Genís Bosch Pou
LP

Ruby:
A Programmer's Best Friend

Introducció

Ruby, un llenguatge de programació interpretat i multiparadigma, va ser inventat per el japonès Yukihiro Matsumoto. Encara que va ser llençat al públic el 1995, Matsumoto portava utilitzant-lo des del 1991, per a ús propi.
Durant els primers anys, va explicar que pretenia aconseguir un llenguatge que millorés la productivitat, així com la diversió dels programadors, i que es basava en Python i Perl. El nom d'aquest llenguatge, de fet, prové de la semblança sintàctica amb Perl, "perla" en català. Així doncs, el llenguatge de Matsumoto també agafa el nom d'una joia, en aquest cas el rubí.

Per què Ruby?

Ruby és un llenguatge multiparadigma, però la seva principal virtut és ser un llenguatge orientat a objectes, així com tenir una sintaxi molt amigable, que facilita la feina al programador, encara que perd eficiència respecte a altres llenguatges. Després de més de 20 anys, lluny d'estar en decadència, Ruby és molt utilitzat en la creació d'aplicacions web, tot gràcies el famós framework Ruby on Rails, que facilita la creació de back-ends per a tot tipus de pàgines web.

Característiques

Es basa en el principi de la menor sorpresa, que afirma que un llenguatge amb unes funcionalitats semblants a altres llenguatges ha de mantenir una semblança a l'hora de fer les coses, per a que programadors experimentats en altres llenguatges, puguin aprendre aquest sense haver de canviar de forma radical el sistema de fer les coses.

En Ruby, gairebé tot és un objecte, incloent els tipus primaris com String, Char, Int, Float.. Això permet tractar els tipus “bàsics” d’una forma més sorprenent i intuitiva, tal com veurem a continuació:

```
String number = “121218”;
int result = Integer.parseInt(number);
```

En Java, com a exemple de llenguatge de programació orientat a objectes, per a fer la conversió de String a Int, hem de cridar a la classe Integer per a que ens retorni un enter amb la seva funció parseInt. En canvi, si l’String ja pertany a la classe, no és necessari cridar la classe Integer, i podem utilitzar el mètode que la pròpia classe String ens ofereix, com en el cas de Ruby:

```
number = “121218”
result = number.to_i
```

Com ja hem mencionat, es tracta d’un llenguatge interpretat, per el que això el fa un entorn de desenvolupament ràpid d'aplicacions, ja que no requereix de compilació i el temps de programació i prova es redueix considerablement. Es tracta d’un llenguatge multiplataforma, i per tant altament portable, i disposa de moltes llibreries estàndard que el complementen.

Ruby, a més, és multiparadigma. Això és, pot resoldre problemes de formes diverses. Per una banda, com molts llenguatges, és imperatiu, el que permet executar instruccions, una després de l’altre. També, però, accepta la programació funcional, basada en el càlcul lambda i l’ús matemàtic. Finalment, i per el que és més conegut, permet la programació orientada a objectes.

Per el que fa al sistema de tipus, Ruby està fortament tipat, és a dir, no permet assignacions que canvien el tipus, encara que es resol en temps d’execució. No gens menys, funciona amb tipat dinàmic, per el que no s’han de declarar les variables, ja que al inicialitzar-les ja agafen el tipus del contingut que contenen. Vegem un exemple:

```
x = "3"
y = “Número “ + x #“Número 3”
```

Al fer l’assignació x = “3”, el tipus per la variable passa a ser automàticament de [Char], conegut també com a String. Així doncs, al assignar a la variable y la cadena de caràcters “Número “ i afegir-hi el contingut de x, obtenim el resultat vàlid i esperat “Número 3”.

En canvi, si intentem interactuar de forma diferent amb els tipus:

```
x = "3"
y = x + 3 #ERROR
```

Ara, a la variable y se li assigna el valor contingut en la variable x sumat amb 3, però l'intèrpret de Ruby s'adona que la variable x no conté el mateix tipus que el número 3, de la classe Integer, i ens dóna error.

