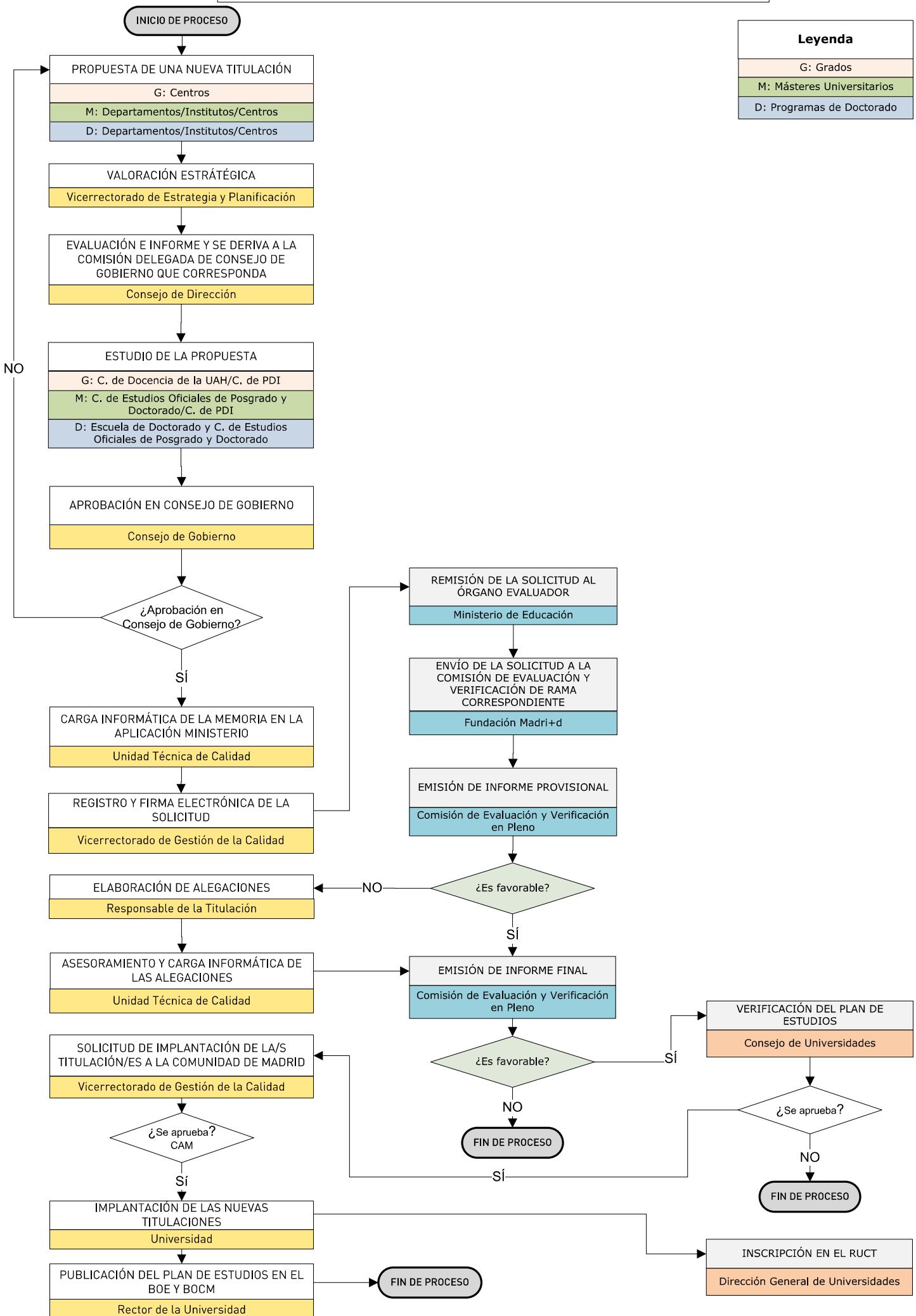


DESCRIPCIÓN DEL PROCESO DE VERIFICACIÓN DE TITULACIONES



Resumen de esquemas

Rosalía Peña



Universidad
de Alcalá



Esquema Componer

- Resumen: una secuencia de condicionales complejos se puede simplificar agrupando adecuadamente los datos involucrados en las condiciones.
- Ejemplo: con fechaPlana=anno*10000+mes*100+dia
mayor de dos fechas , estación de fecha desarrollado en capítulo 5.

Un alumno de la universidad tiene una titulación, curso, grupo, expediente, nombre, apellidos, fecha de nacimiento, asignaturas matriculado,...Ordena a los alumnos de la universidad por titulación, dentro de titulación por curso, y posteriormente por grupo. Dentro de un mismo grupo irán por orden de expediente

Esquema Descomponer:

- Resumen: al partir un dato en sus partes se puede generalizar el algoritmo de lo que se desea conseguir
- Ejemplo: nombre de los números entre el 30 y el 999, desarrollado en capítulo 5.

Las 40 cartas de una baraja se han esquematizado con los números del 1 al 40, colocando primero las 10 de oros, en orden ascendente de valor, copas, espadas y bastos sucesivamente. Haz un subprograma que devuelva el valor y el palo como 2 enteros.

Esquema agrupar y seleccionar:

- Resumen: creamos una secuencia con los valores posibles, de la que seleccionamos por posición, en vez de codificar una serie de condicionales anidados en una alternativa múltiple. El código desarrollado así es menos complejo y más fácil de mantener
- Ejemplo: día de la semana, en capítulo 5 o último día del mes, desarrollado en capítulo 8.

Las 40 cartas de una baraja se han esquematizado con los números del 1 al 40, colocando primero las 10 de oros, en orden ascendente de valor, copas, espadas y bastos sucesivamente. Reusando el subprograma anterior, haz un subprograma que devuelva el valor y el nombre de la carta, por ejemplo 10 es el rey de oros, y 11 el as de copas.

Esquema Contador

- Resumen: para saber cuantos elementos de una secuencia satisfacen una condición. Antes de entrar en el bucle se inicializa el contador. Se recorre la secuencia con un bucle. Dentro del bucle se incrementa el contador para cada elemento que satisfaga lo que estamos contando
- Ejemplo: número de preguntas acertadas en el examen test, desarrollado en capítulo 6.

En el juego de cartas de “la escoba” interesa saber ¿cuántos oros, cuántos sietes, cuantas cartas hay en un mazo?

Reflexiona sobre la ventaja de contar cada concepto en un bucle o agrupar todas las cuentas en un solo bucle. (ahora solo céntrate en esto).

Esquema Acumulador (o Totalizador)

- Resumen: es una variación del anterior, antes de entrar en el bucle se inicializa el acumulador. Dentro del cuerpo del bucle $\text{acumulador}=\text{acumulador}+\text{elemento}$.
- Ejemplo: dinero en monedero, desarrollado capítulo 6.

En el juego de cartas de “la escoba” pueden jugar entre 2 y 10 jugadores. Tras barajar, se colocan cuatro cartas sobre el tapete y 3 a cada jugador. Por turnos, si con 1 carta de su mano y las que pude de la mesa suma 15, se las lleva a su mazo, si no, deja una de su mano. Si con ello se lleva todas las cartas de la mesa hace “una escoba”, y se le da un punto. Esto se puede ir contando mientras dura el juego. Cuando ya no quedan cartas en la baraja, el jugador que tenga más cartas, más sietes, más oros en su mazo recibe un punto extra. Si dos jugadores coinciden en el valor máximo, no se lleva ninguno el punto correspondiente. Diseña los tipos necesarios para jugar a la escoba.

Haz un subprograma que recibiendo el mazo de los jugadores, y sus escobas devuelva los puntos de cada jugador.

Puedes usar `max(secuencia) → valor` y `secuencia.count() → valor`

Esquema Bucle con condición múltiple y discriminación a la salida

- Resumen: en la condición del bucle coordinamos todos los criterios de salida del bucle, despreocupándonos de ellos en el cuerpo del bucle. Solo a la salida determinamos por cuál de las causas finalizó el bucle.
- Ejemplo, posición de un elemento en una secuencia, desarrollado en el capítulo 7.

Si alguno de los jugadores ha alcanzado 10 o más puntos, se termina el juego.

Esquema Centinela

- Resumen: Añadir el elemento buscado al final de la secuencia evita controlar en cada pasada no salgamos de la lista. Al salir del bucle, si el elemento encontrado es el último, implica que no estaba antes de añadirlo. Sobrecoste de la inserción y posiblemente la eliminación de un elemento (dependiendo del tipo de datos empleado), pero ahorra evaluaciones booleanas, por ello es conveniente para búsquedas en secuencias largas, principalmente si residen en memoria principal.
- Ejemplo desarrollado en el capítulo 7.

Si alguno de los jugadores ha alcanzado 10 o más puntos, se termina el juego.

Pedir dato al usuario

- Resumen: el responsable de que el dato satisfaga las condiciones del mundo real es siempre el subprograma que pide el dato. En un bucle (1-n) pedir el dato *hasta* que cumpla la semántica impuesta por el mundo real. Además, si la semántica del dato es compleja, conviene subprogramar “esDato”.
- Ejemplo: enteroPedido, DNIPedido y contrasennaPedida. Desarrollados en el capítulo 7.

esFecha(f) devuelve True si f es una fecha válida (no te lo piden). Se desea pedir la fecha de nacimiento de un empleado para darlo de alta en nómina. La legislación española impide que trabajen los menores de 16 años. tFecha es un namedtuples de dia,mes, anno. La función today de la biblioteca datetetime devuelve la fecha del sistema.
¿subprogramas edad?

Esquema Menú y Aplicaciones de gestión

- Resumen:
 - subprograma **pintaMenú**, aunque conceptualmente sencillo es propenso a sufrir modificaciones.
 - leeOpcion** pide un valor *hasta* que sea una de las opciones ofrecidas en el menú. ¿**pideEntero**? **alternativa múltiple** distribuye la tarea al subprograma que resuelve cada opción.
- Para **menú continuo**, meter todo lo anterior en un bucle (1-n), añadiendo la opción salir, de modo que la aplicación permite seleccionar, sucesivas opciones, volviendo a mostrar el menú, hasta que el usuario seleccione salir
- Ejemplo: Aplicación de combinatoria en el capítulo 7.

Construye el cuerpo ejecutivo de la aplicación de gestión de empleados de la empresa (nombre, apellidos, DNI, fNacim, categoría, sueldo) ¿Qué acciones puede haber con cada empleado?

NORMAS: No tenga ninguna hoja sobre su mesa sin su nombre. No salga del aula en los primeros 15 minutos después de entregado el enunciado. No use bolígrafo verde. No use más que las funciones presentes en el núcleo de Python. No haga preguntas individuales al profesor. Lea detenidamente todo el ejercicio antes de contestar y organice su tiempo.

ENUNCIADO

- 1) (1) Explique en una frase que hace el siguiente código (no me lea las instrucciones, es decir, no mencione for, if, bucle, alternativa, 33, inverso, global,...)

```
def par(inverso):
    inverso =GLOBAL
    for local in range (1,GLOBAL):
        if inverso%local ==0: print(local)
GLOBAL= 33
print (par(GLOBAL))
```

- 2) (1) Que habría que modificar en el programa anterior para que tarde aproximadamente la mitad de lo que tarda, haciendo lo mismo. ¿de qué tipo es este error: sintaxis, ejecución, lógica, estilo, o pensamiento computacional?

- 3) (1) El programa anterior tiene varios errores de estilo, y algún otro error computacional. Escriba una versión de este código eliminándolos.

- 4) (1) Los estudios estadísticos sobre la violencia de género disponen de datos de sentencias que implican órdenes de alejamiento de diferentes comunidades autónomas durante los últimos 5 años. Se desea mostrar una tabla como en la figura. Diseñe las estructuras necesarias

	Madrid	Burgos	Galicia
2017	1	6	10
2016	2	7	0
2015	3	8	10
2014	4	9	0
2013	5		
Total	15

Media/C	...		=ΣComAutónoma[i][j]/(num de años) variando j
Media global	...		=ΣCCAA[i][i]/(num de años*num comunidades). Variando i y J

Nota: la media de una Comunidad es la suma de las sentencias habidas en los sucesivos años. La media global es la suma de todas las sentencias dividido por el número total de celdas en la tabla.

5) (1 punto) ¿Qué hace el siguiente código?

```
L1=[]
for i in range (2,10,3):
    L1+=[i]
L2=L1+[0]
L3=L1
L1[2]=11
L3[0]=10
L1+=[4,8]*L2[-1]+[6]
print(L1,L2,L3[::-1])
print(L3[4])
```

6) (1 punto) Indique el ámbito de las variables vigentes, y visibles cuando se acaba de ejecutar la línea 3 ¿de qué variable se está haciendo un uso global?

1	temp=9 #es variable. se leerá por teclado en futuro
2	def estornudo (n,dado):
3	perro=n+dado-1
4	return perro//dado==n//dado
5	
6	def fiebre (n):
7	try:
8	temp=str(input("introduce temp:"))
9	gripe=int(temp)
10	if estornudo(gripe,n) == True:
11	print(temp,"sobrevive.")
12	if estornudo (gripe,n) == False:
13	print(gripe,"muere")
14	except ValueError:
15	print("temperatura no es válida.")
16	import math

17	print(fiebre(int(math.sqrt(temp))))
----	-------------------------------------

- 1) (0,5 punto*4) En el código hay errores conceptuales importantes (de al menos 4 tipos diferentes) reescribe las líneas correspondientes indicando que efecto tiene sobre la calidad del soft cada uno de los errores.

1	temp=9 #es variable. se leerá por teclado en futuro
2	def estornudo (n,dado):
3	perro=n+dado-1
4	return perro//dato==n//dato
5	
6	def fiebre (n):
7	try:
8	temp=str(input("introduce temp:"))
9	gripe=int(temp)
10	if estornudo(gripe,n) == True:
11	print(temp,"sobrevive.")
12	if estornudo (gripe,n) == False:
13	print(gripe,"muere")
14	except ValueError:
15	print("temperatura no es válida.")
16	import math
17	print(fiebre(int(math.sqrt(temp))))

PROPUESTA DE SOLUCION

- 1) (1) Explique en una frase que hace el siguiente código (no me lea las instrucciones, es decir, no mencione for, if....)

```
def par(inverso):
    inverso =GLOBAL
    for local in range (1,GLOBAL):
        if inverso%local ==0: print(local)
GLOBAL= 33
print(par(GLOBAL))
```

Muestra los divisores
del argumento+None

- 2) (1) ¿Que habría que modificar en el programa anterior para que tarde aproximadamente la mitad de lo que tarda, haciendo lo mismo? ¿de qué tipo es este error: sintaxis, ejecución, lógica, estilo, o pensamiento computacional?

(0,65) El divisor mayor de un número es menor o igual que su media, → se puede evitar la mitad de las pasadas del bucle

(0,35) Compila → **sintaxis**, ejecuta → **ejecución**. Consigue los resultados esperados → **lógica**. Tiene muchos errores de estilo, pero no es lo que aquí preguntan. Me están hablando de eficiencia → **error de pensamiento computacional**

El enunciado pide que reduzca el tiempo a la mitad. Evitar una instrucción de asignación, no puede tener este efecto, habiendo un for

- 3) (1) El programa anterior tiene varios errores de estilo, y algún otro error computacional. Escriba una versión de este código eliminándolos.

```
def par(inverso):          #error nombre identificadores
                           #falta documentación
    inverso =GLOBAL       #modifica arg entrada antes de consultarla
                           # acceso global a variable
    for local in range (1,GLOBAL): # acceso global a variable
        if inverso%local ==0: print(local)
#PROBADOR
GLOBAL= 33                 #error nombre identificadores
print(par(GLOBAL))          #par es un procedimiento. Se está
                            #llamando como función

def mostrarDivisores (num):
    """int-->nada
OBJ Mostrar divisores de num
```

```

PRE: num >0"""
    for n in range (1,num//2+1):
        if num % n ==0: print(n)
#PROBADOR
n= 33
mostrarDivisores (n)
mostrarDivisores (4)

```

- 4) (1) Los estudios estadísticos sobre la violencia de género disponen de datos de sentencias que implican órdenes de alejamiento de diferentes comunidades autónomas durante los últimos 5 años. Se desea mostrar una tabla como en la figura. Diseñe las estructuras necesarias

	Madrid	Burgos	Galicia
2017	1	6	10
2016	2	7	0
2015	3	8	10
2014	4	9	0
2013	5		
Total	15
Media/C	...		=ΣComAutónoma[i][j]/(num de años) variando j
Media global			=ΣCCAA[i][i]/(num de años*num comunidades). Variando i y J

Nota: la media de una Comunidad es la suma de las sentencias habidas en los sucesivos años. La media global es la suma de todas las sentencias partido por el número total de celdas en la tabla.

- tComunidad = tupla con los datos de cada uno de los cinco años (orden creciente)
- tJurisprudencia = tupla de 17 tuplas tComunidad (Esquema tuplas anidadas, equivalente a array bidireccional en otros lenguajes)
- CCAA es una constante tupla con el nombre de cada una de las 17 comunidades autónomas
- El orden de las comunidades autónomas es el mismo en CCAA que en tJurisprudencia
- tPeriodo= tupla de años (enteros)

Entiendo que es un estudio que está capturando los datos a posteriori, por ello diseño tuplas. Si a captura de datos es simultánea a la explotación, quizás diseñaríamos listas. En cualquier caso, secuencias.

5) (1 punto) ¿Qué hace el siguiente código?

```
L1=[]
for i in range (2,10,3):
    L1+=[i]
L2=L1+[0]
L3=L1
L1[2]=11
L3[0]=10
L1+=[4,8]*L2[-1]+[6] # Advertid que esta expresión es distinta de L1=L1+[4,8]*L2[-1]+[6]
print(L1,L2,L3[::-1])
print(L3[4])
```

El mínimo a contestar es lo que figura en verde. Hemos ido poniendo los pasos para que se pueda seguir el proceso que nos conduce a la solución.

L1=[]	L1=[]
for i in range (2,10,3):	
L1+=[i]	L1=[2,5,8]
L2=L1+[0]	L2=[2,5,8,0]
L3=L1	L3 es L1
L1[2]=11	L1=L3=[2,5,11]
L3[0]=10	L1=L3=[10,5,11]
L1+=[4,8]*L2[-1]+[6]	L1=L3=[10,5,11]+[4,8]*0+[6]=[10,5,11,6]
print(L1,L2,L3[::-1])	[10,5,11,6] [2,5,8,0][6,11,5,10]
print(L3[4])	Fuera de rango

7) (1 punto) Indique el ámbito de las variables vigentes, y visibles cuando se acaba de ejecutar la línea 3 ¿de qué variable se está haciendo un uso global?

1	temp=9 #es variable. se leerá por teclado en futuro
2	def estornudo (n,dado):
3	perro=n+dado-1
4	return perro//dato==n//dato
5	
6	def fiebre (n):
7	try:
8	temp=str(input("introduce temp:"))
9	gripe=int(temp)
10	if estornudo(gripe,n) == True:
11	print(temp,"sobrevive.")
12	if estornudo (gripe,n) == False:
13	print(gripe,"muere")
14	except ValueError:
15	print("temperatura no es válida.")
16	import math
17	print(fiebre(int(math.sqrt(temp))))

PP	fiebre	estornudo	visible
	n	n	estornudo.n
temp	temp		fiebre.temp
	gripe		gripe
		perro	perro
		dado	dado

Hay 7 variables vigentes, dos parejas de ellas comparten nombre, con lo que no es posible acceder a la del ámbito del ancestro más lejano, por tanto, hay 5 variables visibles, y no se hace uso global de ninguna de ellas.

Uso global significa consultar o modificar una variable desde otra pieza de código que no es la que la ha creado. Python no permite modificaciones globales, pero si consultas. Por convenio, cuando construimos un programa que realiza consulta global, lo indicamos expresamente en la PRE.

Una vez más os recomiendo ejecutad, línea a línea, este programa en Python tutor, e viendo cómo se crean y mueren las variables, el “frame”=ámbito al que pertenecen las variables. Inventad una situación en la que haya consulta global a variables.

ERRORES FRECUENTES

No se ha entendido qué es uso global, porque todos los alumnos han identificado una u otra variable, cuando no hay ningún uso global. Se hace USO GLOBAL cuando se consulta (se accede a) una variable en una pieza de código diferente a su ámbito. Es decir, desde un subprograma que no es el que la ha creado. Por ejemplo, si entre la línea 2 y 3 incluyéramos dado=dado+temp+gripe+n, se estaría haciendo uso global de temp y de gripe, pero no de n.

Hay gran confusión en este tema. Parece que las definiciones de ámbito, vigencia y visibilidad se han memorizado, pero no se entienden y, por tanto, no se han sabido aplicar.

En la mayoría de los libros explican el concepto de ámbito partiendo de un ejemplo que tiene solo un pp y un subprograma, en ese caso solo se puede hacer uso global de variables de pp, y por eso a veces simplifican y dicen “las var del pp son globales y las de subp son locales”. Pero lo importante, no es ponerle un adjetivo a las var. No es cómo SON, sino cómo se usan. Si el subprograma “padre” llama al subprograma “hijo” el hijo puede ver las variables de padre (y las del abuelo, que sería el programa principal), y hacer un uso global de ellas.

- 8) (0,5 punto) En el código hay errores conceptuales importantes (de al menos 4 tipos diferentes) reescribe las líneas correspondientes indicando que efecto tiene sobre la calidad del soft cada uno de los errores.

1	temp=9 #es variable. se leerá por teclado en futuro #7,8
2	def estornudo (n,dado): #7,8
3	perro=n+dado-1

4	return perro//dado==n//dado	
5		
6	def fiebre (n):	#6,7,8
7	try:	
8	temp=str(input("introduce temp:"))	#1,10
9	gripe=int(temp)	
10	if estornudo(gripe,n) == True:	#2
11	print(temp,"sobrevive.")	
12	if estornudo (gripe,n) == False:	#2, 3
13	print(gripe,"muere")	
14	except ValueError:	
15	print("temperatura no es válida.")	
16	import math	#5
17	print(fiebre(int(math.sqrt(temp))))	#4

Encuentro los siguientes errores

- 1) El retorno de input es siempre str, por tanto, sobra en la línea 8, uniendo 8 y 9
 $\text{divisor}=(\text{int}(\text{input}(\text{"introduce dividendo: "})))$
 Llamar a una función menos aumenta la eficiencia, disminuye la complejidad. Ahorro memoria de la variable auxiliar → eficiencia.
- 2) Las expresiones booleanas ya tienen un valor de verdad. No hay que compararlas con una constante
 Una operación menor → legibilidad, eficiencia (tiempo de proceso)
- 3) La línea 12 pregunta lo contrario que la 10, por tanto, es simplemente un else.
 Una estructura de control menos baja la complejidad, aumenta legibilidad, menor tiempo de ejecución.
 Las líneas 10 a 14 se pueden reducir a

```
if esDivisible(dividendo, divisor):
print(dividendo, "divisible")
else: print(dividendo, "no divisible")
```

- 4) Fiebre no es una función si no un procedimiento. Por ello, nunca puede ser un argumento de un subprograma y no debería figurar en el print. Debe reescribirse:
`mostrarDivisible(int(sqrt(n)))`

Llamada a un procedimiento menos supone disminución de la complejidad, aumento de la legibilidad, menos consumo de tiempo.

- 5) No es conveniente importar toda la biblioteca cuando solo requiero un recurso.
 Memoria que ocupa, tiempo de traerla, complejidad en la llamada con notación punto disminuye la legibilidad.

Modificar las líneas 16 y 17

```
from math import sqrt,
mostrarDivisible(int(sqrt(n)))
```

allí.

- 6) Como es un procedimiento el nombre debería ser un verbo. Afecta a la legibilidad y por tanto a la mantenibilidad. En el ejercicio 7 se han propuesto nombres adecuados para cada una de las variables.

```
def mostrarDivisible (divisor):
```

- 7) Le falta documentación

- 8) Nombres incorrectos de todos los identificadores

- 9) Posición de las secciones de código

- 10) El try debería englobar solo lo problemático, el único problemático es int(algo_recie_leido)

ERRORES FRECUENTES

-Algunos alumnos afirman que no puede haber gestión de excepciones en subprogramas. No habría ningún motivo para ello

EL código no tiene ni errores de ejecución ni de compilación, de lo contrario la respuesta a ¿Qué hace este código? sería: No compila o Aborta. Tampoco queda claro como contestar a dicha pregunta, si el código tuviera errores de lógica (es decir, no hace lo que debería hacer). EL código hace lo que hace. SI el enunciado no indica lo que debería hacer, es imposible decir que tiene errores de lógica.

Los errores conceptuales (o de pensamiento computacional) se refieren a empleos inadecuados de recursos de programación, mediante los cuales conseguimos eficacia, es decir, que el programa haga lo que queríamos, pero producimos código ineficiente (realiza operaciones innecesarias, o consume más memoria de la imprescindible), ciclomáticamente más complejo de lo necesario (con más instrucciones de control de flujo de

¿por qué ha de haber un float en la línea 8/9???? Os estais dejando engañar por nobres de variables mal puestas

Incluso alumnos que detectan errores argumentan mal la causa.

Pocos alumnos indican cómo afecta a la calidad el error.

Muchos alumnos proponen sustituir la línea 12 por elif. No es así, es solo un else

Por completitud se añade todo el código, con la evolución de las simplificaciones, a partir de aquí no se había pedido.

```
from math import sqrt

def esDivisible (dividendo,divisor):
    """
        # traducción textual
    aux=dividendo+divisor-1
    return aux//divisor==dividendo//divisor
    """
    #elimino paso intermedio
    return (dividendo+divisor-1)//divisor==dividendo//divisor
```

```
'''  
    return dividendo%divisor==0      #y para esto puede que no lo  
    subprogramara  
  
def mostrarDivisible (divisor):  
    try:  
        divisor=(int(input("introduce dividendo:")))  
        if  
            esDivisible(dividendo,divisor):  
                print(dividendo,"divisible")  
            else: print(dividendo,"no divisible")  
    except ValueError:  
        print("numero introducido no es válido.")  
  
#Sin subprograma y documentado  
  
def mostrarDivisible (divisor):  
    """int-->nada  
    OBJ: pide a usuario un número e indica si es divisible por  
    divisor  
    PRE: divisor>0'''  
  
    try:  
        divisor=(int(input("introduce dividendo:")))  
        if  
            esDivisible(dividendo,divisor):  
                print(dividendo,"divisible")  
            else: print(dividendo,"no divisible")  
    except ValueError:  
        print("numero introducido no es válido.")  
  
#probador  
n=9  
mostrarDivisible(int(sqrt(n)))
```

Normas: Este ejercicio es una evaluación individual. No puede usar libro ni hojas sueltas, memorias extraíbles, ni teléfono. Use internet solo para bajarse la biblioteca y para el manual que ofrece Python desde su IDLE oficial.

Entrega: Una vez terminado el ejercicio, avise a su profesor que va a entregar y, desde su correo de la uah, envíe un correo al profesor (rpr@uah.es), con asunto: "II_pecl1_apellidos_nombre". Sin texto del mensaje, con **un solo archivo** adjunto cuyo nombre será el mismo que el del asunto.

Recordatorios:

- 1) `print(var,end= 'textoFinal')` imprime 'texto final al final del print. Por defecto, salta una línea
- 2) Los parámetros de escritura con formato son
- 3) El operador * repite la secuencia n veces y tiene el formato: secuencia *n

Patrón	Tipo de dato
<code>%[n] d</code>	entero
<code>%[n].[d] f</code>	coma flotante
<code>%[n]s</code>	cadena de caracteres

ENUNCIADO

Una aplicación para una notaría confecciona una tabla de comprobación de las asignaciones en que se han distribuido los bienes entre los herederos de una testamentaría. En la figura presentamos un ejemplo parcial. La columna 1 contiene los nombres de los herederos, la 2, el coeficiente de la herencia a que tienen derecho. La 3, 4, y 5 diferentes tipos de bienes, la 6 es la suma de las 3 columnas anteriores en esa fila, de modo que en la fila Total de esta columna aparece el total de la masa hereditaria.

Heredero	Acciones	Deudas	Inmuebles	Total
Pepe	600.00	-10.20	1000.25	
Juan	700.00	0.00	2000.00	
Total	1300.00	-10.20	3000.25	4290.05

Ejemplo de datos:

```
herederos=('Pepe','Juan')      #tHerederos es una tupla de n
strings con los nombres
distribucion= ((600.0,700.0), (10.2,0.0)(1000.25,2000.0))
#tDistribuc tupla(tuplas) agrupa tDeudas, tInmuebles, tAcciones
```

- 1) (3 puntos) Hacer un subprograma que recibe los datos necesarios para calcular el total de un tipo de propiedades (ya sean acciones, deudas o inmuebles) y lo deja a disposición del programa llamante.
- 2) (3 puntos) hacer un programa que pinte la parte de tabla que está visible en el ejemplo actual. Apréciese que los datos están perfectamente encolumnados y contienen solo dos cifras decimales.
- 3) (3 puntos) Construya un subprograma que devuelva el número de suspensos (0..4,99), aprobados (5..6,99), notables(7..8,99) y sobresalientes(9..10), dada una secuencia que contiene las calificaciones numéricas (0...10) de un grupo de alumnos. Documente cuidadosamente los casos de prueba (1 punto). No olvide que debe hacer el menor número de preguntas.

- 1) (3 puntos) Hacer un subprograma que recibe los datos necesarios para calcular el total de un tipo de propiedades (ya sean acciones, deudas o inmuebles) y lo deja a disposición del programa llamante.

```
def totalPropiedad(tipo):
    """ tupla(num) --> float
    OBJ: total de un tipo de propiedades
    """
    total = 0.0
    for valor in tipo:
        total += valor
    return total

# PROBADOR
herederos = ('Pepe', 'Juan') # tHerederos es una tupla de n
string con los nombres
distribucion = ((600.0, 700.0), (-10.2, 0.0), (1000.25, 2000.0))
# tDistribuc tupla(tuplas) agrupa tDeudas, tInmuebles, TAcciones
print('El total de acciones debe de dar 1300.0:', totalPropiedad(distribucion[0]))
```

Errores Frecuentes

- 1) Frecuentemente documentaríamos "" tupla(num) →...) donde num agrupa int y float.
Advertid que si no se indica en la doc de los tipos de arg este hecho, debe indicarse en la PRE:; Si la tupla agrupa str y num, la operación suma no está definida
 - 2) Debemos acostumbrarnos a no hacer conversión implícita de tipos, por lo cual:
"total=0.0"

2) (3 puntos) hacer un programa que pinte la parte de tabla que está visible en el ejemplo actual. Apréciese que los datos están perfectamente encolumnados y contienen solo dos cifras decimales.

#distribucion= tupla de tuplas paralelas → misma longitud tuplas interiores

```

# Programa principal
print('Heredero\tAcciones\tDeudas\tInmuebles\tTotal')
print('=====|=====|=====|=====|=====')
for i in range(len(herederos)):
    print('%s\t%.2f\t%.2f\t%.2f' % (herederos[i], distribucion[0][i],
distribucion[1][i], distribucion[2][i]))

acciones = totalPropiedad(distribucion[0])
deudas = totalPropiedad(distribucion[1])
inmuebles = totalPropiedad(distribucion[2])

```

```
print('Total\t\t%0.2f\t\t%0.2f\t\t%0.2f\t\t%0.2f' % (acciones, deudas,
inmuebles, totalPropiedad((acciones, deudas, inmuebles))))
```

ERRORES FRECUENTES:

El enunciado pide un programa. Muchos alumnos han hecho un procedimiento. El error en esta dirección no es grave, ya que el probador hace de programa principal. En la contraria es muy grave, pues un programa no se podrá integrar en la aplicación. En cualquier caso, vuelvo a recordar la importancia de leer las especificaciones

- 3) (3+1 puntos) Construya un subprograma que devuelva el número de suspensos (0..4,99), aprobados (5..6,99), notables(7..8,99) y sobresalientes(9..10), dada una secuencia que contiene las calificaciones numéricas (0...10) de un grupo de alumnos. Documente cuidadosamente los casos de prueba (1 punto). No olvide que debe hacer el menor número de preguntas.

```
def resumenCalif(evaluacion):
    """ tupla(float) --> tupla(4 floats)
    OBJ: número de suspensos, aprobados, notables y sobresalientes en
evaluacion
    PRE: 0<=evaluacion[i] <= 10
    """
    nSuspensos = 0
    nAprobados = 0
    nNotables = 0
    nSobresalientes = 0
    # Adviertasé que si las calificaciones estuvieran ordenadas, habría un
    # algoritmo más eficiente
    for nota in evaluacion:
        if nota <= 4.99: nSuspensos += 1
        elif nota <= 6.99: nAprobados += 1
        elif nota <= 8.99: nNotables += 1
        else: nSobresalientes += 1
    return nSuspensos, nAprobados, nNotables, nSobresalientes

# PROBADOR
print('Debe dar 1 suspenso, 2 aprobados, 3 notables y 2 sobresalientes:', resumenCalif((0, 5.5, 6.8, 7, 7.8, 8.99, 9.5, 10.0)))
print('Debe dar 0 suspensos, 1 aprobados, 2 notables y 2 sobresalientes:', resumenCalif((5.0, 7.0, 8.0, 9.0, 10.0)))
```

ERRORES FRECUENTES

- Los nombres de las variables: Si un alumno pregunta a otro “en la calle” dime suspensos, es probable que le conteste “juan, pedro....”, por el contrario si pregunta: numSuspensos o contSuspensos, probablemente conseguirá la respuesta deseada.
- La propuesta que ofrecemos requiere solo 3 evaluaciones booleanas. Muchos alumnos han propuesto algoritmos que consumen el triple de los recursos necesarios, haciendo preguntas innecesarias. Copio como ejemplo, un código de alumno que consume 9 evaluaciones booleanas innecesarias. Es decir, “les cuesta el cine 12€ en vez de 3”:
PRE: 0<=nota[i]<=10

```
"""
suspensos=0
aprobados=0
notables=0
sobresalientes=0
for i in range(len(n)):
    if n[i]>=0 and n[i]<=4.99:           #rpr está en la pre      +2
        suspensos+=1
    elif n[i]>5 and n[i]<=6.99:          #rpr salidría por anterior +2
        aprobados+=1
    elif n[i]>7 and n[i]<=8.99:          #rpr salidría por anterior +2
        notables+=1
    elif n[i]>9 and n[i]<=10:            #rpr es un else !!! +3
        sobresalientes+=1
```

Resolución de problemas para ingenieros
con Python estructurado

Recopilación errores frecuentes

Rosalía Peña



Universidad
de Alcalá



U1. Contexto de la programación

- Solucionar un problema diferente al que ha solicitado el usuario (¿es un error de lógica?)
- Pensar que esta asignatura consiste en aprender Python en vez de pensamiento computacional
- Pensar que puedes cambiar tu razonamiento en la última semana o el último mes.

U2:Tipos de datos + Referenciación

- Desbordamiento.
- Redondeo.
- Pérdida de propiedad conmutativa: el orden importa.
- Convenio de nombres:
 - Constantes con mayúsculas
 - Variables minúsculas, compuestos pal1Pal2 o pal1_pal2
- Recomendado evitar expresiones mixtas entre tipos.
- Pep8 recomienda expresar const float con al menos un decimal.

U3. Programas secuenciales

- No mantener el orden de las secciones de un programa.

documentación

importación *

declaración de tipos*

declaración de constantes

declaración de variables*

declaración de procedimientos y funciones*

 con sus secciones a modo fractal *

cuerpo ejecutivo

entradas

proceso

salidas

- No documentar el archivo.py.
- Toda solicitud a usuario debe llevar mensaje conciso.
- Toda salida debe quedar documentada para el usuario.

* Aun no estudiado

U4: Subprogramación

- Los subprogramas son independientes: El nombre de arg formales y reales no tiene ni que coincidir, ni que ser diferentes.
- Más de un `return` por función.
- Todo subprograma debe estar documentado:
 - tiposEntrada—>Salida;
 - OBJ: mezcla nombre subprograma con arg;
 - PRE: (si la hay)
- Convenio de nombres:
 - Procedimiento → verbo acción
 - Función booleana → verbo de estado
 - Función otro tipo → sustantivo
- El uso global de constantes está permitido hay que indicarlo en la PRE:. Uso global de variables no permitido ([Python no deja modificar](#))
- La llamada a procedimiento es una instrucción, mientras que una función es un valor, por tanto forma parte de una expresión. Ej:
 - `print(procedimiento())` imprimirá none
 - `funcion()` como una instrucción, el resultado de la ejecución será inaccesible

U5: Errores frecuentes en condicionales

Sean expBool cualquier expresión booleana, vBool cualquier variable booleana, ki cualquier constante (i=1, 3..n) y ci cualquier secuencia de instrucciones (cuerpo)

Ineficiente	Mejorado	Recuerda...	Ventajas
<pre>if expBool: vBool=True else: vBool=False</pre>	vBool=expBool	A las variables booleanas, como cualquier otras se les puede asignar una expresión, no solo una constante	Menor complejidad ciclomática Menor complejidad algorítmica Menos que escribir Menos que leer
<pre>if expBool==True: c</pre>	<pre>if expBool: c</pre>	Las expresiones booleanas ya tienen un “valor de verdad”, es decir, son verdaderas o falsas, por si mismas	Menor complejidad algorítmica Menos que escribir Menos que leer
<pre>if valBool==k1: c1 if valBool==ki: c2</pre>	<pre>if vBool==k1:c1 elif vBool==ki: ci</pre>	varBool valiera k1 es imposible que valga k2	Menor complejidad ciclomática Menor complejidad algorítmica Menos que escribir y leer
<pre>PRE:vBool en {k1,ki} if vBool==const1:c1 elif... elif vBool==ki: ci</pre>	<pre>if vBool==k1:c1 else: ci</pre>	La precondition marca los valores posibles, si vBool, si no es ninguno de los anteriores, necesariamente es el último	Menor complejidad ciclomática Menor complejidad algorítmica Menos que escribir Menos que leer
<pre>if expBool: c1,c2 else: c1,c3</pre>	<pre>c1 if expBool: c2 else: c3</pre>	El condicional sirve para diseñar acciones diferentes en el if y en el else, si parte de las acciones sin comunes, hay que sacarlas fuera, para no redundar código	Facilitar el mantenimiento Menos que escribir Menos que leer
<pre>PRE: expBool""" if expBool: print('ERROR')</pre>	<pre>PRE: expBool"""</pre>	No debes contralar que se cumpla la precondition de un subprograma	Reparto de tareas, facilitar mantenimiento

U5: Gestión de excepciones

- Imprescindible encerrar toda petición de int o float a usuario en su correspondiente gestión de excepciones (excepto en probadores).

U6: Bucle for

- Cambiar parámetros control de bucle: ini, fin, inc, cont.
- ¿valor del contador a la salida del bucle?
- Anidar dos bucles con = contador.
- Un if que involucra al controlador del for, → for mal planteado.

```
for i in range(4):
    if i==2:break
    print('otra vez', i)

valor=ini
for i in range(fin):
    if i%2 ==0: valor+=3
    else: valor=valor+2.5
```

QueNoHacerConFor.py

```
""" Ejemplos de qué no hacer dentro de un for"""
```

```
ini=1
fin=4
inc=1

for cont in range(ini,fin,inc):
    print('tocado contador', cont)
    cont=cont-1

print('contador fuera del bucle', cont)

for cont in range(ini,fin):
    fin=fin+2
    print('tocado ini, fin')
print('extremo fuera del bucle', fin)
```

```
miLista=[1,2,3]
for i in range(len(miLista)):
    miLista.append(i*2)
print(miLista)
```

```
for i in range(2):
    for i in range(3):
        print(i)
```

U7: Bucles con condición

- Hacer la comparación de floats por igual o por distinto.
- Salirse del bucle con un `break` (No pertenece a la programación estructurada).
- No estudiar la convergencia.
- Un `if` dentro del bucle relacionado con las variables de control del bucle es sospechoso. **Esquema bucle condición múltiple**.
- Orden de las condiciones múltiples.
- Olvidar casos de prueba: todos los puntos extremos (primero, último, imposibles...).

U8: Registros

- No he localizado ninguno

U9: Errores frecuentes:

- El nombre: ~~recurrencia~~.
- Bucle de más: la propia recursividad es un bucle.
- Olvidar caso base.
- No garantizar la convergencia.
- Pensar que recursividad es mejor solución que la iteración.

Resolución de problemas para ingenieros
con Python estructurado

U1

El contexto de la programación

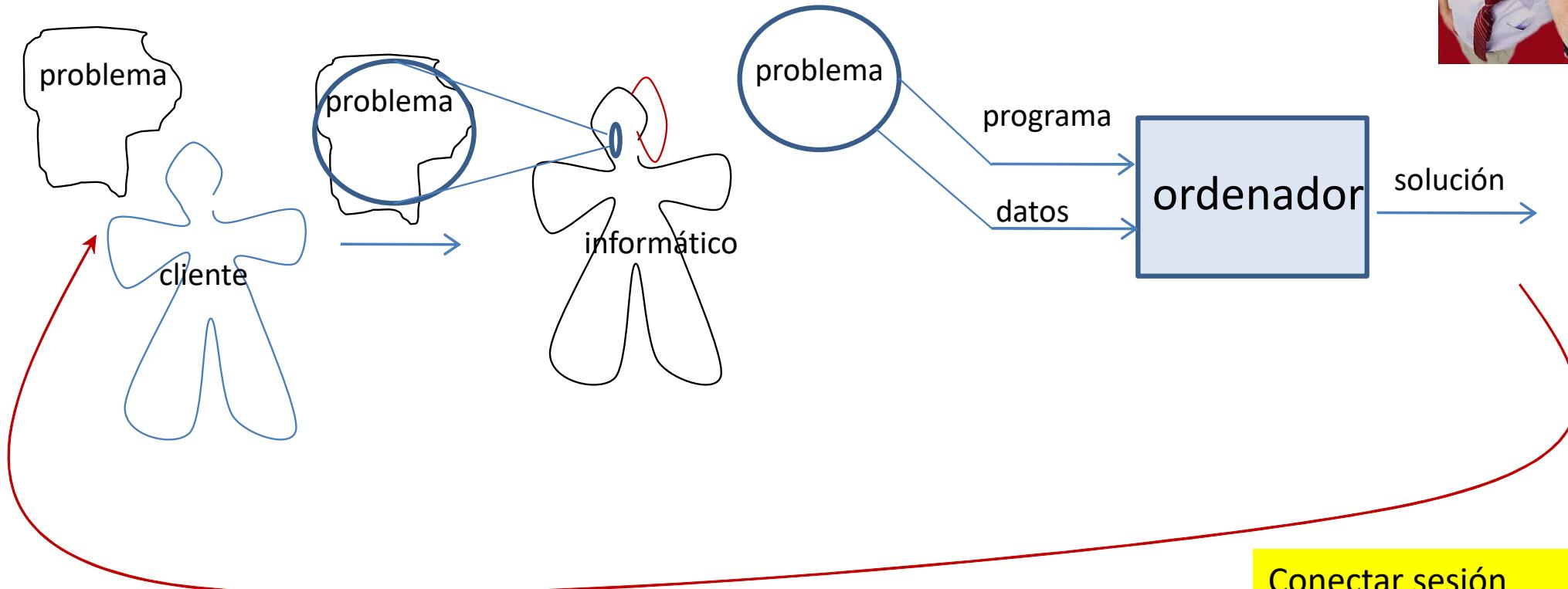
Rosalía Peña



Universidad
de Alcalá



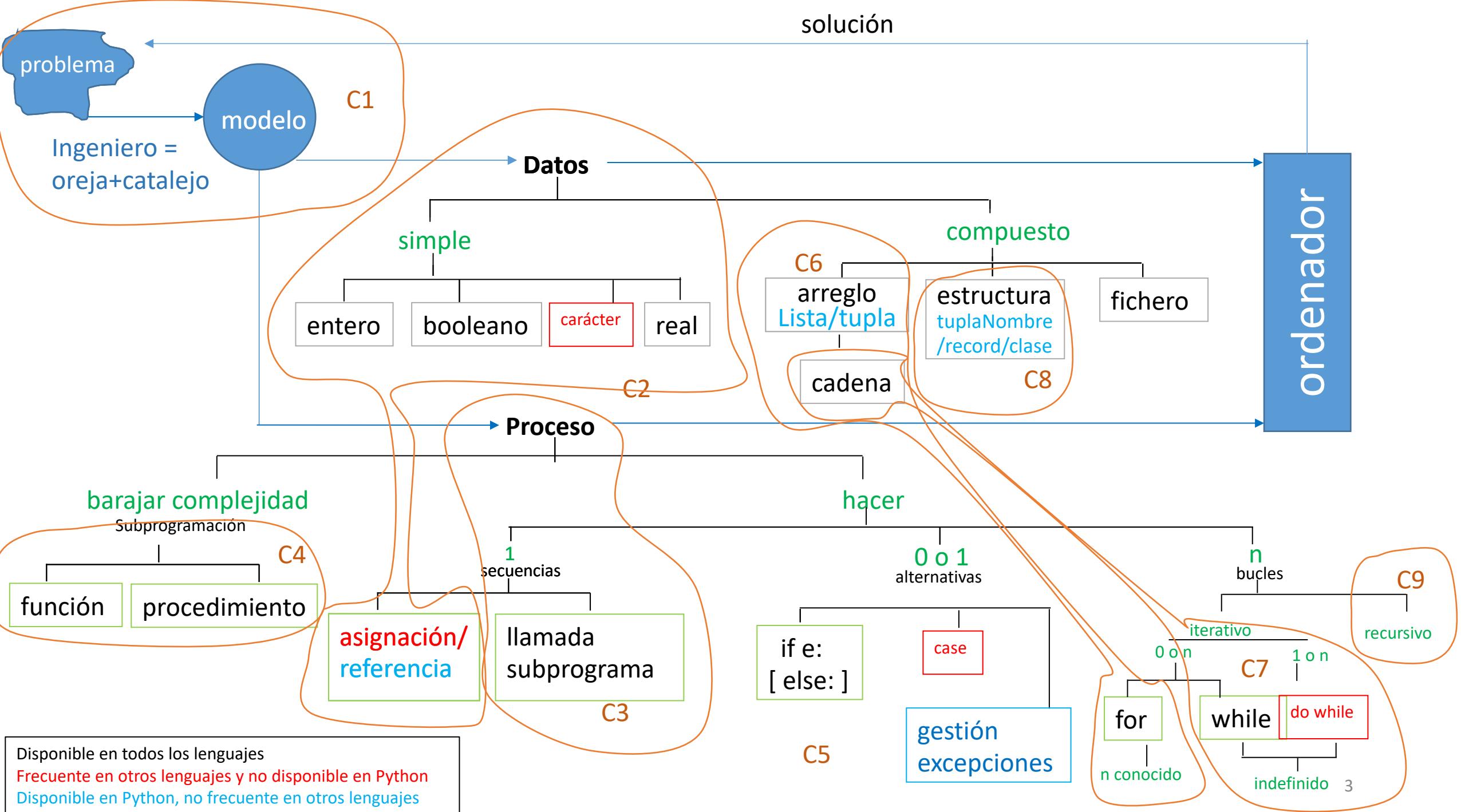
¿Qué es un ingeniero?



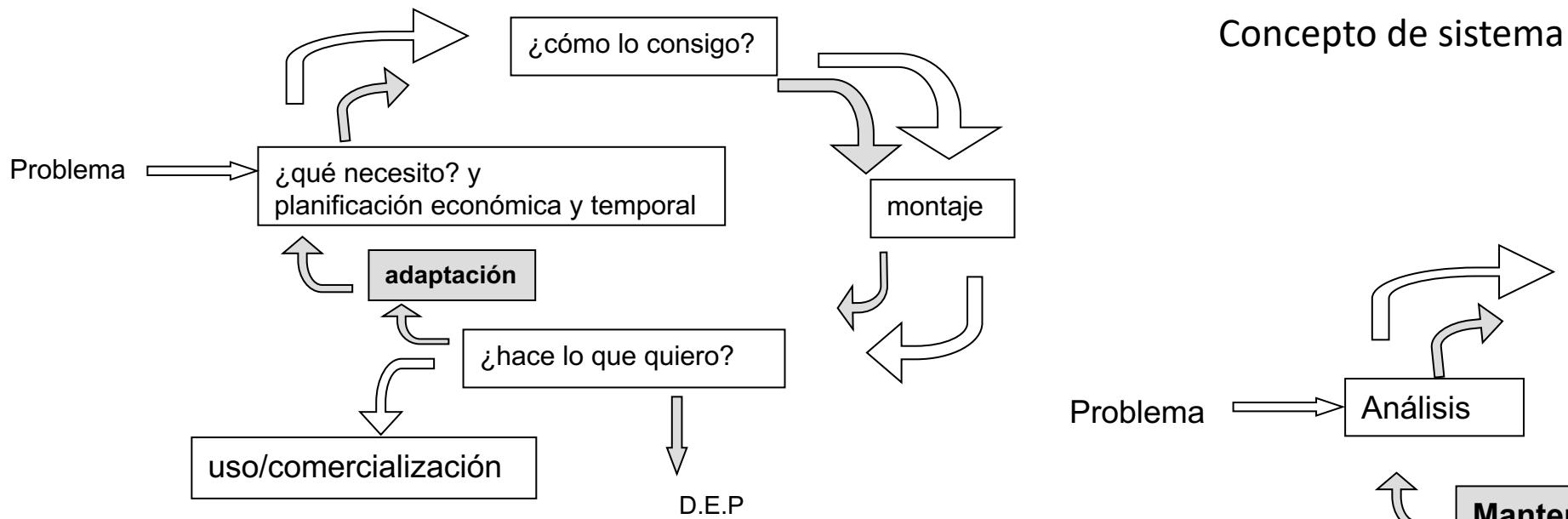
Conectarse sesión FPuah1.1
Ape1 ape2, nombre

¿Conclusión?: Por favor, copia!!!

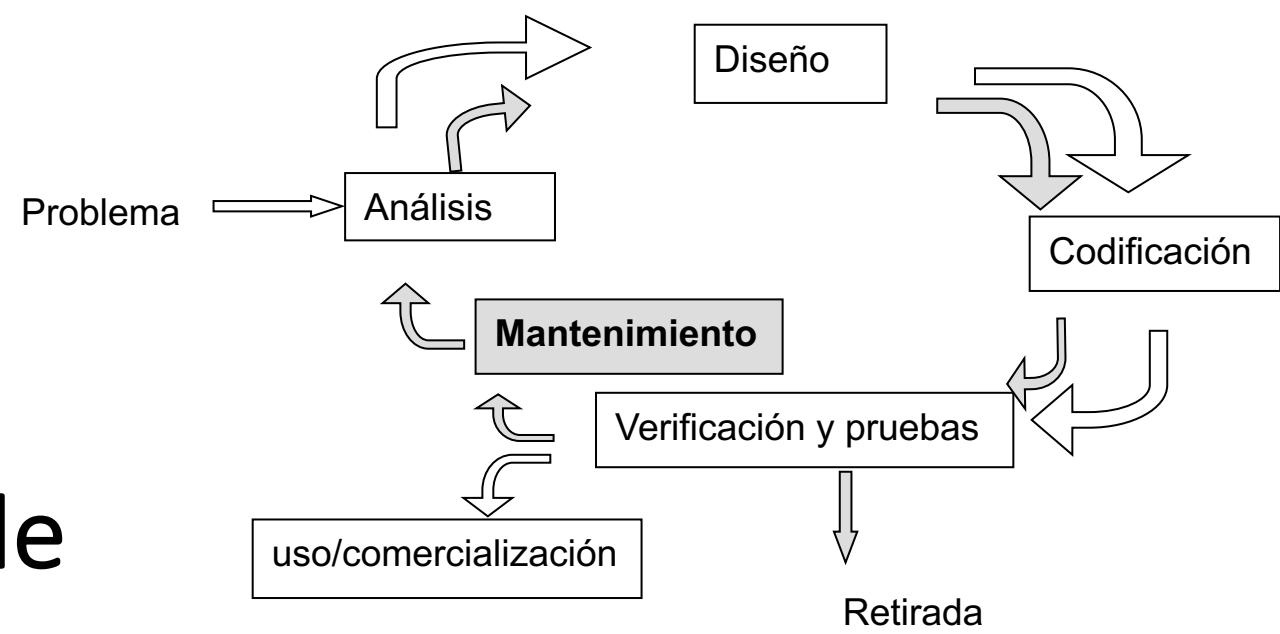
Cuestión 1 y 2



Ciclo de vida de una aplicación de ingeniería

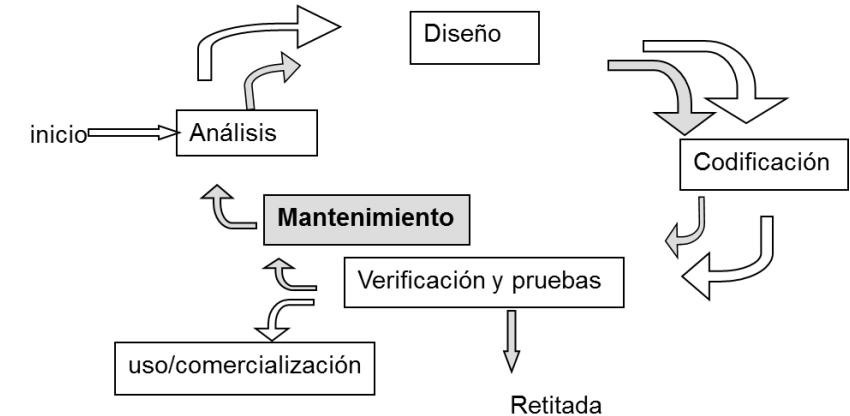


Concepto de sistema



Ciclo de vida de aplicación de
Ingeniería informática

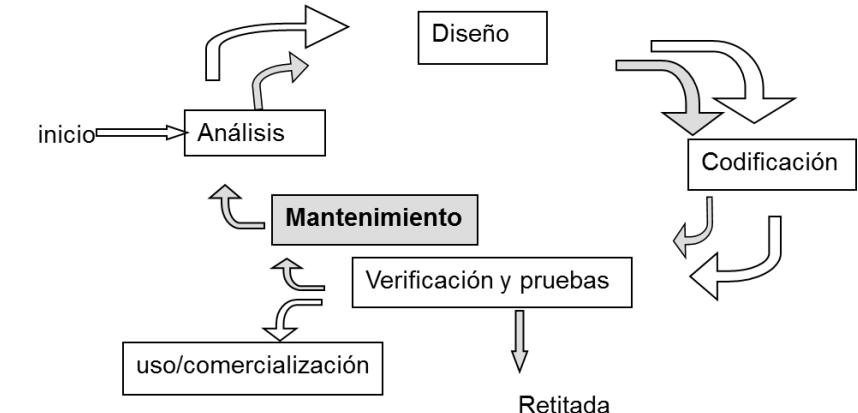
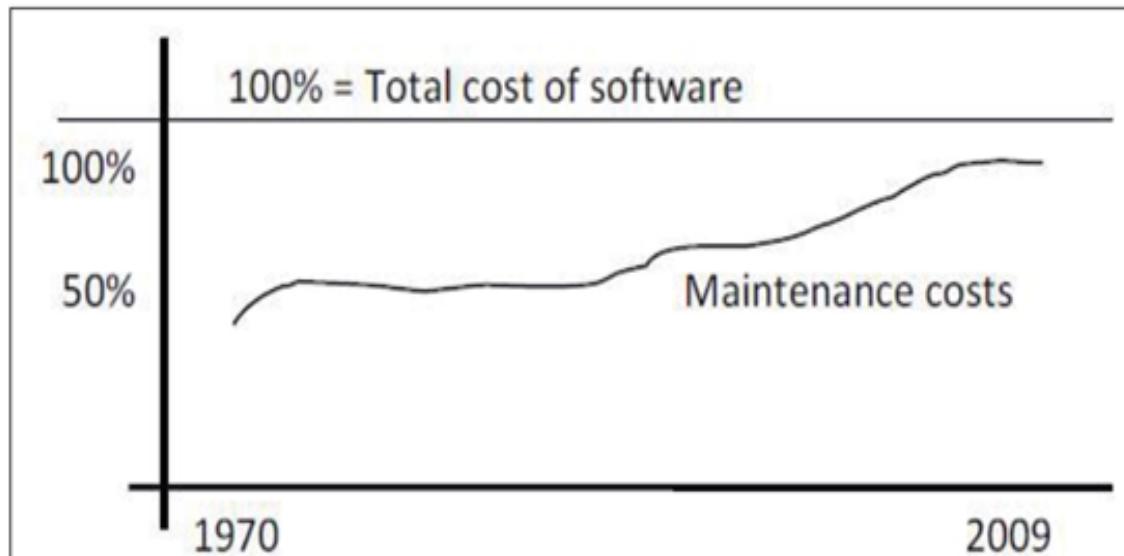
Ciclo de vida: Productos



Fase	Producto
Análisis	Especificaciones Estudio de viabilidad
Diseño	Algoritmo Especificación de los casos de prueba
Codificación	Programa Documentación técnica Documentación de usuario
Verif. y pruebas	Certificación/Especificación de mejora
Mantenimiento	Gestión de la configuración

Ciclo de vida: Costes

Cuestión 3. ¿% en mantenimiento?



