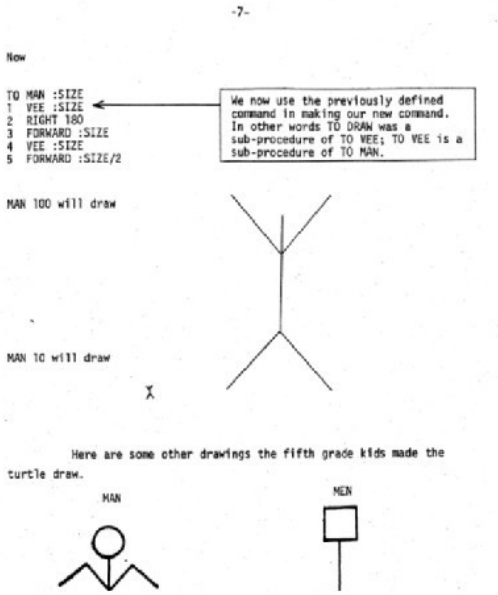
```
TO person :size
  right 180
  vee :size
  right 180
  forward :size
  vee :size
  forward :size/2
  penup
  forward 50
  pendown
  circle :size
END

person 100
person 10
```



1. Make a turtle

The picture shows one of our turtles . . . so-called in honor of a famous species of cybernetic animal made by Grey Walter, an English neurophysiologist. Grey Walter's turtles had life-like behavior patterns built into its wiring diagram. Ours have no behavior except the ability to obey a few simple commands from a computer to which they are attached by a wire that plugs into a control-box that connects to a telephone line that speaks to the computer, which thinks it is talking to a



-7-

Now

```
TO MAN :SIZE
1  VEE :SIZE
2  RIGHT 180
3  FORWARD :SIZE
4  VEE :SIZE
5  FORWARD :SIZE/2
```

We now use the previously defined command in making our new command. In other words TO DRAW was a sub-procedure of TO VEE; TO VEE is a sub-procedure of TO MAN.

MAN 100 will draw

MAN 10 will draw

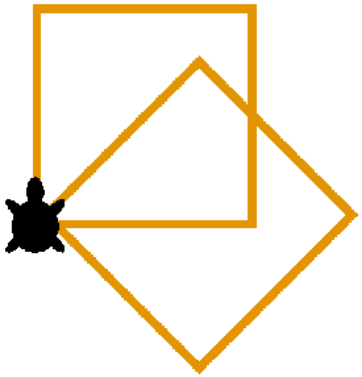
Here are some other drawings the fifth grade kids made the turtle draw.

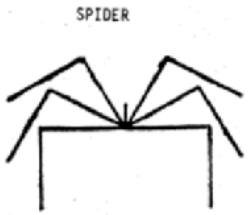
MAN

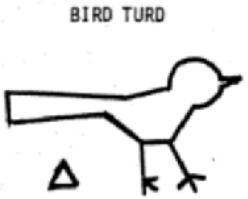
MEN

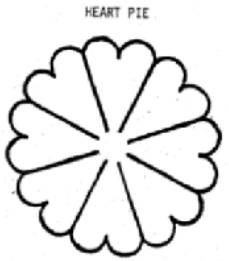
A CREATIVE PROCESS

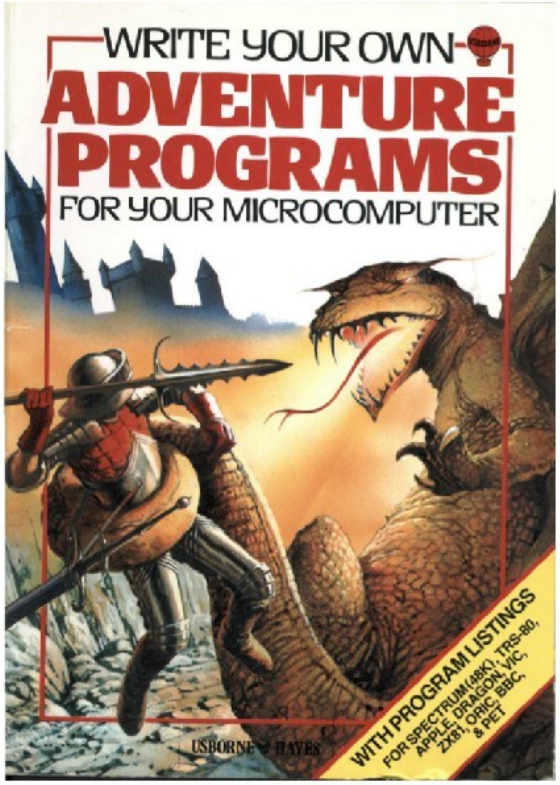
Understanding the problem
Exploring solutions
Pattern recognition
Abstraction
Sharing ideas

