



Tecnología de la programación

Raúl García García


Curso 2024/2025

Universidad San Pablo-CEU

Escuela Politécnica Superior


Campus de Montepríncipe

1



Elección de un lenguaje de programación

2




El coste de los errores (*bugs*)

- » Fred Brooks [1]:
 - “...annual lines-of-code programmer productivity is constant, independent of programming language” [2] [3]
 - “...productivity seems constant in terms of elementary statements, a conclusion that is reasonable in terms of the thought a statement requires and the errors it may include” [1] (p. 94)
- » Referencias:
 - [1] F. P. Brooks. *The Mythical Man-Month: Essays on Software Engineering*. Addison Wesley, Boston, MA, 1995.
 - [2] W. M. Taliaferro. Modularity. the key to system growth potential. *IEEE Software*, 1(3):245-257, July 1971.
 - [3] R. W. Wolverton. The cost of developing largescale software. *IEEE Transactions on Computers*, C-23(6):615-636, June 1974.

18-ene.-25 Curso 2024/2025 página 3

3

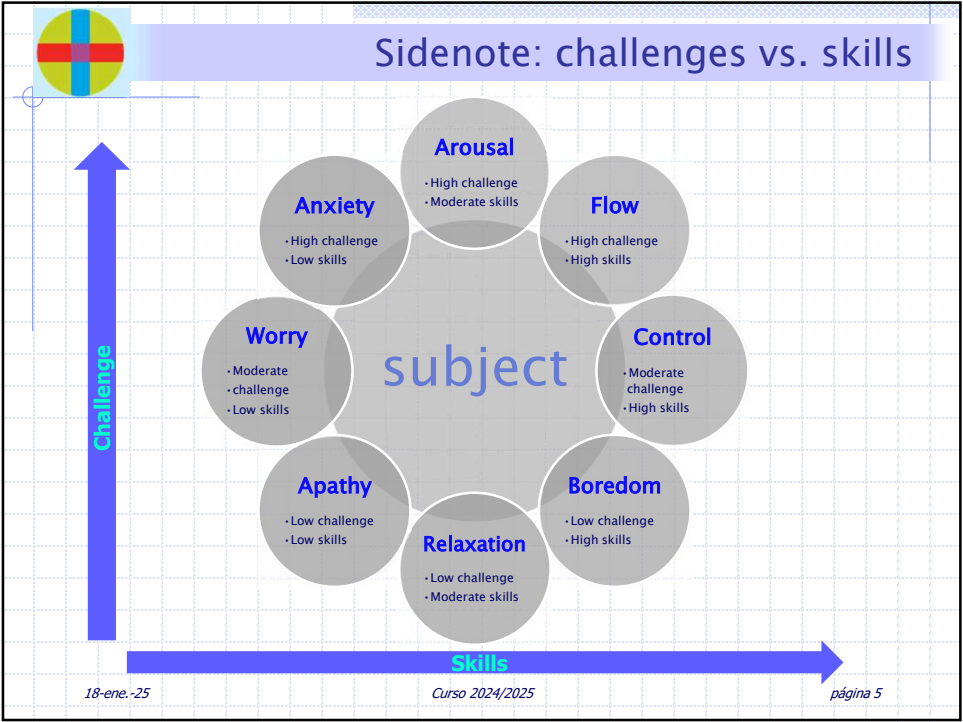


Elegir un lenguaje

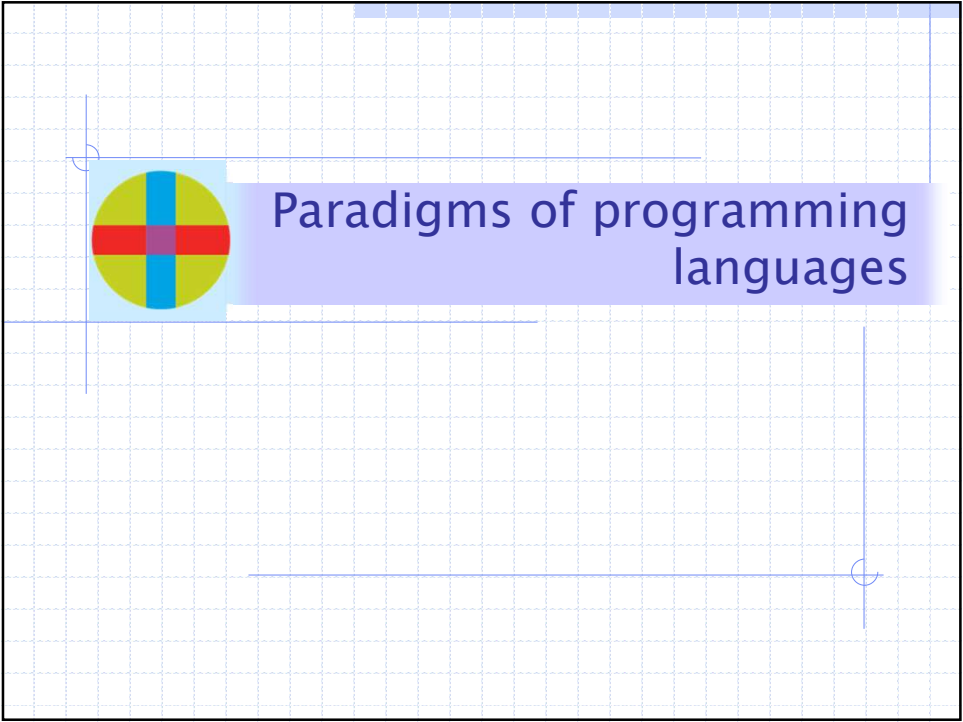
- » Elección del **lenguaje de programación** (en teoría)
 - Según Brooks, cuanto más expresivo (de alto nivel) sea el lenguaje, menos errores cometeremos
 - La alta expresividad suele conllevar peores prestaciones
 - › Balancear coste del programador / coste de las herramientas
 - Puntos importantes:
 - › El **paradigma del lenguaje de programación** debe ser adecuado al problema y a los programadores
 - › Cuanto más rico sea el lenguaje en **estructuras de datos**, más expresividad permite
 - › Un buen **sistema de tipado** bien entendido y usado
- » En la realidad, suelen usarse tres principios:
 - Vincent principle
 - L'Oreal principle
 - Sailor principle