→ mantenibilidad

Evolución del porcentaje de la inversión de software destinada a mantenimiento
<http://www.ncbi.nlm.nih.gov/pubmed/23572866>

Criterios calidad del soft

Hilo conductor → TEMARIO

Eficacia ≈ Robustez ≈ Fiabilidad

Recursos

- ☺ Lupa (buscar restricciones mundo real)
- Condicionales
- Gestión de excepciones

Mantenibilidad → Legibilidad

- Semántica (convenio identificadores)
- Orden de las secciones (±convenio)
- Documentación (convenio vemos después)
- Sangrado (lo fuerza Python)
- No redundancia: subprogramación
- Diseño

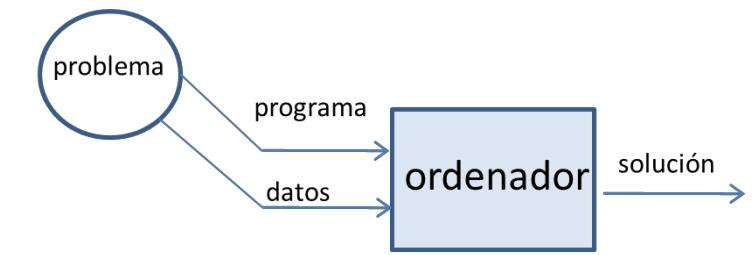
Reusabilidad

Copia, usa estándares, documenta, organiza
Subprogramación, diseño

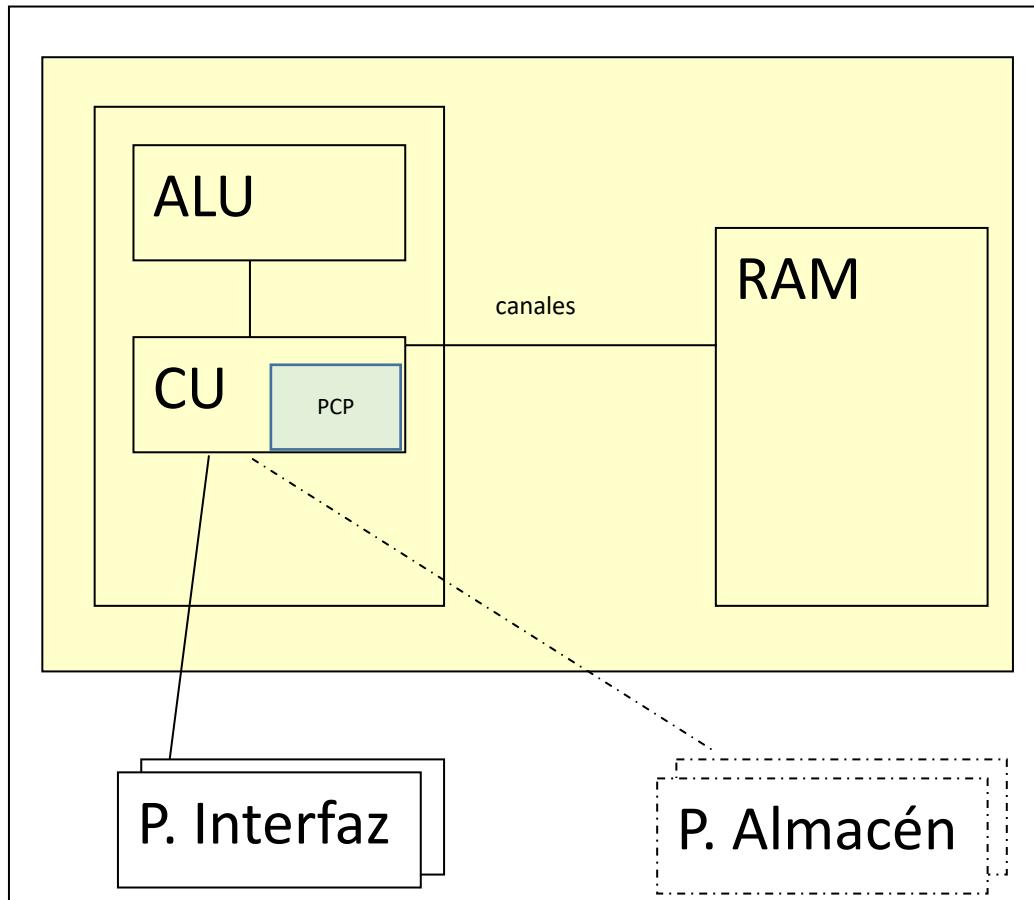
Eficiencia: tiempo, memoria (iniciación)

Diseño, algoritmia

Ordenador: Componentes



Arquitectura de von Newman 1945



ORDENADOR= SERVICIAL SOCIO COMPLEMENTARIO
Pero hay que hablarle en su idioma.



Microprocesador =CPU+RAM

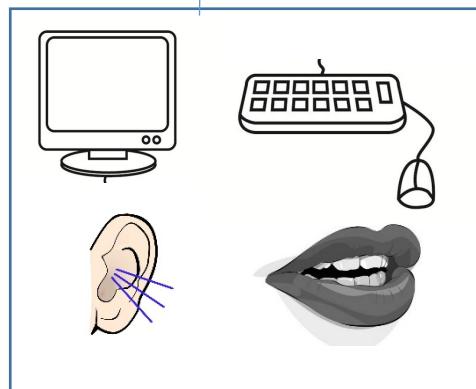
ALU:
Unidad aritmético lógica



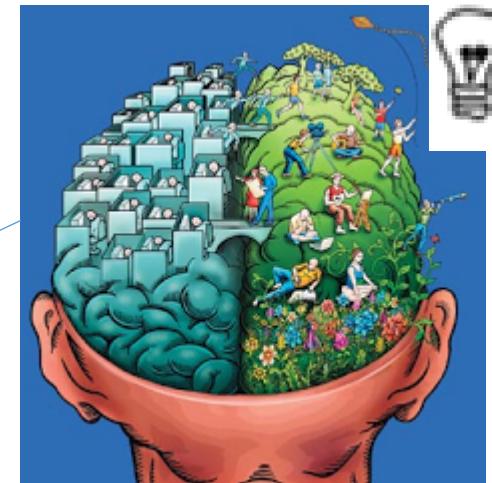
CU:
Unidad control



PI:
Periféricos de interfaz

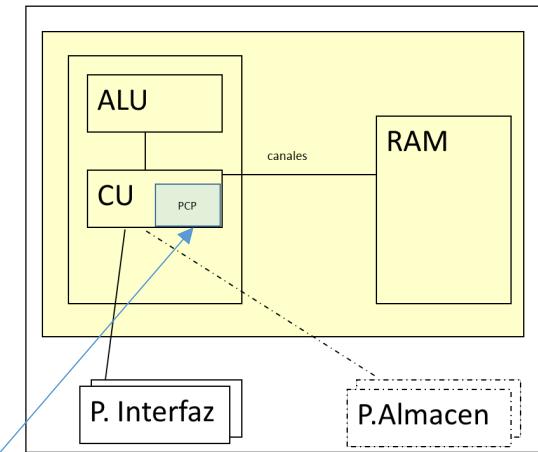


RAM: Memoria (Random Access)



bit

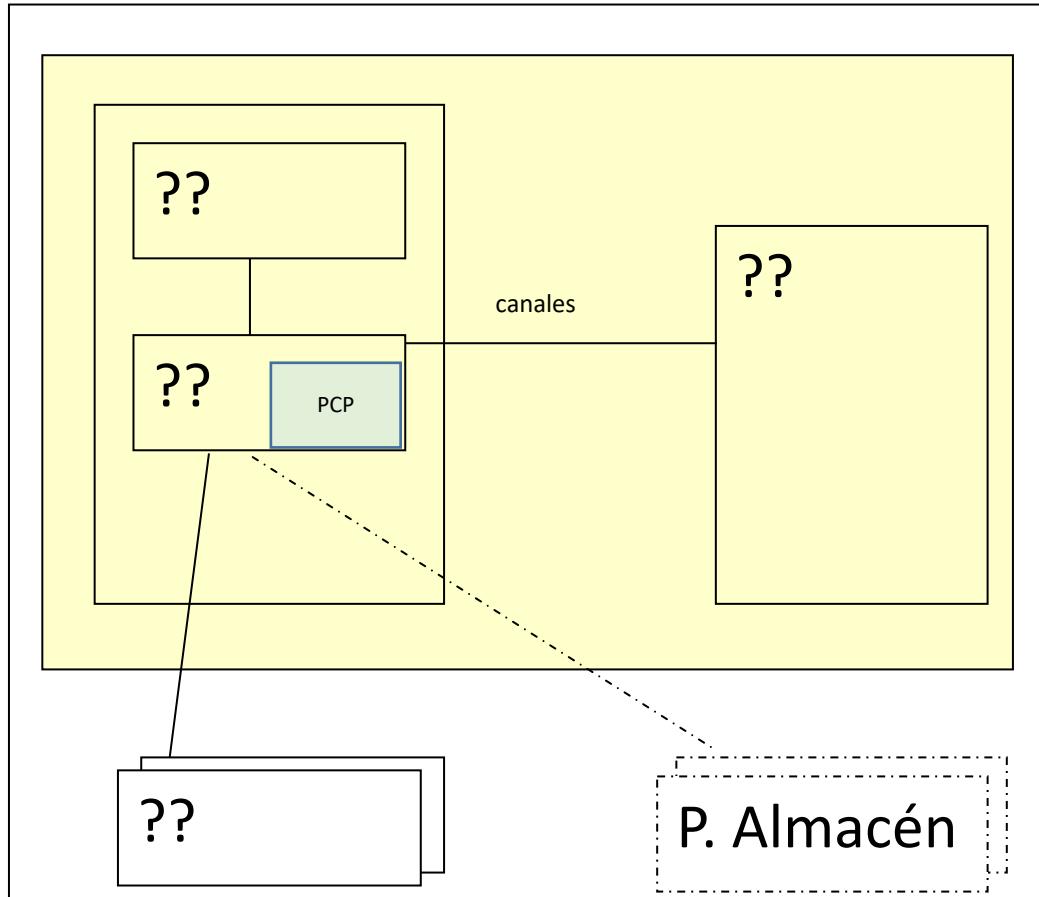
Puntero contador de programa



Periféricos de almacenamiento

Ordenador: componentes

Arquitectura de von Newman 1945

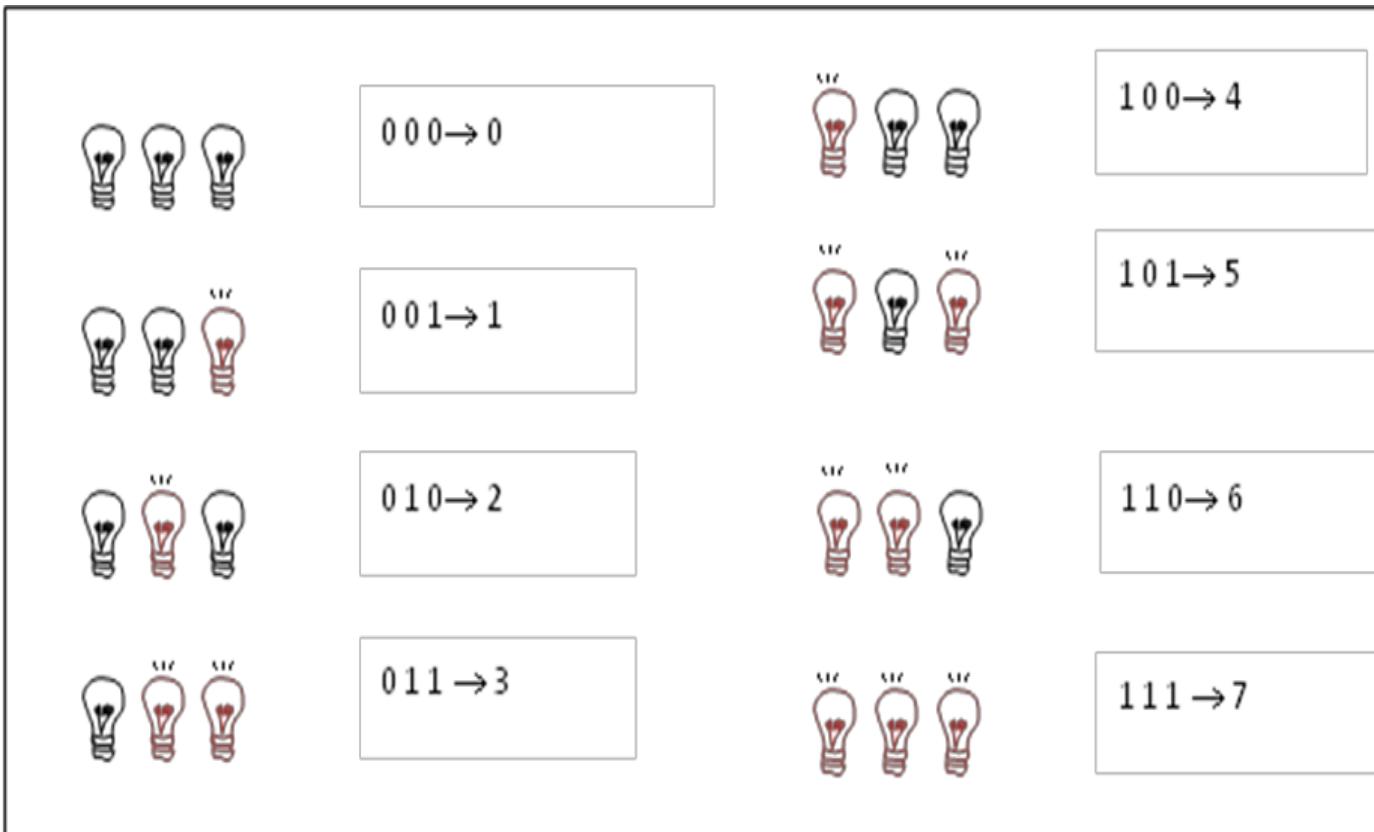
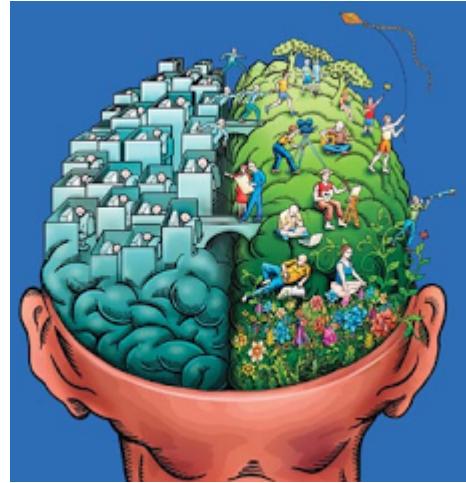


Cuestión 4. ¿componentes?

¿Cómo almacena información?

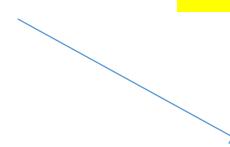


Dos estados



bits	estados
1	2
2	4
3	8
...	...
n	??

Cuestión 5. Generaliza...



En mundo real
no existe infinito

¿Cómo almacena información?: datos enteros

REPASO:
sistemas
numeración

$$165_{(\text{base } 10)} = 1 * \underbrace{100}_{10^2} + 6 * \underbrace{10}_{10^1} + 5 * \underbrace{1}_{10^0}$$
$$\rightarrow 100 + 60 + 5 = 165_{(\text{base } 10)}$$

$$10100101_{(\text{base } 2)} = 1 * \underbrace{128}_{2^7} + 0 * \underbrace{64}_{2^6} + 1 * \underbrace{32}_{2^5} + 0 * \underbrace{16}_{2^4} + 0 * \underbrace{8}_{2^3} + 1 * \underbrace{4}_{2^2} + 0 * \underbrace{2}_{2^1} + 1 * \underbrace{1}_{2^0}$$
$$128+ 0+ 32 + 0 + 0 + 4 + 0 + 1 = 165_{(\text{base } 10)}$$

Positivos

- 8 bits $\rightarrow 2^8 = 256$ estados
- 16 bits $\rightarrow 2^{16} = 65536$
-

¿Cuántos bits necesito para el signo?

Codificación

de 0 a 255

de 0 a 65535

¿255+1? \rightarrow desbordamiento (overflow)
lo menos malo

RESUMEN: Según el **número de bits** y la **codificación** \rightarrow diferentes **tipos de enteros**. Siempre dominio **finito** \rightarrow **desbordamiento**

Positivos y negativos

- 32 bits Complemento a dos \rightarrow de -2^{31} a $+2^{31}-1$
-

¿Cómo almacena información?: alfabéticos

- 26mayúsculas+ 26 minúsculas+10 dígitos= 62 $\rightarrow 2^6 = 64$
- {.,;?+-*\Ññ... }
- $2^8 = \text{byte, octeto} = 256$
- griego, chino,... $\rightarrow 2^{16} = 65536$ caracteres
- codificación tamaño variable UTF8.

Codificación: ASCII

RESUMEN: carácter alfabético = número de bits + **codificación** → **finito.**

Necesidad de estándar.

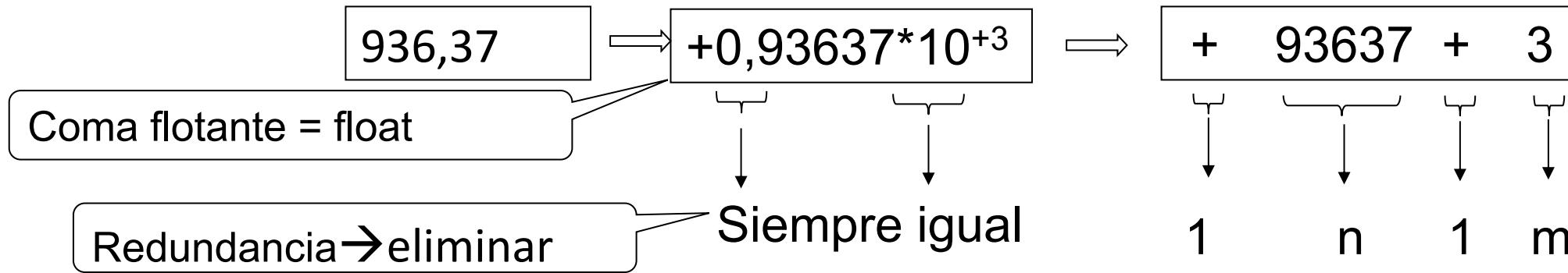
DEC	HEX	OCT	CHAR	DEC	HEX	OCT	CH	DEC	HEX	OCT	CH	DEC	HEX	OCT	CH
0	0	000	NUL	32	20	040		64	40	100	@	96	60	140	`
1	1	001	SOH	33	21	041	!	65	41	101	A	97	61	141	a
2	2	002	STX	34	22	042	"	66	42	102	B	98	62	142	b
3	3	003	ETX	35	23	043	#	67	43	103	C	99	63	143	c
4	4	004	EOT	36	24	044	\$	68	44	104	D	100	64	144	d
5	5	005	ENQ	37	25	045	%	69	45	105	E	101	65	145	e
6	6	006	ACK	38	26	046	&	70	46	106	F	102	66	146	f
7	7	007	BEL	39	27	047	'	71	47	107	G	103	67	147	g
8	8	010	BS	40	28	050	(72	48	110	H	104	68	150	h
9	9	011	TAB	41	29	051)	73	49	111	I	105	69	151	i
10	A	012	LF	42	2A	052	*	74	4A	112	J	106	6A	152	j
11	B	013	VT	43	2B	053	+	75	4B	113	K	107	6B	153	k
12	C	014	FF	44	2C	054	,	76	4C	114	L	108	6C	154	l
13	D	015	CR	45	2D	055	-	77	4D	115	M	109	6D	155	m
14	E	016	SO	46	2E	056	.	78	4E	116	N	110	6E	156	n
15	F	017	SI	47	2F	057	/	79	4F	117	O	111	6F	157	o
16	10	020	DLE	48	30	060	0	80	50	120	80	112	70	160	p
17	11	021	DC1	49	31	061	1	81	51	121	Q	113	71	161	q
18	12	022	DC2	50	32	062	2	82	52	122	R	114	72	162	r
19	13	023	DC3	51	33	063	3	83	53	123	S	115	73	163	s
20	14	024	DC4	52	34	064	4	84	54	124	T	116	74	164	t
21	15	025	NAK	53	35	065	5	85	55	125	U	117	75	165	u
22	16	026	SYN	54	36	066	6	86	56	126	V	118	76	166	v
23	17	027	ETB	55	37	067	7	87	57	127	W	119	77	167	w
24	18	030	CAN	56	38	070	8	88	58	130	X	120	78	170	x
25	19	031	EM)	57	39	071	9	89	59	131	Y	121	79	171	y
26	1A	032	SUB	58	3A	072	:	90	5A	132	Z	122	7A	172	z
27	1B	033	ESC	59	3B	073	;	91	5B	133	[123	7B	173	{
28	1C	034	FS	60	3C	074	<	92	5C	134	\	124	7C	174	
29	1D	035	GS	61	3D	075	=	93	5D	135]	125	7D	175	}
30	1E	036	RS	62	3E	076	>	94	5E	136	^	126	7E	176	~
31	1F	037	US	63	3F	077	?	95	5F	137	_	127	7F	177	DEL

¿Cómo almacena información?: booleanos

- True/False → Es suficiente con 1 bit.
- Los canales están preparados para octetos.

RESUMEN: Los lenguajes suelen **codificar** como subconjunto de enteros.

¿Cómo almacena información?: reales



- Ej: 32 bits y n=26 y m=4 →números entre 10^{-16} y 10^{+16} con 9 cifras significativas
- 754 del IEEE: Doble precisión 32 bits.

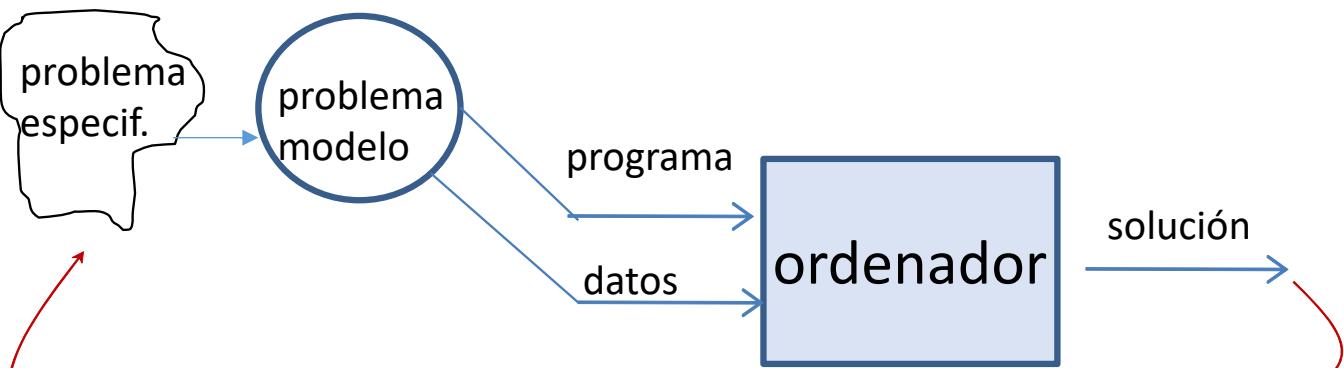
¿Cuál es el siguiente a 1 en los enteros?

¿Cuál es el siguiente a 1.0?

¿Cuántos números reales hay entre 1.0 y 1.1?

RESUMEN: **algunos** reales se representan con nº fijo de bits + codificación
→ **desbordamiento + redondeo.**

¿Estrategia de solución?

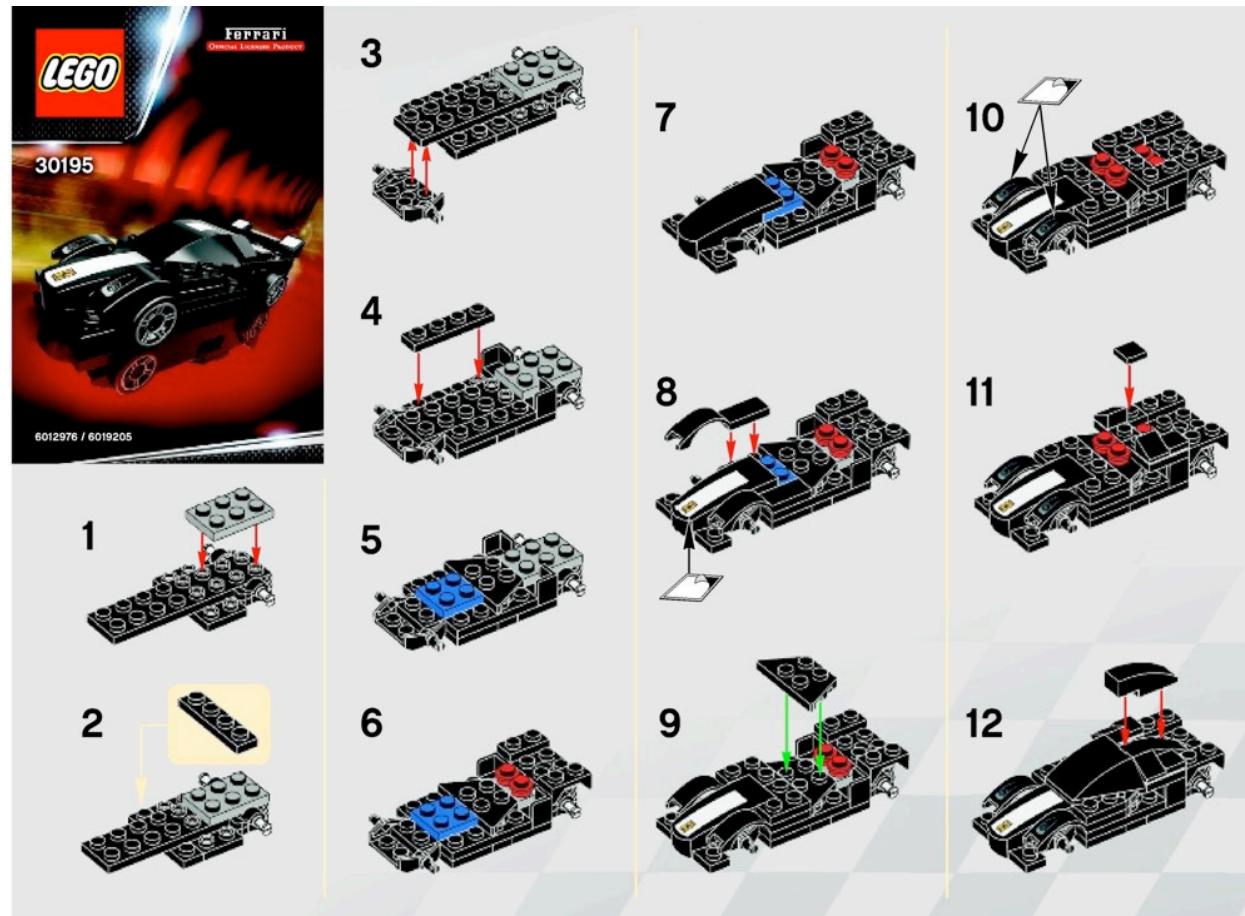


especificaciones → diseño → estrategia de resolución → codificación → programa

Algoritmo: secuencia ordenada de pasos que resuelve un problema.

Al-Khwarizm: matemático persa (siglo IX) que enunció los pasos para la resolución de algunos problemas de aritmética.

Ejemplos de algoritmos



Algoritmo (muy popular)...

Estirar el brazo izquierdo

Estirar el brazo derecho

Tocar el hombro derecho con brazo izquierdo

Tocar el hombro izquierdo con brazo derecho

Poner la mano izquierda en la nuca

Poner la mano derecha en la nuca

Poner la mano izquierda en la cadera

Poner la mano derecha en la cadera

Giro completo de la cadera

Saltar y girar 90a la derecha

Otros algoritmos...

Protocolo quirúrgico

Receta de cocina

Patrón de tejer un jersey

¿Pero qué es un programa?

ORDENADOR = servicial socio,
complementario,
incansable,
preciso;
pero... intransigente a ambigüedades
¿llevas hora?

Ejemplo de programa en lenguaje Python.

```
sumando1 = 3
sumando2 = 5
resultado = sumando1+sumando2
print(resultado)
```

Lenguaje de programación = lenguaje controlado

Realmente cada ordenador solo habla su lenguaje: lenguaje máquina

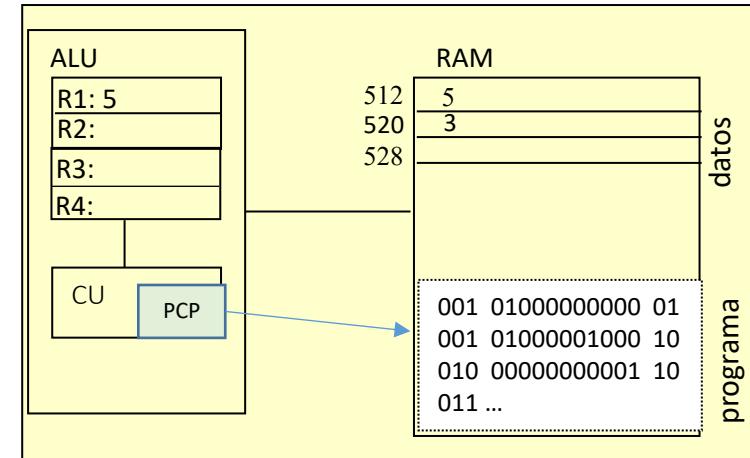
Programa = algoritmo escrito en lenguaje de programación

Operaciones máquina

- 4 registros en la ALU
- 8 circuitos operativos
 - 1. Mover de RAM a registro
 - 2. Sumar Ri, con Rj → Rj
 - 3. Mover de registro a RAM
 - 4. ¿es mayor R1 que R2?
 - 5. ...
 - 8. ...

- Instrucciones de 16 bits: op | dir | reg
- ¿Cuánta memoria puede gestionar?

Teorema de von Newman



instrucción = operación dirección registro

PCP →	1 Llevar el contenido de la posición de memoria 512 al R1 de ALU	⇒ 1 ₍₁₀₎	512 ₍₁₀₎	1 ₍₁₀₎
	2 Llevar el contenido de la posición de memoria 520 al R2 de ALU	⇒ 1 ₍₁₀₎	520 ₍₁₀₎	2 ₍₁₀₎
	3 Sumar R1 y R2 resultado en R1	⇒ 2 ₍₁₀₎	1 ₍₁₀₎	2 ₍₁₀₎
	4 Llevar R1 a posición 528 de memoria	⇒ 3 ₍₁₀₎	528 ₍₁₀₎	1 ₍₁₀₎

¿Dónde y cómo se almacena un programa?

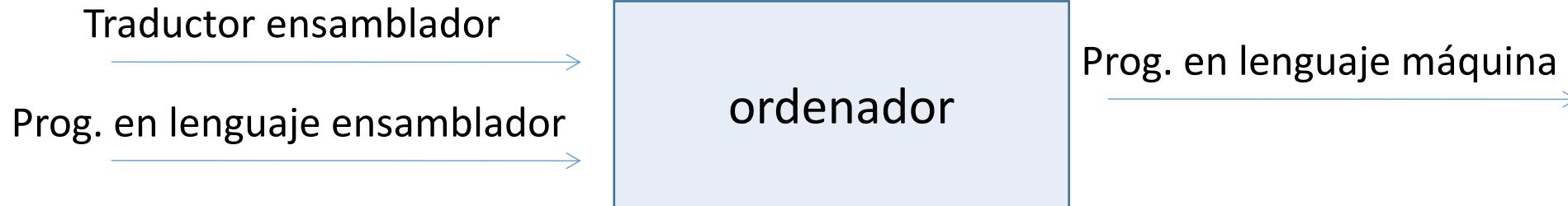
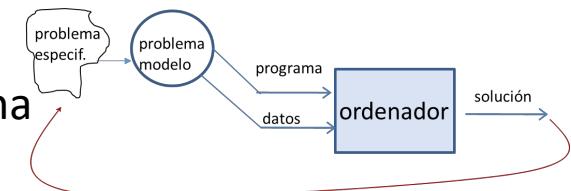
Lenguaje ensamblador

- Nombres nemotécnicos para posiciones de memoria
- Nombres nemotécnicos para operaciones
- macros

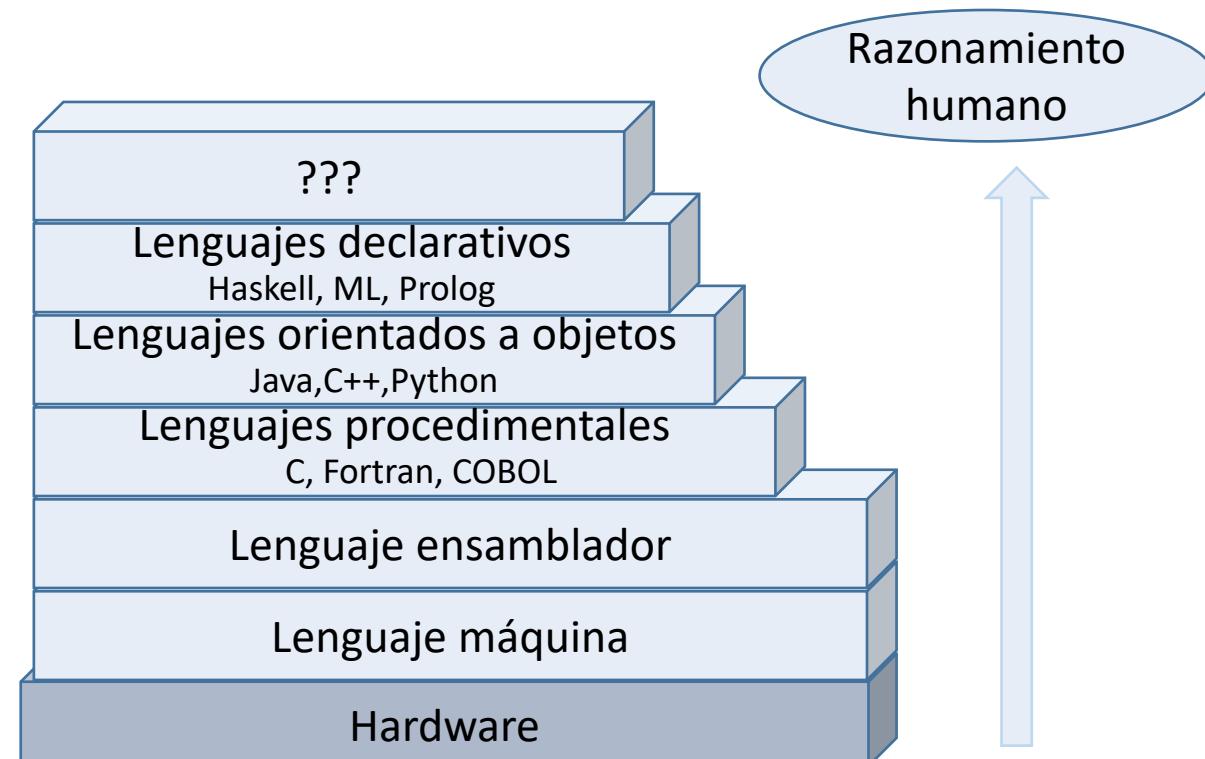
Ejemplo programa
* en Python $c = a + b$
* en lenguaje ensamblador

MOV	a	R1
MOV	b	R2
SUM	R1	R2
MOI	c	R1

Pero... Un nuevo problema: El ordenador no entiende ensamblador, solo lenguaje máquina



Lenguajes de alto nivel



>2300 lenguajes

Acercamiento a un área de aplicación:

Fortran: *FORmula TRANslator*; (1954)

BASIC: *Beginner's All-purpose Symbolic Instruction Code*,

COBOL: *Common Business Oriented Language* (1960),

Lisp: *List Processing*

Pascal: facilitar un aprendizaje sólido y riguroso

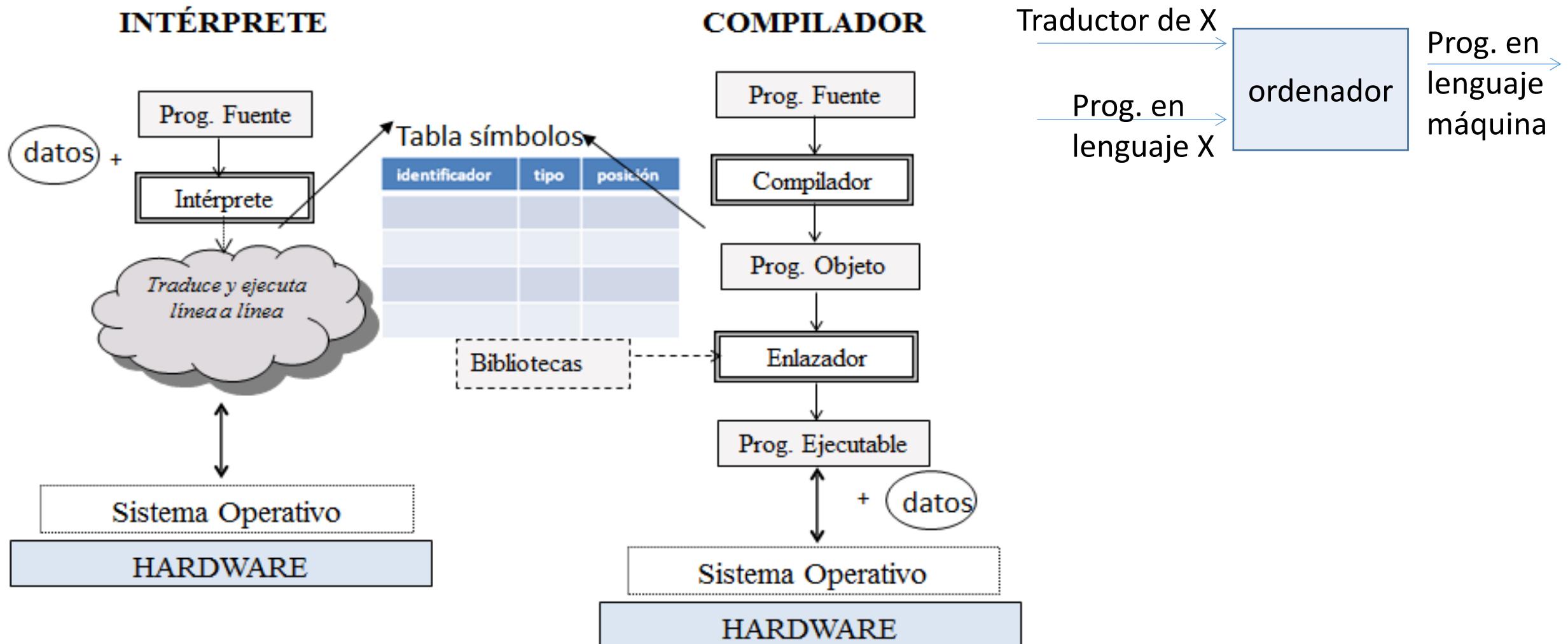
C: desarrollo de S. operativos y manejo de la máquina (1972).

Java: diseñado para barajar la complejidad de las aplicaciones.

Python: ¿? (1990. 2008 v3)



Traducción de lenguaje de alto nivel a lenguaje máquina

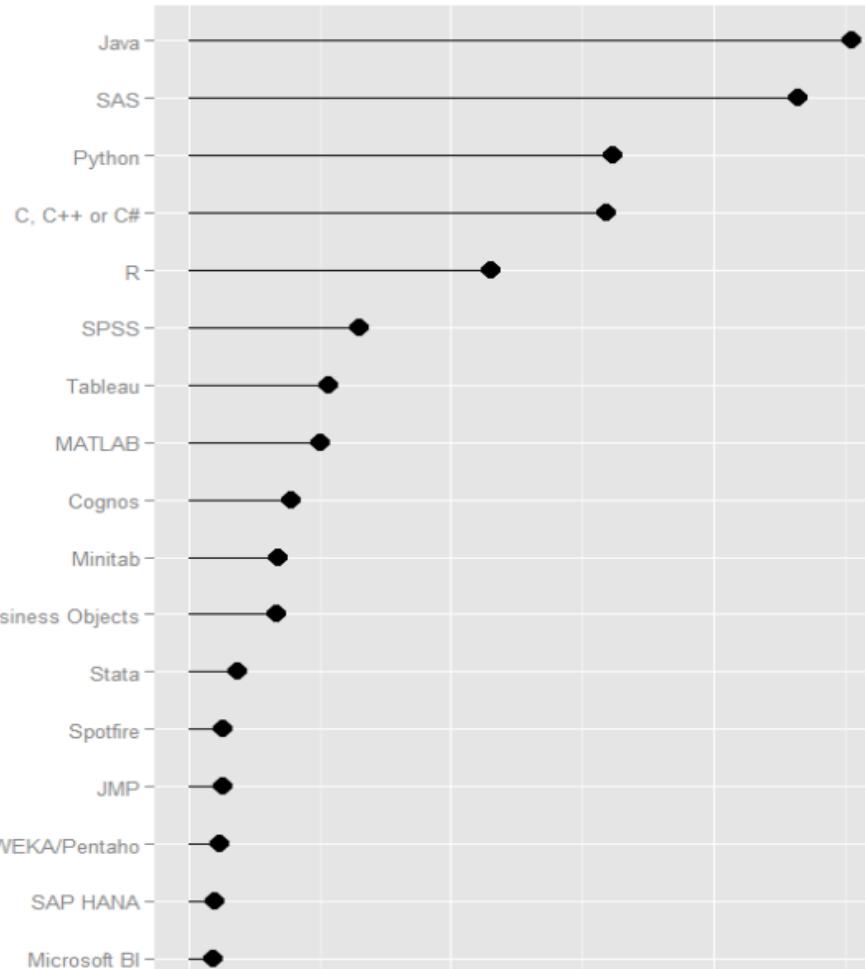


El lenguaje Python

- Muy expresivo, programas más cortos que en otros lenguajes.
- Sintaxis elegante, compromiso con la legibilidad.
- Interactivo → rápida retroalimentación al aprendizaje.
- Facilita la realización de pruebas.
- Muchos recursos: bibliotecas, documentación, foros y herramientas de visualización.
- Mensajes de error claros.
- Diversos paradigmas de programación: Nosotros subconjunto procedural modular, pero incluye orientación a objetos y programación funcional.
- Rico juego de estructuras de datos que se manipulan de modo sencillo.
- Es de libre distribución.
- Multiplataforma → portable.
- Impacto creciente en el entorno profesional.

Python en el entorno profesional

The More Popular Analytics Software



Ofertas de empleo en indeed.com

Bob Muenchen. <http://r4stats.com/2014/02/25/job-trends-improved>

lenguaje	Tecnoempleo	Infoempleo	Infojobs
Java	836	264	1338
C	110	292	250
Python	79	28	163

Ofertas de empleo en lenguajes España (09/2014)

Cuestión 8. FPuah1.1

El Zen de Python

Beautiful is better than ugly.

Explicit is better than implicit.

Simple is better than complex.

Complex is better than complicated.

Flat is better than nested.

Sparse is better than dense.

Readability counts.

Special cases aren't special enough to break the rules.

Although practicality beats purity.

Errors should never pass silently.

Unless explicitly silenced. In the face of ambiguity, refuse the temptation to guess.

There should be one-- and preferably only one --obvious way to do it. *Seguro que hay una manera obvia de resolverlo,*

preferiblemente solo una

(--> sigue pensandolo, no te pongas a programarlo hasta que la encuentres)

Although that way may not be obvious at first unless you're Dutch.

Now is better than never. Although never is often better than *right* now.

If the implementation is hard to explain, it's a bad idea.

Si tu código es difícil de explicar, está mal

If the implementation is easy to explain, it may be a good idea

Si es fácil de explicar, puede que esté bien

Namespaces are one honking great idea -- let's do more of those!

Pensamiento computacional

Los ordenadores son máquinas que ejecutan programas.

Los programas resuelven problemas utilizando algoritmos escritos en un lenguaje de programación.

El usuario ejecuta los programas que alguien puso a su alcance listos para usarse.

Programar es dar órdenes al socio servicial, fiable e incansable.

El pensamiento computacional permite expresar la estrategia de solución de los problemas en una forma que la pueda realizar un ordenador.

Aprender a programar es:

Gratificante (con tolerancia a frustración)

Requiere cambios actitudinales

→ requiere tiempo de maduración

No se olvida

→ no necesitas “empollarlo” el día antes del examen



Cuestión 10.



Errores: Clasificación

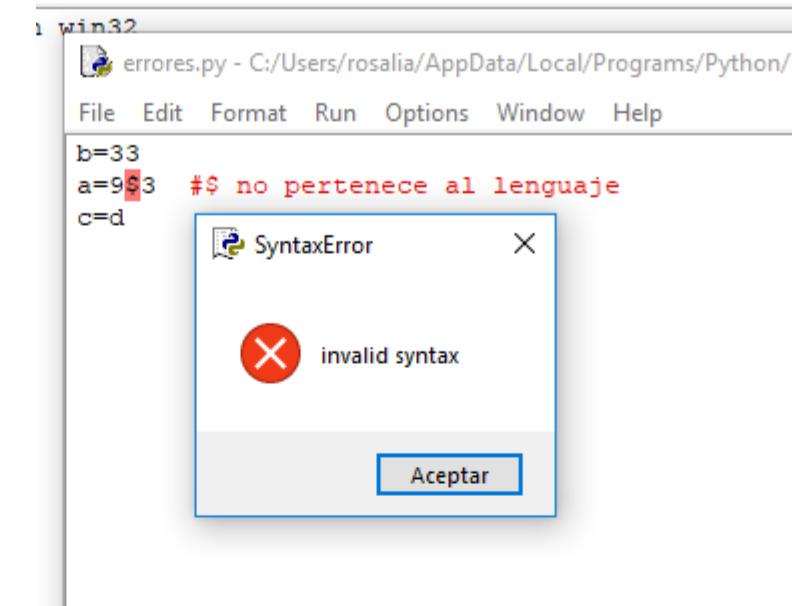
- Estirar el brazo izquierdo
- Estirar el brazo derecho
- Tocar el hombro derecho con mano izquierda
- Tocar el hombro izquierdo con mano derecha
- Poner la mano izquierda en la nuca
- Poner la mano derecha en la nuca
- Poner la mano izquierda en la cadera
- Poner la mano derecha en la cadera
- Giro completo de la cadera
- Saltar y girar 90 a la derecha

Tiempo de traducción
Tiempo de ejecución
Lógica
Estilo
Pensamiento computacional

Errores: Clasificación

- Estirar el brazo izquierdo
- Estirar el brazo derecho
- Tocar el hombro derecho con mano izquierda
- Tocar el **hombre** izquierdo con **mazo** derecha
- Poner la mano izquierda en la nuca
- Poner la mano derecha en la **nunca**
- Poner la mano izquierda en la cadera
- Poner la mano derecha en la **caldera**
- Giro completo de la cadera
- Saltar y girar 90 a la derecha

Tiempo de traducción
Tiempo de ejecución
Lógica
Estilo
Pensamiento computacional



Errores en **tiempo de traducción**:

hombre, mazo, nunca y caldera no pertenecen al contexto . Fáciles corregir por programador.

Errores: Clasificación

- Estirar el brazo izquierdo
- Estirar el brazo derecho
- Tocar el hombro derecho con hombro izquierdo
- Tocar el hombro izquierdo con mano derecha
- Poner la mano izquierda en la nuca
- Poner la mano derecha en la nuca
- Poner la mano izquierda en la cadera
- Poner la mano derecha en la cadera
- Giro completo de la cadera
- Saltar y girar 90 a la derecha

Tiempo de traducción
Tiempo de ejecución
Lógica
Estilo
Pensamiento computacional

¿otro problema?

Errores: Clasificación

- Estirar el brazo izquierdo
- Estirar el brazo derecho
- Tocar el **hombro** derecho con **hombro** izquierdo
- Tocar el hombro izquierdo con mano derecha
- Poner la mano izquierda en la nuca
- Poner la mano derecha en la nuca
- Poner la mano izquierda en la cadera
- Poner la mano izquierda en la cadera
- Giro completo de la cadera
- Saltar y girar 90 a la derecha

Tiempo de traducción
Tiempo de ejecución
Lógica
Estilo
Pensamiento computacional

The screenshot shows a Windows desktop environment. In the top right corner, there are two windows: one titled "WhatsApp" and another titled "errores.py - C:/Users/rosalia/AppData/Local/Programs/Python/Python36-32/errores.py". Below these, the taskbar has icons for File Explorer, Edge browser, and File Explorer again. In the center, there is a Python terminal window titled "Python 3.6.1 Shell". The terminal's title bar also shows "Python 3.6.1 (v3.6.1:69c0db5, Mar 21 2017, 17:54:52) [MS on win32]". The terminal content is as follows:

```
File Edit Shell Debug Options Window Help
Python 3.6.1 (v3.6.1:69c0db5, Mar 21 2017, 17:54:52) [MS on win32]
Type "copyright", "credits" or "license()" for more info
>>>
RESTART: C:/Users/rosalia/AppData/Local/Programs/Python/Python36-32/errores.py
Traceback (most recent call last):
  File "C:/Users/rosalia/AppData/Local/Programs/Python/Python36-32/errores.py", line 3, in <module>
    c=d
NameError: name 'd' is not defined
>>>
```

Errores en **tiempo de ejecución**:

Solicitud imposible. El programa aborta.

Detecta en fase de pruebas o **el usuario**

Errores: Clasificación

- Estirar el brazo izquierdo
- Estirar el brazo derecho
- Tocar el hombro derecho con **mano** izquierda
- Tocar el hombro izquierdo con mano derecha
- Poner la mano izquierda en la nuca
- Poner la mano derecha en la nuca
- Poner la mano **izquierda** en la cadera
- Poner la mano **izquierda** en la cadera
- Giro completo de la cadera
- Saltar y girar 90 a la derecha



The image shows a screenshot of a Windows desktop environment. In the top-left corner, there is a taskbar with icons for File Explorer, a search bar labeled 'Que desea hacer:', and buttons for 'Iniciar sesión' and 'Cambiarsesión'. The main window is a Python 3.6.1 development environment. It consists of two panes: an 'editor' pane at the top containing the code for 'errores.py', and a 'shell' pane below it showing the execution results.

Editor (errores.py):

```
""" radio de un círculo, a partir de su área"""
perímetro= 6.28
PI=3.14
radio=perímetro/2*PI #¿que está mal?
print('circf de 6,28 tiene radio =',radio)
print ('pero el perímetro de este radio es:=', 2*PI*radio) #comprobación
```

Shell:

```
Python 3.6.1 (v3.6.1:69c0db5, Mar 21 2017, 17:54:52) [MSC v.1900 32 bit on win32]
Type "copyright", "credits" or "license()" for more information.
>>>
RESTART: C:/Users/rosalia/AppData/Local/Programs/Python/Python36-32/errores.py
circf de 6,28 tiene radio = 9.8596
pero el perímetro de este radio es:= 61.918288000000004
>>> |
```

Errores de lógica: el programa funciona, termina proporcionando resultados incorrectos. **Cuestión 9. evaluación**
Difíciles de detectar. Potencialmente graves consecuencias.