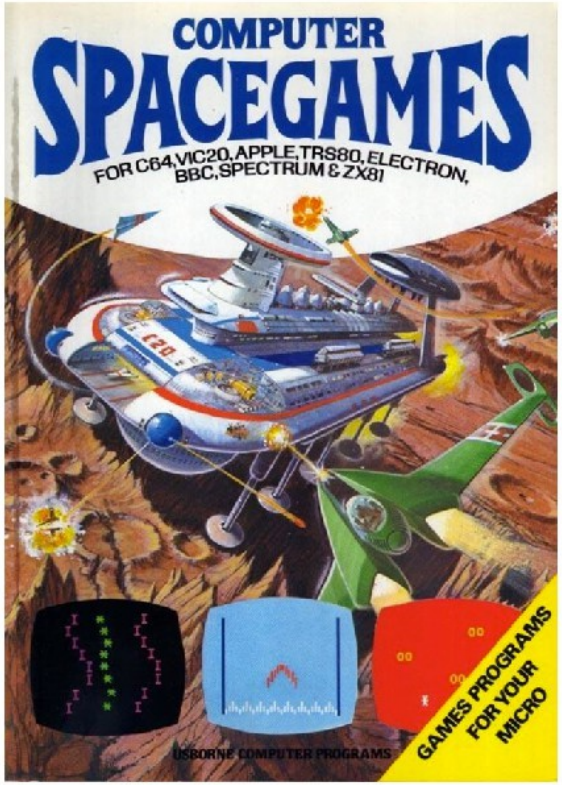


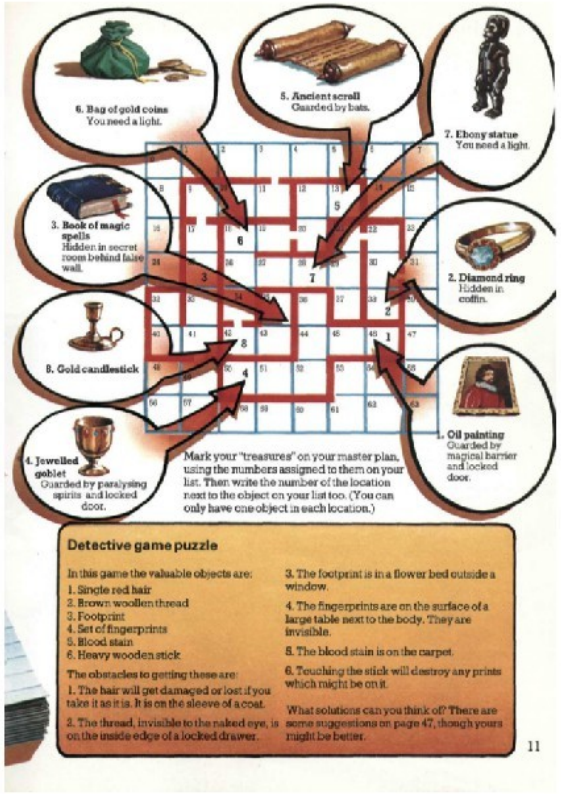







WRITE YOUR OWN
ADVENTURE PROGRAMS
FOR YOUR MICROCOMPUTER
WITH PROGRAM LISTINGS FOR SPECTRUM (48K), TRS-80, APPLE, DRAGON, VIC, ZX81, ORIC, BBC, & PET
USBORNE HAYES


COMPUTER
SPACEGAMES
FOR C64, VIC20, APPLE, TRS80, ELECTRON, BBC, SPECTRUM & ZX81
GAMES PROGRAMS FOR YOUR MICRO
USBORNE COMPUTER PROGRAMS


