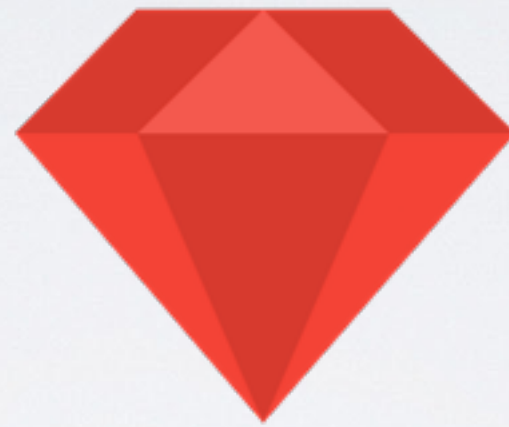


DESAFIO **LATAM**

RUBY



Introducción a Ruby

¿QUÉ APRENDEREMOS HOY?

- Entenderemos las bases de ruby
- Aprenderemos una buena forma de instalar ruby y manejar multiples versiones
- Tipos de datos básicos
- Estructuras de control (if unless / while)

EN ESPECÍFICO VEREMOS:

- ☐ Instalación de ruby
- ☐ Acceso al Intérprete
- ☐ Identificadores
- ☐ Literales
- ☐ Expresiones de control (if, do, while)

¿QUÉ ES RUBY?

- Ruby es un lenguaje de programación creado en la década de los 90's por Yukihiro Matsumoto
- La fecha de inicio es ambigua por que su primera versión fue en 1993 pero la versión 1.0 no fue liberada hasta 1996.



INSTALANDO RUBY

- Existen múltiples versiones de ruby, y puede ser que para trabajar en un proyecto específico tengamos que volver a una versión anterior, para solucionar el problema existe RVM (ruby virtual machine).
- Para instalar RVM <https://rvm.io>
- con RVM instalado: **rvm install 2.3.1**
- Para convertir a 2.3.1 en la versión por defecto
 - **rvm use 2.3.1 --default**

¿QUÉ VERSIÓN DE RUBY INSTALAR?

- La última versión a la hora de la construcción de este curso es la 2.5.0, sin embargo este curso ha sido creado y probado con la versión 2.3.1, recomendamos la **2.3.1** o 2.3.3

REVISANDO LA INSTALACIÓN

- Podemos saber si estamos ocupando la versión de ruby de rvm al hacer:

```
$ which ruby
```



```
/Users/gonzalosanchez/.rvm/rubies/ruby-2.3.1/bin/ruby
```

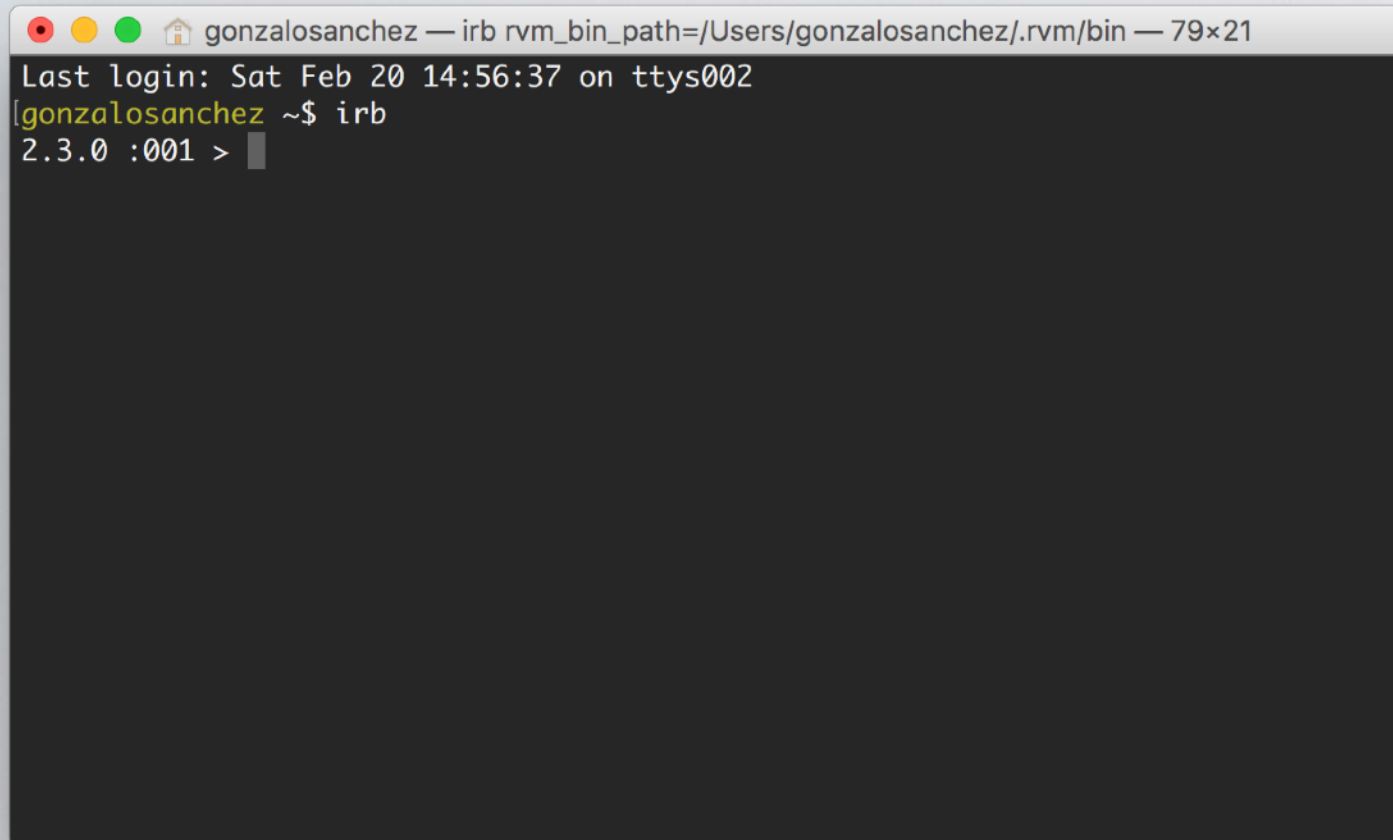

CHECKPOINT

- ☒ Instalación de ruby
- ☐ Acceso al Intérprete
- ☐ Identificadores
- ☐ Literales
- ☐ Expresiones de control (if, do, while)

