Informática

2022/23

Grado en Ingeniería Aeroespacial, Primer Curso

Dr. Arno Formella

Departamento de Informática
Escola Superior de Enxeñaría Informática
Universidade de Vigo

22/23

profesorado (teoría)

Profesor:	Arno FORMELLA
Web:	`http://formella.webs.uvigo.es`
Correo:	formella@uvigo.es
Despacho:	ESEI, 309 (Edificio Politécnico)
Despacho remoto:	Sala profesorado 932
	(pedir cita y clave por correo electrónico)
Tutorías	Martes 11.00–14.00
	Jueves 11.00–14.00
usamos:	MOOVI (MOODLE DE LA UVIGO)
	`https://moovi.uvigo.gal`

profesorado (prácticas)

Profesora:	Alba NOGUEIRA RODRÍGUEZ
Web:	
Correo:	alnogueira@uvigo.es
Despacho:	
Despacho remoto:	Sala profesorado 1755 (pedir cita y clave por correo electrónico, o mirar moovi)
Tutorías:	

tutorías e idiomas

- Cambios puntuales de tutorías via aviso web.
  (yo en mi página principal)
  y/o mediante correo via plataforma teledocencia.
- Idiomas: Galego, Castellano, English, Deutsch.
- Las transparencias serán en castellano.
- Habrá textos e información en inglés.
  (sobre todo: todos los programas)

horario

	LUNES	MARTES	MIÉRCOLES	JUEVES	VIERNES
0900-0930	ALG (T) - Aula multiusos (inicio: 0845)		ALG (T) - Aula multiusos (inicio: 0845)	INF (T) - Aula multiusos	
0930-1000	ALG (T) - Aula multiusos	INF (T) - Aula multiusos	ALG (T) - Aula multiusos	INF (T) - Aula multiusos	
1000-1030	ALG (T) - Aula multiusos (finaliza: 1015)	INF (T) - Aula multiusos	ALG (T) - Aula multiusos (finaliza: 1015)	FIS1 (T) - Aula multiusos	FIS1 (T) - Aula multiusos
1030-1100	CAL1 (T) - Aula multiusos	CAL1 (T) - Aula multiusos	FIS1 (T) - Aula multiusos	FIS1 (T) - Aula multiusos	FIS1 (T) - Aula multiusos
1100-1130	CAL1 (T) - Aula multiusos	CAL1 (T) - Aula multiusos	FIS1 (T) - Aula multiusos	EG (T) - Aula multiusos	EG (T) - Aula multiusos
1130-1200	CAL1 (T) - Aula multiusos	CAL1 (T) - Aula multiusos		EG (T) - Aula multiusos	EG (T) - Aula multiusos
1200-1230	CAL1 (Q2/Q3) - Lab CA08	CAL1 (Q1) - Lab CA08	INF (S2) - Lab B1 EG (S1) - Lab B2	INF (S3) - Lab B1 EG (S2) - Lab B2 (inicio: 1215)	INF (S1) - Lab B1 EG (S3) - Lab B2 (inicio: 1215)
1230-1300	CAL1 (Q2/Q3) - Lab CA08	ALG (Q3) - Lab B1 CAL1 (Q1) - Lab CA08	INF (S2) - Lab B1 EG (S1) - Lab B2	INF (S3) - Lab B1 EG (S2) - Lab B2	INF (S1) - Lab B1 EG (S3) - Lab B2
1300-1330	CAL1 (Q2/Q3) - Lab CA08	ALG (Q3) - Lab B1 CAL1 (Q1) - Lab CA08	INF (S2) - Lab B1 EG (S1) - Lab B2	INF (S3) - Lab B1 EG (S2) - Lab B2	INF (S1) - Lab B1 EG (S3) - Lab B2
1330-1400	CAL1 (Q2/Q3) - Lab CA08	ALG (Q3) - Lab B1 CAL1 (Q1) - Lab CA08	INF (S2) - Lab B1 EG (S1) -Lab B2	INF (S3) - Lab B1 EG (S2) - Lab B2 (finaliza: 1415)	INF (S1) - Lab B1 EG (S3) - Lab B2 (finaliza: 1415)
1400-1430		ALG (Q3) - Lab B1			
1430-1500					
1600-1630	ALG (Q1/Q2) - Lab B1				
1630-1700	ALG (Q1/Q2) - Lab B1				
1700-1730	ALG (Q1/Q2) - Lab B1				
1730-1800	ALG (Q1/Q2) - Lab B1				

días de clases

lunes	martes	miércoles	jueves	viernes	sábado	domingo
Sep 5, 22	Sep 6, 22	Sep 7, 22	Sep 8, 22	Sep 9, 22	Sep 10, 22	Sep 11, 22
Sep 12, 22	Sep 13, 22	Sep 14, 22	Sep 15, 22	Sep 16, 22	Sep 17, 22	Sep 18, 22
Sep 19, 22	Sep 20, 22	Sep 21, 22	Sep 22, 22	Sep 23, 22	Sep 24, 22	Sep 25, 22
Sep 26, 22	Sep 27, 22	Sep 28, 22	Sep 29, 22	Sep 30, 22	Oct 1, 22	Oct 2, 22
Oct 3, 22	Oct 4, 22	Oct 5, 22	Oct 6, 22	Oct 7, 22	Oct 8, 22	Oct 9, 22
Oct 10, 22	Oct 11, 22	Oct 12, 22	Oct 13, 22	Oct 14, 22	Oct 15, 22	Oct 16, 22
Oct 17, 22	Oct 18, 22	Oct 19, 22	Oct 20, 22	Oct 21, 22	Oct 22, 22	Oct 23, 22
Oct 24, 22	Oct 25, 22	Oct 26, 22	Oct 27, 22	Oct 28, 22	Oct 29, 22	Oct 30, 22
Oct 31, 22	Nov 1, 22	Nov 2, 22	Nov 3, 22	Nov 4, 22	Nov 5, 22	Nov 6, 22
Nov 7, 22	Nov 8, 22	Nov 9, 22	Nov 10, 22	Nov 11, 22	Nov 12, 22	Nov 13, 22
Nov 14, 22	Nov 15, 22	Nov 16, 22	Nov 17, 22	Nov 18, 22	Nov 19, 22	Nov 20, 22
Nov 21, 22	Nov 22, 22	Nov 23, 22	Nov 24, 22	Nov 25, 22	Nov 26, 22	Nov 27, 22
Nov 28, 22	Nov 29, 22	Nov 30, 22	Dec 1, 22	Dec 2, 22	Dec 3, 22	Dec 4, 22
Dec 5, 22	Dec 6, 22	Dec 7, 22	Dec 8, 22	Dec 9, 22	Dec 10, 22	Dec 11, 22
Dec 12, 22	Dec 13, 22	Dec 14, 22	Dec 15, 22	Dec 16, 22	Dec 17, 22	Dec 18, 22
Examen	Jan 17, 23					

prerrequisitos

- ningunos, menos ganas de aprender

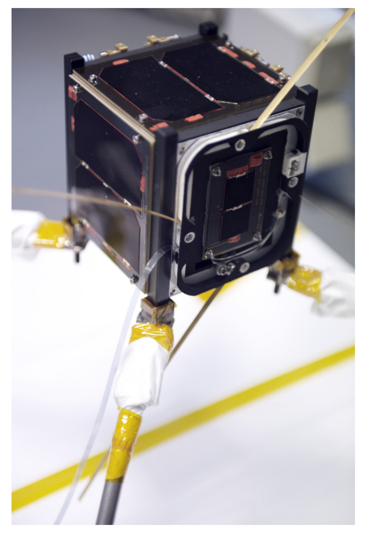
pruebas de evaluación continua previstas

- entregas semanales en prácticas
  (no se evaluan, pero documentan asistencia)
- seguramente 3 evaluables en prácticas
  (semanas marcadas en amarilla)
- quizá 1 o 2 evaluables en teoría
- examen final (parte práctica y teórica)
  17 de enero de 2023 (y 13 de julio de 2023)

¿Quién soy yo?

- http://formella.webs.uvigo.es
- http://lia.esei.uvigo.es

al espacio...



4 Satélites:

XaTcobeo, 14/02/2012

HumSAT, 21/11/2013

Serpens, 20/08–17/09/2015

Lume I, 28/12/2018 (Santi)

HumSAT software evolution (lines of code en C)

Lines

110000
105000
100000
95000
90000
85000
80000
75000

12-01 12-04 12-07 12-10 13-01 13-04 13-07 13-10 14-01 14-04 14-07 14-10 15-01

contenido (optimista) según guía docente

Tema	Subtema
Introducción a la informática	Hardware: componentes básicos Conceptos básicos de software Sistemas operativos Herramientas colaborativas Seguridad informática Redes de computadoras / big data
Conceptos de programación básicos	Tipos de lenguajes de programación: bajo y alto nivel Variables Funciones Control de flujo Entrada/salida
Conceptos de programación avanzados	Tipos de datos avanzados Excepciones Programación orientada a objetos
Programación orientada a la resolución de modelos numéricos usados en la ingeniería	Librerías matemáticas Cálculo paralelo Representación gráfica

documento de transparencias

- Este documento crecerá durante el curso,
  ojo, no necesariamente solamente al final.
- Habrá más documentos (capítulos de libros, manuales, etc.) con que trabajar durante el curso (algunos/muchos en inglés).
- Los ejemplos de programas y algoritmos (en teoría) serán en inglés.
- Las transparencias no están (posiblemente/probablemente) ni correctos ni completos.
- Las transparencias no son suficientes (incluso *chapadas*) para aprobar la asignatura.

competencias

Competencias	Tipo	Cod.
Que los estudiantes hayan demostrado poseer y comprender conocimientos en un área de estudio que parte de la base de la educación secundaria general, y se suele encontrar a un nivel que, si bien se apoya en libros de texto avanzados, incluye también algunos aspectos que implican conocimientos procedentes de la vanguardia de su campo de estudio	saber hacer	CB1
Conocimientos básicos sobre el uso y programación de los ordenadores, sistemas operativos, bases de datos y programas informáticos con aplicación en ingeniería.	saber hacer	CE3

competencias

Competencias	Tipo	Cod.
Capacidad de análisis, organización y planificación	ser	CT1
Liderazgo, iniciativa y espíritu emprendedor	ser	CT2
Capacidad de comunicación oral y escrita en la lengua nativa	ser	CT3
Capacidad de aprendizaje autónomo y gestión de la información	ser	CT4
Capacidad de resolución de problemas y toma de decisiones	ser	CT5
Capacidad de comunicación interpersonal	ser	CT6
Capacidad de razonamiento crítico y autocrítico	ser	CT8
Capacidad de trabajo en equipo de carácter interdisciplinar	ser	CT9

¿qué es eso?

¿qué es eso?

¿qué es eso?

EXPLANATION OF RECORDING COVER DIAGRAM

BINARY CODE DEFINING PROPER SPEED (3.6 seconds/ROTATION)
TO TURN THE RECORD (| = BINARY 1, — = BINARY 0)
EXPRESSED IN 0.70 × 10^{-9} seconds, THE TIME PERIOD ASSOCIATED
WITH THE FUNDAMENTAL TRANSITION
OF THE HYDROGEN ATOM

OUTLINE OF CARTRIDGE WITH STYLUS
TO PLAY RECORD (FURNISHED ON
SPACECRAFT)

PICTORIAL PLAN VIEW OF RECORD

ELEVATION VIEW OF CARTRIDGE

ELEVATION VIEW OF RECORD

PLAYING TIME, ONE SIDE = ~1 hour

THIS DIAGRAM DEFINES THE LOCATION OF OUR SUN UTILIZING 14
PULSARS OF KNOWN DIRECTIONS FROM OUR SUN. THE BINARY
CODE DEFINES THE FREQUENCY OF THE PULSES.

