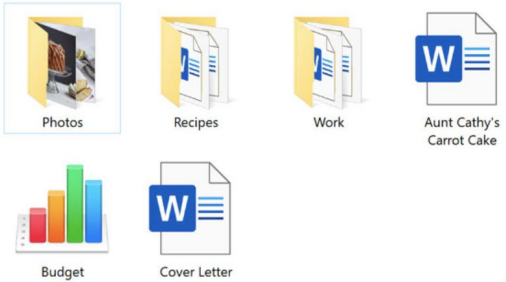
Clase 5. Manejo de archivos

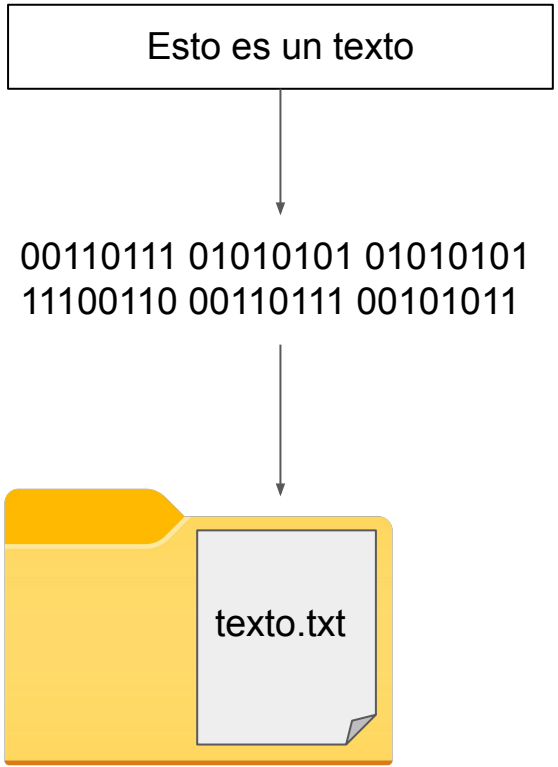
Archivos

- Un archivo es una secuencia de **bytes** que es almacenado en un dispositivo. Un archivo se identifica por su nombre y por el directorio que lo compone.
- Cada tipo de archivo se almacena de forma distinta, siempre se **codifican** a código binario para almacenarse.

Datos (lo que vemos)	
Archivos	00110111 01010101 01010101 11100110 00110111 00101011

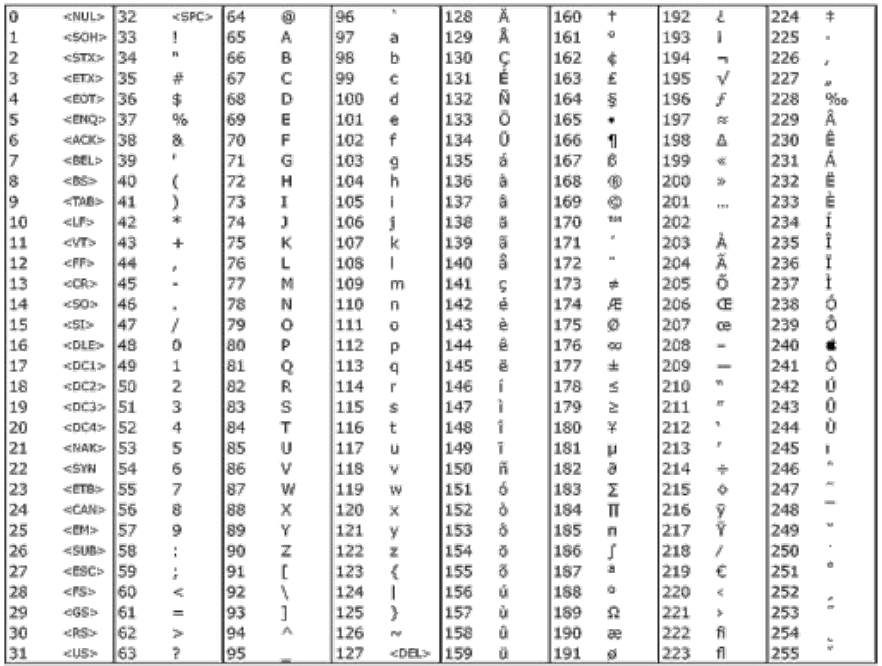
Archivos:
texto simple

- Un archivo simple (sin estilos, solamente caracteres) se codifica a bytes, y luego se almacena en el sistema de archivos en un directorio con la extensión **.txt**