18-ene.-25 Curso 2024/2025 página 4


4



5



6




Principal Paradigms

- » Imperative / Procedural
- » Functional / Applicative
- » Logic
- » Object-Oriented
- » Concatenative / Stack-based
- » Concurrent
- » Scripting

- » In reality, very few languages are “pure”
 - Most combine features of different paradigms


18-ene.-25 Curso 2024/2025 página 7

7




Where Do Paradigms Come From?

- » **Paradigms** emerge as the result of social processes in which people develop ideas and create principles and practices that embody those ideas
 - Thomas Kuhn. “*The Structure of Scientific Revolutions*.”
- » Programming paradigms are the result of people’s ideas about how programs should be constructed
 - ... and formal linguistic mechanisms for expressing them
 - ... and software engineering principles and practices for using the resulting programming language to solve problems



18-ene.-25 Curso 2024/2025 página 8

8




Imperative Paradigm

- » **Imperative (procedural)** programs consists of actions to effect **state change**, principally through assignment operations or side effects
 - Fortran, Algol, Cobol, PL/I, Pascal, Modula-2, Ada, C
 - Why does imperative programming dominate in practice?
- » **Object-Oriented Programming (OOP)** is not always imperative, but most OO languages have been imperative
 - Simula, Smalltalk, C++, Modula-3, Java
 - Notable exception: CLOS (Common Lisp Object System)
- » We'll talk later about OOP

18-ene.-25 Curso 2024/2025 página 9

9




Functional and Logic Paradigms

- » Focuses on **function evaluation**; avoids updates, assignment, mutable state, side effects
- » Not all **functional** languages are “pure”
 - In practice, rely on non-pure functions for input/output and some permit assignment-like operators
 - › E.g., (set! x 1) in Scheme
- » **Logic** programming is based on **predicate logic**
 - Targeted at theorem-proving languages, automated reasoning, database applications (Prolog)
 - Recent trend: declarative programming (XSLT)

18-ene.-25 Curso 2024/2025 página 10

10




Concatenative and Stack-based Paradigms

- » **Concatenative**
 - Programming language in which terms correspond to functions and in which the **juxtaposition** of terms denotes the composition of functions
 - Concatenative programming replaces function application, common in other programming styles, with **function composition** as the default way to build subroutines
- » **Stack-based**
 - Programming languages operate upon one or more **stacks**, each of which may serve different purposes
 - Programming constructs in other programming languages may need to be modified for use in a stack-oriented programming language
- » **Examples:** FORTH, Factor, PostScript, min

18-ene.-25 Curso 2024/2025 página 11

11



Concurrent and Scripting Languages

- » **Concurrent programming** hacen varias cosas a la vez cuts across imperative, object-oriented, and functional paradigms
ADA
- » **Scripting** is a very “high” level of programming
 - Rapid development; glue together different programs
 - Often dynamically typed, with only int, float, string, and array as the data types; no user-defined types
 - Often weakly typed: a variable ‘x’ can be assigned a value of any type at any time during execution
- » Very popular in Web development
 - Especially scripting active Web pages

18-ene.-25 Curso 2024/2025 página 12


python originalmente se hizo para scripting

php igual

suelen ser muy mixtos

pensados para manipular prog completos escritos en otros lenguajes

12



Unifying Concepts


- » Unifying language concepts
 - **Types** (both built-in and user-defined) → especificar una serie de restricciones sobre lo que son y pueden hacer esos datos (int, string, ...)
 - › Specify constraints on functions and data
 - › Static vs. dynamic typing
 - **Expressions** (e.g., arithmetic, boolean, strings)
 - **Functions/procedures**
 - **Commands**
- » Need to study how these are defined syntactically, used semantically, and implemented pragmatically
- » **Important:** see which **entity** is a **first-class** one in the language (function? predicate? object? etc.)

cual es su sintaxis, que significa

saber que es lo que el lenguaje trata como VIP, es lo que hace bien el lenguaje

18-ene.-25 Curso 2024/2025 página 13

13




First-class entities

- » Programming language theorists define a **first-class entity** as a program entity that can be:
 - Created at runtime
 - Assigned to a variable or element in a data structure
 - Passed as an argument to a function
 - Returned as the result of a function
- » Integers, strings, and sometimes real numbers are typical examples of **first-class** entities
- » But classes, functions, objects are not always first-class entities in several programming languages