6. Bag of gold coins
You need a light.
5. Ancient scroll
Guarded by bats.
7. Ebony statue
You need a light.
3. Book of magic spells
Hidden in secret room behind false wall.
2. Diamond ring
Hidden in coffin.
8. Gold candlestick
1. Oil painting
Guarded by magical barrier and locked door.
4. Jewelled goblet
Guarded by paralysing spirits and locked door.
Mark your "treasures" on your master plan, using the numbers assigned to them on your list. Then write the number of the location next to the object on your list too. (You can only have one object in each location.)
Detective game puzzle
In this game the valuable objects are:
1. Single red hair
2. Brown woollen thread
3. Footprint
4. Set of fingerprints
5. Blood stain
6. Heavy wooden stick
The obstacles to getting these are:
1. The hair will get damaged or lost if you take it as it is. It is on the sleeve of a coat.
2. The thread, invisible to the naked eye, is on the inside edge of a locked drawer.
3. The footprint is in a flower bed outside a window.
4. The fingerprints are on the surface of a large table next to the body. They are invisible.
5. The blood stain is on the carpet.
6. Touching the stick will destroy any prints which might be on it.
What solutions can you think of? There are some suggestions on page 47, though yours might be better.
11

IDEAS JUST COVERED

Functions and libraries

Pre-built library functions such as `Forward`

Looping

Repeat things using `repeat`

Abstraction

Created our own functions such as `square`

Parameters

Allowed for adjustments of numbers such as `length`

ABSTRACTION

Problem

Solution

```
repeat 3 [
  forward 100
  right 120
]
```

Abstraction

```
TO triangle
  repeat 3 [
    forward 100
    right 120
  ]
END
```

Re-Use

ABSTRACTION

Problem

Solution

```
repeat 3 [
  forward 100
  right 120
]
```

Abstraction

```
TO triangle
  repeat 3 [
    forward 100
    right 120
  ]
END
```

Re-Use

```
left 90
repeat 2 [
  triangle
  forward 25
]
```

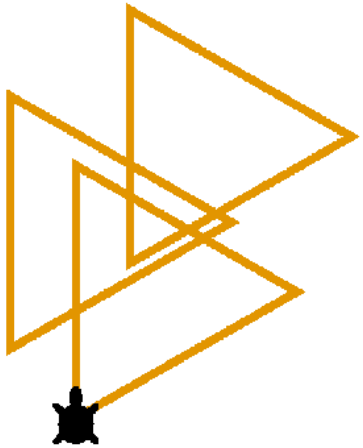
WHY ABSTRACT YOUR CODE?

- Simplifies your code
- Allows yourself (and others) to re-use your work
- Libraries and frameworks provide pre-built abstractions

PARAMETERS

Without parameters

```
TO triangle
  repeat 3 [
    forward 100
    right 120
  ]
END
```

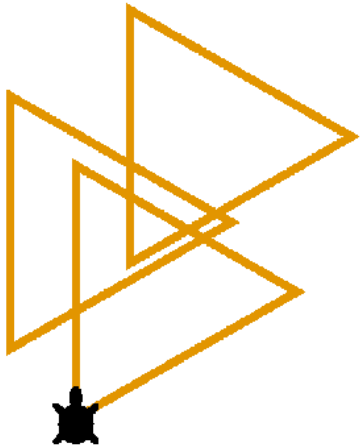


With parameters

PARAMETERS

Without parameters

```
TO triangle
  repeat 3 [
    forward 100
    right 120
  ]
END
```



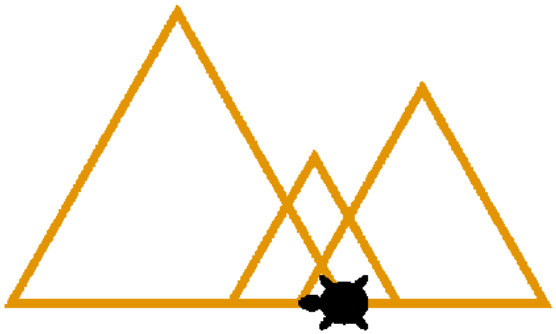
With parameters

```
TO triangle :size
  repeat 3 [
    forward :size
    right 120
  ]
END
```

WHY USE PARAMETERS?

