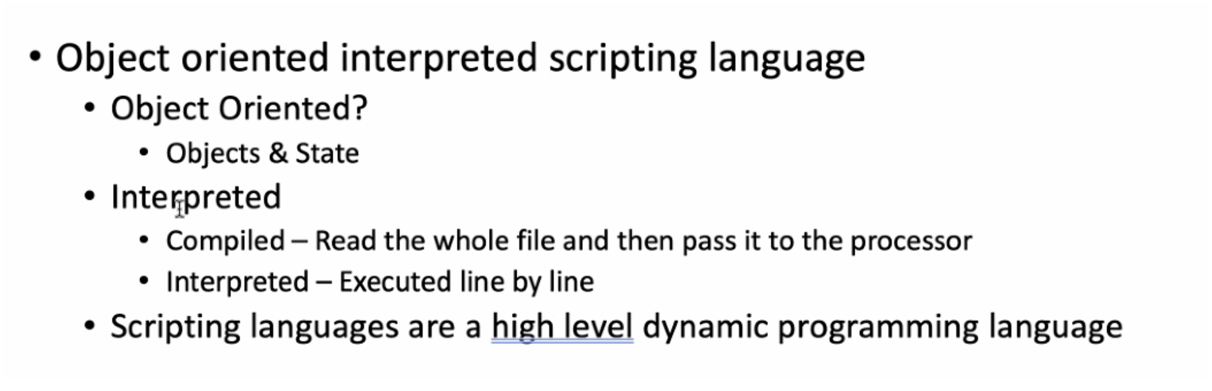
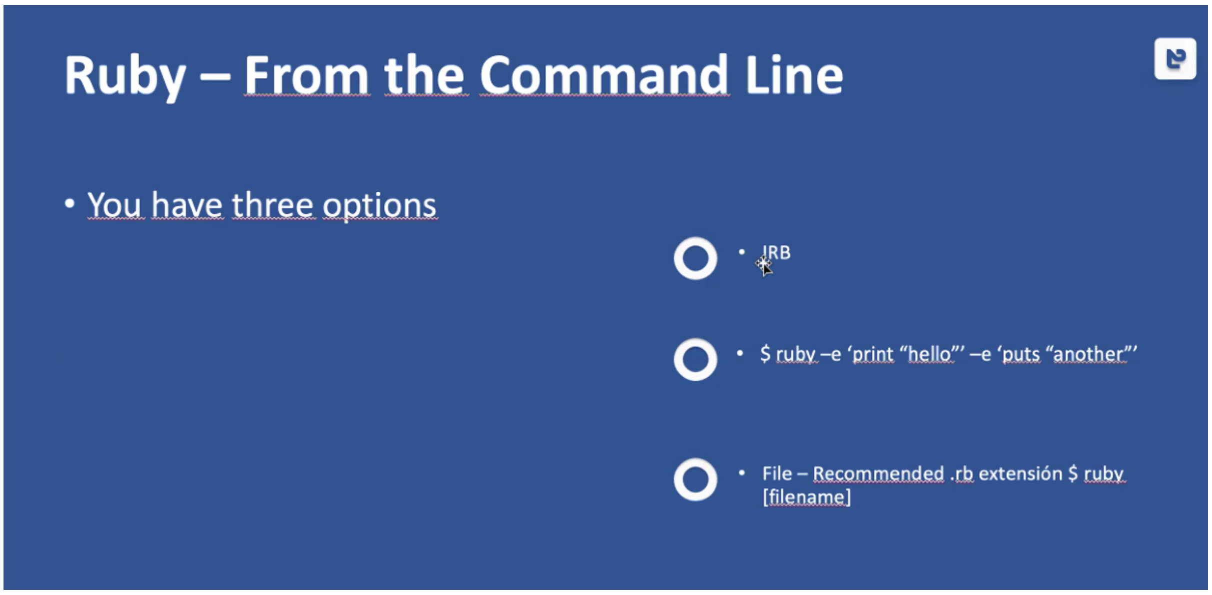
Veremos ruby 3.0