18-ene.-25 Curso 2024/2025 página 14

14



Design Choices

- » **C**: efficient imperative language with static types
- » **C++**: object-oriented language with static types and ad hoc, subtype and parametric polymorphism
- » **Java**: imperative, object-oriented, and concurrent programming with static types & garbage collection
- » **Scheme**: lexically scoped, applicative-style recursive programming with dynamic types
- » **Standard ML**: functional programming with strict (eager) evaluation and polymorphic type inference
- » **Haskell**: pure functional programming with non-strict (lazy) evaluation
- » **Erlang**: support distributed, fault-tolerant, soft real-time, highly available, non-stop applications

18-ene.-25
Curso 2024/2025
página 15


si quieres hacerlo orientado a objetos pasas por referencia, si no valor

eager -> lo hace en el momento
lazy -> lo hace cuando se lo pide


soft real-time: deja elegir entre computacion normal o en tiempo real.

15 computacion en tiempo real: te da el resultado que puede en una cantidad de tiempo RAPIDA (puede estar mal el resultado). Mejor resultado en tiempo predeterminado.

computacion normal: prog que resuelve BIEN y tarda lo que tarda



Billion-Dollar Mistake



Failed launch of Ariane 5 rocket (1996)

- \$500 million payload; \$7 billion spent on development

Cause: software error in inertial reference system

- Re-used Ariane 4 code, but flight path was different
- 64-bit floating point number related to horizontal velocity converted to 16-bit signed integer; the number was larger than 32,767; inertial guidance crashed

18-ene.-25
Curso 2024/2025
página 16



Estructuras de datos

Un lenguaje es más rico cuanto mejores son sus tipos de datos

Muy importante mirar los tipos de datos de un lenguaje

17



Contenido


- » Conceptos generales
 - Definición de dato
 - Clasificación de los datos
- » Tipos de datos
 - Datos básicos
 - Datos derivados
 - Datos estructurados
 - Estructuras estáticas de datos
 - Estructuras dinámicas de datos

18-ene.-25 Curso 2024/2025 página 18

18

pasa por referencia, creas un identificador nuevo

como son gratis, hacerlos semanticos



Definición de dato

» **DATO**: Unidades de tratamiento dentro de un sistema informático. Los datos de entrada junto con los datos de salida constituyen lo que se denomina **INFORMACIÓN**.

DISEÑO DE UN PROGRAMA

1. Definir *estructuras de datos*

2. Definir *operaciones* sobre los datos.

» Componentes de un dato:

- **IDENTIFICADOR**: nombre utilizado para referenciar al dato.
- **TIPO**: rango de valores que puede tomar el dato.
- **VALOR**: elemento que pertenece al rango de valores según el tipo.

int i = 3

tipo

id

valor

18-ene.-25

Curso 2024/2025


página 19

definir estrucuras de datos es mas importante. Lenguajes imperativos dan a entender que es lo contrario, pero no es asi.

pesa 0

lo que pesan son los valores

TIPO: convierten chorro de bits en la semantica. Pasa a un numero/cadena/... dicen que valores son validos/invalidos. Tambien da limitaciones



Clasificación de los datos

DATOS BÁSICOS	NUMÉRICOS			ENTERO REAL COMPLEJO INTERVALO DECIMAL RACIONAL
				CARÁCTER CADENA LÓGICO ENUMERADO
DATOS DERIVADOS				PUNTERO REFERENCIA
DATOS ESTRUCTURADOS	INTERNOS	ESTÁTICOS	LINEALES	TABLA VECTOR MATRIZ
		DINÁMICOS	LINEALES	LISTA/PILA/COLA CONJUNTO DICCIONARIO
			NO LINEALES	ÁRBOL GRAFO
	EXTERNOS			FICHERO FLUJO BASE DE DATOS
	COMPUESTO			REGISTRO

18-ene.-25


Curso 2024/2025

página 20

16 importantes (Azul)
10 que estan bien saber (grises)

Los que no funcionan con algebra de boole son los ordenadores quanticos

es el nativo del ordenador, importante entenderlo bien



Datos lógicos

» Un **dato lógico** o **booleano** describe a una magnitud que tiene sólo dos valores o estados

- verdadero-falso, si-no, etc.

» Su representación depende del lenguaje de programación:

- true/false, 0/1, 0/-1, V/F, ...