Benefits:

- Makes your code easier to read
- Makes your code more re-usable and generic



```
TO triangle :size
  repeat 3 [
    forward :size
    right 120
  ]
END
```

```
left 90
triangle 80
forward 50
triangle 50
forward 20
triangle 100
```

PROGRAMMING CONCEPTS (USING RUBY)

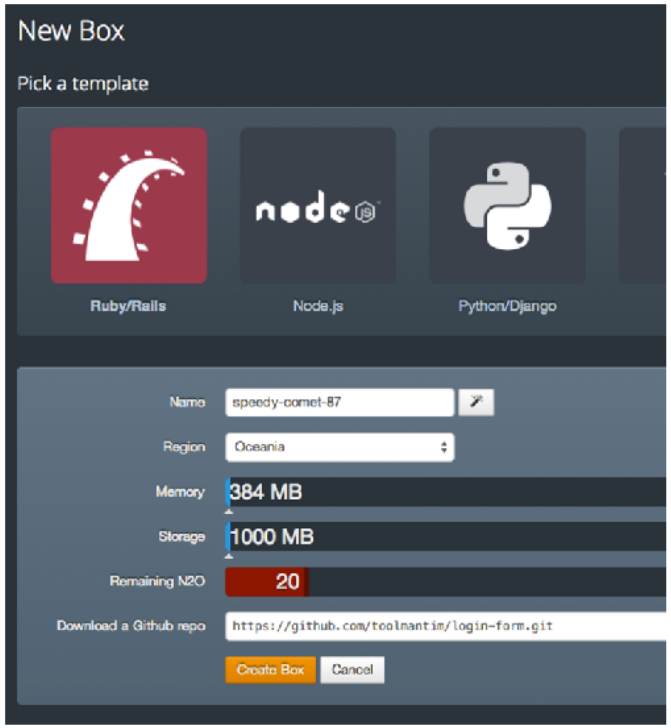
FIRST RUBY PROGRAM

- Open https://nitrous.io/
- Create a new Rails box
- Create a new file hello.rb inside Workspace with:

```
puts "Hello world"
```

- Save the file
- In the console:

```
ruby hello.rb
```



WHAT IS RUBY?

A language designed and developed in 1993
by Yukhiro Matsumoto (aka “Matz”)

"I hope to see Ruby help every programmer in the world to be productive, and to enjoy programming, and to be happy. That is the primary purpose of Ruby language."

VARIABLES
STORING THINGS

```ruby
name = "Sally"
puts name # this will print "Sally"

age = 2013 - 1993
puts age # what will this print?
```

BASIC DATA TYPES

```ruby
puts "Sally"

puts 20

puts [1,2,3]

“Sally” + [1,2,3] # won’t work
```

CONDITIONALS

```ruby
if age < 12
  puts "Child"
elsif age > 12 and age < 20
  puts "Teenager"
else
  puts "Adult"
end
```

CONDITIONALS

```ruby
if name == "Sally"
  puts "User is Sally"
else
  puts "User is not Sally"
end
```

COMPARISON OPERATORS

Operator	Description	Example (a =4 and b= 2)
==	Equal	a == b *false*
!=	Not Equal	a != b *true*
>	Greater than	a > b *true*
<	Less than	a < b *true*
>=	Greater than or equal to	a <= b *false*
<=	Less than or equal to	a <= b *false*
<=>	same value? return 0 less than? return -1 greater than? return 1	a <=> b *1*
.eql?	same value and same type?	1.eql?(1.0) *false*

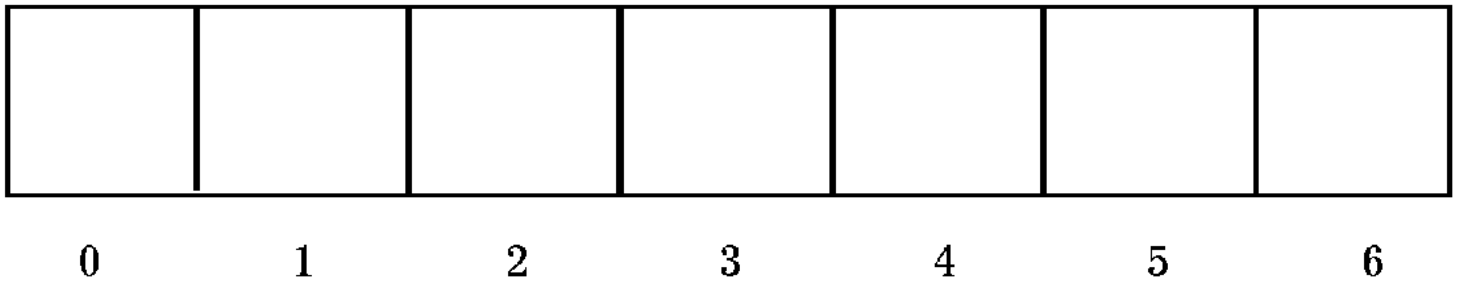
LOOPS
REPEATING THINGS

```ruby
# Print your name 10 times

10.times do
  puts "Don"
end
```