Ruby is

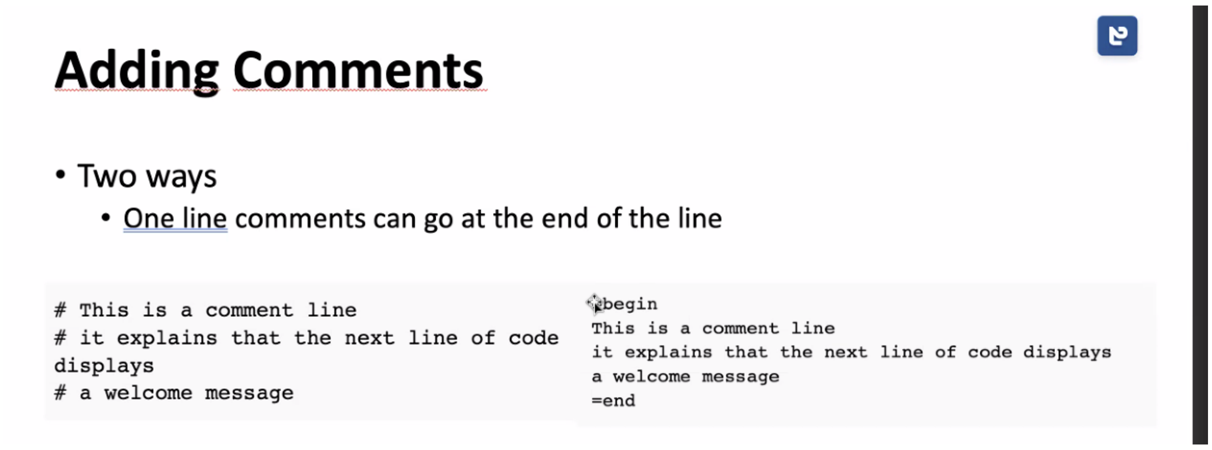


- Object oriented interpreted scripting language
  - Object Oriented?
    - Objects & State
  - Interpreted
    - Compiled – Read the whole file and then pass it to the processor
    - Interpreted – Executed line by line
  - Scripting languages are a high level dynamic programming language



Ruby – From the Command Line

- You have three options
  - IRB
  - $ ruby –e 'print "hello"' –e 'puts "another"'
  - File – Recommended .rb extensión $ ruby [filename]



Adding Comments

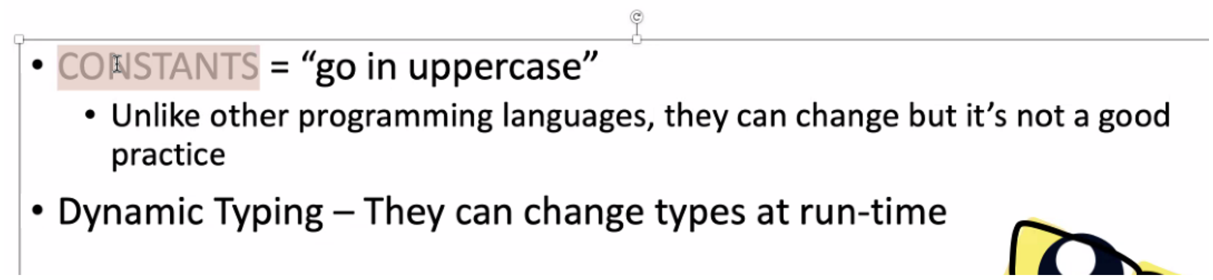
- Two ways
  - One line comments can go at the end of the line

```
# This is a comment line
# it explains that the next line of code displays
# a welcome message
```

