

Unidad 1.2.

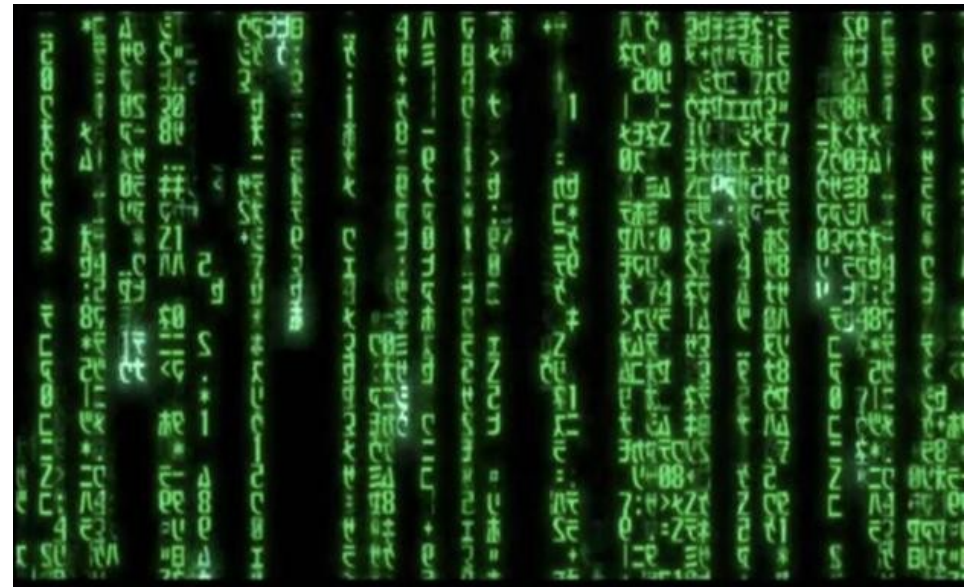
La importancia del código

GUI vs. Code



Graphical User Interface (GUI)