» Se basan en el **álgebra de Boole**

Operaciones:

- Suma lógica (+): OR
- Producto lógico (·): AND
- Negación lógica (-): NOT

AND	0	1
0	0	0
1	0	1

OR	0	1
0	0	1
1	1	1

NOT	0	1
	1	0


18-ene.-25

Curso 2024/2025

página 21

21

algunos van en little y otros en big



Almacenamiento de datos multibyte

» **Endianness**

- El término inglés *endianness* ("extremidad") designa el formato en el que se almacenan los datos de más de un byte en un ordenador
- Tipos de *endianness*:
 - > **Little-endian**: Intel si pongo un 5, lo pongo en la celda mas baja
 - > **Big-endian**: Motorola, PowerPC, SPARC, etc. en la celda mas alta
 - > **Bi-endian**: ARM, MIPS, DEC Alpha a veces uno a veces otro (una pata que al soldar se elige)

Arquitectura little-endian

Dato a enviar: 5

3210

0005

Valor: $0 \times 2^{24} + 0 \times 2^{16} + 0 \times 2^8 + 5$

Arquitectura big-endian

0123

0005

Valor: $5 \times 2^{24} + 0 \times 2^{16} + 0 \times 2^8 + 0$

Dato recibido: 83.886.080

18-ene.-25

Curso 2024/2025

página 22

como hay ordenadores con diferentes procesadores, hay que saber programarlo para que se pueda intercambiar tipos de datos entre procesadores


22

memoria
Bus de direcciones:
Bus de datos: el size depende del size del registro de mem (64bits lo normal)

para evitar movidas, todas las memorias se miden en bytes
todas las celdas son de 1 byte, direcciones tienen cada byte

Si tengo que poner un num de 32 bits en la mem, ocupara 4 celdas.
Empiezas por arriba/abajo depende del design del procesador
Big-endian, little-endian,...

Los procesadores de 8bits no necesita formato de datos



Datos numéricos (I)

» Numéricos

- Numérico entero (con y sin signo)
- Numérico real
- Otros: racional, monetario, complejo, etc.

» Numérico entero (sin signo)

- **Binario puro:** (longitud de la palabra)

25 → 00011001

de 0-255

si entra un 256 -> pasa a 0 overflow
- **Decimal codificado en binario (BCD):** agrupaciones de 4 bits por cada dígito decimal.

25 → 0010 0101

25

25

2

5

cualquier valor que supere a 9 en cada bloque es ilegal


cobble usa BCD

18-ene.-25

Curso 2024/2025

página 23

23



Datos numéricos (II)

» Numérico entero (con signo)

- **Binario (Módulo y signo MS):** signo en el bit más a la izquierda.

pero existen dos 0s
El +0 y el -0

25 → 0 0011001

-25 → 1 0011001

de 0-127

Rango de representación: $-2^{n-1}+1 \leq x \leq 2^{n-1}-1$

- **Complemento a 1 (C-1):** igual a MS para positivos .
Negativos: cambio 0s por 1s.

sigue habiendo dos 0s

25 → 0 0011001

-25 → 1 1100110

0=> 1

1=> 0

Rango de representación: $-2^{n-1}+1 \leq x \leq 2^{n-1}-1$.

18-ene.-25


Curso 2024/2025

página 24

24

3: 0000 0011
-5: 0000 0101
c1: 1111 1010
c2: 1111 1011

1111 1110



Datos numéricos (III)

» Numérico entero (con signo)


- Complemento a (C-2):** igual a MS para positivos.
Negativos: C-1 y suma 1. complemento a 1 + 1
 $25 \rightarrow 0\ 0011001$
 $-25 \rightarrow 1\ 1100110 + 1 = 1\ 1100111$
Rango de representación: $-2^{n-1} \leq x \leq 2^{n-1}-1$
- Exceso a 2^{n-1} :** No hay bit para el signo.
Valor = número + exceso (2^{n-1}).
 $25 \rightarrow 128 + 25 = 153 \rightarrow 10011001$
 $-25 \rightarrow 128 - 25 = 103 \rightarrow 01100111$
Rango de representación: $-2^{n-1} \leq x \leq 2^{n-1}-1$
- Decimal desempaquetado, Decimal empaquetado, ...**

18-ene.-25

Curso 2024/2025

página 25

al hacer el 0
0000000
1111111 + 1 = 0000000
solo un 0



Datos numéricos (IV)


» Numérico real

- Números muy grandes o muy pequeños
- » Representación:
 - Punto decimal:** dígitos del 0 al 9 más el punto decimal:
 $25.32 \quad -48759.4596 \quad -0.41526 \dots$
 - Notación científica o exponencial:** "mantisaEcaracterística"
Número = mantisa * Base de exponenciación ^{característica}
 - › **Mantisa:** número real
 - › **Base de exponenciación (E):** base decimal ('e').
 - › **Característica:** exponente correspondiente a un número entero con su signo

18-ene.-25

Curso 2024/2025

página 26



Datos numéricos (V)

» Numérico real (notación científica o exponencial)

2.5E3 → 2.5×10³

0.25E-2 → 0.25×10⁻²

• Representación interna (norma IEEE 754)

> Base: 2 ó 10.