THE DIAGRAMS BELOW
DEFINE THE VIDEO PORTION OF THE RECORDING

GENERAL APPEARANCE OF WAVE FORM OF
VIDEO SIGNALS FOUND ON THE RECORDING

BINARY CODE TELLS TIME OF THE SCAN (~8 msec)

SCAN TRIGGERING

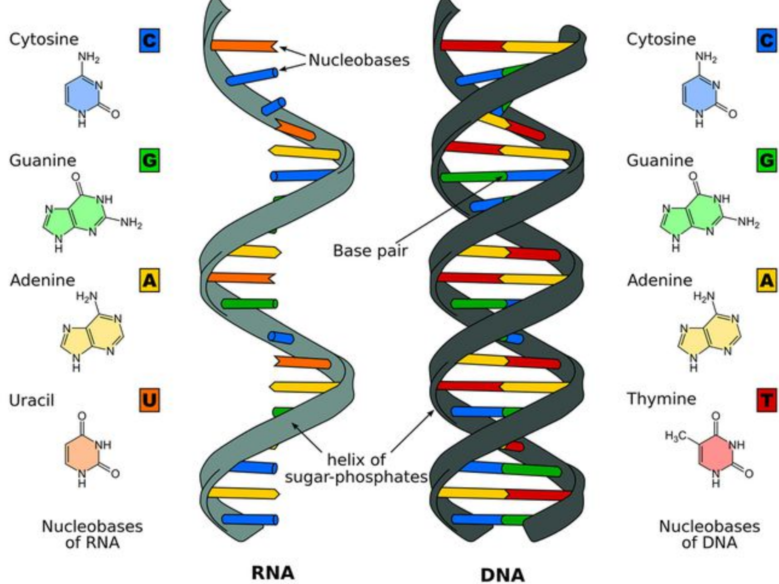
VIDEO IMAGE FRAME SHOWING DIRECTION OF SCAN
BINARY CODE INDICATES TIME OF EACH SCAN SWEEP
(512 VERTICAL LINES PER COMPLETE PICTURE)

IF PROPERLY DECODED, THE FIRST IMAGE
WHICH WILL APPEAR IS A CIRCLE

THIS DIAGRAM ILLUSTRATES THE TWO LOWEST STATES OF THE HYDROGEN ATOM.
THE VERTICAL LINES WITH THE DOTS INDICATE THE SPIN MOMENTS OF THE
PROTON AND ELECTRON. THE TRANSITION TIME FROM ONE STATE TO THE
OTHER PROVIDES THE FUNDAMENTAL CLOCK REFERENCE USED IN ALL THE
COVER DIAGRAMS AND DECODED PICTURES.

a ver si *alguien* algún día entiende...

¿qué es eso?



Nota: imágenes de voyager se ha obtenido de https://en.wikipedia.org/wiki/Voyager_Golden_Record, Public Domain, la imagen de la DNA/RNA de https://commons.wikimedia.org/wiki/File:Difference_DNA_RNA-EN.svg Creative Common license.

entonces: nociones básicas

- datos
- información
- conocimiento
- sabiduría

- informática

informática (currícula según ACM)

ACM *association for computing machinery* (2005)

- ciencias de la computación
- arquitectura de ordenadores
- ingeniería de software
- sistemas de información
- tecnologías de la información

informática (currícula según ACM)

ACM *association for computing machinery* (2013)
(https://dl.acm.org/doi/book/10.1145/2534860)

AL	Algorithms and Complexity	AR	Architecture and Organization
CN	Computational Science	DS	Discrete Structures
GV	Graphics and Visualization	HCI	Human-Computer Interaction
IAS	Information Assurance and Security	IM	Information Management
IS	Intelligent Systems	NC	Networking and Communications
OS	Operating Systems	PBD	Platform-based Development
PD	Parallel and Distributed Computing	PL	Programming Languages
SDF	Software Development Fundamentals	SE	Software Engineering
SF	Systems Fundamentals	SP	Social Issues and Professional Practice

y cada uno de estos apartados tiene una lista de recomiendaciones asociada.

informática ...

- algoritmos eficientes, tecnología de procesadores, lenguajes de programación, compiladores, teoría de computabilidad, bases de datos, ordenadores paralelos, sistemas software, interface de usuarios, etc.
- disciplinas con mucha mezcla: redes digitales, inteligencia artificial, robótica, minería de datos, sistemas de control, sistemas de ayuda de decisión, sistemas de información etc.
- Se aplica la informática en casi todo hoy en día.

datos

- representación simbólica de *algo*
- cuantitativo (ordenable), cualitativo (descriptivo)
- codificable
- almacenable
- transmisible
- procesable
- interpretable

datos

- Con la interpretación de los datos se convierten en información,
- la interrelación de información en un contexto proporciona conocimiento,
- la adecuada aplicación del conocimiento la convierte en sabiduría.

codificación

- normalmente se usa codificación binaria: zero's y uno's
- bit
- 1 byte = 8 bits (unidad básica a nivel hardware)
- símbolos con codificación de tamaño fijo
- símbolos con codificación de tamaño variable

interpretación	dato	valor	
números	01011001	89	8 bit unsigned int
letras	01011001	Y	ASCII char
colores	01011001	este color	3-3-2 rgb color code

la codificación de los datos aquí son números en base a 2

unidades de almacenamiento de datos (prefijos ISQ)

prefijo	símbolo	factor	
yotta	Y	1000^8	10^{24}
zetta	Z	1000^7	10^{21}
exa	E	1000^6	10^{18}
peta	P	1000^5	10^{15}
tera	T	1000^4	10^{12}
giga	G	1000^3	10^9
mega	M	1000^2	10^6
kilo	k	1000^1	10^3

con eso tenemos por ejemplo GHz (gigahercios) o TB (terabytes)

unidades de almacenamiento de datos (prefijos IEC)

prefijo	símbolo	factor	
yobi	Yi	1024^8	2^{80}
zebi	Zi	1024^7	2^{70}
exbi	Ei	1024^6	2^{60}
pebi	Pi	1024^5	2^{50}
tebi	Ti	1024^4	2^{40}
gibi	Gi	1024^3	2^{30}
mebi	Mi	1024^2	2^{20}
kibi	Ki	1024^1	2^{10}

lectura de una vida

- 80 caracteres por línea (son 80 B)
- por 60 líneas por página (son 5 kB, 0.5 hoja)
- por 2 páginas por hoja (son 10 kB, 1 hoja)
- por 50 hojas por día (son 500 kB, 50 hojas)
- por 400 días al año (son 200 MB, 20 000 hojas)
- por 50 años de la vida (son 10 GB, 1 000 000 hojas)
- 100 m (1000 folios son unos 10 cm) de libros en estantería es el tope!

lectura en voz alta

- unos 200 palabras por minuto (unos 1000 letras) (es un 1 kB)
- por 60 minutos la hora (son 60 kB)
- por 10 horas el día (son 600 kB)
- por 400 días al año (son 240 MB)
- por 50 años de la vida (son 12 GB)

curiosamente +o- lo mismo de antes...

- unos 100 palabras por minuto (equivalente a unos 500 letras) (son unos 500 B)
- ergo unos 6 GB en 50 años de vida.
- (record de mundo 360 palabras por minuto con stenotype)

curiosamente +o- lo mismo (bueno la mitad) de antes...

- para audiolibros se recomienda no más de unos 200 palabras por minuto (unos 1000 letras) (es un 1 kB)
- y sigue entonces: unos 12 GB para toda una vida!

curiosamente +o- lo mismo de antes...

podemos comprimir los datos...

- texto se comprime a unos 20 % de su tamaño original
- pues una memoria de USB de 8GB llega para toda la vida
  puede almacenar todo lo que se puede escuchar, leer y escribir
- a ver como te apañas hacerlo en parelelo...
- y se transmite todo en una red moderna (1 Gb/s) en 1 min.

eso se llama análisis de requisitos: cuanta memoria necesito para cierta tarea. otros ejemplo:

- ¿cuánta memoria hay que poner en una caja negra para grabar todos los datos (seleccionados) de cierto tiempo de vuelo?
- ¿cuánta memoria hay que poner en una nave espacial para almacenar los datos captados por sus sensores (que todavía no se pude mandar a tierra)?

software

- conjunto de programas, es decir, secuencia de instrucciones, que se pueden ejecutar en un ordenador
- sistemas operativos
- aplicaciones para *usuarios* intermedios y finales
- herramientas de desarrollo
- firmware (drivers p.ej. tarjetas gráficas o de redes, bios)

Existe software que a su vez genera software.

Miramos un poco lo que hay en mi ordenador (o el vuestro).

codificación y desarrollo de software

- binario: se escribe directamente instrucciones en bits y bytes
- ensemblador: se escribe instrucciones de un procesador específico legibles por un ser humano
- lenguaje de programación: se escriben algoritmos en un lenguaje formal (legible por un ser humano) tratable por otro programa que lo traduce
- lenguaje de modelado: se escribe con lenguajes de modelado (incluso gráficos) que a su vez se traducen a lenguajes formales

Es decir, existe una pila de herramientas para conseguir la elaboración de software que finalmente se ejecuta en un ordenador.

lenguajes de programación

- instrumento de alto nivel para programar un ordenador
- Python, Java, C/C++, Pascal, Fortran, Cobol, Lisp, Ada, C#, Prolog...
- usamos Python (creado 1991)
- bifurcación no compatible después de versión 2.7
- lenguaje interpretado (ejecución *línea-por-línea*)
- en comparación: lenguaje compilado
  (traducción de todo el programa a código de procesador)
- versiones actuales 2.7.18 (abril 20) y 3.11 (septiembre 22)
- (la versión 2.7. ya no se sigue desarrollando... *sunset reached*)

Procedural: divide programa en partes más pequeñas o subprogramas (subrutinas, funciones, procedimientos) que se invocan entre sí y manipulan directamenta los datos. Ejemplos: C, Pascal, modula...

Orientado a objetos: evolución de los procedurales, utilizan como abstracción el concepto de clase. Un programa está formado por un conjunto de objetos, instancias de una clase, que llevan a cabo acciones y se comunican con otros objetos utilizando el concepto de mensajes. Ejemplos: C++, Java, Python, ...

Funcional: programación en base a definición de funciones (matemáticas), que se invocan (muchas veces) de manera recursiva.
Ejemplos: Lisp, Haskell, Camel, Erlang, ...

Lógica/declarativa: programación basada en representación del conocimiento y lógica de predicados, no expresan el flujo de control sobre los datos directamente.
Ejemplo: Prolog, SQL,...

Veremos solamente el paradigma procedural quizá con pinceladas de objetos. Lenguajes modernas cubren varios paradigmas (C++, Python).

Diseño de software...

Análisis se analiza todos los requisitos del software por desarrollar, p.ej. requisitos: generales, funcionales, del rendimiento, de las interfaces, operacionales, de los recursos, del diseño, de la seguridad/privacidad, etc.

Diseño se diseño (con un formalismo acordado) todos los componentes e interfaces del producto

Implementación se implementa con el (o los) lenguaje(s) de programación el software

Existen métodos más o menos formales para realizar eso, pueden ser incrementales (clásicos) o iterativos (modernos, ágiles).

Verificación/Validación se verifica que todo el software es correcto, es decir, no contiene fallos; *lo que hace el software, lo hace bien* y se valida que todo el software cumple con todos los requisitos como producto (ya siendo correcto), *el software hace lo que tiene que hacer*

Uso/despliegue se distribuye el software al sitio donde se usa

Mantenimiento se mantiene el software tal que siga funcionando en su entorno (quizá hace falta arreglar fallos que no se han detectado antes, adaptar el software a nuevos entornos o sus versiones, adaptar sus interfaces a nuevas circumstancias, etc.)

Documentación se describe de forma adecuada todo, es decir, desde el análisis hasta el mantenimiento, unas partes destacadas son el manual de instalación y el manual de usuario (o manual de operaciones).