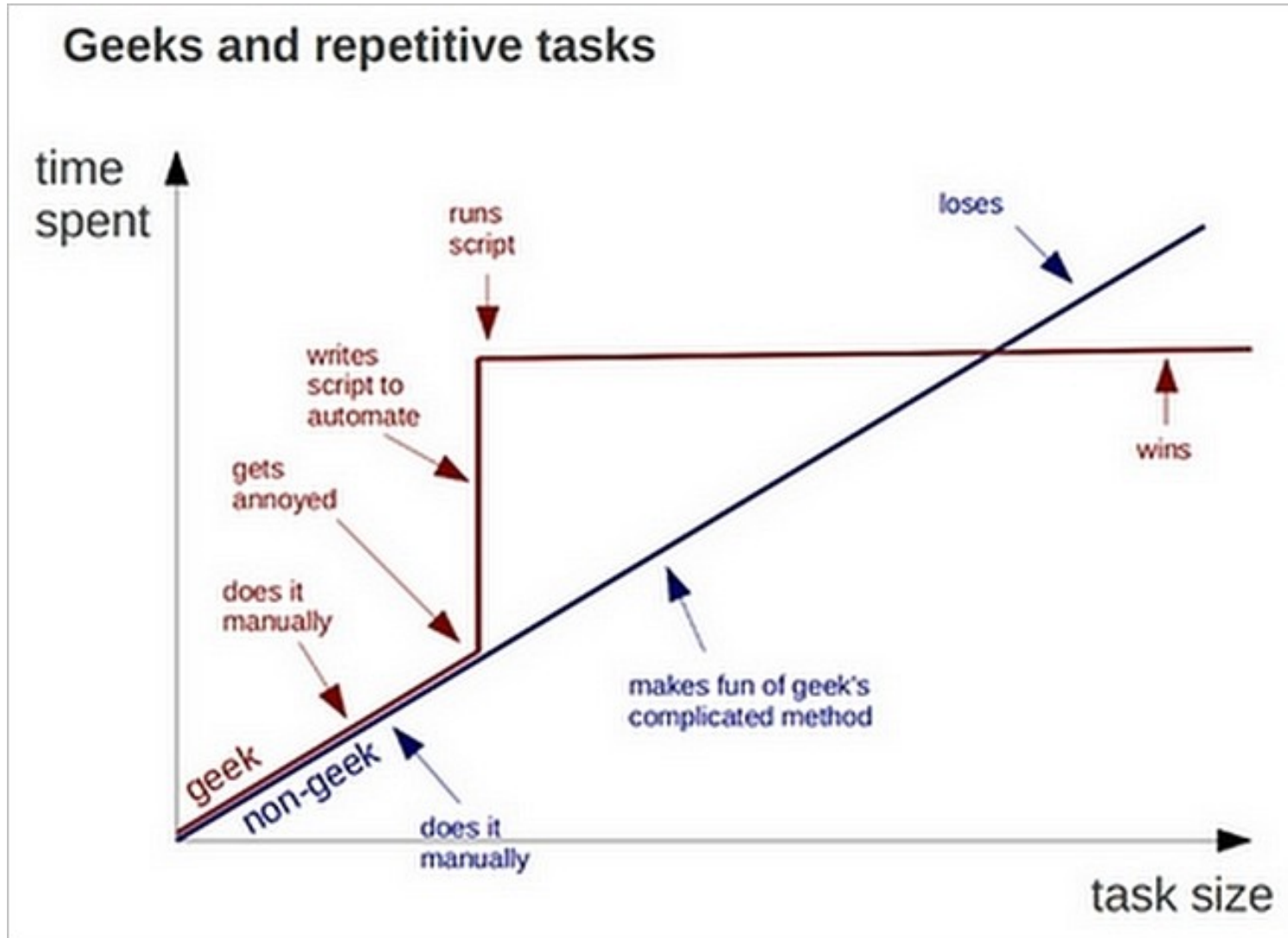
Code



Pros y contras

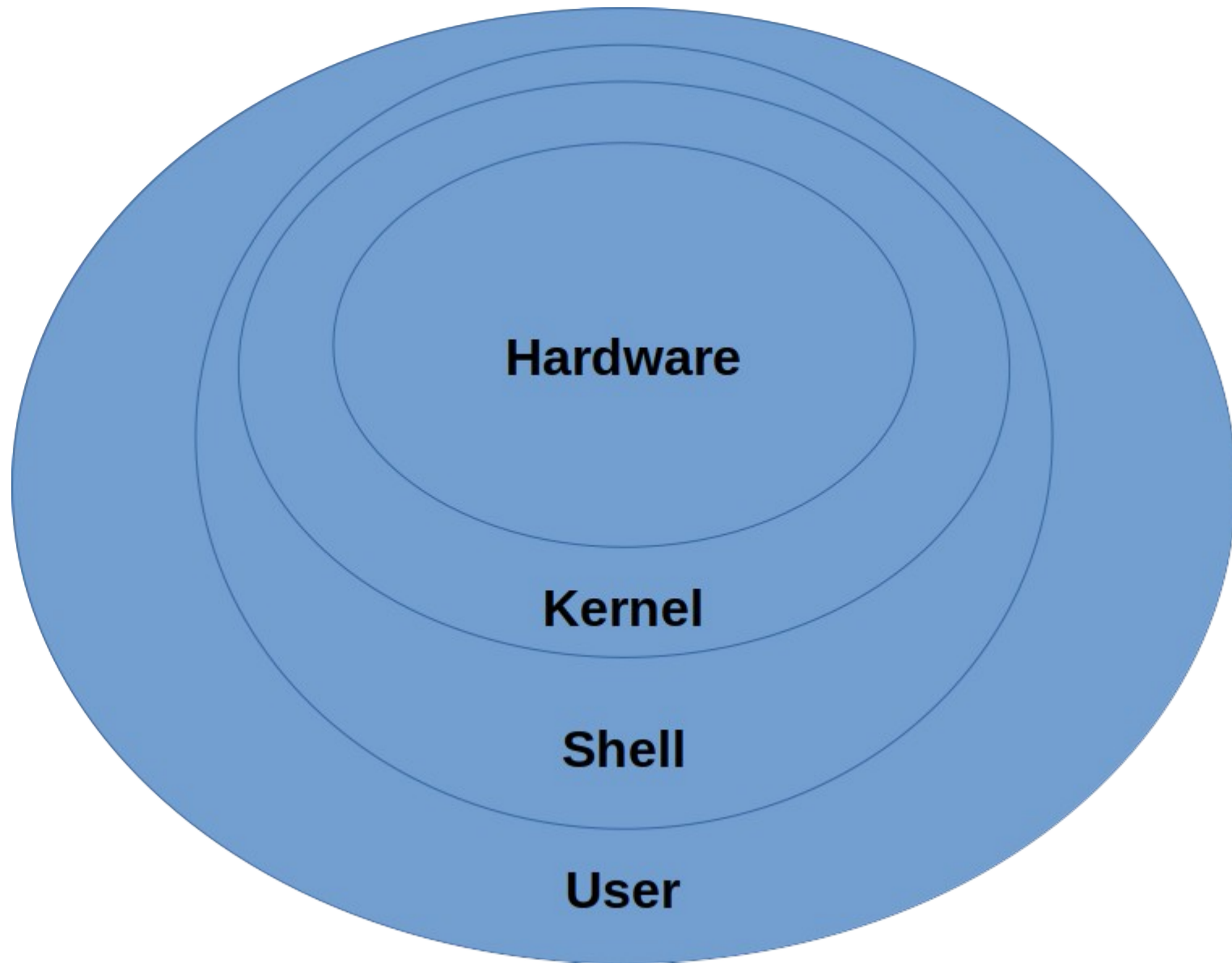
	Pros	Contras
GUI 	<ul style="list-style-type: none">• Fácil de aprender.• Intuitivo• Visualmente agradable• Eficiente pero... depende	<ul style="list-style-type: none">• Incrementa errores humanos• Baja reproducibilidad• Ineficiente para rutinas complejas• Muchas veces, software privativo
Code 	<ul style="list-style-type: none">• Reduce errores humanos• Favorece reproducibilidad• Empodera• Libera• Alfabetiza computacionalmente• Aprendes mas lenguajes.• Incrementa competitividad• Es cool y te hace ver muy inteligente• Muchas opciones software libre	<ul style="list-style-type: none">• Curva de aprendizaje empinada• Inicialmente poco intuitivo• Visualmente poco agradable para no geeks• Requiere paciencia• Poco eficiente para hacer cosas muy sencillas.