> Mantisa: binario puro (MS, C-1, C-2, 0.M / 1.M)

> Característica: exceso a 2ⁿ⁻¹-1 (ó MS)

Simple precisión (32 bits):

▪ Signo: 1 bit (pos. 31)

▪ Exponente: 8 bits (pos. 23 a 30)

▪ Mantisa: 23 bits (pos. 0 a 22)

Doble precisión (64 bits):

▪ Signo: 1 bit (pos. 63)

▪ Exponente: 11 bits (pos. 52 a 62)

▪ Mantisa: 52 bits (pos. 0 a 51)

0 0 1 0 1 1 0 1 1 1 1 0 0 1 0 1 1 1 0 0 0 0 1 1 0 1 0 1 0 0 0 0 0

SIGNO

EXPONENTE


MANTISA

18-ene.-25

Curso 2024/2025

página 27

27



La norma IEEE 754: valores normalizados

» Valor V de un número binario IEEE 754 de exponente E y mantisa M:

• El exponente E es un número binario representado en exceso a n (n depende de la precisión del número)

• La mantisa M representa una fracción binaria en notación normalizada (hay que sumar una unidad ya que se asume que el primer bit de la mantisa, potencia 0, vale 1)

> Por ejemplo, si el contenido de la mantisa es M=0.254 el valor de la mantisa es 1+0.254

• Ejemplo (precisión simple):

0 01111111 110100000000000000000000 = +1.8125

S exponente mantisa

S = 0 (+)

E = 127 → E - 127 = 0


M = 1×2⁻¹ + 1×2⁻² + 0×2⁻³ + 1×2⁻⁴ + ... = 0.8125

18-ene.-25

Curso 2024/2025

página 28

28




La norma IEEE 754: ceros

- » Definición del **cero**: puesto que el número se supone almacenado en forma normalizada, no es posible representar el cero (la mantisa se supone siempre precedida de un 1)
- » Por esta razón se convino la siguiente representación del cero:

$$\begin{array}{l} \underbrace{0}_{S} \underbrace{00000000}_{\text{exponente}} \underbrace{000000000000000000000000}_{\text{mantisa}} = +0 \\ \\ \underbrace{1}_{S} \underbrace{00000000}_{\text{exponente}} \underbrace{000000000000000000000000}_{\text{mantisa}} = -0 \end{array}$$

18-ene.-25
Curso 2024/2025
página 29

29




La norma IEEE 754: infinitos

- » **Infinitos**: se ha convenido que cuando todos los bits del **exponente** están a 1 y todos los de la **mantisa** a 0, el valor es $\pm\infty$ (según el valor de S)
- » Esta distinción ha permitido a la norma definir procedimientos para continuar las operaciones después que se ha alcanzado uno de estos valores (después de un *overflow*)

$$\begin{array}{l} \underbrace{0}_{S} \underbrace{11111111}_{\text{exponente}} \underbrace{000000000000000000000000}_{\text{mantisa}} = +\infty \\ \\ \underbrace{1}_{S} \underbrace{11111111}_{\text{exponente}} \underbrace{000000000000000000000000}_{\text{mantisa}} = -\infty \end{array}$$

18-ene.-25
Curso 2024/2025
página 30

30



La norma IEEE 754: valores desnormalizados

» En este caso no se asume que haya que añadir un 1 a la mantisa para obtener su valor

- Se identifican porque todos los bits del **exponente E** son 0 pero la **mantisa M** presenta un valor **distinto de cero** (en caso contrario se trataría de un cero)
- El primer bit de la mantisa es la potencia 0, no la -1
- Se trata de números muy pequeños cercanos al cero

0 00000000 011010000000000000000000 = **4.77544580** $\times 10^{-39}$

S exponente mantisa

S = 0 (+) E = 0 (desnormalizado)


M = $0 \times 2^0 + 1 \times 2^{-1} + 1 \times 2^{-2} + 0 \times 2^{-3} + 1 \times 2^{-4} + \dots = 0.8125$

V = $\pm (0+M) \times 2^{E-127} = + (0+0.8125) \times 2^{-127} =$

= **4.77544580** $\times 10^{-39}$

18-ene.-25 Curso 2024/2025 página 31

31



La norma IEEE 754: valores no numéricos

» Denominados **NaN** ("Not a number")

- Se identifican por un **exponente** con todos sus valores a 1, y una **mantisa distinta de cero**
- Existen dos tipos: **QNaN** ("Quiet NaN") y **SNaN** ("Signaling NaN"), que se distinguen dependiendo del valor 0/1 del bit más significativo de la **mantisa**
 - QNaN tiene el primer bit a 1, y significa "Indeterminado" (0/0)
 - SNaN tiene el primer bit a 0 y significa "Operación no válida" (underflow, overflow, número complejo, exceso de precisión)

0 11111111 100001000000000000000000 = **QNaN**


S exponente mantisa

1 11111111 00100010001001010101010 = **SNaN**

S exponente mantisa

18-ene.-25 Curso 2024/2025 página 32

32



La norma IEEE 754: cálculo de valores

» El **valor decimal** V de una representación binaria estándar, puede calcularse mediante las siguientes fórmulas:

$$V = \pm (1+M) \times 2^{E-127}$$
$$V = \pm (0+M) \times 2^{E-127}$$
$$V = \pm (1+M) \times 2^{E-1023}$$
$$V = \pm (0+M) \times 2^{E-1023}$$

simple precisión normalizado

simple precisión no normalizado

doble precisión normalizado

doble precisión no normalizado

1 10001101 011010000000000000000000

S exponente mantisa

$S = 1 \text{ (-)}$

$E = 141 \rightarrow E - 127 = 14$

$M = 0 \times 2^{-1} + 1 \times 2^{-2} + 1 \times 2^{-3} + 0 \times 2^{-4} + 1 \times 2^{-4} \dots = 0.40625$