Python

- miramos un poco `www.python.org`
- miramos un poco python en la consola
- miramos un poco python en IDLE

Python y su primer uso

- hemos visto el uso directo del interpreter de python
- hemos visto el uso del entorno de IDLE
- hemos visto cómo generar un simple fichero y ejecutar su contenido con el interpreter
- más en las prácticas... y más en estas clases...
- y/o intentad repetir lo que has visto en clase!
- haced propios experimentos, trabajad en grupos!

variables

Una variable es el concepto con el cual los lenguajes de programación suelen trabajar con sus datos. Las características principales de las variables son:

- nombre
- tipo
- valor
- representación
- ámbito

Dependiendo del tipo de las variables, los lenguajes de programación permiten realizar operaciones o cálculos sobre dichas variables.

nombre de una variable

- se refiere a un identificador que inequivocadamente identifica la variable
  (normalmente legible/interpretable por un humano)
- Python: cadena de letras de longitud arbitraria (minúsculas y mayúsculas), dígitos, y el `'_'` que no comienzan con un dígito
- ejemplos: `myVar`, `my_var`, `_myvar`, `Var1`, `Var_2`
- en otros lenguajes pueden estar permitidos otros símbolos, p.ej., `'%'` o `'$'` o Unicode caracteres
- están excluidos como nombres de variables las palabras propias del lenguaje (*key words*), p.ej., `while`, `for`, `in`, `not`, etc.
- el nombre de una variable se traduce en el proceso de transformación del programa a código del ordenador en una dirección de memoria (bueno más o menos...)

¿qué nombre elegir?

- no existe una norma o acuerdo en general
- la semántica del nombre debe ajustarse a la situación del uso de la variable (matemática, física, finanzas, gestión etc.)
- se suele usar nombre cortes si la variable *vive* corto tiempo.
- se suele usar nombres largos si el ámbito es más grande
- hay convenciones para el uso de prefijos y sufijos (yo por ejemplo escribo como sufijo la unidad de una entidad física si no es SI (p.ej., `distance_km`, `time_hour`)
- hay nombres comunes, p.ej. `i` para una variable entero que controla un bucle, `x` e `y` para coordinadas etc.

en un grupo de desarrollo se debe cumplir con una única disciplina

tipo de una variable

- se refiere a una semántica que caracteriza la variable
- puede estar identificado a su vez por un nombre, p.ej. `int`, `float`, `string`, etc.
- la asignación del tipo puede ser estática o dinámica (Python usa tipado dinámico)
- se suele tener formas de convertir ciertos tipos en otros tipos, p.ej. `i=int(s)` en Python
  (bueno, de hecho crea una nueva variable...)
- el lenguaje puede ser más o menos restrictivo en la conversión implícita de tipos
- el tipo de una variable determina qué operaciones y funciones se pueden realizar con la variable
- tipos comunes en muchos lenguajes son:
  `int`, `float`, `double`, `boolean`
- existen tipos de datos, incluso formas de construirlos dinámicamente, más complejos (veremos algo más adelante)

valor de una variable

- el valor de una variable es la interpretación de su representación en bits/bytes
- valores típicos son números enteros, números de punto flotante, o cadenas de símbolos, o valores booleanos
- valores se suelen escribir también como constantes en el lenguaje de programación, p.ej. `25`, `3.1415`, `"hola"`, `True`
- el valor de una variable puede ser mutable o no, p.ej. en Python cadenas son inmutables

representación de una variable

- es la secuencia de bits que finalmente representan el valor de una variable
- dependen del sistema donde se ejecuta el programa
(o en el cual se realiza las transformaciones a código nativo)
- ejemplos:
  - números enteros de tamaño fijo,
  - números enteros de tamaño variable,
  - números de punto flotante de tamaño fijo (p.ej. 64 bit),
  - etc.
- dos sistemas pueden ser o no ser compatibles en sus representaciones que es un hecho de tener en cuenta en interfaces entre sistemas
(por ejemplo: sistemas *little endian* o *big endian*)

ámbito de una variable

- una variable tiene un ámbito (*scope*) en el cual o durante existe
- es decir, se crea la variable (con su memoria correspondiente), y se puede operar con ella
- dependiendo del lenguaje se crea con su primer uso (Python) o con una declaración específica
- existe la posibilidad que nombres están escondidos detrás otros si se declara los mismos nombres en construcciones anidadas (*shadowing*)

un programa en el lenguaje de programación C

```
#include <stdio.h>

int main(
) {
  int age = 25; // type is integer,
                // name is age,
                // and value is 25
  printf("My age is %d.\n", age);
}
```

proceso de generación del programa

- dado que C es un lenguaje de programación que se compila,
- es decir, otro programa llamado *compilador* traduce el texto escrito por el ser humano a un programa ejecutable por el ordenador,
- tenemos que invocar el compilador, una línea de comando sería:
  `gcc age.c -o age`
- que nos genera el ejecutable `age`.
- observa que ciertos fallos se detecta durante la fase de compilación

ejemplos de uso de variables en C

el programa en lenguaje del procesador (ensamblador):

```
        .file        "age.c"
        .section        .rodata
.LC0:
        .string        "My age is %d.\n"
        .text
        .globl        main
        .type        main, @function
main:
.LFB0:
        .cfi_startproc
        pushq        %rbp
        .cfi_def_cfa_offset 16
        .cfi_offset 6, -16
        movq        %rsp, %rbp
        .cfi_def_cfa_register 6
        subq        $16, %rsp
        movl        $19, -4(%rbp)
        movl        -4(%rbp), %eax
        movl        %eax, %esi
        movl        $.LC0, %edi
        movl        $0, %eax
        call        printf
        movl        $0, %eax
        leave
        .cfi_def_cfa 7, 8
        ret
        .cfi_endproc
.LFE0:
        .size        main, .-main
        .ident        "GCC: (Ubuntu 5.4.0-6ubuntu1~16.04.12) 5.4.0 20160609"
```

¿cómo se obtiene tal extracto?

- se puede ordenar al compilador que genere también el programa en ensamblador,
- es decir, visualizando las instrucciones que ejecutaría la CPU,
- tenemos que invocar el compilador pidiendo que genere tal fichero intermedio, una línea de comando sería:

  `gcc -S age.c`
- que nos genera tal fichero ensamblador `age.s`

ejemplos de uso de variables en C++

un programa en el lenguaje de programación C++

```
#include <iostream>

int main(
) {
  int age{25}; // type is integer,
               // name is age,
               // and value is 25
  std::cout << "My age is " << age << ".\n";
}
```

ejemplos de uso de variables en Java

un programa en el lenguaje de programación Java

```
public class Age {
  public static void main(String[] args) {
    int age=25;
    System.out.println("My age is "+age+".");
  }
}
```

compilación para C++ y Java

- el proceso para C++ sería:
  `g++ age.cc -o agecc`
  que genera un ejecutable `agecc`.
- el proceso para Java sería:
  `javac Age.java`
  que a su vez genera un fichero con un contenido interpretable
- por eso se tiene que ejecutar con un programa de interpretación:
  `java Age`
  parecido como lo hacemos con programas en python.

resumen

- hemos visto en clase en directo
- como se compila un fichero en código fuente
- mediante un compilador
- para obtener un programa ejecutable (en caso de C y C++) o interpretable (en caso de Java)

comentarios en Python

- Las líneas que comienzan por almohadilla (#) son ignoradas.
- Si una línea contiene una almohadilla, se ignora hasta el final de línea (si la almohadilla no forma parte de una cadena).
- Los grupos de líneas delimitados por triples comillas simples (''') o dobles (""") que no son cadenas son ignorados.

- Se debe documentar sobre todo lo que no es obvio, las interfaces (en el sentido amplio de la palabra), y los casos límite.
- Es decir: Los comentarios son las respuestas a preguntas del *¿Cómo?* y del *¿Por qué?*
- Se debe usar, por ejemplo, doxygen (o javadoc, u otra herramienta buena) para generar automáticamente la documentación del código.

ejemplos de comentarios

```
# This is a comment in Python

print "Hello World" # This is also a comment in Python

a = "Here a '#' that is not a comment"

""" This is an example of a multiline
comment that spans multiple lines
"""

'''This is an example of a multiline
comment that spans multiple lines
'''

s = '''Here a multiline
string that spans multiple lines'''
```

operadores

- son símbolos, normalmente comunes del ámbito matemático, que definen operaciones entre variables (y constantes)
- ojo, a veces la semántica no es exactamente la misma que se suele usar en el ámbito matemático
- Python usa operadores como infijos, p.ej. `a+12`
- Python: típicos operadores son `+`, `-`, `*`, `/`, `**`, `//`, `%`
- también existen operadores para otros tipos, p.ej. booleanos `not`, `and`, `or`, etc.
- o operadores para comparaciones, p.ej. `==`, `>=`, `<`, etc.

precedencia de operadores

- de menor a mayor:

`or`	Boolean OR
`and`	Boolean AND
`not x`	Boolean NOT
`in`, `not in`, `is`, `is not`, `<`, `<=`, `>`, `>=`, `!=`, `==`	Comparisons, including membership tests and identity tests
`|`	Bitwise OR
`^`	Bitwise XOR
`&`	Bitwise AND
`<<`, `>>`	Shifts
`+`, `-`	Addition and subtraction
`*`, `@`, `/`, `//`, `%`	Multiplication, matrix multiplication, division, floor division, remainder [5]
`+x`, `-x`, `~x`	Positive, negative, bitwise NOT
`**`	Exponentiation [6]

- en el mismo nivel se evalua de izquierda hacia la derecha (con la excepción de `**`, donde es al revés)
- en caso de dudas: usa paréntesis

cadenas en Python...

- son secuencias o vectores de símbolos
- sus constantes se delimitan con simples, o dobles comillas, o con triples simples o triples dobles comillas (multi-línea)
- el acceso a símbolos individuales se realiza con corchetes e índice, p. ej., `s[2]` es la tercera letra en la cadena `s`
- los símbolos están representados en una codificación (UTF-8 por defecto en código fuente)
- se tiene acceso también a la representación en bytes (anteponer una `b` a la cadena, ojo...)

- existen posibilidades de *usar* símbolos en cadenas que no están en el teclado con secuencias especiales, por ejemplo,
  `u'\u03B1'` produce el símbolo griego α
  (la `u` significa que se usa codificación de Unicode)
- no nos vamos a liar más en estos momentos con las cadenas... mirad detenidamente el manual para cada caso/requisito en concreto.
  (https://docs.python.org/3.11/howto/unicode.html)

representación de números en diferentes sistemas

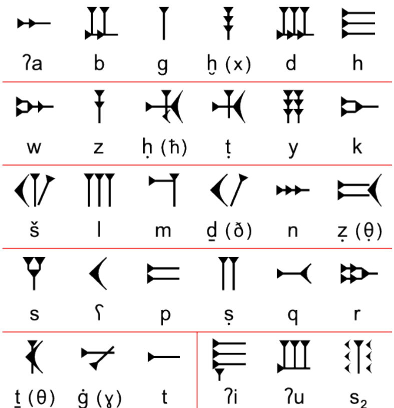
palos	|	||	|||	||||	|||||	||||||	|||||||	||||||||	...
romano	I	II	III	IV	V	VI	VII	VIII	...
árabe	١	٢	٣	٤	٥	٦	٧	٨	...
maya	·	:	⋮	⁞	ı	·ı	:ı	⋮ı	...
decimal	1	2	3	4	5	6	7	8	...
binario	1	10	11	100	101	110	111	1000	...

