Informática

2018/19

Grado en Ingeniería Aeroespacial, Primer Curso

Dr. Arno Formella

Departamento de Informática
Escola de Enxeñaría Aeronáutica e do Espazo
(Escola Superior de Enxeñaría Informática)
Universidade de Vigo

18/19

profesorado (teoría)

Profesor: Arno FORMELLA
Web: http://formella.webs.uvigo.es
Correo: formella@uvigo.es
Tutorías Martes 9:30–13:30 y 16:30–18:30

usamos: FAITIC/TEMA
https://faitic.uvigo.es/index.php

profesorado (prácticas)

Profesor: Leandro RODRÍQUEZ LIÑARES
Web: ???
Correo: leandro@uvigo.es
Despacho: ESEI, 405
Tutorías: Miércoles y jueves 15:30–17:00

tutorías e idiomas

- Cambios puntuales de tutorías via aviso web
  (yo en mi página principal)
  y/o mediante correo via FAITIC/TEMA
- Idiomas: Galego, Castellano, English, Deutsch
- Las transparencias serán en castellano.
- Habrá textos e información en inglés.
- Hay dos grupos de prácticas en inglés.

horas de dedicación (planificación según guía)

	pres.	no-pres.	suma
Actividades introductorias	0.5	–	0.5
Sesión magistral	23.0	46.0	69.0
Prácticas en aulas de informática	20.0	40.0	60.0
Probas prácticas	4.5	5.5	10.0
Resolución de problemas y/o exercicios	2.0	6.0	8.0
Pruebas de respuesta larga	2.5	–	2.5
Suma	52.5	97.5	150.0

horas de trabajo (este curso)

	pres.	no-pres.	suma
Actividades introductorias	0.5	–	0.5
Sesión magistral	23.0	46.0	69.0
Prácticas en aulas de informática	20.0	40.0	60.0
Probas prácticas	3.5	6.5	10.0
Resolución de problemas y/o exercicios	0.0	9.0	9.0
Pruebas de respuesta larga	1.5	–	1.5
Suma	48.5	101.5	150.0

horas de trabajo (que se hace)

	pres.	no-pres.
Actividades introductorias	organizar	–
Sesión magistral	escuchar	repasar
	preguntar	analizar
		profundizar
		debatir
Prácticas en aulas de informática	aprender	preparar
	programar	completar
		documentar
Probas prácticas	programar solución	resumir
Resolución de problemas	elaborar respuesta	buscar
y/o exercicios		coordinar
Pruebas de respuesta larga	contestar	estudiar

prerrequisitos

- ningunos

evaluación continua

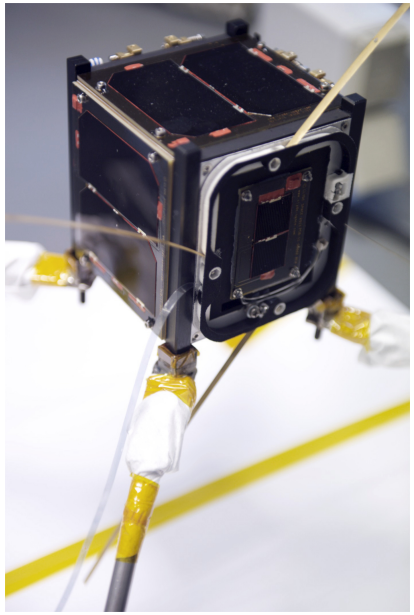
- entregas semanales en prácticas
  (no se evalua, pero documentan asistencia)
- dos evaluables en prácticas (con ordenadores)
- un evaluable en teoría
- examen final

vos y nos

¿Quién soy yo?

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

al espacio...



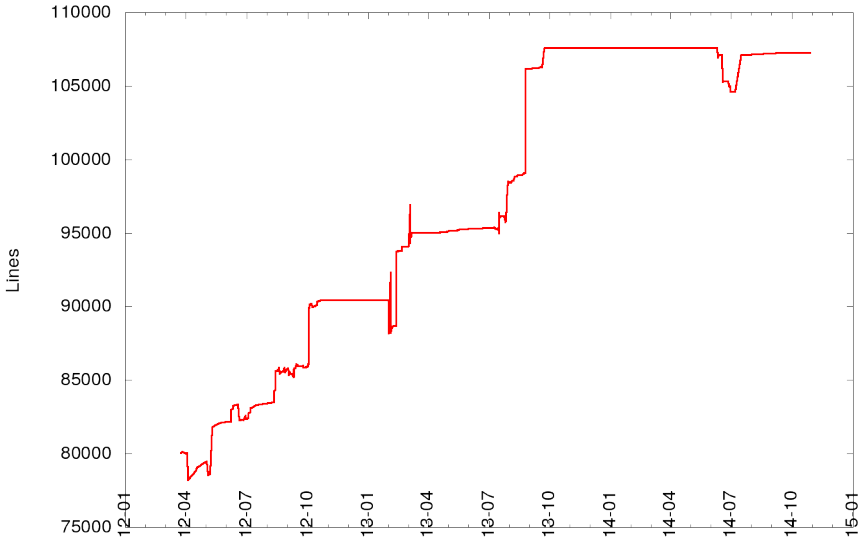
3 Satélites:

XaTcobeo, 14/02/2012

HumSAT, 21/11/2013

Serpens, 20/08–17/09/2015

HumSAT software evolution (lines of code en C)



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 (muchos en inglés).
- Los ejemplos de programas y algoritmos 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

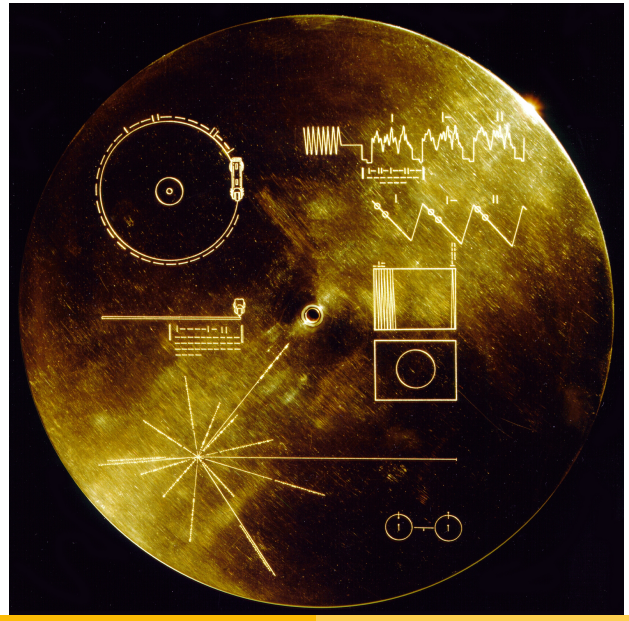
Competencias	Tipo	Cod.
Capacidad de análisis, organización y planificación	ser	CT1
Liderazgo, iniciativa y espíritu emprendedor	ser	CT2
Capacidad de comunicación oral y escrita en la lengua nativa	ser	CT3
Capacidad de aprendizaje autónomo y gestión de la información	ser	CT4
Capacidad de resolución de 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

nociones básicas

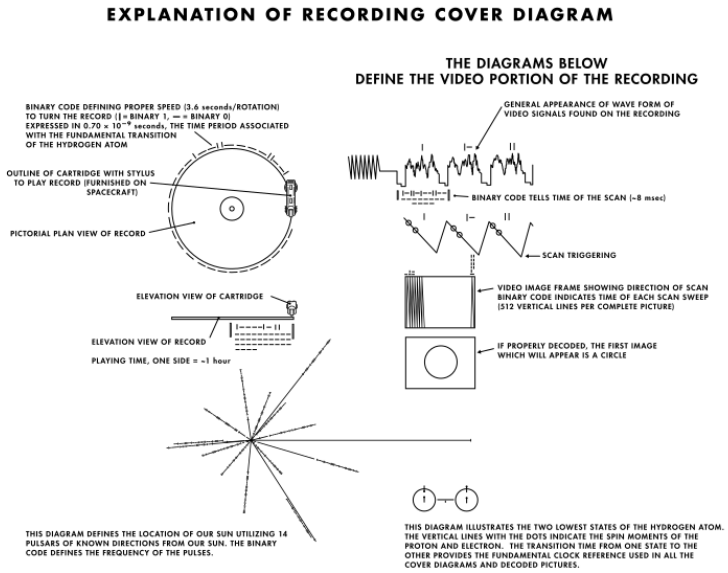
- informática
- datos
- información
- conocimiento
- sabiduría

¿qué es eso?

¿datos, información, conocimiento, sabiduría?

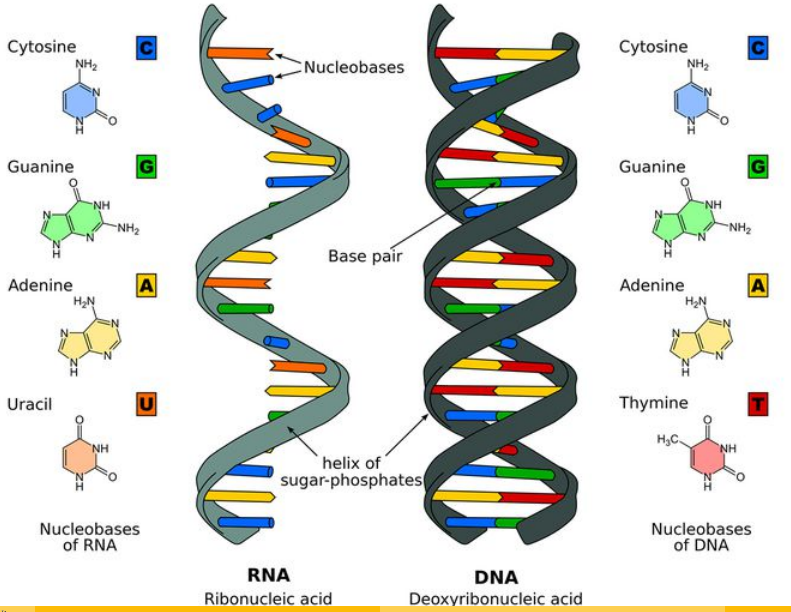


¿datos, información, conocimiento, sabiduría?



¿a ver si *alguien* algún día entiende?