VENTAJAS DE RUBY

- Interpretado
- Sintaxis sencilla y elegante
- Meta Programming
- Ruby on Rails

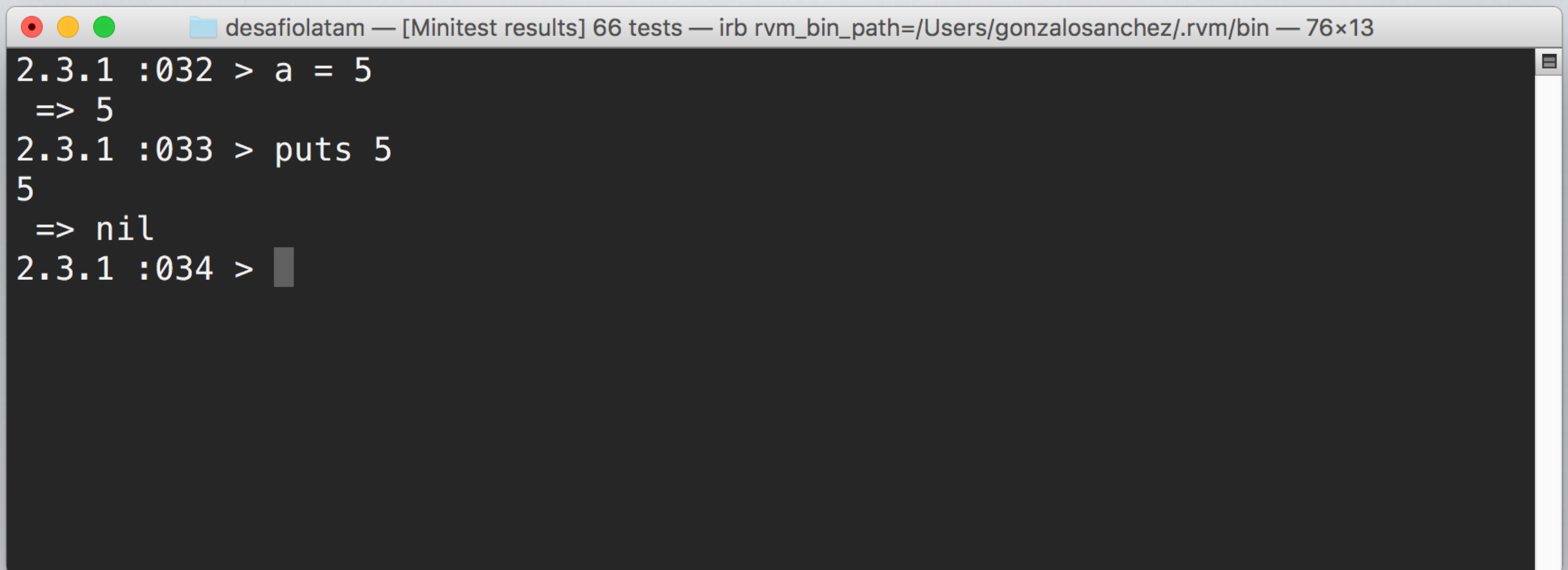
EL INTÉRPRETE



```
gonzalosanchez — irb rvm_bin_path=/Users/gonzalosanchez/.rvm/bin — 79x21
Last login: Sat Feb 20 14:56:37 on ttys002
gonzalosanchez ~$ irb
2.3.0 :001 > 
```

Sirve para hacer
pruebas interactivas
con ruby

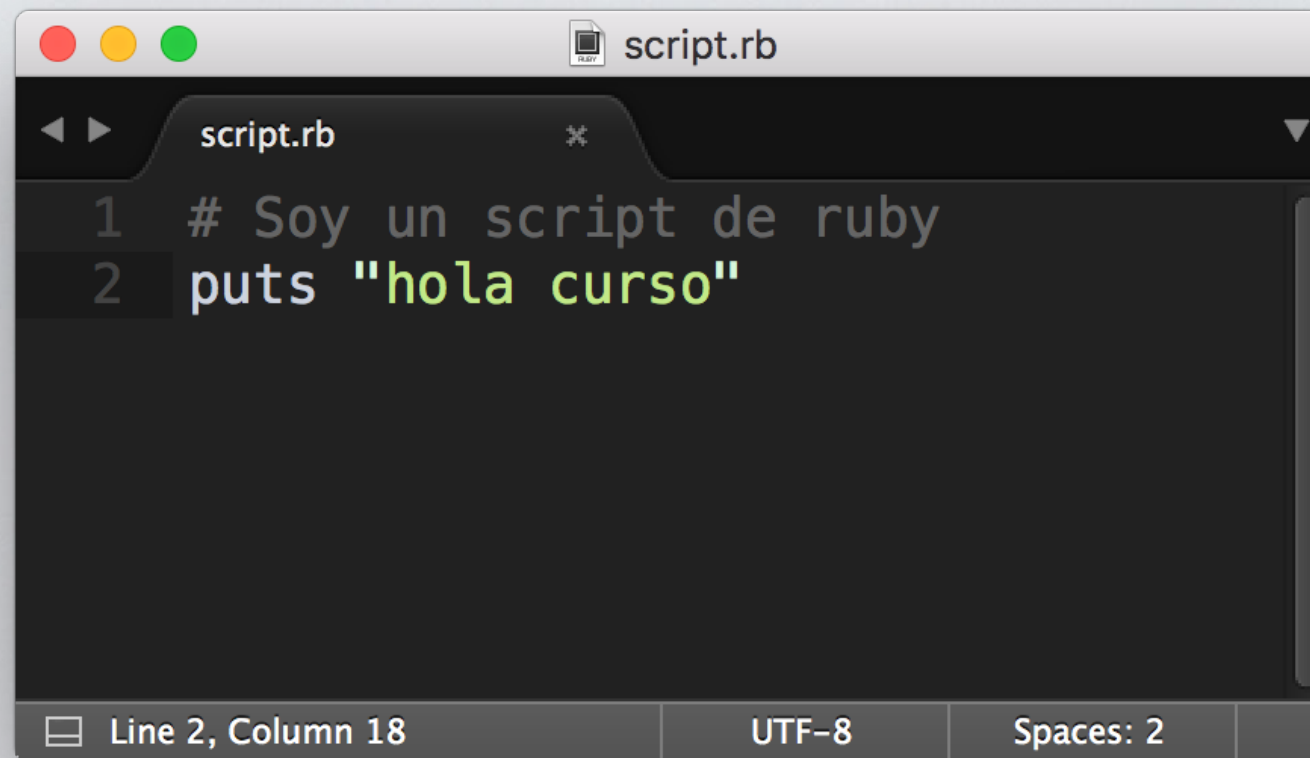
DENTRO DEL INTÉRPRETE



```
desafiolatam — [Minitest results] 66 tests — irb rvm_bin_path=/Users/gonzalosanchez/.rvm/bin — 76x13
2.3.1 :032 > a = 5
=> 5
2.3.1 :033 > puts 5
5
=> nil
2.3.1 :034 >
```

Al escribir `a = 5`, se asigna 5 a la variable `a`. El intérprete además te muestra el resultado de la última línea: al poner `puts 5`, es mostrar en pantalla 5, pero el resultado de esa operación es un valor nulo, por eso después muestra nulo.