- el invento del zero fue un gran logro
- basándose en los dígitos (p.ej., de 0 a 9) se escribe un valor teniendo en cuenta la posición del dígito
  (de la derecha a la izquierda, como la escritura árabe)
- asumiendo una base b (p.ej., 2 o 10 o 16) se calcula el valor v de una secuencia de dígitos $d_n d_{n-1} \ldots d_2 d_1 d_0$ (con $d_i \in [0, b)$):

$$v = \sum_{i=0}^{n} d_i \cdot b^i$$

alfabeto o abecedario

- probablemente el primer alfabeto (dos versiones con su orden en tabla) en la ciudad Ugarit (hoy Siria), 1500 AC
- símbolos para fonémas ya se inventarón antes (posiblemente en egipto)



codificación de un alfabeto o abecedario

en el ordenador recomiendo usar solamente los siguientes *alfabetos*, más bien codificaciones de letras/dígitos/símboles/caracteres:

- ASCII, 7 bits en un byte, 127 caracteres posibles (digamos: la intersección de todos, funciona basicamente siempre)
- UTF-8, 1 hasta 4 bytes, 1.112.064 caracteres posibles (digamos: muy usado, recomendado, funciona basicamente simpre, pero cierto cuidado si se cambia idioma).
- Unicode, 4 bytes, en principio 4.294.967.296 caracteres posibles (digamos: funciona siempre, tiene mapeo a todos los antiguos, pero ficheros grandes)

la tabla ASCII (7 bit) original

USASCII code chart

b_7 b_6 b_5 →					0 0 0	0 0 1	0 1 0	0 1 1	1 0 0	1 0 1	1 1 0	1 1 1
Bits b_4 ↓	b_3 ↓	b_2 ↓	b_1 ↓	Column → / Row ↓	0	1	2	3	4	5	6	7
0	0	0	0	0	NUL	DLE	SP	0	@	P	`	p
0	0	0	1	1	SOH	DC1	!	1	A	Q	a	q
0	0	1	0	2	STX	DC2	"	2	B	R	b	r
0	0	1	1	3	ETX	DC3	#	3	C	S	c	s
0	1	0	0	4	EOT	DC4	$	4	D	T	d	t
0	1	0	1	5	ENQ	NAK	%	5	E	U	e	u
0	1	1	0	6	ACK	SYN	&	6	F	V	f	v
0	1	1	1	7	BEL	ETB	'	7	G	W	g	w
1	0	0	0	8	BS	CAN	(	8	H	X	h	x
1	0	0	1	9	HT	EM	)	9	I	Y	i	y
1	0	1	0	10	LF	SUB	*	:	J	Z	j	z
1	0	1	1	11	VT	ESC	+	;	K	[	k	{
1	1	0	0	12	FF	FS	,	<	L	\	l	|
1	1	0	1	13	CR	GS	-	=	M	]	m	}
1	1	1	0	14	SO	RS	.	>	N	^	n	~
1	1	1	1	15	SI	US	/	?	O	_	o	DEL

Masaram Gondi (2017), Yezidi (2020) y Tangsa (2021)

	11D0	11D1	11D2	11D3	11D4	11D5
0	11D00	11D10	11D20	11D30	11D40	11D50
1	11D01	11D11	11D21	11D31	11D41	11D51
2	11D02	11D12	11D22	11D32	11D42	11D52
3	11D03	11D13	11D23	11D33	11D43	11D53
4	11D04	11D14	11D24	11D34	11D44	11D54

	10E8	10E9	10EA	10EB
0	10E80	10E90	10EA0	10EB0
1	10E81	10E91	10EA1	10EB1
2	10E82	10E92	10EA2	
3	10E83	10E93	10EA3	
4	10E84	10E94	10EA4	

	16A7	16A8	16A9	16AA	16AB	16AC
0	16A70	16A80	16A90	16AA0	16AB0	16AC0
1	16A71	16A81	16A91	16AA1	16AB1	16AC1
2	16A72	16A82	16A92	16AA2	16AB2	16AC2
3	16A73	16A83	16A93	16AA3	16AB3	16AC3
4	16A74	16A84	16A94	16AA4	16AB4	16AC4

emojis disponibles en Unicode en junio 2018

¿retrocemos a los tiempos de antes del invento del alfabeto...?!?¡

en 2022: `https://unicode.org/emoji/charts/full-emoji-list.html`

estructuras de control en Python

- para manipular el flujo de control de un programa
- se usa estructuras de control como, entre otras:
  - condiciones: `if` con sus `elif` y `else` etc.
  - bucles: `while` y `for`
  - saltos a otros punto: `break`, `continue`, etc.
  - subrutinas: `def` de funciones
- otros lenguajes pueden tener construcciones adicionales/diferentes
- existen otros nombres para el concepto subrutina: procedimientos (*procedures*), métodos (*methods*), funciones (*functions*)

programa con bucle while

```python
number = 23
running = True
while running:
    guess = int(input('Enter an integer : '))
    if guess == number:
        print('Congratulations, you guessed it.')
        # this causes the while loop to stop
        # running = False
        break
    elif guess < number:
        print('No, it is a little higher than that.')
    else:
        print('No, it is a little lower than that.')
else:
    print('The while loop is over.')
    # Do anything else you want to do here
print('Done')
```

bucle infinito con salida `break`

```
while True:
    s = input('Enter something : ')
    if s == 'quit':
        break
    print('Length of the string is', len(s))
    print('Done')
else:
    # the next line is never executed
    print("ha...!!")
```

hemos visto con estos ejemplos (y más en clase)

- hay diferente formas de terminar el bucle
- el caso `else:` del bucle ayuda en determinar de que manera ha terminado el bucle
- haz tus propios experimentos
- la condición como y cuando terminar el bucle requiere cierto piensamiento... ¿no es mejor (o más natural) terminar con la condición `s==''` en lugar de la condición `s=='quit'`?

bucle `for` con salida `break`

```
for i in range(10):
    print("Value: ",i)
    if(i==4):    # comment or not comment...
      break
else:
    # This line is never executed!
    print("Terminated normally!")
```

bucle `for` con comienzo, fin, y paso

```
for i in range(3,10,2):
    print("Value: ",i)
else:
    print("Terminated normally!")

# what happens if step is 0?
```

hemos visto (con más ejemplos en clase)

- importante es tener claro
  - con qué condición (o estado) se comienza un bucle,
  - cómo se avanza dentro del bucle, y
  - con qué condición (o estado) se termina el bucle
- hemos visto la semántica de `range(begin,end,step)`
- sobre todo
  - que el punto `end` no pertenece al rango,
  - que no funciona con números flotantes,
  - que funciona con pasos negativos,
  - que puede ser un rango vacio

bucle `while` como sustituto de bucle `for`

```
begin=3
end=10
step=0
i=begin
while i<end:
    print("Value: ",i)
    i=i+step
else:
    print("Terminated normally!")
```

hemos visto (con más ejemplos en clase)

- se puede sustituir un bucle `for` por un bucle `while`
- los problemas en el diseño de bucles suelen ser tres:
  - el correcto comienzo,
  - la correcta terminación, y
  - el correcto paso
- dichas situaciones deben ser comprobadas para cualquier caso que se puede dar en el programa (pruebas y depuración)
- tal como está el ejemplo, se trata de un bucle infinito

subrutinas en Python

- se definen antes de su primer uso (con `def`)
- pueden acceptar parámetros (o argumentos)
- pueden devolver un resultado

```
import math

def f(x,a,b,c):
    return a*x*x+b*x+c

print(f(3,1,2,3))
fx=f(0,1,2,3)
print(fx)
print(f(3,math.sqrt(2),2,3))
```

otro ejemplo de funciones

```
import math

def pol2rec(r,phi):
  return r*math.sin(phi),r*math.cos(phi)

def rec2pol(x,y):
  return math.sqrt(x*x+y*y),math.atan2(y,x)

x=1.0
y=1.0
print(rec2pol(x,y))
r,p=rec2pol(x,y)
print(pol2rec(r,p))
if (x,y)!=pol2rec(r,p):
    print("not reciprocal?")
```

hemos visto (con más ejemplos en clase)

- el programa de antes tiene un fallo
  (¿te acuerdas cuál era el problema?)
- hemos visto que hay funciones matemáticas, como el arcotangente, que tienen su aquel y agradecemos que ya están bien tratados los casos en el paquete `math`
- (he comentado que el caso `atan2(0,0)` está diferente en diferentes lenguajes de programación, ya que matemáticamente no está definido, pero muchos devuelven 0 o π dependiendo del signo del segundo parámetro)
- hemos visto que la reciprocidad de operaciones que conocemos de cálculo no se refleja bien en la programación por cuestiones de representación de números en bits (y el rondeo que conlleva)

argumentos: por defecto

- A los últimos parámetros se pueden asignar un valor por defecto.
- En general se pueden pasar los argumentos a una función especificando un valor a su nombre (así no importa el orden).

```
def price(base,tax=21,discount=0):
  return base*(1-discount/100)*(1+tax/100.0)

# using default tax and no discount
print(price(100))
# using explicit tax and some discount
print(price(40,4,3))
# using default tax, but some discount
print(price(30,discount=5))
```

argumentos: cantidad variable

- Se pasa un número variable de argumentos anteponiendo un asterisco.
- Los argumentos están disponibles en una tupla.

```
def average(*values):
  sum=0
  for value in values:
    sum+=value
  return sum/len(values)

print(average(1,2))
print(average(1,2,3,4,5,6))
print(average(1))
print(average())
```

un polinomio con número de coeficientes variable

```
import math

def p(x,*coef):
    s=0.0
    i=0
    for c in coef:
      s=s+ c * x**i
      i=i+1
    return s

print(p(3,3,2,1))
print(p(3,3,2))
print(p(1,0,1,2,3,4,5,6,7,8,9,10))
```

retornar

- `return` acabe inmediatamente la ejecución de una función
- puede devolver valores al punto de la llamada
- todas las funciones tienen un `return` implícito al final (sin valores)
- es recomendable que una función siempre devuelve valores del mismo tipo con todos sus `return`

retornar en algún momento

```
def RemindMe(n):
    cnt=0
    while True:
        print("I will study python today")
        cnt += 1
        if (cnt==n):
            print("did it!")
            return

RemindMe(-10)
```

retornar tipos diferentes (no-recomendado)

```
def maximum(x,y):
    if x>y:
        return x
    elif x<y:
        return y
    else:
        return "the number are equal"
        #return x

print(maximum(1,23))
print(maximum(5,2))
print(maximum(7,7))
print(maximum(maximum(5,2),maximum(7,7)))
print(maximum( \
  maximum("hola","zacharias"), \
  maximum("pan","pan")) \
)
```

estructuras de datos

- son datos (del mismo o de diferentes tipos)
- en conjunto con funciones/métodos/operadores
- que permiten
  - el acceso a los datos
  - la modificación de los datos
  - la obtención de propiedades de los datos
- proporcionan un nivel más alto de abstracción
- suelen estar implementadas de forma eficiente (respecto a memoria y/o tiempo de ejecución)

ejemplos de estructuras de datos

- listas
- tuplas, vectores, matrices, etc.
- conjuntos
- diccionarios
- archivos
- árboles
- grafos

disponibilidad de estructuras de datos

- algunos lenguajes ya proporcionan ciertas estructuras de datos
- se pueden implementar otras estructuras de datos según necesidades
- su disponibilidad permite realizar aplicaciones de forma más fácil (modularidad, diseño con componentes)
- Python proporciona directo: tuplas, listas, conjuntos, diccionarios
- Python proporciona indirecto: mucho! según paquetes importados

- es una secuencia de elementos
- los elementos pueden ser de diferentes tipos
- los elementos y la lista son mutables
- se construye con corchetes: `L=[1,2,"hi",True]`
  (o por comprensión, o con `list(`*iterable*`)`)

listas: ejemplos de acceso, modificación y propiedades

- acceso:
  - mediante índices positivos desde el principio
  - mediante índices negativos desde el final
  - mediante *slices* a sublistas: `L[start:stop:step]`
- modificación:
  - `L[0]='some'` modificar un elemento
  - `L.append('more')` añadir un elemento
    (también `insert`, `remove`, `pop`, `index`, `count`, `sort`)
  - `L+=L` concatenar la lista (aquí sería duplicar)
- propiedades:
  - `len(L)` longitud de la lista
  - `L==[]` ¿es lista vacía?