ARRAYS

STORING MULTIPLE THINGS



ARRAYS

STORING MULTIPLE THINGS

```ruby
# Store 3 names
names = ["Don", "Sally", "Roger"]

# Print all the names
names.each do |name|
  puts name
end
```

ARRAYS

STORING MULTIPLE THINGS

```ruby
# Store 3 names
names = ["Don", "Sally", "Roger"]

# Print all the names (with indices)
names.each_with_index do |name, index|
  puts "#{index}: #{name}"
end
```

FUNCTIONS

DEFINING YOUR OWN

```ruby
# Define a function called "say_hello"
def say_hello_to(name)
  puts "Hello #{name}"
end

say_hello_to "Don"   # Prints "Hello Don"
say_hello_to "Sally" # Prints "Hello Sally"
say_hello_to "Roger" # Prints "Hello Roger"
```

LEARNING RUBY

Interactive web learning: http://dash.generalassembly.com/

Another interactive web learning: http://tryruby.org/

Why's Poignant Guide to Ruby: http://mislav.uniqpath.com/poignant-guide/book/

Learn to program with Ruby: http://pragprog.com/book/ltp2/learn-to-program

READING DOCUMENTATION

For built-in Ruby methods: http://ruby-doc.org/

For Ruby gems: http://rubydoc.info/

Reference book: http://pragprog.com/book/ruby4/programming-ruby-1-9-2-0

WEB FRAMEWORKS

WHAT IS A WEB FRAMEWORK?

Toolkit for building web applications

Has library functions for handling:

- Requests
- Responses
- Templates
- Cookies

RUBY WEB FRAMEWORKS

Two main ones in Ruby:

- Rails
  Kitchen sink, pre-built
- Sinatra
  Minimal, put it together yourself

SINATRA

Sinatra has the concept of:

- Routes
- Views (templates)

You add the rest yourself.

RAILS

Rails has the concept of:

- Routes
- Controllers
- Views (templates)
- Models (database)
- Mailers (email)
- and more…

LIBRARIES ARE PACKAGED AS "GEMS"

Rails
Sinatra
Devise
Carrier Wave
Geocoder

"There's a gem for that"