¿nuestros datos?



imágenes

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.

informática (currícula según ACM)

ACM *association for computing machinery*

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

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, 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

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

codificación

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

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}

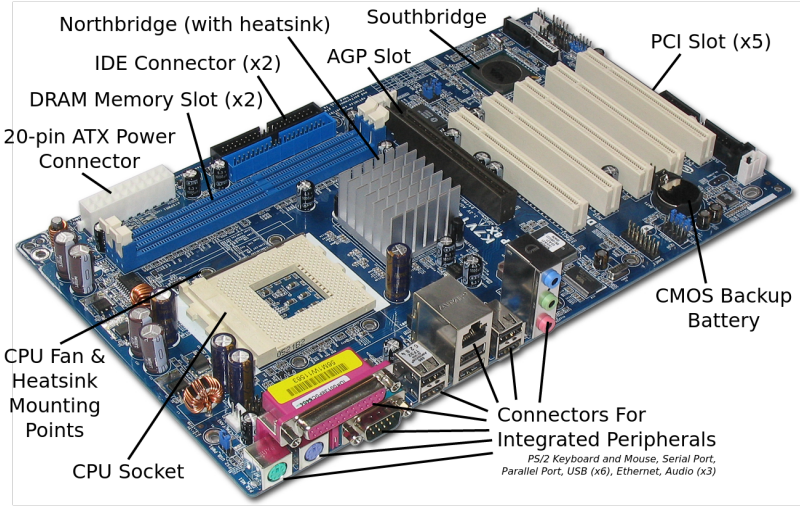
lectura de una vida

- 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)
- 10 GB de texto se comprime a unos 20 %
- pues una memoria de USB de 2GB llega para toda la vida
- 100 m de libros en estantería es el tope.
- se transmite en una red moderna entre 15 seg. y 5 min.

ordenador o computadora

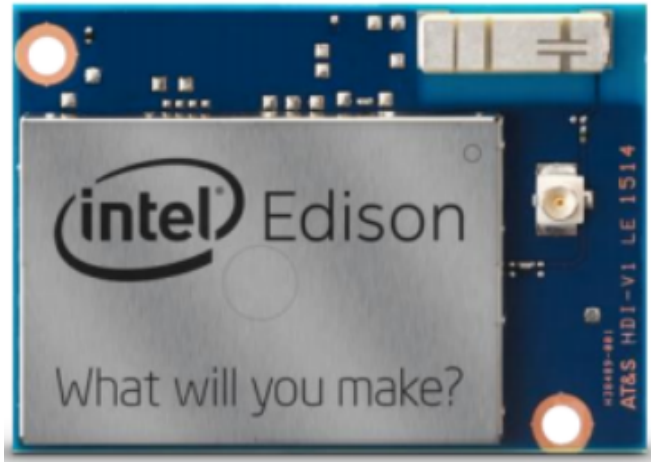
- Es una máquina (o dispositivo) digital
- con medio de entrada de datos
- con medio de almacenamiento de datos (registros y memoria)
- con unidad(es) de procesamiento
- con medio de salida de datos.
- El procesamiento transforma datos *por un lado* en datos *por otro lado*.
- Los nuevos datos cambian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

¿ordenador o computadora?



¡no!

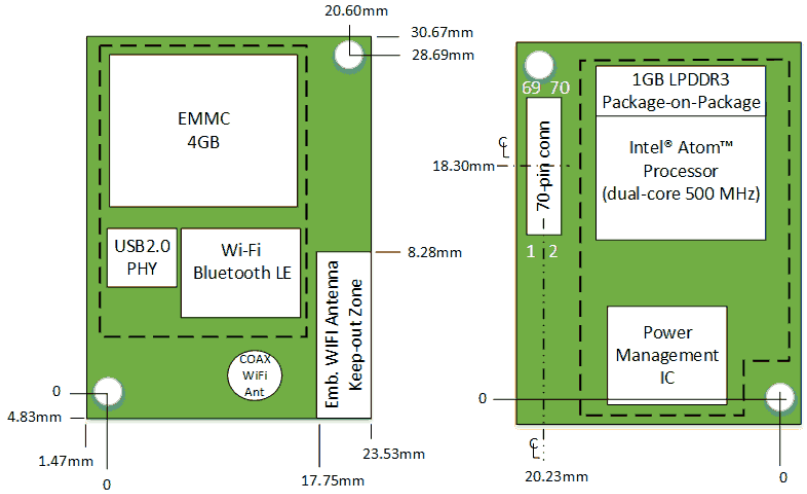
¿ordenador o computadora?



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

¡sí!

¿ordenador o computadora?