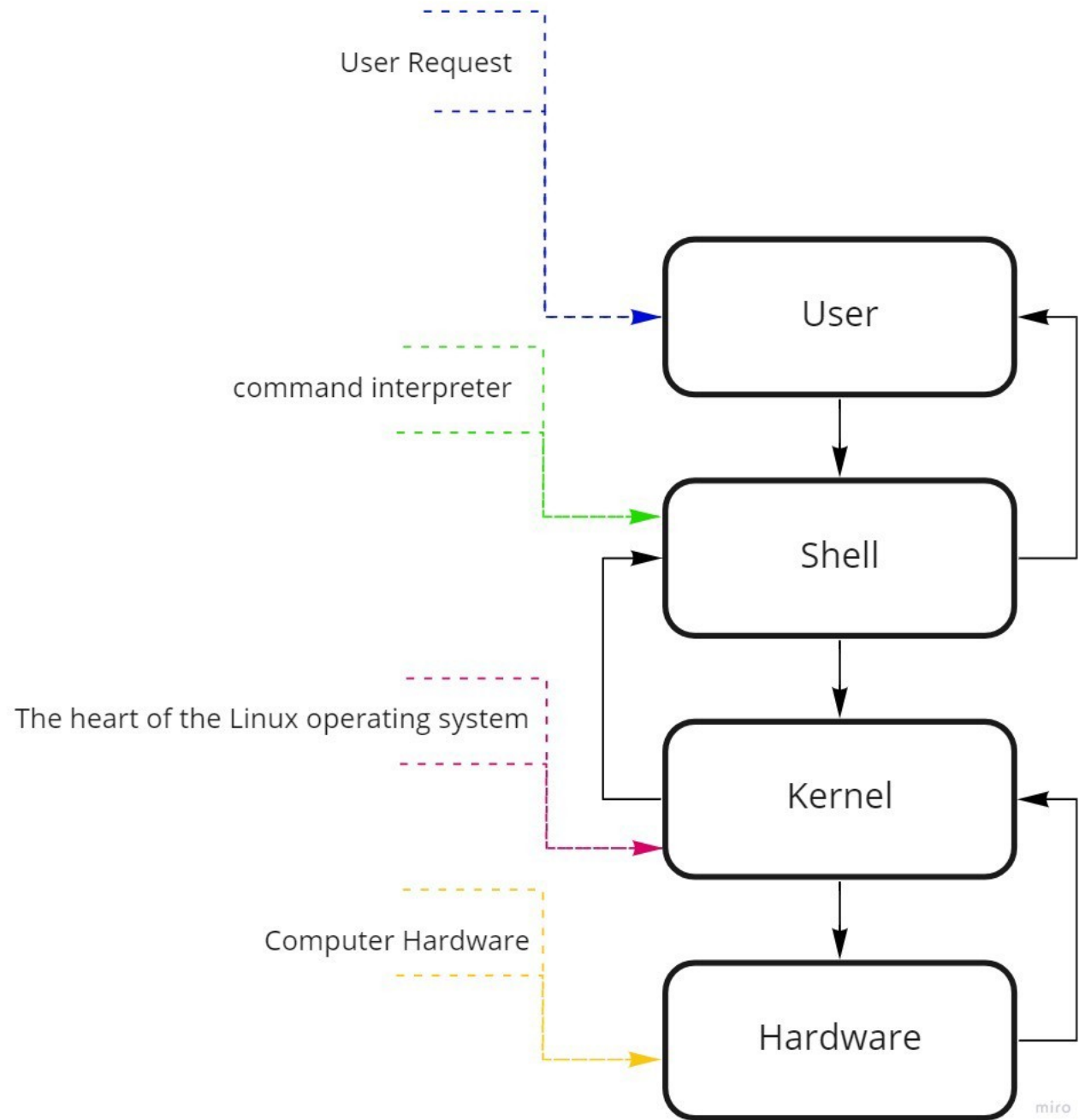
Eficiencia



Kernel: Software que controla la computadora, acceso recursos (memoria, CPU, disco duro, tarjeta de video)

Shell: lee comandos tecleados por el usuario y los ejecuta. Puede ser usado para escribir programas (ej. bash shell)







Hola mundo!