Al tractar-se d'un llenguatge orientat a objectes, les variables poden ser de 4 nivells diferents. En primer lloc, una variable continguda dins d'alguna funció és una variable local, mentre que si aquesta es troba fora dels mètodes, funcions i classes, es tracta d'una variable global. Al afegir classes, apareixen les variables de classe i les d'instància. Una variable de classe és comuna per a tots els objectes d'aquella classe, mentre que una variable d'instància és única per a cada instància de la mateixa classe.

A més, al treballar amb estructures no primàries de dades, Ruby ens ofereix iteradors i clàusules per a recórrer-els. En cas de trobar-se una instància que ja no està apuntada (per tant ha estat eliminada, hi ha un recolector de brossa automàtica per a eliminar la instància definitivament, i no deixar-la des-referenciada en memòria durant tota l'execució.

Finalment, però no per això menys important, una de les característiques més úniques de Ruby, els blocs. Es tracta d'una porció de codi tancada entre claudàtors, o entre un do i un end. Tot i que això pot semblar una forma d'agrupar instruccions i per tant, tractar-se simplement d'un mètode, el potencial és molt més gran. Vegem primer un exemple d'un bloc simple:

```
def method
  yield('hi', 1)
end

method{|str,num| puts str + ' ' + num.to_s}
```

Al utilizar yield en un mètode, aquest utilitzarà el bloc que se li ha passat, en aquest cas "method", i l'executarà. Així doncs, la sortida serà "Hi 1".

Els blocs no són una classe, però poden convertir-s'hi gràcies als procediments. Al utilitzar l'objecte Proc, és possible encapsular un bloc dins un objecte i guardar-lo en una variable o passar-lo a un mètode. Un exemple simple d'un bloc encapsulat podria ser:

```
proc1 = Proc.new do |param|
  puts "Sòc un bloc encapsulat. M'han enviat " + param + " com a
paràmetre."
end

proc1.call
```

Així, proc1, que hauria de ser un bloc, passa a ser un bloc encapsulat en un objecte de tipus Proc, i permet ser cridat com un objecte amb els seus mètodes. No solament això, sinó que també permet ser passat com a paràmetre, ja que es tracta d'un objecte. Això ens dóna pas a poder construir mètodes que retornin procediments, i així obtenir conceptes de programació com el *lazy evaluation*, *uncurry* o *estructures de dades infinites*. Veiem un cas en el que aconseguim "simular" el *compose* de Haskell, per exemple:

```
def compose proc1, proc2
  Proc.new do |x|
    proc2.call(proc1.call(x))
  end
end

squared = Proc.new do |x|
  x * x
end

doubled = Proc.new do |x|
  x + x
end

doubleandsquare = compose doubled, squared
```

```
puts doubleandsquare.call(5)
```

Primer, al definir el mètode compose, especifiquèm que necessita dos paràmetres, anomenats proc1 i proc2. Dins del mètode, diem que es tracta d'un proc que obté com a paràmetre una x, i que fa la composició de proc2(proc1(x)). Llavors, aquests dos paràmetres són creats com a processos, per el que tindran el mètode call. Finalment, inicialitzem el proc doubleandsquare amb els 2 paràmetres tipus Proc, i fem la crida d'aquest (és una classe) a el mètode call, que ens executarà la seva definició. D'aquesta manera podem obtenir el compose de Haskell, així com altres funcions d'ordre superior, ja que podem passar mètodes com a paràmetres gràcies als procediments.

Principals aplicacions de Ruby

Com que, com ja hem mencionat anteriorment, es tracta d'un llenguatge orientat a objectes, els seus usos poden ser els mateixos que Java o C++, per exemple, però el rendiment serà més baix, així que en general és difícil veure programes d'aquests tipus escrits en Ruby. En canvi, gràcies a la semblança que comparteix amb Python i Perl, és utilitzat també per la comunitat científica per a programes de medicina o biología.
També, gràcies a la metaprogramació que ofereix, és a dir, la capacitat d'aconseguir un programa que pugui escriure programes, permet ajudar als programadors a escriure grans quantitats de codi que han de ser força repetitives.