LOS SCRIPTS



```
1 # Soy un script de ruby
2 puts "hola curso"
```

Line 2, Column 18 UTF-8 Spaces: 2

Se pueden crear archivos .rb y correr con ruby desde el terminal

```
$ ruby archivo.rb
```

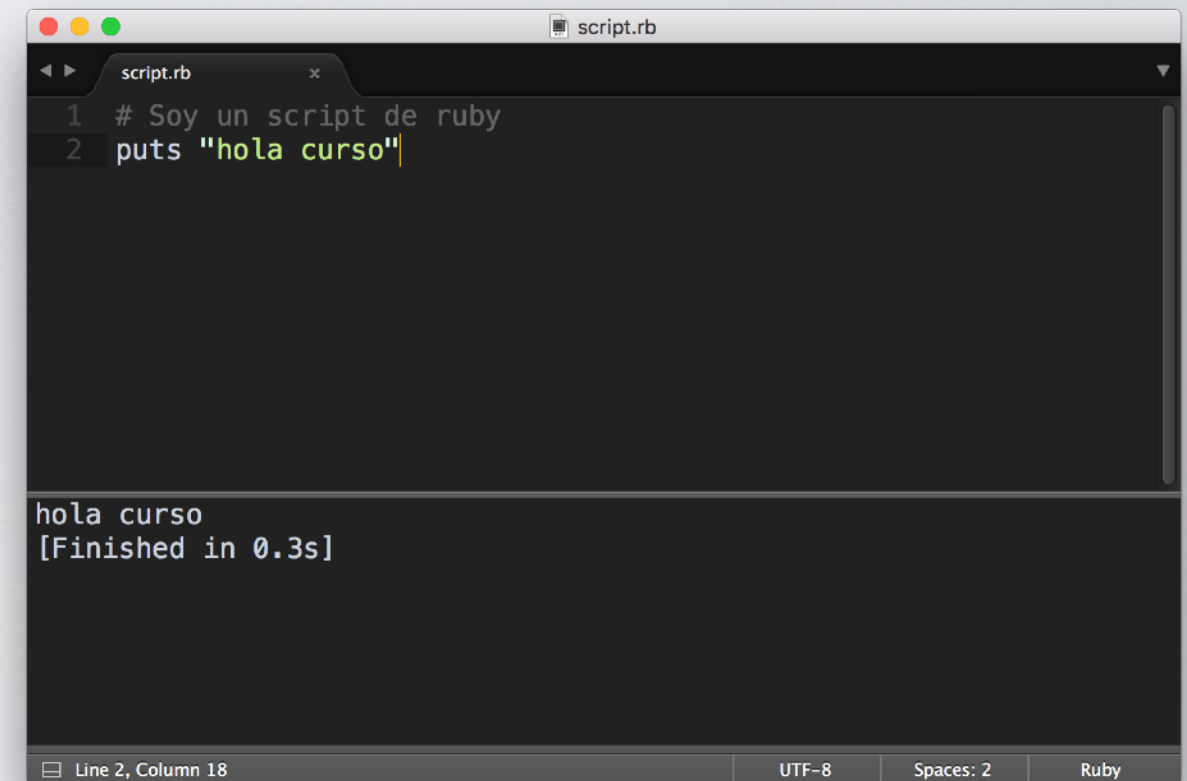
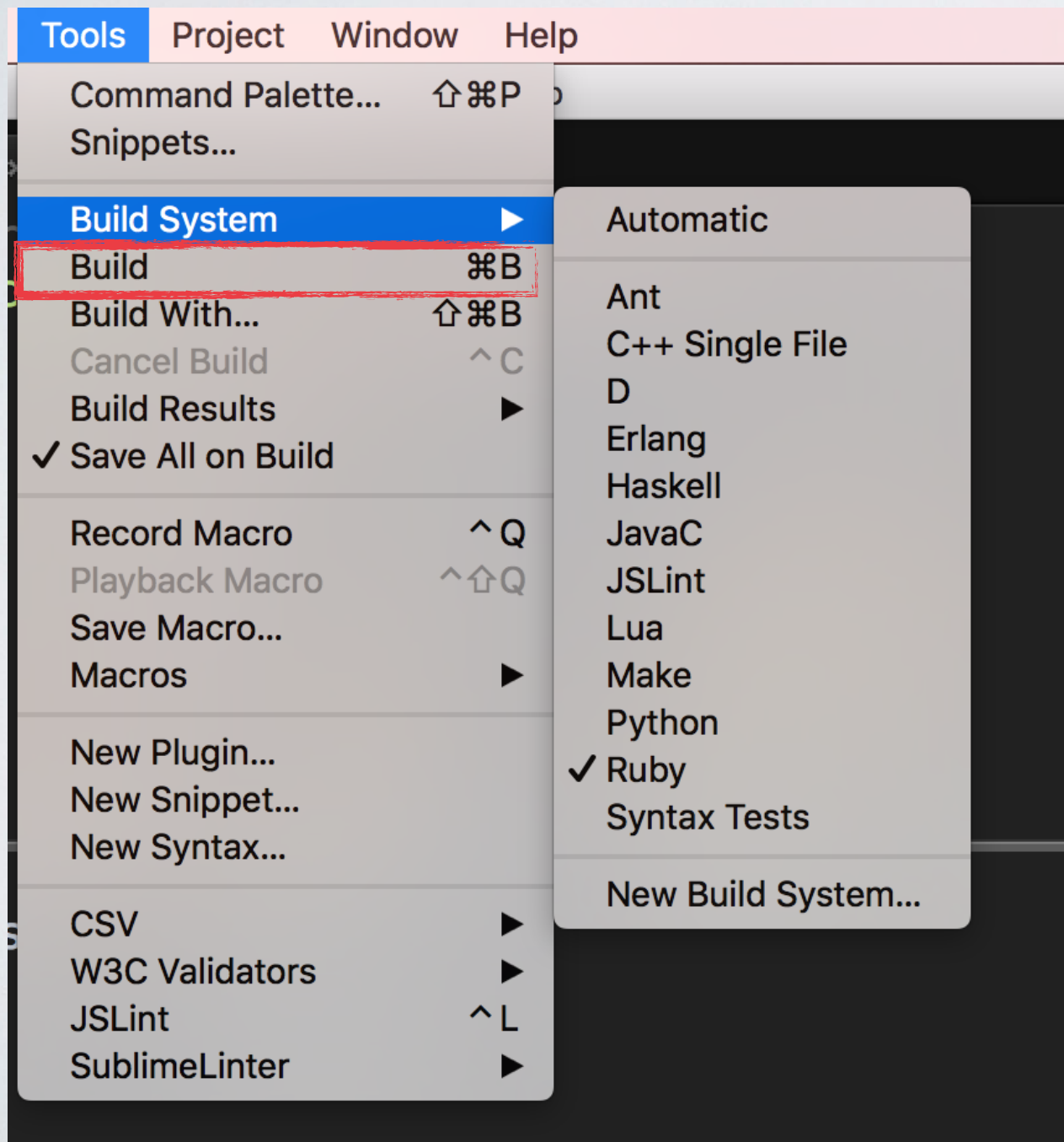
PARA SALIR DEL INTÉRPRETE OCUPAMOS EXIT