Errores: Clasificación



- Tiempo de traducción
- Tiempo de ejecución
- Lógica
- Estilo
- Pensamiento computacional

Errores de estilo: Nombres inadecuados, falta de documentación, función con más de un return, código redundante... **Dificultan mantenimiento → aumenta costes**

Errores de pensamiento computacional: añadir instrucciones inútiles o mas complejas de lo necesario, elección de un algoritmo inadecuado... **Provocan ineficiencia y dificultan mantenimiento.**

¿En cual de las clases anteriores encajan los errores de desbordamiento?

¿En cual de las clases anteriores encajan los errores de redondeo?

RESUMEN

- Resolveremos problemas con programación **estructurada modular**
 - Pensamiento computacional.
 - Actitudes.
 - Materializando la solución en Python.
- Nuestros programas serán:
 - **robustos**, fiables, eficaces → resuelven el problema siempre que y solo cuando deben
 - **mantenibles**, legibles,
 - **reusables**.
- Un ordenador: ALU+UC+RAM+PI+buses (+PA)
- El ordenador almacena datos y programas en la RAM: nº fijo de bits + codificación (**estándares**)
- Solo es ejecutable directamente el lenguaje máquina. Ensamblador y A. Nivel requieren un traductor.
 - Compilador.
 - Intérprete.
- Errores de compilación, de sintaxis y de lógica. Errores de estilo y de comprensión.
- El aprendizaje requiere trabajo personal constante.

U1: ERRORES FRECUENTES

- Solucionar un problema diferente al que ha solicitado el usuario (¿es un error de lógica?).
- Pensar que esta asignatura consiste en aprender Python cuando el objetivo es entrenar el pensamiento computacional.
- Pensar que puedes cambiar tu razonamiento en la última semana o el último mes del curso.

Trabajo personal Unidad 1.

Trabajo personal 1.1. Ponte en la piel de tu cliente, ¿cómo resulta más atractivo el coche que le quieres vender?

- a) si le dices que lo has construido tu desde el más mínimo tornillo, partiendo de cero, en tu taller personal y que es el primer coche que haces en tu vida,
- b) o si le dices que está hecho por tu gran equipo de ingenieros, que habéis vendido ya muchas unidades, reutilizando los mismos conocimientos. Que está ampliamente probado. Que, habéis empleado los conocimientos de otros ingenieros, de hecho, utiliza la tecnología de los motores de Ford; el interior es un diseño de Roche Bobois que obtuvo un premio en 2005; los neumáticos son los de alta gama de Michelin, el *airbag* está construido a semejanza de los que emplea Ferrari en sus modelos de F1, etc.

¿Por qué ante tu software iba a reaccionar de distinta manera el cliente?

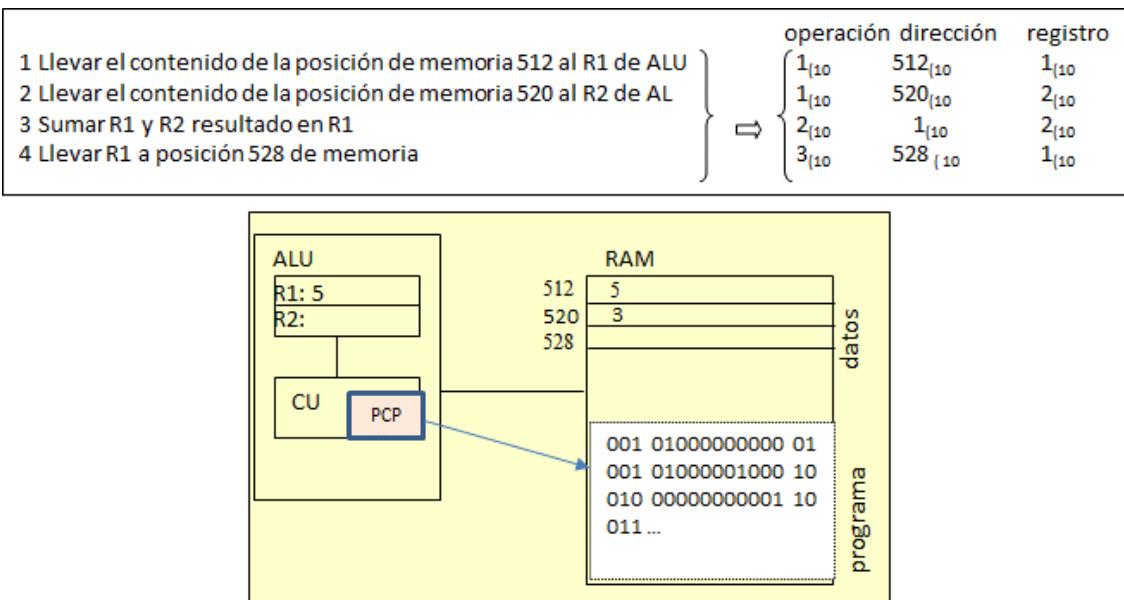


Figura 1.6. Un programa en lenguaje máquina

Trabajo personal 1.2. Completa la instrucción 4 del programa de la Figura 1.6, siguiendo las explicaciones que se dan en lenguaje español.

Trabajo personal 1.3 (con pista). Un lenguaje máquina que tiene instrucciones de 16 bits, 3 para las operaciones, 11 para las direcciones de memoria y 2 para los registros operativos de la ALU ¿cuántos circuitos tiene como máximo dicha máquina?, ¿cuántas posiciones de memoria puede direccionar?, ¿cuántos registros operativos tiene su ALU?

Trabajo personal 1.4. Cuando se empieza a trabajar con un lenguaje de programación es típico escribir un primer programa de tipo “Hola mundo”, es decir, conseguir simplemente mostrar un texto por pantalla donde saludemos al universo ☺. En <http://www.roesler-ac.de/wolfram/hello.htm> tienes el “Hola mundo” en 366 lenguajes de programación diferentes. Por curiosidad: dale un vistazo. Confirma desde este primer ejemplo, la simplicidad de Python comparada con otros lenguajes, por ejemplo Cobol, Java o C++.

Trabajo personal 1.5. Existe una tercera opción de traducción de lenguaje de alto nivel, que aúna las ventajas de compilador e intérprete, basada en el concepto de máquina virtual. Esta es la alternativa que usa Java. Documéntate sobre su funcionamiento.

Trabajo personal 1.6. Copia el código de la Figura 1.5 a Pythontutor y ejecuta línea a línea presionando el botón Forward. Fíjate en que, según avanza el puntero de ejecución (flecha azul), a través del programa, las variables en el programa, se van incluyendo en la tabla de símbolos (Pythontutor la llama frame).

Trabajo personal 1.7. Un magnífico profesor de la Universidad de Oviedo recogió, en clave de humor, las dificultades que encontrarás en tu primer acercamiento a la programación. Puesto que Python enfatiza el humor, el documento está en consonancia con nuestro contexto. Pese a su estilo desenfadado, es un artículo muy bueno y ha sido publicado en una revista puntera de investigación. Léelo. http://di002.edv.uniovi.es/~cernuda/noprog_ESP.html

Trabajo personal 1.8. Hemos mencionado algunas de las propiedades que debe tener un buen software: robustez, eficacia, legibilidad, mantenibilidad. Defínelas.

Trabajo personal 1.9. Coge papel y bolígrafo y fuérzate a definir con precisión los términos: estándar, RAM, ALU, CU, interfaz, periférico de Interfaz, periférico de almacenamiento, bit, byte, programa, instrucción, lenguaje máquina, código fuente, intérprete, compilador, versión de un software. Confirma la validez de tu respuesta. Definir con precisión es una de las habilidades que hemos de desarrollar como programadores. Ejercítala. Por otra parte, este ejercicio hace patente que hemos revisado una buena colección de conceptos en este capítulo ¡bien!

Trabajo personal 1.10. Pasa a decimal el número binario 10101.

Trabajo personal 1.10. Has decidido aprender a resolver problemas con el ordenador. Vas a trabajar muchas horas sentado delante del ordenador, de libros y papel en sucio. Es muy importante que cudes tu cuerpo para que este proceso. Lee el documento sobre ergonomía que te proporciona este espacio virtual.

Propuesta de solución al trabajo personal

1.3: Si no estás seguro de tu razonamiento, profundiza en el significado de la Figura 1.3.

1.6: Captura de pantalla con Pythontutor

The screenshot shows a web browser window for www.pythontutor.com/visualize.html#mode=display. The main area features a social network graph with nodes labeled Anna, Bill, Haley, Jim, Ying, Robert, Maria, Dmitri, and Omar. A central node contains the text "Free Two-Hour Python Course!". Below the graph, a banner for "UNIVERSITY of WASHINGTON" is visible. On the left, there's a "Python 3.3" code editor with the following code:

```
1 sumando1 = 3
2 sumando2 =5
3 resultado = sumando1+sumando2
4 print(resultado)
```

Below the code, there are links to "Edit code" and "Live programming". A legend indicates that green arrows point to the "line that has just executed" and red arrows point to the "next line to execute". A note at the bottom says: "NEW! Click on a line of code to set a breakpoint. Then use the Forward and Back buttons to jump there." At the bottom of the code editor are navigation buttons: << First, < Back, Step 3 of 4, Forward >, and Last >>. To the right of the code editor is a "Print output" panel with a text input field and a "drag lower right corner to resize" instruction. Below it are "Frames" and "Objects" sections. The "Global frame" section shows variables: sumando1 with value 3 and sumando2 with value 5.

Resolución de problemas para ingenieros con Python estructurado

U2

Tipos de datos y referenciación

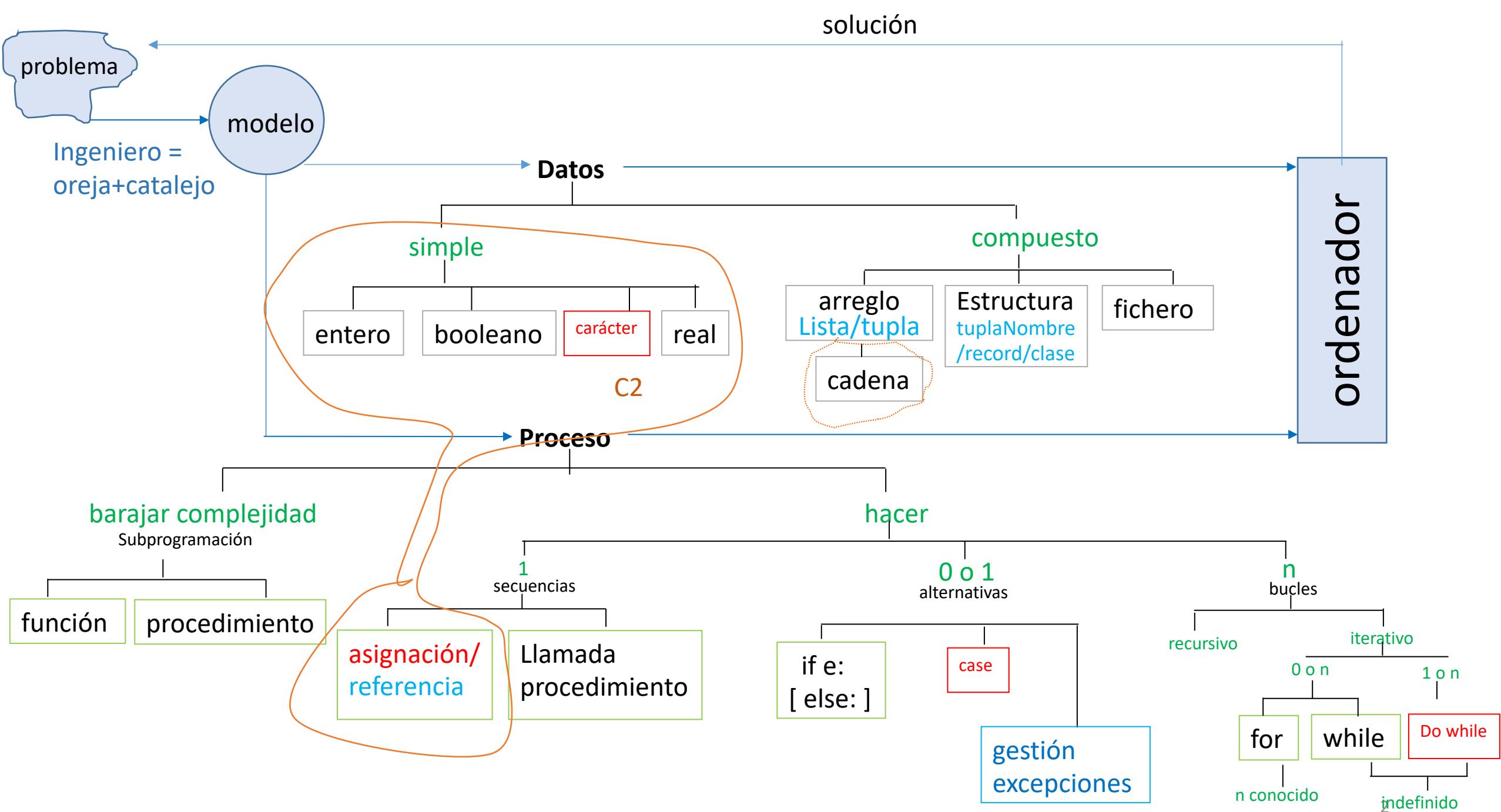
Rosalía Peña

Descárgate Qpython3
de la tienda en línea.

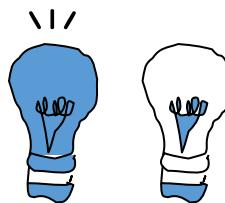
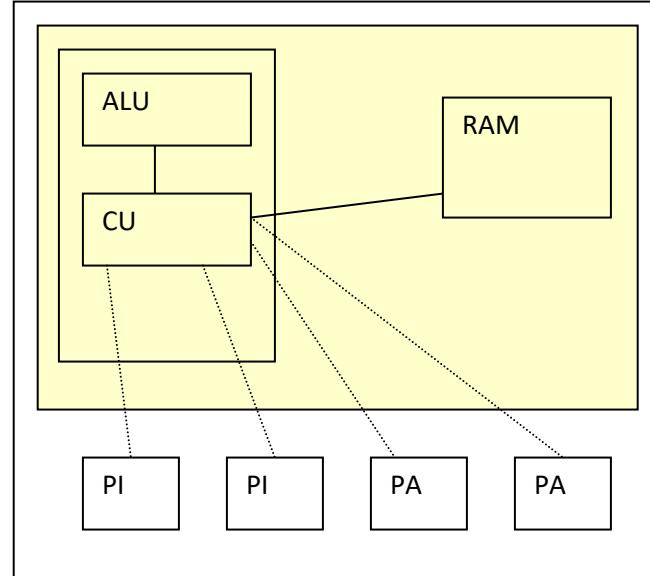


Universidad
de Alcalá



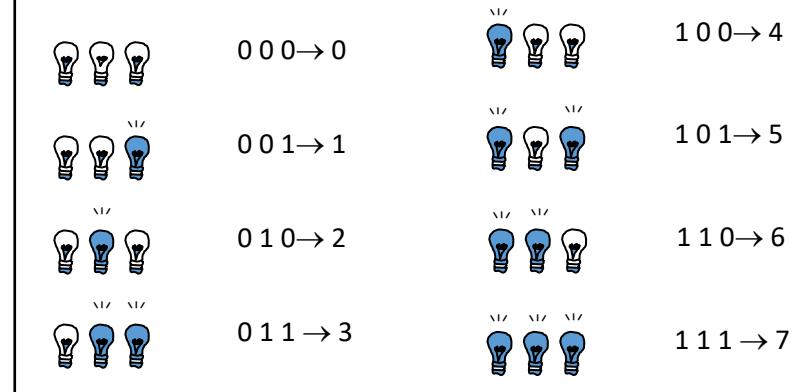


Recuerda cómo almacena la información el ordenador



bit

Naturales



Enteros: complementario a 2

TABLA DE CARACTERES DEL CÓDIGO ASCII

1	®	25	↓	49	1	73	I	97	a	121	y	145	æ	169	-	193	±	217	J	241	±	
2	●	26		50	2	74	J	98	b	122	z	146	È	170	-	194	218	219	242	≥	242	
3	♥	27		51	3	75	K	99	c	123	(147	ò	171	¾	195	¶	220	221	243	≤	243
4	♦	28	-	52	4	76	L	100	d	124)	148	ò	172	-	196	219	220	244	¬	244	
5	◆	29	↔	53	5	77	M	101	e	125)	149	ò	173	í	197	+	221	222	245	¬	245
6	◆	30	▲	54	6	78	N	102	f	126	-	150	ú	174	«	198	222	223	246	÷	246	
7	■	31	▼	55	7	79	O	103	g	127	»	151	ù	175	»	199	223	224	247	≈	247	
8	■	32	6	56	8	80	P	104	h	128	ÿ	152	ÿ	176	»	200	224	225	248	°	248	
9	■	33	!	57	9	81	Q	105	i	129	ü	153	ö	177	»	201	225	226	249	*	249	
10	■	34	"	58	:	82	R	106	j	130	é	154	Ü	178	»	202	226	227	250	.	250	
11	■	35	#	59	;	83	S	107	k	131	á	155	ç	179	»	203	227	228	251	✓	251	
12	■	36	\$	60	<	84	T	108	l	132	ñ	156	£	180	»	204	228	229	252	Σ	252	
13	■	37	%	61	=	85	U	109	m	133	à	157	ú	181	»	205	229	230	253	²	253	
14	■	38	&	62	>	86	V	110	n	134	á	158	þ	182	»	206	230	231	254	*	254	
15	■	39	/	63	?	87	W	111	o	135	ç	159	f	183	»	207	231	232	255	PRESIONA LA TECLA Alt	255	
16	►	40	(64	@	88	X	112	p	136	é	160	á	184	»	208	232	233	233	Θ	233	
17	►	41)	65	À	89	Y	113	q	137	ë	161	í	185	»	209	234	235	234	δ	234	
18	‡	42	*	66	B	90	Z	114	r	138	è	162	ó	186	»	210	235	236	235	∞	235	
19	!!	43	+	67	C	91	[115	s	139	í	163	ú	187	»	211	236	237	236	φ	236	
20	¶	44	,	68	D	92	\	116	t	140	í	164	ñ	188	»	212	237	238	237	o	237	
21	§	45	-	69	E	93	^	117	u	141	í	165	ñ	189	»	213	238	239	238	Ø	239	
22	‡	46	.	70	F	94	~	118	v	142	À	166	º	190	»	214	239	240	239	Ø	240	
23	‡	47	/	71	G	95	—	119	w	143	Á	167	º	191	»	215	240	241	240	Ø	241	
24	†	48	0	72	H	96	—	120	x	144	É	168	¸	192	»	216	241	242	241	Ø	242	

Char en Python UTF8=16 bits

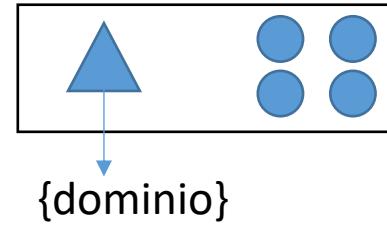
Reales: IEEE 754

Se almacena en:

- Secuencia de bits
 - De tamaño concreto
 - Usando un **código**
- Dominio **finito**

Tipo de datos simple

Objeto dato definido sobre un dominio, y sus operaciones

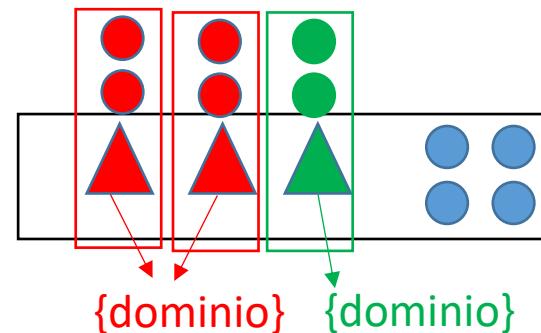


tipo	objeto	dominio	operaciones
mes	7	={1,2,3,4....12}	num_dias, siguiente, anterior
diaSemana	'martes'	{'lunes','...','domingo'}	siguiente, anterior, ¿laborable?
sueldo	1275.43	salarioMinimo<=sueldo<=80000	+,-,*,%....¿cobrado?...

Tipo de datos compuesto

Conjunto de objetos dato, con sus operaciones.

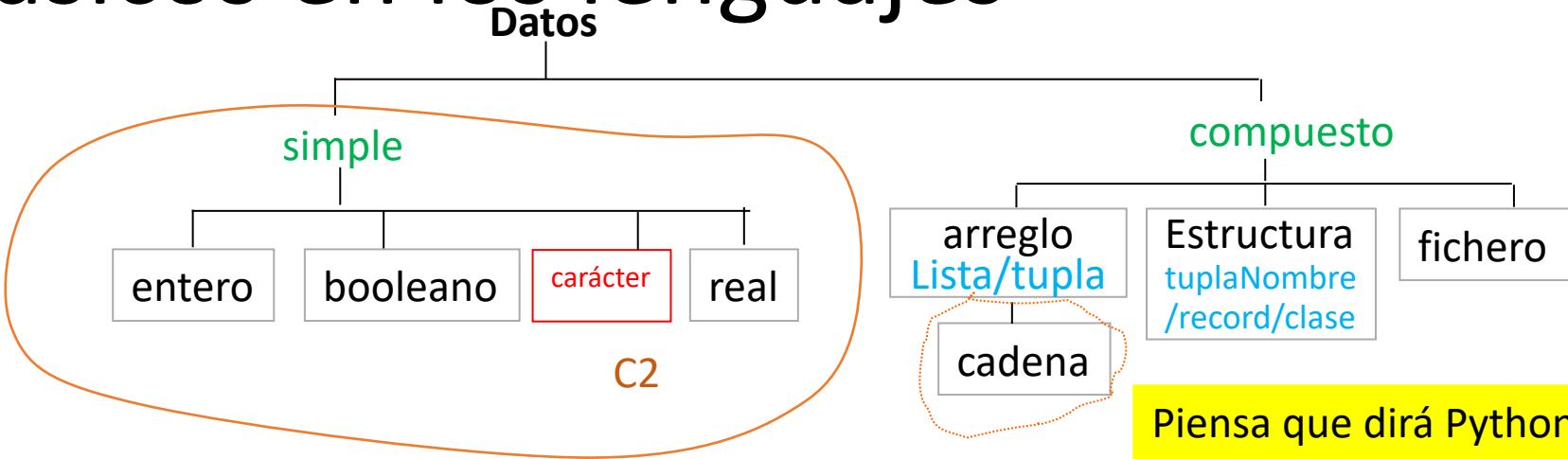
Cada elemento conserva sus propias operaciones.



tipo compuesto	elementos	operaciones
fecha	dia, mes, anno	diaSera (fecha,dias), diasTrascurridos(f1,f2)...
grupo	Alumno1, Alumno2,...	imprimirActa, asignarAula...
alumno	Nombre, titulación, asignatura...	matricular, asignarGrupo,...

Tipos básicos en los lenguajes

Una ventaja de lenguajes interpretados es la consola. Usamos para proceso de aprendizaje



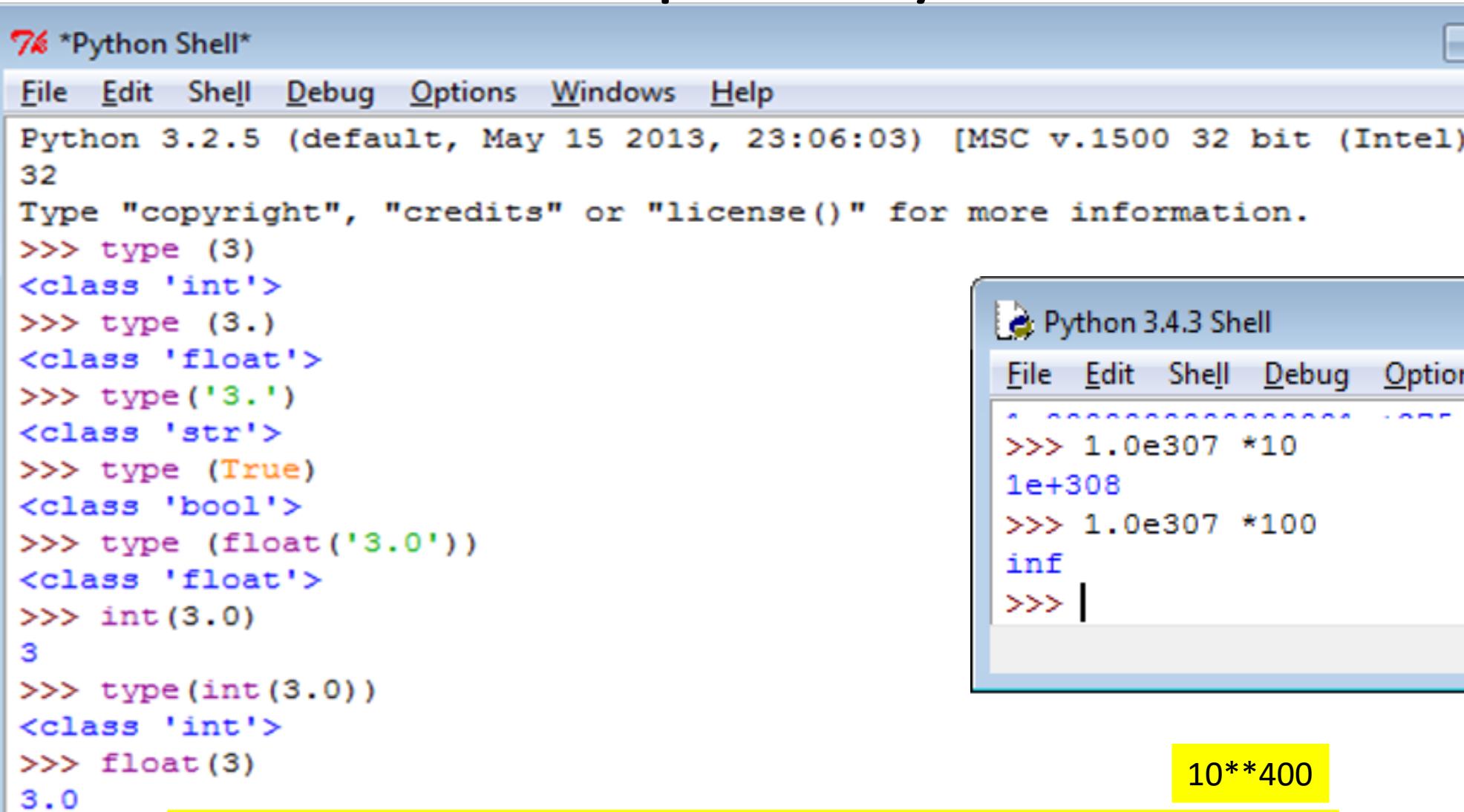
- Función type():
 - int,
 - float: caracterizado por el punto “.” PEP8 preferible “.0”
 - str: caracterizado por las comillas ‘cadena’
 - Cambio explícito de tipo: int(), float(), str()... :”cast”
 - Dominio finito:
 - Errores de desbordamiento
 - Errores de redondeo

Piensa que dirá Python y prueba en modo consola:

type (3)
type(3.0)
type ('3.0')
3.

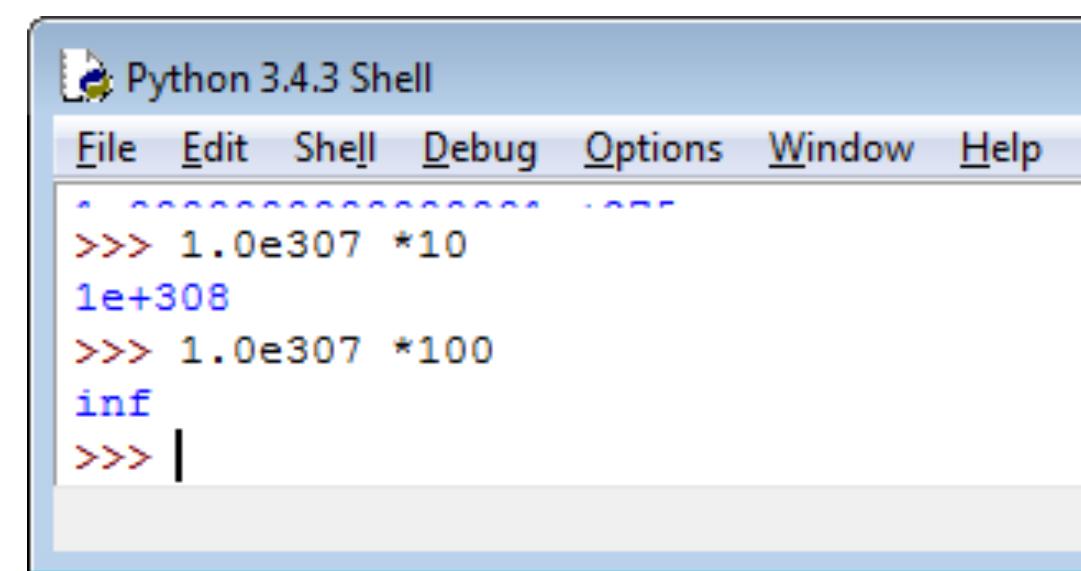
float(3)	10**30
float('3')	
int (3.0)	type(int (3.0))

Probad en el intérprete Python



```
76 *Python Shell*
File Edit Shell Debug Options Windows Help
Python 3.2.5 (default, May 15 2013, 23:06:03) [MSC v.1500 32 bit (Intel)]
32
Type "copyright", "credits" or "license()" for more information.

>>> type (3)
<class 'int'>
>>> type (3.)
<class 'float'>
>>> type('3.')
<class 'str'>
>>> type (True)
<class 'bool'>
>>> type (float('3.0'))
<class 'float'>
>>> int(3.0)
3
>>> type(int(3.0))
<class 'int'>
>>> float(3)
3.0
```



```
Python 3.4.3 Shell
File Edit Shell Debug Options Window Help
>>> 1.0e307 *10
1e+308
>>> 1.0e307 *100
inf
>>> |
```

10**400

¿Porqué no exactamente el mismo comportamiento en todos los dispositivos?

Caracterización de una cadena

'' y " "

- no intercambiables
- Y útiles
- Por economía ''

Alternativa en otros lenguajes:
caracteres de escape.

Probad en el intérprete:

```
>>> hola  
>>> abs
```

```
>>> hola  
Traceback (most recent call last):  
  File "<pyshell#25>", line 1, in <module>  
    hola  
NameError: name 'hola' is not defined
```

```
>>> 'Hola'  
>>> 'Juan dijo: "hola"'  
>>> " juan dijo 'Hola" '
```

```
>>> 'Hola'  
'Hola'  
>>> 'Juan dijo: "hola"'  
'Juan dijo: "hola"'
```

Operaciones internas

- tipo operador tipo → tipo; op_int {+|-|*|//|...} op_int → res_int,
op_float{+|-|*|/} op_float → res_float.
- Garantiza poder concatenar operaciones $2+3+6=...$
- Sobrecarga:

`2 + 3=... ; 2.+3.= ... '2'+'3' =`

- Ojo división real
división entera
- Resto de la división entera `15%12=` Aritmética del reloj, o modular.
- Algunas operaciones en formato de función:
 - `divmod (numerador,denominador),`
 - `round (7.78)`
 - `...abs(-7), ya irán saliendo...`

`7.0/2.0=`

`7//2 =`

`15%12=`

`int (7.78)`

¿diferencia `round() / int ()?`

Operaciones con operandos de diferente tipo

- Promoción implícita (diferente en diferentes lenguajes)

2+3. =

5/2=2.5

5.0//2.0 = 2.0

2+'3'= ??

Preferible conversión explícita

The screenshot shows a Python Shell window with the title bar "76 *Python Shell*". The menu bar includes File, Edit, Shell, Debug, Options, Windows, and Help. The main area displays the following Python interactions:

```
>>> 3+2
5
>>> 3+2.0
5.0
>>> 3+'2'
Traceback (most recent call last):
  File "<pyshell#10>", line 1, in <module>
    3+'2'
TypeError: unsupported operand type(s) for +: 'int' and 'str'
```

The first three lines show successful type promotions: integer addition with a float, integer division with a float, and integer division with another integer. The final line demonstrates that Python does not support adding an integer and a string directly, resulting in a `TypeError`.

Tipos básicos: Precedencia numéricos

$$2+3+5=10 ;$$

$$2+3*5= ?$$

Prelación
Prioridad
~~Preferencia~~
Prelación diferente en diferentes lenguajes

Precedencia	Operación	Resultado	Ejemplo	
1	$-x$	Cambio de signo	$a=3, -a=-3$	
1	$+x$	Operador identidad (no hace nada)	a	
2	$x + y$	Suma x e y	$3+2=5$	
2	$x - y$	Resta y de x	$5-2=3$	
3	$x * y$	Multiplica x por y	$2+3*5 =17$	A igual prelación,
3	x / y	División real		de izquierda a derecha
3	$x // y$	División entera	$13//6/2$	
3	$x \% y$	Resto de la división entera de x/y		
4	$x^{**} y$	x elevado a y	$3*2^{**}3+4=28$	
5	(Expresión)	Paréntesis: fuerza el orden ejecución	$(2+3)*5=25$	

¡OJO! $6/2*3$ es distinto de $6/2/3$ ¿= $6/(2*3)$?

Operadores de comparación

- en todos los tipos, entre dos operandos de “=tipo” → boolean
 - No tipos mezclados
 - == igual
 - != distinto
 - > mayor
 - < menor
 - >= mayor igual
 - <= menor igual

Tipos básicos: booleanos

Dominio: True | False

Operaciones: **and**, **or**, **not**

precedencia diferente en distintos lenguajes

Precedencia	Operación	Nombre	Resultado
1	p or q	Disyunción	True siempre que uno de los operandos es True
2	p and q	Conjunción	True solo si ambos operandos son True
3	not p	Negación	True si p era False y viceversa

Diseña una prueba de la precedencia de **or** y **and**

Evaluación perezosa o en cortocircuito:

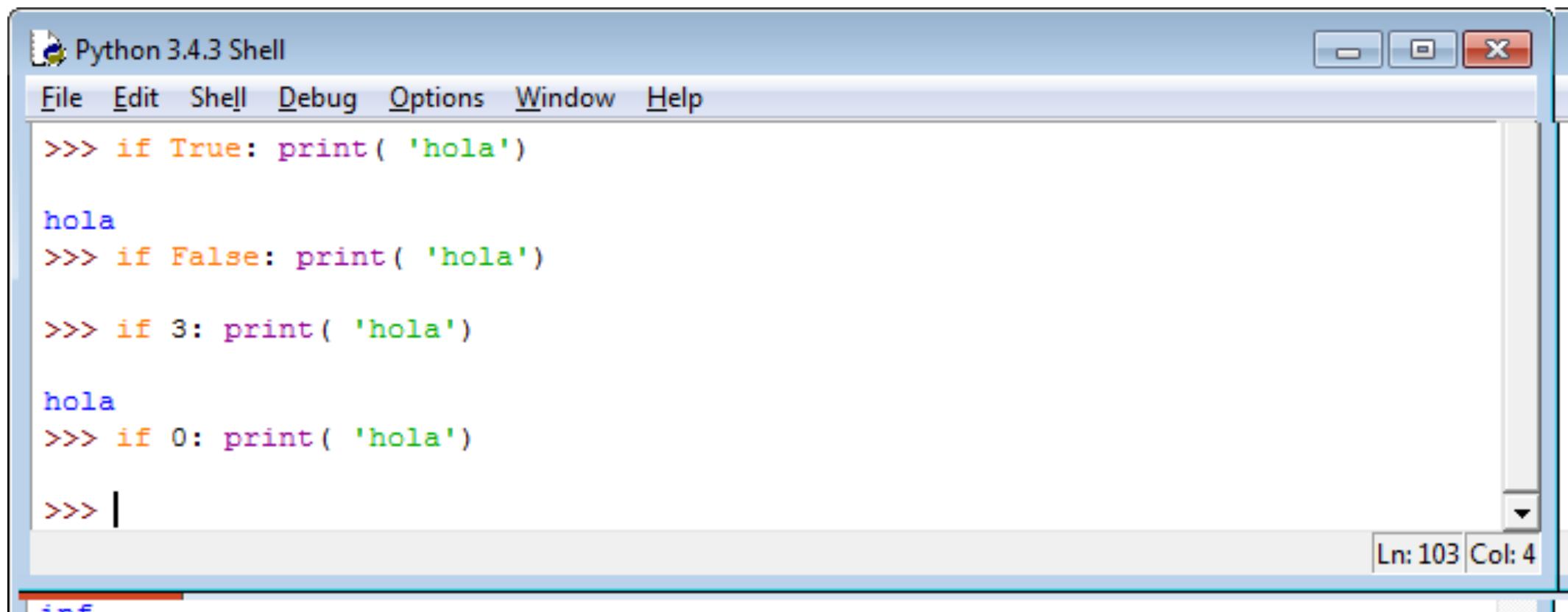
V or cualquierCosa = V

F and cualquierCosa = F

```
>>>True or False and False  
True  
>>> (True or False) and False  
False  
>>> True or (False and False)  
True
```

Probad con intérprete

```
>>> type (True)  
<class 'bool'>
```



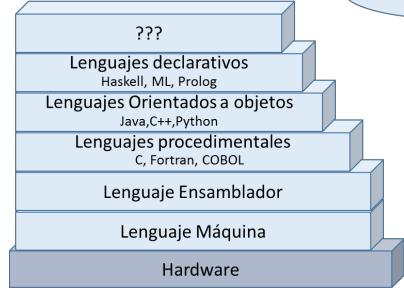
The screenshot shows a window titled "Python 3.4.3 Shell". The menu bar includes File, Edit, Shell, Debug, Options, Window, and Help. The main area contains the following Python code and its output:

```
>>> if True: print( 'hola')  
  
hola  
>>> if False: print( 'hola')  
  
>>> if 3: print( 'hola')  
  
hola  
>>> if 0: print( 'hola')  
  
>>> |
```

The status bar at the bottom right indicates "Ln: 103 Col: 4".

Identificadores

Ya desde ensamblador



- Nombre simbólico a posición de memoria → variable y constante nominal
- Reglas de formación comunes en los lenguajes:
 - Empieza por letra {A..Z, a..z}
 - Siguientes {A..Z, a..z, 0..9, _}
 - Resto de caracteres prohibidos {espacio en blanco, ‘;’, ?, á, ñ,}
 - Excepto palabras reservadas: **Probad: and=3**

and	as	assert	break	class	continue	def	del	elif	else
except	finally	for	from	global	if	import	in	is	lambda
not	or	pass	raise	return	try	while	with	yield	

- Sensible a mayúsculas/minúsculas? **Probad: And=3**

Identificadores:

Convenios de legibilidad

- Significativo: ~~ejercicio1~~, ~~pepe~~, radio, hipotenusa.
- Sin caracteres especiales (por la costumbre en otros lenguajes): ~~año~~, anno, agno, ~~perímetro~~.
- Si dos palabras:
 - primera_segunda: fecha_nacimiento (preferido),
 - primeraSegunda: fechaNacimiento,
 - Manteniendo el criterio elegido.
- Para las variables empieza en minúscula: nombre, apellido, titulación
 - Excepto costumbre en mundo real: DNI, P*V=n*R*T
- Para las constantes nominales en mayúsculas: PI, NUM_COLUM, MAX_ALUMN

Lenguajes con fuertes control de tipo (no en Python)

- Sección declaración (/inicialización).
- Tipo fijo durante toda la ejecución.
- i,j,k,l,m,n ... para enteros no relevantes en mundo real

Expresión

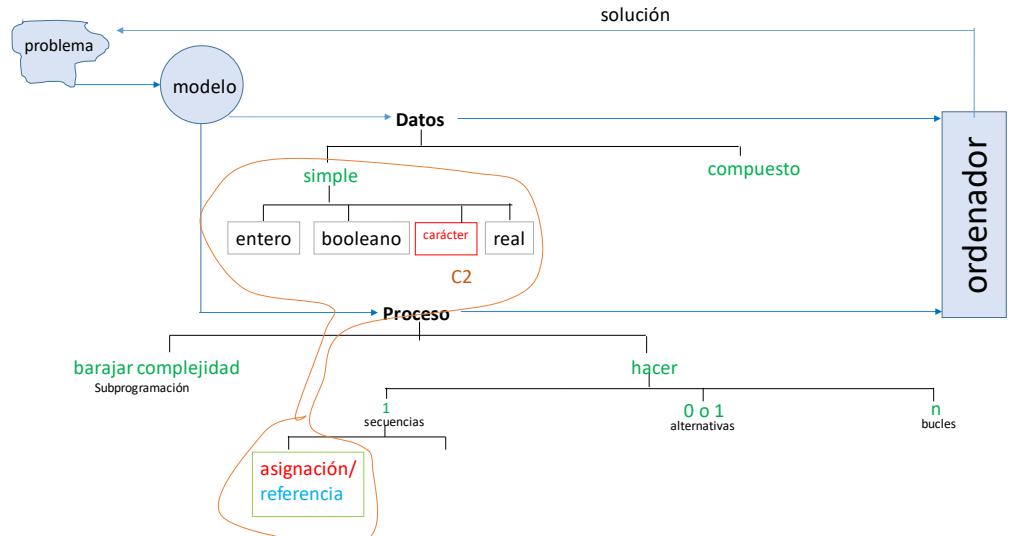
- Definiciones por intención/extensión

Por extensión:

- Una constante
- Una variable
- Una función
- Cualquier combinación de las anteriores, operadas con operadores adecuados
- Nada más es una expresión

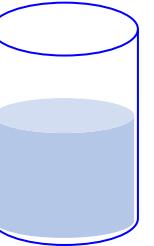
Referenciación

- Sintaxis: Identificador = expresión
- Proceso (1 vez):
- Ej: `area= base*altura/2.0`
 - Búsqueda en la tabla de símbolos
 - Lectura de las variables de la derecha
 - Escritura de las variables de la izquierda
- `i=i+1` !!!!



Asignación

otros lenguajes



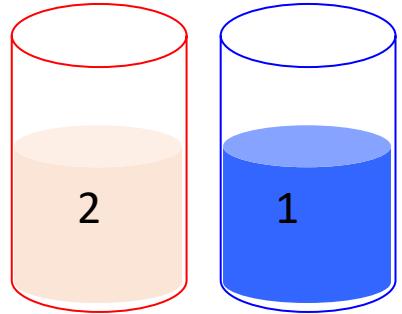
versus

Referenciación

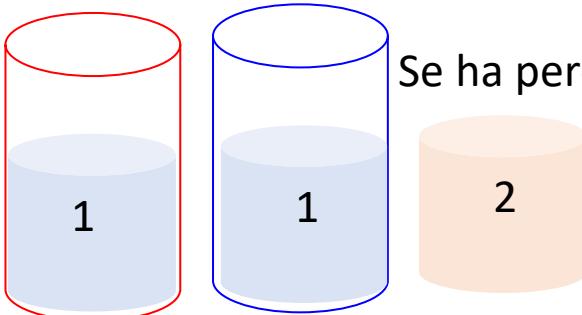
Python



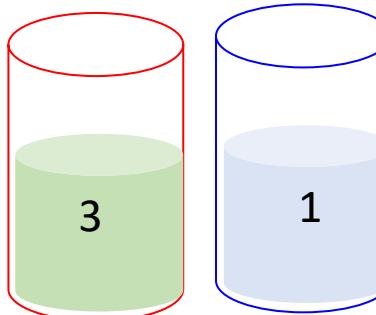
$a = 2$
 $b = 1$



$a = b$

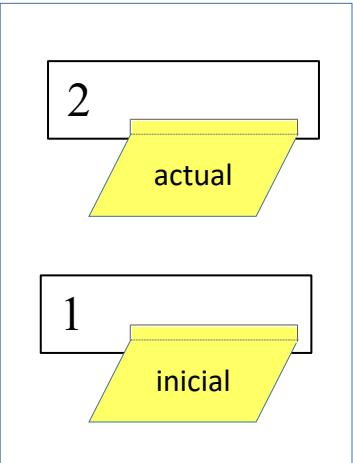


$a = b + 2$

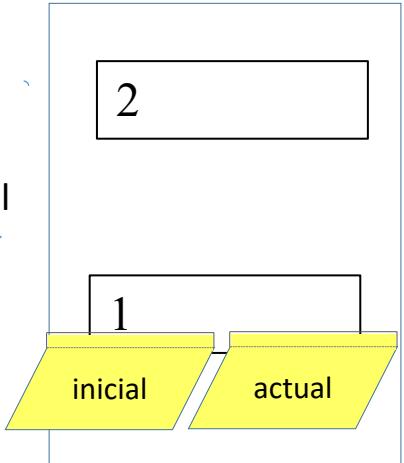


Asignación produce una copia → son posiciones independientes

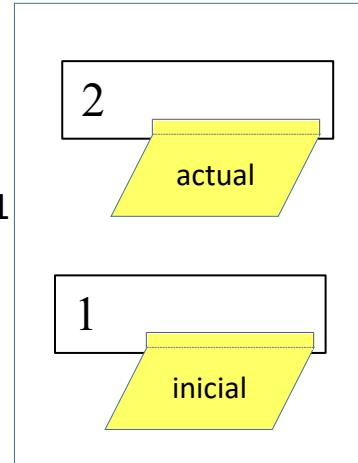
$a = 2$
 $b = 1$



$actual = inicial$



$actual = inicial + 1$

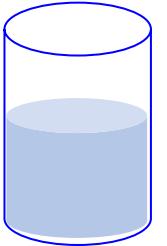


Referenciación es un pseudónimo → son la misma posición

Pero... parece que llegamos a lo mismo.

Asignación

otros lenguajes



versus

Referenciación

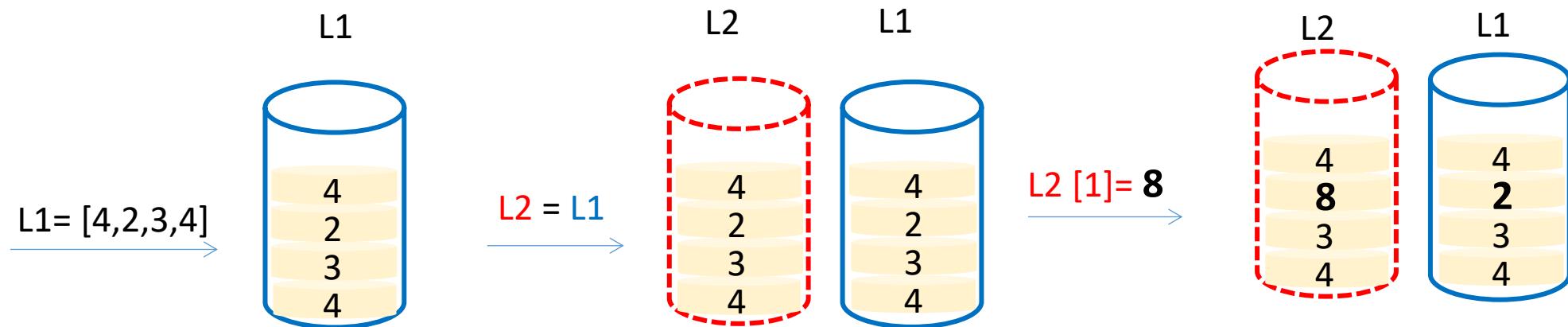
Python



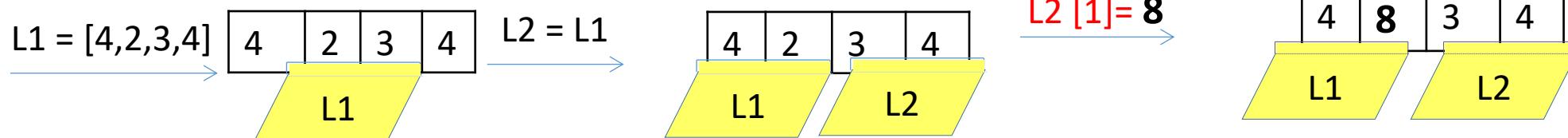
Implicaciones:

en **dato compuesto**, con componentes **modificables** sin re-asignar/re-referenciar el compuesto

Asignación:
Otros lenguajes



Referenciación:
Python

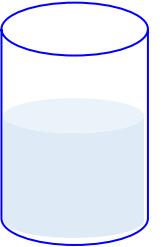


¿L1[2]?

Los tipos de datos compuestos, para los que Python da servicio de modificar elementos individuales, sin re-referenciar el dato completo, les llama **mutables**. Los tipos que sólo pueden cambiar su valor con referenciación los llama **inmutables**.

Asignación

otros lenguajes



versus

Referenciación

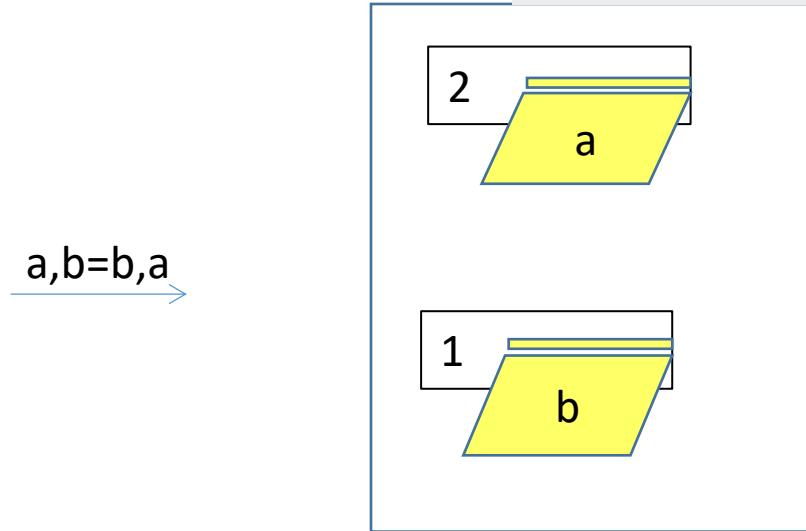
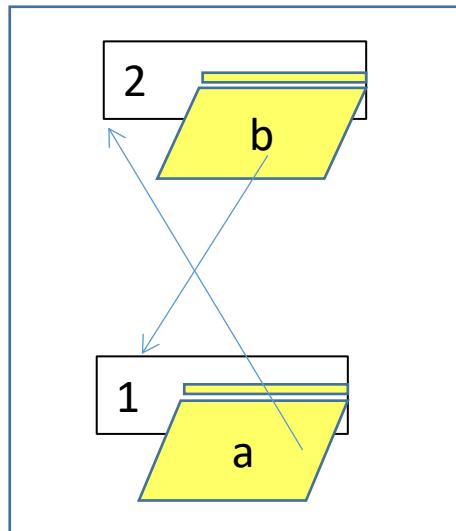
Python



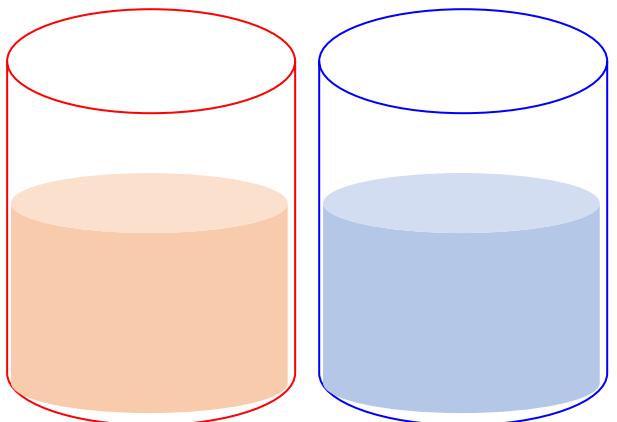
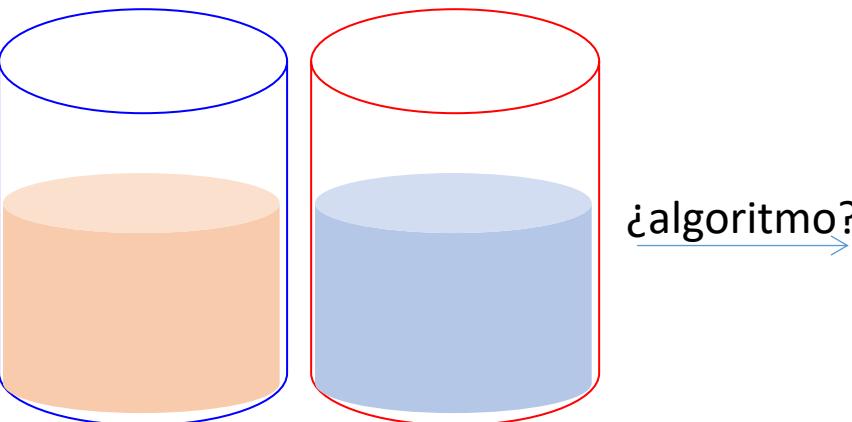
Implicaciones:

- b) intercambiar los valores de las variables

Referenciación: muy fácil



Asignación: Escribe el algoritmo, piénsalo con vasos



Resumen

- El **entorno interactivo** es una potente calculadora. Típico en lenguajes interpretados
- Los tipos simples en *Python* son **enteros, coma flotante y boolean**.
- Cada tipo de datos está definido sobre un dominio=conjunto concreto y finito de valores. Un intento de almacenar un valor fuera del dominio del tipo produce **desbordamiento**.
- int tiene una representación exacta (en su rango), float errores de **redondeo**.
- Cada tipo de datos posee un conjunto de operadores que tienen una **prioridad** asociada.
- Debido a los errores de redondeo y de desbordamiento, se pierde asociatividad y **comutatividad** de los operadores.
- Una **expresión** es una combinación de constantes, variables y funciones operados con los operadores correspondientes al tipo.
- **variable** = nombre simbólico que referencia a un dato (póstit). En otros lenguajes de programación es un nombre para un contenedor de datos.
- La operación de referenciación asocia el resultado de una expresión de un tipo a la variable que, por tanto, es del mismo tipo que la expresión referenciada.