Archivos:
texto simple

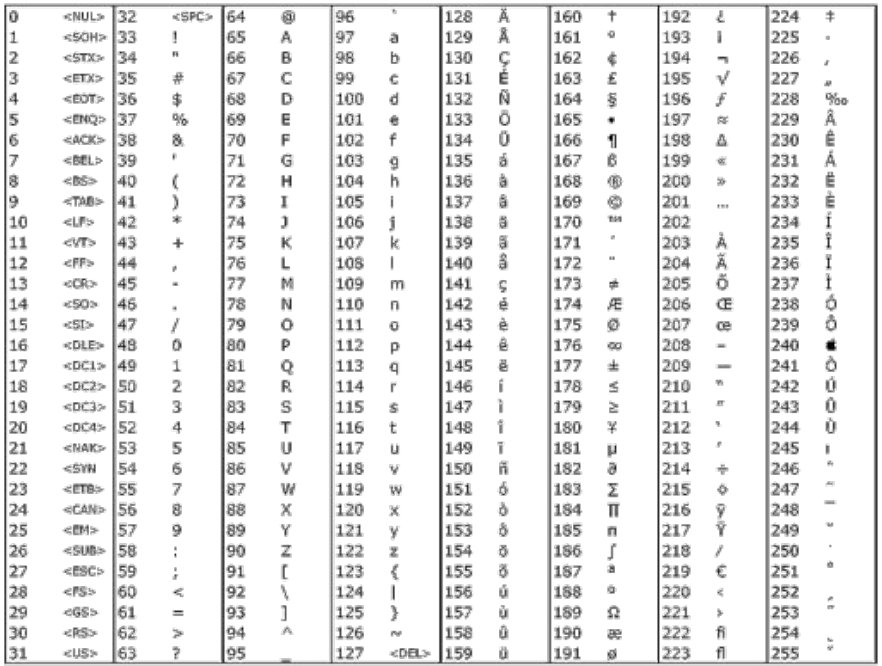
- Como se convierte un caracter en código binario? Se usan **estándares** que muestran como convertir cada caracter. Los estándares son convenciones que deberían ser universales (pero hoy no lo son).
- El estándar **ASCII** fue uno de los primeros en implementarse. Asigna a cada caracter un número entre 0-255 y luego lo convierte a binario, al usar ese rango se asegura que se pueda representar en un byte.



0	<NUL>	32	<SPC>	64	@	96	`	128	Ä	160	†	192	¿	224	‡
1	<SOH>	33	!	65	A	97	a	129	Å	161	°	193	¡	225	·
2	<STX>	34	"	66	B	98	b	130	Ç	162	¢	194	¬	226	‚
3	<ETX>	35	#	67	C	99	c	131	É	163	£	195	√	227	„
4	<EOT>	36	$	68	D	100	d	132	Ñ	164	§	196	ƒ	228	‰
5	<ENQ>	37	%	69	E	101	e	133	Ö	165	•	197	≈	229	Â
6	<ACK>	38	&	70	F	102	f	134	Ü	166	¶	198	Δ	230	Ê
7	<BEL>	39	'	71	G	103	g	135	á	167	ß	199	«	231	Á
8	<BS>	40	(	72	H	104	h	136	à	168	®	200	»	232	Ë
9	<TAB>	41	)	73	I	105	i	137	â	169	©	201	…	233	È
10	<LF>	42	*	74	J	106	j	138	ä	170	™	202		234	Í
11	<VT>	43	+	75	K	107	k	139	ã	171	´	203	À	235	Î
12	<FF>	44	,	76	L	108	l	140	å	172	¨	204	Ã	236	Ï
13	<CR>	45	-	77	M	109	m	141	ç	173	≠	205	Õ	237	Ì
14	<SO>	46	.	78	N	110	n	142	é	174	Æ	206	Œ	238	Ó
15	<SI>	47	/	79	O	111	o	143	è	175	Ø	207	œ	239	Ô
16	<DLE>	48	0	80	P	112	p	144	ê	176	∞	208	–	240	
17	<DC1>	49	1	81	Q	113	q	145	ë	177	±	209	—	241	Ò
18	<DC2>	50	2	82	R	114	r	146	í	178	≤	210	"	242	Ú
19	<DC3>	51	3	83	S	115	s	147	ì	179	≥	211	"	243	Û
20	<DC4>	52	4	84	T	116	t	148	î	180	¥	212	'	244	Ù
21	<NAK>	53	5	85	U	117	u	149	ï	181	µ	213	'	245	ı
22	<SYN	54	6	86	V	118	v	150	ñ	182	∂	214	÷	246	ˆ
23	<ETB>	55	7	87	W	119	w	151	ó	183	∑	215	◊	247	˜
24	<CAN>	56	8	88	X	120	x	152	ò	184	∏	216	ÿ	248	¯
25	<EM>	57	9	89	Y	121	y	153	ô	185	π	217	Ÿ	249	˘
26	<SUB>	58	:	90	Z	122	z	154	ö	186	∫	218	/	250	˙
27	<ESC>	59	;	91	[	123	{	155	õ	187	ª	219	€	251	˚
28	<FS>	60	<	92	\	124	|	156	ú	188	º	220	‹	252	¸
29	<GS>	61	=	93	]	125	}	157	ù	189	Ω	221	›	253	˝
30	<RS>	62	>	94	^	126	~	158	û	190	æ	222	fi	254	˛
31	<US>	63	?	95	_	127	<DEL>	159	ü	191	ø	223	fl	255	ˇ