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)

unidad de procesamiento central (CPU): inventos

- Charles Babbage (1837): primer concepto de ordenador general
- Alan Turing (1936): *universal Turing machine*
  (modelo matemático de la computabilidad)
- Konrad Zuse (1941): primer ordenador general construido
- Mark I (1944): architectura Harvard
- John von Neumann (1945): architectura Princeton

hay tantos dibujos de CPU's que hay libros sobre el tema,
de todas 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

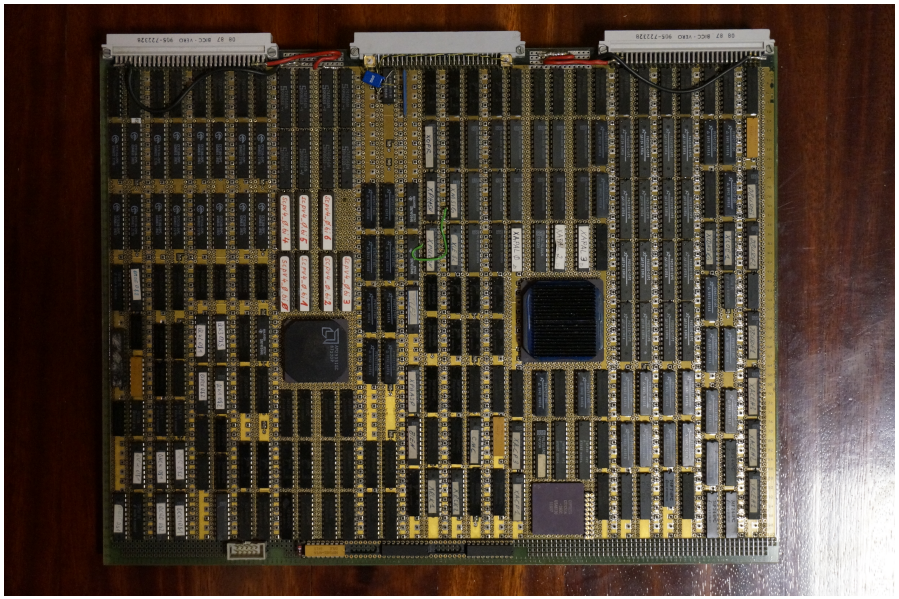
- reloj (con fases) que determina cuando se cambia el estado
- unidad lógica y aritmética (ALU) que calcula
- registros donde se almacena el estado
- buses de conexión que pasan datos de un lado a otro(s)
- unidad de control (microprograma) que simplifica el control y permite solapamiento (mejor eficiencia)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

un ejemplo simple de una CPU, pero no tanto...

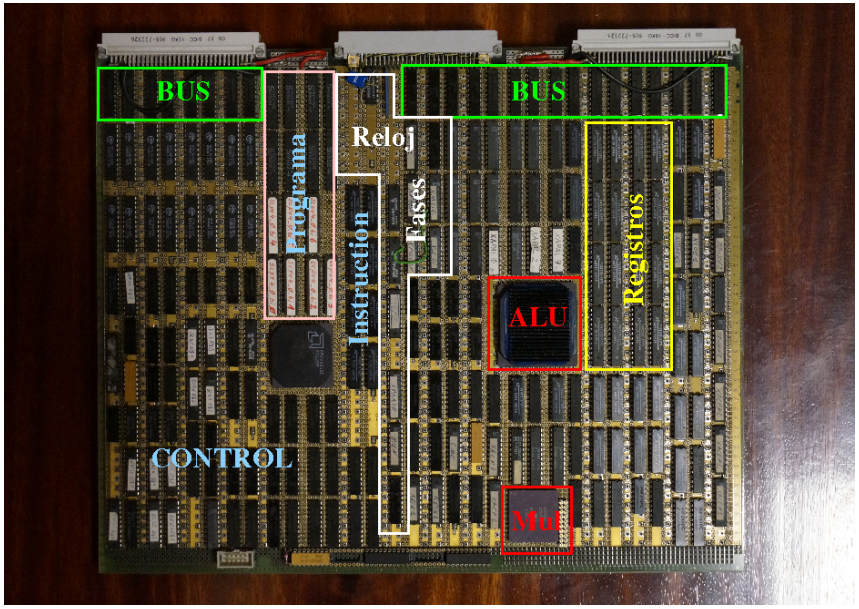
construimos una CPU

- 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

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)

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

video en la red sobre el funcionamiento de una CPU

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

tipos de memoria en un ordenador

- registros de la CPU (orden de kB)
- cachés de diferente tipo (datos, instrucciones, comunicaciones) y niveles (L1, L2, L3) (orden de MB)
- memoria principal (orden de GB)
- memoria externa, discos (orden de TB)
- memoria en red, o en la nube (orden de ZB)
- actualmente se genera unos 16 ZB anualmente

luego existen memorias especiales:
NVRAM, BIOS, Firmware-ROM, memoria gráfica

unidad artimético-lógico (ALU)

- circuitos que realizan las operaciones aritméticas y lógicas
- construimos una ALU que suma dos enteros (positivos) de 3 bits para entender la base
- ALU contienen: sumadores, multiplicadores, divisores, operaciones sobre bits (lógicos AND, OR, EXOR, rotaciones, etc.)
- trabajan con diferentes codificaciones: números enteros, con o sin signo, números flotantes, y combinaciones más complejas (MMX, SSE, etc.)
- estan conectados a registros y buses
- estan 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).

sistema informática

- 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

redes de ordenadores y la nube

- 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

software

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

Existe software que a su vez genera software.

codificación y desarrollo de software

- binario: se escribe directamente instrucciones en bits y bytes
- ensemblador: se escribe instrucciones legible por un humano
- lenguaje de programación: se escribe algoritmos en un lenguaje formal (legible por un 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)
- cambios no compatibles 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 2.7.15 y 3.7.0 (usamos 3.5)

paradigmas de programación

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

Orientado a objetos: 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 mensajes. Ejemplos: C++, Java ...