U2: ERRORES FRECUENTES

- Desbordamiento.
- Redondeo.
- Pérdida de propiedad conmutativa: el orden importa.
- Convenio de nombres:
 - Constantes con mayúsculas
 - Variables minúsculas, compuestos pal1Pal2 o pal1_pal2
- Recomendado evitar expresiones mixtas entre tipos.
- Pep8 recomienda expresar const float con al menos un decimal.

Ejercicios

Complete la siguiente función.

```
def multiplos_5_recursivo(n):
    """ int -> int
    OBJ: número de múltiplos de 5 menores a n (en valor absoluto) """
    if n<0 : # código 1
        n *= -1
    if n < 5:
        cuantos= 0      # código 2
    else:
        if n %5 == 0: # código 3
            cuantos= 1 + multiplos_5_recursivo(_____) # código 4
        else:
            cuantos= _____ # código 5
    return cuantos

print(multiplos_5_recursivo(20))
```

Ejercicios

¿Qué hace el siguiente código?

```
def importe(patatas):  
    """int-->int  
PRE: patatas>=0"""  
    if patata<=1: aPagar=1  
    else: aPagar=patatas*importe(patatas-1)  
    return aPagar  
  
tomate=int(input(': '))  
print(importe(tomate))
```

Ejercicios

¿Qué hace el siguiente código?

```
def niIdea(n):  
    """int-->nada  
OBJ: ???  
PRE: ???  
"""  
    if n>=2:  
        niIdea(n//2)  
    print(n%2, end=' ')
```

Preguntas tipo examen Tema 2

- 1) La prelación de los operadores es diferente en distintos lenguajes de programación; operadores con igual prelación se ejecutan de derecha a izquierda o de izquierda a derecha.

- a) De saber: Indique en qué orden se ejecutan en Python los operadores `and` y `ord`
b) De razonar: Diseñe un conjunto de pruebas que le permitan determinar el orden de las operaciones `and` y `ord` en Python
c) tipo test: conocer

indique la salida de las siguientes expresiones

```
1>>> True or False and False  
2>>> (True or False) and False  
3>>> True or (False and False)  
4>>> False and False or True
```

- d) tipo test: razonar

Para comprobar si, en un determinado lenguaje, los operadores `and` y `ord` tienen la misma prioridad y en su caso si se ejecutan por la derecha, se probaron las expresiones:

```
1>>> True or False and False  
2>>> (True or False) and False  
3>>> True or (False and False)  
4>>> False and False or True  
5>>> False or True and False  
6>>> False and True or False
```

- a) 1,2,y 3 son suficientes
b) Se requieren 1,2,3,y 5
c) Se requieren 1,2,3,y 4
d) Se requieren 1,2, 5 y 6

- 2) La prelación de los operadores es diferente en distintos lenguajes de programación, operadores con igual prelación se pueden ejecutar de derecha a izquierda o de izquierda a derecha.

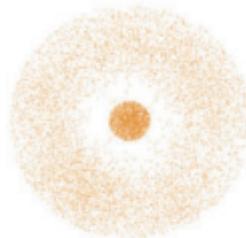
Diseñe un conjunto de pruebas para poner en evidencia el orden en que se ejecutan los operadores `*` y `/` en determinado lenguaje de programación

- 3) En un átomo hay mucho espacio libre. El radio del núcleo se puede aproximar por $R = 1,2 \times 10^{-15} \times A^{1/3}$ metros. Donde A es la suma de protones y neutrones. La corteza está muy alejada del núcleo, como se muestra en la figura. Varias teorías atómicas aproximan la distancia entre las sucesivas órbitas electrónicas. Comparativamente, estas distancias son mucho menores que la distancia de cualquiera de ellas al núcleo. Se desea aproximar el radio del átomo. ¿Qué es más seguro?:

- Sumar las distancias interelectrónicas, y a ello sumarle la distancia del primero al núcleo y el propio radio del núcleo o, por el contrario,
- empezar por el radio e ir sumando en orden los sucesivos datos

Para centrar las ideas, incluso exagerando los datos le propongo comparar las expresiones

- a) $4.0e-12 + 3.0e-2 + (4.879e-18 + 5.794e-14 + 2.033e-14 + \dots)$
- b) $4.0e-12 + 3.0e-2 + 4.879e-18 + 5.794e-14 + 2.033e-14 + \dots$



TRABAJO PERSONAL

Trabajo personal 2.1. Un euro vale 166,386 de las antiguas pesetas. Pídele a Python que te indique cuántos euros son 1000 pesetas.

Trabajo personal 2.2. En el ejercicio anterior ¿has escrito $1000/166.386$? Pues has obtenido el resultado correcto, pero es preferible que te acostumbres a hacer la conversión explícita de tipos.

Trabajo personal 2.3. Pide al intérprete de Python que indique cuántas horas (completas) han transcurrido en 145 minutos. Puede que cuando hayas escuchado por primera vez “división entera” te haya parecido extraño, pero ¿te das cuenta de que es mucho más útil de lo que en principio parecía?

Trabajo personal 2.4. Si has estudiado 145 minutos y te preguntaran ¿cuánto tiempo has estudiado? contestarías 2 horas y 25 minutos. En el ejercicio anterior le pediste a Python que calculara las horas transcurridas. Ahora pídele los minutos.

Trabajo personal 2.5 (resuelto). Los profesores solemos utilizar dos decimales cuando trabajamos con las notas parciales de un alumno, pero en las actas se desea calificar con puntos enteros o medios (es decir: $0 / 0.5 / 1 / 1.5 / \dots / 9.5 / 10$). Combinando operaciones y funciones enteras y en coma flotante, encuentra una fórmula general que convierta una nota con dos decimales a la calificación correspondiente en el acta. Pista (aunque te parezca obvio, esta es la pista): 0.5 (que es lo que quiero conseguir) es la mitad de 1.

Tu propuesta debe funcionar para todos los casos de prueba que adjuntamos.

Nota	En acta
8,89	9,0
8,50	8,5

Trabajo personal 2.6. Durante la Guerra del Golfo (1991) un fallo del Patriot provocó la muerte de 28 soldados y 100 heridos. Un fallo en el diseño de Pentium provocó unas pérdidas de 475 millones de dólares y la credibilidad de la empresa fabricante: Intel. Investiga cuál fue la causa. Por ejemplo, en “los 20 desastres más famosos del software” (ver en [desastres]).

Trabajo personal 2.7 (resuelto). Para confirmar que has entendido la tabla de la precedencia, evalúa manualmente las expresiones y confirma tu predicción en el intérprete:

- $20//3+20//(7+-4)$
- $20%-3+5*2$
- $\text{round}(4.5/2.0+3.6)+2*4$

Trabajo personal 2.8. La conversión de grados Farenheit a Célsius viene dada por la fórmula: $c = \frac{F-32}{1,8}$

El papel arde a 451°F (por cierto, un libro muy interesante de Ray Bradbury). Calcula a cuántos grados Celsius arde el papel. Debe salirte 232,8°C y si no, revisa el orden de prelación de las operaciones.

Trabajo personal 2.9. La ley de los gases ideales afirma que en un recipiente rígido cerrado (de volumen V constante expresado en litros) que contiene un gas, al aumentar la presión (P en atmósferas) aumenta la temperatura (T en grados Kelvin) según la siguiente fórmula:

$$PV = nRT$$

siendo n el número de moles del gas y R la constante universal de los gases, cuyo valor es 0,082 atmósfera litro/Kmol.

Píde a Python que calcule la temperatura de 3 moles de gas, contenidos en un recipiente de 7 litros a 2 atmósferas de presión. Si no sale 56,9°K, revisa el orden de prelación de las operaciones.

Trabajo personal 2.10. Lee el número 18 de “20 desastres famosos relacionados con el software” [desastres]. Un tratamiento contra el cáncer que tuvo como consecuencia 8 personas muertas y 20 heridas de gravedad. Los responsables del software fueron acusados de asesinato. Comprueba que la causa fue un error en el orden en que se realizaron las

Trabajo personal 2.11. En el calendario gregoriano estableció que un año es bisiesto si es divisible entre 4, a menos que sea divisible entre 100. Sin embargo, si un año es divisible entre 100 y además es divisible entre 400, también resulta bisiesto. Pregúntale a Python si 2015 fue bisiesto.

Trabajo personal 2.12 (resuelto). Acertijo ¡vamos a jugar! En una mesa hay tres sombreros negros y dos blancos. Tres señores en fila india se ponen un sombrero al azar cada uno y sin mirar el color.

- El tercero de la fila ve el color de los sombreros de los otros dos. El segundo solo ve el color del sombrero del primero y el primero no ve a ninguno.
- Se les pregunta por el color de sus sombreros.
- El tercero contesta que no puede responder.
- A continuación, el segundo asegura que no puede responder. En ese momento, el primero dice el color del suyo.

¿Cuál es este color y qué lógica usó para saberlo?

Trabajo personal 2.13. Un prisionero está encerrado en una celda que tiene dos puertas, una conduce a la muerte y la otra a la libertad. Cada puerta está custodiada por un vigilante, el prisionero sabe que uno de ellos siempre dice la verdad, y el otro siempre miente, pero no sabe cuál. Los vigilantes se conocen entre sí. Para elegir la puerta por la que pasará solo puede hacer una pregunta a uno de los vigilantes ¿Qué pregunta ha de hacer?

Trabajo personal 2.14. Un polinomio de grado m es la suma de $m+1$ monomios. Un monomio está formado por un coeficiente y una incógnita elevada a un exponente. El exponente de los monomios va desde m, hasta 0, decrementando de uno en uno.

Evalúa un polinomio de grado 4 con coeficientes (1.5, 43.0, 1/5, 23.2, 1) para el valor de la incógnita 0,028497. El objetivo de este ejercicio es que percibas que el uso de variables en Python evita errores de tecleo, ahorra tiempo y favorece la legibilidad. Valor esperado: 1.66.

Trabajo personal 2.15. De los siguientes identificadores indica cuales son inválidos y porqué. Despúes pruébalo en el editor.

miVariable	MIVARIABLE	mi-variable	mi variable
estaVacio?	estáVacío	estaVacio	7dias

Trabajo personal 2.16 (resuelto). La capacidad de elegir identificadores adecuados es fundamental en la construcción del pensamiento computacional, para practicarla, encuentra un nombre para describir de forma global:

1. Al par de temperaturas máxima y mínima de una serie de mediciones.
2. La lista lunes, martes, miércoles, jueves, viernes, sábado, domingo.
3. La lista lunes, martes, miércoles, jueves, viernes.
4. Los números tales que $n \% 2 == 0$.
5. Los números 5, 10, 15, 20, 25, 30...
6. La secuencia rojo, naranja, amarillo, verde, azul, añil y violeta.
7. La lista norte, sur, este y oeste

Propuesta solución:

1) **solución y justificación b)**

```
1>>> True or False and False
True
2>>> (True or False) and False
False
3>>> True or (False and False)
True
4>>> False and False or True
True
```

- 1 si tuvieran igual prioridad y el orden fuera de izq a der daría lo mismo que 2
4 si igual prioridad y de der a izq daría False
comparando las pruebas 2 y 3 con 1 permiten afirmar que and tiene prioridad sobre ord

d)

1 si tuvieran igual prioridad y el orden fuera de izq a der daría lo mismo que 2

4 si igual prioridad y de der a izq daría False

comparando las pruebas 2 y 3 con 1 permiten afirmar que and tiene prioridad sobre ord

2) Por ejemplo

$8.0 / 3.0 * 4.0$

$(8.0 / 3.0) * 4.0$

$8.0 / (3.0 * 4.0)$

$8.0 * 3.0 / 4.0$

Referencias

[desastres] <http://www.variablenotfound.com/2008/11/>.

Resolución de problemas para ingenieros con Python estructurado

U3

Programas secuenciales

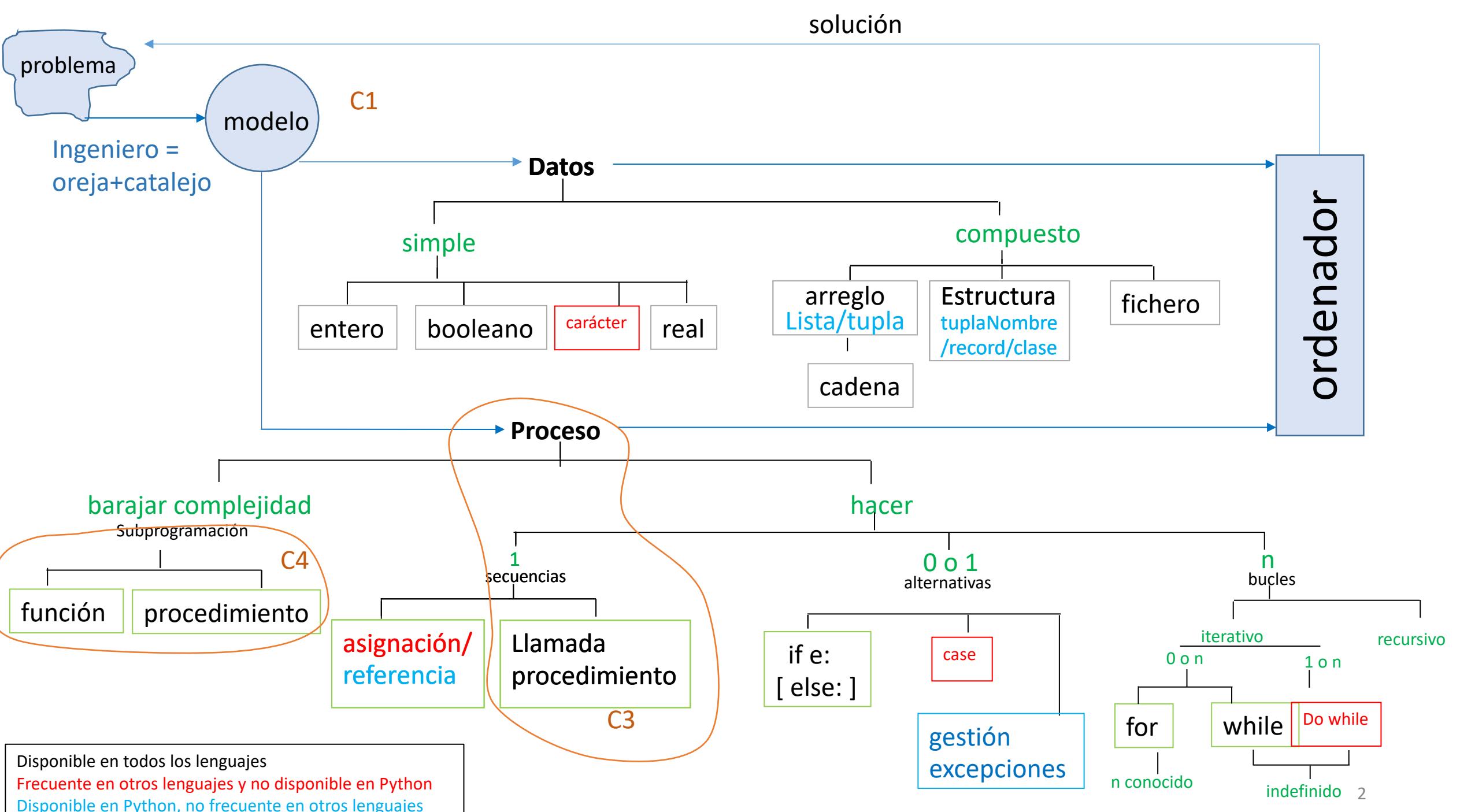
La interfaz de usuario

Rosalía Peña



Universidad
de Alcalá





Programas secuenciales

- | | |
|--|-----------------------------------|
| 3.1 Programa y algoritmo | Ya visto |
| 3.2 El entorno de edición |lab |
| 3.2.1 Estructura de un programa | |
| 3.3 La interfaz de usuario | |
| 3.3.1 Control de la salida | |
| 3.3.2 Captura de datos | |
| 3.3.3 Escritura con patrones de formato | |
| 3.5 Factores que afectan a la legibilidad del software | |
| 3.5.1 Elección de identificadores | |
| 3.5.2 Barajar la complejidad: subprogramación | |
| 3.5.3 Elegimos la programación estructurada y modular | |
| 3.5.4 Comentarios en un programa | |
| 3.5.5 Otras consideraciones | |
| 3.6 Documentación técnica y de usuario | Estudio personal del alumno |
| 3.6.1 Diagrama de contexto | |

Estructura de un programa en ≠ lenguajes

documentación

importación *

declaración de constantes

declaración de tipos*

declaración de variables (en otros lenguajes)

declaración de procedimientos y funciones*

con sus secciones a modo fractal *

cuerpo ejecutivo

entradas

proceso

salidas

* Aun no estudiado

Documentación y de

- **Doc. al principio de la pieza**

"""Objetivo: El objetivo del presente programa es calcular el perímetro y área de una figura geométrica denominada círculo, como son las propiedades ampliamente conocidas como perímetro y área.

"""OBJ: perímetro y área de un círculo cuyo radio es r.

- **Doc. cuando hay algo complejo**

```
a=2+3      #a se iguala a la suma de dos y tres
print(a)    #imprime a
```

- **Desactivar código**

```
*****  
* PROGRAMA: geoCirculo.py  
* Objetivo: perímetro y área de círculo de radio r.  
* Entradas: radio  
* Salidas: área, perímetro  
*  
* Autor: Francisco Pérez  
* Fecha: 20/03/2015  
* Creative Commons: Reconocimiento-NoComercial-  
*****
```

```
...
""" PROBADOR """
a=-3
print ('el valor absoluto de', a, 'es', abs(a))
...
y=abs(a)
```

La interfaz de usuario: Salida (estilo Pascal)

Procedimiento `print(expre [,expre]n [,sep=str][,end=str]` → nada

Polimorfismo: número variable de parámetros

<code>>>> print(1,2,3)</code> 1 2 3	Por defecto —
<code>>>> print(1,2,3, sep=', .. ')</code> 1, .. 2, .. 3	Por una coma seguida de dos blancos
<code>>>> print(1,2,3, sep=' - ')</code> 1-2-3	Por guiones
<code>>>> print(1,2,3, sep='\t')</code> 1 2 3	Por tabulador
<code>>>> print(1,2,3, sep=' ')</code> 123	Sin separación. Entre las comillas no hay nada
<code>>>> print(1,2,3, sep='\n')</code> 1 2 3	Por un salto de línea

Argumento `end`: cadena a la finalización. Por defecto, salta línea .Cadena vacía si no hacer nada

La interfaz de usuario: Salida (estilo C)

Patrón	Tipo de dato
%[n] d	entero
%[n].[d] f	coma flotante
%[n]s	cadena de caracteres

```
El gato tiene 4 patas y su peso medio es de 4.500000 kg  
=>>> |  
  
El gato tiene 4 patas y su peso medio es de 4.50 kg  
=>>>
```

```
animal = 'gato'  
patas = 4  
pesoM = 4.5  
print('El %s tiene %d patas y su peso medio es de %f kg'% (animal, patas, pesoM))
```

```
print('El %s tiene %d patas y su peso medio es de %.2f kg'% (animal, patas, pesoM))
```

```
print('          ANIMAL              PATAS      PES  
print('          =====              =====      ===  
print('%12s %17d %17.2f'% ('gato', 4, 4.5))  
print('%12s %17d %17.2f'% ('elefante', 4, 6000.))  
print('%12s %17d %17.2f'% ('araña', 8, 0.058))  
print('%12s %17d %17.2f'% ('ciempies', 46, 0.15))
```

```
          ANIMAL              PATAS      PESO MEDIO  
          =====              =====      ======  
          gato                4           4.50  
elefante            4           6000.00  
araña               8           0.06  
ciempies            46          0.15  
>>>
```

La interfaz de usuario: Captura

¿cómo te llamas? _María
hola María

Función **input(msg)**

str-->str

```
"""Saludo personalizado"""
nombre=input('¿Como te llamas? ')
print('hola '+nombre)
```

```
Calcula área de círculo de radio 1
"""area circulo"""
PI=3.1416
r=1 #cm
print('el área del círculo de radio',r,'es:', PI*r**2,'cm2' )
```

GENERALIZA para que pide el radio y devuelve el área del círculo

```
"""area circulo"""
PI=3.1416
r=float(input('radio en cm?:'))
print('el área del círculo de radio',r,'es:', PI*r**2,'cm2' )
```

Usuario introduce 1,5 en vez de 1.5

radio en cm?:1,5

Traceback...

ValueError: could not convert string to float: '1,5'

OJO: Siempre que se pida a un usuario un dato “no str” hay que hacer gestión de excepciones.

Gestión de excepciones:

Devuelve control a programador tras error de ejecución

```
try:  
    bloque1  
{except [tipoError]:  
    bloquej }n
```

tipoError= **ZeroDivisionError/ValueError/RuntimeError/TypeError/NameError...**

U3. ERRORES FRECUENTES

- No mantener el orden de las secciones de un programa.

documentación

importación*

declaración de tipos*

declaración de constantes

declaración de variables*

declaración de procedimientos y funciones*

con sus secciones a modo fractal *

cuerpo ejecutivo

entradas

proceso

salidas

- No documentar el archivo.py.
- Toda solicitud a usuario debe llevar mensaje conciso.
- Toda salida debe quedar documentada para el usuario.

* Aun no estudiado

Trabajo personal:

Pide al intérprete de Python que indique cuántas horas (completas) han transcurrido en 145 minutos

¿Cuántos minutos?

Trabajo personal 2.5 (resuelto). Los profesores solemos utilizar dos decimales cuando trabajamos con las notas parciales de un alumno, pero en las actas se desea calificar con puntos enteros o medios (es decir: 0 / 0.5 / 1 / 1.5 /... / 9.5 /10). Combinando operaciones y funciones enteras y en coma flotante, encuentra una fórmula general que convierta una nota con dos decimales a la calificación correspondiente en el acta. Pista (aunque te parezca obvio, esta es la pista): 0.5 (que es lo que quiero conseguir) es la mitad de 1.

Tu propuesta debe funcionar para todos los casos de prueba que adjuntamos.

Nota	En acta
8,89	9,0
8,50	8,5
8,45	8,5
8,24	8,0

Trabajo personal: ASCII Art

Se llama Arte ASCII a la habilidad de hacer dibujos por composición de caracteres.

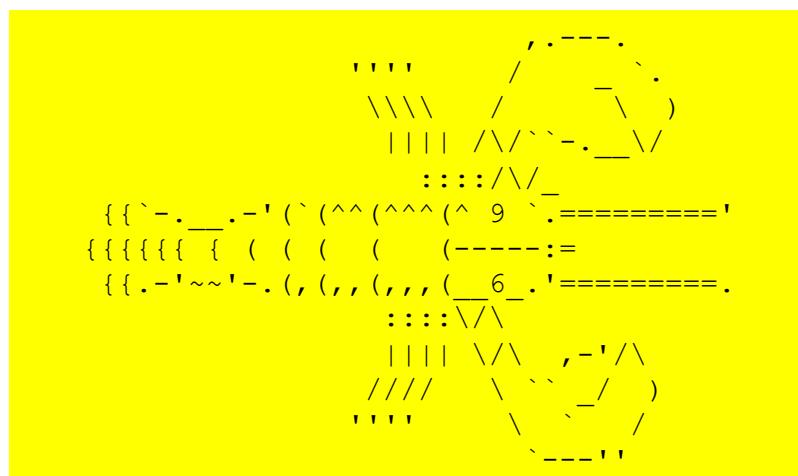
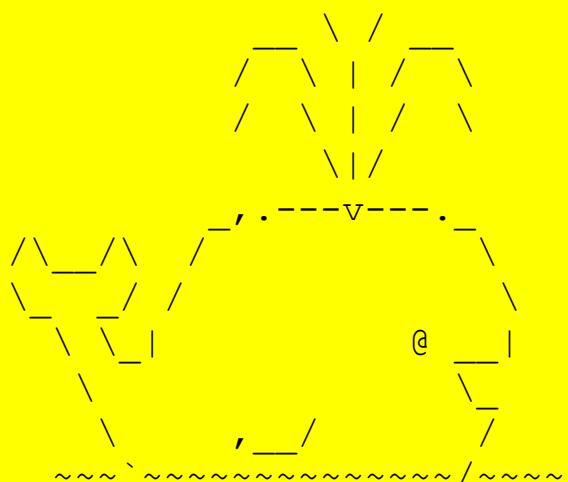
El siguiente programa imprime una gran letra E:

```
"""\t\t\tE GORDA\t\t\t"""
print('EEEEEE')
print('E')
print('EEE')
print('E')
print('EEEEEE')
```

Construye un programa que reproduzca el siguiente patito:



Un poco más difícil, puedes afrontar:



Dejaremos la chica para otro momento ;-)

Trabajo personal:

Trabajo personal. En la pantalla de interfaz calcula los minutos que hay en 3 horas, 20 min, 30 segundos.

Trabajo personal. Haz un programa que pida cuantas horas, y minutos hay en una cantidad de tiempo indicada por el usuario en minutos.

Trabajo personal 3.6. Flexibiliza el ejercicio anterior de modo que pase minutos a días, horas, minutos, de modo que el usuario introduzca en cada ejecución los datos por teclado.

Trabajo personal 3.7. Haz un programa que indique los días, horas, minutos y segundos que hay una cantidad de segundos introducida por el usuario (desprecia las fracciones de segundo).

Trabajo personal 3.8 (resuelto). Modifica `geometriaCirculo_v3` para que proporcione solamente dos decimales en la salida.

Trabajo personal 3.9. Escribe un programa que pase una temperatura expresada en grados Fahrenheit a grados Celsius.

Trabajo personal:

Un euro vale 166,386 de las antiguas pesetas. Píde a Python que indique cuántos euros son 1000 pesetas.

Trabajo personal 3.10 ¿Cómo llamarías a un programa que solicita al usuario una cantidad expresada en dólares e imprime por pantalla su equivalente en euros? Elige los nombres más adecuados para las variables involucradas ¿algún dato es una constante?, ¿cuál será el nombre adecuado para ella?

Construye la frase más corta que describa el objetivo del programa. Construye la frase más corta que describa al usuario lo que se desea que introduzca. Documéntate para saber con cuantos decimales debes expresar la salida si el programa se va a usar para calcular el valor de las acciones de una empresa ¿estás seguro que la fuente a la que has recurrido para buscar esta información es la adecuada?

Ahora estás en disposición de construir el programa. Adelante.

Resolución de problemas para ingenieros
con Python estructurado

U4

Subprogramación

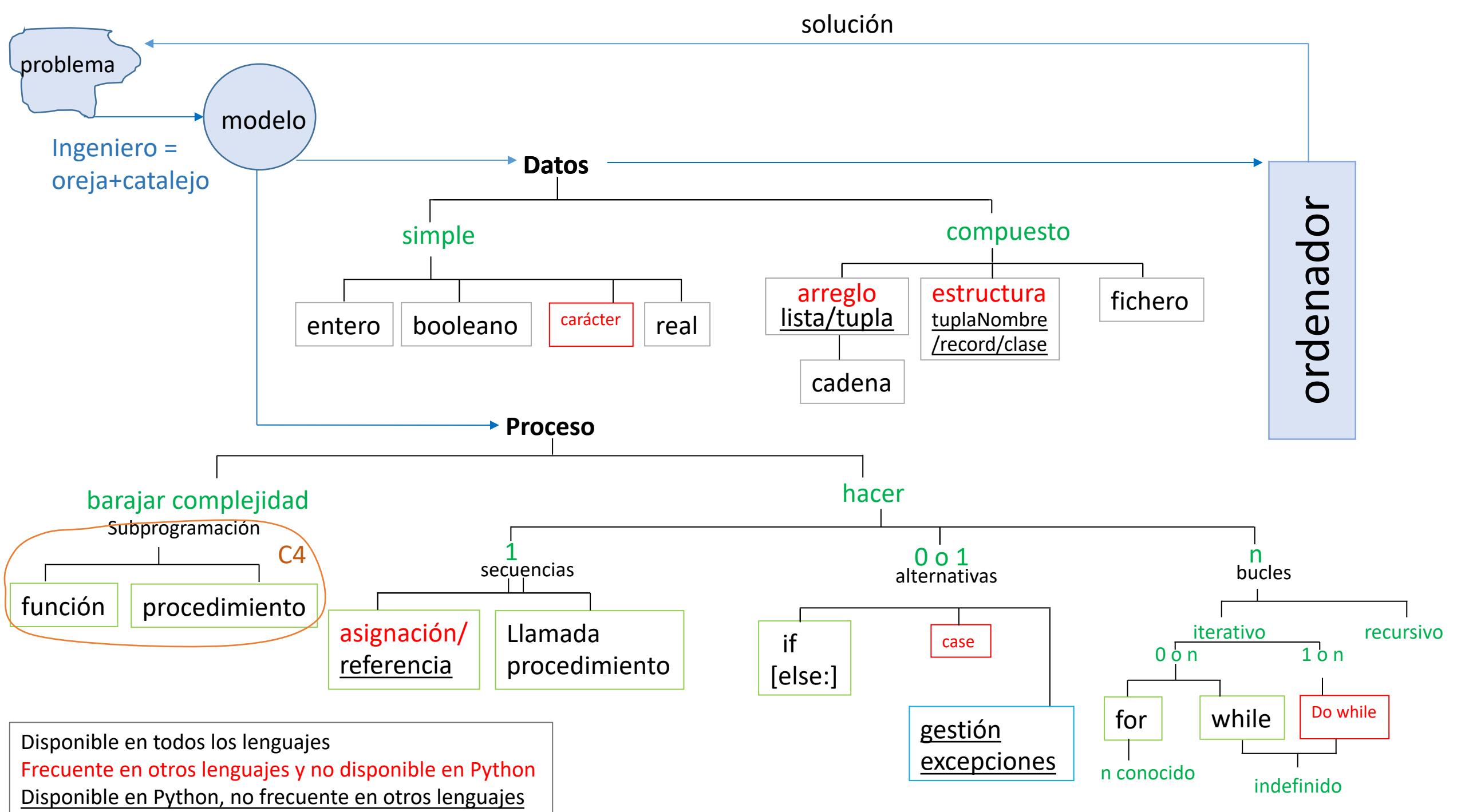
Probablemente el tema más importante del curso
iiiA por el reuso!!!

Rosalía Peña

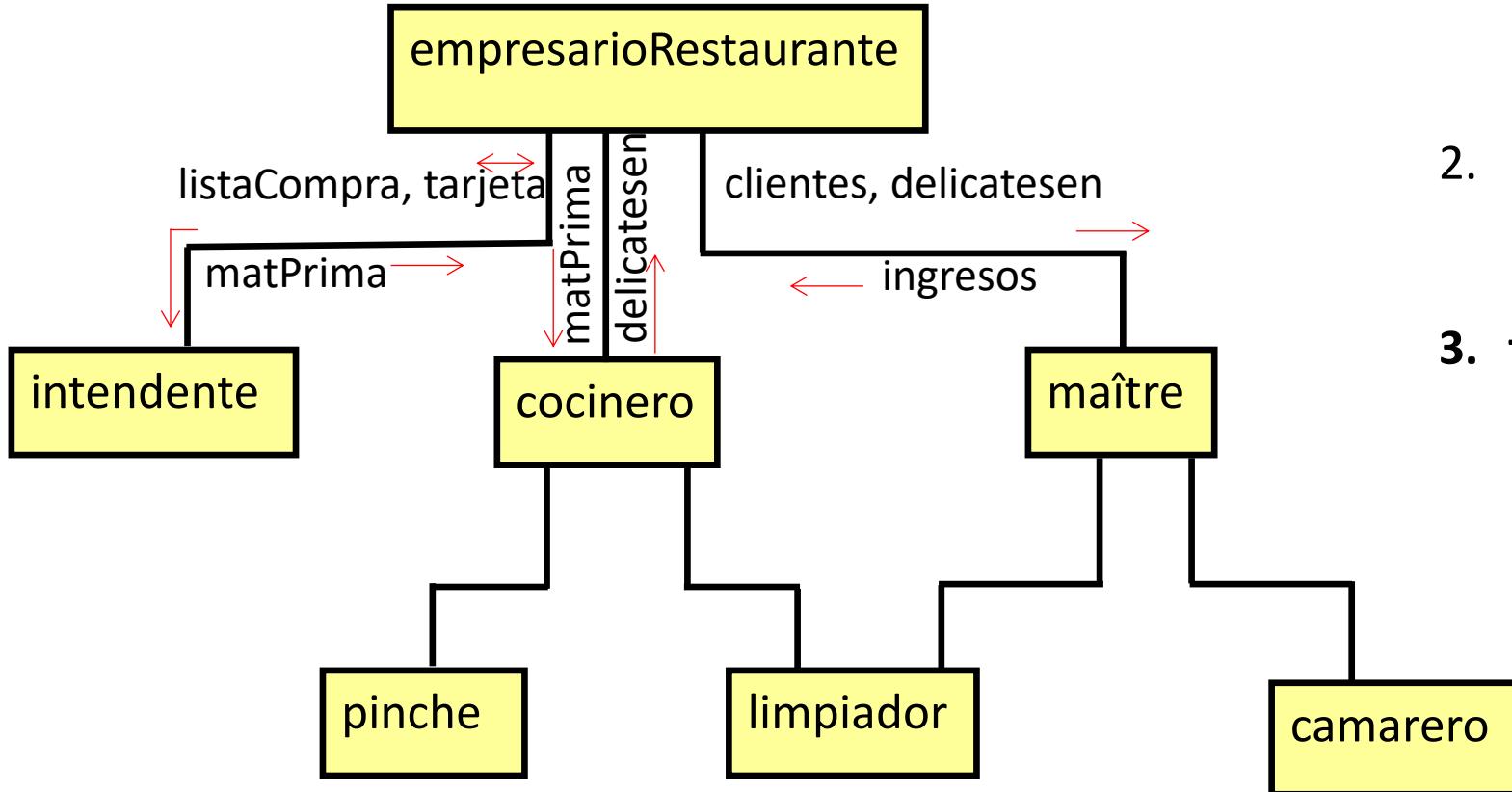


Universidad
de Alcalá





Divide y vencerás



BENEFICIOS

1. **permite reutilizar código**
 - en el propio programa
 - en futuros programas
2. **resolver problemas complejos (7±3)**
 - pequeño cada vez
 - **reparto de tareas** (equipo)
3. **facilita mantenimiento**
 - **legibilidad**
 - **probar piezas independientes**
 - evita **redundancia**

Argumentos de entrada/de salida/de entrada-salida

Descomposición funcional= diseñar definiendo módulos independientes que cooperan en la resolución de un problema.

Definición del subprograma= Patrón comportamiento=contrato → Parámetros formales/reales

Llamada a subprogramas

Prog. Estructurada

- Función: : devuelve **un valor**

```
nombre = input('introduzca nombre ') # en expresión
```

- Procedimiento : hace

```
print ('hola', nombre) # como una instrucción
```

Prog. OO

- Método: es el objeto el que lanza la acción

```
>>> cad = 'Hola'  
>>> cad.upper()  
'HOLA'
```

Llamada a subprogramas de bibliotecas

¡¡Python es mucho Python!!: Núcleo + bibliotecas internas +¡¡¡ bibliotecas externas!!!!

Biblioteca o módulo: Conjunto de subprogramas y constantes

- Incorpora un subprograma (o dato) a mi programa

```
>>> sqrt(4)  
...Error:name 'sqrt'  
>>>pi
```

```
>>> from math import pi, sqrt  
>>> pi  
3.141592653589793
```

```
>>> sqrt(4)  
2.0
```

- Déjame accesible toda la biblioteca

```
>>> import math
```

- Ayuda de biblioteca:

```
>>> help(math)
```

- Ayuda de subprograma:

```
>>> sqrt(
```

```
>>> cos(pi)  
...Error: name 'cos' is not defined
```

```
>>> math.cos(pi)  
-1
```

Espacios de nombres
Notación punto

- Módulos que contienen módulos:

```
>>> import datetime  
>>> datetime.date.today()
```

Crea y usa tu propios subprogramas



SINTAXIS: **def** nombre_subprograma (lista_argumentos):
 cuerpo

- Argumentos: lista vacía, E y E/S
 - Función: return expresión (arg S y E/S)
 - Identificadores:

Criterios para nombres

Legibilidad/mantenibilidad

- Documentación: legibilidad (+ayuda en Python)

""”Lo que entra → [lo que sale/nada]

OBJ:

PRE: si la hay

Legibilidad/mantenibilidad/reuso

11

- Desarrollo incremental

- probador
 - deshabilita probador pero mantenlo

Robustez Mantenibilidad

- Uso de tus subprogramas

- ¿Cómo funciona prog con subprog? Pythontutor →

¿Cómo funciona un programa con subprogramas?

* PROGRAMA: centrar

*

*OBJ: probar el subprograma que centra un rótulo en pantalla

*

nColum=76 #tamaño actual de la pantalla

```
def centrarRotulo (rotulo):
```

"""string--> nada

OBJ: centra rótulo, subrayado con signos =, +linea encima y debajo

PRE: tam(rotulo)<=nColumn en uso global

"""

tam=len(rotulo)

lado=(nColum-tam)//2-1

print ()

print(' '*lado,rotulo)

print(' '*lado,'='*tam)

print()

#Probador

frase = 'Don Quijote de la Mancha'

centrarRotulo(frase) # distinto nombre

centrarRotulo('Cervantes') # constante

rotulo = 'El famoso hidalgo don Quijote de la Mancha'# mismo nombre

centrarRotulo(rotulo)

<http://www.pythontutor.com/visualize.html#mode=edit>

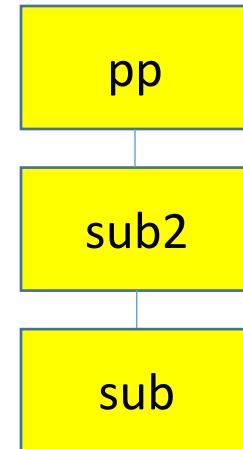


Crea y usa tu propias bibliotecas

- Agrupa subprogramas y constantes
- Recuerda REUSOOOOOO
 - Organiza tu disco+copias de seguridad
 - Criterio de agrupamiento: por tipo de datos (Ej:libFecha),
por tipo de acciones (Ej: libEstadistica)
- Nombre
- Documentación: ayuda
 - Objetivo general
 - Cabeceras de subprogramas y constantes

Ámbito, vigencia, visibilidad

- Ámbito de una variable es la pieza de código en que se ha creado.
- Vigencia es el tiempo en que la variable vive en memoria
- Visibilidad es la zona de código en que puede ser usada


EJEMPLO:

```
def sub(y):
    z=3
    return y+z
```

```
def sub2(a):
    c=22
    return a+z +sub(z)
z=4
y=3
a=sub2(3)
b=2
```

Ámbito
 sub y, z
 sub2 a,c,
 prog. principal
 z,y,a,b

Vigencia
 sub y, z, a,c, z,y,a,b
 sub2 a,c, z,y,a,b
 prog. principal z,y,a,b

Visibilidad
 sub y, z, a,c,b
 sub2 a,c, z,y,b
 prog. principal z,y,a,b

Igual que con variables, ocurre con subprogramas: Ámbito, vigencia, visibilidad

Ámbito, vigencia, visibilidad

- Consulta global: Aviso en PRE:
- Modificación global “inadvertido” en otros lenguajes. **global** en Python
- Parámetros formales son del subprograma
- Parámetros reales son del llamante
- Ambito, vigencia, visibilidad de **subprogramas** (Igual que con variables)
- Calidad:
 - Cohesión alta: el nombre sale solo
 - Acoplamiento bajo: independencia, autocontenido...

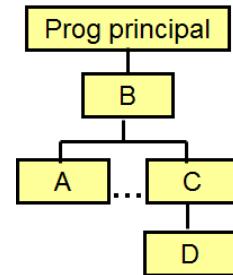
Anidamiento de subprogramas

pruSubAnidados.py

```
1      """***** prueba de subprogramas anidados *****
2
3 def A(a):
4     print('\t'*2,a,'En A')
5     return a*2
6
7 def B(b):
8     def C (c,d):
9         def D():
10            print('\t'*3,'D saluda')
11            D()
12            print('\t'*2,c,d,'En C')
13            return c*2
14
15 e = A(b)
16
17 j = 3
18 print('en pp')
19 B(j)
20 print('en pp')
```

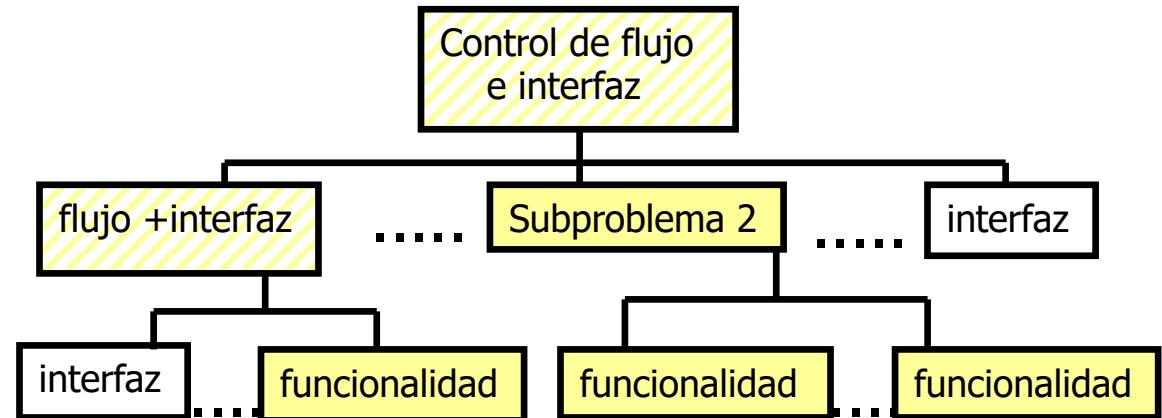
encapsulados

Descomposición funcional



Resumen: Metodología de la subprogramación

- 1. Identifica si es necesario un subprograma.**
- 2. Define clara y concisamente la tarea ¿qué datos necesitas para realizarla?.**
- 3. Identifica el tipo de subprograma.**
- 4. Nombra el subprograma.**
- 5. Documenta: arg E/S, OBJ, PRE.**
- 6. Diseña los casos de prueba.**
- 7. Localiza excepciones.**
- 8. Reutiliza.**
- 9. Codifica.**
- 10. Optimiza el código.**
- 11. Prueba.**
- 12. Organiza tus archivos.**



U4: Errores frecuentes

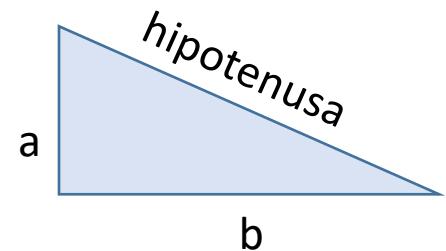
- Los subprogramas son independientes: El nombre de arg formales y reales no tiene ni que coincidir, ni que ser diferente.
- Uso global de constantes está permitido, pero hay que indicarlo en la PRE:. Modificación global de variables no permitido ([Python, ni siquiera deja modificar](#)).
- Duplicar una constante compleja en cada subprograma que la usa.
- Empezar a codificar (paso 9) sin haber hecho los 8 pasos anteriores

U4: Errores frecuentes

Error	Ejemplo erróneo	Correcto	Explicación
No documentar subprograma	def subprogr(arg) sin doc cuerpo	tiposEntrada—>Salida; OBJ: mezcla nombre subprogr con arg; PRE: (si la hay)	Imposibilita el reuso
Nombre inadecuado del subprograma	def ejercicio1 O def Javier_lopez	Procedimiento ↳ verbo acción Función booleana ↳ verbo de estado Función otro tipo ↳ sustantivo	Disminuye mantenibilidad y reuso
Llamar a un procedimiento, como si fuera una función	print(procedim(arg))	procedimient(arg)	La llamada a procedimiento es una instrucción. Hará su trabajo, pero imprimirá "None"
Llamar a una función como si fuera un procedimiento	funcion(arg)	print(función(arg)) o var=funcion(arg)+var2	La llamada a función ES UN VALOR. Hará su trabajo, pero el resultado será inaccesible.
Darle valor a un argumento de entrada antes de usarlo	def f(arg1, arg2): arg1=input('msj') arg2=expresión	Si ha de asignarle valor al entrar, no es argumento de entrada	¿para que lo has pasado?
Devolver una var a la que no se le ha dado valor en el cuerpo del subprograma	def f() cuerpoNoTocaVar return var	Todo arg de salida debe haber sido (o posibilidad de ser) modificado por el subprograma	¿puede ser un olvido?
Poner un return sin argumentos de salida	def f() cuerpo return (nada)	¿quizá querías hacer un procedimiento y no una función?	return quiere decir devolver ¿Qué devuelves?

Ejemplos:

Hipotenusa de un triángulo rectángulo



```
*****
```

Probador de hipotenusa

```
*****
```

```
def hipotenusa (a,b):  
    """float,float-->float  
OBJ: hipotenusa de triángulo rectángulo de lados a,b  
PRE: a,b >0  
from math import sqrt  
return sqrt(a**2+b**2)
```

```
"""PROBADOR"""  
print('hipotenusa =', hipotenusa (3.0,4.0))
```

La ayuda de contexto devuelve la documentación que hemos preparado en el subprograma

```
>>> hipotenusa(
```

```
(a, b)  
float,float-->float  
OBJ: hipotenusa de triángulo rectángulo de lados a,b  
PRE: a,b >0
```

Ejemplos:

En 1582 el calendario gregoriano estableció que un año es bisiesto si es divisible entre 4, a menos que sea divisible entre 100. Sin embargo, aunque un año sea divisible entre 100, si además es divisible entre 400, también es bisiesto.

bisiesto.py

```
1      **** Probador de esBisiesto ****
2
3  def esBisiesto (anno):
4      """int-->bool
5          OBJ: True si el año es bisiesto anno
6          PRE: año>=1582
7      return anno%4==0 and anno%100!=0 or anno%400==0
8
9      """ PROBADOR """
10     print(1955, esBisiesto(1955)) # no múltiplo de 4
11     print(1956, esBisiesto(1956)) # múltiplo de 4
12     print(1900, esBisiesto(1900)) # múltiplo de 100, pero no de 400
13     print(2000, esBisiesto(2000)) # múltiplo de 400
14
15     1955 False
16     1956 True
17     1900 False
18     2000 True
```

Ejemplos:

Calcula el área y el perímetro de un círculo de radio r

```
geometriaCirculo_v5.py
1 """*****
2 * Objetivo: prueba funciones área y perímetro *
3 *****
4 from math import pi
5
6 def area(r):
7     """float-->float
8         OBJ: área de círculo de radio r
9         PRE: r>=0
10        return pi*r**r
11
12    def perimetro (r):
13        """ float-->float
14            OBJ: perímetro de circunferencia de radio r
15            PRE: r>=0
16            return 2.0*pi*r
17
18 #PROBADOR
19 print ('área= %6.2f'%(area (1)), 'm2')
20 print ('perímetro=%6.2f'%(perimetro(1)), 'm')
```

```
***** probador de siglo *****
```

```
def siglo (anno):
```

```
    """ int-->int
```

```
OBJ: Siglo correspondiente a anno
```

```
PRE: anno <> 0
```

```
# return (anno-1)//100+1      o con 1 operación menos
```

```
# return (anno+99)//100
```

```
# pero, no hace bien los negati.
```

```
# anno//abs(anno)           #obtiene el signo de
```

```
# int(anno/abs(anno))       peor solución
```

```
# s = anno//abs(anno)*((anno+99)//100) esto no
```

```
# s = anno//abs(anno)*((abs(anno)+99)//100) se
```

```
s = anno//abs(anno)*(abs(anno)+99)//100
```

```
return s
```

```
"""PROBADOR""""
```

```
print('Siglo %3d debe dar -2' %siglo(-200))
```

```
print('Siglo %3d debe dar  2' %siglo(200))
```

```
print('Siglo %3d debe dar -3' %siglo(-201))
```

```
print('Siglo %3d debe dar  3' %siglo(-201))
```

is centenas
año 1 d.c.,
iglo II. Del

EN RESUMEN esta era la respuesta esperada

```
***** probador de siglo *****
```

```
def siglo (anno):
```

```
    """ int-->int
```

```
OBJ: Siglo correspondiente a anno
```

```
PRE: anno <> 0
```

```
s = anno//abs(anno)*(abs(anno)+99)//100
```

```
return s
```

"""PROBADOR""""

```
print('Siglo %3d debe dar -2' %siglo(-200))
```

```
print('Siglo %3d debe dar  2' %siglo(200))
```

```
print('Siglo %3d debe dar -3' %siglo(-201))
```

```
print('Siglo %3d debe dar  3' %siglo(-201))
```

Trabajo personal:

1.-Cuando construyamos programas para explotación mimaremos el aspecto de la entrada y la salida. El conjunto de ambas cosas se llama **interfaz de usuario**. Posiblemente pondremos al principio un rótulo, indicando al usuario qué aplicación está usando. El rótulo estará centrado en la pantalla y subrayado, dejando una línea en blanco por encima y otra por debajo. Por ejemplo:

Aplicación de combinatoria

=====

Trabajo personal 4.7 (resuelto). Flexibiliza el subprograma centrarRotulo (que está en la pag 7 de este documento) para que el signo con el que se subraye y el tamaño de la ventana puedan ser distintos en diferentes ejecuciones.

Trabajo personal:

2.-Puede resultar útil en un futuro un subprograma que pase una cantidad de euros a pesetas. 1€=166,386 pesetas. Hazlo y pruébalo.

3.-Vayamos con otro ejemplo: la ley de los gases ideales indica que el volumen V (en litros) ocupado por n moles de un gas a presión P (en atmósferas) y temperatura T (en grados Kelvin), responde a la siguiente fórmula:

$$V = nRT/P$$

siendo R la constante universal de los gases, cuyo valor es 0,082 atmósfera litro/Kmol. Hagamos un subprograma que calcule el volumen de un gas a partir de su presión y temperatura y probémoslo.

4.-Se dice que un gas está en condiciones ideales (o condiciones normales) cuando está a 1 atmósfera de presión y 0°C, es decir 273, 15ºK. Hagamos un subprograma que calcule el volumen de un gas en condiciones ideales. ¡¡¡No olvides reusar tu esfuerzo!!! Tu nuevo subprograma tendrá una línea de código. Probémoslo.

Traelo hecho a la siguiente sesión de lab, que te lo evaluaré.

5.-Python permite dar valores por defecto en la definición de un subprograma. Documéntate y estudia como se hace esto. Hazlo en el trabajo personal 3. Para ello, ten en cuenta que la presión es mucho mas estable que la temperatura. Ahora, ya no te haría falta el ejercicio 4.

Trabajo personal:

6.-Los ingleses utilizan unidades de medida diferentes del resto de Europa, grados Farenheit en vez de Célsius, libras en vez de kilos, pintas como unidad de volumen, millas como unidad de longitud, etc. Prepara una biblioteca que nos facilite dialogar con los ingleses. Una biblioteca no es más que una colección de subprogramas y constantes almacenados en un archivo ".py". Deja el probador de cada subprograma documentado tras cada subprograma ¿Cuál será el nombre de esta biblioteca? Usa los subprogramas desde otro módulo.

7.-Cuando tengas tu mejor versión de cada subprograma de esta biblioteca ve al foro de trabajo de la asignatura, Si no está subido:

crea un hilo para este subproblema y súbelo, solicitando mejoras.

Si está: Contrasta tu solución con la de tus compañeros y propón mejoras a las suyas.

La evaluación de esta semana será a partir de vuestras aportaciones en el foro

Trabajo personal: traed hecho en la siguiente sesión de laboratorio

8 Trabajo personal (resuelto en libro). Una fábrica funciona ininterrumpidamente (7días*24horas). Se desea calcular el tiempo, expresado en horas y minutos, que ha trabajado un empleado, sabiendo el momento de entrada y el de salida (expresados en horas y minutos). Un trabajador no puede trabajar más de 8 horas seguidas, de modo que el momento de entrada y el de salida, corresponden al mismo día (si entrada \leq que la salida) o a días consecutivos (en caso contrario). Caso de haber trabajado más de 8 horas, el programa dará una alerta.

Completa las salidas esperadas (valores de x e y) en los siguientes casos de prueba (**no lo programes aún**):

Caso prueba	11:00 11:15	12:33 20:33	12:30 14:15	23:00 2:13	23:55 2:15	6:15 19:30
Salida deseada	xh, ym	xh, ym	xh, ym	xh, ym	xh, ym	xh.ym ¡ALERTA!

9 Trabajo personal. El tiempo transcurrido entre horas es potencialmente reusable. Haz la interfaz del subprograma (cabecera), **pero no desarrolles el cuerpo hasta que no hayamos visto la instrucción if**. También puedes definir la cabecera del subprograma que hace la alerta, si el tiempo es mayor que un número de horas, pero no lo mezcles en el mismo subprograma. Cada subprograma debe resolver un solo problema, de lo contrario es menos reusable ¿Es responsabilidad del subprograma que las horas y minutos sean correctos, o por el contrario es una precondition?

Serán los primeros ejercicios que hagamos nuevos en la sesión

Ordenar 3 números

Diseña y justifica los **casos de prueba** del algoritmo que ordena 3 números de menor a mayor.

Casos de prueba:

3 valores pueden estar ordenados de 6 maneras. Hay que probar los 6 casos.

- 1, 2, 3
- 1, 3, 2
- 2, 3, 1
- 2, 1, 3
- 3, 1, 2
- 3, 2, 1

Ordenar 3 números

Pinta el diagrama de flujo del algoritmo que ordena 3 números de menor a mayor.

Usa los casos de prueba para confirmar que funciona correctamente. Y modifica el algoritmo si es necesario.