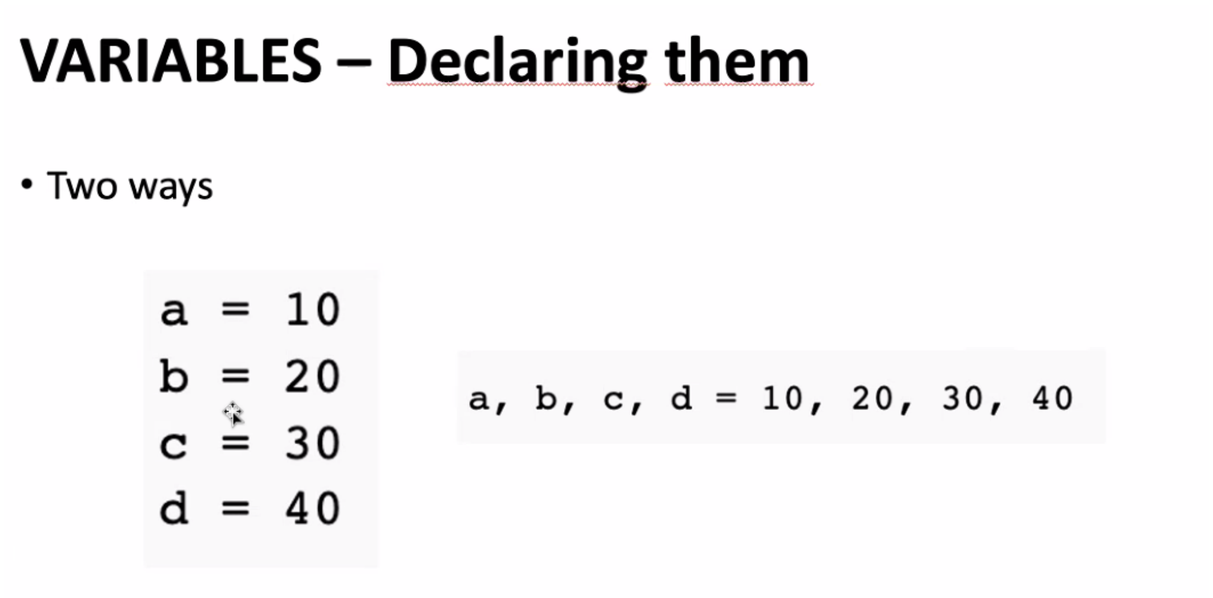
```
=begin
This is a comment line
it explains that the next line of code displays
a welcome message
=end
```

It stills not a good practice to use commentaries, your code should be self explanatory by names…

Constants
Go on uppercase. They can change but it is not good practice.



- CONSTANTS = “go in uppercase”
  - Unlike other programming languages, they can change but it’s not a good practice
- Dynamic Typing – They can change types at run-time



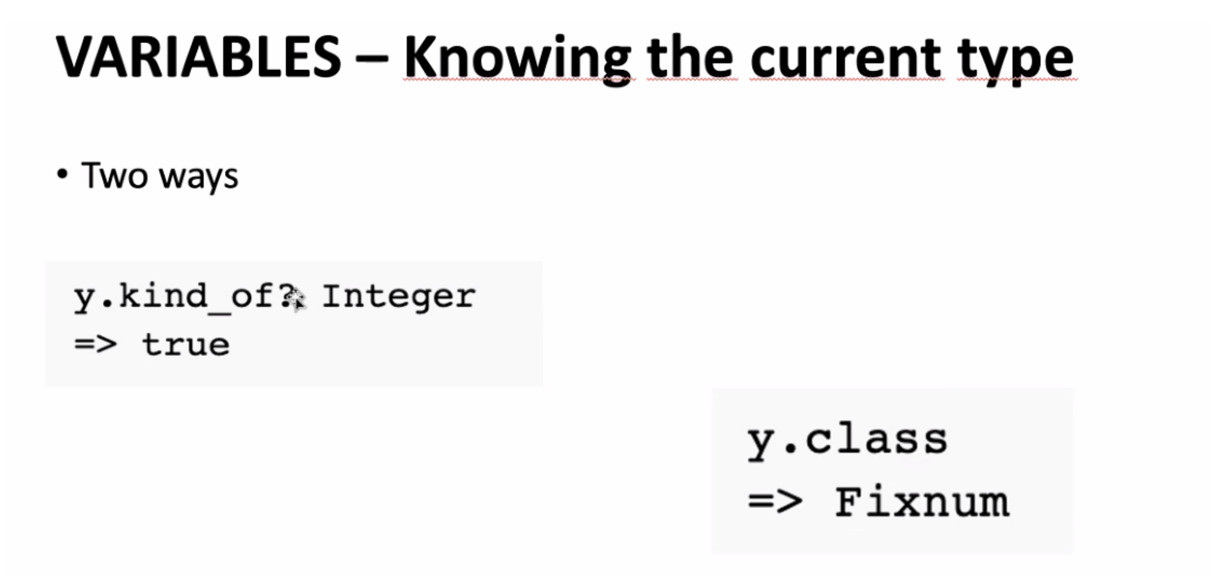
VARIABLES – Declaring them

- Two ways

```
a = 10
b = 20
c = 30
d = 40
```

```
a, b, c, d = 10, 20, 30, 40
```

It is a matter of preference which one to use.
But the first one is more readable.