Funcional: programación en base a definición de funciones, que se invocan de manera recursiva. Ejemplo: Lisp, Haskell, Camel ...

Lógica: programación basada en representación del conocimiento y lógica de predicados. Ejemplo: Prolog

Veremos solamente el procedural, quizá pinceladas de objetos.

Python

- www.python.org
- instalado en los equipos de prácticas
- interface de desarrollo simple (IDLE)
- versiones 2.7.xx y 3.7.xx (usamos 3.5)
- muy completo, muchas librerías
- hasta cierto punto intuitivo
- echamos un vistazo... (más en prácticas)

algoritmos

- secuencia finita (*receta*) de pasos (o instrucciones) bien definidos para lograr una tarea
- los pasos en principio deben ser ejecutables por un humano
- en mi opinión: 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
- ejemplo: ordenación de números
- algoritmos se caracterizan por su complejidad tanto en tiempo necesario (medido en instrucciones o pasos) como en memoria usada (tamaño de las configuraciones intermedias)

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)
- nota: los conceptos basicamente permiten calcular las mismas funciones computables (si se requiere reproducibilidad)
- algoritmos secuenciales (los vemos)
- algoritmos paralelos (no los vemos)

ejemplo de algoritmo

ordenamos números (o palabras)

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

ejemplo de algoritmo

Calcular π mediante un algoritmo Monte Carlo:

```
# pi with monte carlo randomized algorithm

import random

hits=0.0
tries=0
while tries<1000000:
  x=random.uniform(-1.0,1.0)
  y=random.uniform(-1.0,1.0)
  if x*x+y*y<=1.0:
    hits+=1.0
  tries+=1

print("pi= ",4.0*hits/tries)
```

representación de números en diferentes sistemas

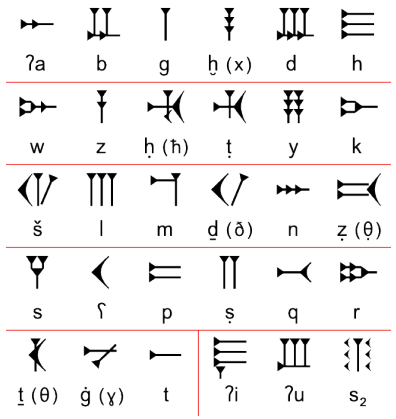
palos	|	||	|||	||||	|||||	||||||	|||||||	||||||||	···
romano	I	II	III	IV	V	VI	VII	VIII	···
árabe	١	٢	٣	٤	٥	٦	٧	٨	···
maya	| · |	| : |	| ⁝ |	| ⁞ |	| I |	| ·I |	| :I |	| ⁞I |	···
decimal	1	2	3	4	5	6	7	8	···
binario	1	10	11	100	101	110	111	1000	···

- el invento del zero fue un gran logro
  (siglo 9, en la India, también los Maya)
- 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 la ciudad Ugarit (hoy Siria), 1500 AC
- símbolos para fonémas ya se inventarón antes (posiblemente en egipto)

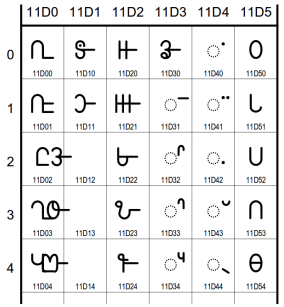


codificación de un alfabeto o abecedario

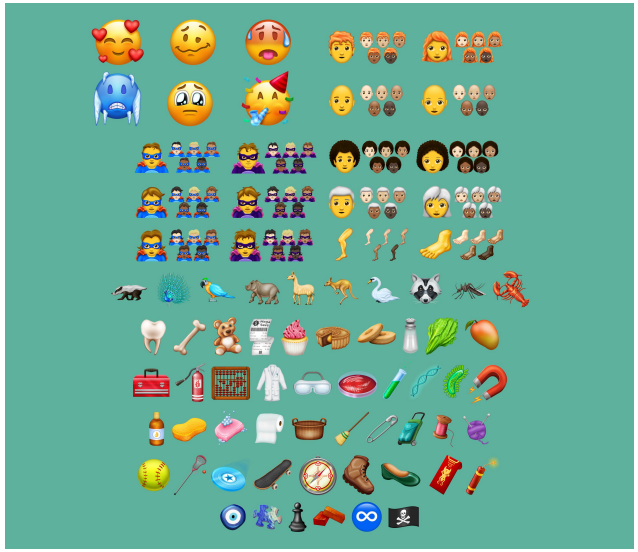
en el ordenador recomiendo usar solamente los siguientes *alfabetos*, más bien codificaciones de letras/dígitos/símboles/caracteres:

- ASCII, 7 bits en un byte, 127 caracteres posibles
- UTF-8, 1 hasta 4 bytes, 1.112.064 caracteres posibles
- Unicode, 4 bytes, en principio 4.294.967.296 caracteres posibles

Masaram Gondi caracteres (añadido al Unicode en junio 2017)



emojis disponibles en Unicode en junio 2018



La vuelta a los tiempos de antes del invento del alfabeto...?!?

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 (minúsculas y mayúsculas), dígitos, y el `'_'` que no comienzan con un dígito de longitud arbitraria
- 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...)