Se entra con **irb**
para salir **exit**

CHECKPOINT

- ☒ Instalación de ruby
- ☒ Acceso al Intérprete
- ☐ Identificadores
- ☐ Literales
- ☐ Expresiones de control (if, do, while)

DESDE SUBLIME



ANTES DE AVANZAR

- Recordemos un poco sobre como un computador lee un lenguaje de programación

¿CÓMO UN COMPUTADOR LEE UN LENGUAJE DE PROGRAMACIÓN?

- Todo lenguaje de programación tiene una estructura léxica, este es un set de reglas que define como debe escribirse el programa.
- Para eso el lenguaje lee el código y lo separa en tokens, esos tokens pueden ser comentarios, literales, identificadores y palabras reservadas y todo lo que el lenguaje especifique.

** **Léxico** es el conjunto de palabras que conforman un determinado [lecto](#) y, por extensión, también se denomina así a los diccionarios que los recogen

EJEMPLOS DE ESTAS REGLAS

- Sensibilidad a las mayúsculas, a != A
- Indentación (sangrado) o termino de líneas con ;
- Espacios en blanco y saltos de líneas
- Comentarios (texto que es omitido al leer el código)
- Literales (valores o textos, como por ejemplo 1.2, o "hola")
- Identificadores (por ejemplo las variables o los nombres de las funciones)
- Palabras reservadas (start, stop, break, while, if)

IDENTIFICADORES EN RUBY

- **Los identificadores** en ruby **son** los **nombres de las variables**, métodos y clases.
- Un identificador válido es aquel que empieza con una letra (de la a la z), puede empezar con un guión abajo y puede ser seguido por números.
- Son sensibles a las mayúsculas
- Un identificador válido **no** puede empezar con un número

VARIABLES EN RUBY

```
a = 5
```

- Las variables se distinguen por ser un identificador válido (a es el identificador)

```
año = 1997
```

```
canción = "Hello"
```

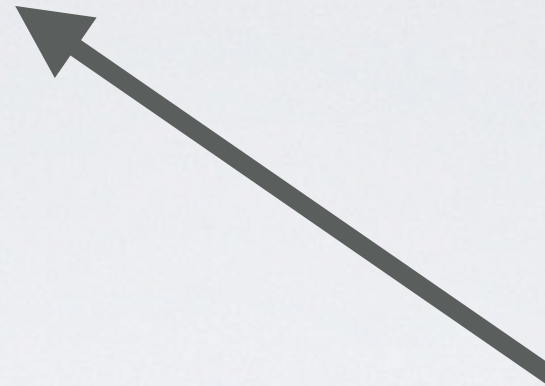
- Los identificadores son sensibles a las mayúsculas
- A una variable le podemos asignar un literal

EJERCICIO

En el intérprete intentar definir las siguientes variables

- `a = 5`
- `a = 6`
- `b1 = "hola"`
- `1b = "hola"`

1B = "HOLA"



```
SyntaxError: (irb):6: syntax error, unexpected  
tIDENTIFIER, expecting end-of-input
```

no es un identificador válido

"HOLA" = "FOO"

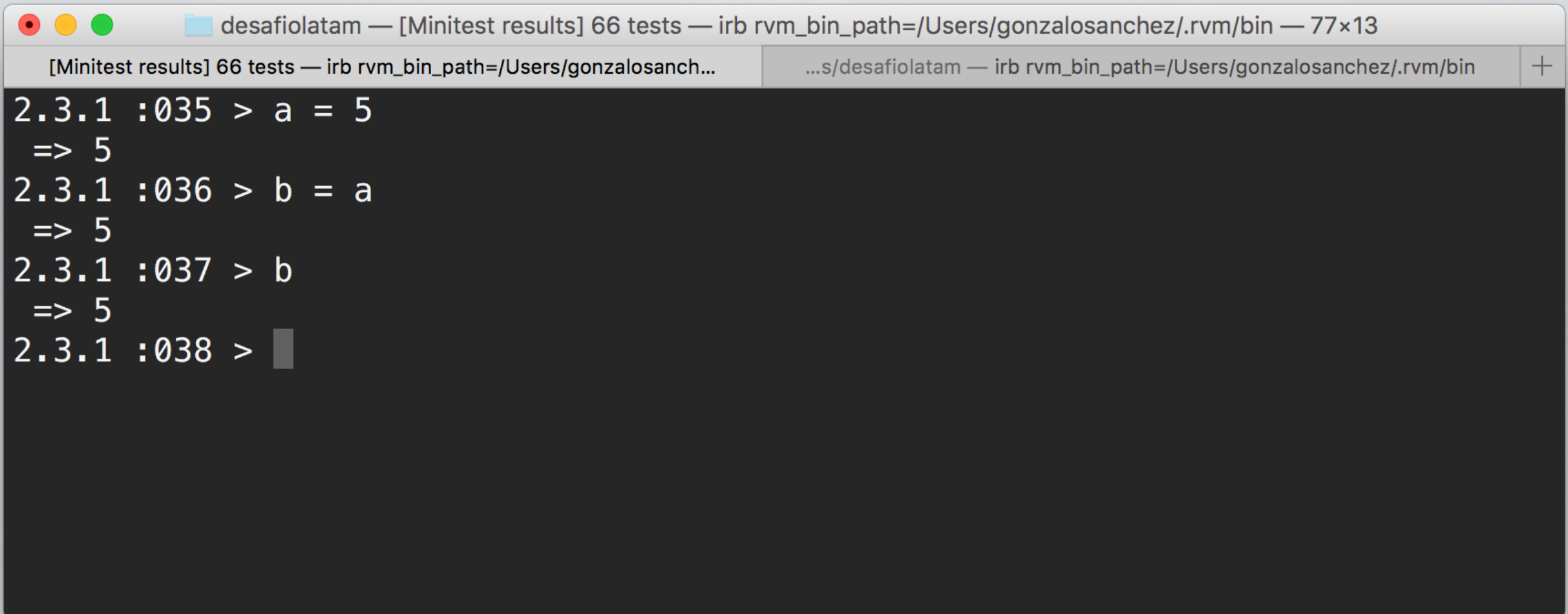
```
SyntaxError: (irb):5: syntax error, unexpected '=', expecting end-of-input
"hol" = "foo"
  ^
```

¿Cómo podríamos entender este error?