Tot i així, el seu principal ús és per la creació d'aplicacions web, gràcies al framework Ruby on Rails, que facilita l'experiència als programadors.
Ruby on Rails és, per tant, un poderós framework enfocat en la creació de pàgines web que combina Ruby amb HTML, CSS i JavaScript. Permet correr aplicacions sobre un servidor, i fa la funcionalitat de back-end de l'aplicació, és a dir, la capa d'accés a les dades. El seu ús es basa, per tant, en el model vista-controlador, i incorpora dos principis bàsics, la Convention Over Configuration (COC)i el Don't Repeat Yourself (DRY).

La COC permet evitar moltes configuracions, generant-les per defecte si no s'especifica el contrari. Per exemple, si sap de l'existència d'una classe anomenada Usuari i és coneixedora de que aquesta necessita una base de dades, crearà una base de dades anomenada Usuaris si no se li demana especificament que tingui un altre nom.
Per altre banda, DRY permet evitar feina repetida al programador, repassant les declaracions i repetint-les en lloc del programador si és necessari.

Amb aquest framework, moltes pàgines i aplicacions web han estat creades, entre elles algunes de bastant conegudes, com seria el cas de Twitter, Airbnb, Github, Slideshare, Askfm o UrbanDictionary.

Dades sobre Ruby

Com molts altres llenguatges, Ruby té els seus pros i les seves contres, i això suposa ser agradat per molts programadors, però també ser odiat per uns quants. Veiem l'índex de popularitat de Ruby:

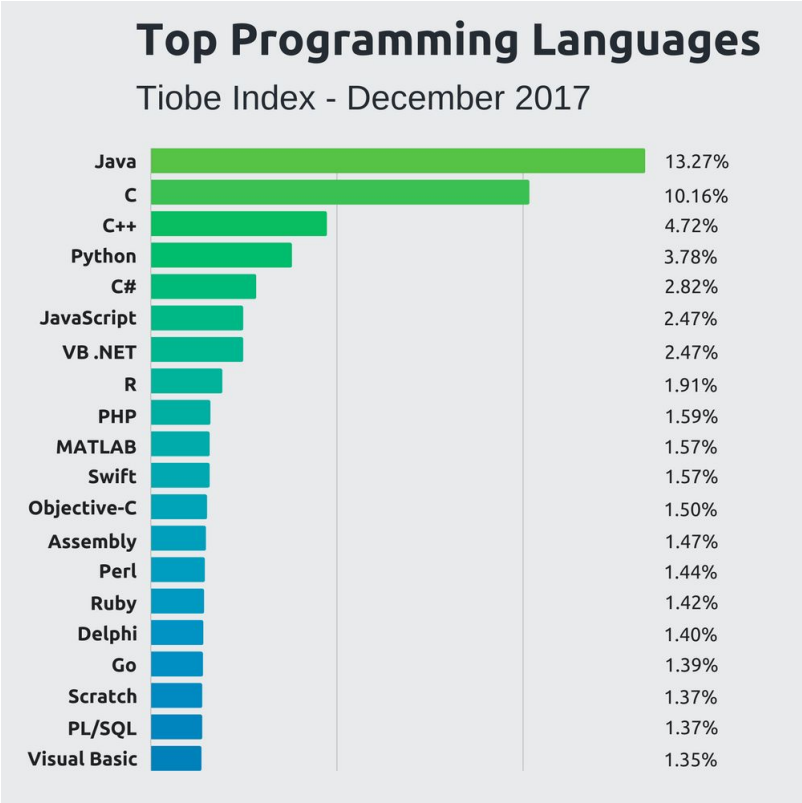

Figura 1. Popularitat (en %) dels llenguatges de programació més famosos

Com podem veure en la Figura 1, Ruby és un llenguatge minoritari. No és que passi desconegut (es tracta del 15é més utilitzat al desembre del 2017), però no és el principal llenguatge de programació orientada en objectes, ni tant sols el segon. Tot i això, ha arribat a assolir un índex de popularitat similar al seu “pare”, Perl, per el que no s’ha de desestimar el seu potencial.