User Request

Source code

```
print ("Hola mundo!")
```



command interpreter

The heart of the Linux operating system

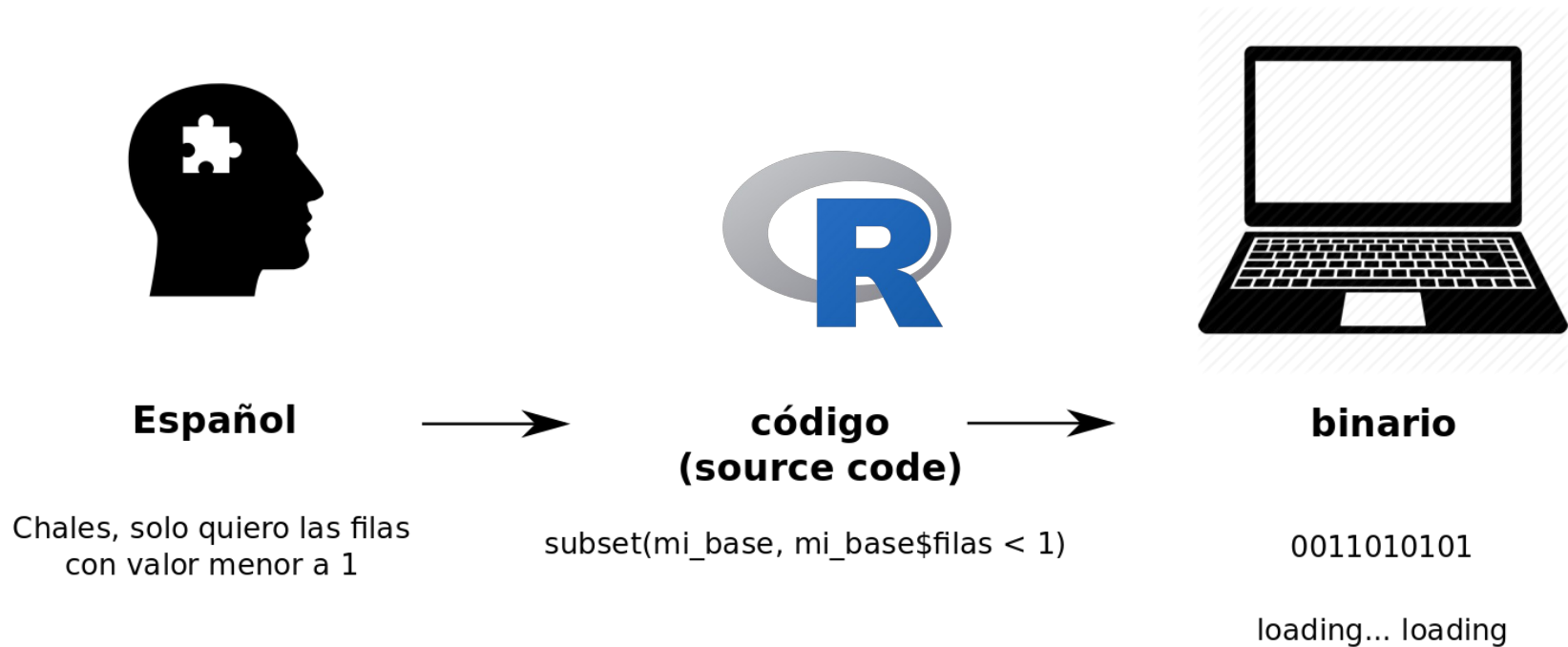
Computer Hardware



miro

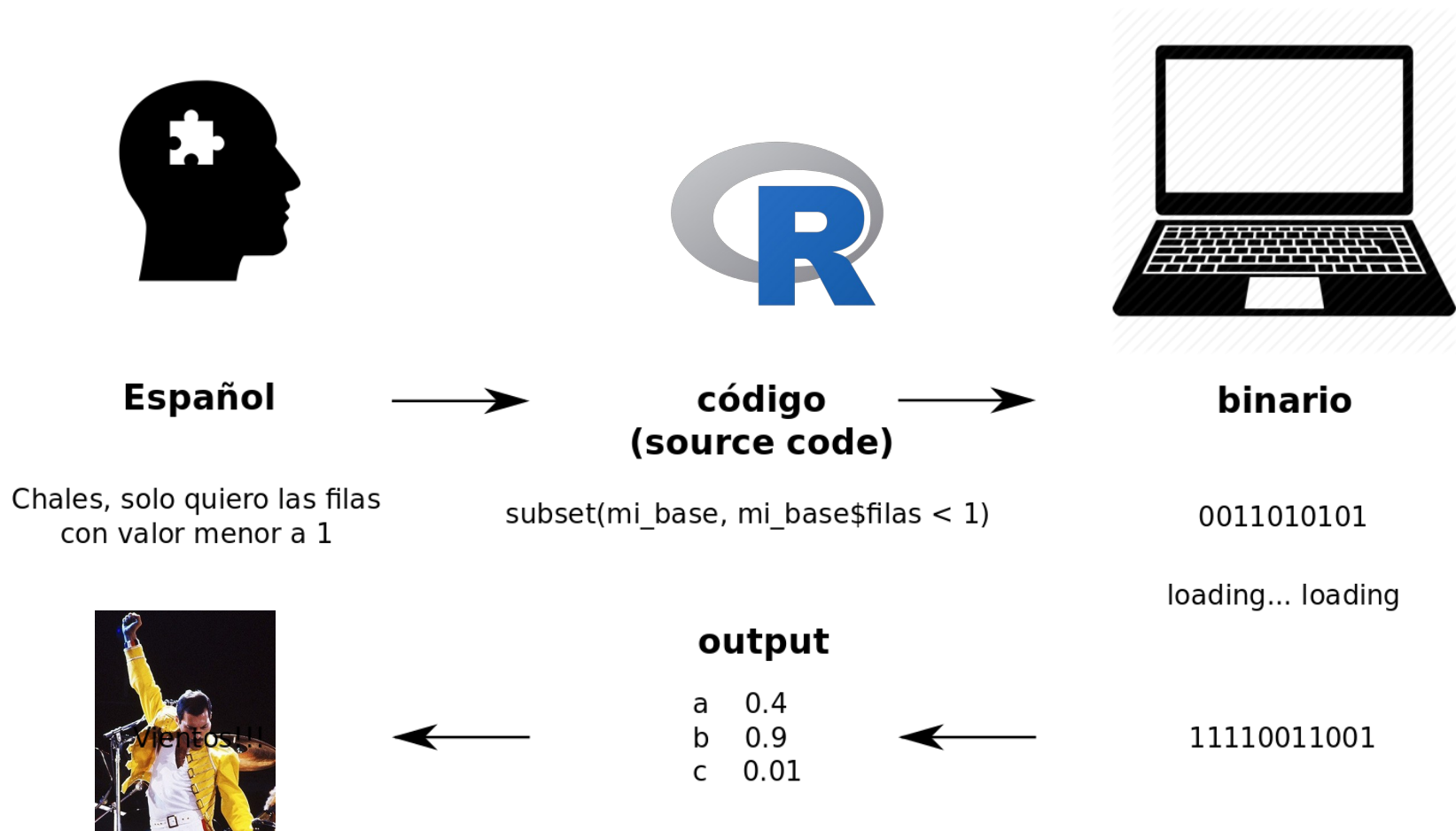
¿Qué es coding?

El poder escribir en un lenguaje que la computadora pueda traducir para luego entender.



¿Qué es coding?

El poder escribir en un lenguaje que la computadora pueda traducir para luego entender.



¿Qué es un programa?



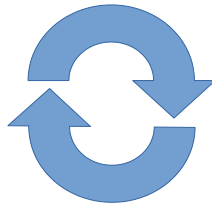
Source Code



Paquetes (ej. lme4)
Librerías

Software (ej. R)

Computadora procesa



Resultado

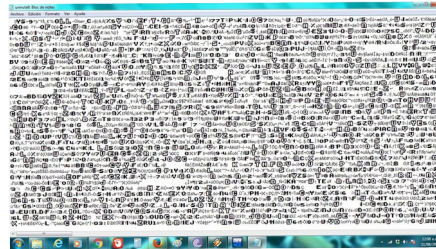
Programas compilados vs interpretadores



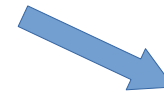
Source code
(text file)

Compila

```
7- # glist Functions -----
8
9- glist <- function(...) {
10   gl <- list(...)
11   if (length(gl) == 0L ||
12       all(sapply(gl, okGlistelt, simplify=TRUE))) {
13     # Ensure glist is "flat"
14     # Don't want glist containing glist ...
15     if (!all(sapply(gl, is.grob)))
16       gl <- do.call("c", lapply(gl, as.glist))
17     class(gl) <- c("glist")
18     return(gl)
19   } else {
20     stop("Only 'grobs' allowed in 'glist'")
21   }
22 }
```



compilados



Programas compilados vs interpretadores



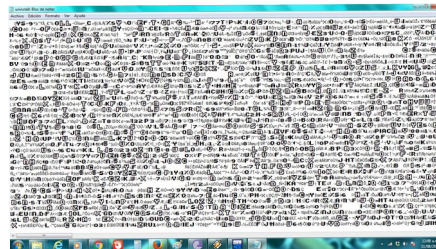
Source code
(text file)