¿VÁLIDO O NO?

2 = "hola"

A UNA VARIABLE LE PODEMOS ASIGNAR EL VALOR DE OTRA VARIABLE

A screenshot of a macOS terminal window. The title bar shows a folder icon and the text "desafiolatam — [Minitest results] 66 tests — irb rvm_bin_path=/Users/gonzalosanchez/.rvm/bin — 77x13". The terminal has two tabs: "[Minitest results] 66 tests — irb rvm_bin_path=/Users/gonzalosanchez..." and "...s/desafiolatam — irb rvm_bin_path=/Users/gonzalosanchez/.rvm/bin". The terminal content shows the following Ruby code and its output:

```
2.3.1 :035 > a = 5
=> 5
2.3.1 :036 > b = a
=> 5
2.3.1 :037 > b
=> 5
2.3.1 :038 > 
```


CONSTANTES

Ingresa en el terminal $A = 5$ y luego $A = 6$
¿Qué sucede?

PUEDES PROFUNDIZAR EN LA SINTAXIS DE RUBY EN:

https://www.tutorialspoint.com/ruby/ruby_syntax.htm

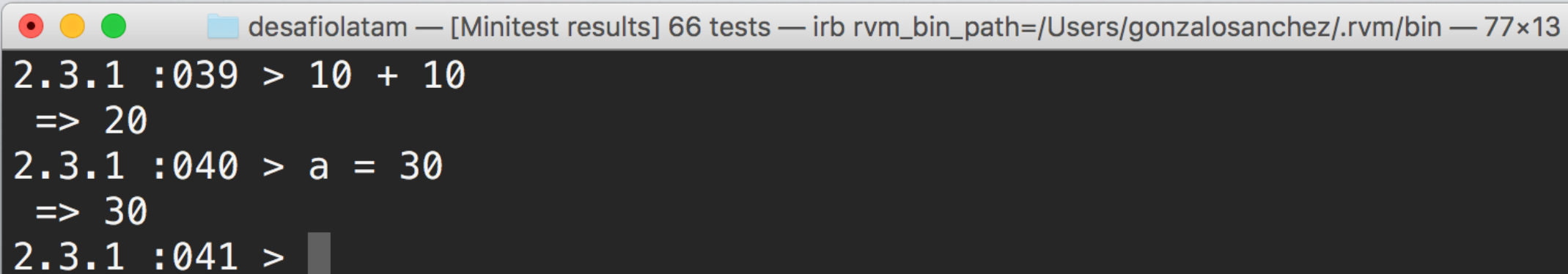
[https://books.google.cl/books?
id=jcUbTcr5XWwC&pg=PA26&lpg=PA26&dq=ruby+lexical+structure&source=bl&
ots=fJljyeasaB&sig=Sp6StIH-
e6Pft9DpbDRBtFN5bj8&hl=en&sa=X&redir_esc=y#v=onepage&q=ruby%20lexical%
20structure&f=false](https://books.google.cl/books?id=jcUbTcr5XWwC&pg=PA26&lpg=PA26&dq=ruby+lexical+structure&source=bl&ots=fJljyeasaB&sig=Sp6StIH-e6Pft9DpbDRBtFN5bj8&hl=en&sa=X&redir_esc=y#v=onepage&q=ruby%20lexical%20structure&f=false)

CHECKPOINT :)

- ☒ Instalación de ruby
- ☒ Acceso al Intérprete
- ☒ Identificadores
- ☐ Literales
- ☐ Expresiones de control (if, do, while)

LITERALES

Los literales son los valores que asignamos a las variables, o utilizamos para operar.

A screenshot of a terminal window with a dark background and light text. The window title bar shows "desafiolatam — [Minitest results] 66 tests — irb rvm_bin_path=/Users/gonzalosanchez/.rvm/bin — 77x13". The terminal content shows three lines of Ruby code being executed in an IRB session, with the results of each operation displayed on the following line.

```
2.3.1 :039 > 10 + 10
=> 20
2.3.1 :040 > a = 30
=> 30
2.3.1 :041 > █
```

https://ruby-doc.org/core-2.3.1/doc/syntax/literals_rdoc.html

TIPOS DE DATOS

Algunos muy frecuentes son:

- String
- Fixnum
- Float
- NilClass
- TrueClass, FalseClass
- Array
- Symbol
- Hash
- Time

ENTRADA Y SALIDA DE DATOS

con gets podemos capturar
datos desde el teclado

```
a = gets
```

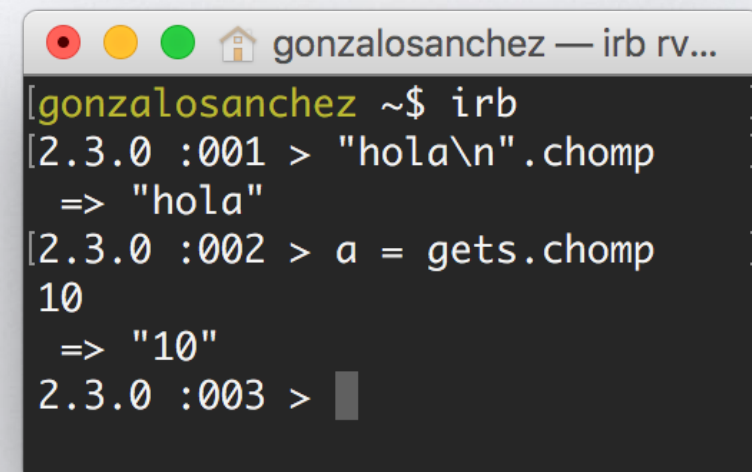
```
2.3.0 :001 > a = gets
10
=> "10\n"
2.3.0 :002 > puts a
10
=> nil
2.3.0 :003 > █
```

con puts y print podemos
mostrarlas

```
puts a
```

```
print a
```

gets guarda el salto de línea
introducido por el enter,
el método .chomp lo elimina



```
gonzalosanchez — irb rv...
[gonzalosanchez ~$ irb
2.3.0 :001 > "hola\n".chomp
=> "hola"
2.3.0 :002 > a = gets.chomp
10
=> "10"
2.3.0 :003 > █]
```