$V = \pm (1+M) \times 2^{E-127} = - (1+0.40625) \times 2^{14} = -23040$

18-ene.-25

Curso 2024/2025

página 33

33



La norma IEEE 754: rangos

» Rangos de representación de la norma IEEE 754:


IEEE 754	Precisión simple	Precisión doble	Precisión cuádruple
nº de bits	32	64	128
signo	1	1	1
exponente	8	11	15
mantisa	23	52	112
exceso del exponente	127	1023	262143
valor más pequeño desnorm.	$2^{-126-23} = 2^{-149} \approx 1.5 \times 10^{-45}$	$2^{-1022-52} = 2^{-1074} \approx 5 \times 10^{-324}$	$2^{-16493} \approx 10^{-4965}$
valor más pequeño	$2^{-126} \approx 1 \times 10^{-38}$	$2^{-1022} \approx 2 \times 10^{-308}$	$2^{-16382} \approx 3 \times 10^{-4932}$
valor más grande	$(1 + (1 - 2^{-23})) \times 2^{127} \approx 3.4 \times 10^{38}$	$(1 + (1 - 2^{-52})) \times 2^{1023} \approx 2 \times 10^{308}$	$2^{16384} - 2^{16272} \approx 1 \times 10^{4932}$
precisión en dígitos ₁₀	23 bits, $\pm 2^{-150}$ 7	52 bits, $\pm 2^{-1075}$ 15	112 bits, $\pm 2^{-16494}$ 33

18-ene.-25

Curso 2024/2025

página 34

34



Datos de carácter

» Utilizados para representar un carácter dentro del juego de caracteres definido por el fabricante del ordenador

» Juegos de caracteres clásicos:

- **ASCII** (*American Standard Code for Information Interchange*)
 - › 7 bits (128 caracteres)
- **EBCDIC** (*Extended Binary Coded Decimal Interchange Code*)
 - › 8 bits (256 caracteres)

» **UNICODE** (www.unicode.org):

- Representa caracteres (*characters*) mediante números (*code points*) que se dibujan mediante ideogramas o glifos (*glyphs*)
- Puede representar hasta 2⁰⁹⁷¹⁵² (2²¹) elementos
- Todos los lenguajes del mundo están incluidos
- Requiere **codificación** para los *code points*
 - › La más usada y conocida es **UTF-8**

18-ene.-25

Curso 2024/2025

página 35

35




Tabla de códigos ASCII

» El código ASCII es un código de 7 bits que se suele almacenar en 8 bits.

- Obsérvese que una letra minúscula es el código ASCII de la mayúscula más 20 hexadecimal (32 decimal)

00 NUL	10 DLE	20 SP	30 0	40 @	50 P	60 `	70 p
01 SOH	11 DC1	21 !	31 1	41 A	51 Q	61 a	71 q
02 STX	12 DC2	22 "	32 2	42 B	52 R	62 b	72 r
03 ETX	13 DC3	23 #	33 3	43 C	53 S	63 c	73 s
04 EOT	14 DC4	24 \$	34 4	44 D	54 T	64 d	74 t
05 ENQ	15 NAK	25 %	35 5	45 E	55 U	65 e	75 u
06 ACK	16 SYN	26 &	36 6	46 F	56 V	66 f	76 v
07 BEL	17 ETB	27 '	37 7	47 G	57 W	67 g	77 w
08 BS	18 CAN	28 (38 8	48 H	58 X	68 h	78 x
09 HT	19 EM	29)	39 9	49 I	59 Y	69 i	79 y
0A LF	1A SUB	2A *	3A :	4A J	5A Z	6A j	7A z
0B VT	1B ESC	2B +	3B ;	4B K	5B [6B k	7B {
0C FF	1C FS	2C ,	3C <	4C L	5B \	6C l	7C
0D CR	1D GS	2D -	3D =	4D M	5D]	6D m	7D }
0E SO	1E RS	2E .	3E >	4E N	5E ^	6E n	7E ~
0F SI	1F US	2F /	3F ?	4F O	5F _	6F o	7F DEL