Compila

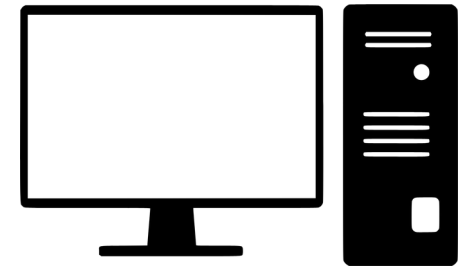


compilados

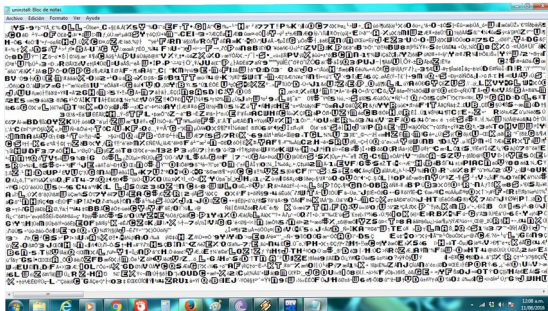
```
7 # glist Functions -----
8
9 glist <- function(...) {
10   gl <- list(...)
11   if (length(gl) == 0L ||
12       all(sapply(gl, okGlistelt, simplify=TRUE))) {
13     # Ensure glist is "Flat"
14     # Don't want glist containing glist ...
15     if (!all(sapply(gl, is.grob)))
16       gl <- do.call("c", lapply(gl, as.glist))
17     class(gl) <- c("glist")
18     return(gl)
19   } else {
20     stop("Only 'grobs' allowed in 'glist'")
21   }
22 }
```



```
7 # glist Functions -----
8
9 glist <- function(...) {
10   gl <- list(...)
11   if (length(gl) == 0L ||
12       all(sapply(gl, okGlistelt, simplify=TRUE))) {
13     # Ensure glist is "Flat"
14     # Don't want glist containing glist ...
15     if (!all(sapply(gl, is.grob)))
16       gl <- do.call("c", lapply(gl, as.glist))
17     class(gl) <- c("glist")
18     return(gl)
19   } else {
20     stop("Only 'grobs' allowed in 'glist'")
21   }
22 }
```



Interprete



Compiled (compilados): Programas que tienen que ser traducidos a lenguaje binario (compilados) para ser ejecutados sin necesidad de traducir de nuevo. Son muy eficientes ya que no se utiliza recurso de procesamiento para traducir más que una sola vez.

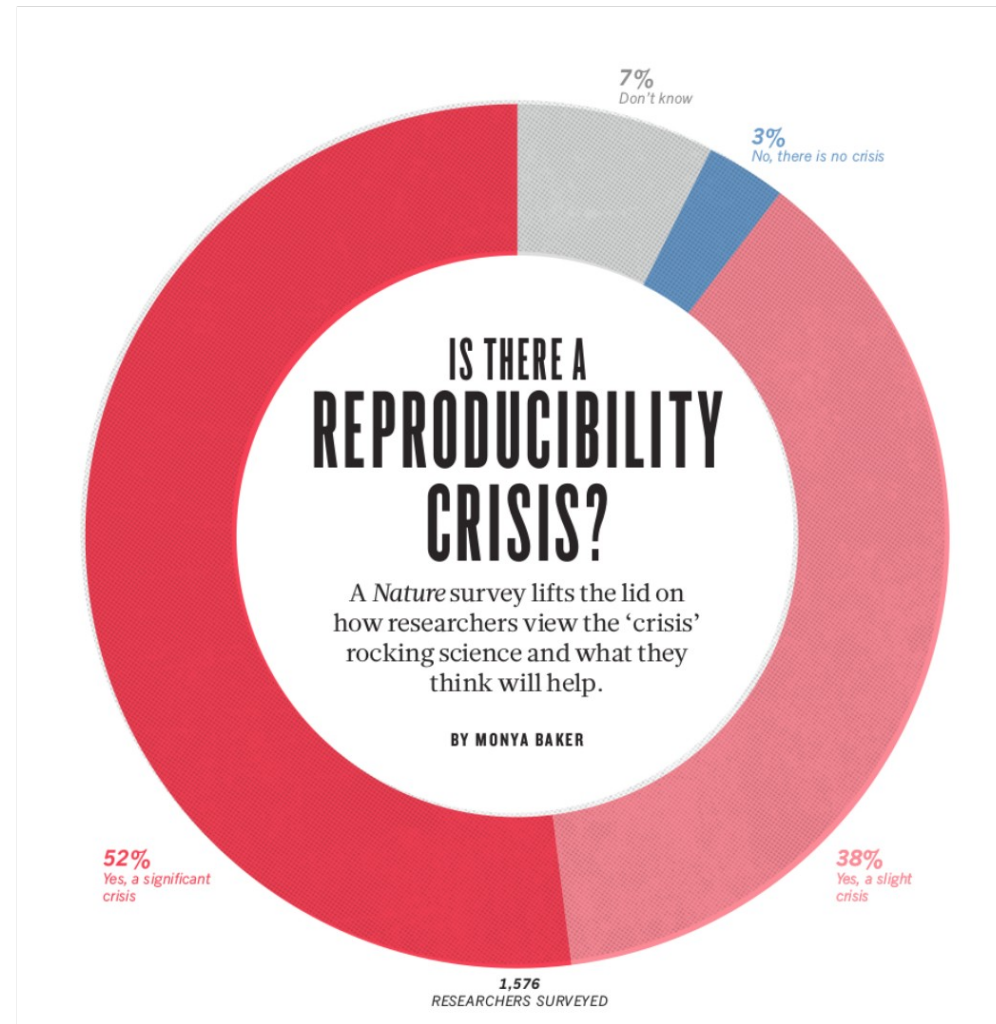
Interpreted (Interpretadores): Programas que a pesar de estar compilados, interpretan el source code. Esto implica que tienen que reinterpretar cada línea de código Cada vez que se ejecuta.