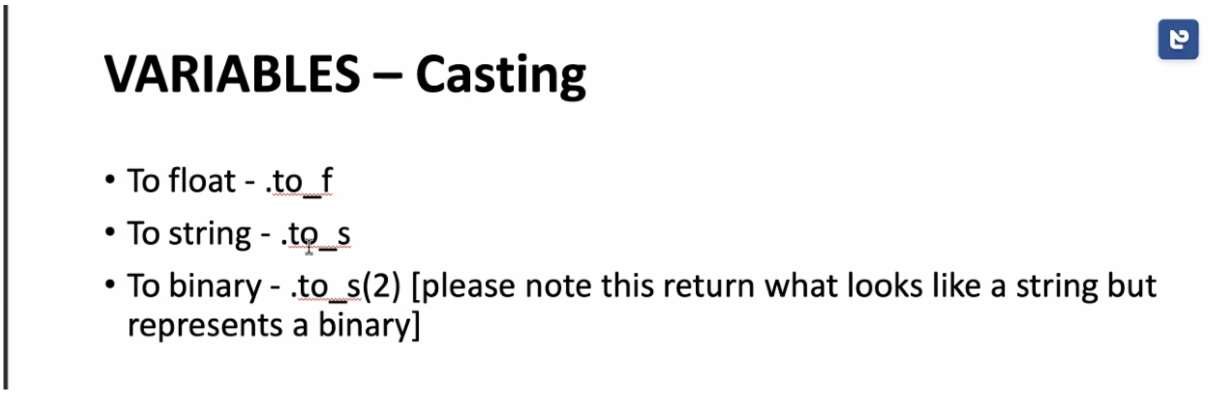
VARIABLES – Knowing the current type

- Two ways

```
y.kind_of? Integer
=> true
```

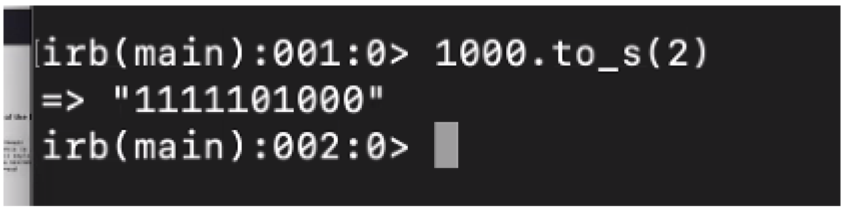
```
y.class
=> Fixnum
```

is_a? is like .kind_of, but is_a also do another thing?