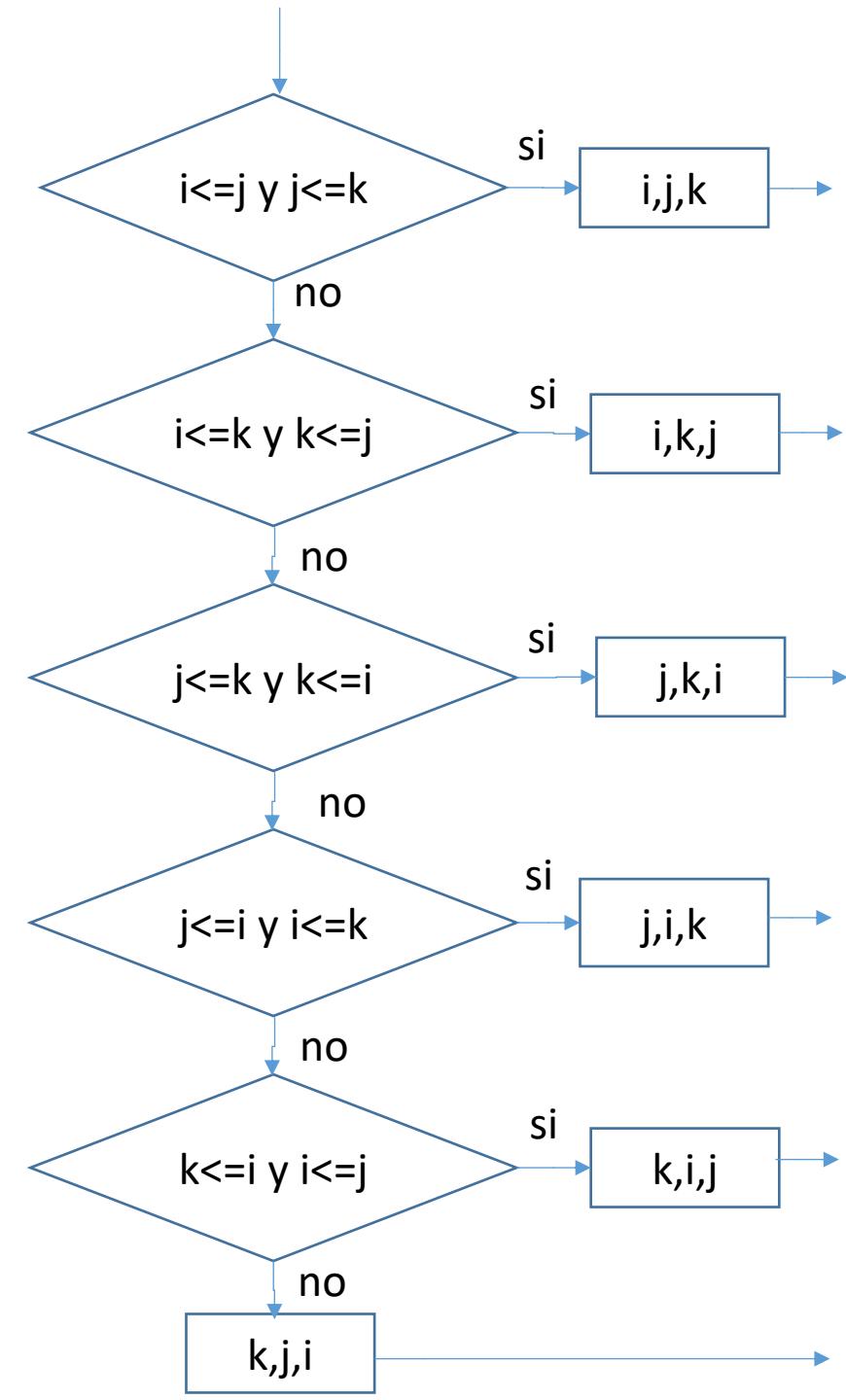
Complejidad ciclomática

$M = \text{Número de condiciones} + \text{Número de salidas de la pieza}$

- Cada operador de comparación 1 punto
- Cada operador booleano or y and 1 punto
- Cada operador not 2 puntos

Complejidad=21

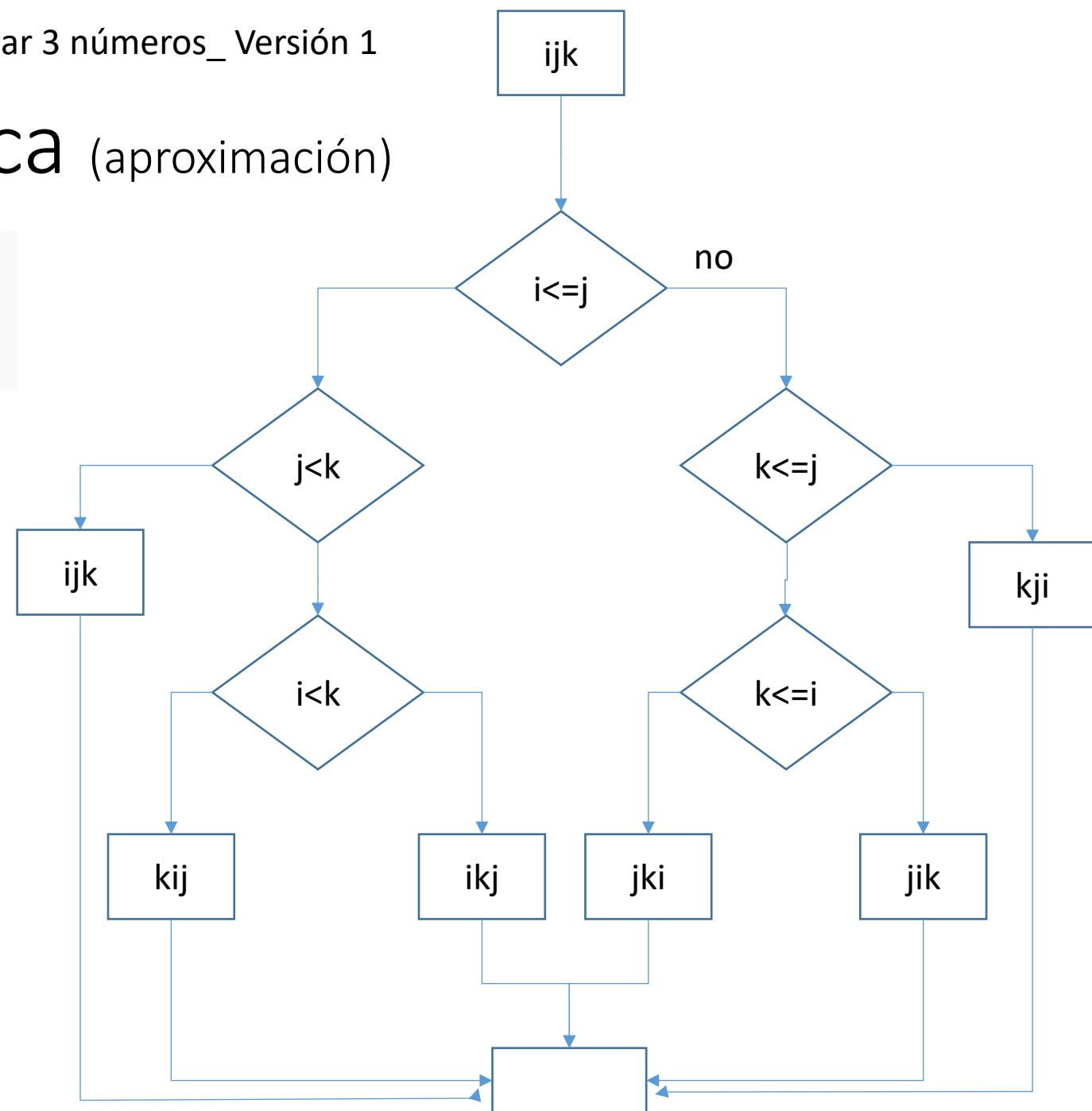
Casos de prueba
 1, 2, 3
 1, 3, 2
 2, 3, 1
 2, 1, 3
 3, 1, 2
 3, 2, 1



Complejidad ciclomática (aproximación)

$M = \text{Número de condiciones} +$
 $\text{Número de salidas de la pieza}$

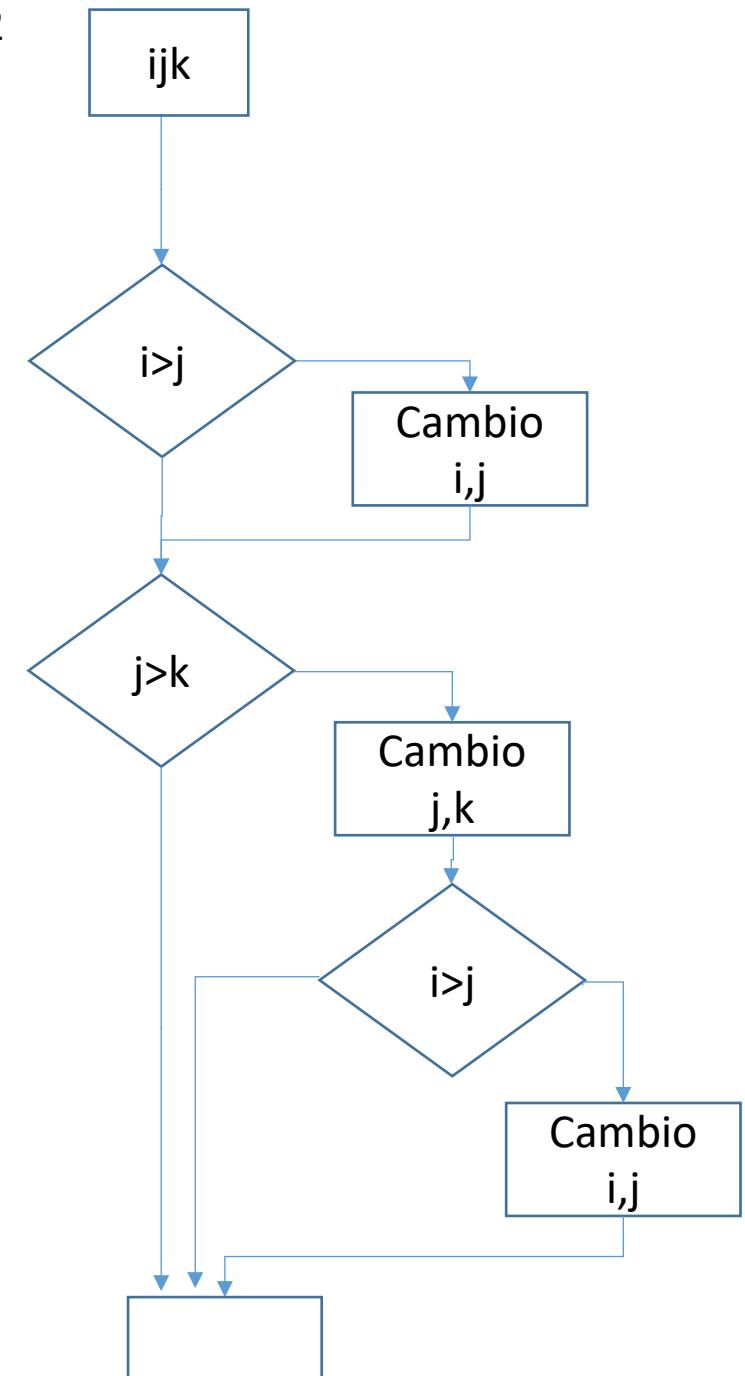
Complejidad= 6



Complejidad ciclomática

$M = \text{Número de condiciones} +$
 $\text{Número de salidas de la pieza}$

Complejidad= 4



Ordenar 3 números

Escribe un algoritmo que tras leer tres enteros los escriba en pantalla ordenados de menor a mayor (sin usar los métodos y funciones de ordenación de Python).

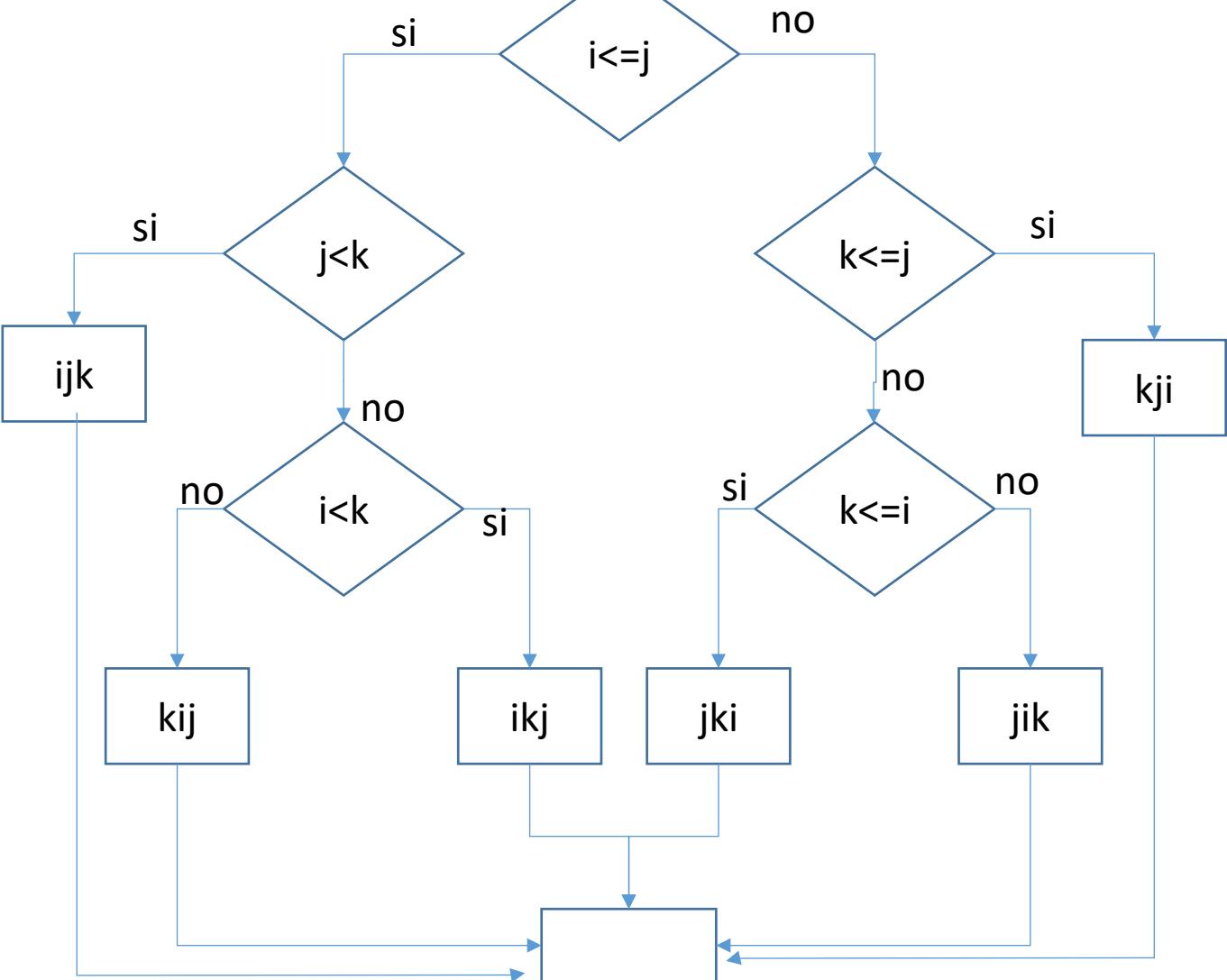
Observaciones preliminares:

- El enunciado pide un algoritmo, no pide estrictamente un programa, por lo que podrías hacerlo en Python, en pseudocódigo o con un diagrama de flujo. Lo escribimos en Python.
- El enunciado no especifica si debe estar modularizado, por lo cual valen ambas opciones. He optado por subprogramar, puesto que deseo reusar.
- Cuando un subprograma toca la interfaz de usuario es menos probable reusarlo, de modo que he hecho una función y pospuesto la escritura.
- Ya que Python es un lenguaje con débil control de tipo, el subprograma ordena 3 valores de cualquier tipo en que esté definida la comparación por mayor y menor (en todos realmente). En la doc se refleja que es válido para cualquier numérico (float o int)).

Ordenar 3 números

ijk

Versión 1



"""
ordenar 3 números """

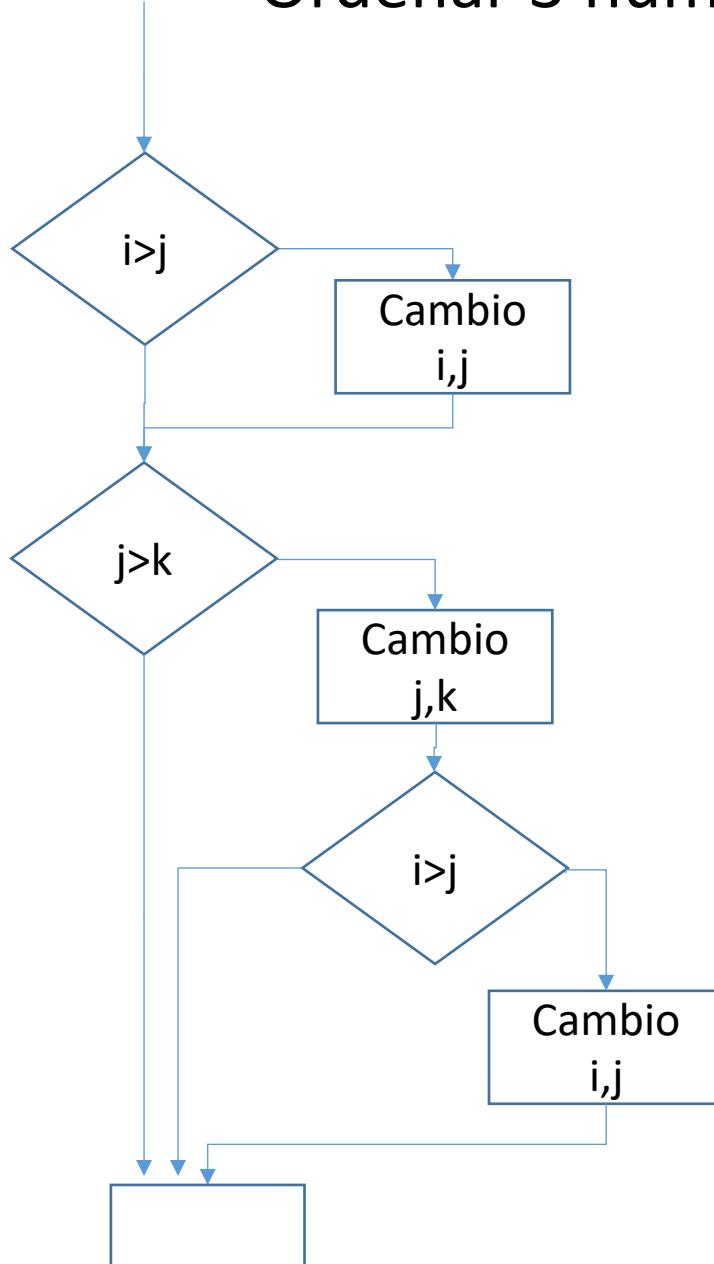
```
def ordenados3_v1(i,j,k):  
    """num,num,num-->num,num,num  
OBJ: devuelve los 3 valores  
ordenados ascendente"""  
    if i<=j:  
        if j<=k: ord = (i,j,k)  
        elif i<=k: ord = (i,k,j)  
        else: ord = k,i,j  
    elif k<=j: ord = (k,j,i)  
    elif k<=i: ord = (j,k,i)  
    else:ord = (j,i,k)  
    return ord  
'''
```

#Probador

```
print(ordenados3_v1(1,2,3))  
print(ordenados3_v1(1,3,2))  
print(ordenados3_v1(2,3,1))  
print(ordenados3_v1(2,1,3))  
print(ordenados3_v1(3,1,2))  
print(ordenados3_v1(3,2,1))  
'''
```

Ordenar 3 números

Versión 2



```
def cambiados(i,j):  
    #en otros lenguajes es bastante más complejo  
    #PERO espera y verás que aun es mas fácil  
    """num,num,num-->num,num,num  
OBJ: intercambia valores"""  
    return j,i  
  
def ordenados3_v2 (i,j,k):  
    """int,int,int-->int,int,int  
OBJ: devuelve los 3 valores ordenados  
ascendente"""  
    if i>j: i,j = cambiados(i,j)  
    if j>k:  
        j,k = cambiados(j,k)  
        if i>j:i,j = cambiados(i,j)  
    return i,j,k  
  
#Probador  
print (ordenados3_v2(1,2,3))  
print (ordenados3_v2(1,3,2))  
print (ordenados3_v2(2,3,1))  
print (ordenados3_v2(2,1,3))  
print (ordenados3_v2(3,1,2))  
print (ordenados3_v2(3,2,1))
```

Ordenar 3 números

¡Pero Python es LA REPERA!

;-) Ventajas de la referenciación en vez de asignación

Prueba en tu editor:

```
i=1  
j=2  
i,j=j,i  
print(i,j)
```

Versión 3

```
def ordenados3_v3 (i,j,k):  
    """num,num,num-->num,num,num  
    OBJ: devuelve los 3 valores ordenados  
    ascendente"""  
    if i>j: i,j = j,i  
    if j>k:  
        j,k = k,j  
        if i>j:i,j = j,i  
    return i,j,k  
  
#Probador  
print (ordenados3_v3(1,2,3))  
print (ordenados3_v3(1,3,2))  
print (ordenados3_v3(2,3,1))  
print (ordenados3_v3(2,1,3))  
print (ordenados3_v3(3,1,2))  
print (ordenados3_v3(3,2,1))
```

Complejidad algorítmica

- cantidad de recursos que necesita un algoritmo para resolver el problema → determinar la eficiencia de dicho algoritmo.
 - Tiempo
 - Memoria

El tiempo de ejecución es proporcional a recuento de operaciones

Por cada operador de comparación (`==, !=, >, >=, <, >=`) suma 1,
por cada operador booleano (or, and, not) o aritmético (+, -, *, /, //, %, etc.) suma 1
Por cada referenciación suma 1

Ordenar 3 números

```
""" mayor de 3 números """
def ordenados3_v1(i,j,k):
    """num,num,num-->num,num,num
    OBJ:los 3 valores ordenados
    ascendente"""
    if i<=j:                                #peso Sum
        if j<=k: ord = (i,j,k)                #4    1+4
        elif i<=k: ord = (i,k,j)              #4    1+1+4
        else: ord = k,i,j                     #3    1*3+3
    elif k<=j: ord = (k,j,i)                #4    1+4
    elif k<=i: ord = (j,k,i)                #4    1*2+4
    else:ord = (j,i,k)                      #3    1*3+3
    return ord                                #máximo =6
```

En promedio $\frac{1}{6}$ vendrán en cada uno de los 6 órdenes posibles → promedio
 $=\frac{1}{6}((1+4)+(2+4)+(3+3)+(1+4)+(2+4)+(3+3)=5,66$
operaciones/ejecución

```
def ordenados3_v3 (i,j,k):
    """num,num,num-->num,num,num
    OBJ: devuelve los 3 valores ordenados
    ascendente"""
    if i>j:                                #1
        i,j = j,i                            #2
    if j>k:                                #1
        j,k = k,j                            #2
        if i>j:                            #1
            i,j = j,i                        #2
    return i,j,k                            #máximo =9
```

En promedio: los dos primeros if siempre (2)+ los dos primeros intercambios y el tercer intercambio solo $\frac{1}{2}$ de las veces el tercer intercambio solo $\frac{1}{4}$ de las veces.
 $\text{Promedio}=2+1/2(2+2+1)+1/4(2)= 5$

¡Aún mas simple!... ¿te he dicho alguna vez que Python es LA REPERA?

Si ya has dado listas... Ordenar 3 números Versión 4

```
def ordenados3_v4 (i,j,k):  
    """num,num,num-->num,num,num  
OBJ: devuelve los 3 valores ordenados ascendente"""  
    aux=[i,j,k] #construyo una lista  
    aux.sort() #la ordeno  
    return tuple(aux) #devuelvo tupla con elementos de lista
```

Probablemente tarda más que la versión 3, pero usar los servicios del lenguaje:

- Ahorra tiempo de programación y pruebas.
- Aumenta la legibilidad.
- Menor riesgo de fallos (por ser programas más probados que los propios).
- Posibilidad de que esté optimizado.

Cuando programas toma todas las medidas para que tu código sea robusto, mantenible, reusable y eficiente. Pero si puedes reusar evita programar.

Resumen

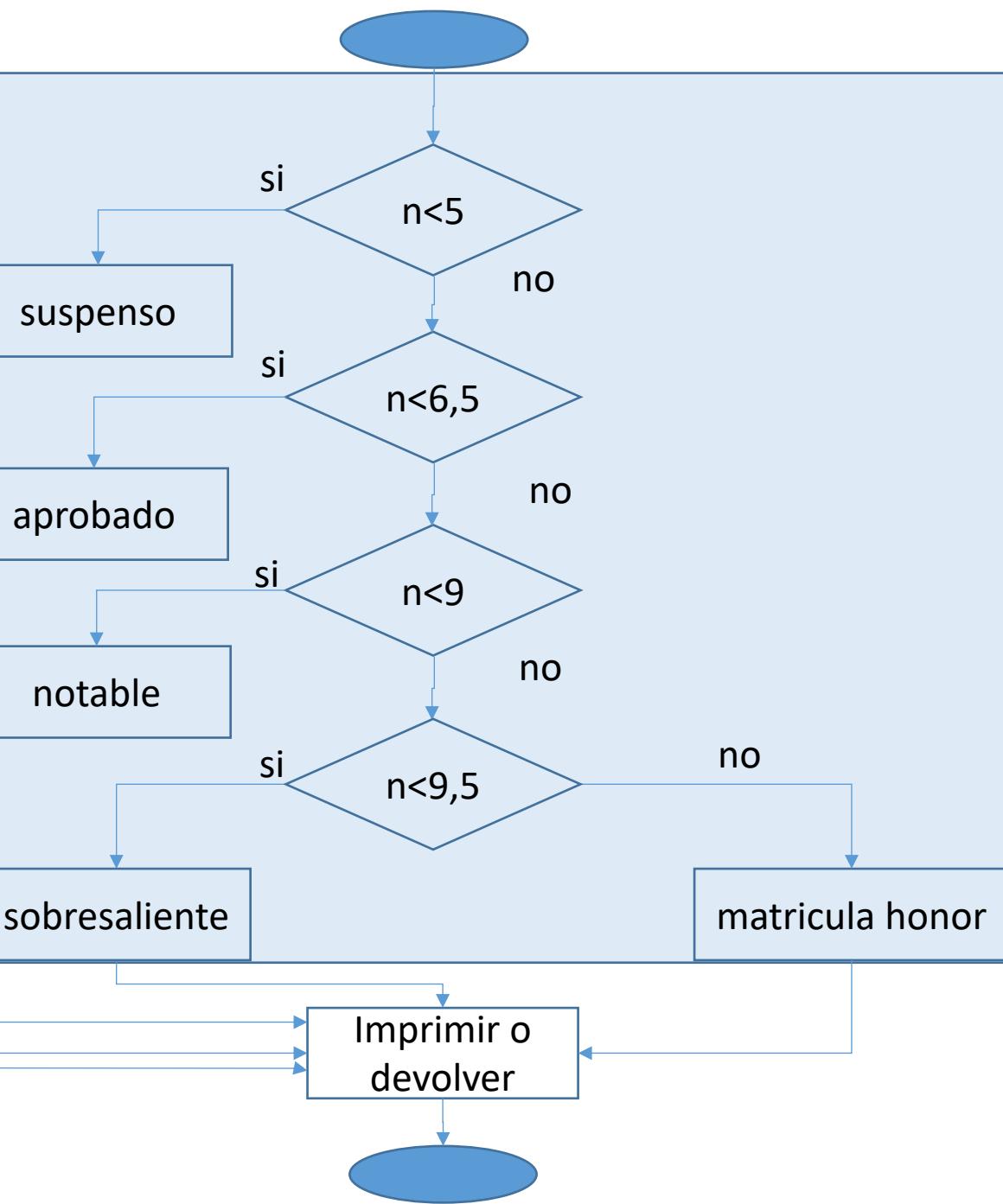
- De las versiones presentadas, ordenados3_v3 tiene la menor complejidad ciclomática. Es más legible y por tanto menos propensa a errores, mas fácil de programar.
- En el peor de los casos, v3 es mucho más eficiente que v1, pero menos que v2. En promedio es la más eficiente.
- **¡¡Python es genial!! ¡¡Fantástico!! ¡¡Comodísimo!!**
- Si tu versión tiene 6 if planos es inaceptablemente cara.
- Si tu versiones lleva operadores booleanos también es más cara de lo necesario.
- La alternativa múltiple se consigue con if anidados “en escalera”

Días que tiene el mes (Alternativa múltiple)

- Casos de prueba
- Diagrama de flujo (la parte problemática)
- Traducirlo a Python ¿Subprogramo? Si es que si.... Aplica el dodecálogo.
- Recuerda que uno de los puntos del dodecálogo es reuso... ¿has hecho ya algo útil para éste?
- Adelante...
- Mira si puedes aportar mejoras al correspondiente hilo del foro
- Usa Pythontutor si algo no te funciona, para trazarlo paso a paso

Nombre de los días de la semana (Alternativa múltiple)

- Datetime en su módulo date tiene la función today que devuelve la fecha de hoy.
- Tambien tiene una función que devuelve el día de la semana en número: el lunes es 0, martes 1....
- Haz un subprograma que devuelva el día de la semana que es hoy en letras.
- En un par de semanas tendremos una potente herramienta que nos permitirá hacerlo aun mas sencillo, pero ya puedes hacer un buen trabajo



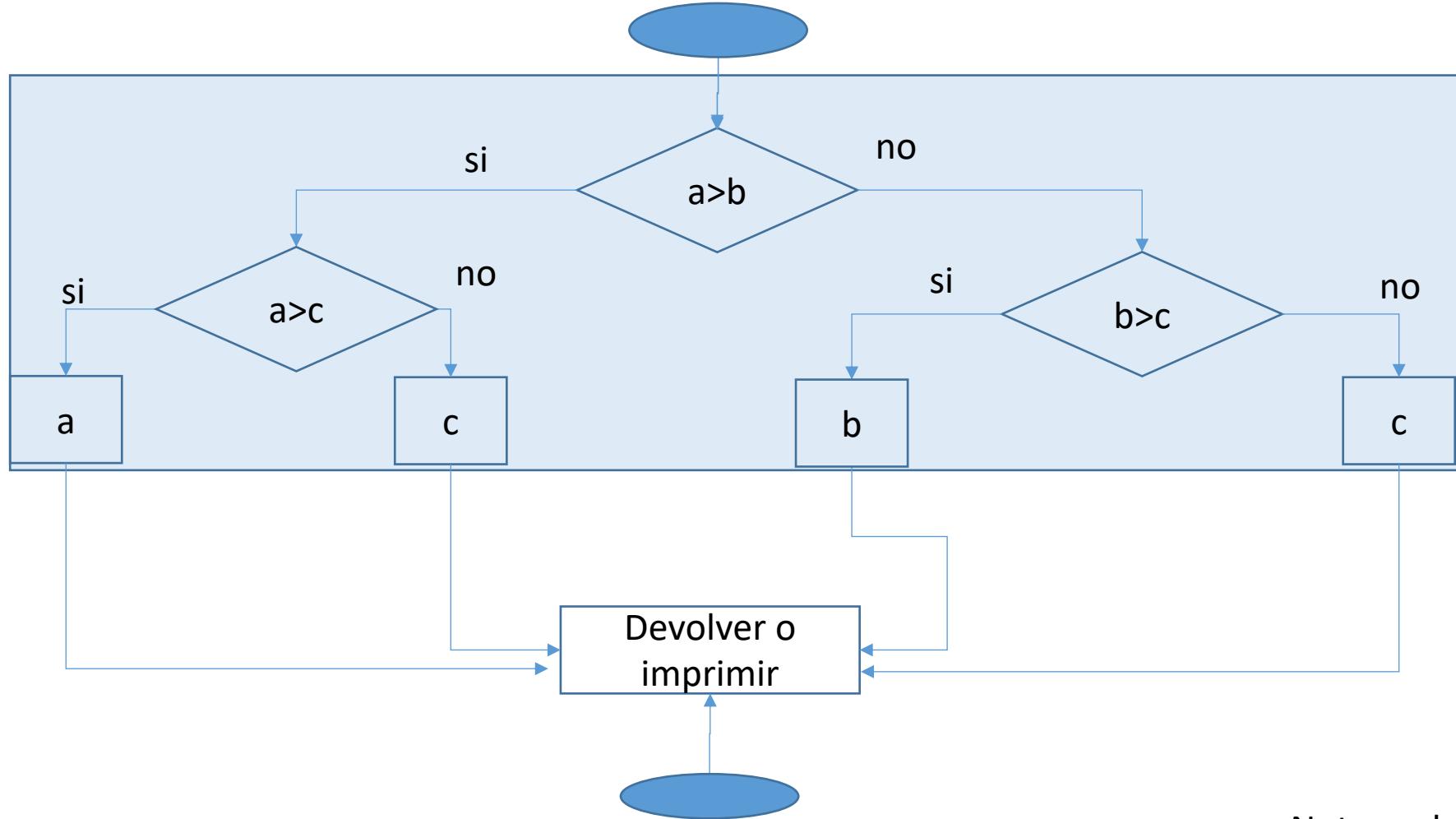
Nota a texto

Dibuje el diagrama de flujo de un buen algoritmo para pasar de una calificación numérica ($0 \leq \text{cal} \leq 10$) a suspenso (0 a 4,99), aprobado (5 a 6,49), notable (6,5 a 8,99), sobresaliente (9 a 9,49) y matrícula de honor (9,5 a 10)

Nota: se han incluido inicio y fin por completitud, pero como hay en día se plantea el diagrama de flujo para aspectos puntuales, sería suficiente con la zona recuadrada

Mayor de 3

(1,5)Dibuje el diagrama de flujo de un buen algoritmo para calcular el mayor de 3 números



Nota: se han incluido inicio y fin por completitud, pero como hay en día se plantea el diagrama de flujo para aspectos puntuales, sería suficiente con la zona recuadrada

Antes de empezar...

- ¿Qué hace `a=a==False`? a inicializada a boolean
- En 1582 el calendario gregoriano estableció que un año es bisiesto si es divisible entre 4, a menos que sea divisible entre 100. Sin embargo, aunque un año sea divisible entre 100, si además es divisible entre 400, también es bisiesto.

La función `esBisiesto` identifica si un año es bisiesto:

```
def esBisiesto (anno):  
    """int→bool  
    OBJ: True si el año es bisiesto anno  
    PRE: año>=1582  
    return anno%4==0 and anno%100!=0 or anno%400==0
```

Resolución de problemas para ingenieros con Python estructurado

U5

Condicionales y gestión de excepciones

!!!!A por la **robustez**!!!!

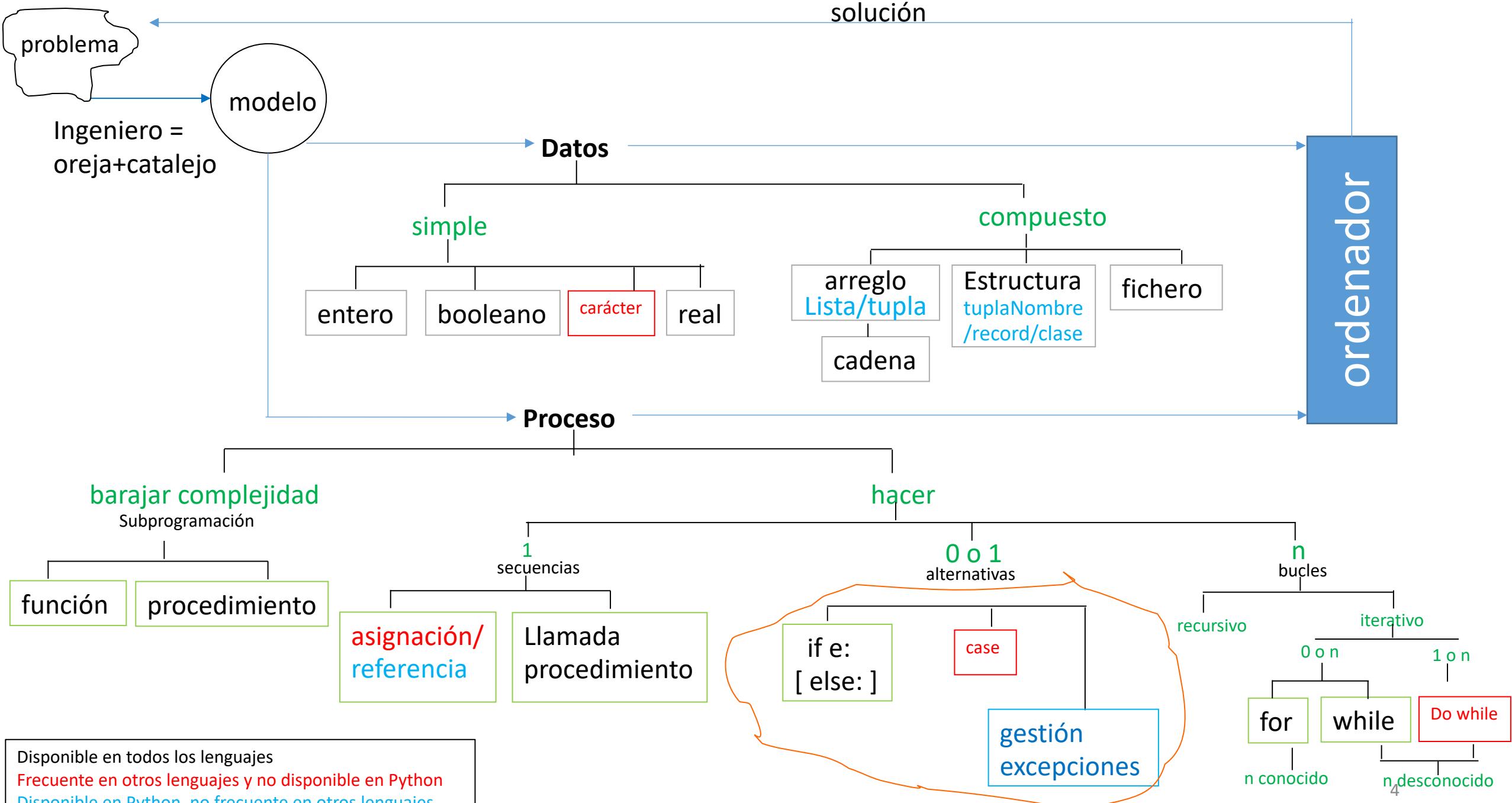
Rosalía Peña



Resolución de problemas con Python

Características de software que guían este curso⁽¹⁾	Herramientas
Mantenibilidad → Legibilidad	Semántica (identificadores), documentación, subprogramación, diseño...
Reusabilidad	Subprogramación, diseño
Robustez ≈ eficacia	Condicionales
Eficiencia (iniciación)	Diseño, algoritmia

(1) Todas ellas comprendidas en **ISO 25010**. <http://iso25000.com/index.php/normas-iso-25000/iso-25010º>



Diseño de casos de prueba



[HTTP://DEPROFESIONLIBRERA.BLOGSPOT.COM](http://DEPROFESIONLIBRERA.BLOGSPOT.COM)

Una fábrica funciona ininterrumpidamente (7dias*24horas). Se desea calcular el tiempo, expresado en horas y minutos, que ha trabajado un empleado, sabiendo el momento de entrada y el de salida (expresados en horas y minutos). Un trabajador no puede trabajar más de 8 horas seguidas, de modo que el momento de entrada y el de salida, corresponden al mismo día (si entrada <= que la salida) o a días consecutivos (en caso contrario). Caso de haber trabajado más de 8 horas, el programa dará una alerta. Completa las salidas esperadas (valores de x e y) en los siguientes casos de pruebas:

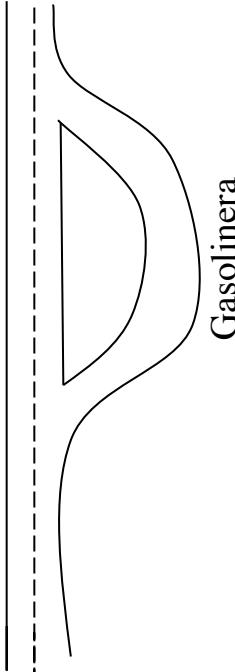
Caso prueba	11:00 11:15	12:33 20:33	12:30 14:15	23:00 2:13	23:55 2:15	6:15 19:30
Salida deseada	xh, ym	xh, ym	xh, ym	xh, ym	xh, ym	xh, ym ¡¡ALERTA!!

¿falta un caso? 1:45|1:00 .

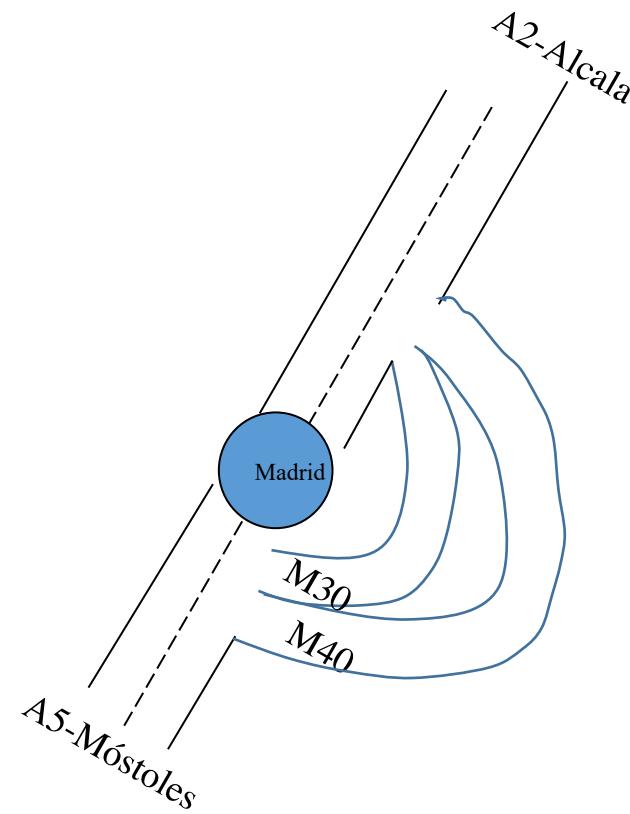
Hasta ahora: instrucciones secuenciales {referenciación + llamada a procedimientos} todas se hacían exactamente una vez.

Condicionales

Santander



Cádiz

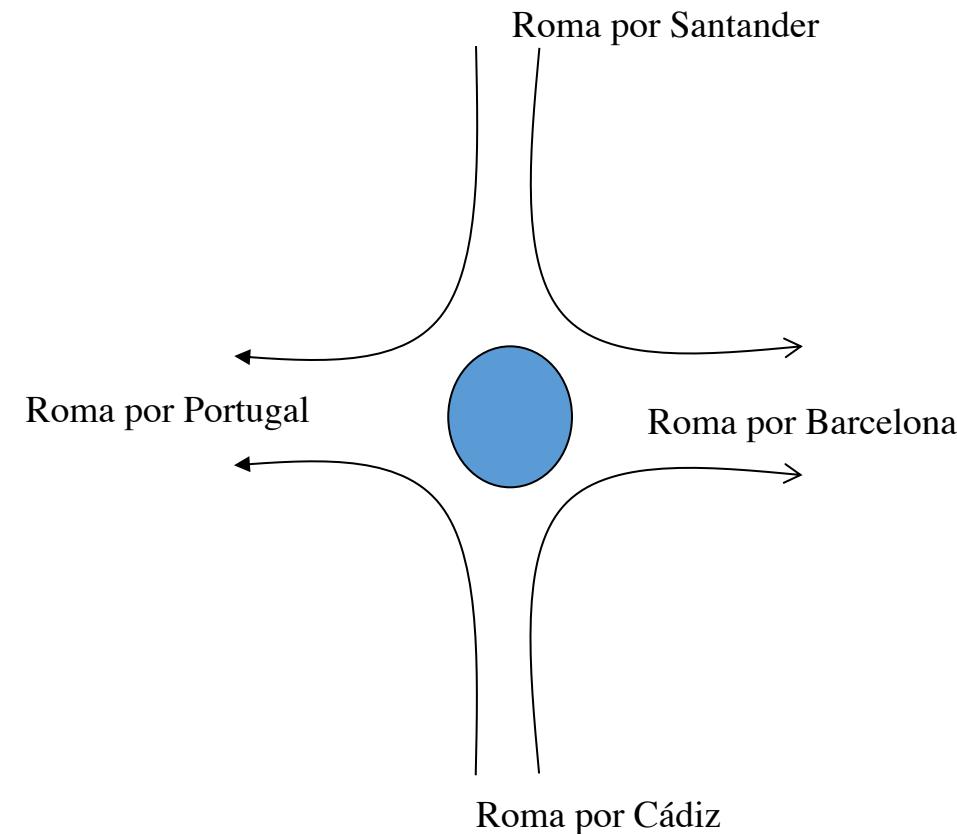


SI condicion:ENTONCES

```
if deposito_vacio or hambre:  
    entrar_gasolinera()
```

SI condicion ENTONCES.... SINO

```
if hora_punta: tomar(R2)  
else: tomar(M40)
```



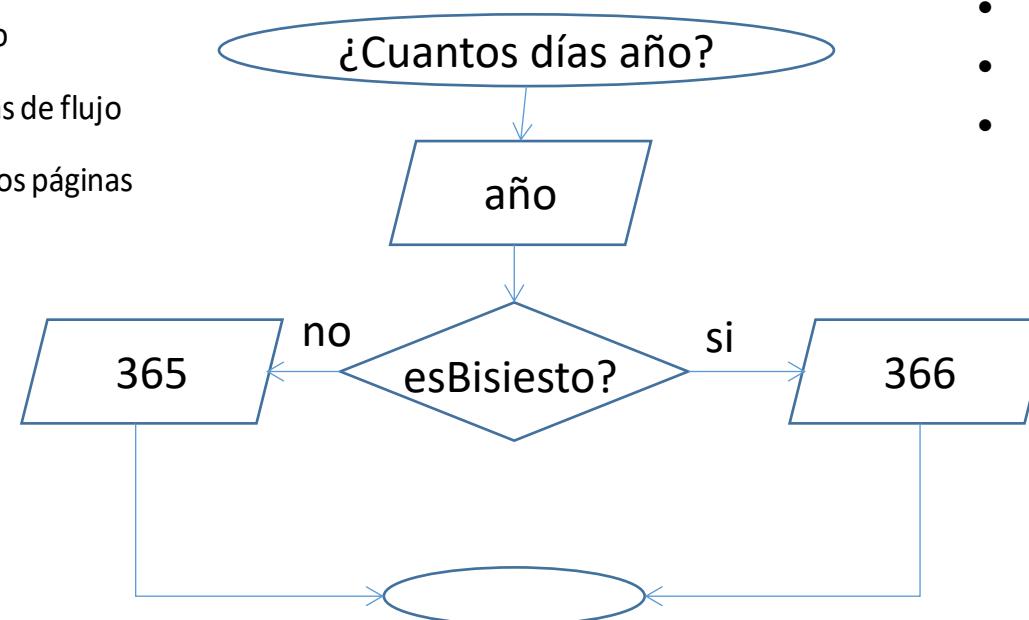
CASO de que ordinal HAZ

```
CASE mes:  
1,3,5,7,8,10,12: dias=31  
4,6,9,11: dias=30
```

Diagrama de flujo: lenguaje gráfico para expresar algoritmos

Vocabulario:

	Símbolo inicial y terminal
	Proceso
	Entrada/Salida
	Decisión
	Dirección del flujo
	Conector de líneas de flujo
	Conector entre dos páginas



Sintaxis:

- Empieza y acaba con óvalo.
- A cada rectángulo, paralelepípedo o rombo llega exactamente una flecha.
- De cada rectángulo, paralelepípedo sale exactamente una flecha.
- De cada rombo salen al menos dos flechas.
- Conector o llega o sale flecha.
- Óvalo o sale una o llega una o varias.

Normas de estilo:

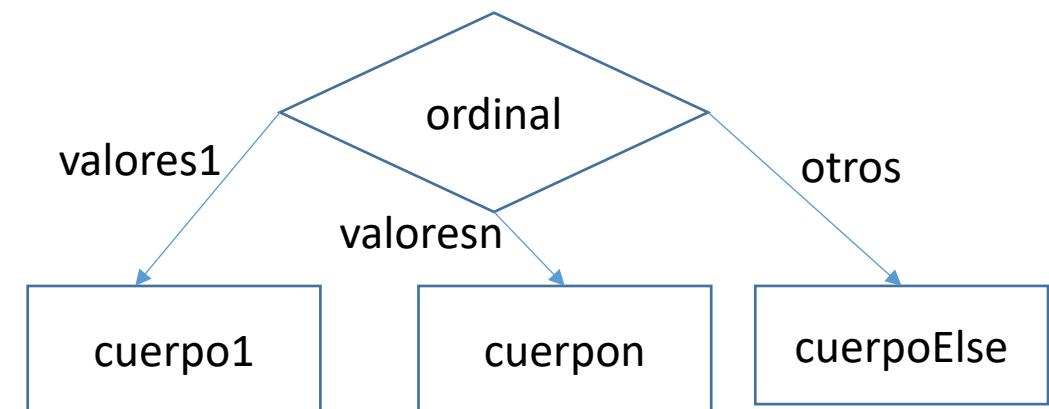
- Arriba → abajo; Izq → der.
- Líneas no se cruzan
- ...

Sintaxis del condicional Python

```
if expresion_bool:  
    cuerpo1  
[else:                      # [ ] = opcional  
    cuerpo2]
```

Sintaxis de CASE en otros lenguajes:

```
CASE ordinal DO:  
    valores1:cuerpo1  
    ...  
    valoresn: cuerpon  
    [ELSE: cuerpoElse]
```



Un ejemplo: imprime el mayor de dos números, o "iguales"

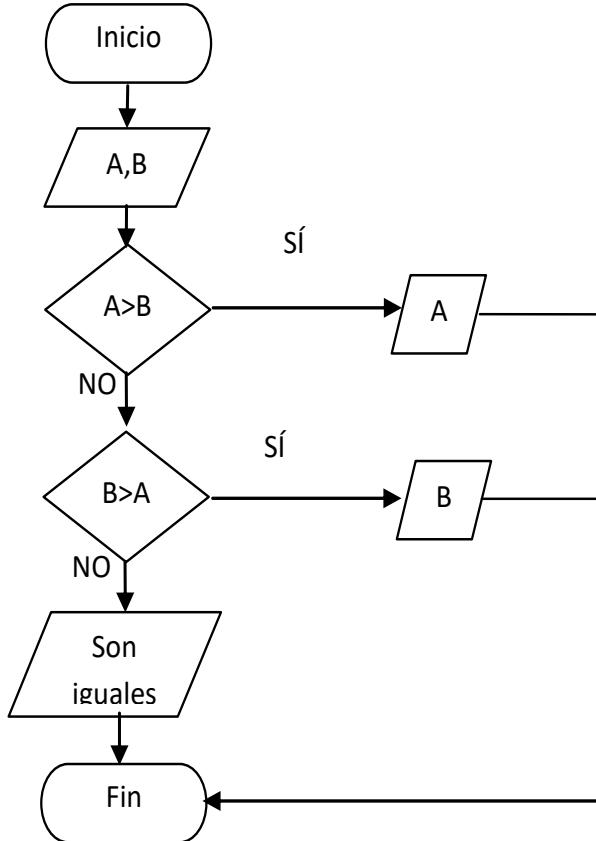
Casos de prueba:

1, 2 → segundo: 2

2, 1 → primero: 2

2, 2 → iguales: 2

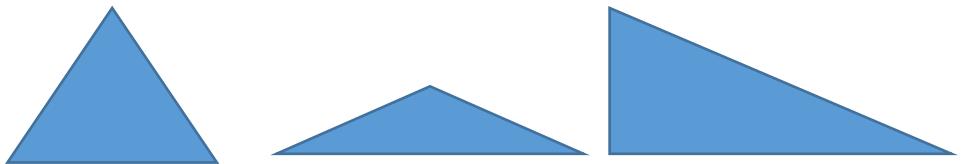
Diagrama de flujo



Programa

```
*****  
* PROGRAMA: MayorDe2numeros.py  
* OBJ: pide 2 números e indica cual es el mayor,  
* o si son iguales  
*****  
  
a = float(input('primer número: ')) #OJO incompleto  
b = float(input('segundo número: '))  
if a>b:print ('el primero es mayor', a)  
else:  
    if b>a: print ('el segundo es mayor',b)  
    else:print ('son iguales:',b)  
  
if a>b:print ('el primero es mayor', a)  
elif b>a: print ('el segundo es mayor',b)  
else:print ('son iguales:',b)
```

Otro ejemplo: Tipo triángulo ¿equilátero/isósceles/escaleno?



Diseño de casos de prueba (pista: 3 son obvios !A por ellos! En segunda pensada, me salen mínimo 5)

a	b	c	objetivo
3.0	3.0	3.0	equilátero
3.0	3.0	2.0	isósceles ab
3.0	2.0	3.0	isósceles ac
2.0	3.0	3.0	isósceles bc
3.0	4.0	5.0	escaleno

Otro ejemplo: Tipo triángulo ¿equilátero/isósceles/escaleno?

Diferentes estilos de programación

```
*****  
*          PROGRAMA: tipoTriangulo.py          *  
*  OBJ: tipo de triangulo {equilatero/isósceles/escaleno}  *  
* formado por tres lados introducidos por usuario          *  
*****  
  
a = float(input('a: '))  
b = float(input('b: '))  
c = float(input('c: '))  
  
if a==b and a==c and c==b: tipoTri = 'equilatero'  
    1   2   3   4   5  
  
if a!=b and a!=c and b!=c: tipoTri = 'escaleno'  
    6   7   8   9   10  
  
if a==b and a!=c and b!=c or a==c and a!=b and b!=c or \  
    11  12  13  14  15  16  17  18  19  20  21  22  
    c==b and a!=c and b!=a: tipoTri = 'isósceles'  
    23  24  25  26  27
```

print ('el triángulo',a,';',b,';',c,'es', tipoTri)

```
if a==b==c: tipoTri = 'equilátero'  
    1   2  
elif a!=b!=c!=a: tipoTri = 'escaleno'  
    3   4   5  
else: tipoTri = 'isósceles'
```

Otro ejemplo: Tipo triángulo

Eficiencia= recursos consumo proceso ¿son manías?