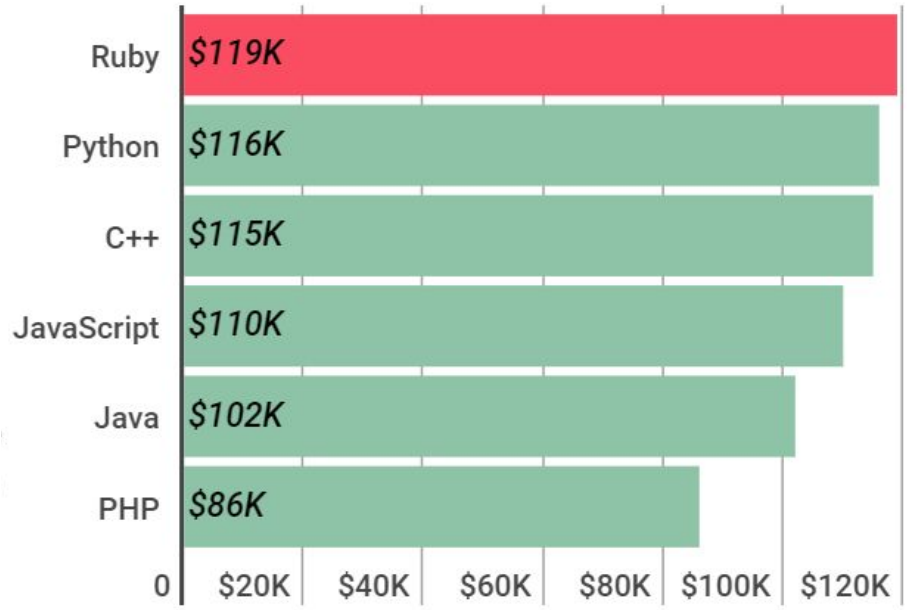

Figura 2. Salari anual ingressat pels programadors a USA en funció del llenguatge

Mentre que Ruby no és un dels llenguatges més famosos, gràcies al seu framework Ruby on Rails, com bé podem veure a la Figura 2, és un dels més ben pagats, ja que amb aquesta eina s'han generat aplicacions realment importants com Twitter o Airbnb, on els programadors s'han emportat autèntiques fortunes.

Així doncs, Ruby es tracta d'un llenguatge de programació multiparadigma, tot i que actualment molt enfocat en la creació d'aplicacions web. Cal destacar que és un llenguatge molt sol·licitat als programadors web, sobretot el domini de Ruby on Rails per a la facilitat que aporta a l'hora de la creació de les aplicacions.

Bibliografía

Ruby. (1995-2018). Web Oficial:
https://www.ruby-lang.org/es/

Entrada a la Wikipèdia sobre Ruby:
https://es.wikipedia.org/wiki/Ruby

“Aprende a Programar con Ruby”, de RubySur (Usuaris de Ruby d'Amèrica del Sud).
Disponible a:
http://rubysur.org/aprende.a.programar/

Ruby on Rails. Web Oficial:
http://rubyonrails.org.es/

“Por qué aprender Ruby?” de Uriel Hernández, 2018. Disponible a:
https://codigofacilito.com/articulos/por-que-aprender-ruby

“11 cosas que debes saber sobre Ruby on Rails” de Diego Arias, 2016. Disponible a:
http://blog.desafiolatam.com/11-razones-para-aprender-ruby-on-rails/

“Qué es y para que sirve Ruby on Rails?” de Olga Berrios, 2016. Disponible a:
http://www.labroma.org/blog/2016/01/11/que-es-y-para-que-sirve-ruby-on-rails/

“A simple rails architecture info-graphic” de /AbishekAditya (Reddit), 2017. Disponible a:
https://www.reddit.com/r/ruby/comments/5lfxy0/a_simple_rails_architecture_infographic/

“Ruby Code Examples” de WikiBooks, 2018. Disponible a:
https://en.wikibooks.org/wiki/Ruby_Code_Examples

“Bloques y Procedimientos” de RubySur (Usuaris de Ruby d’Amèrica del Sud).
Disponible a:
http://rubysur.org/aprende.a.programar/capitulos/bloques.html