NUL Null

SOH Start of heading

STX Start of text

ETX End of text

EOT End of transmission

ENQ Enquiry

ACK Acknowledge

BEL Bell

BS Backspace

HT Horizontal tab

LF Line feed

VT Vertical tab

FF Form feed

CR Carriage return

SO Shift out

SI Shift in

DLE Data link escape

DC1 Device control 1

DC2 Device control 2

DC3 Device control 3

DC4 Device control 4

NAK Negative acknowledge

SYN Synchronous idle

ETB End of transmission block

CAN Cancel

EM End of medium

SUB Substitute

ESC Escape

FS File separator

GS Group separator

RS Record separator

US Unit separator

SP Space

DEL Delete

36




Tabla de códigos EBCDIC

» El código EBCDIC es un código de 8 bits

STX Start of text

DLE Data Link Escape

BS Backspace

ACK Acknowledge

SOH Start of Heading

ENQ Enquiry

ESC Escape

BYP Bypass

CAN Cancel

RES Restore

SI Shift In

SO Shift Out

DEL Delete

SUB Substitute

NL New Line

LF Line Feed

RS Reader Stop

PF Punch Off

DS Digit Select

PN Punch On

SM Set Mode

LC Lower Case

CC Cursor Control

CR Carriage Return

EM End of Medium

FF Form Feed

TM Tape Mark

UC Upper Case

FS Field Separator

HT Horizontal Tab

VT Vertical Tab

UC Upper Case

DC Data Control

DC Data Control

CU Control Unit

CU Control Unit

SY Synchronizing

ET End of Text

N/ Not Used

SC Signaling

IG Ignored

IR Interrupt


IU Invalid

00 NUL	20 DS	40 SP	60 —	80	A0	C0 {	E0 \
01 SOH	21 SOS	41	61 /	81 a	A1 ~	C1 A	E1 S
02 STX	22 FS	42	62	82 b	A2 s	C2 B	E2 T
03 ETX	23	43	63	83 c	A3 t	C3 C	E3 U
04 PF	24 BYP	44	64	84 d	A4 u	C4 D	E4 V
05 HT	25 LF	45	65	85 e	A5 v	C5 E	E5 W
06 LC	26 ETB	46	66	86 f	A6 w	C6 F	E6 X
07 DEL	27 ESC	47	67	87 g	A7 x	C7 G	E7 Y
08	28	48	68	88 h	A8 y	C8 H	E8 Z
09	29	49	69	89 i	A9 z	C9 I	E9
0A SMM	2A SM	4A ¢	6A *	8A	AA	CA	EA
0B VT	2B CU2	4B	6B ,	8B	AB	CB	EB
0C FF	2C	4C <	6C %	8C	AC	CC	EC
0D CR	2D ENQ	4D (6D	8D	AD	CD	ED
0E SO	2E ACK	4E +	6E ∇	8E	AE	CE	EE
0F SI	2F BEL	4F	6F '2	8F	AF	CF	EF
10 DLE	30	50 &	70	90	B0	D0 }	F0 0
11 DC1	31	51	71	91 j	B1	D1 J	F1 1
12 DC2	32 SYN	52	72	92 k	B2	D2 K	F2 2
13 TM	33	53	73	93 l	B3	D3 L	F3 3
14 RES	34 PN	54	74	94 m	B4	D4 M	F4 4
15 NL	35 RS	55	75	95 n	B5	D5 N	F5 5
16 BS	36 UC	56	76	96 o	B6	D6 O	F6 6
17 IL	37 EOT	57	77	97 p	B7	D7 P	F7 7
18 CAN	38	58	78	98 q	B8	D8 Q	F8 8
19 EM	39	59	79	99 r	B9	D9 R	F9 9
1A CC	3A	5A !	7A :	9A	BA	DA	FA
1B CU1	3B CU3	5B \$	7B #	9B	BB	DB	FB
1C IFS	3C DC4	5C ·	7C @	9C	BC	DC	FC
1D IGS	3D NAK	5D)	7D '	9D	BD	DD	FD
1E IRS	3E	5E ;	7E =	9E	BE	DE	FE
1F IUS	3F SUB	5F →	7F "	9F	BF	DF	FF

18-ene.-25

Curso 2024/2025

página 37



Unicode

Tipos de datos: datos derivados

- » Se emplean para referenciar otro dato (de otro tipo) mediante algún medio, por ejemplo usando la dirección de memoria del otro dato (**puntero**)
- » Características de los datos de tipo puntero:
 - **Tipo**: Mismo tipo que el del dato al que apunta.
 - **Contenido**: la dirección del dato al que apunta.
 - **Indirección**: referencias indirectas al dato al que apunta.
 - **Reutilización**: se puede utilizar para apuntar a otro dato del mismo tipo que el puntero.

18-ene.-25 Curso 2024/2025 página 39


39

Tipos de datos: datos estructurados

- » **Internos**: residen en la memoria principal del ordenador
- » **Externos**: residen en la memoria auxiliar del ordenador (dispositivos de almacenamiento masivo)
- » **Estáticos**: su tamaño no puede cambiar durante la ejecución del programa
- » **Dinámicos**: su tamaño puede cambiar durante la ejecución del programa
- » **Lineales**: sólo pueden estar enlazados a un elemento anterior y uno posterior
- » **No lineales**: pueden enlazarse a más de un elemento, de diferentes formas
- » **Compuestos**: formados por el programador mediante agregaciones de datos básicos y derivados

18-ene.-25 Curso 2024/2025 página 40

40



Tipos de datos: estructuras estáticas

TABLAS

» Estructura estática de datos constituida por un número fijo de **elementos** del mismo tipo en direcciones de memoria contiguas

» Clasificación:

- Unidimensionales (**vectores**)
- Bidimensionales (**matrices**)
- Multidimensionales (**poliedros**)

d

d+1

d+2

d+3

d+4

d+5

d+6

LUNES

MARTES

MIÉRCOLES

JUEVES

VIERNES

SABADO

DOMINGO

VECTOR

Alumno 1

Alumno i

Alumno n

Asignatura 1

Asignatura 2

Asignatura 3

Asignatura 4


5	8	0	4	0	0	0	0
0	4	0	0	0	0	0	0
2	7	9	6	8	8	1	3
0	0	0	0	0	0	0	0

MATRIZ

18-ene.-25

Curso 2024/2025

página 41



Tipos de datos: estructuras dinámicas (I)

DATOS ESTRUCTURADOS	INTERNOS	DINÁMICOS	LINEALES	LISTA/PILA/COLA CONJUNTO DICCIONARIO
			NO LINEALES	ÁRBOL GRAFO

» **LISTA**: estructura dinámica de datos constituida por **elementos** (habitualmente, pero no necesariamente, del mismo tipo) denominados **nodos**. El orden de los elementos es **significativo**.