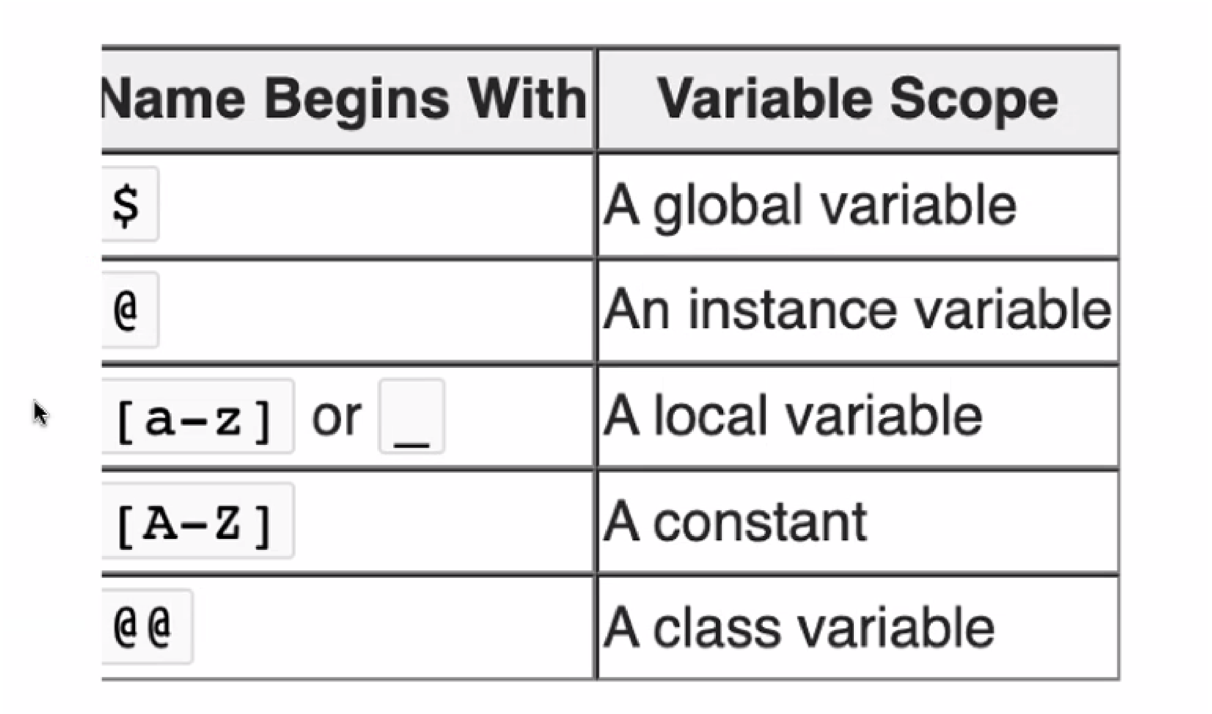
VARIABLES – Casting

- To float - .to_f
- To string - .to_s
- To binary - .to_s(2) [please note this return what looks like a string but represents a binary]



```
irb(main):001:0> 1000.to_s(2)
=> "1111101000"
irb(main):002:0>
```

To_s can convert it to binary?



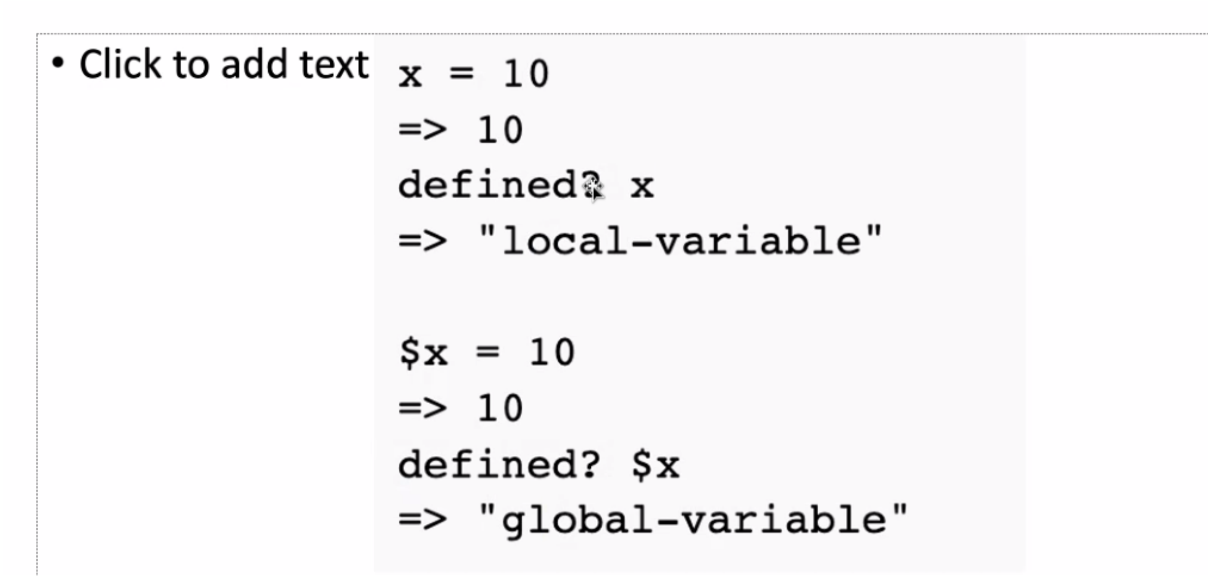
Name Begins With	Variable Scope
$	A global variable
@	An instance variable
[a-z] or _	A local variable
[A-Z]	A constant
@@	A class variable

$global variable -> most useful when we are using something stored in our session.
Communicate something from the controller to the module (not good practice but necessary).

local variable -> if put inside if, only available inside if.

constants -> declare normally at the beginning of the class.

