

Primeras aproximaciones con Ruby

1. Ruby:

- Hay que tener varias cosas en cuenta de Ruby antes de nada, la primera de ellas es que es **no se compilan** los archivos de Ruby, se ejecutan tal cual y con una peculiaridad más, se va ejecutando **línea a línea** esto provoca que:
 1. Como no compilas no sabes los errores que puedes tener en tu código hasta que pasas por **todas las líneas** (y si hay líneas que rara vez pasas entonces podrías acumular errores sin darte cuenta).
 2. Di **adiós a los puntos y comas** al final de cada sentencia: solo tienes que poner una en cada línea y se interpretan igual que en los demás lenguajes.
 3. Ya no tienes que escribir llaves: normalmente cuando se hace algo que implique bloque (**class X, for, while, def...**) se va a esperar siempre un **end** que implica el fin de bloque: normalmente la IDE nos va a poner siempre el end, pero ten cuidado.
 4. Para los proyectos, como no compilas los archivos, no se usa un Makefile en vez de eso se usa un **Rakefile**, que si te líabas mucho con el Makefile no te preocupes, la IDE está aquí para ayudarte, pero tenlo en cuenta.
 5. Ten **mucho cuidado** si quieres separar una línea muy larga en varias, puedes usar **** para indicar que continua en la siguiente línea, por ejemplo:

```
string = "line #1line #2line #3"
```

```
Podemos ponerlo como: string = "line #1"\      "line #2"\  
"line #3"
```

- Normalmente en Ruby se usa la notación **snake_case**, que como habrás podido notar es la de separar las palabras por el guión bajo, sin embargo en **PDOO** vamos a usar **camelCase** por razones técnicas, pero ten en cuenta que si haces cosas en Ruby se usa la notación **snake_case**.
- La mentalidad de programación en Ruby tienes que cambiarla, personalmente hay una cosa esencial que cambia mucho de Java/C++ a Ruby en lo que es la manera de pensar como trabajar con **arrays**, esto lo veremos más adelante, pero ten en cuenta de que hay que “cambiar totalmente el chip”.
- Se acabó lo de poner los tipos a las variables. Ruby es un lenguaje de **tipado dinámico**, esto significa que si tu declaras una variable por ej **var**, no tienes que poner el tipo y lo mejor de todo es que puedes hacer **variar**

su tipo cuando tu quieras (puedes hacer lo que quieras con var). Así si pasas parámetros a funciones no hace falta que indiques su tipo.

- Aunque esto nos permite mucha **versatilidad** no hay que abusar de ello para evitar liarnos a nosotros mismos y perder la cuenta de que era la variable **var** y también para dar legibilidad al código.
- Para comentar se usa **#** que comenta una línea, para comentar varias líneas hay otra manera que es usando **=begin** y **=end** y entre medias puedes comentar lo que quieras, pero siempre poniendo ambas al principio de líneas separadas (aunque he leído que la gente prefiere usar varias **#**).
- Existe **return** pero normalmente no se usa, cuando se llega al final de la función, se devuelve la última salida que se ha producido. Por ello, si quieres estar seguro de devolver lo que quieres solo tienes que poner al final de la función lo que quieres devolver pero sin el return.
- Vamos a ver varios ejemplos de como declarar funciones (métodos) siempre es con **def nombre_metodo(param)** en caso de que no tenga parametros la función entonces no hace falta poner nada.

```
Ej. 1 (sin parametros) Ruby  def nombre_metodo      # cosas      #
cosas      lo_que_quiero_devolver  end
```

```
Ej. 2 (con parametros) Ruby  def nombre_metodo(mi_var, otra_var)
# cosas      # cosas      lo_que_quiero_devolver  end
```

2. Clases:

- Debemos fijarnos que en Ruby (como en Java) todo son clases, vamos a tener que trabajar con la idea de que hagas lo que hagas vas a usar clases. Por ello hay que tener claro como son las clases en Ruby.
 - Las clases en Ruby son muy versátiles, mientras ejecutamos se pueden reescribir métodos suyos o añadir datos miembro... pero en PDOO vamos a ser más
 - Todas las clases deben tener un método **initialize** que sería lo que nosotros denominamos un **constructor** y es **ÚNICO**, luego veremos como crear varios constructores. Para evitar líos con Ruby, en PDOO vamos a darles un estado siempre a todas las variables en el **intialize** (aunque no es necesario realmente).
 - Los datos miembro (**atributos**) siempre son privados y los métodos son públicos por defecto, **cuidado** porque que algo sea privado en Ruby tiene mucha fuerza:
1. Si un método es privado entonces **No** podemos asignar lo que devuelva el método a nada, ni siquiera entre objetos de la misma clase, solo puede

asignarse a algo si es el mismo objeto que invoca. Sin embargo **Si** se pueden llamar normalmente.

2. Como los atributos pasa igual, no podemos consultarlos y/o modificarlos de otros objetos, a no ser que tengamos un `attr_` especificado (ahora lo veremos).
3. Solo se pueden acceder a atributos con total libertad si es del propio objeto; y asignar si es tu propio método privado; es decir si al llamar a los métodos o acceder a las variables no es del tipo `var = obj.metodo_privado` o `obj.variable`, incluso `self.obj` daría fallo. Solo cuando podemos poner `var = metodo_privado, @variable` es cuando tenemos acceso (“no hay nada delante con un punto”).
4. Para implementar los métodos `set/get` podemos hacerlo como normalmente lo hacemos, pero hay una opción más rápida y es usando los `attr_`. Se usan:
 - `attr_accessor :var` permite consultar/modificar el atributo `var`
 - `attr_reader :var` permite consultar el atributo `var`
 - `attr_writer :var` permite modificar el atributo `var`