y mucho más, ¡mira manual de Python! (*best friend forever :-)*)

estructuras de datos en Python: tuplas

- es una secuencia de elementos
- los elementos pueden ser de diferentes tipos
- los elementos y la tupla son inmutables
- se construye con paréntesis: `T=(1,2,"hi",True)`
- Una función con número de argumentos variable agrupa sus parámetros implicitamente en una tupla.

tuplas: ejemplos de acceso y propiedades

- acceso:
  - mediante índices positivos desde el principio
  - mediante índices negativos desde el final
  - mediante *slices* a subtuplas: `T[start:stop:step]`
- propiedades:
  - `len(T)` longitud de la tupla
  - `T==()` ¿es tupla vacía?

y mucho más, ¡mira manual de Python!

estructuras de datos en Python: conjuntos

- es una colección de elementos
- los elementos pueden ser de diferentes tipos
- el conjunto es mutable
- se construye con llaves: `S={1,2,"hi",True}`
  (o por comprensión, o con `set(`*iterable*`)`)
- permite operaciones de conjuntos tanto con funciones como con operadores
- existe una versión como `frozenset` que es inmutable y plana
  (no contiene a su vez estructuras de datos como elementos)

- acceso:
  - mediante operador `in`
- modificación:
  - `S.add(element)`, `S.remove(element)`
  - `S.intersection(other)` intersección de conjuntos (también `union`, `difference`, `symmetic_difference`)
  - `S|=other` unión del conjunto con otro (hay más operadores)
- propiedades:
  - `len(S)` longitud del conjunto
  - `S=={}` ¿es conjunto vacía?
  - `S<=other` ¿es subconjunto?

y mucho más, ¡mira manual de Python!

estructuras de datos en Python: diccionarios

- es una colección de elementos en parejas clave:valor
- los elementos pueden ser de diferentes tipos
- los elementos y el diccionario son mutables
- se construye con `dict(`*parejas*`)`
- o con claves y parejas
  `D={'yo':'ich','hola':'hallo'}`

diccionarios: ejemplos de acceso, modificación y propiedades

- acceso:
  - mediante las claves y corchetes `D[key]`
  - mediante `D.values()`, `D.keys()`, o `D.items()`
- modificación:
  - `D['yo']='some'` modificar o añadir un elemento
- propiedades:
  - `len(L)` longitud de la lista
  - `L=={}` ¿es diccionario vacía?

y mucho más, ¡mira manual de Python!

listas por comprensión

- se construye los elementos mediante un búcle `for`
  `[x*x for x in range(10)]`
- se puede añadir condiciones `[x*x for x in range(10) if x%2==0]`
- búcles pueden ser anidados `[x*y for x in [1,2,3] for y in [3,4,5] ]`
- se puede construir conjuntos y diccionarios por comprensión

y mucho más, ¡mira manual de Python y prácticas!

archivos

- Los archivos o ficheros se almacenan en el sistema de almacenamiento del ordenador (por ejemplo, el disco duro).
- Con los ficheros se puede realizar una serie de operaciones, por ejemplo: abrir, leer, escribir, extender, cerrar, borrar, renombrar etc.
- vemos unos simples ejemplos

archivos, uso simple

```
file = open("myfile.txt","w")
file.write("first line?")
file.write("second line?")
file.write("\nthird line?")
file.close()

file=open("myfile.txt","r")
print(file.read())
file.close()

file=open("myfile.txt","r")
for l in file.readlines():
  print(l)
file.close()
```

archivos `csv`

- Ficheros `csv` son archivos formateados de forma simple: las entradas están separadas por comas (o por otro símbolo si queremos)
- por eso: `csv` *comma separated values*
- Python proporciona un módulo para trabajar con ficheros en formato `csv`

archivos `csv`, uso simple

```
import csv

file=open("mycsv.csv","rt")
reader=csv.reader(file,delimiter=";")
names=set()
numbers=set()
num_rows=0
for row in reader:
  num_rows+=1
  names.add(row[0])
  numbers.add(int(row[1]))
file.close()

print(names)
print(numbers)
```

variables globales

- Python permite con `global` el uso de variables globales, es decir, una función puede asumir que tal variable existen en un ámbito global.
- Tal uso puede provocar efectos llamados segundarios ya que valores de variables pueden varias inesperadamente.
- Existe una variante `nonlocal` que permite usar variables de ámbitos superiores en funciones anidadas,
- El uso de variables globales se debe reducir a lo máximo y siempre documentar de forma muy clara.

variables globales: ejemplo simple

```
def printme():
        global a
        a='some'
        print(a)

a='hola'
printme()
print(a)
```

Argumentos de programas

- Podemos pasar argumentos a un programa que queremos ejecutar.
- Se hace con el paquete `sys` que proporciona una lista de los argumentos de la línea de comando.

```
import sys

print("Number of arguments: {0}".format(len(sys.argv)))
print("Arguments:")
for arg in range(len(sys.argv)):
    print("{0:>4}: {1} ".format(arg,sys.argv[arg]))
```

Argumentos, un ejemplo más

```
import sys

if len(sys.argv)<4:
  print("please use: \"num1 op num2\" as arguments")
  sys.exit()

L=sys.argv[1:]
if L[1]=='+':
  r=int(L[0])+int(L[2])
elif L[1]=='-':
  r=int(L[0])-int(L[2])
else:
  print("unsupported operation")
  sys.exit()

print("{}{}{}={}".format(L[0],L[1],L[2],r))
```

Acceso al sistema de ficheros

- Con el paquete `glob` se obtiene acceso al sistema de ficheros.
- Ojo con experimentes: puedes dañar tus datos!

```
import glob

py_files= glob.glob("[Aa]*")

for f in py_files:
  print("found file:",f)
```

HAY que tener cuidado:

```
from subprocess import call
call(["ls", "-l"])
#call(["rm", "-rf /"])
```

... y ejecuta el programa (con la línea descomentada) como administrador (root) :-)
Te borra todo el disco duro.

Puede ser un *troyaner* embebido en algún código de python bajado desde la red.

un ejemplo de cálculo (el problema)

- Queremos calcular el volumen de un cuerpo modelado con una malla de triángulos...
- ¿Hay una herramienta ya disponible?
- ¿Programamos nuestra herramienta?
- ¿Buscamos una *solución* en la red?
- ¿Es todo un proceso fácil, seguro, rápido?

un ejemplo de cálculo (solución)

- Tenemos que entender la matemática detrás...
- Tenemos que tener acceso a los datos (ficheros)...
- Tenemos que saber que se cumplen las condiciones requeridas...
  - son triángulos que forman una malla cerrada (o varias mallas cerradas)
  - todos los triángulos están definidos con orientación idéntica
  - la superficie no se interseca a si misma
    (esta parte muchas veces no está mencionada en la red!!)
- (Comprobar las condiciones de aplicación es (a veces) la tarea más compleja, el mero cálculo es fácil...)

un ejemplo de cálculo (código)

```python
import csv
import sys

def Det(a,b,c): # volumen del paralelepipedo es producto mixto o determinante
  return c[0]*(a[1]*b[2]-b[1]*a[2])+c[1]*(a[2]*b[0]-b[2]*a[0])+c[2]*(a[0]*b[1]-b[0]*a[1])

def ReadAndCompute(filename):
  f=open(filename,"rt")
  reader=csv.reader(f,delimiter=" ",skipinitialspace=True)
  vertex_cnt=0
  volume=0.0
  for row in reader:
    if len(row):
      if row[0]=="vertex":
        if vertex_cnt==0:
          a=[float(row[1]),float(row[2]),float(row[3])]
        elif vertex_cnt==1:
          b=[float(row[1]),float(row[2]),float(row[3])]
        else:
          c=[float(row[1]),float(row[2]),float(row[3])]
        vertex_cnt+=1
      if vertex_cnt==3:
        volume+=Det(a,b,c)
        vertex_cnt=0
  f.close()
  print("volume ",volume/6) # dividimos entre 6 porque son pirámides triángulares

print("simple volume calculator, (c) 2018-2020 Arno Formella.")
if len(sys.argv)<2:
  print("first argument should specify file (in correct format!)")
else:
  ReadAndCompute(sys.argv[1])
```

Recursión

- La recursión es la forma de expresar un cálculo con la idea de divide-y-vencerás, es decir, se asume una solución para una instancia del problema más pequeño cuyo resultado se usa para calcular el resultado deseado.
- cálculo del factorial: $f(n) = n \cdot f(n-1)$
- pero tener en cuenta que: $f(0) = 1$
- Compiladores modernos son capaces de automáticamente convertir una recursión en una iteración que suele ser más eficiente (en tiempo de cálculo y uso de memoria).

Factorial recursivo

```
def factorial(n):
  if n==0:
    return 1
  return n*factorial(n-1)

print(factorial(5))
```

Programas modulares

- Para conseguir una mejor legibilidad, manejabilidad y reusabilidad se suele dividir programas en módulos.
- Conceptos parecidos son el uso de objetos o clases, el uso de paquetes, o el uso de bibliotecas, cada uno con sus características específicas dependiendo del lenguaje de programación.
- Los módulos suelen contener funcionalidades relacionadas entre si y pueden contener datos propios, con o sin acceso desde fuera.
- Los módulos más simples contienen solamente definiciones de funciones.

Módulos en python

- Ya usamos los módulos: `math`, `random`, `sys`, `csv`
- Se hacen disponibles con la sentencia `import`.
- Con `from XXXX import *` se importa directamente las definiciones y ya no hace falta el prefijo del módulo para el acceso.
- Solamente si dos módulos importados definen funciones con el mismo nombre se tiene que usar el prefijo en la llamada.

Un módulo simple

```python
# Fibonacci numbers module

def fib(n):
  # write Fibonacci series up to n
  a, b = 0, 1
  while b < n:
    print("{0}, ".format(b),end="")
    a, b = b, a+b
  print()

def fiblist(n):
  # return Fibonacci series up to n as list
  result = []
  a, b = 0, 1
  while b < n:
    result.append(b)
    a, b = b, a+b
  return result
```

El uso del módulo

```
import fibo

fibo.fib(4)
list = fibo.fiblist(4)
print("Result: ",list)
```

miramos con el interprete...

Se puede usar un módulo como script (o guión)

```
# used when executed as script
if __name__ == "__main__":
  import sys
  print(fiblist(int(sys.argv[1])))
```

- es decir, el método principal está disponible cuando se invoca como script (guión) ejecutable
- miramos con el interprete...
- Otra utilidad es `dir` que lista el contenido del módulo (usa como `dir(modul)` o `dir()`).
- Diferentes módulos se pueden unir en un paquete de python realizando una jerarquía de ficheros python según una estructura determinada.