@@ class variable -> static variable, also if inherited, the child will have the same value.



• Click to add text

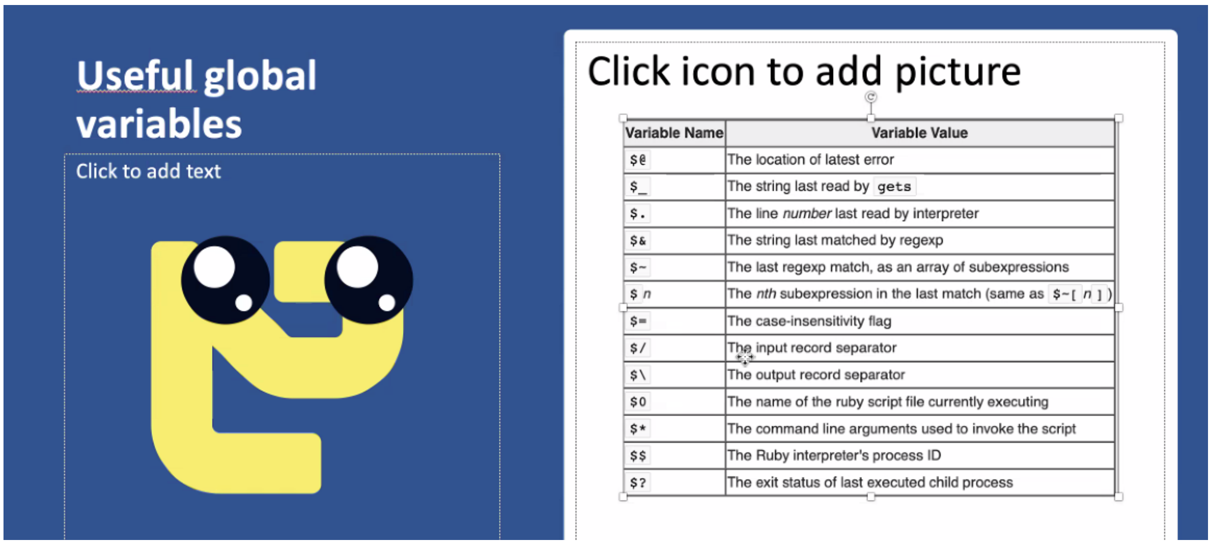
```
x = 10
=> 10
defined? x
=> "local-variable"

$x = 10
=> 10
defined? $x
=> "global-variable"
```

.defined? -> tell you what type of variable it is.

<u>Why do some methods have “?”</u>
usually the ones that end with ?, returns a true or false, or the type, etc.

<u>The “!”</u>
some methods do not modify the variable itself, only return another value.
the ones that end with “!” wil always change the variable