HOW TO INSTALL GEMS

```
$ gem install rails
$ gem install geocoder
```

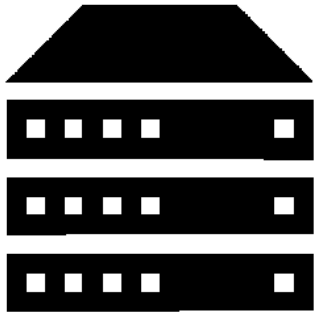
BUNDLER

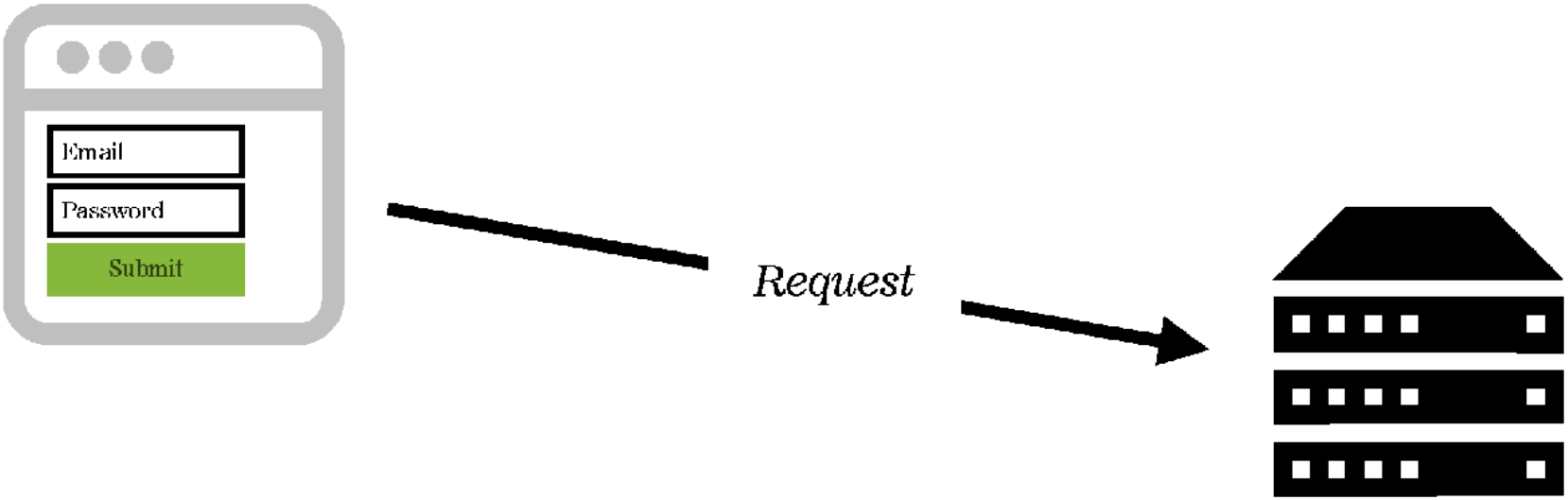
You normally put a list of all the gems you use for your project in a Gemfile, and use Bundler to install all the gems for you.