Archivos:
texto simple

- Como se puede ver, hay caracteres no visibles por los usuarios que también tienen un código.
- Los caracteres no visibles incluyen:
  - Fin de línea
  - Tabulador
  - Fin de archivo
- El “fin de línea” se representa con:
  - ***\n*** en sistemas Unix (Ubuntu, Linux)
  - ***\r*** en Macintosh
  - ***\r\n*** en Windows
- Deberían ser los mismos sin importar el SO, pero no lo son



0	<NUL>	32	<SPC>	64	@	96	`	128	Ä	160	†	192	¿	224	‡
1	<SOH>	33	!	65	A	97	a	129	Å	161	°	193	¡	225	·
2	<STX>	34	"	66	B	98	b	130	Ç	162	¢	194	¬	226	‚
3	<ETX>	35	#	67	C	99	c	131	É	163	£	195	√	227	„
4	<EOT>	36	$	68	D	100	d	132	Ñ	164	§	196	ƒ	228	‰
5	<ENQ>	37	%	69	E	101	e	133	Ö	165	•	197	≈	229	Â
6	<ACK>	38	&	70	F	102	f	134	Ü	166	¶	198	Δ	230	Ê
7	<BEL>	39	'	71	G	103	g	135	á	167	ß	199	«	231	Á
8	<BS>	40	(	72	H	104	h	136	à	168	®	200	»	232	Ë
9	<TAB>	41	)	73	I	105	i	137	â	169	©	201	…	233	È
10	<LF>	42	*	74	J	106	j	138	ä	170	™	202		234	Í
11	<VT>	43	+	75	K	107	k	139	ã	171	´	203	À	235	Î
12	<FF>	44	,	76	L	108	l	140	å	172	¨	204	Ã	236	Ï
13	<CR>	45	-	77	M	109	m	141	ç	173	≠	205	Õ	237	Ì
14	<SO>	46	.	78	N	110	n	142	é	174	Æ	206	Œ	238	Ó
15	<SI>	47	/	79	O	111	o	143	è	175	Ø	207	œ	239	Ô
16	<DLE>	48	0	80	P	112	p	144	ê	176	∞	208	–	240	
17	<DC1>	49	1	81	Q	113	q	145	ë	177	±	209	—	241	Ò
18	<DC2>	50	2	82	R	114	r	146	í	178	≤	210	“	242	Ú
19	<DC3>	51	3	83	S	115	s	147	ì	179	≥	211	”	243	Û
20	<DC4>	52	4	84	T	116	t	148	î	180	¥	212	‘	244	Ù
21	<NAK>	53	5	85	U	117	u	149	ï	181	µ	213	’	245	ı
22	<SYN	54	6	86	V	118	v	150	ñ	182	∂	214	÷	246	ˆ
23	<ETB>	55	7	87	W	119	w	151	ó	183	Σ	215	◊	247	˜
24	<CAN>	56	8	88	X	120	x	152	ò	184	∏	216	ÿ	248	¯
25	<EM>	57	9	89	Y	121	y	153	ô	185	π	217	Ÿ	249	˘
26	<SUB>	58	:	90	Z	122	z	154	ö	186	∫	218	⁄	250	˙
27	<ESC>	59	;	91	[	123	{	155	õ	187	ª	219	€	251	˚
28	<FS>	60	<	92	\	124	|	156	ú	188	º	220	‹	252	¸
29	<GS>	61	=	93	]	125	}	157	ù	189	Ω	221	›	253	˝
30	<RS>	62	>	94	^	126	~	158	û	190	æ	222	ﬁ	254	˛
31	<US>	63	?	95	_	127	<DEL>	159	ü	191	ø	223	ﬂ	255	ˇ