A *quine* (una cosa curiosa)

```
Get a sheet of paper and writing instrument;
copy the sentence in quotes that occurs after
this sentence, then write a quote mark, then
copy the sentence again, then put a final quote.
"Get a sheet of paper and writing instrument;
copy the sentence in quotes that occurs after
this sentence, then write a quote mark, then
copy the sentence again, then put a final quote."
```

quines en Python

- Un *quine* es un programa (no vacio) que imprime a si mismo.
- Si un lenguaje es Turing-completo (y puede generar salidas adecuadas) siempre es posible construir un *quine*.
- python2.X:

```
s='s=%r;print s%%s';print s%s
```

- python3.X:

```
s='s=%r;print(s%%s)';print(s%s)
```

- Es un resultado de la teoría de computabilidad:
  algo como un teorema de punto fijo de la informática.

algoritmos

- secuencia finita (*receta*) de pasos (o instrucciones) bien definidos para lograr una tarea
- los pasos, en principio, deben ser ejecutables por un humano
- las instrucciones deben realizar una transformación finita del estado (o configuración) del sistema
  (alteraciones finitas de bits de la configuración del sistema)
- lograr una tarea significa que se puede verificar una propiedad del estado del sistema (posiblemente parcial) mediante un algoritmo
- un programa es un algoritmo formulado en un lenguaje formal de tal forma que (después de posibles transformaciones automáticas) se puede ejecutar en un ordenador

complejidad y computabilidad

- algoritmos se caracterizan por su complejidad tanto en tiempo necesario (medido en instrucciones o pasos) como en memoria usada (tamaño de las configuraciones intermedias)
- existen *problemas* que no son computables
  o en otras palabras: hay cosas que no tienen solución
- Ejemplos: equivalencia entre gramáticas formales, averiguar si un programa ejecuta todas sus líneas.

variantes de algoritmos

https://www.youtube.com/watch?v=ixTddQQ2Hs4
https://www.youtube.com/watch?v=kPRA0W1kECg

- algoritmos deterministas
- algoritmos randomizados (tipo Las Vegas, tipo Monte Carlo)
- [algoritmos no-deterministas (no los vemos), concepto teórico]
- nota: los conceptos basicamente permiten calcular las mismas funciones computables (si se requiere reproducibilidad)

- algoritmos secuenciales (los vemos)
- [algoritmos paralelos (no los vemos)]

observa: hay que saber exactamente lo que se quiere hacer

ordenamos números (o palabras, o espaguetis)

- necesitamos establecer cuál es el criterio para ordenar
- por ejemplo: de menor a mayor valor, o vice versa
- necesitamos unas operaciones (instrucciones) base
- por ejemplo: dado una pareja de valores, ordena la pareja
- aplicamos esta operación consecutivamente sobre diferentes parejas para finalmente obtener la secuencia ordenada (claro, el truco es qué parejas seleccionar!)
- es conveniente tener un algoritmo que verifique que la secuencia esté ordenada
- *animación en pizarra...*

algoritmo determinista

- el problema de comprobar si una secuencia de números está ordenada (por ejemplo de menor a mayor valor) tenía solución bastante sencilla:
- comprobamos que todas las parejas adyacentes estén bien ordenadas.
- es un algoritmo determinista.

algoritmo randomizado tipo Las Vegas

- ordenamos una secuencia de números de forma divertida:
- reordenamos los números al azar y comprobamos si (por suerte) estén ordenados,
- en caso que sí: hemos ordenado,
- en caso que no: repetimos el proceso.
- es un algoritmo randomizado tipo Las Vegas.
- observa: ordena! aunque no sabemos cuanto tiempo necesitará.

algoritmo randomizado tipo Monte Carlo

- calculamos el valor de π tirando dardos a una diana (redonda) de forma aleatoria,
- contamos cuantos dardos alcanzan la diana y cuantos quedan por lo menos dentro del marco (quadrado) alrededor
- llamamos D y M las cantidades
- entonces $\pi \approx 4D/(D+M)$.
- es un algoritmo randomizado tipo Monte Carlo.
- observa: calcula un resultado preciso (correcto) solamente con cierta probabilidad pero sabemos exactamente cuanto tiempo necesita.

ejemplo de algoritmo

Calcular π mediante un algoritmo Monte Carlo:

```
# pi with Monte Carlo randomized algorithm

import random

D=0
M=0
while D+M<1000000:
  x=random.uniform(-1.0,1.0)
  y=random.uniform(-1.0,1.0)
  if x*x+y*y<=1.0:
    D+=1
  else:
    M+=1

print("pi= ",4.0*D/(D+M))
```

Bucle principal de XaTcobeo *(some debug removed)*

```
int main() {
#if defined(USE_MIMIC)      // used when complete simulation on computer enabled
  XMS_DEBUG(1,"\n\nXatcobeo Mission Software Mimicry\n\n");
  timer_reset();                                      // initialize real time system
  sigsetjmp(init,-1);                     // needed to simulate unexpected power-down
  FLASH_init();                              // initialize simulation of flash memory
  ttc_init();                           // initialize telecommand-telemetry subsystem
  set_obsw_reset(ObswReset);                                   // simulated total reset
#else
  ObswReset();                                                // total reset at power-up
#endif

  XMS_DEBUG(1,"\n\nXatcobeo Mission Software\n\n");
  Init();                                                   // initialize state-machine
  while(1) {                                    // never ending handling of five states
    ControlHandleNextTaskNoEclipse();
    switch(system_variable.current_mode) {
      case MODE_STARTUP:       Startup(); break;
      case MODE_NOMINAL:       Nominal(); break;
      case MODE_OPERATION:     Operation(); break;
      case MODE_COMMUNICATION: Communication(); break;
      default:                    // may happen when current_mode gets corrupted
        ControlModeChange(MODE_SAFEHOLD);
        ErrorProgramFlow(XEC_OBSW_MAIN_MODE);
      case MODE_SAFEHOLD:      Safehold(); break;
    }
  }
  for(;;);                          // just in case, wait for hardware power-down

  XMS_DEBUG(1,"Main: we should never get here\n");
  return 0;
}
```

máquina de estados de HumSAT-D

idle/periodically

Off

by telecommand

Init

Startup

by scheduler

Safehold

Nominal

Comm.

on error

autonomous

User

algoritmos de ordenación (deterministas)

- visualizamos (en pizarra) el algoritmo de *bubble sort*
- visualizamos (en pizarra) el algoritmo de *merge sort*

- la complejidad en tiempo es diferente
- *bubble sort* necesita en el orden de n^2 operaciones base para ordenar n valores
- *merge sort* necesita en el orden de $n \log n$ operaciones base para ordenar n valores

unas funciones de utilidad:

```
def PairIsSorted(v,i,j):
    return v[i]<=v[j]

def IsSorted(v):
    for i in range(0,len(v)-1):
        if not PairIsSorted(v,i,i+1):
            return False
    return True

def PairSort(v,i,j):
    if not PairIsSorted(v,i,j):
        v[i],v[j]=v[j],v[i]
```

el algoritmo de ordenación (terminación por comprobación):

```
def BubbleSort(v):
    while not IsSorted(v):
        for i in range(0,len(v)-1):
            PairSort(v,i,i+1)

#import random
#v = random.sample(range(10001),10001)
v = [ 8,2,4,3,6,1,7,5 ]

print(v)
BubbleSort(v)
print(v)
```

bubble sort unos comentarios

Después de haber visto tal algoritmo, mencionamos:

- deberíamos comprobar formalmente que el algoritmo es correcto (es una tarea de la informática)
- tal como está implementado el algoritmo implicitamente garantiza que la secuencia esté ordenada al final (bueno, si la función `IsSorted()` es correctamente implementada), tal técnica facilita la depuración de programas
- si hay solamente unos pocos valores grandes en desorden, el algoritmo es bastante rápido, es decir, el tiempo de ejecución depende tanto del orden inicial de la secuencia como de n, en otras palabras, el tiempo puede ser en el orden de n en el caso mejor, o en el orden de n^2 en el caso peor
- el algoritmo no necesita mucha memoria adicional

La función recursiva de ordenación:

```
def mergesort(w):
  s=len(w)
  if s>1:
    u=w[0:s//2]      # primera parte
    v=w[s//2:s]      # segunda parte
    mergesort(u)     # ordenar primera parte
    mergesort(v)     # ordenar segunda parte
    merge(u,v,w)     # intercalar las dos partes

#import random
#v = random.sample(range(10001),10001)
v=[4,5,3,7,8,1,2,6]

print(v)
mergesort(v)
print(v)
```

Ordenación recursiva: la intercalación

La función que intercala dos listas ordenadas `u` y `v` en una lista `w` (asumiendo longitudes de las colecciones adecuadas):

```
def merge(u,v,w):
  for i in range(0,len(w)):
    if u!=[] and v!=[]:
      if u[0]<v[0]:
        w[i]=u.pop(0)
      else:
        w[i]=v.pop(0)
    elif u==[]:
        w[i]=v.pop(0)
    else:
        w[i]=u.pop(0)
```

merge sort unos comentarios

Después de haber visto tal algoritmo, mencionamos:

- deberíamos comprobar formalmente que el algoritmo es correcto (es una tarea de la informática)
- tal como está implementado el algoritmo NO garantiza implicitamente que la secuencia esté ordenada al final (por eso lo comprobamos a posteriori)
- independiente del orden inicial de la secuencia, *merge sort* realiza básicamente siempre las mismas operacionces, es decir, tiene un tiempo de ejecución que solamente depende de n, siempre es en el orden de $n \log n$
- el algoritmo necesita memoria adicional (una copia de la secuencia)

merge sort en Python (versión iterativa)

```
def Merge(v,w,L,M,R):
    i,j = L,M
    for k in range(L,R):
        if j>=R or ( i<M and PairIsSorted(v,i,j) ):
            w[k] = v[i]
            i = i+1
        else:
            w[k] = v[j]
            j = j+1

def MergeSort(w):
    s=1
    n=len(w)
    while s<n:
        v = w[:]
        for L in range(0,n,2*s):
            Merge(v,w,L,L+s,min(L+2*s,n))
        s = 2*s
```

programa principal para *merge sort* en Python

```
import random
v = random.sample(range(1000001),1000001)
#v = [ 8,2,4,3,6,1,7,5 ]
#v = [ 'd','f','a','z','c' ]
#v = [ "This","is","a","short","sentence" ]

MergeSort(v)

if IsSorted(v):
    print('is sorted')
else:
    print('is NOT sorted')
```

Asumiendo que el tiempo para ejecutar la operación base con un elemento es siempre lo mismo, y que actuemos con datos de unos $n = 1000000$ elementos...

- si necesitamos 1 segundo para calcular con un algoritmo de orden lineal (orden de n),
- necesitaríamos unos 20 segundos con un algoritmo de orden cuasi-lineal (orden de $n \log n$),
- necesitaríamos unos 11 días con un algoritmo de orden cuadrático (orden de n^2),
- necesitaríamos unos 31000 años con un algoritmo de orden cúbico (orden de n^3),
- necesitaríamos más de 2 veces la edad del universo con un algoritmo de orden cuártico (orden de n^4),
- y necesitaríamos cuasi eternamente :-) con un algoritmo exponencial (orden de 2^n).

Sistema operativo

- programa (monolítico o modular) de un sistema informático que gestiona los recursos hardware y provee servicios a las aplicaciones
- permite una capa de abstracción para las aplicaciones
- provee una estandarización de desarrollo para componentes de nivel bajo
- Windows, Linux, MAC OS, iOS, Android, etc.

Herramientas o utilidades

- programas genéricos de servicio
- ejemplos: compresores, archivadores, antivirus, etc.
- entornos de programación
- ejemplos: editores (gedit, Vi, notepad), compiladores, IDEs (IDLE, PyCharm, eclipse, NetBeans), etc.
- programas de aplicación específica
  - navegadores: Firefox, Chrome, Safari, Edge, Opera, etc.
  - ofimática: document editor, hoja de cálculo, calendarios, etc.
  - ocio: gestor de videos o música, juegos, etc.

Funciones básicas del sistema operativo

Gestion de procesos: permite que varios programas se ejecuten a la vez (cuasi- o realmente en paralelo), permite sistemas multi-tarea, multi-usuario, servidores

Abstracción hardware: trabaja con los diferentes modelos de hardware (p.ej: distintos discos duros) de forma que los programas no tengan que preocuparse de los detalles.

Gestión de la Entrada/Salida: permite leer y escribir información en los dispositivos de E/S. Incluye la gestión de archivos y redes.

Gestión de memoria: permite que los programas usen la memoria RAM de forma protegida sin notar que hay otros programas usándola al mismo tiempo

Bases de datos

- Es un conjunto de datos normalmente de un ámbito específico que están almacenado de forma sistemática.
- Se gestionan mediante software de control llamando DBMS (*database management software*) que permite manipular los datos mediante un lenguaje (por ejemplo SQL *Structured Query Lenguage*).
- Permiten operaciones eficientes incluso a grandes cantidades de datos almacenados en sistemas distribuidos.
- Son componentes principales de sistemas de información (tanto en la Web como en empresas y bancos).
- Ejemplos: Oracle, mongoDB, MySQL, PostgreSQL (y muchos más)
- Para la gestion las propiedades ideales llamados ACID son muy importantes: *Atomicity, Consistency, Isolation, and Durability*, especialmente en sistemas concurrentes.