SALUDAR

```
desafiolatam — [Minitest results] 66 tests — irb rvm_bin_path=/Users/gonzalosanchez/.rvm/bin — 77x13
2.3.1 :043 > puts "Cuál es tu nombre?"
Cuál es tu nombre?
=> nil
2.3.1 :044 > nombre = gets.chomp
Gonzalo
=> "Gonzalo"
2.3.1 :045 > puts "Hola " + nombre
Hola Gonzalo
=> nil
2.3.1 :046 > █
```

PARA REUTILIZAR CÓDIGO LO HAREMOS EN SUBLIME

saludar.rb

```
< >  saludar.rb  x
1  puts "Ingresa tu nombre"
2  nombre = gets.chomp
3  puts "Hola " + nombre
4
5
```

gonzalosanchez — [Minitest results] 66 tests — -bash — 77

```
$ ruby saludar.rb
Ingresa tu nombre
Gonzalo
Hola Gonzalo
gonzalosanchez ~
$
```


EN RUBY LOS LITERALES SON OBJETOS

Los objetos tienen métodos, como por ejemplo un string tiene un método para transformar su valor a un entero.

```
"10".to_i ==> 10
```

SUMA DE NÚMEROS

suma.rb

```
puts "Ingresa dos número"  
número1 = gets.chomp  
número2 = gets.chomp  
puts número1.to_i + número2.to_i
```


CREAR UN CÓDIGO QUE TRANSFORME
LOS GRADOS CELCIUS EN KELVIN

LA DIVISIÓN

```
gonzalosanchez — irb rvm_bin_path=/Users/gonzalosanchez/.rvm/bin — 54x20
2.3.1 :001 > 5 / 2
=> 2
2.3.1 :002 > █
```


LA DIVISIÓN ENTRE DOS NÚMEROS ENTEROS ES UN ENTERO

Para diferenciar un número entero de un flotante ruby busca el . en el número

```
gonzalosanchez — irb rvm_bin_path=/Users/gonzalosanchez/.rvm/bin — 54x20
2.3.1 :003 > 5.0 / 2.0
=> 2.5
2.3.1 :004 > █
```

SI EL NUMERADOR O EL DENOMINADOR SON
FLOTANTES ENTONCES EL RESULTADO ES
FLOTANTE

LOS OPERADORES NO EXISTEN COMO TALES

sólo existen los métodos, cada objeto tiene sus métodos

2.methods

```
=> [:%, :&, :*, :+, :-, :/, :<, :>, :^, :|, :~, :-@, :**, :<=>, :<<, :>>, :<=, :>=, :==, :===, :[], :in  
spect, :size, :succ, :to_s, :to_f, :div, :divmod, :fddiv, :modulo, :abs, :magnitude, :zero?, :odd?, :even  
?, :bit_length, :to_int, :to_i, :next, :upto, :chr, :ord, :integer?, :floor, :ceil, :round, :truncate, :  
downto, :times, :pred, :to_r, :numerator, :denominator, :rationalize, :gcd, :lcm, :gcdlcm, :+@, :eql?, :  
singleton_method_added, :coerce, :i, :remainder, :real?, :nonzero?, :step, :positive?, :negative?, :quo,  
:arg, :rectangular, :rect, :polar, :real, :imaginary, :imag, :abs2, :angle, :phase, :conjugate, :conj,  
:to_c, :between?, :instance_of?, :public_send, :instance_variable_get, :instance_variable_set, :instance  
_variable_defined?, :remove_instance_variable, :private_methods, :kind_of?, :instance_variables, :tap, :  
is_a?, :extend, :define_singleton_method, :to_enum, :enum_for, :=~, :!~, :respond_to?, :freeze, :display  
, :send, :object_id, :method, :public_method, :singleton_method, :nil?, :hash, :class, :singleton_class,  
:clone, :dup, :itself, :taint, :tainted?, :untaint, :untrust, :trust, :untrusted?, :methods, :protected  
_methods, :frozen?, :public_methods, :singleton_methods, :!, :!=, :__send__, :equal?, :instance_eval, :i  
nstance_exec, :__id__]
```

CUANDO SUMAMOS

2+2

```
=> [:%, :&, :*, :+, :-, :/, :<, :>, :^, :|, :~, :-@, :**, :<=>, :<<, :>>, :<=, :>=, :==, :===, :[], :inspect, :size, :succ, :to_s, :to_f, :div, :divmod, :fdiv, :modulo, :abs, :magnitude, :zero?, :odd?, :even?, :bit_length, :to_int, :to_i, :next, :upto, :chr, :ord, :integer?, :floor, :ceil, :round, :truncate, :downto, :times, :pred, :to_r, :numerator, :denominator, :rationalize, :gcd, :lcm, :gcdlcm, :+@, :eql?, :singleton_method_added, :coerce, :i, :remainder, :real?, :nonzero?, :step, :positive?, :negative?, :quo, :arg, :rectangular, :rect, :polar, :real, :imaginary, :imag, :abs2, :angle, :phase, :conjugate, :conj, :to_c, :between?, :instance_of?, :public_send, :instance_variable_get, :instance_variable_set, :instance_variable_defined?, :remove_instance_variable, :private_methods, :kind_of?, :instance_variables, :tap, :is_a?, :extend, :define_singleton_method, :to_enum, :enum_for, :=~, :!~, :respond_to?, :freeze, :display, :send, :object_id, :method, :public_method, :singleton_method, :nil?, :hash, :class, :singleton_class, :clone, :dup, :itself, :taint, :tainted?, :untaint, :untrust, :trust, :untrusted?, :methods, :protected_methods, :frozen?, :public_methods, :singleton_methods, :!, :!=, :__send__, :equal?, :instance_eval, :instance_exec, :__id__]
```

ruby entiende 2.+(2)

.+ es el nombre del método, 2 el argumento recibido

SI EL OBJETO NO SOPORTA LA OPERACIÓN OBTENDREMOS UN ERROR

```
gonzalosanchez — irb rvm_bin_path=/Users/gonzalosanchez/.rvm/bin — 90x15
[gonzalosanchez ~$ irb
2.3.0 :001 > a = Object.new()
=> #<Object:0x007fa85a0bf220>
2.3.0 :002 > a + 2
NoMethodError: undefined method `+' for #<Object:0x007fa85a0bf220>
      from (irb):2
      from /Users/gonzalosanchez/.rvm/rubies/ruby-2.3.0/bin/irb:11:in `<main>'
2.3.0 :003 > nil + 3
NoMethodError: undefined method `+' for nil:NilClass
      from (irb):3
      from /Users/gonzalosanchez/.rvm/rubies/ruby-2.3.0/bin/irb:11:in `<main>'
2.3.0 :004 > █
```