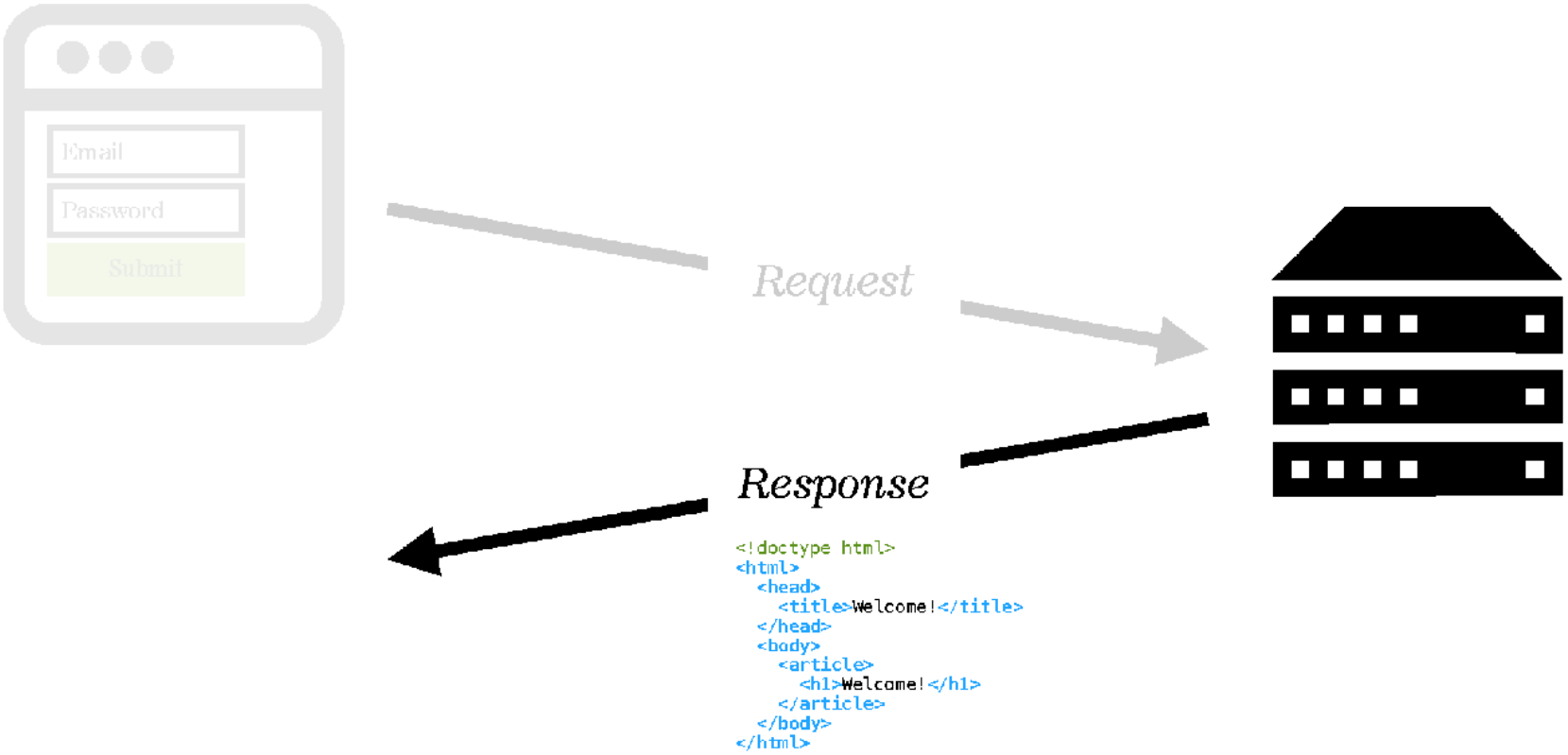
```
$ bundle install
```

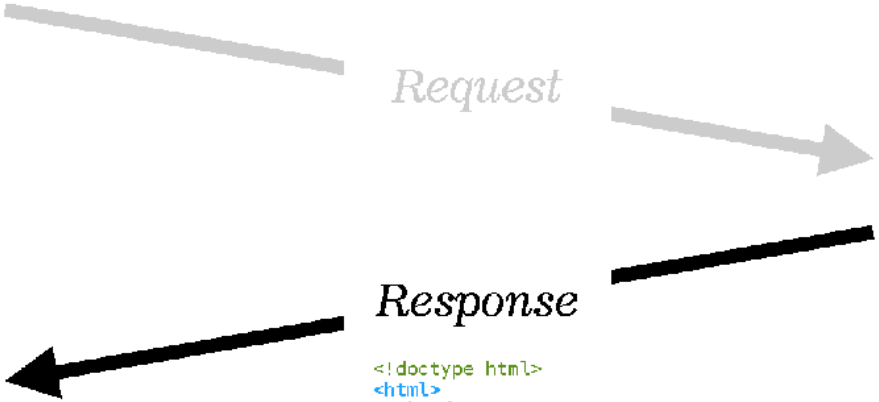
RECAP

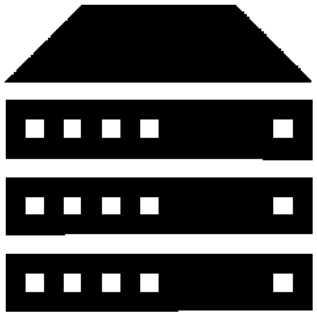

Email
Password
Submit




Email
Password
Submit
Request


Email
Password
Submit
Request
Response
<!doctype html>
<html>
<head>
<title>Welcome!</title>
</head>
<body>
<article>
<h1>Welcome!</h1>
</article>
</body>
</html>






```
<!doctype html>
<html>
  <head>
    <title>Welcome!</title>
  </head>
  <body>
    <article>
      <h1>Welcome!</h1>
    </article>
  </body>
</html>
```

REQUEST/RESPONSE DEVELOPER TOOLS

http://login-form.bunts.io

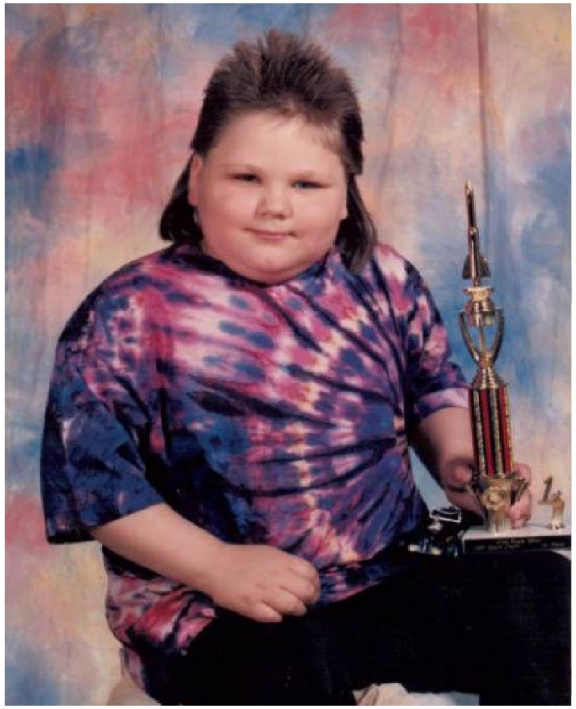
Source code:

http://github.com/buntine/login-form

APIS

Use APIs

- *Grab a picture from Reddit.com*
  using their API
- *Email that picture to a friend*
  using the Mandrill API