Ejercicio

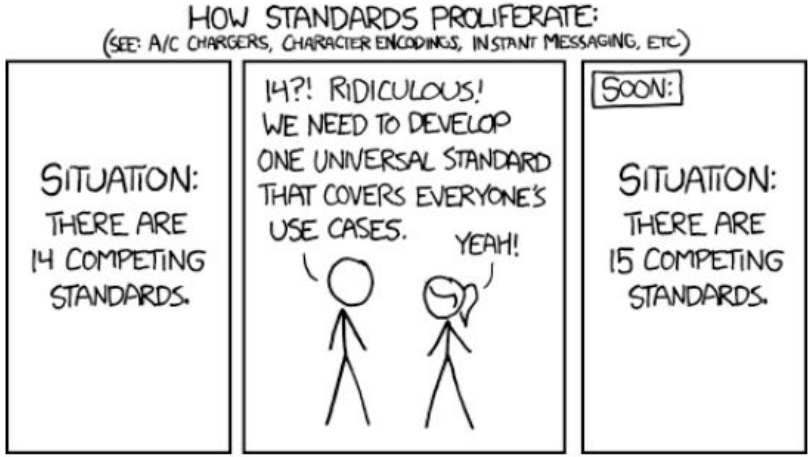
Tiempo ⏱️

Buscar los códigos ASCII para los siguientes caracteres:

- A
- a
- @
- L
- 😱

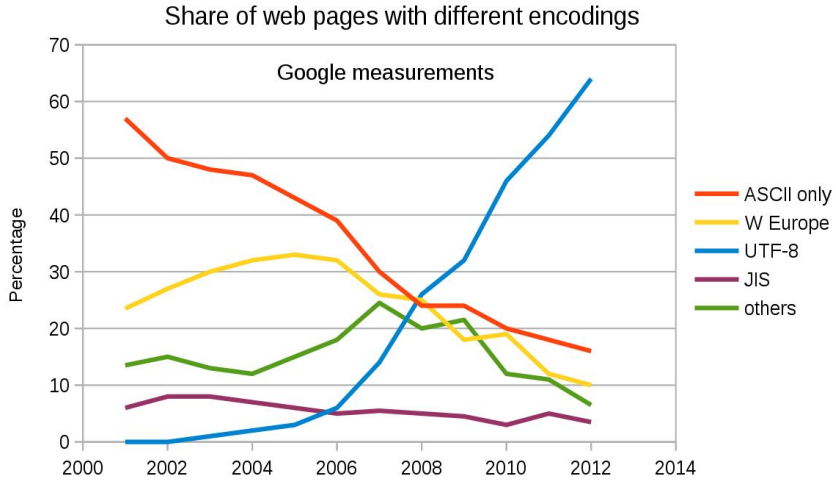
Archivos:
texto simple

- El estándar ASCII es finito (hasta 256 caracteres) por lo que no puede incluir nuevos caracteres como letras en otros alfabetos, emojis, etc.
- Se crearon otros estándares para poder representar más caracteres. Pero en lugar de universalizarlo lo diversificaron.



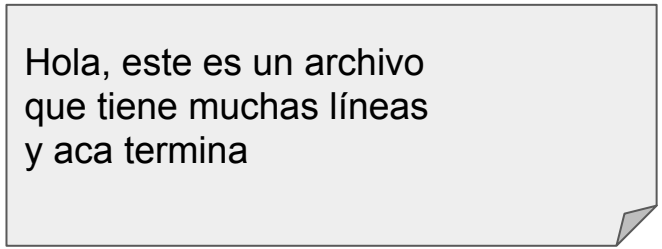
Archivos:
texto simple

- Hoy en dia el mas utilizado es **UTF-8** que utiliza de 1 a 4 bytes para cada caracter. Permite representar todos los caracteres de hoy en dia.
- Incluye dentro al estándar ASCII, por lo que un archivo codificado en ASCII puede ser interpretado por un intérprete de UTF-8.
- Usualmente las librerías para leer/escribir archivos, nos van a pedir que especifiquemos qué estándar vamos a utilizar.