Versión	Máximo	Medio (sobre 33 triángulos de cada tipo)	Tiempo
v1	27	$27 * 99 = 2637$	$(2637 / 346,5) = 7,61$
v2	12	$12 * 99 = 1188$	$3,43$
v3	5	$2 * (33 + 33/2) + 5(33 + 33/2) = 346,5$	$1,0$
v4	5	$2 * 33 + 5 * 66 = 396$	$1,14$

- Si hubiera 2 transportes de Madrid a Barcelona que tardan 1 frente a 7,61 horas ¿cuál elegirías?
- Si tuvieras que contratar a un programador para tu empresa, y de dos candidatos, uno programa al estilo de v1 y otro al de la v4 ¿a cuál contratarías?

Otro ejemplo: Tipo triángulo ¿equilátero/isósceles/escaleno?

Diseño de casos de prueba Aun nos faltan condiciones



a	b	c	objetivo
3.0	3.0	3.0	equilátero
3.0	3.0	2.0	isósceles ab
3.0	2.0	3.0	isósceles ac
2.0	3.0	3.0	isósceles bc
3.0	4.0	5.0	escaleno
-2.	X	X	[1] ¿Lado adecuado?
1.0	2.0	6.0	¿es triángulo?

[1] Probamos sólo un lado, porque en el capítulo siguiente aprenderemos a reusar el código que pide un lado. Si no es así, habrá que probar las tres entradas

Gestión de excepciones:

$$ax^2+bx+c=0,$$

$$raiz = \frac{-b \pm \sqrt{b^2 - 4ac}}{2a}$$

El cuerpo ejecutivo se va complicando para manejar casos especiales

```
radicando=b**2-4*a*c
if radicando<0: print("No tiene solución real")      # imaginaria
elif a!=0:
    sol1=(-b+sqrt(radicando)) / (2*a)
    sol2=(-b-sqrt(radicando)) / (2*a)
    print("Las soluciones son: ",sol1,"y",sol2)
elif b!=0 :print("Tiene una única solución",-c/b)      #a=0 b!=0 y c=0 o c!=0
elif c==0: print("Tiene infinitas soluciones")          #a=0,b=0 y c=0
else: print("No tiene solución")                         #a=0,b=0 y c!=0
```

```
""" segundoGrado
*OBJ: raíces ax**2+bx+c=0,
a,b,c introducidos por usuario """
from math import sqrt
a = float(input('a: '))
b = float(input('b: '))
c = float(input('c: '))

r = sqrt(b**2-4*a*c)
x1 = (-b+r) / 2.0/a
x2 = (-b-r) / 2.0/a
print('x1 =',x1,' x2 = ',x2)
```

Aun queda una situación problema: el usuario introduce valor inadecuado, Ej: coma en vez de punto: a: 1,5

Gestión de excepciones:

Devuelve control a programador tras error de ejecución

```
try:  
    bloque1  
{except [tipoError]:  
    bloquej }n
```

tipoError= **ZeroDivisionError/ValueError/RuntimeError/TypeError/NameError...**

Gestión de excepciones

$ax^2+bx+c=0$,

$$raiz = \frac{-b \pm \sqrt{b^2 - 4ac}}{2a}$$

Parecido a versión original, pero poco orientativo al usuario

```
*****  
* PROGRAMA: segundoGrado *  
*OBJ: raíces de ecuación ax**2+bx+c=0, a,b,c introducidos por usuario *  
*****  
  
from math import sqrt  
  
try:  
    a = float(input('a: '))  
    b = float(input('b: '))  
    c = float(input('c: '))  
    r = sqrt(b**2-4*a*c)  
    x1 = (-b+r)/2.0/a  
    x2 = (-b-r)/2.0/a  
    print('x1 = ', x1, ' x2 = ', x2)  
  
except:  
    print('estos datos dan problemas')
```

```
*****  
* PROGRAMA: segundoGrado *  
*OBJ: raíces de ecuación  $ax^2+bx+c=0$ , a,b,c introducidos por usuario *  
*****  
from math import sqrt  
try:  
    a = float(input('a: '))  
    b = float(input('b: '))  
    c = float(input('c: '))  
except ValueError:  
    print('los valores han de ser numéricos')  
else:  
    try:  
        r = sqrt(b**2-4*a*c)  
        x1 = (-b+r)/2.0/a  
        x2 = (-b-r)/2.0/a  
        print('x1 =',x1,' x2 = ',x2)  
    except ZeroDivisionError:  
        if b!=0: print('ecuación primer grado. x=', -c/b)  
        elif c==0: print ('la ecuación tiene infinitas soluciones')  
        else: print('la ecuación no tiene solución')  
    except ValueError:  
        print('no tiene raíces reales')  
    except:  
        print('estos datos dan problemas')
```

Condicionales anidados:

```
*****  
*           PROGRAMA: estacion.py  
* Obj: estación del año correspondiente dia/mes dados por usuario  
* OJO: PP es un probador --> no comprueba validez de la entrada  
*****  
def estacion (d,m):  
    """int,int→str  
    * devuelve la estación que corresponde a d/m (dia/mes)  
    * 20/03 al 19/06 Primavera  
    * 20/06 al 21/09 Verano  
    * 22/09 al 20/12 Otoño  
    * 21/12 al 19/03 Invierno  
PRE dia, mes formarían parte de una fecha válida  
if (m==12 and d>=21)or m==1 or m==2 or (m==3 and d<=19):  
    estac = 'invierno'  
elif (m==3 and d>=20) or m==4 or m==5 or (m==6 and d<=19):  
    estac = 'primavera'  
elif (m==6 and d>=20) or m==7 or m==8 or (m==9 and d<=21):  
    estac = 'verano'  
else: estac = 'otoño'  
return estacion
```

* ¿proced/func?
* nombre?
* argumentos E/S?
* PRE: ?

```

#Probador incómodo y no reusable
dia = int(input('día: '))
mes = int(input('mes: '))
# aqui iría la validación de fecha
print ('la estación de',dia,'/',mes,'es',
      estacion(dia,mes))

```

```

"""\\"PROBADOR"""\\" # reusable, incluye todos los casos de prueba
print ('la estación de',22,'/',12,'es', estacion(22,3), 'invierno')#vale d>=20
print ('la estación de',22,'/',1,'es', estacion(22,1), 'invierno')#vale m=2
print ('la estación de',19,'/',3,'es', estacion(19,3), 'invierno')#vale d<=19

print ('la estación de',22,'/',3,'es', estacion(22,3), 'primavera')
print ('la estación de',22,'/',5,'es', estacion(22,5), 'primavera')
print ('la estación de',17,'/',6,'es', estacion(17,6), 'primavera')

print ('la estación de',22,'/',6,'es', estacion(22,6), 'verano')
print ('la estación de',22,'/',7,'es', estacion(22,7), 'verano')
print ('la estación de',17,'/',9,'es', estacion(17,9), 'verano')

print ('la estación de',22,'/',9,'es', estacion(22,9), 'otoño')
print ('la estación de',22,'/',11,'es', estacion(22,11), 'otoño')
print ('la estación de',17,'/',12,'es', estacion(17,12), 'otoño')

```

RESTART: C:\ROSALIA\trabajo\docencia\Fundamentos de programación en Python\códigos del libro\estacion_v1.py
 la estación de 22 / 12 es primavera invierno
 la estación de 22 / 1 es invierno invierno
 la estación de 19 / 3 es invierno invierno
 la estación de 22 / 3 es primavera primavera
 la estación de 22 / 5 es primavera primavera
 la estación de 17 / 6 es primavera primavera
 la estación de 22 / 6 es verano verano
 la estación de 22 / 7 es verano verano
 la estación de 17 / 9 es verano verano
 la estación de 22 / 9 es otoño otoño
 la estación de 22 / 11 es otoño otoño
 la estación de 17 / 12 es otoño otoño

Evitar condicionales anidados: Esquema agrupar

```
*****  
*          PROGRAMA: estacion.py          *  
* Obj: estación del año correspondiente dia/mes dados por usuario      *  
* Con esquema agrupar          *  
*****"  
  
def estacion (d,m):  
    """devuelve la estación que corresponde a d/m (dia,mes)          *  
    * 20/03 al 19/06 Primavera          *  
    * 20/06 al 21/09 Verano          *  
    * 22/09 al 20/12 Otoño          *  
    * 21/12 al 19/03 Invierno          *  
PRE dia, mes formarían parte de una fecha válida          """"  
    1   2  
plana=mes*100+dia      #agrupo para comparar de una vez          Contamos el número de operaciones=  
    3  
if plana<=319:estacion = 'invierno'  
elif plana<=619:estacion = 'primavera'  
elif plana<=921:estacion = 'verano'  
elif plana<=1221:estacion = 'otoño'  
else: estacion = 'invierno'  
  
return estacion  
    4  
    5  
    6
```

Condicionales anidados:

volvemos a la primera versión

```
*****  
*           PROGRAMA: estacion.py           *  
* Obj: estación del año correspondiente dia/mes dados por usuario *  
*****  
def estacion (d,m) :  
    """int,int→str  
    * devuelve la estación que corresponde a d/m (dia/mes)      *  
    * 20/03 al 19/06 Primavera                                *  
    * 20/06 al 21/09 Verano                                    *  
    * 22/09 al 20/12 Otoño                                     *  
    * 21/12 al 19/03 Invierno                                 *  
PRE dia, mes formarían parte de una fecha válida          """  
    1   2   3   4   5   6   7   8   9   10  11  Contamos el número de operaciones  
if (m==12 and d>=21) or m==1 or m==2 or (m==3 and d<=19) : tarda  $33/6 = 5,5$  veces mas  
    estac = 'invierno'  
    12  13  14  15  16  17  18  19  20  21  22  
elif (m==3 and d>=230) or m==4 or m==5 or (m==6 and d<=19) : PERO, además:  
    estac = 'primavera'  
    23  24  25  26  27  28  29  30  31  32  33  
elif (m==6 and d>=20) or m==7 or m==8 or (m==9 and d<=21) : ¿Cuántos casos de prueba requiere  
    estac = 'verano'                                         cada versión?  
else: estac = 'otoño'  
return estacion  
21
```

Evitar condicionales anidados: Esquema desagrupar

Haz un subprograma que imprime el nombre de un número dado (del 1 al 4 por brevedad)

```
1     def muestraCardinal_1_a_4 (n) :  
2         """int→nada  
3             OBJ: escribe el cardinal de n  
4             PRE: 1<=número=< 4  
5                 """
```

Cuando se emite un cheque hay que escribir el número en cifras y en letras. Y si te pido ...

Haz un subprograma que escriba el nombre de un número cardinal, comprendido entre 1 y el 99.

Los números en el sistema decimal están formados por centenas, decenas y unidades. En español, el nombre de los primeros números es irregular y hay que nombrarlos de uno en uno. Pero del 30 en adelante, el nombre es sistemático y está formado a partir de sus componentes.

Haz un subprograma que escriba el nombre de un número cardinal, comprendido entre 30 y 99.

```
5     def muestraCardinal (n):
6         """int→nada
7             OBJ: escribe el cardinal de n
8             PRE: 30<=número<
9
10            if n>=90: print('noventa', end=' ')
11            elif n>=80: print('ochenta',end=' ')
12            elif n>=70: print('setenta',end=' ')
13            elif n>=60: print('sesenta',end=' ')
14            elif n>=50: print('cincuenta',end=' ')
15            elif n>=40: print('cuarenta',end=' ')
16            elif n>=30: print('treinta',end=' ')
17            n=n%10                      #unidades
18            if n==1: print(' y uno')
19            elif n==2: print(' y dos')
20            elif n==3: print(' y tres')
21            elif n==4: print(' y cuatro')
22            elif n==5: print(' y cinco')
23            elif n==6: print(' y seis')
24            elif n==7: print(' y siete')
25            elif n==8: print(' y ocho')
26            else: print(' y nueve')
27
28 # PROBADOR
29 n = int(input('numero: '))
30 if 30<=n<=99: muestraCardinal(n)
31 else: print('fuera de rango 30<=número<=99')
```

TRAZADO DE PROGRAMAS:

Trabajo personal 5.17 (resuelto). Indica qué problema resuelve el siguiente programa: No indiques línea por línea que va haciendo el programa. Sintetiza una sola frase.

```
queHace2.py
1     """ ¿cuanto mola una carcajada? """
2     jaja=int(input())
3     jiji=int(input())
4     jojo=int(input())
5     if jaja>jiji:
6         if jaja>jojo:
7             mola=jaja
8         else:
9             mola=jojo
10    else:
11        if jiji>jojo:
12            mola=jiji
13        else:
14            mola=jojo
15    print('la que más mola es:', mola)
```

U5: Errores frecuentes en condicionales

Sean expBool cualquier expresión booleana, vBool cualquier variable boolena, ki cualquier constante ($i=1, 3..n$) y cuerpo cualquier secuencia de instrucciones (cuerpo)

Ineficiente	Mejorado	Recuerda...	Ventajas
<pre>if expBool: vBool=True else: vBool=False</pre>	vBool=expBool	A las variables booleanas, como cualquier otras se les puede asignar una expresión, no solo una constante	Menor complejidad ciclomática Menor complejidad algorítmica Menos que escribir Menos que leer
<pre>if expBool==True:cuerpo</pre>	if expBool:cuerpo	Las expresiones booleanas ya tienen un “valor de verdad”, es decir, son verdaderas o falsas, por si mismas	Menor complejidad algorítmica Menos que escribir Menos que leer
<pre>if valBool==k1: cuerpo1 if valBool==ki: cuerpo2</pre>	if vBool==k1:cuerpo1 elif vBool==ki:cuerpo1	varBool valiera k1 es imposible que valga k2	Menor complejidad ciclomática Menor complejidad algorítmica Menos que escribir Menos que leer
<pre>PRE:vBool>=k if vBool<k:print('error') else: cuerpo</pre>	cuerpo	Cada subprograma hace su oficio. Si hay precondición, será otro el responsable.	Menor complejidad ciclomática Menor complejidad algorítmica Menos que escribir y leer
<pre>PRE:vBool en {k1,ki,kn} if vBool==const1:c1 elif... elif vBool==kn: cn</pre>	if vBool==k1:c1 elif... else: ci	La precondición marca los valores posibles, si vBool, si no es ninguno de los anteriores, necesariamente es el último	Menor complejidad ciclomática Menor complejidad algorítmica Menos que escribir Menos que leer

U5: Errores frecuentes gestión de excepciones

Ineficaz	Mejorado	Recuerda...	Ventajas
<pre>a=int(input('msj')) #o a=float(input('msj'))</pre>	<pre>try: a=int(input('msj')) except: ...</pre>	Es imprescindible encerrar toda petición de int o float a usuario en su gestión de excepciones (excepto en probadores).	robustez
<pre>try: todo_el_cuerpo_ejecutivo except:Print("Algo" fue mal')</pre>	<pre>try: solo_lo_peligroso except tipo_error: print('mal por tal_cosa') resto_cuerpo_ejecutivo</pre>	Protege solo la parte imprescindible. Ofrece el msj de error lo mas preciso posible.	Facilidad de uso para el usuario

Resumen

- Las condicionales (`if else`) controlan que bloque se ejecute 0 o 1 veces.
- Los diagramas de flujo ayudan a visualizar algoritmos complejos.
- Casos de prueba = toma de contacto antes de desarrollar su solución+ Guían diseño + fase de pruebas.
- Es responsabilidad del programador garantizar la robustez del programa.
- La gestión de excepciones (`try...except...else`) facilita la codificación de situaciones problemáticas, ya que en vez de codificar el código para evitarlas permite recuperarse de ellas.
- Ante algoritmos complejos busca alternativas. Agrupar y desagrupar datos.
- Complejidad ciclomática → medida de dificultad en comprender (mantenibilidad)
- Complejidad algorítmica → medida del consumo de recursos (eficiencia)

Trabajo personal:

El Haz un programa que pida al usuario un año, y si es posterior a XXXX que imprima si es bisiesto.
No tengo que recordarte el reuso de esBisieta.

Haz un subprograma que devuelva el número de días que tiene un mes (expresado en número) en un año no bisiesto.

En las actas universitarias es necesario expresar la calificación numérica y también la textual según el siguiente criterio:

$0 \leq \text{nota} < 5$ suspenso, $5 \leq \text{nota} < 7$ aprobado, $7 \leq \text{nota} < 9$ notable $9 \leq \text{nota} < 10$ sobresaliente, nota=10 M. Honor.

Haz un subprograma que reciba una calificación numérica comprendida entre 0 y 10 y devuelva la calificación textual. Recuerda anidar los condicionales.

Trabajo personal:

Motivación: Una fábrica funciona ininterrumpidamente (7días*24horas). Se desea calcular el tiempo, expresado en horas y minutos, que ha trabajado un empleado, sabiendo el momento de entrada y el de salida (expresados en horas y minutos). Un trabajador no puede trabajar más de 8 horas seguidas, de modo que el momento de entrada y el de salida, corresponden al mismo día (si entrada <= que la salida) o a días consecutivos (en caso contrario). Caso de haber trabajado más de 8 horas, el programa dará una alerta.

El tiempo transcurrido entre horas es potencialmente reusable. Subprograma esta parte. También puedes subprogramar la alerta, si el tiempo es mayor que un número de horas, pero no lo mezcles en el mismo subprograma. Cada subprograma debe resolver un solo problema, de lo contrario es menos reusable ¿Es responsabilidad del subprograma que las horas y minutos sean correctos, o por el contrario es una precondición?

Caso prueba	11:00 11:15	12:33 20:33	12:30 14:15	23:00 2:13	23:55 2:15	1:45 1:00	6:15 19:30
Salida deseada	0h,15m	8h, 0m	1h, 45m	3h, 13m	2h, 20m	23:15	13h.15 ¡ALERTA!
Motivo	me>ms						

Resolución de problemas para ingenieros
con Python estructurado

U6

Tipos de datos secuencias y
bucle definido

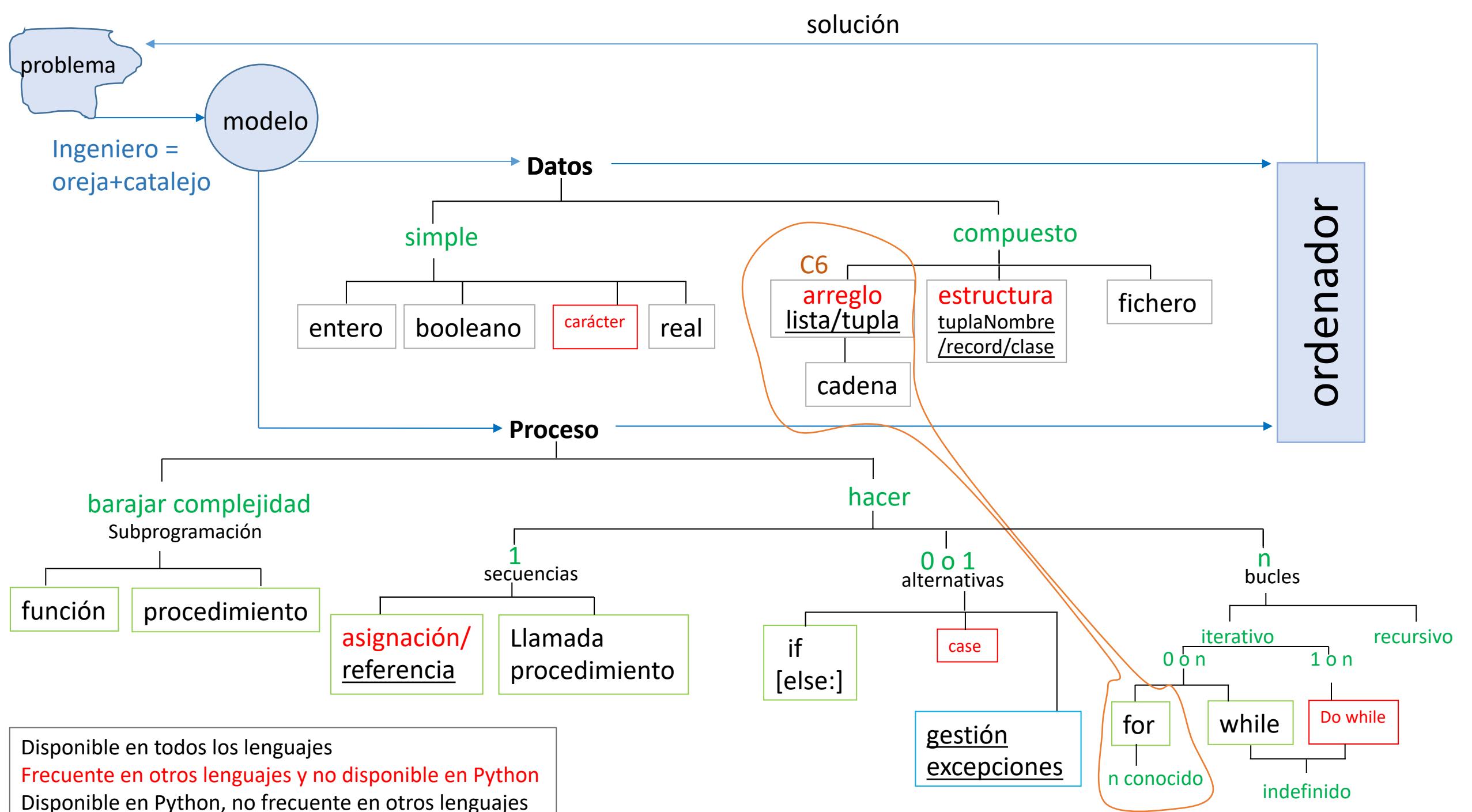
!!!!Abstracción!!!!

Rosalía Peña



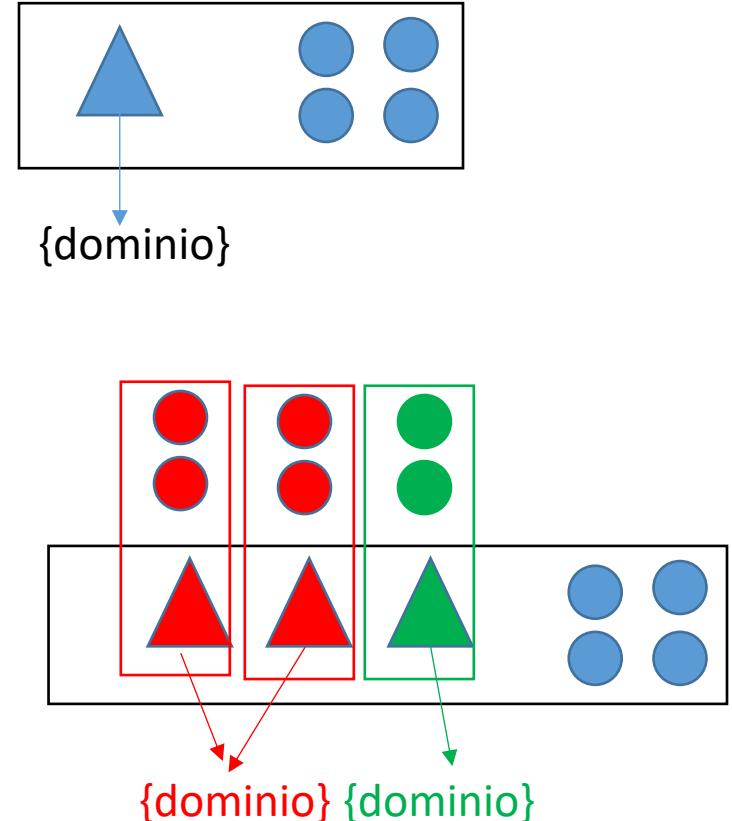
Universidad
de Alcalá





Tipo de datos simple

Objeto dato definido sobre un dominio, y sus operaciones



Tipo de datos compuesto

Conjunto de objetos dato, potencialmente de distinto tipo
con sus operaciones Ej: fecha, aula, color(RGB)
Cada elemento conserva sus operaciones

Todos del mismo tipo
Accesibles por posición



Taller: coche1, coche2, ..., cochen
Aula: Alumno1, alumno2,...
Farmacia: med1,med2,...
Equipo deportivo: jug1,jug2,...

Potencialmente de tipo distinto
Accesibles por nombre



Secuencias: común a tuplas, listas, cadenas

- Creación

general

Prueba en la consola
cad=str()
type(cad)

tp=tuple()
type(tp)
lista=list()
type(lista)

vacía

cad2=''
type(cad2)

tp2=()
type(tp2)
lista2=[]
type(lista2)

unitaria

c='abcdef'
type(cad3)

t=(1)
type(tp3)

lis=[1]
type(lista3)

t=1,

c='abcdef' c[0] c[1] c[5] c[-1] c[-6]

c[6]

- Acceso a elementos: $s[i]$; $0 \leq i \leq n-1$;
hacia atrás $-1 \geq i \geq n$; aborto $-(n+1) \leq i \leq n$

- segmentador:
operador *slice*

Secuencia[ini:fin:inc]

c[0:5:2]
c[2::]

```
semana='lunes','martes','miércoles','jueves','viernes','sábado', 'domingo'  
laborables = semana[0:5]  
# ('lunes', 'martes', 'miércoles', 'jueves', 'viernes')  
alternos2 = semana[1:5:2]  
#('martes', 'jueves')  
terribles=semana[:3]  
#('lunes', 'martes', 'miércoles')  
Sonnando=semana[3:]  
# ('jueves', 'viernes', 'sábado', 'domingo')
```

Bucle definido

- Sintaxis

```
for elemento in secuencia:  
    cuerpo
```

```
""" ***** listar días de la semana (no subprogams) ***** """
semana = 'lunes', 'martes', 'miércoles', 'jueves', 'viernes', 'sábado', 'domingo'
```

```
for dia in semana:  
    print(dia)
```

Sintaxis
for e in s:
 cuerpo

Bucle definido: Esquema contador

```
cont=0
for elemento in secuencia:
    if condicion: cont=cont+1
print/return cont
```

cont+=1

¿Cuántos meses tienen 31 días?

Proc/fun? Nombre? Argumentos? OBJ? PRE?

```
uDMes=(31,28,31,30,31,30,31,31,30,31,30,31)
```

```
def numMesesLargos (uDMes):
    """tupla-->int
    OBJ: número meses con 31 días """
    cont=0
    for uD in uDMes:
        if uD==31: cont+=1
    return cont
```

ALTERNATIVA 2

```
def numMesesLargos ():
    """tupla-->int
    OBJ: número meses con 31 días
    PRE: uDMes definido """

```

ALTERNATIVA 3

```
def numMesesLargos ():
    """tupla-->int
    OBJ: número meses con 31 días"""
    uDMes=(31,28,31,30,31,30,,30,31)
```

Sintaxis
for e in s:
 cuerpo

Bucle definido: Esquema totalizador

```
total=0.0
for elemento in secuencia:
    total+=elemento
print/return total
```

El histórico de salarios contiene, tras el DNI del empleado, el total del salario percibido por un empleado cada mes. Haz un subprograma que calcula los ingresos anuales, de cara a la declaración de la renta.

Proc/fun? Nombre? Argumentos? OBJ? PRE?

```
def totalAnual(hists):
    """ tupla --> float
    OBJ: Total acumulado por el empleado cuyo histórico es hists"""
    total = 0.0
    for pagado in hists[1:]:          # todos los elementos excepto el cero
        total = total+pagado
    return total
```

Sintaxis
for e in s:
 cuerpo

Bucle definido en otros lenguajes

PARA cont DESDE inicio HASTA fin, incremento HAZ

Cuerpo

Generador de secuencias range

```
for j in range (3,11,2):  
    print(j, end=' ', sep=', ')  
>>>3,5,7,9
```

range (inicio, fin, incremento)
Inicio incluido. Fin excluido
Ejemplo:

¿Subprogramo?
Proc/fun? Nombre? Argumentos? OBJ? PRE?

Pinta la tabla de multiplicar del 7

```
""" **** * Tabla de multiplicar del 7 **** """  
for i in range(0,10+1):  
    print (7, '*', '%2d'%i, '=', i*7)
```

Bucles anidados

Sintaxis
for e in s:
 cuerpo

1) Programa que pinta la tabla de multiplicar de los pares del 1 al 10.

2) Programa que genera la siguiente salida:

¡¡¡Jo!!! Que pena no haber subprogramado antes

Tabla de multiplicar

=====

	1	2	3	4	5	6	7	8	9	10
1	1	2	3	4	5	6	7	8	9	10
2	2	4	6	8	10	12	14	16	18	20
3	3	6	9	12	15	18	21	24	27	30
4	4	8	12	16	20	24	28	32	36	40
5	5	10	15	20	25	30	35	40	45	50
6	6	12	18	24	30	36	42	48	54	60
7	7	14	21	28	35	42	49	56	63	70
8	8	16	24	32	40	48	56	64	72	80
9	9	18	27	36	45	54	63	72	81	90
10	10	20	30	40	50	60	70	80	90	100

Sintaxis
for e in s:
 cuerpo

Programa el factorial de n: $0!=1; n!=2*3*4...*n$; no definido para $n < 0$

¿Subprogramo?
Proc/fun? Nombre? Argumentos? OBJ? PRE?

```
def factorial(n) :  
    """int-->int  
OBJ: n!  
PRE: 0<=n<=30  
    """  
  
    f = 1  
    for i in range (2, n+1) :  
        f = f*i  
    return f
```

Esquema Agrupar+seleccionar (más elegante que *case*)

Subprograma que devuelve el nombre del día, recibido el número de día semana

```
def nombreDia(dS):
```

```
    """int -->str
```

OBJ: Nombre del día de la semana correspondiente a dS:

0=lunes,...,6=domingo

PRE: $0 \leq dS \leq 6$

"""

```
if dS==0: nD = 'lunes'
```

```
elif dS==1: nD = 'martes'
```

```
elif dS==2: nD = 'miércoles'
```

```
elif dS==3: nD = 'jueves'
```

```
elif dS==4: nD = 'viernes'
```

```
elif dS==5: nD = 'sabado'
```

```
else: nD = 'domingo'
```

Semana = 'lunes', 'martes', 'miércoles', 'jueves', 'viernes', 'sábado', 'domingo'

```
return nDc
```

¿1 ó 2 parámetros ?

```
def nombreDia(dias, dS) :
```

```
    """ int -->str
```

OBJ: Nombre del día de la semana correspondiente a dS: **0=lunes,...,6=domingo**

PRE: $0 \leq dS \leq 6$

"""

```
return dias[dS]
```

U6: Errores frecuentes for

- Cambiar parámetros control de bucle:
ini, fin, inc, cont.
- ¿valor del contador a la salida del bucle?
- Anidar dos bucles con = contador.
- Un if que involucra al controlador del for, → for mal planteado.

```
for i in range(4):
    if i==2:break
    print('otra vez', i)

valor=ini
for i in range(fin):
    if i%2 ==0: valor+=3
    else: valor=valor+2.5
```

QueNoHacerConFor.py

```
""" Ejemplos de qué no hacer dentro de un for"""
```

```
ini=1
fin=4
inc=1

for cont in range(ini,fin,inc):
    print('tocado contador', cont)
    cont=cont-1
print('contador fuera del bucle', cont)

for cont in range(ini,fin):
    fin=fin+2
    print('tocado ini, fin')
print('extremo fuera del bucle', fin)

miLista=[1,2,3]
for i in range(len(miLista)):
    miLista.append(i*2)
print(miLista)

for i in range(2):
    for i in range(3):
        print(i)
```

Ideas de uso de secuencias

- Tuplas como salida de una función
- Agrupar datos del mismo tipo (a los que se accede por posición)
 - Arrays ndimensionales (en Python es secuencia de secuencias)
 - Secuencias paralelas

NO usar secuencias

- Para agrupar datos de tipos variados, con diferentes semánticas y servicios, no existiendo un orden preestablecido entre ellos en el mundo real.
Ejemplo: alumno=nombre, ape, nExpediente, fNacim,becario, famNumerosa
- Aunque Python lo permite, haría programas muy difíciles de mantener

Biblioteca dinero

Cada país acuña billetes y monedas de unos determinados valores. Por ejemplo en los países de la Unión Europea los valores son:

500.0, 200.0, 100.0, 50.0, 20.0, 10.0, 5.0, 2.0, 1.0, 0.5, 0.2, 0.1, 0.05, 0.02, 0.01 euros.

Diseña la fábrica de monedas.

Diseña el monedero para estos países.

Haz un subprograma que indique el contenido total de un monedero.

¿está vacío un monedero?

En la salida de usuario, no es elegante especificar las piezas de las que hay cero unidades. Haz un subprograma que recibe un monedero e imprime cuantas piezas hay de cada tipo (de las que verdaderamente hay).

Trabajo personal 6.19 (resuelto). Haz un subprograma que indique la forma óptima de componer una determinada cantidad, con las piezas acuñadas en el país. Por óptima queremos decir formada por el mínimo número de piezas.

Biblioteca dinero

Trabajo personal 6.20 (resuelto). Para ayudar al cajero de una tienda, haz un subprograma que recibe el importe de la compra y el dinero entregado por el cliente y calcula (devuelve) las vueltas óptimas. Reusa el subprograma anterior. Ten en cuenta las posibles situaciones anómalas, por ejemplo ¿Qué ocurre si el cliente no ha entregado dinero suficiente? ¿y si lo ha entregado exacto?

Trabajo personal 6.22. Hemos conseguido una interesante colección de servicios para gestionar dinero. Haz un módulo con ella, para que esté reusable ¿Qué nombre tiene el archivo? Úsalo desde otro.

¿Qué mas puedes necesitar en una biblioteca monedero?

Master universitario

Trabajo personal 6.13. En un master universitario se imparten n asignaturas. Todo alumno se matricula de todas las asignaturas. Cada asignatura se evalúa con un número entre 0 y 10, con decimales. Diseña el tipo de datos que puede albergar las calificaciones y haz un subprograma que calcule la nota media de las calificaciones de todas las asignaturas de un alumno. Pruébalo.

Secuencias: común a tuplas, listas, cadenas

- Creación general, vacía, unitaria
- Acceso a elementos: $s[i]$; $0 \leq i \leq n-1$; hacia atrás $-1 \geq i \geq -n$; aborto $-(n+1) \leq i \leq n$
- Segmentador: operador *slice*

```
semana='lunes', 'martes', 'miércoles', 'jueves', 'viernes', 'sábado', 'domingo'  
laborables = semana[0:5]  
# ('lunes', 'martes', 'miércoles', 'jueves', 'viernes')  
alternos2 = semana[1:5:2]  
#('martes', 'jueves')  
terribles=semana[:3]  
#('lunes', 'martes', 'miércoles')  
Sonnando=semana[3:]  
# ('jueves', 'viernes', 'sábado', 'domingo')
```

- Operaciones: concatenar +; is ; comparación; in

```
>>> t1=1,2,3,4,5,6,7,8,9,10  
>>> t2=9,  
>>> t1>t2  
False
```

- Funciones: len(s), min(s), max(s)

- Métodos: s.index(x, [i, j]); s.count(x)

Específico de listas

Operación	Acción
<code>L[i] = x</code>	El elemento <code>i</code> de <code>L</code> referencia al valor <code>x</code>
<code>L[i:j] = s</code>	El trozo de <code>L</code> desde <code>i</code> hasta <code>j</code> se sustituye por los elementos de <code>s</code>
<code>L[i:j:k] = s</code>	Los elementos <code>L[i:j:k]</code> pasan a referenciar a lo referenciado por <code>s</code>
<code>del L[i:j]</code>	Borra los elementos de <code>L</code> indicados. Igual que <code>L[i:j] = []</code>
<code>del L[i:j:k]</code>	Elimina de la lista los elementos <code>L[i:j:k]</code>
<code>L.clear()</code>	Elimina todos los elementos de la lista <code>L</code> (igual que <code>del L[:]</code>)
<code>L.append(x)</code>	Añade el elemento <code>x</code> al final de la lista
<code>L.extend(s)</code>	Añade a <code>L</code> los elementos de la lista <code>s</code> (igual que <code>L += s</code>)
<code>L *= n</code>	Con <code>n <= 0</code> , borra <code>L</code> . Actualiza <code>L</code> con su contenido repetido <code>n</code> veces (<code>n > 0</code>), repite las referencias, no copia los objetos.
<code>L.insert(i, x)</code>	Inserta <code>x</code> en <code>L</code> en la posición <code>i</code> (igual que <code>L[i:i] = [x]</code>)
<code>L.pop(i)</code>	Devuelve valor del elemento <code>i</code> y lo quita de <code>L</code> . <code>i</code> opcional, por defecto = <code>-1</code>
<code>L.remove(x)</code>	Elimina de <code>L</code> la primera aparición de <code>x</code> . <code>ValueError</code> si no encontrado
<code>L.reverse()</code>	Invierte los elementos de <code>L</code>
<code>L.sort()</code>	Ordena los elementos siguiendo el criterio de orden del tipo de elemento
<code>L.copy()</code>	Crea una copia de <code>s</code> (igual que <code>L[:]</code>)

Resumen

- Abstracción: Agrupar elementos del mismo tipo. Acceder por posición
 - Otros lenguajes: arreglo (array), tipo cadena (string) es una especialización del arreglo
 - En *Python* varios tipos secuencia: tupla (tuple), lista (list) y cadena (str).
- Los 3 tipos secuencia comparten sintaxis y semántica+ servicios específicos cadenas y listas.
 - Funciones: `len(s)`, `max(s)` y `min(s)`,
 - Métodos: `s.index(x)`, `s.count(x)`.
 - Concatenar secuencias del mismo tipo (`tupla1+tupla2`, `lista1+lista2`, `cad1+cad2`).
- **inmutables** (tupla, cadena) referenciación del elemento completo
- **mutables** (listas) , modificar, añadir y eliminar elementos individuales del tipo compuesto. Cómodo, eficiente, pero peligroso
- Una secuencia representa a un objeto, por lo que una función puede devolver una secuencia.
- `for` repite el cuerpo del bucle por cada elemento de una secuencia. Útil para listar, contar, calculo total o media de los elementos de la secuencia.
- Dentro del cuerpo de un bucle puede ir cualquier instrucción del lenguaje.
- La función `range` genera secuencia enteros que puede ser recorrida por un `for`.
- Si no modificas los parámetros de control del `for` hay garantía de que el programa termina.

Trabajo personal dirigido

Construye un subprograma que calcule el máximo común divisor de 3 números enteros distintos de cero.

```
def maxComunDiv(x,y,z):
    """ int, int, int -> int
        OBJ: máximo común divisor de x,y,z
        PRE: x,y, z son !=0
    """
    top = min(abs(x),abs(y),abs(z))
    for i in range (1, top+1):
        if x%i == 0 and y%i == 0 and z%i == 0:
            #if x%i == y%i == z%i == 0:      #Python puede hablar así:cómo y eficiente
                mcd = i
    return mcd
```

Construye el probador

```
print(maxComunDiv(1,2,3),'debe dar 1')
print(maxComunDiv(9,6,3),' debe dar 3')
print(maxComunDiv(25,-25,25*5),'debe dar 25')
```

Sigue probando

```
print(maxComunDiv(2.3,1,0),'ni de coña')
print(maxComunDiv(0,3,0),'3')
```

```
1 debe dar 1
3 debe dar 3
25 debe dar 25
>>>
```

*TypeError: 'float' object cannot be interpreted as an integer
UnboundLocalError: local variable 'mcd' referenced before assignment*



Trabajo personal dirigido

Cualquier número entero,
excepto el 0, divide a este .

Contémplalo en tu código

```
def maxComunDiv(x,y,z):
    """ int, int, int -> int
        OBJ: maximo común divisor de x,y,z, -2 si infinito """
    # quitar ceros
    maximo=max(abs(x),abs(y),abs(z))
    if maximo==0:div=-2
    else:
        if x==0: x=maximo
        if y==0: y=maximo
        if z==0: z=maximo
        # Ahora si calculo el mcd
        top = min(abs(x),abs(y),abs(z))
        for i in range (1, top+1):
            if x%i == 0 and y%i == 0 and z%i == 0:
                div = i
    return div

#PROBADOR
print(maxComunDiv(1,2,3),'debe dar 1')
print(maxComunDiv(9,6,3),' debe dar 3')
print(maxComunDiv(25,-25,25*5),'debe dar 25')
print(maxComunDiv(0,6,0),'debe dar 6')
print(maxComunDiv(5,-25,0),'debe dar 5')
print(maxComunDiv(0,0,0),'debe dar infinito')
```

Trabajo personal dirigido

```
def maxComunDiv(x,y,z):
    """ int, int, int -> int
        OBJ: maximo común divisor, -1 si error, -2 si infinito """
    div=-1
    try:
        # quitar ceros
        maximo=max(abs(x),abs(y),abs(z))
        if maximo==0:div=-2
        else:
            if x==0: x=maximo
            if y==0: y=maximo
            if z==0: z=maximo
            # Ahora si
            top = min(abs(x),abs(y),abs(z))
            for i in range (1, top+1):
                if x%i == 0 and y%i == 0 and z%i == 0:
                    div = i
    except:
        print ¿Quién es una función para tocar la interfaz de usuario!!!!!
        return div
#print(maxComunDiv(5,-25,0),'5')
#div = i
#return div
#PROBADOR
print(maxComunDiv(2.33,2,3),'debe dar 1')

print(maxComunDiv(1,2,3),'debe dar 1')
print(maxComunDiv(9,6,3),' debe dar 3')
print(maxComunDiv(5,-25,0),'debe dar 5')
print(maxComunDiv(0,0,0),'debe dar infinito')
```

¿y si no existiera la doc de subprogramas?

¿Qué ocurre si x,y o z no son enteros?

¡haz robusto tu código!

¿SOLUCIÓN?

```
#PROBADOR
print(maxComunDiv(1,2,3),'debe dar 1')
print(maxComunDiv(9,6,3),' debe dar 3')
print(maxComunDiv(25,-25,25*5),'debe dar 25')
print(maxComunDiv(0,6,0),'debe dar 6')
print(maxComunDiv(5,-25,0),'debe dar 5')
print(maxComunDiv(0,0,0),'debe dar infinito')
```

Sopa de letras

Trabajo personal

Sopa de letras (8 filas x 8 columnas), localizando todas las coincidencias (en los ocho sentidos posibles: Norte a Sur, Este a Oeste, Noroeste a Sudeste....) de una palabra que el usuario introducirá por teclado. Cuando encuentre una coincidencia, mostrará la posición y el sentido en que se han encontrado, como indica el ejemplo de salida.

h	j	k	p	u	r	e	a
k	o	i	h	q	p	a	l
d	a	l	o	h	t	m	o
i	p	u	a	m	x	z	h
a	m	u	r	o	a	f	t
l	k	l	h	l	l	a	s
e	o	t	o	s	p	r	a
h	l	h	l	i	a	j	e

salida

Encontrada "hola" en 1, 1;
sentido SE

Encontrada "hola" en 3, 5;
sentido O

Encontrada "hola" en 4, 8;
sentido N

Se solicita
 a)(0,5) Justifique las estructuras de datos necesarias
 b)(0,4) descomposición funcional

Sopa de letras

Se solicita

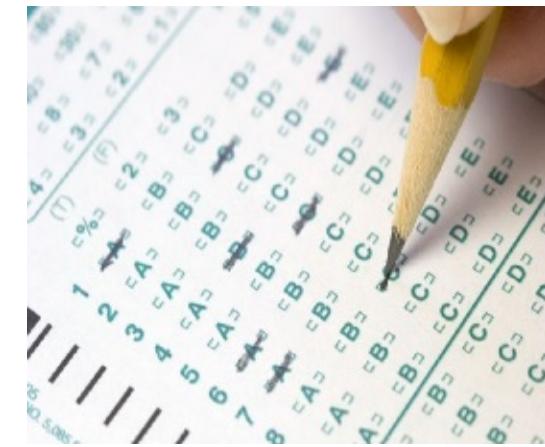
- c) (0,4) carga de la sopa de letras
- d) (0,8) búsqueda de la palabra en una dirección
- e) (0,8) generalización del procedimiento anterior para que busque en las 8 direcciones.
(8 subprogramas casi idénticos buscando cada uno en cada dirección puntuarán lo mismo que uno sólo de ellos)
- f) (0,4) estructuras de datos para esta generalización
- g) (0,5) programa que coordina los módulos

Trabajo personal

Corrección de exámenes tipo test

Una aplicación corrige los exámenes de tipo test del teórico del carnet de conducir. El examen consta de 10 preguntas. Cada pregunta ofrece 5 respuestas, de las cuales solo una es válida. El examinando cumplimenta sus respuestas en una plantilla que será leída por un lector automático.

- Diseña la estructura adecuada para almacenar el examen de un aspirante.
Especifica las restricciones semánticas.



Para realizar la corrección automática se necesitará la plantilla con las respuestas correctas. Escribe la cabecera de un subprograma que pide al examinador la plantilla, ¿qué debe devolver el subprograma?

- Flexibiliza el subprograma para que pueda variar el número de opciones del examen y el número de preguntas de la plantilla en diferentes exámenes.

Un corrector automático toma la primera hoja como plantilla y las siguientes como exámenes

- Diseña la estructura de datos para una convocatoria
- Descomposición funcional

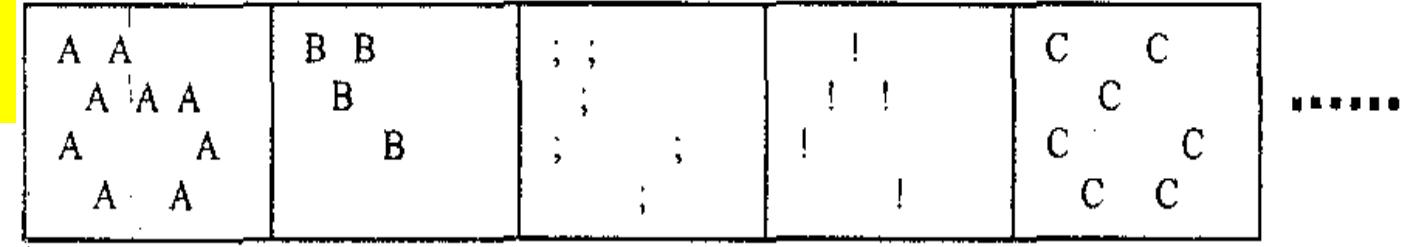
Gestión de polinomios

Se desea una aplicación que de servicio de operaciones con polinomios de una sola incognita (suma, resta, multiplicación, derivada, integral, ...)

Ponte ejemplos de polinomios. ¿Qué es un monomio?

Diseña las estructuras de datos necesarias para albergar un polinomio de una sola incognita.

Trabajo personal Linotipista



Un linotipista trabaja con cajas de moldes de letras mayúsculas y signos ortográficos (estos cinco: “,”, “.”, “;”, “:”, “!”) para formar las secuencias de, a lo sumo mil caracteres que se desea imprimir. El linotipista pone una estúpida condición a sus clientes: puede escribir secuencias de caracteres siempre y cuando la letra a escribir sea mayor o igual que la última letra escrita, excluyendo los signos de puntuación. Por ejemplo, admitiría escribir la cadena “CKL!.,M.Z”, pero no “CLK!.,B.Z”.

En su escritorio, el linotipista tiene cajas con los moldes de las letras ordenadas por orden alfabético, si bien mantiene las cajas con los signos de puntuación intercaladas al azar entre las cajas ordenadas de las letras.

En cada caja sólo tiene moldes de una letra o signo, y cuando se le acaban los moldes de una determinada letra introduce 10 moldes de la misma nuevos – por lo que siempre dispone de letras–, mientras que si se le acaban los de un signo de puntuación no los repone.

Las cajas siempre están unas junto a otras, de manera que cuando se acaban los moldes de un signo ortográfico, saca la caja vacía y junta las que quedan.

Una aplicación evalúa si un determinado texto se puede escribir o no.

Diseña los tipos de datos

Diagrama de descomposición funcional

(*) ejercicio de examen de 2013

U7

Repetición controlada por expresión lógica y cadenas

Flexibilidad

Rosalía Peña



Universidad
de Alcalá



Errores detectados

- input en Python lleva su print incluido
 - letra = input(str(i)+', '+str(j)+': ')
- El programa principal no tiene arg con quien “conversar”, ni PREs a quien ponerle condiciones.
- import biblioteca /
 from biblioteca import recurso1, recurso2
- help(biblioteca)
- Singular/plural
 - lista con los nombres de las direcciones
 - for fila/filas in tablero?

Indentación

- Recomendación PEP8=4 caracteres

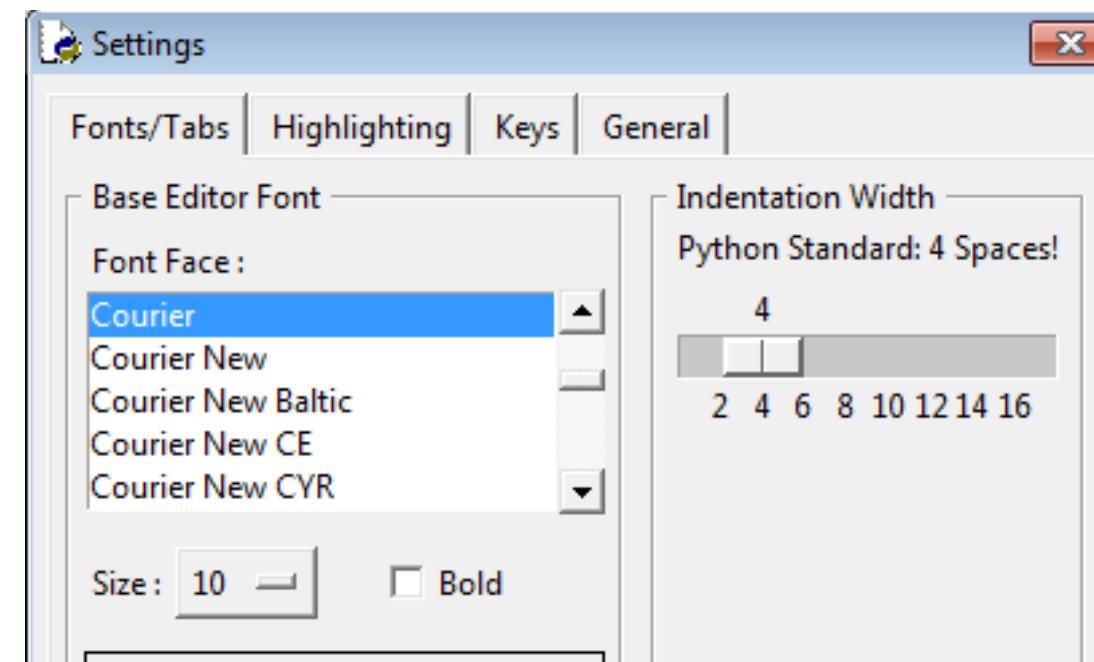
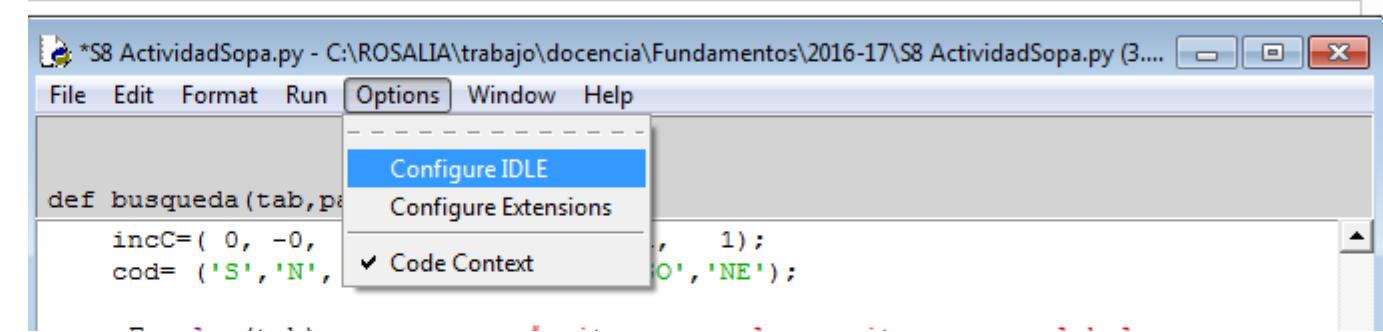


Tabla 4.1. Orden de las secciones de una pieza de código

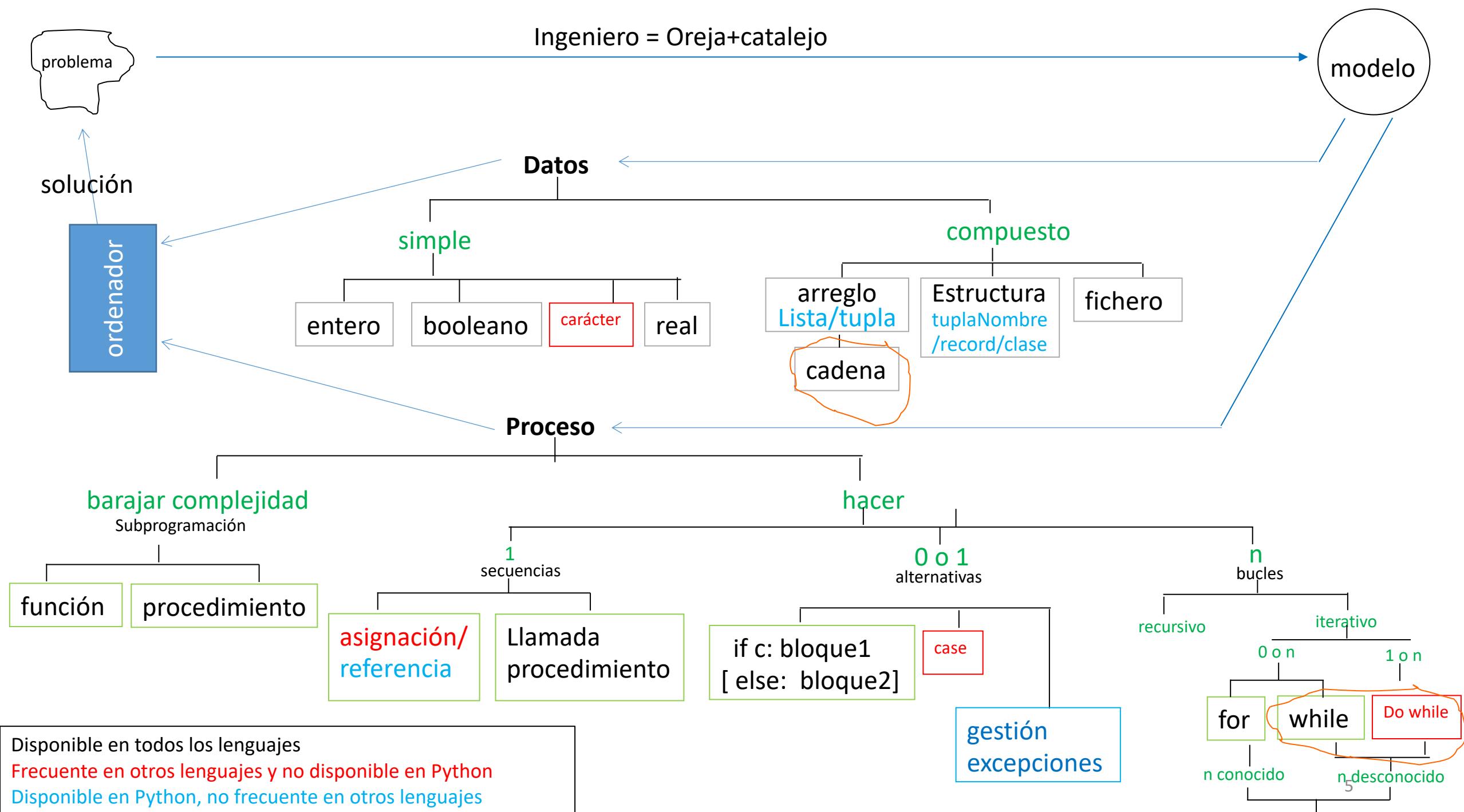
1	Cabecera del programa con documentación
2	Importación
3	Declaración de tipos ¹
4	Declaración de constantes (o no tan constantes) ²
5	Declaración de variables ³
6	Declaración de procedimientos y funciones
7	Cuerpo ejecutivo
7.1	Inicialización de variables
7.2	Entrada del usuario
7.3	Cálculos, gestión, manejo
7.4	Salida al usuario

¹ En Python solo usaremos tipos definidos por el usuario en el capítulo. En esta sección describiremos también como se han diseñado estructuras complejas

² Aunque en Python no existe el concepto de constante, existe en nuestro diseño. Daremos aquí el valor a las “variables” que son constantes, como PI, MAXIMO_ALUMNOS. Recuerda que por convenio, los identificadores de constantes se eligen en mayúsculas.