<u>Clean irb console</u>
control + L

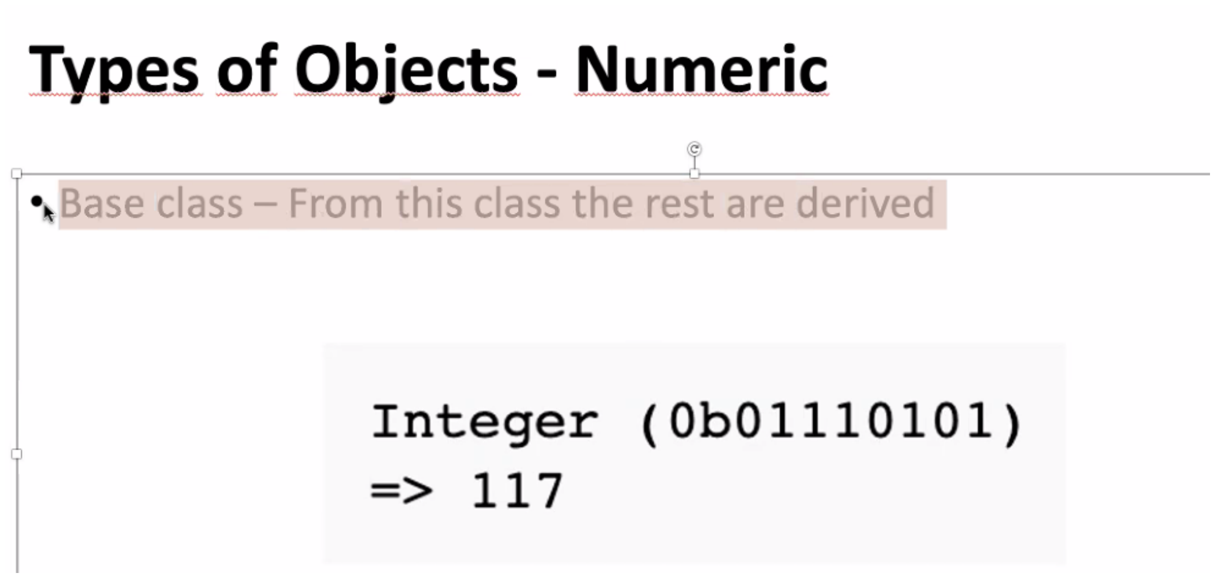


Useful global variables

Click to add text

Click icon to add picture

Variable Name	Variable Value
`$@`	The location of latest error
`$_`	The string last read by `gets`
`$.`	The line *number* last read by interpreter
`$&`	The string last matched by regexp
`$~`	The last regexp match, as an array of subexpressions
`$` *n*	The *nth* subexpression in the last match (same as `$~[` *n* `]`)
`$=`	The case-insensitivity flag
`$/`	The input record separator
`$\`	The output record separator
`$0`	The name of the ruby script file currently executing
`$*`	The command line arguments used to invoke the script
`$$`	The Ruby interpreter's process ID
`$?`	The exit status of last executed child process



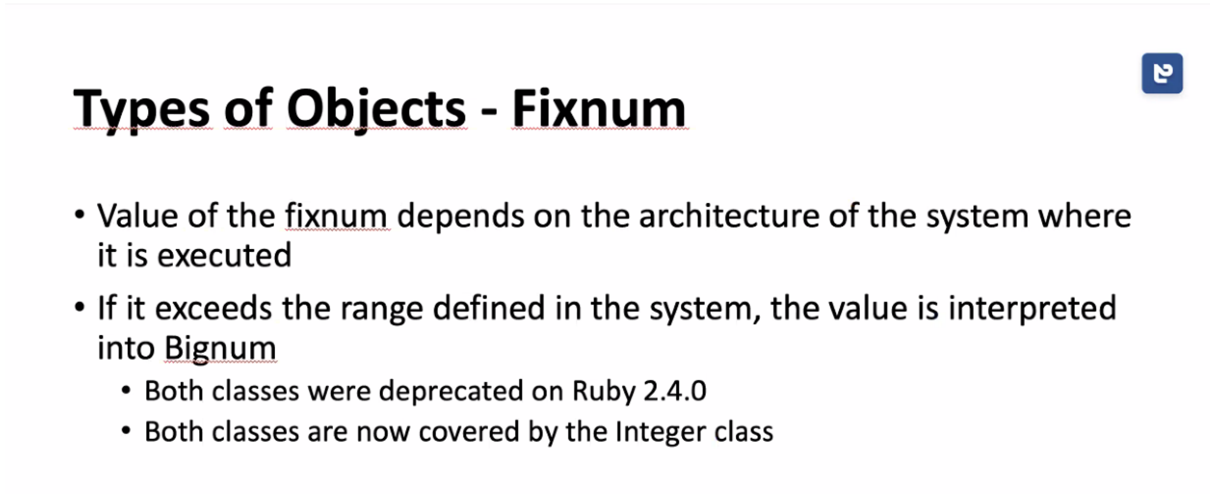
Types of Objects - Numeric

- Base class – From this class the rest are derived

```
Integer (0b01110101)
=> 117
```

All classes in ruby, specially the numeric ones, inherited from that one.

Fixnum is deprecated
We only use integers now.
It is computer architecture dependent.
Only maintainer because of code maintainament.



Types of Objects - Fixnum

- Value of the fixnum depends on the architecture of the system where it is executed
- If it exceeds the range defined in the system, the value is interpreted into Bignum
  - Both classes were deprecated on Ruby 2.4.0
  - Both classes are now covered by the Integer class

Global variables
You can see how they work, defining one in a method, executing it, and calling it on another method.

Also try other ideas.
Also creating in the controller and then using it on the module.