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 puede realizar con la variable
- tipos comunes en muchos lenguajes son:
  `int`, `float`, `boolean`
- existen tipos de datos, incluso formas de construirlos dinámicamente, más complejos (veremos más adelante)

valor de una variable

- el valor de una variable es la interpretación de su representación en bits
- 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

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*)

ejemplos de uso de variables en C

un programa en el lenguaje de programación C

```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);
}
```


ejemplos de uso de variables en C

el programa en lenguaje del procesador (ensemblador):

```
        .file   "age.c"
        .text
        .section        .rodata.str1.1,"aMS",@progbits,1
.LC0:
        .string "My age is %d.\n"
        .section        .text.startup,"ax",@progbits
        .p2align 4,,15
        .globl  main
        .type   main, @function
main:
.LFB23:
        .cfi_startproc
        leaq    .LC0(%rip), %rsi
        subq    $8, %rsp
        .cfi_def_cfa_offset 16
        movl    $25, %edx
        movl    $1, %edi
        xorl    %eax, %eax
        call    __printf_chk@PLT
        xorl    %eax, %eax
        addq    $8, %rsp
        .cfi_def_cfa_offset 8
        ret
        .cfi_endproc
.LFE23:
        .size   main, .-main
        .ident  "GCC: (Ubuntu 7.3.0-16ubuntu3) 7.3.0"
        .section        .note.GNU-stack,"",@progbits
```

ejemplos de uso de variables en C++

un programa en el lenguaje de programación C++

```cpp
#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

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


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 usa, por ejemplo, doxygen (o javadoc, u otra herramienta buena) para generar automáticamente la documentación del código.

ejemplos de comentarios

```python
# 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

precedencia de operadores

```
15 + 59 * 75 / 9 < 2 ** 3 ** 2 and (15 + 59) * 75 % n == 1
#                          ↓
15 + 59 * 75 / 9 < 2 **   9    and (15 + 59) * 75 % n == 1
#                     ↓
15 + 59 * 75 / 9 < 512         and (15 + 59) * 75 % n == 1
#       ↓
15 +  4425   / 9 < 512         and (15 + 59) * 75 % n == 1
#             ↓
15 +        491  < 512         and (15 + 59) * 75 % n == 1
#                                      ↓
15 +        491  < 512         and    74     * 75 % n == 1
#                                           ↓
15 +        491  < 512         and          5550  % n == 1
#                                                 ↓
15 +        491  < 512         and          5550  % 2 == 1
#                                                ↓
15 +        491  < 512         and                0   == 1
#  ↓
  506           < 512         and                0   == 1
#               ↓
              True           and                0   == 1
#                                                   ↓
              True           and                  False
#                             ↓
                            False
```


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)
- 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.

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: procedemientos (*procedures*), métodos (*methods*) funciones (*functions*)

programa con bucle `while`

```
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...!!")
```


bucle `for` con salida `break`

```
for i in range(5):
    print("Value: ",i)
    if (i==4):
        break
else:
    # This line is never executed!
    print("Terminated normally!")
```

subrutinas en Python

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

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

print(f(3,1,2,3))
```


argumentos: por orden

```
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?")
```


argumentos: por defecto

- A los últimos parámetros se pueden asignar un valor por defecto.
- En general se puede 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(h,*values):
  print("hola ",h)
  sum=0.0
  for value in values:
    sum+=value
  return sum/len(values)

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

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 numbers are equal"

print(maximum(1,23))
print(maximum(5,2))
print(maximum(7,7))
print(maximum(maximum(5,2),maximum(7,7)))
```


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

bubble sort en Python

las funciones de ayuda:

```
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]
```


bubble sort en Python

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(1001),1001)
#v = [ 8,2,4,3,6,1,7,5 ]
print(v)
BubbleSort(v)
print(v)
```


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 v[i]<=v[j] ):
            w[k] = v[i]
            i = i+1
        else:
            w[k] = v[j]
            j = j+1

def MergeSort(w):
    for s in [
      2**i for i in range(int.bit_length(len(w)))
    ]:
        v = w[:]
        for L in range(0,len(w),2*s):
            Merge(v,w,L,L+s,min(L+2*s,len(w)))
```


programa principal para *merge sort* en Python

```
#import random
#v = random.sample(range(1001),1001)
#v = [ 8,2,4,3,6,1,7,5 ]
#v = [ 'd','f','a','z','c' ]
v = [ "This","is","a","short","sentence" ]
print(v)
MergeSort(v)
print(v)
```

estructuras de datos

- son datos (del mismo o de diferentes tipos)
- en conjunto con funciones/métodos/operadores
- que permiten
  - el acceso
  - la modificación
  - 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
- árboles
- grafos

disponibilidad de estructuras de datos

- algunos lenguajes ya proporcionan ciertos 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: tuplas, listas, conjuntos, diccionarios

estructuras de datos en Python: listas

- 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'` modificación de 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!

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 los agrupa 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)

conjuntos: ejemplos de acceso, modificación y propiedades

- 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'` modificación o añadir de 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!

quines en Python (una cosa curiosa)

- 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: teorema de punto fijo de la informática.
- ```
  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."
  ```


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 for comas
- 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.