STRINGS

```
gonzalosanchez — irb rvm_bi...  
[2.3.0 :015 > "hola".length  
=> 4  
2.3.0 :016 > ]
```

tienen métodos útiles
para calcular el largo

```
gonzalosanchez — irb rvm_bi...  
[2.3.0 :012 > "hola".reverse  
=> "aloh"  
2.3.0 :013 > ]
```

hay métodos para
invertirlo

```
gonzalosanchez — irb rvm_bin_path=/...  
[2.3.0 :017 > "hola" + " a" + " todos"  
=> "hola a todos"  
2.3.0 :018 > ]
```

Son concatenables

SÍMBOLOS

Son como los strings pero más sencillos

`:soy_un_simbolo`

`“soy_un_string”`

LOS SÍMBOLOS TIENEN MENOS MÉTODOS QUE LOS STRINGS

```
:hola + :c  
NoMethodError: undefined method `+' for :hola:Symbol  
from (irb):25
```


EJERCICIO EN CLASES

Crear un programa que pida el nombre y edad y luego concatenarlas e imprimirlas (mostrarlas)

INTERPOLACIÓN

```
a = 5  
puts "el valor de a es #{a}"
```

Para muchos casos la interpolación resulta más sencilla que la concatenación de strings

La interpolación requiere de comillas dobles

EJERCICIO EN CLASES

Repetir el ejercicio anterior pero
utilizando interpolación

FIXNUMS

Son los números enteros

$a = 5$

Se pueden realizar operaciones típicas sobre ellos

```
gonzalosanchez — irb rvm_bin_path=/Use...  
[2.3.0 :018 > 5 + 2  
=> 7  
[2.3.0 :019 > 5 * 2  
=> 10  
[2.3.0 :020 > 5 ** 2  
=> 25  
[2.3.0 :021 > 5 / 2  
=> 2  
2.3.0 :022 >
```

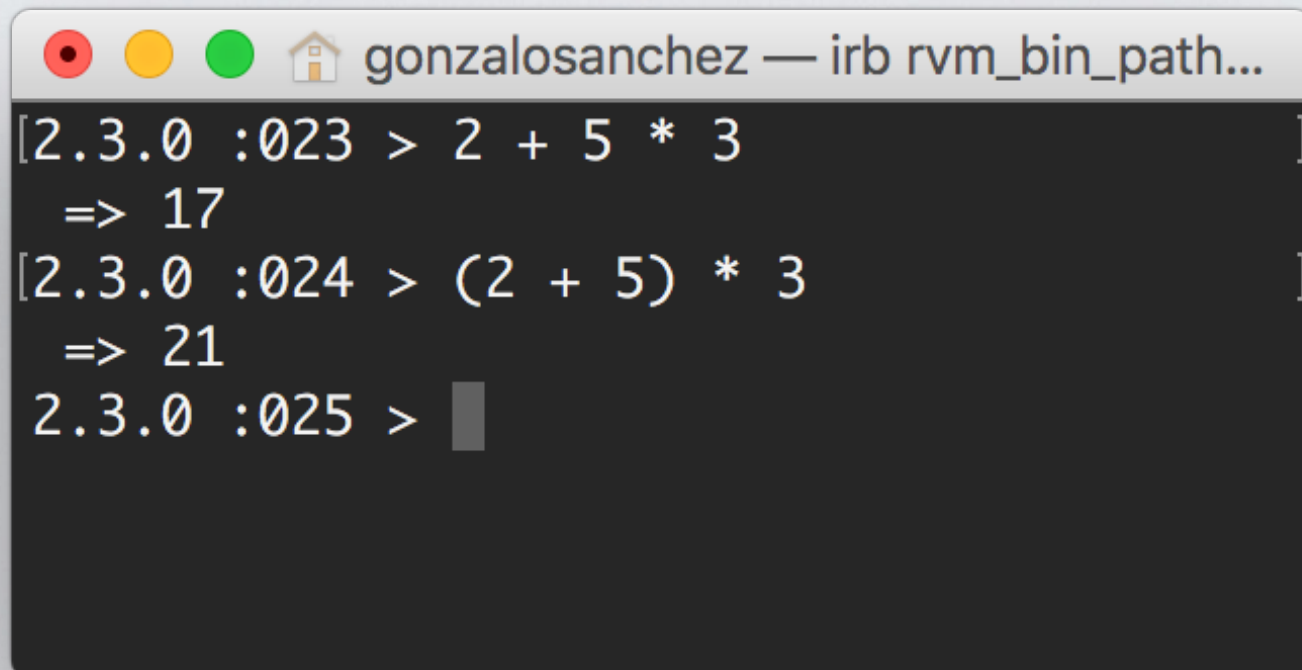
Además debemos tener cuidado que al dividir dos fixnums obtendremos un fixnum

PARA CONVERTIR UN STRING A FIXNUM PODEMOS OCUPAR EL MÉTODO TO_I

A screenshot of a macOS terminal window. The title bar shows three colored window control buttons (red, yellow, green) and a home icon, followed by the text 'gonzalosanchez — irb rvm_bin_path...'. The terminal content shows a Ruby prompt '[2.3.0 :025 > '23'.to_i]' followed by the output '=> 23'. The next line shows the prompt '2.3.0 :026 >' with a grey cursor bar.

```
gonzalosanchez — irb rvm_bin_path...  
[2.3.0 :025 > "23".to_i  
=> 23  
2.3.0 :026 > ]
```

PRECEDENCIA DE OPERADORES



```
gonzalosanchez — irb rvm_bin_path...  
[2.3.0 :023 > 2 + 5 * 3 ]  
=> 17  
[2.3.0 :024 > (2 + 5) * 3 ]  
=> 21  
2.3.0 :025 > █
```


FLOATS

Los números flotantes **son representaciones inexactas** de los números reales y por lo mismo se debe tener precaución al compararlos a escala muy pequeñas.