Abrir archivos

- Para abrir un archivo para empezar a operar en él se utilizar la función **open**.
- Recibe dos parámetros:
  - *file*: Ruta al archivo
  - *modo*: Modo de uso del archivo:
    - **r**: (default) Solo lectura. Levanta una excepción si falla.
    - **w**: Solo escritura (bloquea el archivo primero). Lo crea si no existe.
    - Mas modos
- Retorna un archivo



Python

```
archivo = open("texto.txt")
print(archivo)
# <_io.TextIOWrapper name='texto.txt'
# mode='r' encoding='UTF-8'>
archivo.close()


archivo = open("archivoInexistente.txt")
# Traceback (most recent call last):
#   File "archivos.py", line 1, in <module>
#    archivo = open("archivoInexistente.txt")
# FileNotFoundError: [Errno 2] No such file
# or directory: 'archivoInexistente.txt'
```

Abrir archivos

Hola, este es un archivo
que tiene muchas líneas
y aca termina

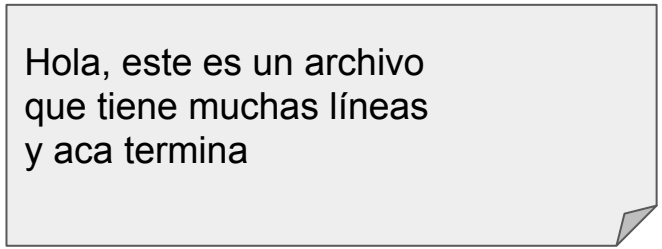
- Nunca olvidar de cerrar un archivo usando **close** por múltiples razones:
  - Muchos archivos abiertos hace que el programa tenga peor performance.
  - Muchas veces, los cambios realizados en el archivo no se hacen efectivos hasta cerrarlos.
  - Algunas acciones bloquean el archivo, por lo que se vuelven no modificables hasta que se cierre.

Python 

```
archivo = open("texto.txt")
print(archivo)
# <_io.TextIOWrapper name='texto.txt'
# mode='r' encoding='UTF-8'>
archivo.close()


archivo = open("archivoInexistente.txt")
# Traceback (most recent call last):
#   File "archivos.py", line 1, in <module>
#    archivo = open("archivoInexistente.txt")
# FileNotFoundError: [Errno 2] No such file
# or directory: 'archivoInexistente.txt'
```

Leer archivos



- Función **read**: Lee todo el archivo y guarda el contenido en una variable.
- **CUIDADO**: Leer todo el archivo junto y guardarlo en memoria puede ser **muy** peligroso si el archivo es grande, puede no entrar en memoria por ejemplo. Usualmente se lee de a partes. Por este motivo esta función **no está recomendada**.

Python

```
archivo = open("texto.txt")
texto = archivo.read()
print(texto)
# Hola, este es un archivo
# que tiene muchas líneas
# y aca termina
```

Leer archivos

- Función **readline**: Lee la siguiente línea no leída y la retorna. Si llego al final del archivo retorna string vacío.
- Lo que hace es leer de a un caracter hasta encontrarse con el caracter “fin de línea”, que depende del SO. Nosotros vamos a usar “\n”.
- Es una opción mucho más segura y más amigable con la memoria.
- Para leer todas se puede hacer un bucle.

Python

```
archivo = open("texto.txt")
linea = archivo.readline()
print(linea)
# Hola, este es un archivo
linea = archivo.readline()
print(linea)
# que tiene muchas líneas
archivo.close()


archivo = open("texto.txt")
linea = archivo.readline()
while (linea != ''):
 print(linea)
 linea = archivo.readline()
# Hola, este es un archivo
#
# que tiene muchas líneas
#
# y aca termina
#
archivo.close()
```

Leer archivos

- Función **readline**: Lee la siguiente línea no leída y la retorna. Si llego al final del archivo retorna string vacío.
- Lo que hace es leer de a un caracter hasta encontrarse con el caracter “fin de línea”, que depende del SO. Nosotros vamos a usar “\n”.
- Es una opción mucho más segura y más amigable con la memoria.
- Para leer todas se puede hacer un bucle.