¿Por qué aprender código?

- Favorece reproducibilidad
- Documenta nuestros procedimientos (carpintería de datos, análisis, comunicación de resultados)
- Abre las puertas a aprender nuevos lenguajes y ser más competitivo
- Empodera



Cr sis de reproducibilidad



De 1,576 investigadores encuestados 70% no han podido replicar el trabajo de otros, 50% ni su propio trabajo.

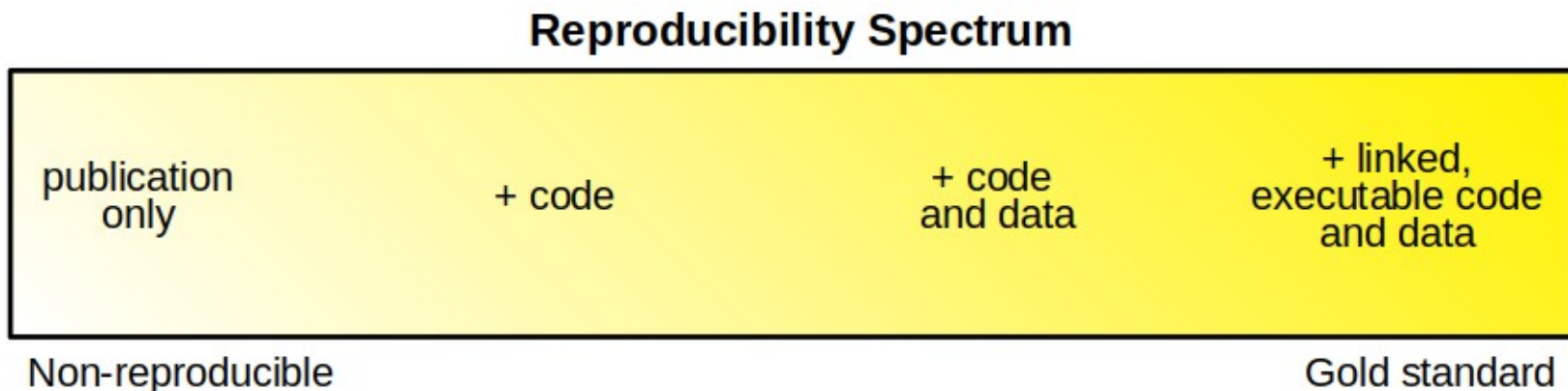
Sobre reproducibilidad

“Non-reproducible single occurrences are of no significance to science.”

Popper (1959)

Reproducibilidad: Requiere mínimos ajustes para obtener los mismos resultados. Implica que el fenómeno puede ser obtenido con pequeñas diferencias en las condiciones experimentales.

Replicabilidad: Requiere que las condiciones sean exactas para obtener los mismos Resultados.



Recomendaciones para reproducibilidad

- **Escribe código** para humanos, no para computadoras
- Deja que la computadora haga el trabajo pesado
- Invierte tiempo en desarrollar funciones que usarás frecuentemente
- Ordena cada proyecto en un directorio particular
- Trabaja con *relative paths*, evita *absolute paths*
- En ese directorio asegurate de tener.
 - .Rproj el file del proyecto R
 - Bases de datos (.xls, .csv, .txt, etc)
 - .R el file donde estará escrito el código
 - .Rdata files si es que son necesarios, que contienen el ambiente y objetos de la sesión
 - README.txt file dando información sobre:
 - Origen de datos
 - ¿Dónde están los files (path)
 - ¿Qué contienen los files?
 - ¿Qué parámetros se usaron para cada programa?
 - ¿Qué versión del programa se uso?
 - ¿Qué librerías y con qué fin se usaron?
- Documenta todo (notebook, ej. Rstudio, Jupyter)
- Se consistente con las secciones de tu código y con el estilo que uses.
- Escribe funciones. Si repites más de dos veces un procedimiento haz una función.
- Crea código que permita que las pruebas estadísticas, tablas, y figuras sean reproducirles.
- Si es posible, busca peer-reviewers (amigos) que chequen código, legibilidad y reproducibilidad.

Herramientas importantes para reproducibilidad

Rstudio



- Ambiente de trabajo con R

Markdown



- Comunicar resultados
- Notebook

Github



- Control de versiones
- Favorece colaboración
- Repositorio de programas importantes

Ejemplo Markdown

Diego Carmona

April 19, 2019

Indice

- 3.2.1. ¿Qué es una lista?
- 3.2.2. ¿Cómo crear una lista?
- 3.2.3. Nombrando elementos de la lista
- 3.2.4. Accediendo a elementos de la lista (indexación)
- 3.2.5. Manipulación de elementos de la lista
- 3.2.6. Uniendo listas
- 3.2.7. Convertir listas a vector
- 3.2.8. Aplicar funciones a elementos en la lista
- 3.2.9. Convertir una lista en dataframe

3.2.1. ¿Qué es una lista?