Ejemplos de floats

2.0
3.14
1.5

Los floats se escriben con .
no con ,

Podemos convertir un string o un entero a float ocupando el método **to_f**

TRUE Y FALSE

```
a = true  
b = false
```

NIL

A diferencia de muchos otros lenguajes populares existe un objeto para los valores nulos, este es nil, lo bueno de que los valores nulos sean un objeto es que podemos hacer cosas como esta:

```
a = nil  
a.nil?
```

```
[2.3.0 :001 > 0 == false  
=> false  
[2.3.0 :002 > 0 == true  
=> false  
[2.3.0 :003 > 1 == true  
=> false  
[2.3.0 :004 > nil == false  
=> false  
2.3.0 :005 > █
```


CHECKPOINT :)

- ☒ Instalación de ruby
- ☒ Acceso al Intérprete
- ☒ Identificadores
- ☒ Literales
- ☐ Expresiones de control (if, do, while)

BLOQUES DE CONTROL (CONDICIONES)

IF Y UNLESS

Forma normal

```
if a == 2  
  puts "a vale 2"  
end
```

```
unless a == 2  
  puts "a es distinto de 2"  
end
```

Forma express

```
puts "a vale 2" if a == 2
```

```
puts "a es distinto de 2" unless a == 2
```

EJERCICIO EN CLASES

MAYOR DE DOS NÚMEROS

El usuario ingresa dos números
se debe determinar cuál es el mayor

elsif / else

```
puts "ingresa un número"  
a = gets.chomp.to_i  
if a > 10  
  puts "tienes más de 10 años"  
elsif a >= 20  
  puts "tienes más de 20 años"  
else  
  puts "eres menor de 10 años"  
end
```

Se debe tener mucho cuidado que utilizar dos ifs no es lo mismo que utilizar if y elsif.

EJERCICIO

El usuario ingresa un número y nos dirá si es positivo, negativo, par, impar o si es 0

MI PRIMERA CALCULADORA

- El usuario debe ingresar dos números y luego una operación, si es $+$ sumar, si es $-$ restar, si es \times multiplicar y si $/$ dividir, en el caso de la división tener cuidado de no perder el resto.

Lógica booleana

Al igual que en clases de pseudocódigo y diagramas de flujo, es posible ocupar lógica booleana a través de las instrucciones `and`, `or` y `not`

`a = true`

`b = false`

`not(a) => ?`

`not(b) => ?`

`(a or b) => ?`

`(a and b) => ?`

LOOPS

While

```
var = 0  
while var < 10  
  puts var  
  var += 1  
end
```

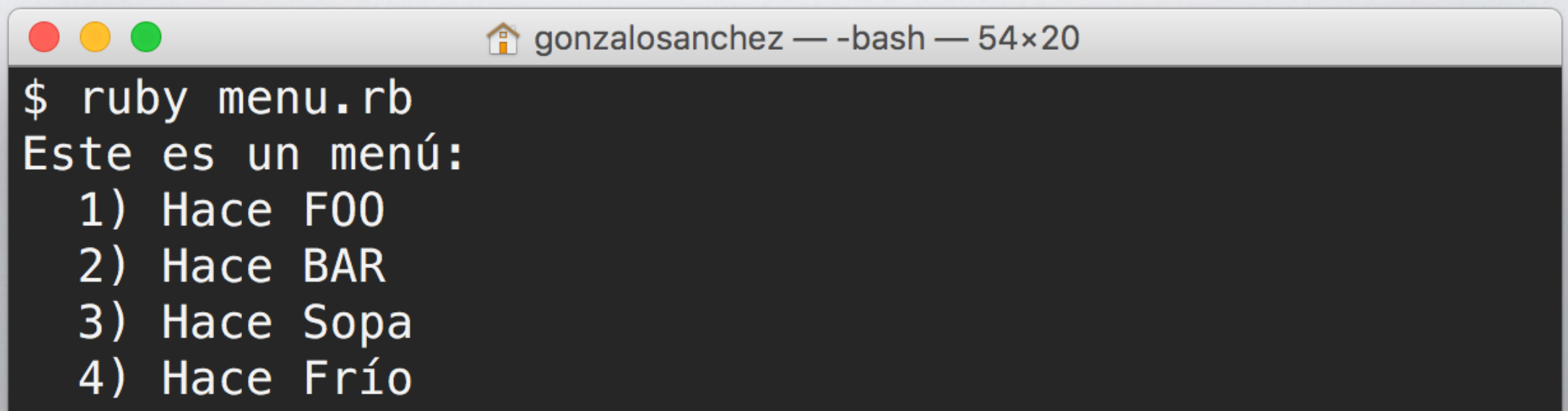
while permite repetir una instrucción mientras se cumpla una condición. Mientras la condición sea verdadera todo lo que esté dentro del bloque se repetirá.

VALIDACIÓN DE ENTRADAS

- Pedir que un usuario ingrese un número entre 0 y 36, volver a pedir hasta que se cumpla la condición

NUESTRO PRIMER MENÚ

- Crear un programa que te de a escoger entre 4 opciones, si es escoge la opción 1, debe decir que escogiste la opción 1 y así mismo para cada opción.
- Al escoger cualquier otra opción que no sea del 1 al 4 se debe volver a imprimir el menú

A screenshot of a macOS terminal window. The title bar shows three colored window control buttons (red, yellow, green) on the left, a home icon and the text 'gonzalosanchez — -bash — 54x20' in the center. The terminal content shows a command prompt '\$' followed by 'ruby menu.rb'. The output of the command is 'Este es un menú:' followed by a numbered list of four options: '1) Hace F00', '2) Hace BAR', '3) Hace Sopa', and '4) Hace Frío'.

```
$ ruby menu.rb
Este es un menú:
  1) Hace F00
  2) Hace BAR
  3) Hace Sopa
  4) Hace Frío
```

x.times

Repite x veces

Forma normal

```
5.times do  
  puts "repitiendo"  
end
```

Forma express

```
5.times { puts "repitiendo" }
```

Si está dentro de un do o {} es un bloque
(pronto veremos más de eso)

x.times con índices

Forma normal

```
5.times do |i|  
  puts "repitiendo: #{i}"  
end
```

Forma express

```
5.times { |i| puts "repitiendo #{i}" }
```

Desafío: Utiliza la instrucción times para contar todos los números pares hasta 100

FOR

```
for i in 0..10  
  puts i  
end
```

“ For es una instrucción que nos permite iterar sobre rangos y arreglos*, por ejemplo (* lo que veremos más adelante) “

EJERCICIO

Mostrar todos los divisores del número 840:

- con while
- con for
- con times

.each

El método .each nos permite movernos a lo largo de un conjunto de números o de letras.

```
puts "Hola \n"
```

```
"hola".each_char do |i|  
  puts i  
end
```

```
puts "Números \n"
```

```
[1,8,2,5,7].each do |i|  
  puts i  
end
```

Esto es un array, los
estudiaremos
más adelante

PARA PROFUNDIZAR

CASE

```
case a
when 1..10
  puts "Estoy entre 1 y 10"
when 11
  puts "Soy 11, o sea más que 10"
when String
  puts "No soy un número soy un string"
else
  puts "Ups, no se que hacer con #{a}"
end
```

“Cuando son muchos los posibles casos, se recomienda ocupar la instrucción case en lugar de varios ifs, ayuda al orden del código.”