Python

```
archivo = open("texto.txt")
for linea in archivo:
 print(linea)
# Hola, este es un archivo
#
# que tiene muchas líneas
#
# y aca termina
archivo.close()
```

with…as…:

- La estructura ***with…….as…..:*** nos sirve para manejar recursos. Lo que nos garantiza es que al finalizar “limpia” el uso del recurso, vamos a usarlo para que cierre los archivos y no tener que usar el close.
- El archivo no estará más abierto luego del bloque **with...as...**

Python

```
with open("texto.txt") as archivo:
 for linea in archivo:
   print(linea)
# Hola, este es un archivo
#
# que tiene muchas líneas
#
# y aca termina
~~archivo.close()~~ ===> No es más necesario, el
with va a cerrarlo internamente.
```

Ejercicio

Tiempo ⏱️

- Se dispone de un archivo con números, uno por línea. Escribir una función que reciba la ruta al archivo y retorne la suma de los números de dicho archivo.

Escribir archivos

- Abrir el archivo en modo “w” (escribir y pisar el contenido anterior) o “a” (agrega al final del archivo).
- Función **write** recibe como parámetro el dato a escribir. Si no existe el archivo lo crea.
- Función **writelines** recibe como parámetro una lista de datos a escribir.

Python

```
with open("textoDeEscritura.txt", "w") as archivo:
 archivo.write("Esto es una frase.")
 archivo.write("Con esta frase termina la linea \n")
 archivo.write("Nueva linea")
```

Esto es una frase.Con esta frase termina la linea
Nueva linea

Ejercicio

Tiempo ⏱️

Escribir una función que a partir de una lista y un nombre de archivo, escriba cada elemento de la lista en un renglón **debajo** de las líneas ya existentes del archivo (no debe sobreescribirlo). En caso que el archivo no exista deberá crearlo.

Como guardar nuestros datos?

- La gran mayoría de veces, no es requerido guardar un archivo de texto simple, si no un conjunto de datos estructurados. Puede usarse un archivo simple como base de datos o algún motor de base de datos.

Datos estructurados: CSV

- CSV = Comma Separated Values (Valores separados por coma).
- Cala línea es una entidad nueva y los valores se separan por comas (aunque puede elegirse otro separador).
- La primer línea tiene los títulos de cada columna para que sepamos el sentido.
- Si una entidad no tiene una propiedad, se puede dejar en blanco pero hay que mantener la estructura (Observar en el ejemplo las observaciones de la jirafa).

```
id,nombre,color,patas,observaciones,rayas
1,zebra,blanco y negro,4,viven en manada,si
2,jirafa,amarillo,4,,no
3,rinoceronte,gris,son muy fuertes,no
```

Datos estructurados: JSON

- JSON: JavaScript Object Notation.
- Es un vector de diccionarios. Donde cada elemento del vector es una entidad y cada propiedad del diccionario es una propiedad de la entidad.
- En cada entidad hay que repetir el título de la propiedad, en un CSV los titulos estan en la primer línea y no se repiten luego.
- No es necesario poner propiedades vacías (en el ejemplo: no todos los animales tienen “observaciones”).

```
[
  {
    "id": 1,
    "nombre": "zebra",
    "color": "blanco y negro",
    "patas": 4,
    "observaciones": "viven en manada",
    "rayas": si
  },
  {
    "id": 2,
    "nombre": "jirafa",
    "color": "amarillo",
    "patas": 4,
    "rayas": no
  },
    {
    "id": 3,
    "nombre": "rinoceronte",
    "color": "gris",
    "patas": 4,
    "observaciones": "son muy fuertes",
    "rayas": si
  }
]
```

Datos estructurados: XML

- XML: Extensible Markup Language
- Similar en conceptos al JSON.
- Se guardan los datos en etiquetas. En el ejemplo:
  - La etiqueta “animales” contienen dentro multiples etiquetas “animal”.
  - Cada etiqueta “animal” tiene dentro las propiedades de cada animal.

```
<animales>
  <animal id="1"
    nombre="zebra"
    color="blanco y negro"
    patas=4
    observaciones="viven en manada"
    rayas="si"
  />
  <animal id="2"
    nombre="jirafa"
    color="amarillo"
    patas=4
    rayas="no"
  />
  <animal id="3"
    nombre="rinoceronte"
    color="gris"
    patas=4
    observaciones="son muy fuertes"
    rayas="no"
  />
</animales>
```

Como guardar nuestros datos?