³ Recuerda que Python no tiene un control de tipo fuerte, es decir no declara el tipo de las variables. En nuestro caso, esta sección no existe.

Ingeniero = Oreja+catalejo



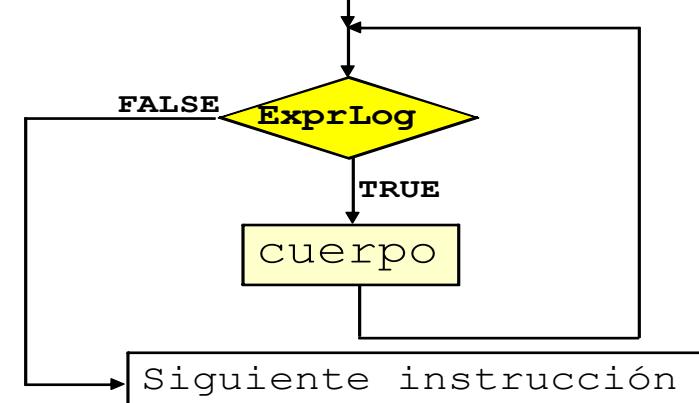
Repetición controlada por expresión lógica

- Sintaxis: `while expresion_booleana:`
`cuerpo`

Ejemplo (discutible):

```
n = 0
while n<10:
    print (n)
    n = n+1
```

```
for n in range(10): print (n)
```



- Si el programa, al llegar al bucle, sabe cuantas veces ha de ejecutar el cuerpo usamos un `for` no `while`
 - Es más cómodo para el programador.
 - Es más eficiente.
 - Es más cómodo de leer → más mantenible, especialmente en su forma:
`for elem in sec: print (n)`
 - pero, además **es más seguro.**
 - Aunque, limitado (a número prefijado de veces)

Repetición controlada por expresión lógica

- Problema: ¿Qué hace? Es casi igual que el anterior
- **Estudio de la convergencia:** Cada pasada está mas cerca del objetivo
- En enteros: teoría de números
- En reales... veamos 2 ejemplos

(1674) Gregory algoritmo (Newton-Raphson \approx 1700): la raíz cuadrada de un número $n \geq 0$,

me invento aprox

mientras $\text{abs}(\text{aprox}-\text{nuevaAprox}) \geq \text{precision}$

$\text{nuevaAprox} = (\text{aprox} + n / \text{aprox}) / 2$

```
n = 0
while n<10:
    print (n)
    n = n-1
```

Casos posibles

Demostración de convergencia



$\text{vieja} = r \rightarrow \text{nueva} = (r + r * r / r) / 2 = r = \text{vieja} \rightarrow \text{acaba}$

$\text{vieja} > r; \rightarrow \text{nueva} = (\text{mayor}_q_r + r * r / \text{mayor}_q_r) / 2 = (\text{mayor}_q_r + x) / 2; 0 < x < r \rightarrow r < \text{nueva} < \text{vieja} \rightarrow \text{acaba}$

$\text{vieja} < r; \text{idem}$

raizNewtonRapson.py

```
1 """probador del cálculo iterativo de la raiz cuadrada de n """
2
3 def sqrtNR(n, precision):
4     """ num, float-->float
5         OBJ: raiz cuadrada de n, con precisión >= precision
6         PRE: n>0, 0<precision<=n
7         """
8     aprox = n/2.0 #me vale cualquier positivo
9     cambio = n # como mínimo su propio valor
10    while cambio>=precision:
11        nAprox = (aprox+n/aprox)/2.0
12        cambio = abs(aprox-nAprox)
13        aprox = nAprox
14        print( 'traza', aprox) #trazador. Deshabilitar
15    return aprox
16 print('raiz =', sqrtNR(9,0.1))
```

SALIDA

traza 3.25

traza 3.0096153846153846

traza 3.000015360039322

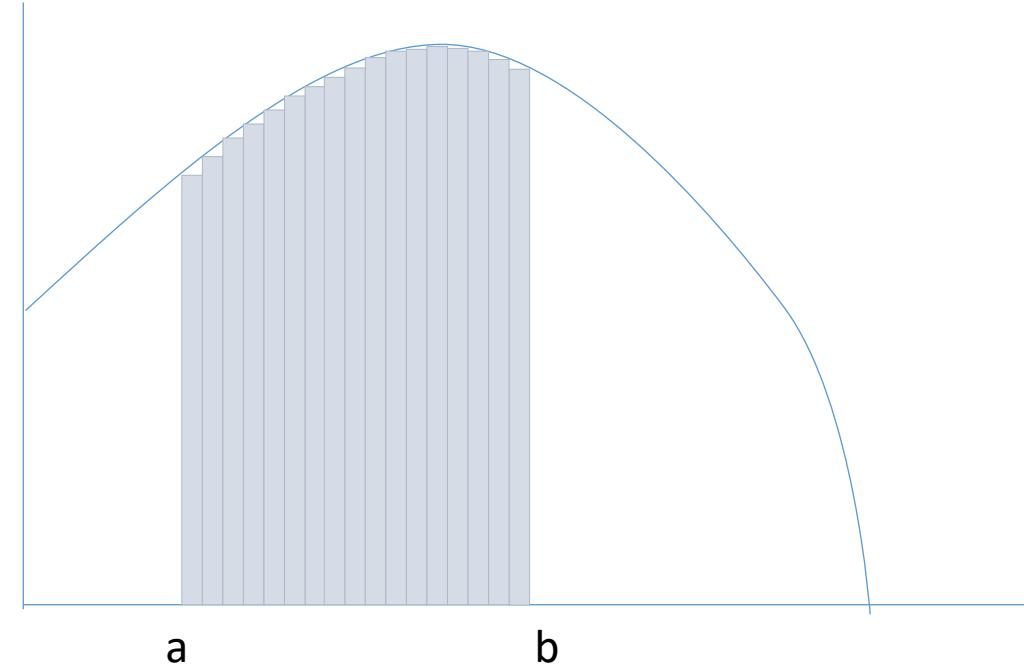
raíz =3.000015360039322

Repetición controlada por expresión lógica

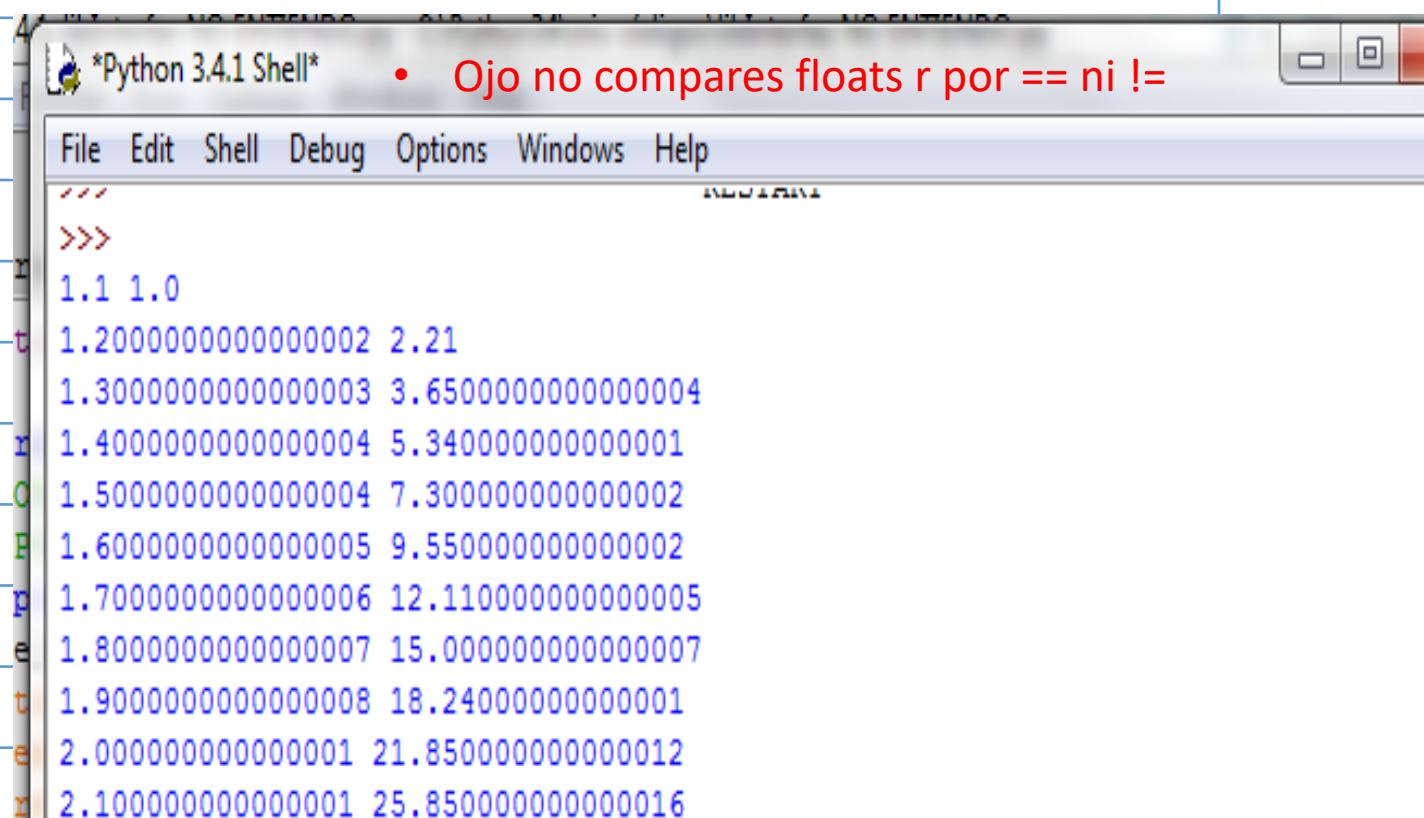
- **Estudio de la convergencia:** Cada pasada está mas cerca del objetivo
- En enteros: teoría de números
- En reales... veamos 2 ejemplos

Integral de x^2 , entre 1.0 y 2.0, paso= 0,01

Acumulo áreas de rectángulos,
empezando por $a^2 * \text{paso}$... y aumento
hasta llegar a $x=b$



```
1 """ Probador de integral X2 entre a y b """
2
3 def integralX2(a,b,inc):
4     """ float,float,float→float
5     OBJ: integral definida entre a y b, en inc incrementos:
6         método de los rectángulos
7     PRE: a<=b, 0<inc<= b-a
8     i = 0
9     x = a
10    while x!= b:
11        i = i+x*x*inc
12        x = x+inc
13        print(x, i)
14    return i
15
16 print(integralX2(1.0,2.0,0.1))
```



```
1 """ Probador de integral X2 entre a y b """
2
3 def integralX2(a,b,inc):
4     """ float,float,float→float
5         OBJ: integral definida entre a y b, en inc incrementos:
6             método de los rectángulos
7         PRE: a<=b, 0<inc<= b-a """
8     i = 0 # acumulador de integral
9     x = a # valor de la abcisa
10    while x <= b:
11        i = i+x*x*inc # acumula el área de un rectángulo
12        x = x+inc
13    print(x, i)
14    return i
15
16 print(integralX2(1.0,2.0,0.1))
```

Bucles (1-n)

Opción	Código ejemplo (en bucle_n.py)
#1 repetir cuerpo del bucle	e= input('Introduce inicial de nombre de estación del año: ') while e.lower() not in ('p', 'v', 'o','i'): e = input('Introduce inicial de nombre de estación del año: ') } } }
#2 Sub-programar cuerpo del bucle	def estacionPedida (): e = input('Introduce inicial de nombre de estación del año: ') error = e.lower() not in ('p', 'v', 'o','i') if error: print(e, 'no es el nombre de una estación del año') print('vuelve a intentarlo.') return error,e error,e = estacionPedida () while error: error,e = estacionPedida () print('hecho')
#3 Forzando entrada	e='imposible' EN GENERAL pedir = True while pedir: e = input('Introduce inicial de nombre de estación del año: ') pedir = e.lower() not in ('p', 'v', 'o','i') if pedir: print(e, 'no es el nombre de una estación del año') print('vuelve a intentarlo.') print('hecho')

Bucles (1-n)

DA TERROR

```
while True:  
    print('hacer lo que sea')  
    if expr_bool: break  
  
print('me sali')
```

Y era INNECESARIO

```
while not expr_bool:  
    print('hacer lo que sea')  
  
    print('me sali')
```

```
condicion= False  
while not condicion:  
    print('hacer lo que sea')  
    condicion = expr_bool  
  
print('me sali')
```

Esquemas de programación

- **Bucle con condición múltiple y discriminación a la salida**

- Ej: Pos de primera aparición de elemento en una lista (si es que está)

```
def posicion(elemento,tupla):  
    """tElemento, tuplaDeElementos-->int  
OBJ: posición de primera aparición de elemento en tupla  
    -1 si no está"""  
    pos = 0                      # posicionar en 1er elemento  
    while pos < len(tupla) and (elemento != tupla[pos]): ← Importa orden condiciones  
        pos = pos+1  
    if pos>=len(tupla): pos = -1 ← ¿por cual salí?  
    return pos  
  
def posicionC(elemento,tupla):  
    """tElemento, tuplaDeElementos-->int  
OBJ: posición de 1ª aparición de elemento en tupla  
    -1 si no está"""\#con la técnica del centinela  
    tupla = tupla+(elemento,)      # añado el elemento  
    pos = 0  
    while (elemento != tupla[pos]): #una sola condición  
        pos = pos+1  
    if pos==len(tupla)-1: pos = -1  
    return pos
```

- **El esquema centinela**

Esquema pedir datos al usuario

```
def cosaPedida (args): # args=restricciones si las hay + msg solicitud
    """ ?,?, str --> tipo de cosa
    OBJ: pide cosa hasta que cumpla restricciones del mundo real"""
    cosa = noEsCosa
    while not exp_bool_tal_q_cosa_esCosa:
        cosa = input('dame una cosa:')
    return cosa
```

nuevaCosa es un nombre también razonable

Ejem: Pedir al usuario ¿Si/NO?

Ejem: la cosa se complica... Pedir al usuario ¿Si/NO?, en inglés, francés, bool,Aceptar/Cancelar..

Esquema pedir datos al usuario

```
def enteroPedido (mini,maxi, msg):  
    """ int,int, str-->int  
    OBJ: pide entero a usuario, entre mini y maxi, mostrando msg  
    PRE: min<=max  
  
    pedir = True  
    while pedir or not (min<=n<=max) :  
        try:  
            n = int(input(msg))  
            pedir = False  
        except:print('debe ser un entero. ', end=' ')  
  
    return n
```

Pedir un entero es tan genérico que no he querido poner un número no válido.
min-1 o max+1 valdrían (podría desbordar)

Esquema pedir datos al usuario

```
def esCosa (cosa) :  
    """tipo de cosa --> booleano  
    OBJ: True si es "cosa" valida  
    return cosa cumple condicionParaSerCosa?  
  
def cosaPedida () :      #restricciones si las hay + msg solicitud  
    """str --> tipo de cosa  
    OBJ: pide cosa hasta que cumpla restricciones del mundo real"""  
    cosa = noEsCosa  
    while not esCosa(cosa) :  
        cosa = input('dame una cosa: ')  
    return cosa
```

Si esCosa no es trivial
subprogramar

```
lectivos=('lunes','martes','miércoles','jueves','viernes')  
def esLectivo(dia) :  
    """str-->bool  
    OBJ: True si dia es lectivo  
    PRE: lectivos definido  
    return dia in lectivos  
    """  
  
def lectivoPedido(msg) :  
    """str-->str  
    OBJ: solicita al usuario dia hasta que sea un lectivo """  
    dia=''  
    while not esLectivo(dia) :  
        dia=input(msg)  
    return dia  
  
print(lectivoPedido('¿que lectivo?: '))
```

Cosa=
DNI
Contraseña
Fecha
Color
Hora
Entero,....

Esquema menú

Desarrolla la aplicación de combinatoria que ofrece un menú en el que el usuario selecciona la operación a ejecutar. Le pide los datos que necesite y le muestra el resultado. La interfaz de usuario deseada es:

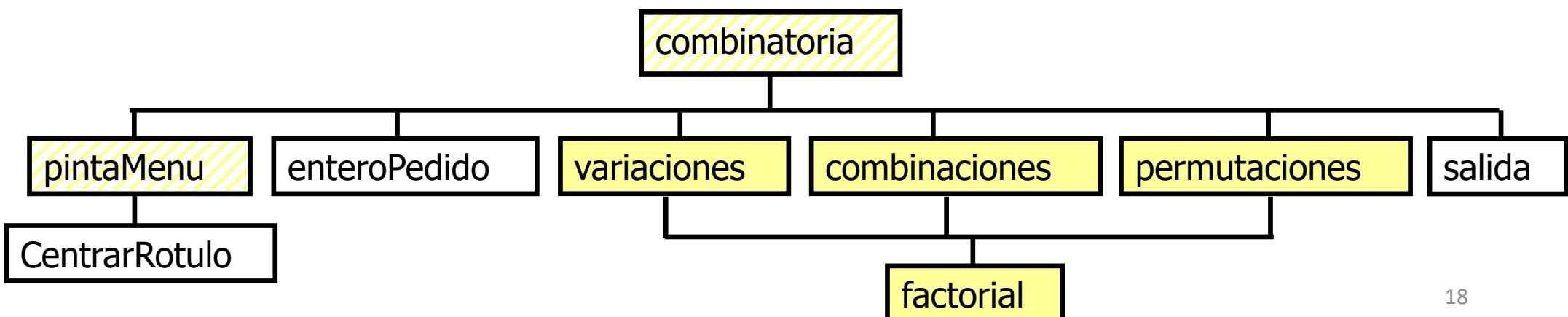
COMBINATORIA

=====

- 1.-Variaciones
- 2.-Combinaciones
- 3.-Permutaciones
- 4.- Salir

Teclee la opción elegida: __

Recordatorio: $V(m,n)=m!/(m-n)!; P(n)=n!; C(m,n)=V(m,n)/P(n); n,m>=0.$



Combinatoria

```
pintaMenu()
opcion = enteroPedido(1,4,'')
if opcion==1:
    m = enteroPedido(1,20,'elementos totales: ','')
    n = enteroPedido(1,m,'elementos a agrupar: ','')
    print('variaciones',m,'de', n,'=', libMat.variaciones(m,n) )
elif opcion==2:
    m = enteroPedido(1,20,'elementos totales: ','')
    n = enteroPedido(1,m,'elementos a agrupar: ','')
    print('combinaciones',m,'de', n,'=', libMat.combinaciones(m,n) )
elif opcion==3:
    m = enteroPedido(1,20,'elementos totales: ','')
    print('permutaciones', m,'=',libMat.permutaciones(m) )
else print('cerrando la aplicación')
```

Un condicional está diseñado para hacer cosas distintas en cada pata... sacad factor común

Cadenas:

- Creación general, vacía, unitaria
- Acceso a elementos: `a[0..n-1]`; hacia atrás `a[-n..-1]`; aborto `i>=n`, operador `slice`
- Operaciones: concatenar; `is` , comparación, `in`
- Funciones: `len`, `min`, `max`
- Métodos: `s.index(x,[i,j])`, `s.count(x)` `help(str)`

Método	Devuelve una...
<code>s.upper()</code>	copia de s con todas las letras en mayúsculas
<code>s.lower()</code>	copia de s con todas las letras en minúsculas
<code>s.replace(old, new[, count])</code>	copia de s con las <code>count</code> ocurrencias primeras de <code>old</code> sustituidas por <code>new</code> . Todas si no especificado
<code>s.strip([chars=_])</code>	copia de s eliminados los caracteres <code>char</code> de cabecera y cola. Por defecto, espacio blanco ' <code>_</code> '
<code>s.swapcase()</code>	copia de s intercambiando minúsculas y mayúsculas
<code>s.title()</code>	copia de s con primera letra de cada palabra en mayúscula y resto en minúscula
<code>s.rsplit(sep=None)</code>	lista de palabras en s, considerando <code>sep</code> como separador

Errores frecuentes en cadenas

- Usar en el programa la posición de las letras en el código ASCII

#MAL

```
letra=input('minúscula: ')
If letra<97 or letra >122:
    print('no es minúscula')
```

#BIEN

```
letra=input('minúscula: ')
If letra<'a' or letra >'z':
    print('no es minúscula')
```

#BIEN

```
letra=chr(randint(ord('a'),ord('z')))
```

- input devuelve un str → cad= ~~str~~(input ('mensaje: '))
-

Trabajo personal:

Construye una función que te devuelva una cadena en mayúsculas, como si no existiera el método upper de Python.

Haz un subprograma que te devuelva m, M, d, o, e si el carácter recibido es minúscula, mayúscula, dígito, u operador

U6: NumPy para implementar array en Python

- Además de tuplas, listas
- ¿por qué? Interpretado/compilado, escrito en C. es más rápido

Trabajo personal dirigido:

MOTIVACION: En una BD, los nombre de personas deben almacenarse sin blancos sobrantes, y siempre la primera letra de cada palabra en mayúscula y el resto en minúscula; de lo contrario, las listas quedarán desordenadas. Haz un subprograma que formatea una cadena: sin blancos innecesarios y la primera de cada palabra en mayúsculas. Resto en minúsculas. (Ejemplo de evolución del pensamiento: 7 pasos)

#1 ¿hay algo hecho?

"""strip de Python: quita blancos de cabeza y cola ¿quita también los de enmedio?

Escribe un código para comprobarlo. """

#2

""" quita todos los blancos de cadena, sin usar operadores especiales de Python. No hagas subprograma, porque esto solo vale para familiarizarnos con cad"""

#3

"""Haz un subprograma que devuelve la primera palabra de una frase omitiendo blancos en cabecera """

sucia= ' Carlos González Soler '

limpia=sucia.strip()

#print(sucia+'*',len(sucia))

#print(limpia+'*',len(limpia))

#No quita blancos de en medio →hacer nuestro subp

aux=""
for letra in sucia:
 if letra!=' ': aux+=letra
print(aux+'*',len(aux))

def primeraPalabra (cad):
 "cad-->cad"

OBJ: quita blancos de cabeza y cola, dejando 1 en medio""

pos=0

aux=""

while pos <len(cad) and cad[pos]==' ': pos+=1

while pos <len(cad) and cad[pos]!=' ':

 aux+=cad[pos]

 pos+=1

return aux

Trabajo personal dirigido:

Punto partida

```
def primeraPalabra (cad):
    "cad-->cad
    OBJ: quita blancos de cabeza y cola, dejando 1 en medio"
    pos=0
    aux=""
    while pos <len(cad) and cad[pos]==' ': pos+=1
    while pos <len(cad) and cad[pos]!=' ':
        aux+=cad[pos]
        pos+=1
    return aux
```

Mientras queden palabras en cad, sacar palabra,+añadir blanco

RESTART: C:\ROSALIA\trabajo\docencia\Fundamentos\apuntes
python\codigos privados\limpiar cadena.py
4 cab centro cola * 17

#4

"""¿consigue este subprograma devolver una cadena sin blancos en cabecera y cola y solo 1 blanco entre palabras? """

```
def cadenaLimpiaBlancos (cad):
    "cad-->cad
    OBJ: quita blancos de cabeza y cola, dejando 1 en medio"
    pos=0
    aux=""
    while pos<len(cad):
        while pos <len(cad) and cad[pos]==' ': pos+=1
        while pos <len(cad) and cad[pos]!=' ':
            aux+=cad[pos]
            pos+=1
        aux+=' '
    return aux
```

#PROBADOR

```
sucia= ' cab centro cola '
limpia=cadenaLimpiaBlancos (sucia)
print(4, limpia+'*',len(limpia))
```

#CASIII, sobran 2 blancos: tras última palabra y tras b de cola

Trabajo personal dirigido:

Punto partida

```
def cadenaLimpiaBlancos (cad):
    "cad-->cad
    OBJ: quita blancos de cabeza y cola, dejando 1 en medio"
    pos=0
    aux=""
    while pos<len(cad):
        while pos <len(cad) and cad[pos]==' ': pos+=1
        while pos <len(cad) and cad[pos]!=' ':
            aux+=cad[pos]
            pos+=1
        aux+=' '
    return aux
```

replanteo casos de prueba

#5 """Haz un subprogr que suprime blancos de cabecera,cola y blancos sobrantes entre palabras """

```
def cadenaLimpiaBlancos (cad):
    "cad-->cad
    OBJ: quita blancos de cabeza y cola, dejando 1 en medio"
    pos=0
    aux=""
    sumBlanco =False
    while pos<len(cad):
        while pos <len(cad) and cad[pos]==' ': pos+=1
        if sumBlanco and pos<len(cad): aux+=' '
        sumBlanco =False
        while pos <len(cad) and cad[pos]!=' ':
            aux+=cad[pos]
            pos+=1
        sumBlanco=True
    return aux
```

#PROBADOR

```
sucia= ' cab centro cola '
sucia2=' sin cola'
limpia=cadenaLimpiaBlancos (sucia)
print(5, limpia+'*',len(limpia))
limpia=cadenaLimpiaBlancos (sucia2)
print(5, limpia+'*',len(limpia))
```

```
RESTART: C:\ROSALIA\trabajo\docencia\Fundamentos\apuntes
python\codigos privados\limpiar cadena.py
5 cab centro cola* 15
5 sin cola* 8
```

Trabajo personal dirigido:

Punto partida

```
def cadenaLimpiaBlancos (cad):
    "cad-->cad
    OBJ: quita blancos de cabeza y cola, dejando 1 en medio"
    pos=0
    aux=""
    sumBlanco =False
    while pos<len(cad):
        while pos <len(cad) and cad[pos]==' ': pos+=1
        if sumBlanco and pos<len(cad): aux+=' '
        sumBlanco =False
        while pos <len(cad) and cad[pos]!=' ':
            aux+=cad[pos]
            pos+=1
            sumBlanco=True
    return aux
```

RESTART: C:\ROSALIA\trabajo\docencia\Fund....
6 Cab Centro C* 12
6 Sin C* 5

#6

"""Haz un subprograma que devuelve la primera letra de cada palabra en mayúsculas el resto en minúsculas y sin blancos sobrantes"""

```
def cadenaFormato (cad):
    "cad-->cad
    OBJ: cad sin blancos sobrantes, primera letra cada pal en mayús"
    pos=0
    aux=""
    sumBlanco =False
    while pos<len(cad):
        while pos <len(cad) and cad[pos]==' ': pos+=1
        if sumBlanco and pos<len(cad): aux+=' '
        sumBlanco =False
        if len(cad)>pos:
            aux+=cad[pos].upper()
            pos+=1
        while pos <len(cad) and cad[pos]!=' ':
            aux+=cad[pos].lower()
            pos+=1
            sumBlanco=True
    return aux
```

Trabajo personal dirigido:

Punto partida

```
def cadenaFormato (cad):
    " " " cad-->cad
    OBJ: cad sin blancos sobrantes, primera letra cada pal en
    mayús"
    pos=0
    aux=""
    sumBlanco =False
    while pos<len(cad):
        while pos <len(cad) and cad[pos]==' ': pos+=1
        if sumBlanco and pos<len(cad): aux+=' '
        sumBlanco =False
        if len(cad)>pos:
            aux+=cad[pos].upper()
            pos+=1
        while pos <len(cad) and cad[pos]!=' ':
            aux+=cad[pos].lower()
            pos+=1
            sumBlanco=True
    return aux
```

"" " ¿algo que optimizar? "" "

```
def cadenaFormato (cad):
    " " " cad-->cad
    OBJ: cad sin blancos sobrantes, primera letra cada pal en mayús"
    pos=0
    aux=""
    long=len(cad)
    sumBlanco =False
    while pos<long:
        while pos <long and cad[pos]==' ': pos+=1
        if sumBlanco and pos<long: aux+=' '
        sumBlanco =False
        if long>pos:
            aux+=cad[pos].upper()
            pos+=1
        while pos <long and cad[pos]!=' ':
            aux+=cad[pos].lower()
            pos+=1
            sumBlanco=True
    return aux
```

Y por supuesto, siempre que edito pruebo

TRABAJO PERSONAL: Para tu biblioteca interfaz prepara:

Real pedido

confirmación pedida

El Documento Nacional de Identidad (DNI) español está compuesto por un número de 8 dígitos, y una letra. El Ministerio de Interior explica cómo se calcula la letra del DNI en <http://www.interior.gob.es/web/servicios-al-ciudadano/dni/calcular-digito-control-nif-nie>.

Haz un subprograma que recibe una cadena y dice si puede ser el DNI de un español.

Haz un subprograma que solicita a un usuario el DNI, hasta que introduzca un DNI válido. Aprovecha el subprograma anterior. Contempla y resuelve la situación en que el DNI tenga menos de 8 caracteres. Permite al usuario escribir la letra en mayúscula o minúscula, pero almacénala en mayúscula.

Microsoft sugiere las propiedades que ha de tener una contraseña segura en:

<http://windows.microsoft.com/es-es/windows-vista/tips-for-creating-a-strong-password>.

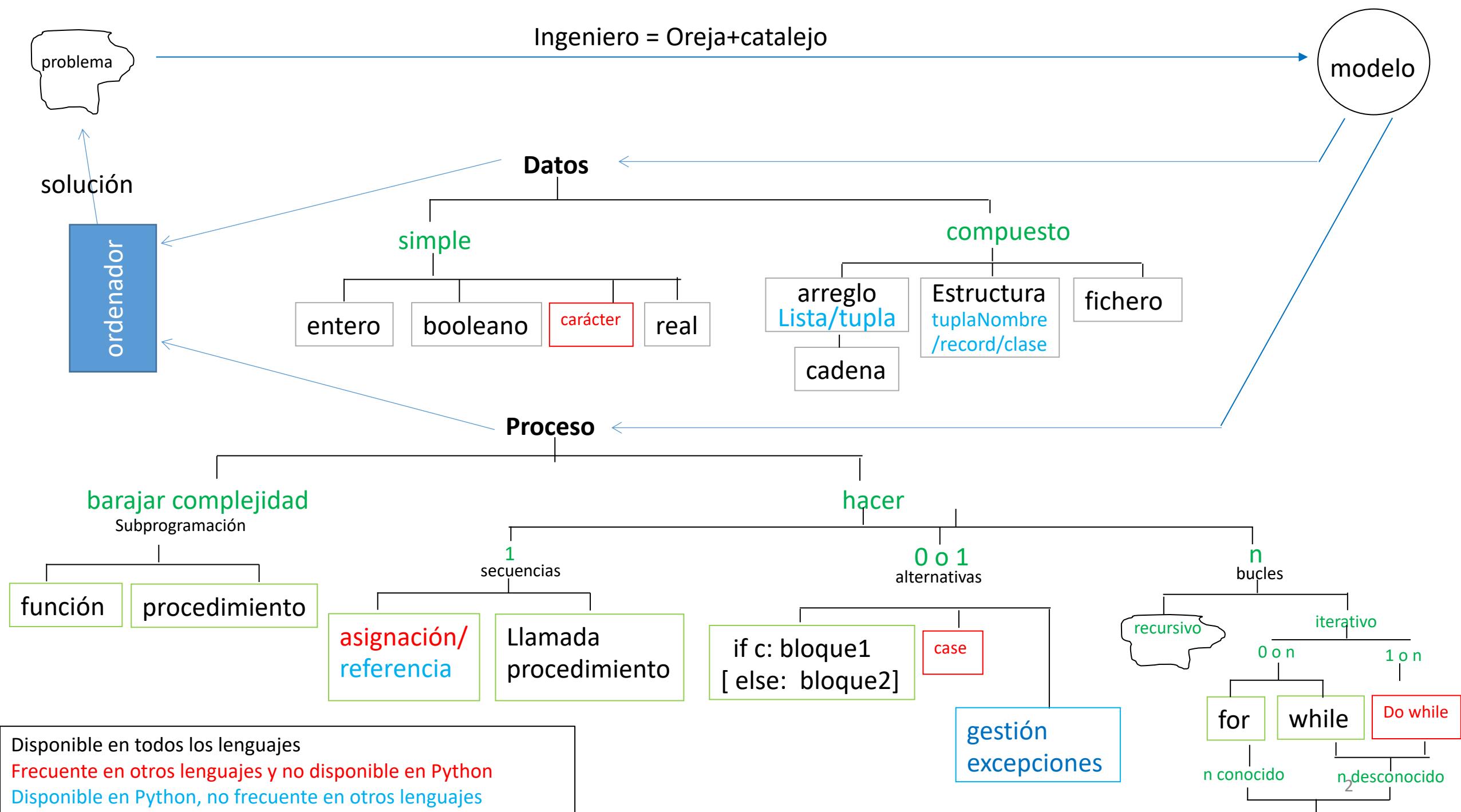
Atendamos a las que se refieren a número y tipos de caracteres. En una segunda versión, atenderemos el resto de recomendaciones. Documéntate a cerca de cómo debe ser una contraseña y haz un subprograma que responda si una cadena de caracteres dada es contraseña válida.

Apoyándote en el ejercicio anterior haz un subprograma que pida al usuario una contraseña hasta que sea válida. Opcionalmente, en la biblioteca getpass de Python encontrarás una función que solicita la entrada de usuario sin hacer eco en la pantalla. Es útil en este momento.

U10

!!!!YA TENEMOS TODOS LOS RECURSOS!!!!

¡¡A JUGAR!!



ERRORES FRECUENTES: identificador de subprograma

- Según el tipo de subprogramas
 - Función bool: ser, estar, tener...
 - Función otroTipo: sustantivo (a ser posible singular)
 - Procedimiento: verbo (excepto calcula o sinónimos)
- Según “OBJ”: Determina la acción mas importante:

```
def validarDNI():
    """....OBJ: pide al usuario un DNI hasta que sea válido"""
    dni = input()
```

```
def DNIpedido():
    """OBJ: pide DNI válido
    ...
    return DNI
```

```
def esValido(dni):
    """True si es un dni válido
    ...
    return boolean
```

ERRORES FRECUENTES: Documentación

Este orden

```
def nombre(arg1, arg2):  
    """ tipoArg1,tipoArg2 →tipoArg3  
    OBJ: frase que incluye nombre, arg1,arg2,arg3  
    [PRE: ...]  
    ...  
    return arg3
```

Un tipo para cada argumento, E/S

lejos, para que se vea bien

ERRORES FRECUENTES: Documentación

```
def nombre(arg1, arg2):
    """ tipoArg1, tipoArg2 → tipoArg3
OBJ: frase que incluye nombre, arg1,arg2,arg3
[PRE: ...]
...
return arg3
```

OBJ explica el papel del
subprograma y
de cada uno de los arg

ERRORES FRECUENTES: Documentación

```
def nombre(arg1, arg2):
```

```
    """ tipoArg1,tipoArg2 →tipoArg3
```

OBJ: frase que incluye **nombre, arg1,arg2,arg3**

[PRE: si no la hay no aparece. Solo hay pres de argi o de constantes o subprogramas en uso global]

...

```
return arg3
```

Que argi es de tipo tipoArgi
ya está dicho

ERRORES FRECUENTES: PRE:

- Si PRE → subp no controla

```
def subp ( arg):  
    """ ....  
  
    PRE: min<=arg<= max           """  
  
    If not min<=arg<=max: .....
```

- Si autónomo no PRE:

```
def alturaCaida ( x0,t):  
    """ ....  
  
    PRE: aceleración=9.8           """  
  
    aceleracion=9.8
```

ERRORES FRECUENTES: Condicionales

- Variables booleanas ¡con todos sus derechos!: también se asignan

```
if letraDNI == codVerificacion[resto]:  
    error = False  
else: error = True
```

```
error = letraDNI == codVerificacion[resto]
```

- Una variable es una expresión de su tipo

```
if error == True:  
    print(' se produjo error')
```

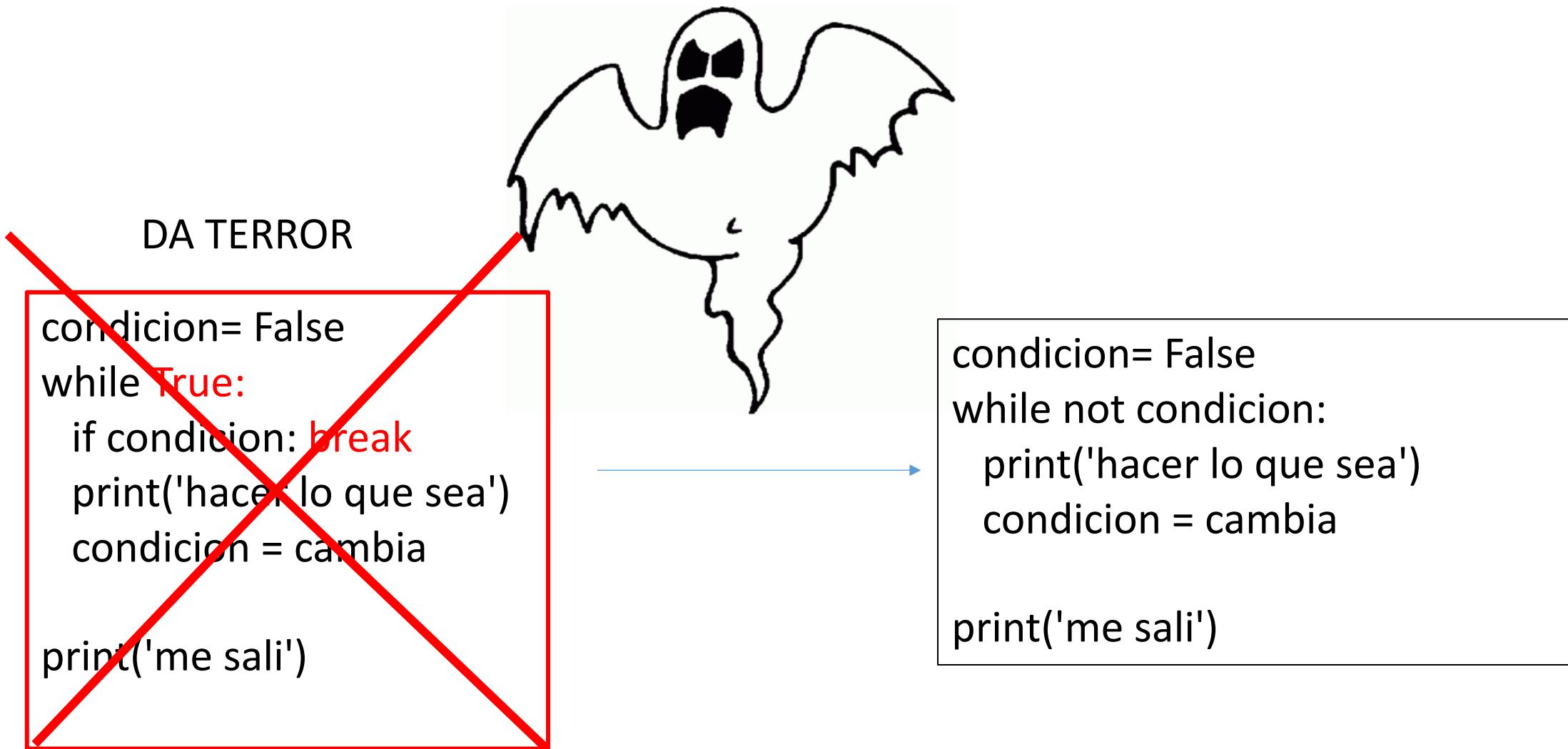
```
if error:  
    print(' se produjo error')
```

- Lo bueno si breve

```
if letraDNI.islower():  
    letraDNI = letraDNI.upper
```

```
letraDNI = letraDNI.upper
```

ERRORES FRECUENTES: control de flujo no estructurado



ERRORES FRECUENTES: input devuelve str

```
nombre=str(input(' nombre: '))
```

```
nombre=(input('nombre: '))
```

```
peso=float(input('introduzca peso en kd: '))
#Cualquier otro tipo requiere try+except
```

ERRORES FRECUENTES: pp, proc, fun

No hay return si no hay arg de salida

Return ??????????

Función tiene 1 y solo un return

y

No toca la interfaz de usuario (excepto pide... que en otros lenguajes es procedimiento)

Procedimiento: IMPRIME → no return

El menú de la gestión de altas era un pp

Lo bueno si breve...:

```
codVerificacion = ['T', 'R', 'W', 'A', 'G', 'M', 'Y', 'F', 'P', 'D', 'X', 'B',  
'N', 'J', 'Z', 'S', 'Q', 'V', 'H', 'L', 'C', 'K', 'E']
```

¿porqué una lista para una constante? → tupla

```
codVerificacion = ('T', 'R', 'W', 'A', 'G', 'M', 'Y', 'F', 'P', 'D', 'X', 'B',  
'N', 'J', 'Z', 'S', 'Q', 'V', 'H', 'L', 'C', 'K', 'E')
```

Mucho mas cómodo: str

```
codVerificacion = 'TRWAGMYFPDXBNJZSQVHLCKE'
```

Mejor recorrer 1 vez que 4

Contexto: La contraseña debe de tener al menos 1 numero, 1 mayúscula, 1 minúscula, 1 otros

```
mayus='ABCD..ZÑÁÉ..Ü'  
minus='abc..zñá..ü'  
digitos='1..9'  
otros='!"·$%...' # horror ¿falta alguno?  
  
for c in contrasenna:  
    if c in mayus: M=1  
  
for c in contrasenna:  
    if c in minus: m=1  
  
for c in contrasenna:  
    if c in digitos: d=1  
  
for c in contrasenna:  
    if c in signos: o=1  
  
return M+m+d+o==4
```

```
mayus='ABCD..ZÑÁÉ..Ü'  
minus='abc..zñá..ü'  
digitos='1..9'  
  
for c in contrasenna:  
    if c in mayus: M=1  
    elif c in minus: m=1  
    elif c in digitos: d=1  
    else: o=1  
  
return M+m+d+o==4
```

Pero Python es comodísimo

```
for c in contrasenna:  
    if c.isupper: M=1  
    elif c.islower: m=1  
    elif c.isdigit: d=1  
    else: o=1  
  
return M+m+d+o==4
```

Usar fuentes ajenas

- En la profesionalidad del ingeniero está hacer **referencia SIEMPRE** a la fuente que ha sugerido la solución que presenta
- Asegúrate de usar una **fuente fiable** (Ej: empresa.org; universidad.es...
no... **pepitoPerez. O niSuPadreLeConece**)
- Incluso en fuentes fiables, se **crítico** con lo que lees y mejóralo,
adáptalo a las necesidades y contexto de tu problema
- **iiiSi copias hazlo bien!!!** (no olvides una línea. No cambies los identificadores intentando disimular la copia, pero
olvidas cambiar alguno)
- Incorpora a tus conocimientos lo que has usado. **Apréndelo**

Biblioteca de gestión de fechas

Un año es bisiesto cuando es múltiplo de 4, excepto que sea múltiplo de 100, aunque también lo es si es múltiplo de 400.
Haz un subprograma que indique si un año es bisiesto.

```
def esBisiesto(anno):  
    """int-->bool  
    OBJ: True si el año es bisiesto anno  
    PRE: año>=1582  
    return anno%4==0 and anno%100!=0 or anno%400==0
```

OJO NO IF

Haz un subprograma que indique el último día del mes recibido como parámetro

```
def ultimoDia(mes):  
    """ int-->int  
    OBJ: máximo día del mes  
    PRE: 1<=mes<=12 ,diasMes definido  
    return diasMes[mes-1] #ODIO que empiece en cero!!!!
```

Haz un subprograma que indica en números árabes el siglo que es una fecha

```
def siglo (anno):  
    """ int-->int  
    OBJ: Siglo correspondiente a anno  
    PRE: anno <> 0  
    s = int(anno/abs(anno)*((abs(anno)+99)//100))  
    return s
```

Haz un subprograma devuelve el romano de un número árabe
(parecido a cambioOptimo en libMonedero)

Biblioteca de gestión de fechas

Declara el tipo fecha, como día mes y año.

```
from collections import namedtuple  
tFecha = namedtuple('Fecha','dia, mes, anno')
```

RESUELTOS EN EL FORO

EsFecha (fecha):

```
""" tFecha-->bool  
OBJ: devuelve True si es una fecha válida  
PRE: f.ann0>1582(año ajuste calendario) """
```

def diaSiguiente (f):

```
""" fecha -> fecha  
OBJ: calcula el dia siguiente a la fecha introducida  
PRE: f es una fecha valida """
```

PROUESTOS EN EL FORO

- fechaPedida
- tiempoTranscurrido (fecha1,fecha2)

PROUESTOS EN EL EQUIPO

Biblioteca de gestión de fechas

Declara el tipo fecha, como día mes y año.

```
from collections import namedtuple  
tFecha = namedtuple('Fecha','dia, mes, anno')
```

RESUELTOS EN EL FORO

EsFecha (fecha):

```
""" tFecha-->bool
```

OBJ: devuelve True si es una fecha válida

PRE: f.anno>1582(año ajuste calendario)

```
"""
```

def diaSiguiente (f):

```
""" fecha -> fecha
```

OBJ: calcula el dia siguiente a la fecha introducida

PRE: debe ser una fecha valida """

PROUESTOS EN EL FORO

- fechaPedida
- TiempoTranscurrido (fecha1,fecha2)

PROUESTOS

- horoscopo (dia,mes)→str
- estación del año(f)→str
- annosTranscurridos (f1, f2)
- diasTrascurridos(f1,f2) →días
- edad = annosTrascurridos (fn, hoy)
- onomásticas (fecha)→string
- esMayorEdad (f)→bool
- cuantosBiestosEntre(a1,a2)→integer
- diaDeSemana(f)→str
- fechaSera (fecha,dias) ->tFecha
- ordenadas(f1,f2) →tFecha,tFecha
- fechaEnLetras(f)→str
- fechaEnBarras(f)→str
- fechaPlana(f)→int
- pintaMes(mes,anno)→nada

fechaPedida

```
def fechaPedida(fMin,fMax,msg):
    """tFecha,tFecha,msg-->tFecha
    OBJ: fecha solicitada al usuario, entre fmin y fmax, mostrando msg
    PRE: fmin, fmax fechas válidas"""
    from libInterfaz import enteroPedido

    fMin,fMax = ordenadas(fMin,fMax) #ahorro una precondición
    print(msg+' ',end='')          #ejem fecha de nacimiento
    valida=False                    #forzar la entrada
    while not valida:
        dia = enteroPedido(1,31,'dia: ','Vuelve a intentarlo:') #con 2 o con 1 msg
        print(' '*(len(msg)+1),end='')
        mes = enteroPedido(1,12,'mes: ','Vuelve a intentarlo:')
        print(' '*(len(msg)+1),end='')
        anno = enteroPedido(fMin.anno,fMax.anno,'año: ','')
        f = tFecha(dia,mes,anno)
        valida = esFecha(f)
    return f
```

Ordenar fechas

```
def fechaAplanada(f):
    """ tFecha --> int
        OBJ: devuelve f en formato numerico aaaammdd
        PRE: es fecha válida
    """
    return f.anno*10000+f.mes*100+f.dia
```

```
def ordenadas(f1,f2):
    """ tFecha,tFecha -->tFecha,tFecha
        OBJ: devuelve f1,f2 ordenadas de menor a mayor
        PRE: son fechas válidas
    """
    f1p = fechaAplanada(f1)
    f2p = fechaAplanada(f2)
    if f1p<f2p: orden = (f1,f2)
    else: orden = (f2,f1)
    return orden
```

Imprime Fecha

```
def imprimeFechaSlash(f):
    """OBJ: imprime una fecha dd/mm/aa
    PRE: f valida
    """
    print(f.dia,'/',f.mes,'/',f.anno, sep='')
```

```
#PROBADOR
imprimeFechaSlash(tFecha(27,2,2003))
```

```
Unidades =('uno','dos'...'nueve')
```

```
Decenas=('diez','veinte',...'noventa')
```

```
Centenas=('mil','dosmil',...'nuevemil')
```

```
Meses=('enero','febrero',..'diciembre')
```

```
# ojo irregulares
```

```
def fechaEnLetra(f1):
```

```
    """ tFecha--> str
```

```
    OBJ: fecha en letra EJ: 03/03/2015 -->
```

```
    PRE: f1 fecha válida
```

```
    """
```

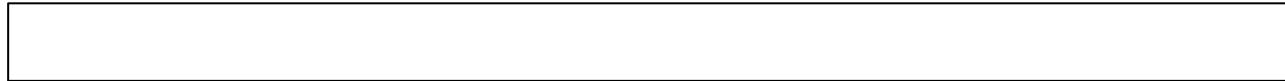
```
return enLetra
```

```
def imprimeFechaEnLetra(f1):
```

```
    """    """
```

```
print (.....)
```

Que fecha será



U9

Estructuras: Agrupación de elementos de distinto tipo

Barajar complejidad

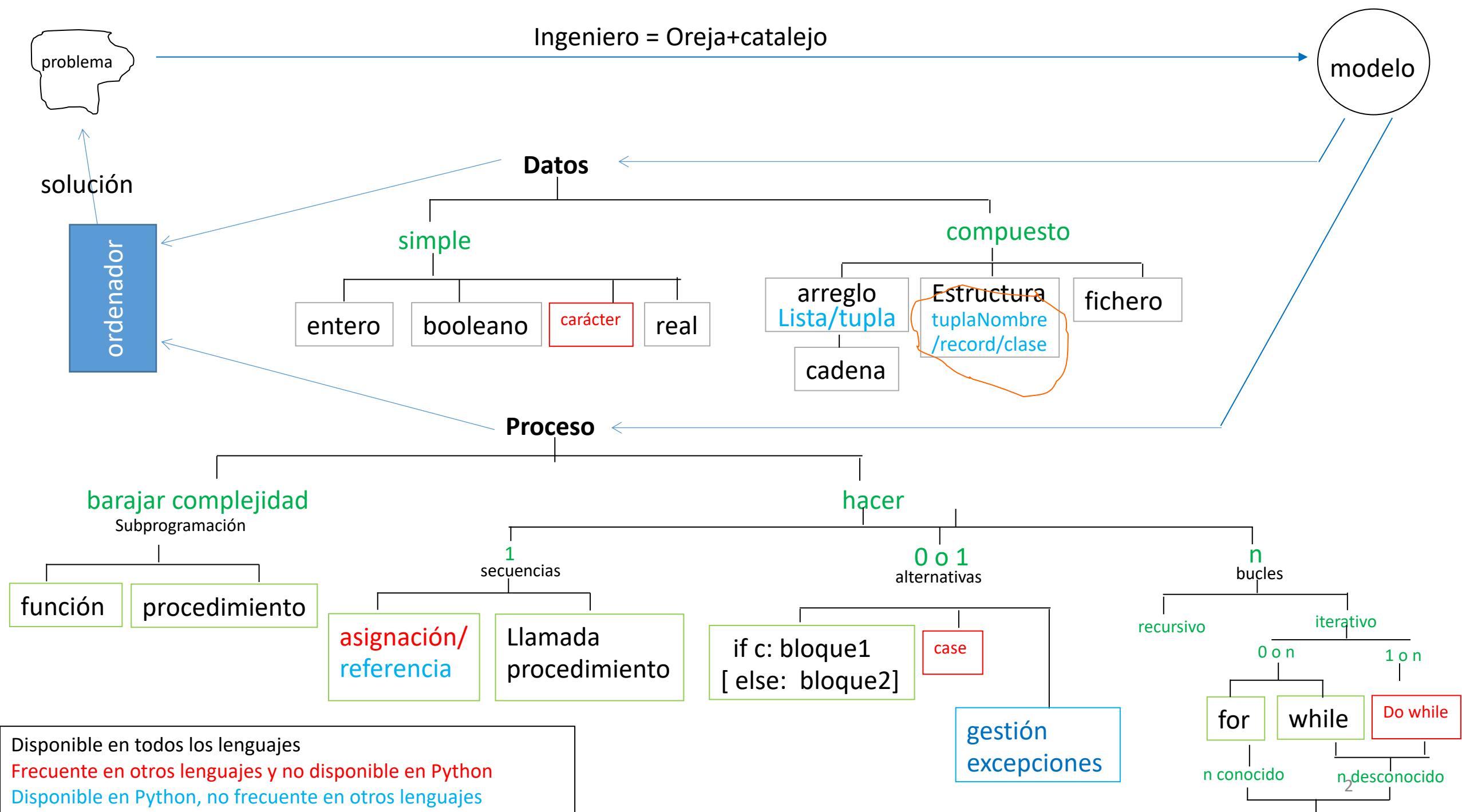
Rosalía Peña



Universidad
de Alcalá



Ingeniero = Oreja+catalejo



STRUCT/STRUCTURE/RECORD

- Elementos todos del mismo tipo → acceso por posición: ARRAY
- ¿sin orden/≠tipo? → acceso por nombre: Tipos definidos por usuario:
→Constructores de tipo.
- Listas y tuplas son tipos en Python (otros ofrecen constructores). Pero ahora tengo que dar nombres. Necesito un constructor.
- Alternativas en Python →

namedTuple, la clase Record, clases de OO “pobres”

inmutables

mutables

- Nuestro interés es diseño de la solución, más que la forma de implementación en un lenguaje concreto.