Ejemplo de bases de datos

https://www.flightradar24.com

flightradar24 LIVE AIR TRAFFIC

Apps | Add coverage | Data / History | Social | Press | About | Business services

AIRCRAFT 406 / 13,050

AIRPORT DELAYS

AIRPORT	ARR	DEP
Lima (LIM)	1.7	3.8
Delhi (DEL)	2.0	2.5
Mumbai (BOM)	1.8	2.3
Shanghai (PVG)	0.4	2.9
Munich (MUC)	0.6	2.3

Go to delay map

TWEETS

The @BoeingAirplanes 787-10 on its way home after a successful @DubaiAirshow. #DAS... 28 minutes ago

About 200 Douglas and McDonnell-Douglas aircraft in the air. Including the Boeing ... 3 hours ago

Nearly 13,000 flights in the air as the sun sets across the Atlantic and Australia... 1 day ago

BLOG POSTS

AvTalk Episode 18: A United 747 Farewell 3 days ago

New Airline Data Pages on Flightradar24 4 days ago

United Says Farewell to the 747 1 week ago

This site requires cookies in order to work properly. Find out more. Got it

Portugal

Spain

Ejemplo de bases de datos

http://www.ine.es/apellidos/formGeneralresult.do?vista=4

Instituto Nacional de Estadística

01 El INE | 03 Métodos y proyectos | 05 Prensa | Censo electoral
02 INEbase | 04 Formación y empleo | 06 Productos y servicios | Sede electrónica

ES | EN

INEbase / Demografía y población / Frecuencias de apellidos
?

Frecuencias de apellidos

- Apellidos por provincia de residencia
- Apellidos por provincia de nacimiento
- Apellidos por nacionalidad

Apellidos por nacionalidad

Seleccione valores a consultar:

País:

Seleccionados **1** Total 141

TOTAL
ESPAÑOLES
EXTRANJEROS
Afganistán

Apellido:

FORMELLA

Consultar

No existen habitantes con el apellido consultado o su frecuencia es inferior a 5 para el total nacional o para la nacionalidad seleccionada

NOTAS:
1) Por secreto estadístico sólo se muestran los apellidos cuya frecuencia es mayor que 5 en alguno de los dos apellidos para el total nacional. Para las nacionalidades seleccionadas se exige además una frecuencia de al menos 5 en alguno de los dos apellidos. Por esta razón, la frecuencia total para algunos apellidos puede no ser igual a la suma de las nacionalidades que se muestran.
2) Datos procedentes de la Estadística del Padrón Continuo a fecha 01/01/2016

Aplicaciones específicas

- diseño (CAD)
- cálculo de propiedades (FEM, CFD)
- simulación
- gestión de empresa (ERP)
- control de producción (u otros procesos)
- y muchísimo más...

Licencias de software

- software privativo

  How can I use the software that is provided as part of the service? We do not sell our software or your copy of it – we only license it. Under our license we grant you the right to install and run that one copy of the software on one licensed device (the first licensed device) for use by one person at a time, but only if you comply with all the terms of this Supplement. The user whose Microsoft account is associated with the software license for the first licensed device is the "licensed subscriber." Provided that you comply with all the terms of this Supplement, you may install and run copies of the software on licensed devices (including on the first licensed device) as follows:

  Office 365 Home: On five PCs/Macs and five tablets, for use only by members of the same household as the licensed subscriber. [1]

  Office 365 University: On one PC/Mac or tablet and one additional PC/Mac or tablet, for use only by the licensed subscriber. [2]

  Office 365 Personal: On one PC/Mac and one tablet, for use only by the licensed subscriber. [1]

- software libre
  https://choosealicense.com/licenses/
- software de servicio (SaaS)

Licencias de software

- dominio público: basicamente podemos hacer lo que nos apetezca
- permisivo: básicamente solo tenemos que decir de donde viene (derecho de autor)
- LGPL: podemos usar como biblioteca para cualquier fin, pero modificaciones o incorporaciones hacen todo el producto copyleft
- copyleft: tiene que estar disponible en código abierto
- proprietario: tiene condiciones muy especificas para sus uso

Hay que respetar la liciencia y los derechos (a veces hay condiciones que distingen entre uso privado, uso comercial, uso educativo, uso de investigación).

Software que no contiene una licencia explícita es sospechoso de fraude.

ordenador o computadora

- Es una máquina (o dispositivo) digital
- con medio de entrada de datos
- con medio de almacenamiento de datos (registros y memoria)
- con unidad(es) de procesamiento
- con medio de salida de datos.
- El procesamiento transforma datos *por un lado* en datos *por otro lado*.
- Los nuevos datos cambian el estado de la memoria.
- Con la ayuda de un programa de instrucciones se ejecutan, paso a paso, cambios del estado según un reloj.

¿es un ordenador o computadora?

Northbridge (with heatsink)
Southbridge
PCI Slot (x5)
IDE Connector (x2)
AGP Slot
DRAM Memory Slot (x2)
20-pin ATX Power Connector
CMOS Backup Battery
CPU Fan & Heatsink Mounting Points
Connectors For Integrated Peripherals
PS/2 Keyboard and Mouse, Serial Port, Parallel Port, USB (x6), Ethernet, Audio (x3)
CPU Socket

¡no!

¿es un ordenador o computadora?

By Mwilde2 (Own work) [CC BY-SA 4.0 (http://creativecommons.org/licenses/by-sa/4.0)], via Wikimedia Commons

¡sí!

¿es un ordenador o computadora?

20.60mm
30.67mm
28.69mm

EMMC
4GB

USB2.0
PHY

Wi-Fi
Bluetooth LE

COAX
WiFi
Ant

Emb. WIFI Antenna
Keep-out Zone

8.28mm
0
4.83mm
1.47mm
0
17.75mm
23.53mm

69 70
70-pin conn
1 2
18.30mm

1GB LPDDR3
Package-on-Package

Intel® Atom™
Processor
(dual-core 500 MHz)

Power
Management
IC

0
20.23mm
0

By Mwilde2 (Own work) [CC BY-SA 4.0 (http://creativecommons.org/licenses/by-sa/4.0)], via Wikimedia Commons

hardware (echando un vistazo...)

- destripamos un ordenador (virtualmente)
- caja, cables, placas, chips, disipadores, discos duros, fuentes de alimentación, connectores, tornillos, pilas, etc. etc. etc.

software (una pincelada...)

- firmware
- middleware, driver
- sistema operativo
- aplicaciones

el desarrollo de software se basa en interfaces estandarizadas

- conexiones (como están conectados los componentes)
- protocolos (cual son las secuencias de comunicación)
- estructuras de datos (como se representa el estado)

ordenadores: una breve historia

- Charles Babbage (1837): *analytical machine*
  (primer concepto de ordenador general)
- Alan Turing (1936): *universal Turing machine*
  (modelo matemático de la computabilidad)
- Konrad Zuse (1941): *Z3*
  (primer ordenador general construido, además tenía 22-bit números flotantes)
- Howard Aiken (1944): *Mark I*
  (architectura Harvard, datos y programa separados)
- John von Neumann (1945): descripción de architectura von Neumann (o Princeton)
  (ya estaba presente en patentes de Zuse en 1936)

Diseños de ordenadores

Hay tantos *dibujos* (diseños) de CPU's que hay libros sobre el tema, de todos comunes es:

- almacenamiento de un estado interno (registros, memoria)
- cambio de estado en uno o varios pasos según un reloj
- interconexión de registros/memoria con componentes funcionales
  (las que calculan funciones booleanas)

unidad de procesamiento central (CPU): componentes

- reloj (con fases) que determina cuando se cambia el estado
- unidad lógica y aritmética (ALU) que calcula
- registros donde se almacena el estado
- buses de conexión que pasan datos de un lado a otro(s)
- unidad de control (microprograma) que simplifica el control y permite solapamiento (mejor eficiencia)
  (Moore and Mealy (1955/56): sistemas finitos de control)
- unidades de entrada/salida

un ejemplo simple de una CPU, sin mucho detalle...

construimos una CPU

- componentes básicos
  - conexiones
  - puertas lógicas
  - registros
- componentes complejos
  - reloj (con fases)
  - unidad lógica y aritmética (ALU)
  - buses de conexión
  - unidad de control
  - unidades de entrada/salida

el camino hacia una CPU, muy por encima, pero con algunos detalles

- si queremos sumar dos números (en sistema decimal), sabemos como hacerlo (se aprende en la primaria)
- lo podemos hacer también en el sistema binario:

$$
\begin{array}{rrl}
 & 10010110101 & \\
 & 10011011 & \\
+ & 1\ \ \ 1\,1\,1\,1\,1\,1 & \text{acarreo} \\
\hline
 & 10101010001 &
\end{array}
$$

funciones binarias

- para eso necesitamos una tabla para *sumar* los bits y tenemos que tratar con el acarreo, por eso necesitamos una tabla *más grande* con tres bits de entrada:

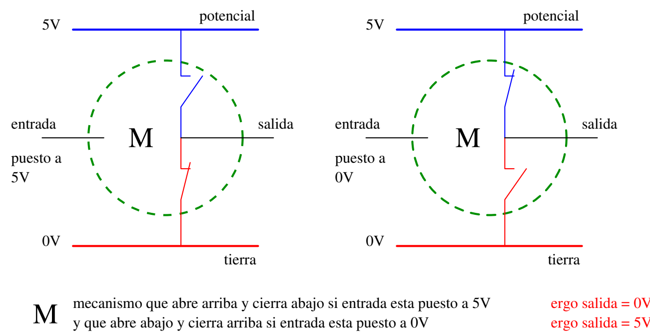
			bit de acarreo	bit de suma
0	0	0	0	0
0	0	1	0	1
0	1	0	0	1
0	1	1	1	0
1	0	0	0	1
1	0	1	1	0
1	1	0	1	0
1	1	1	1	1

- tenemos que *construir* de alguna manera un dispostivo que realiza estas *operaciones* de forma automática

puertas lógicas

- una *puerta lógica* es un dispositivo simple que realiza una operación binaria simple (hay muchos tipos, realizamos solamente dos para ver la idea)
- tenemos que representar los valores 0 y 1 de alguna manera: usamos de forma abstracta un *potencial*, por ejemplo, una tensión eléctrica, una presión neumática, una presión hidráulica, o campo magnético etc.
- o en concreto (como ejemplo): 5V (voltios de tensión) que representa el valor 1, o 0V que representa el valor 0
- ojo en la vida real tendrémos asociado siempre también una corriente (o flujo) y una resistencia, pero lo ignoramos
- ojo, en la vida real cada cambio de estado necesita su tiempo y tiene un intervalo de transición, pero lo ignoramos

puerta lógica: NEG (negación)



el mecanismo se puede realizar en muchas tecnologías, por ejemplo, con transistores

puerta lógica: NOR (negación del 'o' (*or*))

5V potencial

no se hace en realidad asi
pero visualiza el concepto

entrada 1 M salida 5V potencial

entrada 2 M salida

0V

tierra

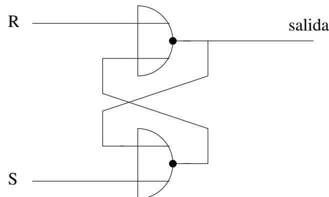
con está puerta podemos realizar cualquier función binaria, conectando las puertas de forma adecuada

realización del XOR (*exclusive or*) con NORs

entradas	salida
0 0	0
0 1	1
1 0	1
1 1	0

registros

- con los NOR se puede realizar registros que guardan un estado (o bien 1 o bien 0)
- por ejemplo, el registro SR-NOR