- Tenemos que implementar algoritmos que sepan leer y escribir para todas esas estructuras de datos? **NO!**
- Python ya tiene librerías para cada uno.
  - *import json*
  - *import xml*
  - *import csv*

Leer CSV

- Vamos a usar la librería ***csv***
- csv.**reader**: Retorna un iterable que en cada iteración va a leer una línea del archivo csv y lo convierte en una lista.
- Como parámetro opcional se le puede indicar cual es el separador. Default: “,”.
- Atención que la primer línea son los títulos.

Python

```
import csv

with open("datos.csv") as archivo:
 lector = csv.reader(archivo)
 for linea in lector:
   print(linea)

Salida:
['id', 'nombre', 'color', 'patas', 'observaciones',
'rayas']
['1', 'zebra', 'blanco y negro', '4', 'viven en
manada', 'si']
['2', 'jirafa', 'amarillo', '4', '', 'no']
['3', 'rinoceronte', 'gris', 'son muy fuertes', 'no']
```

Leer CSV

- Vamos a usar la librería ***csv***
- csv.**DictReader**: Retorna un iterable que en cada iteración va a leer una línea del archivo csv y lo convierte en un diccionario.

Python

```
import csv

with open("datos.csv") as archivo:
 lector = csv.DictReader(archivo)
 for linea in lector:
   print(linea)
   print(linea[“nombre”])

Salida:
OrderedDict([('id', '1'), ('nombre', 'zebra'), ('color', 'blanco y
negro'), ('patas', '4'), ('observaciones', 'viven en manada'), ('rayas',
'si')])
zebra
OrderedDict([('id', '2'), ('nombre', 'jirafa'), ('color', 'amarillo'),
('patas', '4'), ('observaciones', ''), ('rayas', 'no')])
jirafa
OrderedDict([('id', '3'), ('nombre', 'rinoceronte'), ('color', 'gris'),
('patas', 'son muy fuertes'), ('observaciones', 'no'), ('rayas', None)])
rinoceronte
```

Leer CSV

- Vamos a usar la librería ***csv***
- csv.**writer**: Recibe como parámetro el archivo a leer. Retorna escritor que contiene la función **writeRow** que recibe como parámetro los datos a escribir en formato de lista.
- También contiene la función **writeRows** que recibe una lista donde cada elemento va a escribirse como una línea (deberían ser listas).

Python

```
import csv

with open("escribir.csv", "w") as archivo:
 escritor = csv.writer(archivo)
 escritor.writerow(["titulo1", "titulo2", "titulo3"])
 escritor.writerows([
   ["dato1", "dato2", "dato3"],
   ["lapicera", "medio", "fin"],
 ])

Resultado:
titulo1,titulo2,titulo3
dato1,dato2,dato3
lapicera,medio,fin
```

Escribir CSV

- csv.**DictWriter**: Recibe como parámetros el archivo y los títulos en formato de lista. Retorna escritor que contiene las siguientes funciones:
  - **writeheader**: Escribe los títulos en una nueva línea.
  - **writeRow**: Recibe como parámetro un diccionario a escribir y lo escribe en una nueva línea.
  - **writeRows:** Recibe una lista donde cada elemento se escribirá como una línea (deberían ser diccionarios).

Python

```
import csv

with open("escribir.csv", "w") as archivo:
 titulos = ["id", "nombre", "color"]
 escritor = csv.DictWriter(archivo, fieldnames=titulos)
 escritor.writeheader()
 escritor.writerow({
   "id": 1, "nombre": "zebra", "color": "blanco y negro"
 })
 escritor.writerows([
   {"id": 2, "nombre": "jirafa", "color": "amarillo"},
   {"id": 3, "nombre": "rinoceronte", "color": "gris"}
 ])

Resultado:
id,nombre,color
1,zebra,blanco y negro
2,jirafa,amarillo
3,rinoceronte,gris
```

Ejercicio

Tiempo ⏱️

Escribir una función que a partir del nombre de un archivo CSV que tiene la siguiente estructura: [“id”, “producto”, “precio”, “descripción”] y un número n, escriba en otro archivo CSV los productos para los cuales el precio sea mayor a n. El CSV resultante tiene que tener la siguiente estructura: [“production”, “precio”, “descripción”]