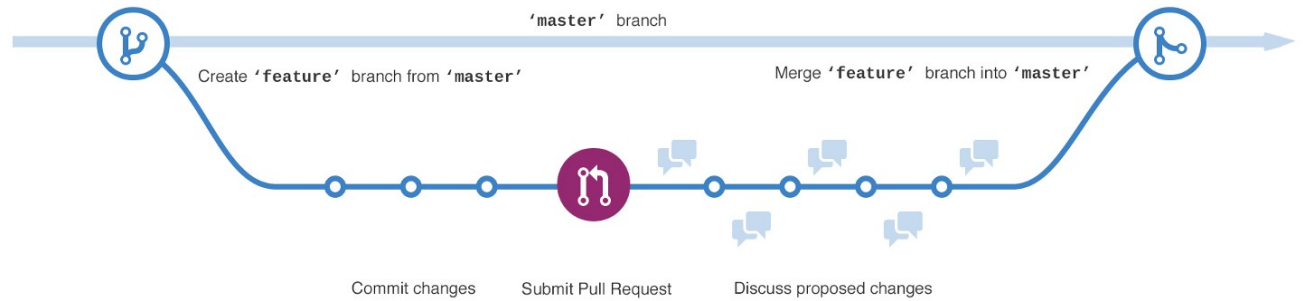
Las **listas** son un tipo de vector (vector genérico) en el cual los elementos contenidos pueden ser de diferente tipo tales como integers, double, characters, hasta otras listas, vectores atómicos, matrices, e incluso bases de datos. Las listas son fundamentales para armar data frames, y programación orientada a objetos.

¿Cómo crear una lista?

Podemos usar dos funciones: 1: `list()` 2: `vector()`

Usando `list()`

```
# Creando lista con elementos
gene_info_index<-list ("F4HVV5", "VAD1", "Protein Vascular Associated death 1",
                      "A thaliana",
                      "MAMLSTASVSGSVDLPRGTMKVDSSASPEVVSDLPPSSPKGSPDRHDPSTSSPSPSRG",
                      598)
gene_info_index
gene_info_index[1] # Se puede acceder al elemento con []
gene_info_index[[1]] # o dos [], el uso de múltiples corchetes reconoce la estructura de la lista
```



"FINAL".doc



FINAL.doc!



FINAL_rev.2.doc



FINAL_rev.6.COMMENTS.doc



FINAL_rev.8.comments5.
CORRECTIONS.doc



FINAL_rev.18.comments7.
corrections9.MORE.30.doc



FINAL_rev.22.comments49.
corrections.10. #@\$%WHYDID
ICOMETOGRADSCHOOL?????.doc



JORGE CHAM © 2012

Actividad

Ensayo sobre software libre vs. software privativo.

Párrafo 1: Introducción

- Punto general de reflexión

- Introduce en general pros y contras de software libre y privativo

- Introduce el punto que sea de interés para ti, estas a favor o en contra

Párrafo 2: Enfocate en Software libre

- Da elementos sobre software libre

- Pros y contras

Párrafo 3: Enfocate en Software privativo

- Da elementos sobre software privativo

- Pros y contras

Párrafo 4: Contrasta Software libre y privativo

- Argumenta este contraste y utilízalo para soportar

- Tu punto de vista introducido en el párrafo 1

Párrafo 5: Concluye apoyando tu postura.

Máximo 300 palabras por párrafo.

Ensayo sobre software libre vs. software privativo.

Párrafo 1: Introducción

- Punto general de reflexión

- Introduce en general pros y contras de software libre y privativo

- Introduce el punto que sea de interés para ti, estas a favor o en contra

Párrafo 2: Enfocate en Software libre

- Da elementos sobre software libre

- Pros y contras

Párrafo 3: Enfocate en Software privativo

- Da elementos sobre software privativo

- Pros y contras

Párrafo 4: Contrasta Software libre y privativo

- Argumenta este contraste y utilízalo para soportar

- Tu punto de vista introducido en el párrafo 1

Párrafo 5: Concluye apoyando tu postura.

Máximo 300 palabras por párrafo.

Videos de apoyo

Richard Stallman (en ingles, en español; respectivamente)

<https://www.youtube.com/watch?v=jUibaPTXSHk>

<https://www.youtube.com/watch?v=onrIApj3Cjs&t=48s>

Linus Torvals TED talk

<https://www.youtube.com/watch?v=MNXIXDbEmVc&t=206S>

Documental sobre GNU-Linux

https://www.youtube.com/watch?v=9ip3UA_04LM

De acuerdo con Richard Stallman qué gran ventaja del free software no es discutida en este Video

<https://www.youtube.com/watch?v=2q91vTvc7YE>