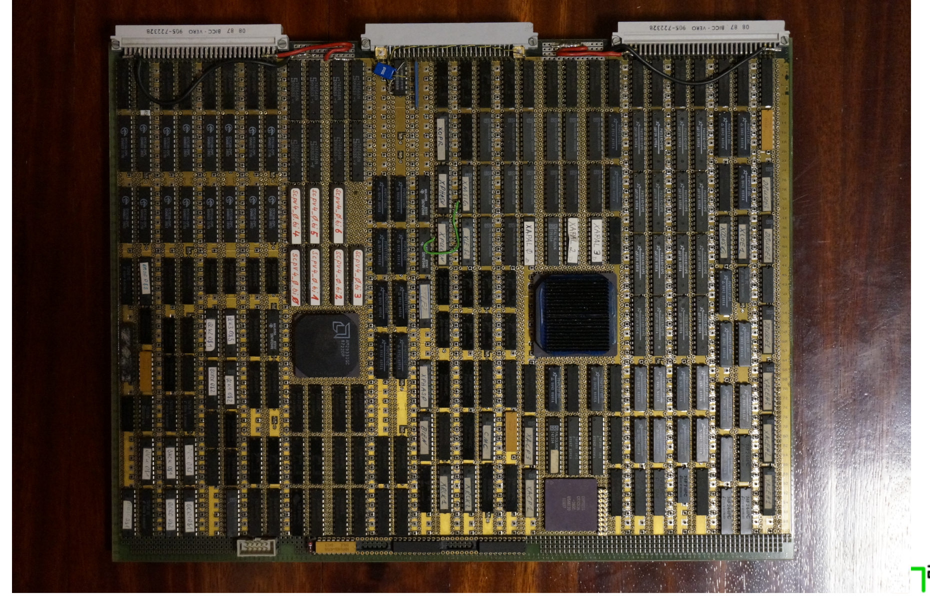


- el truco es: conectar una salida a una entrada y mirar que pasa si hay una transición de una entrada de 1 a 0 o al revés (observa hay unas entradas *prohibidas*, se debe evitar que las dos sean 1)
- existen muchos otros diseños más sofisticados...

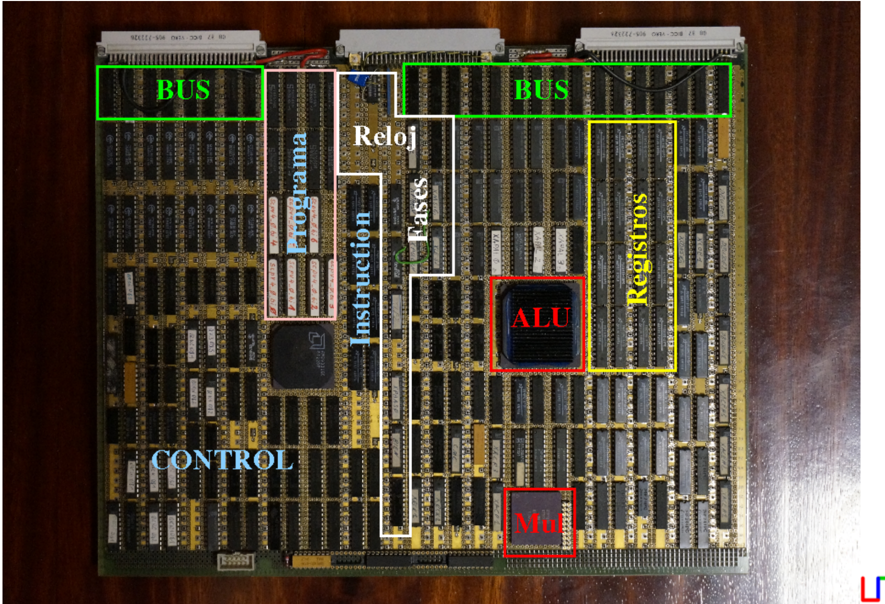
registros

- podemos realizar con las puertas circuitos que calculan cualquier función binaria que nos interesa, incluso con muchas salidas
- podemos guardar los resultados en registros (o memoria que son nada más que registros en una tecnología específica)
- y realizar retroalimentación con una señal de reloj para cambiar de un estado o *configuración* (los 0s y 1s en todos los registros) a otro estado o *configuración* y así sucesivamente para realizar de esta manera algoritmos

Spark CPU (unidad de cálculo entero) 1987/88



Spark CPU (unidad de entero) 1987/88



dispositivos de entrada

- teclado (tecla(s) presionada(s))
- ratón (posición relativa, estados botones)
- tableta táctil (posición absoluta, presión, etc.)
- lápiz óptico (posición relativa, ángulo, presión, etc.)
- periféricos para entrada de datos (sensores)
- incorporan (muchas veces) conversor analógico a digital

principalmente son de dos tipos:

- responden a una interrogación con su estado
- comunican un cambio de estado de forma asíncrona

dispositivos de salida

- monitores (matriz de colores)
- impresoras (2D o 3D, matriz de colores)
- plotter
- altavoces
- antenas
- periféricos para salida de datos (actores)

principalmente son de dos tipos:

- reciben los datos mediante un protocolo de comunicación
- proporcionan el estado de forma continua o pulsada

memorias

- dispositivo que almacena el estado del ordenador
- tanto los datos como las instrucciones (programas)

se diferencian según

- tecnología: electrónico, magnético, óptico, (bio-)químico
- capacidad: pocos bits hasta muchos terabyte
- velocidad de acceso: de pocos nanosegundos hasta varios milisegundos
- ancho de banda: de pocos bauds (bit por segundo) hasta varios gigabyte por segundo
- granularidad de acceso: por bit, por página, por bloque, por sector
- tipo de almacenamiento: volatile, no-volatile, re-escribible
- densidad de almacenamiento (volumen por byte)
- consumo de energía

unidad artimético-lógico (ALU)

- circuitos que realizan las operaciones aritméticas y lógicas
- ALU contiene: sumadores, multiplicadores, divisores, operaciones sobre bits (lógicos AND, OR, EXOR, rotaciones y desplazamientos de bits, etc.)
- trabajan con diferentes codificaciones: números enteros, con o sin signo, números flotantes, y combinaciones más complejas (MMX, SSE, etc.)
- están conectados a registros y buses
- están controlados por la unidad de control (y el reloj)

tipos de ordenadores

- superordenadores (investigación, militar)
- servidores (bancos, grandes empresas, centros de datos)
- ordenadores de uso general (PCs, notebooks)
- ordenadores de uso específico (smartphones, satélites, coches, aviones...)
- ordenadores empotrados (cámaras, relojes, gafas, coches, aviones...)

Las diferencias son cambiantes y difusas, es un campo que evoluciona mucho (lo que hoy está en un teléfono, hace 40 años fue un superordenador).

video en la red sobre el funcionamiento de una CPU

https://www.youtube.com/watch?v=cNN_tTXABUA

video de 20 minutos que resume muchas de las cosas que hemos visto!

- conjunto de componentes interrelacionados
- que consisten de
  - hardware (electrónica, mecánica)
    ordenadores, monitores, ratones, teclados, dispositivos de redes de comunicación, impresoras, memoria, otros dispositivos periféricos.
  - software (programas),
    firmware de control básico, sistemas operativos, programas de desarrollo, editores, compiladores, bases de datos, applicaciones de alto nivel, visualizadores, etc.
  - wetware (personal informático)
    analistas, diseñadores, programadores, operarios

- mediante redes (sobre todo internet)
- se conectan ordenadores de diferentes características
- y se ofrecen servicios (datos/información o cálculo)
- bajo ciertas condiciones
  (paga por servicio, paga con datos, paga por anuncios)

ejemplos: buscadores, juegos, servicios administrativos, noticias, música, videos, ofertas empresariales, cálculo online, redes sociales
peligro: dependencia de tecnología

Sociedad digital

- Hoy muchos dispositivos están conectados a la internet (ordenadores, móviles, elementos domésticos).
- En 2016 ya había más móviles que ordenadores con conexión a la red.
- Muchas gestiones de la vida cotidiana ya se arreglan media acciones digitales (compra-venta, reservas, entrega de documentos, matrículas, acceso a información, etc.)

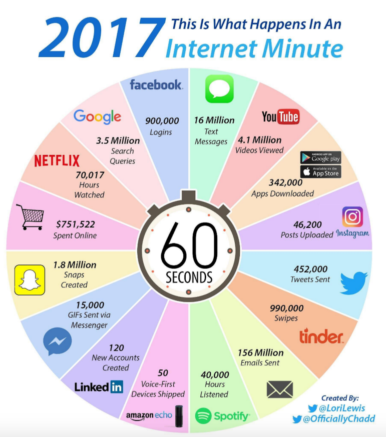
Where is Everyone?

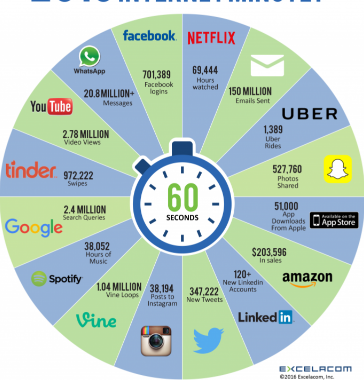
Lokal Marketplace
Newspapers / Magazines
Radio
Television
Websites
Blogs
Social Networks
Social News
Podcasts / Vodcasts
Gaming
Targeted
baekdal.com

1800 1850 1900 1920 1940 1960 1980 1990 1995 1998 2000 2002 2004 2006 2007 2008 2009 2010 2015 2020

http://www.baekdal.com/analysis/market-of-information

un minuto del internet





http://www.visualcapitalist.com/happens-internet-minute-2017

http://www.visualcapitalist.com/what-happens-internet-minute-2016

un minuto del internet

2018 This Is What Happens In An Internet Minute

facebook 973,000 Logins
18 Million Text Messages
YouTube 4.3 Million Videos Viewed
Google play / App Store 375,000 Apps Downloaded
174,000 Scrolling Instagram
481,000 Tweets Sent
1.1 Million Swipes tinder
187 Million Emails Sent
twitch 936,073 Views
amazon echo 67 Voice-First Devices Shipped
38 Million Messages
25,000 GIFs Sent via Messenger
2.4 Million Snaps Created
$862,823 Spent Online
NETFLIX 266,000 Hours Watched
Google 3.7 Million Search Queries
60 SECONDS
Created By: @LoriLewis @OfficiallyChadd

2017 This Is What Happens In An Internet Minute

facebook 900,000 Logins
16 Million Text Messages
YouTube 4.1 Million Videos Viewed
Google play / App Store 342,000 Apps Downloaded
Instagram 46,200 Posts Uploaded
452,000 Tweets Sent
990,000 Swipes tinder
156 Million Emails Sent
Spotify 40,000 Hours Listened
amazon echo 50 Voice-First Devices Shipped
LinkedIn 120 New Accounts Created
15,000 GIFs Sent via Messenger
1.8 Million Snaps Created
$751,522 Spent Online
NETFLIX 70,017 Hours Watched
Google 3.5 Million Search Queries
60 SECONDS
Created By: @LoriLewis @OfficiallyChadd

http://www.visualcapitalist.com/internet-minute-2018

http://www.visualcapitalist.com/happens-internet-minute-2017

datos transferidos en la Red (un ejemplo...):

https://www.visualcapitalist.com/wp-content/uploads/2019/04/data-generated-each-day-wide.html

herramientas *on-line*

- buscadores
- Google, bing, Yahoo!, Baidu, Ask, Aol., DuckDuckGo, WolframAlfa, etc.
- herramientas de filtrado de información
- herramientas de organización de información
- herramientas de generación de contenido
- herramientas para compartir contenido
- herramientas de ofimática en línea
- herramientas de almacenamiento de datos
- herramientas de enseñanza (MOOC)

Tened cuidado con vuestros datos.