Building blocks of code

Good morning! This is a talk that combines code with one of my other favourite things: coffee.

It's a short tour of some of the main concepts you'll come across while writing code.

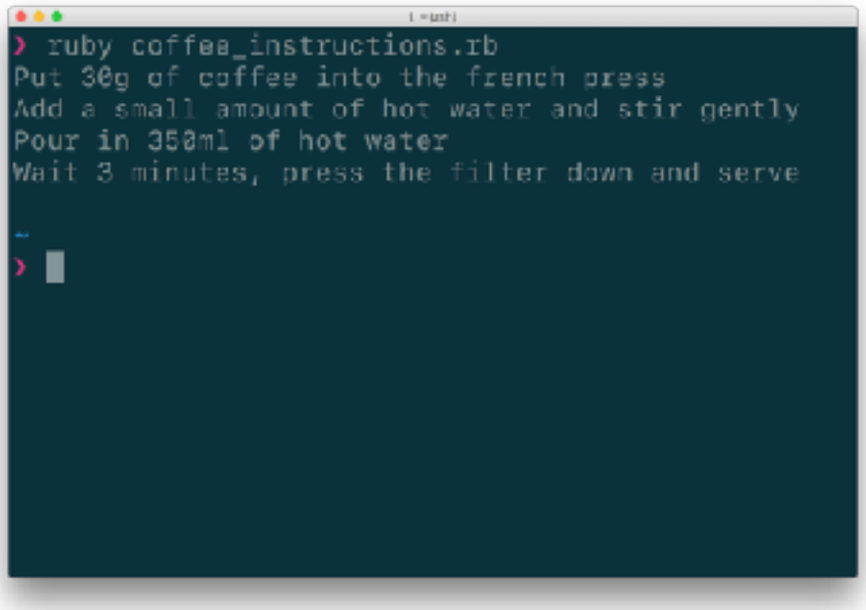
Lines of code

```
puts "Put 30g of coffee into the french press"
puts "Add a small amount of hot water and stir gently"
puts "Pour in 350ml of hot water"
puts "Wait 3 minutes, press the filter down and serve"
```

As Rob mentioned in his talk, writing code is a way of giving instructions to a computer.

Here is some code that displays how to make coffee. This example is using Ruby, but doing the same thing using Python is pretty similar. After writing some code we call the process of asking the computer to carry out the instructions "running the program". When a program is run the computer will go through each line of code starting from the top and working its way down.

Running the program

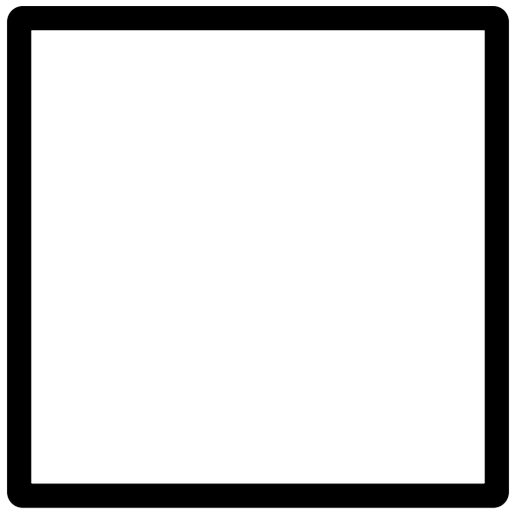


Rails Girls ♡ django girls

So the result is like this.

One of the shortcomings of this program is that each and every time we run it exactly the same instructions are displayed. There's no way to personalise the instructions to adjust the strength of the coffee, or to make a different amount of coffee. Let's fix that using some variables.

Variables



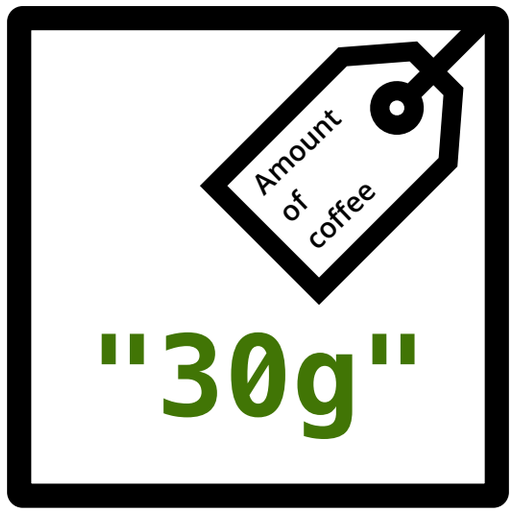
Rails Girls ♡ django girls

A variable is like a box that can be used to hold some information needed by a program.