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:>3}: {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("*.py")

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


un ejemplo de cálculo (proceso)

- 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:
  - son triángulos que forman una malla cerrada
  - todos los triángulos están definidos con orientación idéntica
  - la superficie no se interseca a si misma
- Comprobar las condiciones es la tarea más compleja, el mero cálculo es fácil...

un ejemplo de cálculo (código)

```
import csv
import sys

def Det(a,b,c): # volumen del espato 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(fn):
  f=open(fn,"rt")
  reader=csv.reader(f,delimiter=" ",skipinitialspace=True)
  v=0
  vol=0.0
  for row in reader:
    if len(row):
      if row[0]=="vertex":
        if v==0:
          a=[float(row[1]),float(row[2]),float(row[3])]
        elif v==1:
          b=[float(row[1]),float(row[2]),float(row[3])]
        else:
          c=[float(row[1]),float(row[2]),float(row[3])]
        v+=1
      if v==3:
        vol+=Det(a,b,c)
        v=0
  f.close()
  print("vol ",vol/6) # dividimos entre 6 porque son pirámides no espatos

print("simple volume calculator, (c) 2018 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))
```


Ordenación recursiva

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

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)
```

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 librerías, 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

```
# 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.

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.

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

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.

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

Aplicaciones específicas

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

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]

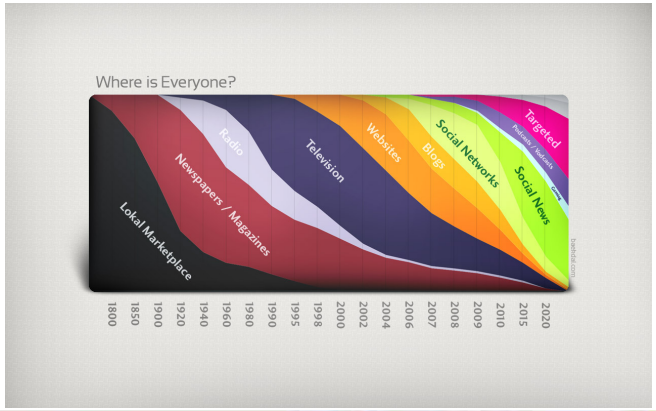
  Office 365 Personal: On one PC/Mac and one tablet, for use only by the licensed subscriber. [1]

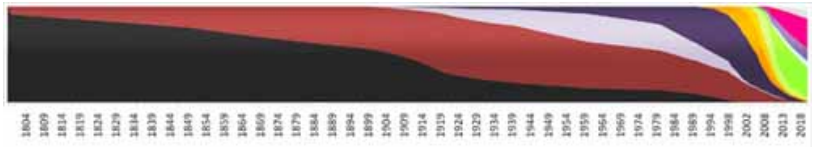
- software libre
  https://choosealicense.com/licenses/
- software de servicio

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.)

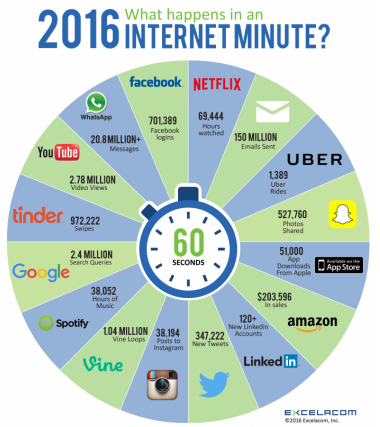
Sociedad digital





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

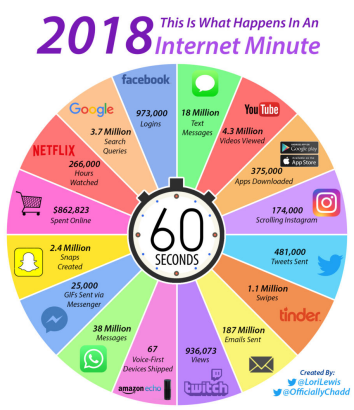
un minuto del internet



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

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

un minuto del internet



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

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

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.

un simple ejemplo...

HAY que tener cuidado:

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

... y ejecuta el programa como administrador (root) :-)

Excepciones

- El uso de excepciones es una técnica en la programación para reaccionar de forma estructurada a situaciones no esperadas durante la ejecución del programa.
- Ejemplos son: división por zero, no-existencia de un fichero, errores de lectura o escritura, interrupciones por el sistema operativo etc.
- El código que se ejecuta si tal excepción ocurre, permite una alternativa al flujo de control del programa normal.

Excepciones en Python

Ejemplos de excepciones estándar en Python son:

- `ImportError`: fallo en sentencia `import`
- `IndexError`: un índice de una estructura fuera de rango
- `NameError`: variable desconocida
- `SyntaxError`: problema con la sintaxis del código
- `TypeError`: una función es invocada con un argumento de tipo incorrecto
- `ValueError`: una función es invocada con tipo correcto, pero valor incorrecto
- lista más completa: `https://docs.python.org/3/library/exceptions.html`
- Otros paquetes y el propio programa pueden definir sus propias excepciones.

Ejemplo de uso de excepciones en Python

```
while True:
  n = float(input("enter nominator: "))
  d = float(input("enter denominator: "))
  try:
    print(" division result: ", n/d)
  except ZeroDivisionError:
    print("you cannot divide by zero")
  if input("another? [Y/n] ")[0].lower() == 'n':
    break
```

Este programa capta solamente algunas de las excepciones posibles.

