**Lambdas**

A lambda is essentially a function without a name, as with everything in Ruby they are also objects.

Lambdas also have methods which can be assigned to variables!

You can also pass in arguments using pipes.

e.g. l = lambda do |string|

l.call(“String here”)

Remember, it is essentially just a method without a name, in most cases it will just be assigned to a variable.

**Lambdas v Blocks**

A block is a piece of code that can’t be stored in a variable and IS NOT an object.

There is no lambda present but there is a new object called yield present.

The most common use for lambdas is passing over blocks to methods.

e.g. def calculate(\*numbers)

yield(\*numbers)

end

calculate(2, 3) { |a, b| a + b } returns 5 ✔

**Getting Modular**

Modules are similar to classes but they only hold behaviour and not state & behaviours like a class.

Instances of modules cannot be created so the methods cannot be called directly. The module must be made part of a class and then called as part of the class.

An example would be a warmup module within a gym class and a dojo class. Each class would have its own distinctive methods but both would also require the warmup module to be included in both classes.

A ‘include WarmUp’ within the class would include the functionality of this method then.

**Hierarchy**

All classes within Ruby are an instance of Class and all modules are an instance of Module.

Touching briefly on inheritance, all classes are modules as Module is the superclass of Class. Object > Module > Class

**Name Spacing**

Name spacing allows classes or modules with the same name to co exist without classing with each other!

e.g. module Perimeter

class Array

def initialize

@size = 400

end

end

end

our\_array = Perimeter::Array.new

ruby\_array = Array.new

Namespacing can save with issues that occur with multiple libraries that may be used in a program.