Clase record

- No es nativo de Python → instalarlo
<https://pypi.org/project/recordclass/>
- **from recordclass import recordclass**
- La sintaxis de la declaración:

```
nombreDelTipo = recordclass(externo, 'campo campo_n')
```

- Ejemplo declaración:

```
tPunto = recordclass('Punto', 'x y')
```

- Creacion de elementos:

```
p=tPunto(2.0,3.0)
```

- Imprimir completo

```
print('puedo imprimir el registro completo',p)
```

```
>>>puedo imprimir el registro completo Punto(x=2.0, y=3.0)
```

- **Convenio: Tipos creados por programador empiezan por t: tFecha, tAlumno**

Clase record

```
from recordclass import recordclass  
tPunto = recordclass('Punto', 'x y')  
p=tPunto(2.0,3.0)
```

- Acceso a elementos

```
print('acceso por notación punto',p.x)  
print('acceso por posición', p[0]) #igual que listas
```

acceso por notación punto 2.0
acceso por posición 2.0

- Record es iterable

```
for coordenada in p:  
    print(coordenada)
```

2.0
3.0

- Modificación elementos

```
p.x=p.x+3  
print(p)      #mantiene resto atributos
```

Punto (x=5.0, y=3.0)

- Los argumentos pueden ser de cualquier tipo, incluso compuesto

```
tSegmento = recordclass('Segmento','pIni, pFin')
```

Biblioteca Punto

```
""" BIBLIOTECA punto: servicios para punto y segmento
* def distancia (p1,p2):distancia entre los puntos p1 y p2
* def tamanno (s): devuelve el tamaño del segmento s
* def puntoAleatorio(): crea un punto aleatorio 0<=x,y<=1
* def estaDentroDeCirculo(p,radio): True si p está dentro
de circulo de radio, con centro 0,0"""

```

```
from recordclass import recordclass
```

```
# declaración de TIPO,
#recomendamos que nombre de tipo empieza por t
tPunto=recordclass('Punto',['x','y'])
tSegmento=recordclass('Segmento','pIni, pFin')
```

Biblioteca Punto

Haz un subprograma que calcule la distancia entre dos puntos.

```
def distancia (p1,p2):  
    """tPunto,tPunto --> float  
    OBJ: distancia entre los puntos p1 y p2"""  
    from math import sqrt  
    return sqrt((p1.x-p2.x)**2+(p1.y-p2.y)**2)
```

#PROBADOR

```
ini = tPunto(1.0,1.0)  
fin = tPunto(2.0,2.0)  
print('distancia entre',ini,'y', fin,'=',distancia(ini,fin))
```

Calcula con él, el tamaño de un segmento pasándole los puntos que lo definen.

```
seg = tSegmento(ini,fin)  
print('el',seg, 'mide %.3f'%(distancia(seg.pIni,seg.pFin)))
```

Biblioteca Punto

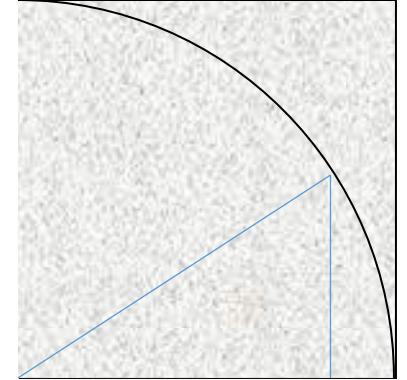
En el lenguaje ordinario hablamos del tamaño de un segmento, no de la distancia entre sus extremos. Haz un subprograma que calcule el tamaño de un segmento.

```
def tamanno (s) :  
    """ tSegmento-->float  
    OBJ: tamaño del segmento s """  
    return distancia(s.pIni,s.pFin)  
  
#PROBADOR  
ini = tPunto(1.0,1.0)  
fin = tPunto(2.0,2.0)  
seg = tSegmento(ini,fin)  
print('el',seg, 'mide %.3f'%tamanno(seg))
```

Subprograma que genera un punto aleatorio.

```
def puntoAleatorio () :  
    """crea un punto aleatorio 0<=x,y<=1  
    from random import random  
    x = random()  
    y = random()  
    return tPunto(x,y) """
```

Biblioteca Punto

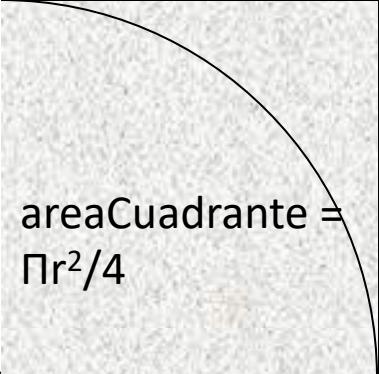


Incluye en el módulo punto una función que calcule si un punto (x,y) está inscrito en una circunferencia de radio r , con centro en el eje de coordenadas. Pista: teorema de Pitágoras.

```
def estaDentroDeCirculo(p, radio) :  
    """True si el punto p está dentro de circulo de radio  
    con centro 0,0"""  
    return p.x*p.x+ p.y*p.y <= radio*radio
```

Método de Montecarlo

$$\text{areaCuadrado} = r * r$$


$$\text{areaCuadrante} = \frac{\pi r^2}{4}$$

Apoyándote en el módulo punto, haz un programa que aproxime π por el método de Montecarlo,
puntos dentro/puntos totales = $\pi r^2 / 4r^2$; despejando $\pi = 4 * \text{puntos dentro/puntos totales}$.

```
"""
    Aproximación de Pi
"""

import libPunto

def piMontecarlo(n):
    """
        valor pi por método de Montecarlo con n puntos de aprox
        PRE: n>1
    """
    cont=0
    for i in range(1,n):
        p = libPunto.puntoAleatorio()
        if libPunto.estaDentroDeCirculo(p,1): cont = cont+1
    return 4* cont/n
```

Método de Montecarlo

Construye una tabla con la aprox de Π para 10, 100, 1.000, 10.000 y 100.000 puntos.

Plantéate antes de empezar, cómo generas el num puntos de cada pasada en función del anterior.

```
print ('n.puntos          pi aprox')
n=10
for i in range (6):
    print ('%7d %21f'%(n,piMontecarlo(n)))
    n=n*10
```

n.puntos	pi aprox
10	3.600000
100	3.240000
1000	3.076000
10000	3.166000
100000	3.137160
1000000	3.142008
>>>	

No es determinista	
n.puntos	pi aprox
10	2.800000
100	3.440000
1000	3.244000
10000	3.132800
100000	3.142800
1000000	3.141228
>>>	

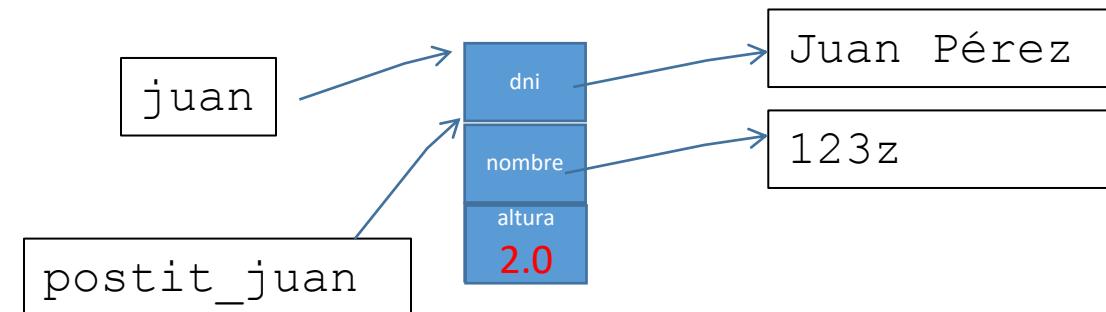
Record es mutable

Diseña el tipo que pueda almacenar una persona, de la que se indicará su nombre, dni y altura .

```
from recordclass import recordclass #, RecordClass
tPersona=recordclass('Persona','dni nombre altura')
#dni:tDni, nombre:tNombreP, altura:float
```

¿Que hace el siguiente código?

```
juan= tPersona ('123z', 'Juan Pérez',1.85)
postit_a_juan=juan
postit_a_juan.altura=2.0      #cambia a juan
print(juan)
```



```
>>>Persona(dni='j', nombre='1', altura=2.0)
```

Record es mutable

```
from recordclass import recordclass #, RecordClass
tPersona=recordclass('Persona','dni nombre altura')
#dni:tDni, nombre:tNombreP, altura:float
```

Haz un subprograma que haga una copia de juan (no un puntero)

```
def copia(p):
    """tPersona-->tPersona
    OBJ: copia de p
    """
    return tPersona(p.nombre,p.dni, p.altura)      # si el atributo es a su vez mutable p.mutable[:]
```

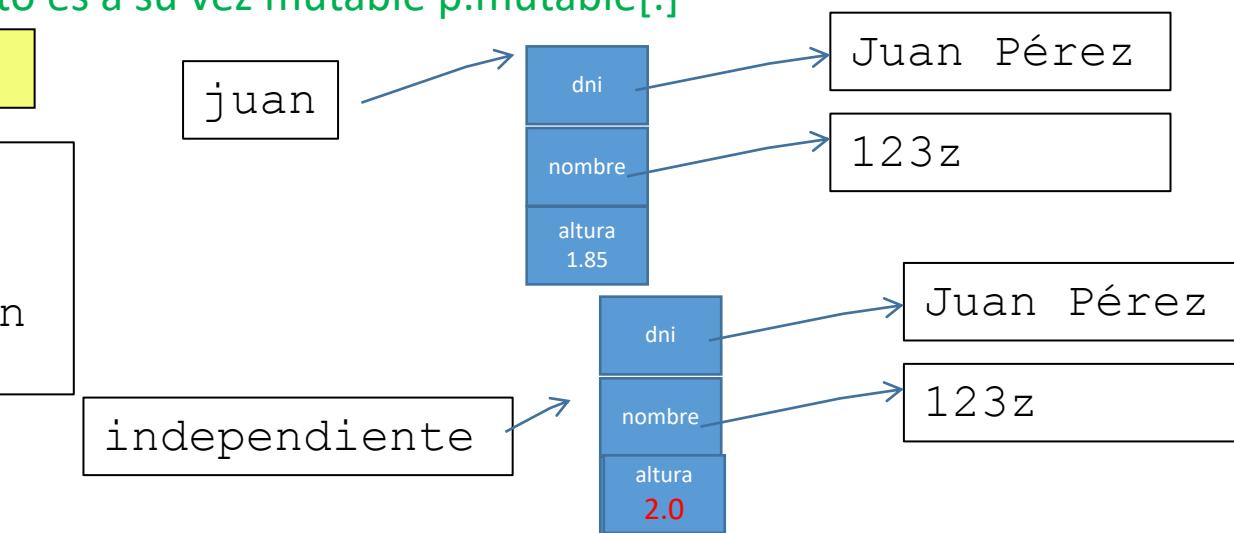
¿Que hace el siguiente código?

```
juan= tPersona ('123z', 'Juan Pérez', 1.85)
independiente=copia(juan)
independiente.altura=2.0      #no cambia juan
print(juan)

>>> Persona(dni='j', nombre='l', altura=1,85)
```

¿Y si uno de los atributos fuera mutable?

¿mapa memoria?



Record es mutable

```
from recordclass import recordclass #, RecordClass
tPersona=recordclass('Persona','dni nombre altura')
#dni:tDni, nombre:tNombreP, altura:float
juan= tPersona ('123z', 'Juan Pérez',1.85)
pedro=tPersona ('123z', 'Pedro López',1.75)
maria=tPersona ('123z', 'María Pérez',1.60)
ana=tPersona ('123z', 'Ana Pérez',1.65)
```

Se desea almacenar, en una sola variable, las personas implicadas en Fundamentos de la Programación. Y otra solo para los profesores. Diseña las estructuras de datos necesarias.

```
#tGrupo=lista(tPersona) orden entrada
```

```
II=(pedro, ana)
IC=(juan, maria)
todos=IC+II
```

EJERCICIOS:

Diseña una estructura de datos para almacenar el color en esta herramienta de dibujo gráfico

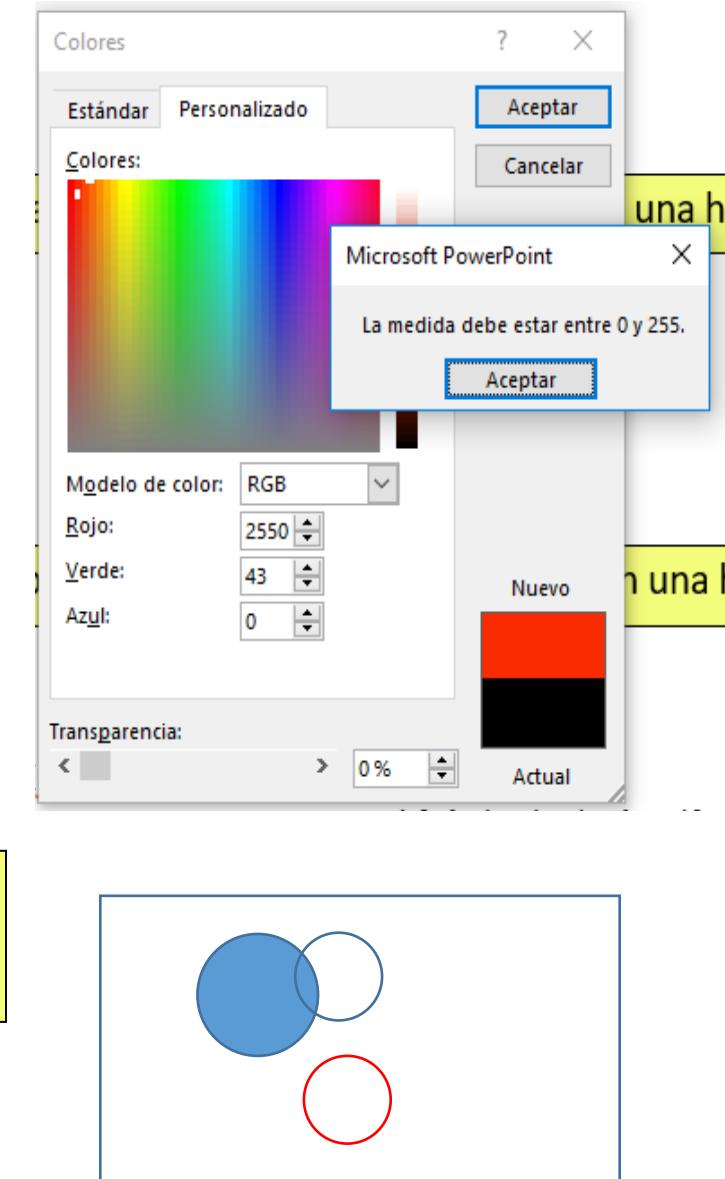
```
#tColorBase 0..255  
#tRGB=tupla(3*tColorBase)
```

haz un subprograma que pide un color, es decir, sus 3 componentes, comprendidas entre 0 y 255

Diseña el tipo de datos para almacenar un círculo en esta herramienta de dibujo gráfico.

```
tCirculo=recordclass('Circulo','origen,radio,cRelleno,cContorno')  
#Circulo=(origen:tPunto+radio:float,cRelleno+cContorno:tRGB')
```

Es necesario documentar las características de cada tipo de datos:
Restricciones de mundo real (atributos y compuesto)
En las secuencias orden entre los elementos+circunstancias especiales



EJERCICIOS:

Directorio de la uah

Todo empleado es responsable del uso adecuado de sus recursos. Se le puede reclamar durante 4 años, si factura de telf. abusiva. ¿Qué hacer cuando se va?

Personal PAS

Seleccione una persona ▾

Personal PDI

Peña Ros, Rosalía

 Ficha personal

Nombre Completo: Peña Ros, Rosalía

Edificio: Edificio Politécnico Superior

Extensión: 6958

Correo: rpr@uah.es

Diseña estructura de datos que almacene la información de la plantilla de la universidad (para el directorio).

```
""""
"Directorio de la uah
from recordclass import recordclass
```

```
tEmpleado=recordClass('Emp','tipo,puesto,edif,telf,email')
#tipo:PAS/PDI, 0<= puesto:int<=numPuestos, edif={R/PT/M/A..}, telf=tTelf, email=tEmail
```

```
#tPlantilla=list(tEmpleado), inserción en orden ape1+ape2+nombre, borrado por marca.
```

EJERCICIOS: Empresa de alquiler de coches

En la PECL1-IC-M-2017 Una empresa de alquiler de coches desea mensualmente hacer la siguiente tabla

Matricu	tipo	alq/h	h	mant	fianza	ingresos	ing_max
0410FJB	furgo	22.00	10.0	75.42	150.00	400.00	
2222HIJ	4p	8.30	72.0	17.71	100.00	789.40	*
MU-1388	5p	10.50	30.0	22.42	50.00	426.50	
8888XYZ	5p	10.50	32.5	23.63	50.00	457.88	
9999XYK	4p	8.30	28.5	19.83	100.00	372.89	
TOTAL			173.0	159.01		2160.40	

(alternativa: tuplas paralelas, cuando no teníamos records)... Diseña las estructuras de datos adecuadas, teniendo en cuenta que el tipo de coche está codificado de 1 a 3, y el precio de alquiler/hora y la fianza dependen solo del tipo de coche.

Diseña estructura de datos que almacene la información de categorías.

```
tTipoCoche=recordclass('TipoCoche', 'cod, nombre_tipo, precio_h, fianza')
#1=< cod int<=3, nombre_tipo:str, 0=<precio_h:float, 0=< fianz:float
#tCategorias= tupla (tTipoCoche), sin orden específico.
```

Diseña estructura de datos que almacene la información de la flota.

```
tCoche=recordclass('Coche','matr, tipo, h_alk, mantenim')
# matr :str, tipo:tTipoCoche.cod, 0=< h_alk:float, 0=<mantenim:float
#tFlota= tupla(tCoche), sin orden especial
```

Como sabes, Python tiene muchos, muchos servicios que otros lenguajes no tienen. Concretamente, la biblioteca datetime tiene todos estos y más, pero la idea es construir la nuestra, como haríamos en C, fortran o Java

EJERCICIOS:biblioteca tiempos

Motivación: Analiza las condiciones que debe satisfacer una fecha. Por ejemplo: 31/01/2014 es una fecha, pero el 29/02/2003 no lo es. Haz un subprograma que indique si una fecha es válida.

Motivación: Haz un subprograma que solicite una fecha al usuario hasta que sea fecha válida.

Motivación: Haz un subprograma que imprima una fecha en formato dd/mm/aa.

Trabajo personal 8.5: En el trabajo personal 6.1 a 6.3 hiciste subprogramas que devolvían el nombre de un mes, y el numeral de un número cardinal. Apoyándote en ellos, haz un subprograma que imprima la fecha en letras, como figura en las cartas y documentos. Ej: 21 de junio de 2015.

Trabajo personal 8.7: Haz un subprograma que devuelva los días transcurridos entre dos fechas.

Trabajo personal 8.8 (como pista está en el libro en seudocódigo): Haz un subprograma que devuelva la fecha que será n días después de una fecha dada. Ej: el 22/06/2015 +10 días será el 02/07/2015.

Pista:`tFecha, int → tFecha`.

Localiza al menos dos subprogramas mas, relevantes para la biblioteca tiempos

Resolución de problemas para ingenieros
con Python estructurado

U9: Recursividad

Otra manera de hacer bucles

Dicen algunos pedagogos que conceptualmente más sencilla

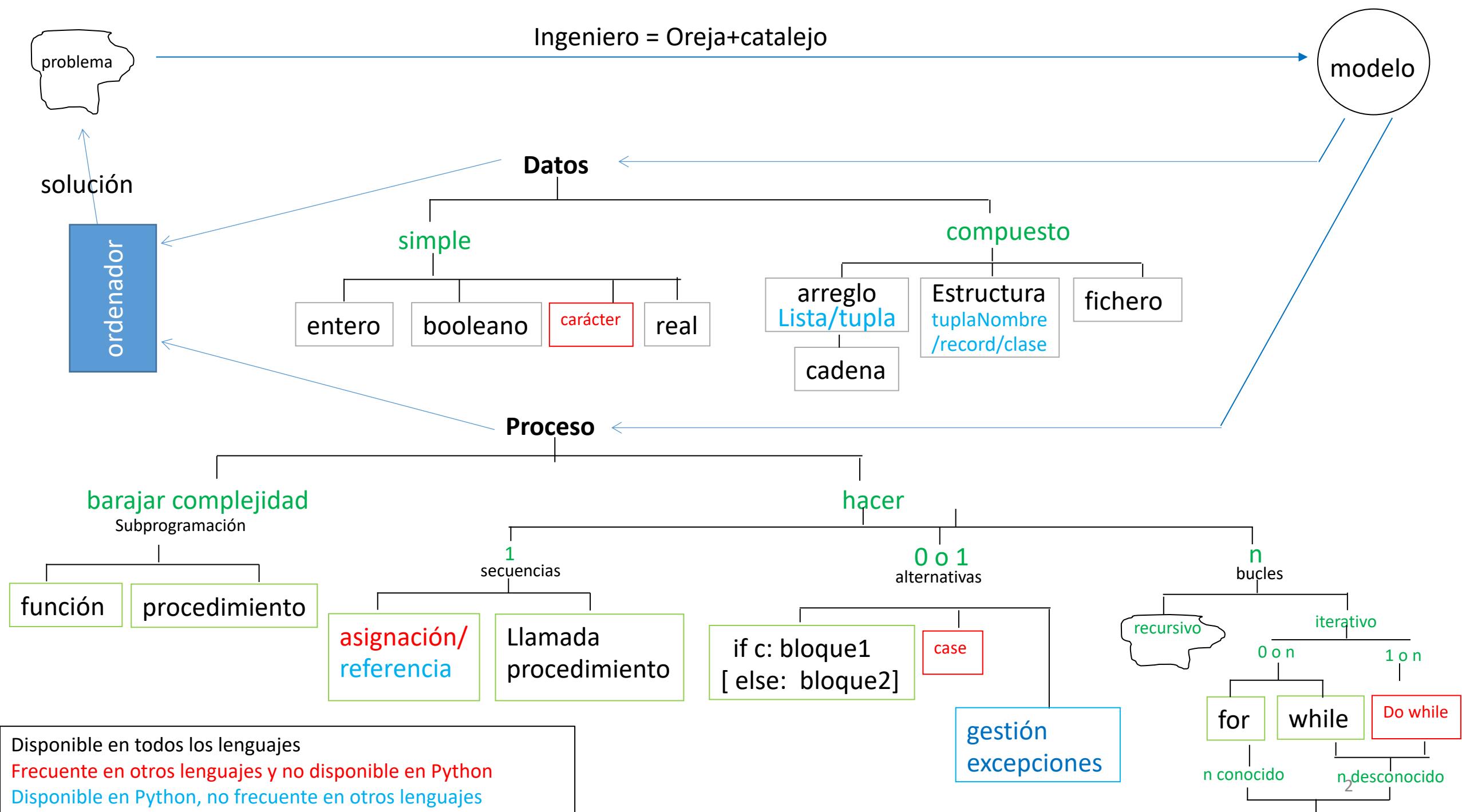
Rosalía Peña



Universidad
de Alcalá



Ingeniero = Oreja+catalejo



Disponible en todos los lenguajes

Frecuente en otros lenguajes y no disponible en Python

Disponible en Python, no frecuente en otros lenguajes

Recordatorio

- 1: `while` permite repetir (iterar) un bloque de instrucciones. OJO programas que no acaban.
- 2: Cuando una pieza de código llama a otra, se apunta el contador del programa en la pila y se transfiere el control al subprograma llamado. Se almacenan en la pila los argumentos y variables locales. Al acabar el subprograma, se liberan las variables, se recoge el puntero de la pila y se reanuda la ejecución por donde iba antes de la llamada.
- 3: Para resolver problemas complejos, hasta ahora, hemos buscado subtareas **diferentes**, más sencillas, que cooperen.

Recursividad:

Enunciar la solución de un problema apoyándonos en uno **semejante**, pero más sencillo:

$$x^n = x \cdot x^{n-1}; x^0 = 1$$

Diseño de un programa recursivo

- Enunciar el caso a resolver en términos de uno, o varios, casos “más sencillos”
- Identificar el (los) caso que se resuelve(n) directamente: **Caso base**
- Garantizar que **la descomposición se aproxime a un caso base**

$$x^n = x \cdot x^{n-1}; \quad x^0 = 1$$

base

recursivo

```
def potenciaR(x, n):  
    """ numero, int-->numero  
    OBJ: x^n  
    PRE: n>=0 """  
    if n==0: p=1  
    else: p = x*potenciaR(x, n-1)  
  
    return p
```

```
print('2^8=', potenciaR(2, 8))
```

- Otra pieza de código realiza la primera llamada

Seguimiento de un programa recursivo

```
""" imprime tupla recursivo """
```

```
def muestraTupla(pos,tupla):
    """int, tupla-->nada
OBJ: muestra elementos de tupla desde pos hasta final
PRE: 0<=pos<=len(tupla)"""

if pos<len(tupla):
    print(tupla[pos],end='')
    muestraTupla(pos+1,tupla)

puente = 'J','V','S','D'
muestraTupla(0,puente)
```

```
muestraTupla(0,puente)                                #copia 1
    imprime J
    muestraTupla(1,puente)                            #copia 2
        imprime V
        muestraTupla(2,puente)                          #copia 3
            imprime S
            muestraTupla(3,puente)                      #copia 4
                imprime D
                muestraTupla(4,puente)          #copia 5
                    4 no es menor que len(puente)
                    Cierra 5a copia de muestraTupla
                    Cierra 4a copia de muestraTupla
                    Cierra 3a copia de muestraTupla
                    Cierra 2a copia de muestraTupla
                    Cierra 1a copia de muestraTupla
                    Acaba el programa principal
```

Seguimiento de un programa recursivo

```
""" imprime tupla recursivo """
def muestraTupla(pos,tupla):
    """int, tupla-->nada
OBJ: muestra elementos de tupla des posa fin
PRE: 0<=pos<=len(tupla)"""

    if pos<len(tupla):
        print(tupla[pos],end=' ')
        muestraTupla(pos+1,tupla)
```

```
""" ??????? """
def sorpresa(pos,tupla):
    """int, tupla-->nada
OBJ: ??????
PRE: 0<=pos<=len(tupla)"""

    if pos<len(tupla):
        → sorpresa(pos+1,tupla)
        → print(tupla[pos],end=' ')
```

www.pythontutor.com/visualize.html#mode=display

Get shared session
are shared sessions?

Python 3.3

```
1 def sorpresa(pos,tupla):
2     """int, tupla-->nada
3     OBJ: ??????
4     PRE: 0<=pos<=len(tupla)"""
5
6     if pos<len(tupla):
7         sorpresa(pos+1,tupla)
8         print(tupla[pos],end=' ')
9
10 sorpresa(0,(3,4,5))
```

[Edit code](#) | [Live programming](#)

Line that has just executed
next line to execute
Click on a line of code to set a breakpoint. Then use the Forward and Back buttons to jump

<< First < Back Step 16 of 20 Forward > Last >>

Print output (drag lower right corner to resize)

5

Frames Objects

Global frame

sorpresa → function sorpresa(pos, tupla)

sorpresa

pos 0

tupla 0 3 4 5

sorpresa

pos 1

tupla 1

sorpresa

pos 2

tupla 2

Return value None

Generate permanent link

9:07

Enunciados recursivos

La sucesión de Fibonacci explica muchos fenómenos de la naturaleza (por ejemplo lee [El diablo de los números de Enzensberger 1997] y [https://es.wikipedia.org/wiki/Sucesi%C3%B3n_de_Fibonacci#La_sucesi%C3%B3n_de_Fibonacci_en_la_naturaleza]).

Inicialmente se definió : $f(n)=f(n-1)+f(n-2)$ con $f(1):=1$, $f(2):=1$
 si bien actualmente se amplía con el término cero:

$$f(n) = f(n-1) + f(n-2) \text{ con } f(0) = 0, f(1) = 1$$

```
def fiboI(n):
    """ int--> int
        OBJ:término n de fibonacci
        PRE: 0<=n<=30 """
    if n==0: f = 0
    else:
        fn_2 = 0
        f = 1
        fn_1 = 1
        for i in range(2,n+1): #incluir n
            f = fn_1+fn_2
            #preparo la siguiente pasada
            fn_2 = fn_1
            fn_1 = f
    return f
```

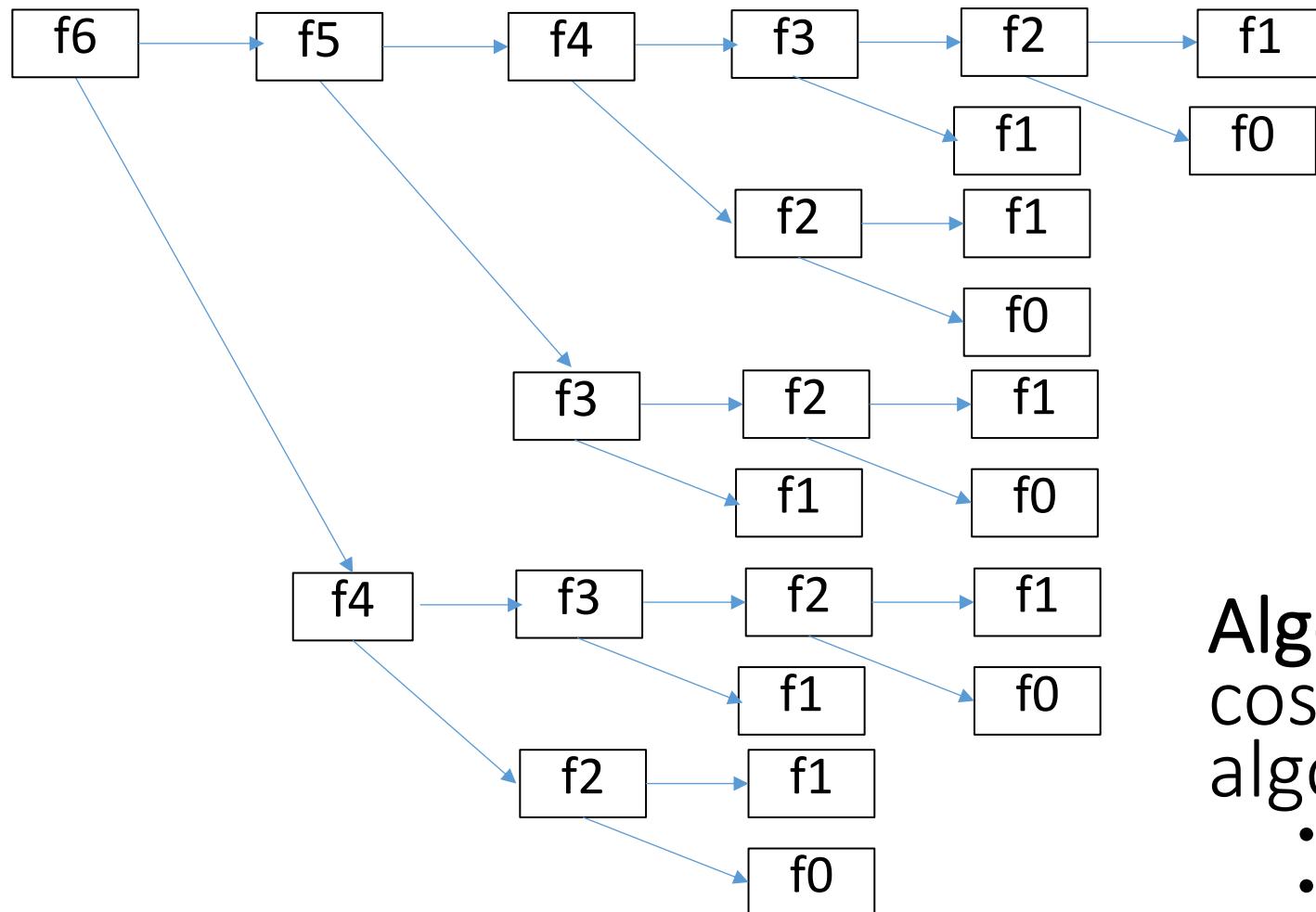
Implementa subprograma Python que calcule el término n.

$$f(n) = f(n-1) + f(n-2) \text{ con } f(0)=0, f(1):=1$$

```
def fiboR(n):
    """ int--> int
    OBJ: término n de fibonacci
    PRE: 0<=n<=30 """
    if n==0:f=0
        elif n==1: f=1
        else: f = fiboR(n-1)+fiboR(n-2)
    return f
```

¿Eficiencia?

$$f(n) = f(n-1) + f(n-2) \text{ con } f(0)=0, f(1):=1$$



```
def fiboR(n):
    """ int--> int
    OBJ: término n de fibonacci
    PRE: 0<=n<=30 """
    if n==0: f=0
    elif n==1: f=1
    else: f = fiboR(n-1)+fiboR(n-2)
    return f
```

Algoritmia: evaluación teórica de costes para comparar eficiencia de algoritmos alternativos:

- Tiempo
- Memoria

Evaluación experimental eficiencia

time.perf_counter/process_time()

$f(n)=f(n-1)+f(n-2)$ con $f(0)=0, f(1):=1$

Y aborté el proceso...

```
""" evaluando el tiempo que tarda Recursividad/Iteración """
from time import perf_counter

print('n          tR          tI          tR-tI          porcentaje')
for n in range (3,48,5):
    t0 = perf_counter()
    f = fiboR(n)
    tR = perf_counter()-t0      #tiempo transcurrido
    t0 = perf_counter()
    f = fiboI(n)
    tI = perf_counter()-t0
    d = tR-tI
    print ('%2d %9.4f %9.4f %9.4f %17.2f'%(n,tR,tI,d,d/tI*100))
n          tR          tI          tR-tI          porcentaje
 3        0.0000        0.0000        0.0000        29.41
 8        0.0001        0.0000        0.0000       282.35
13        0.0006        0.0000        0.0006      3125.64
18        0.0066        0.0000        0.0066     31832.56
23        0.0750        0.0000        0.0749    305345.10
28        0.8195        0.0000        0.8195   3274965.38
33        9.0982        0.0000        9.0982  32046908.47
38      104.5935        0.0000      104.5934 278670814.15
```

Equivalencia

Todos los problemas que tienen solución con una técnica se pueden resolver con la otra

```
def objetivoR(parametros):      #a veces uno mas
    if q(parametros):
        objetivo=valorCasoBase
    else:
        proceso_no_recursivo
        objetivo=h(parametros,objetivoR(g(parametros)))
    return objetivo
```



```
def objetivoI(parametros):
    objetivo= valorCasoBase
    while not q(parametros):
        proceso no recursivo
        objetivo= h(parametros,objetivo)
        parametros=g(parametros)
    return objetivo
```

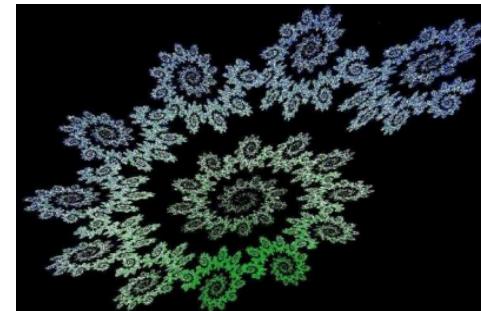
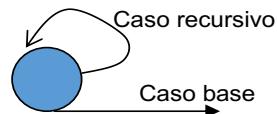
Errores frecuentes: 1) el nombre

La RAE junio 2012 ☹

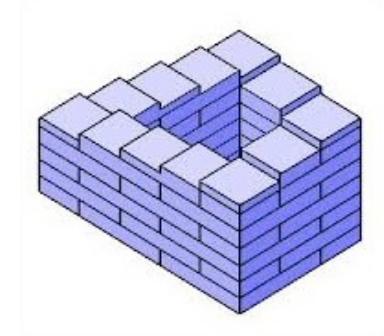
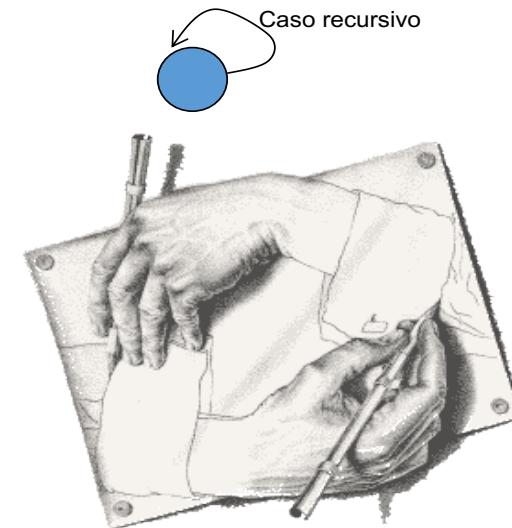
Recurrente:
Se repite con periodicidad



Recursivo:
Se llaman o contienen a si mismos
Hasta un cierto grado



trampantojo



U9: Errores frecuentes:

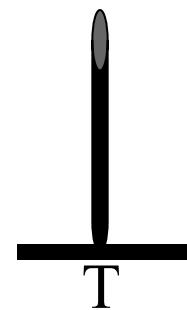
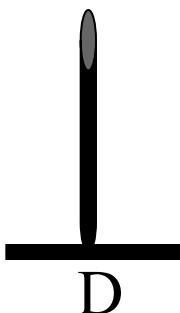
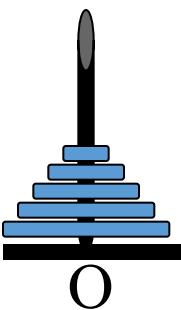
- Bucle de más: la propia recursividad es un bucle
- Olvidar caso base
- No garantizar la convergencia (advertid importancia de PRE)
- ¡¡Pensar: recursividad es “mejor/peor” solución que iteración!!

Resumen

- Recursividad=Repetición por autoreferencia
- Potencia equivalente a la iteración
- Consume más memoria y tiempo que la iteración (n incluso pequeño)
- ¿Cuándo usar recursividad?
 - Cuando simplifique mucho la programación y pruebas
 - Solo si el consumo de recursos no se dispara

RECUSIVIDAD: Torres de Hanoi

Se dispone de 3 soportes y n discos de diferente radio. Inicialmente los discos están apilados en orden creciente en el soporte O (origen). Pasarlos al soporte D(destino), quedando en el mismo orden en que están actualmente. El soporte T(temporal) puede servir de auxiliar. Sólo se puede mover un disco cada vez. En cualquier soporte, en todo momento, cada disco debe tener un radio menor que el que tiene debajo.



Trabajo personal: Es el momento de jugar.

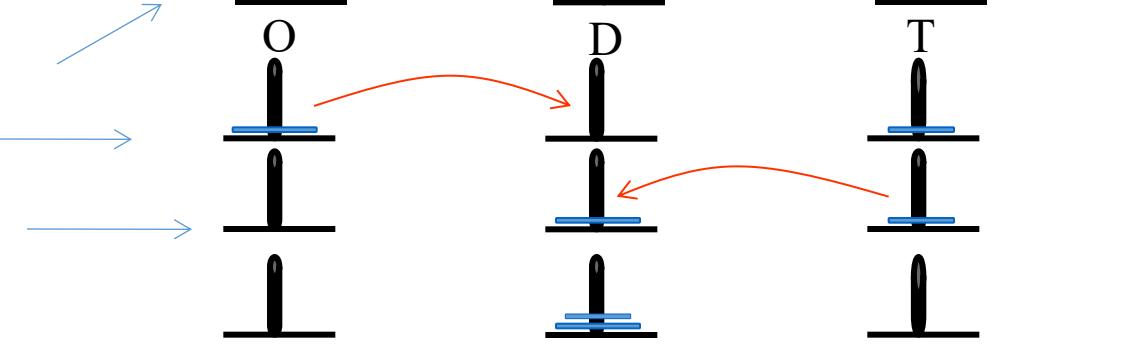
Empieza con un caso sencillo: 2 discos de O a D usando T → también sabes pasar 2 a T usando D
Planteaté pasar 3 discos de O a D usando T → pasa 2 de O a T usando D, y el tercero a D y pasa 2 de T a D usando O

.....

RECUSIVIDAD: Torres de Hanoi

pasar (2,O,D,T):

pasar 1 de O (origen) a T (temp) con D

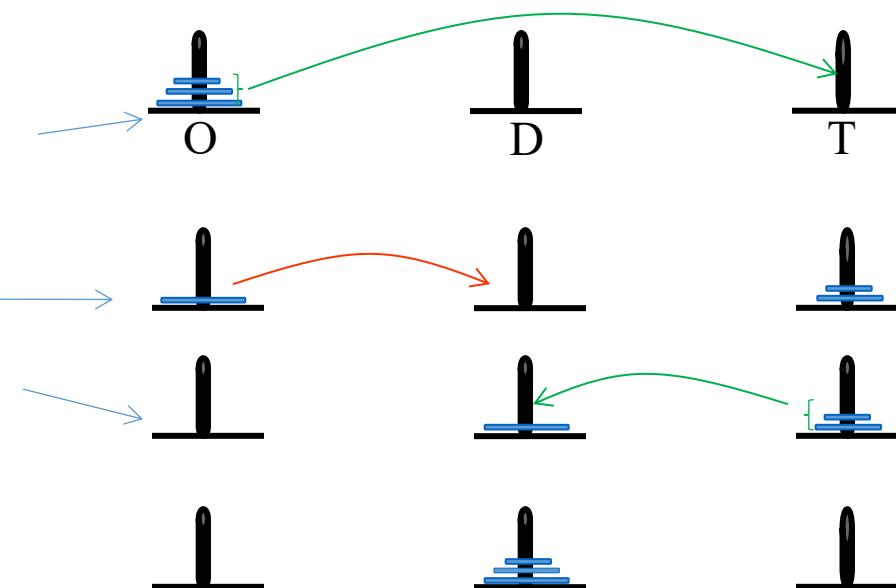


pasar(3,a,b,c):

pasar 2 de O (origen) a T (temp) con D

- pasar 1 de O a D con T
- mover 1 de O a T
- pasar 1 de D a O con T

mover 1 de O (origen) a D (destino)



pasar 2 de T (temp) a D (destino) con O

- pasar 1 de T a O con D
- mover 1 de T a D
- pasar 1 de O a D con T

pasar(num,origen,destino,temp)

7.1 RECURSIVIDAD: Torres de Hanoi

¿con 4 para generalizar?

mover disco 1 de O a T
mover disco 2 de O a D
mover disco 1 de T a D
mover disco 3 de O a T
mover disco 1 de D a O
mover disco 2 de D a T
mover disco 1 de O a T
mover disco 4 de O a D
mover disco 1 de T a D
mover disco 2 de T a O
mover disco 1 de D a O
mover disco 3 de T a D
mover disco 1 de O a T
mover disco 2 de O a D
mover disco 1 de T a D

Pasar 4 de origen a destino usando temp

pasar(4,O, D,T)

Pasar 3 de origen a temp usando destino

pasar(3,O, T, D)

Mover la cuarta de origen a destino

print('mueva disco',4,'de',O, 'a', D)

Pasar 3 de temp a destino, usando origen

pasar (3,T, D, O)

def pasar (n,o,d,t):

"""int,str,str,str-->nada

OBJ: mueve n discos de palo (origen), a palo d (destino), usando temporal

""""

PRE: n>=0

if n>0:

 pasar (n-1,o,t,d)

 print('mueva el disco',n,'de',o,'a',d)

 pasar(n-1,t,d,o)

9 Estudio personal: Algoritmos de ordenación

Apartado 8.4 Búsqueda y ordenación

9 Trabajo personal

Trabajo personal . Que hace el siguiente código

```
def importe(patatas):  
    """int-->int  
PRE: patatas>=0"""  
    if patata<=1: aPagar=1  
    else: aPagar=patatas*importe(patatas-1)  
    return aPagar  
  
tomate=int(input(': '))  
print(importe(tomate))
```

Trabajo personal. El algoritmo de Euclides para el cálculo del máximo común divisor (año 300 A.C) aprovecha la observación de que el $\text{mcd}(\text{mayor},\text{menor})$ debe “cabrer sin resto” en el resto entre mayor y menor, es decir, es también mcd de este resto.

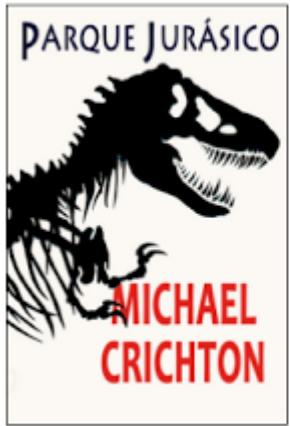
$$\text{mcd}(\text{mayor},\text{menor})=\text{mcd}(\text{menor},\text{mayor}\% \text{menor}).$$

Cuando el resto sea cero habremos acabado.

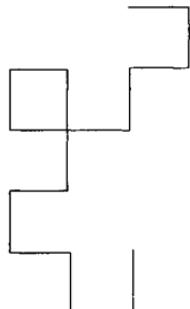
Haz un subprograma recursivo calcule el mcd de dos números.

9 Trabajo personal Curvas del dragón

<http://edupyteron.blogspot.com.es/2013/07/la-curva-de-dragon-el-fractal-de-parque.html>



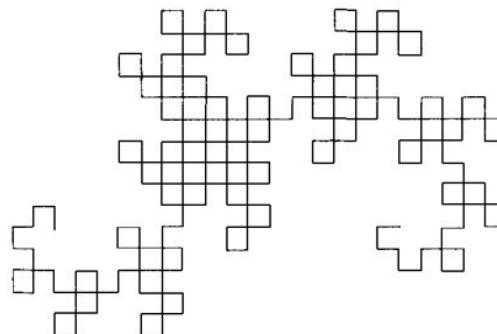
PRIMERA ITERACIÓN



En los primeros dibujos de la curva fractal habrá pocos indicios que permitan conocer la estructura matemática subyacente.

IAN MALCOLM

CUARTA ITERACIÓN



Inevitablemente, empiezan a aparecer inestabilidades matemáticas subyacentes.

IAN MALCOLM

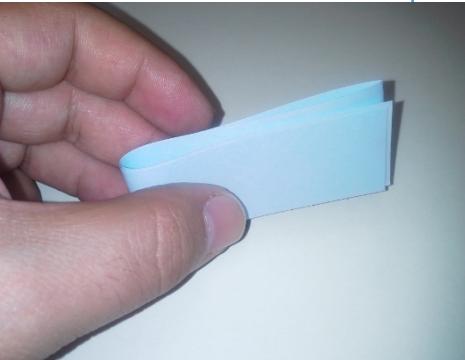
SÉPTIMA ITERACIÓN



La matemática exigirá cada vez más valor para hacer frente a sus inferencias.

IAN MALCOLM

9 Trabajo personal Curvas del dragón

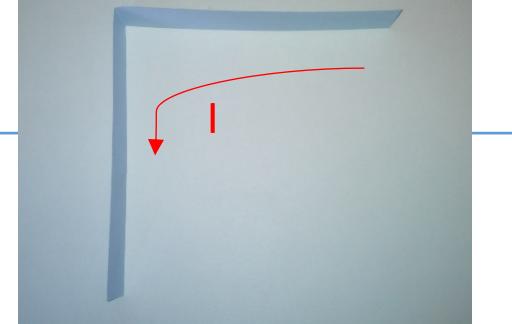


doblo
1

doblo
+I+

→

queda
I



había
2

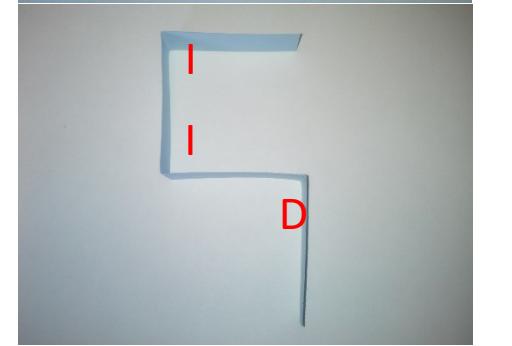
I
I

+I+

I
D

→

IID



había
3

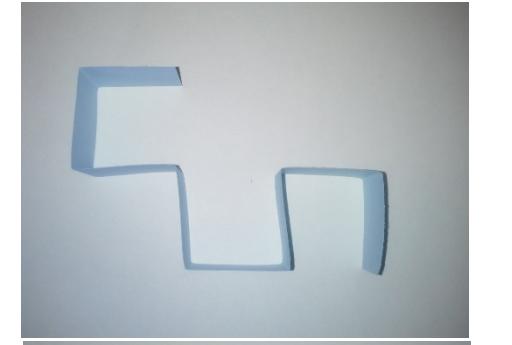
IID
IID

+I+

DII
IDD

→

IIDIIDD



4

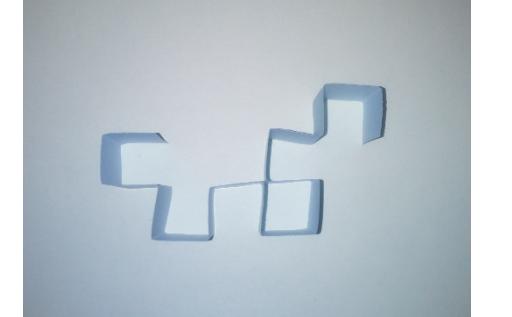
IIDIIDD
IIDIIDD

+I+

DDIIDII
IIDDIDD

→

IIDIIDDIIDDDIDD



5

→ IIDIIDDIIDDDIDDIDDIIDIIDDDIDD

9 Trabajo personal Curvas del dragón

Haz un subprograma recursivo que genere una cadena de caracteres con los giros que debe dar la tortuga (en este subprograma en concreto, no copies la propuesta que hace la página referenciada, pues usa recursos propios de Python que no son generalizables a otros lenguajes y que no hemos visto). El código del manejo de la tortuga es el siguiente:

```
from turtle import *

def pinta_dragon(n, x):
    """Dibuja una curva de dragón de orden n en donde
    cada segmento de la curva es de longitud x.
    """
    fd(x)                                #avanzar x
    for g in curva_dragon(n):
        if g == 'I':
            lt(90)                         #girar left 90º
        else: # g == 'D'
            rt(90)                         #girar right 90º
        fd(x)
```

```
hideturtle()
pensize(3)
color('SteelBlue')
speed('fastest')
setheading(180)
pinta_dragon(8, 15)
done()
```

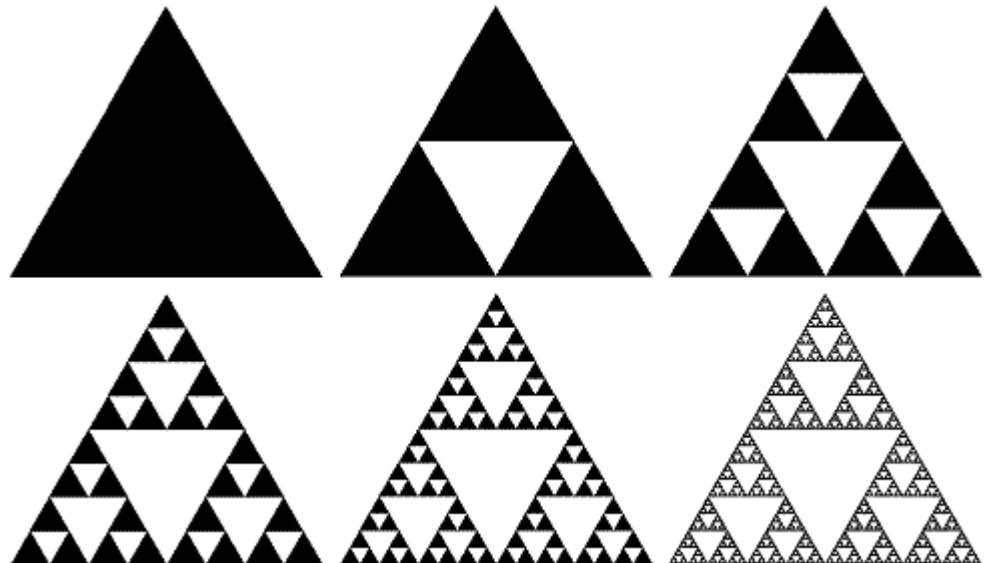
9 Trabajo personal: Mas fractales

En la siguiente dirección tienes la descripción de la curva de koch y el copo de nieve. A por ellos.

<http://graficos.conclase.net/curso/?cap=003b>

Triangulo de Sierpinski

Triangulo de Sierpinski: Se construye pintando un triángulo blanco, con vértices en la mitad de cada uno de los lados, en cada uno de los triángulos negros presentes



- a) Haz un programa que pinte el triangulo de grado n con la tortuga de python
- b) Haz un programa recursivo que indique el nº de triángulos blancos presentes en el paso n.
- c) Haz un programa recursivo que indique la cantidad de tinta de toner que consume la impresora si solo pintamos los bordes de los triángulos en el paso n, sabiendo que para un cm se pintan 50 puntos y cada punto consume $3 \cdot 10^5$ g.
- d) Haz un programa recursivo que calcule la cantidad de tinta consumida si se rellenan los triángulos, a base de 40 líneas por cm.