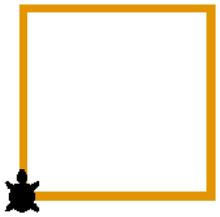


EXERCISE: TURTLE

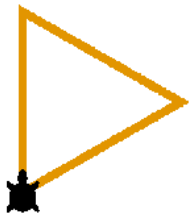
Imagine there was a turtle that understood the instructions (functions) on the right.

Break up into *groups of 2*

Write down a recipe (routine) for each of the pictures below, using the functions available.



Square



Triangle

Double Square

Documentation

Available turtle functions

- `Forward` x `metres`
- `Turn right` x `degrees`
- `Turn left` x `degrees`
- `Repeat` x `times [`
  - *instructions*

  `]`

ALGORITHMS

AKA FUNCTIONS, METHODS, ROUTINES

Square

```
Repeat 4 times [
  Forward 5 metres
  Turn right 90 degrees
]
```

Triangle

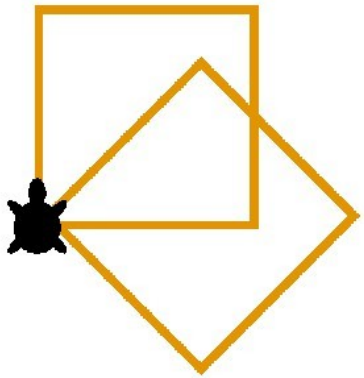
```
Repeat 3 times [
  Forward 5 metres
  Turn right 120 degrees
]
```

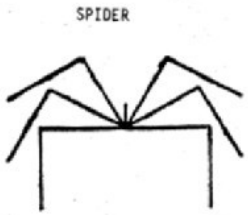
Double Square

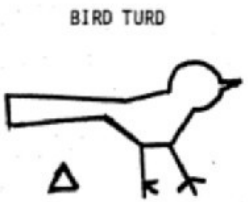
```
Repeat 2 times [
  Repeat 4 times [
    Forward 5 metres
    Turn right 90 degrees
  ]
  Turn right 45 degrees
]
```

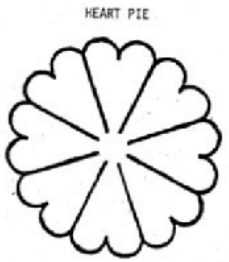
A CREATIVE PROCESS

Understanding the problem
Exploring solutions
Pattern recognition
Abstraction
Sharing ideas









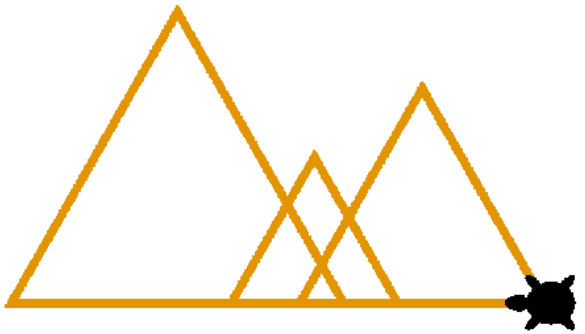
WHY ABSTRACT YOUR CODE?

- Simplifies your code
- Allows yourself (and others) to re-use your work
- Libraries and frameworks provide pre-built abstractions

WHY USE PARAMETERS?

Benefits:

- Makes your code easier to read
- Makes your code more re-usable and generic



```
TO triangle :size
  repeat 3 [
    forward :size
    right 120
  ]
END
```

```
left 90
triangle 80
forward 50
triangle 50
forward 20
triangle 100
```

FIRST RUBY PROGRAM

- Open https://nitrous.io/
- Create a new Rails box
- Create a new file hello.rb inside Workspace with:

```
puts "Hello world"
```

- Save the file
- In the console:

```
ruby hello.rb
```

WHAT IS A WEB FRAMEWORK?

Toolkit for building web applications

Has library functions for handling:

- Requests
- Responses
- Templates
- Cookies

RAILS AND SINATRA

Rails has the concept of:

- Routes
- Controllers
- Views (templates)
- Models (database)
- Mailers (email)
- and more...