Más sobre excepciones en Python

- Se puede usar más de un bloque de `except`.
- Se puede realizar un bloque de excepcion con varias causas, p.ej., `except(RangeError,ValueError)`
- Se puede captar todas las excepciones simplemente usando `except:` (sin especificar ninguna en concreto)
- Se puede anidar excepciones.
- Se puede añadir un bloque `else` que se ejecuta cuando no se lanza ninguna excepción.
- Se puede añadir un bloque `finally` que se ejecuta siempre.

Otro ejemplo de uso de excepciones en Python

```
try:
  while True:
    try:
      n = float(input("enter nominator: "))
      d = float(input("enter denominator: "))
      print(" division result: ", n/d)
    except ZeroDivisionError:
      print("you cannot divide by zero")
    except ValueError:
      print("please write floats...")
    if input("another? [Y/n] ")[0].lower() == 'n':
      break
except:
  print("catching any exception");
```

No es buena idea captar (e ignorar) todas las excepciones!

y otro ejemplo ...

```
try:
  while True:
    try:
      n = float(input("enter nominator: "))
      d = float(input("enter denominator: "))
      print(" division result: ", n/d)
    except ZeroDivisionError:
      print("you cannot divide by zero")
    except ValueError:
      print("please write floats...")
    else:
      print("seemed that division was possible")
    finally:
      if input("another? [Y/n] ")[0].lower() == 'n':
        break
except:
  print("catching any exception, what was wrong ???");
```

y más sobre excepciones en Python

- Se puede pasar de una excepción con `pass`.
- Se puede lanzar o re-lanzar excepciones con `raise`.
- Se tiene acceso al texto de la excepción con `as`.
- Se pueden definir excepciones propias, pero para eso necesitamos el concepto de clases (pero no habrá tiempo...).

Otro ejemplo de uso de excepciones en Python

```
try:
  while True:
    try:
      n = float(input("enter nominator: "))
      d = float(input("enter denominator: "))
      if d==1.0:
        raise ValueError("I don't like to divide by 1")
      print(" division result: ", n/d)
    except ZeroDivisionError:
      print("you cannot divide by zero")
    else:
      if input("another? [Y/n] ")[0].lower() == 'n':
        break
except ValueError as val:
  print("oh oh...",val)
```

mira el manual de Python para más información...

Excepciones de ejecución

- Recomendación: Se usan excepciones solamente para casos excepcionales, es decir, si pasa algo no esperado.
- Excepciones en programas (especialmente en programas concurrentes) se convierten rápidamente en pesadillas.
- Mira Safe (Disciplined) Exception Handling principle con sus dos posibilidades de acción
  - fallo: re-establecer una invariante necesaria (y basicamente terminar el programa)
  - re-intentar: re-hacer la operación con (quizá) cierta modificación

Hay quien que dice: cuanto menos excepciones mejor :-)

Paquetes interesantes

Python ofrece muchísimos paquetes, unos interesantes son

- `numpy`: rutinas numéricas
- `scipy`: rutinas científicas
- `matplotlib`: rutinas para visualización

en conjunto cubren gran parte de lo que ofrece Matlab.
Miramos unos ejemplos, hay muchos ejemplos, tutoriales, y casos de uso en la red.
Experimenta!

Solucionar un sistema de ecuaciones

```
import numpy as np

try:
  A=np.array([[0,0,0],[0,0,0],[0,0,0]],dtype=np.float64)

  B=np.array([10,11,12],dtype=np.float64)

  print("let us solve:\n",A,"\ntimes vector X =\n",B)

  Ainv=np.linalg.inv(A)
  X=np.dot(Ainv,B.T)

  print("The solutions are: ")
  print(" x = {:+.3f}".format(X[0]))
  print(" y = {:+.3f}".format(X[1]))
  print(" z = {:+.3f}".format(X[2]))
except np.linalg.linalg.LinAlgError:
  print("ha: go studying algebra...!!!")
```


Visualizar grafos de funciones

```
import numpy as np
import matplotlib.pyplot as plt

X  =np.linspace(-np.pi,np.pi,256,endpoint=True)
C,S=np.cos(X),np.sin(X)

plt.plot(X,C)
plt.plot(X,S)

plt.show()
```


Visualizar el conjunto de Mandelbrot

```
import numpy as np
import matplotlib.pyplot as plt

ITERATIONS=100
DENSITY   =1000 # warning: execution speed

# decreases with square of DENSITY
x_min,x_max = -2,1
y_min,y_max = -1.5,1.5
x,y=np.meshgrid(
  np.linspace(x_min,x_max,DENSITY),
  np.linspace(y_min,y_max,DENSITY)
)
c=x+1j*y # complex grid
z=c.copy()
fractal=np.zeros(z.shape,dtype=np.uint8)+255

for n in range(ITERATIONS):
  print("Iteration %d" % n)
  z*=z
  z+=c
  mask=(fractal==255) & (abs(z)>10)
  fractal[mask]=254*n/float(ITERATIONS)

plt.imshow(
  np.log(fractal),cmap=plt.cm.hot,
  extent=(x_min,x_max,y_min,y_max)
)
plt.title('Mandelbrot Set')
plt.xlabel('Re(z)')
plt.ylabel('Im(z)')
plt.show()
```