Variables

For instance we can put the amount of coffee we'd like to use in a variable.

Variables



Rails Girls ♡ django girls

And then attach a little label to that variable so that we've got a name that can be used to refer to it

Variables

```ruby
amount_of_coffee = "30g"
amount_of_water = "350ml"
brewing_time = "3 minutes"

puts "Put #{amount_of_coffee} of coffee into the french press"
puts "Add a small amount of water and stir gently"
puts "Pour in #{amount_of_water} of hot water"
puts "Wait #{brewing_time}, press the filter down and serve"
```

Rails Girls ♡ djangogirls

Here's what that looks like in code

There's some new lines to create some variables

And then the existing code has been updated to use the variables in the places where we previously had the fixed amounts.

Conditions

```ruby
if time.hour > 16
  coffee_type = "decaf"
else
  coffee_type = "regular"
end

puts "Put #{amount_of_coffee} of #{coffee_type} coffee
into the french press"
```

The next feature to look at is "conditions". A condition lets the computer make a decision.

Here, the program will do something different depending on the time of day

Conditions

```
if time.hour > 16
  coffee_type = "decaf"
else
  coffee_type = "regular"
end

puts "Put #{amount_of_coffee} of #{coffee_type} coffee
into the french press"
```

If the condition is true then this part of the code will happen

Conditions

```
if time.hour > 16
  coffee_type = "decaf"
else
  coffee_type = "regular"
end

puts "Put #{amount_of_coffee} of #{coffee_type} coffee
into the french press"
```

If not then this line of code will be run instead

Conditions

```
if time.hour > 16
  coffee_type = "decaf"
else
  coffee_type = "regular"
end

puts "Put #{amount_of_coffee} of #{coffee_type} coffee
into the french press"
```

And then either way the program will continue from the next line after the condition

Methods

```
puts "Put 30g of coffee into the french press"
puts "Add a small amount of hot water and stir gently"
puts "Pour in 350ml of hot water"
puts "Wait 3 minutes, press the filter down and serve"
```

As a program grows it's helpful to split it up into separate blocks of code knows as either "methods" or "functions".

A method is created and given a name...

Methods

```ruby
def display_coffee_instructions
  puts "Put 30g of coffee into the french press"
  puts "Add a small amount of hot water and stir gently"
  puts "Pour in 350ml of hot water"
  puts "Wait 3 minutes, press the filter down and serve"
end
```

Rails Girls ♡ django girls

Like this

There's a couple of reasons to use methods. They help give structure to the program and prevent it from just becoming a single long stream of code. Perhaps more importantly they're really useful if we'd like to do the same thing more than once in our program. Once we've created a method we tell the computer to run the code inside that method with just a single line of code

Methods

```
display_coffee_instructions
```

Like so

And if we'd like to make 2 lots of coffee, instead of duplicating *all* of the instructions we can just do this

Methods

```
display_coffee_instructions
display_coffee_instructions
```

And that brings us to the last feature I'd like to introduce today. If there's something we'd like to tell the computer to repeat a lot of times we can use a loop. So instead of

Functions

```
display_coffee_instructions
display_coffee_instructions
display_coffee_instructions
display_coffee_instructions
display_coffee_instructions
```

this.

We can write something like:

Loops

```
5.times do
  display_coffee_instructions
end
```

This

But what makes loops super-powerful is that if, for whatever reason, we don't know in advance how many times we need to repeat the loop they can be combined with variables, like this:

Loops

```
number_of_guests.times do
  display_coffee_instructions
end
```

Or used with a condition, like this

Loops

```
while more_people_need_coffee? do
  display_coffee_instructions
end
```

The building blocks

- Variables
- Conditions
- Functions
- Loops

So those are the building blocks. Hopefully during todays workshops you'll get to try out using all of these and see how they fit together.

Thank you