- **PILA**: lista basada en el criterio **LIFO** (*last in first out*, el último que entra es el primero que sale). Los elementos se añaden apilándose unos sobre otros y sólo se tiene acceso al que está en la parte superior.
- **COLA**: lista basada en el criterio **FIFO** (*first in first out*, el primero que entra es el primero que sale). Los nuevos elementos se añaden en la parte de atrás y la extracción de elementos se realiza por la de delante.

» **CONJUNTO**: estructura dinámica constituida por **elementos** (habitualmente, pero no necesariamente, del mismo tipo). Los **elementos no están ordenados** y sólo es significativa su pertenencia o no al conjunto y la posibilidad de que estén repetidos o no.

» **DICCIONARIO** o **MAPA**: estructura dinámica de datos formada por una serie de **parejas** (**clave** → **valor**). El orden de las parejas puede ser **significativo** o no, así como la repetición de claves.

18-ene.-25

Curso 2024/2025

página 42

Tipos de datos: estructuras dinámicas (II)

LISTAS

» Tipos de listas:

- Contiguas
- Simplemente enlazadas
- Doblemente enlazadas
- Circulares

» Operaciones:

- Inicializar lista
- Insertar, eliminar o buscar un nodo
- Copiar, borrar la lista completa
- Subdividir o concatenar listas

CAMPO INFORMACIÓN CAMPO PUNTERO

18-ene.-25 Curso 2024/2025 página 43

43

Tipos de datos: estructuras dinámicas (III)

PILAS

» Utilizado en compiladores e intérpretes

» Operaciones:

- Inicializar pila
- Apilar (*push*) o desapilar (*pop*) un elemento
- Pila llena
- Pila vacía
- Top

18-ene.-25 Curso 2024/2025 página 44

44

Tipos de datos: estructuras dinámicas (IV)

COLAS

» Elementos interiores inaccesibles

» Operaciones:

- Inicializar cola
- Encolar (*enqueue*)
- Desencolar (*dequeue*)
- Cola llena
- Cola vacía

18-ene.-25 Curso 2024/2025 página 45

45

Tipos de datos: estructuras dinámicas (V)

CONJUNTOS

» Elementos no ordenados dentro del conjunto

» Operaciones:

- Inicializar conjunto
- Insertar (*insert*) o eliminar (*delete*) un elemento
- Pertenencia de un elemento al conjunto
- Número de repeticiones de un elemento
- Conjunto vacío
- Cardinalidad

18-ene.-25 Curso 2024/2025 página 46

46

Tipos de datos: estructuras dinámicas (VI)

DICCIONARIOS o MAPAS

» Asocian **valores** a **claves**

» Operaciones:

- Inicializar mapa
- Insertar pareja (clave, valor)
- Eliminar pareja (clave, valor)
- Lista de claves
- Lista de valores
- Valor de una clave
- Pertenencia de una clave

DICCIONARIO o MAPA

CLAVES	VALORES
C1	V1
C2	V1
C3	V2
C4	V3

18-ene.-25 Curso 2024/2025 página 47

47

Tipos de datos: estructuras dinámicas (VII)

ÁRBOL – representación de estructuras jerarquizadas

• Características:

- › Un antecesor
- › Varios sucesores
- › Un único elemento llamado **raíz** (sin antecesor)

GRAFO – conjunto de **nodos** conectados por **arcos**


• Características:

- › Los arcos pueden tener sentido (dirigido) o no
- › Puede presentar ciclos o ser acíclico

ÁRBOL GRAFO

18-ene.-25 Curso 2024/2025 página 48

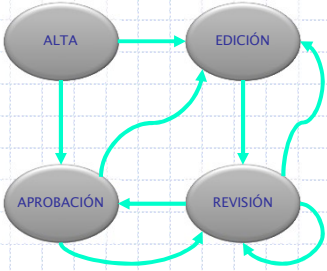
48



Ejemplo de grafos

- » Supongamos que estamos diseñando un programa de control de versión de documentos. Durante el análisis se recoge la siguiente información:
- » Es posible dar de alta un documento.
- » Una vez dado de alta se puede pasar a editar el documento. Si ya existe el documento se puede pasar directamente a aprobación.
- » Una vez editado, se puede pasar a revisión.
- » De revisión puede ser enviado de nuevo a edición, o dejar pendiente de revisión de nuevo. Otra posibilidad es que se proceda a la aprobación del documento.
- » Una vez que está aprobado, el paso del tiempo puede dar lugar a inconsistencias en el contenido del mismo por lo que es posible que se envíe a revisar o directamente a edición.

	ALTA	EDICIÓN	REVISIÓN	APROBACIÓN
ALTA		X		X
EDICIÓN			X	
REVISIÓN		X	X	X
APROBACIÓN		X	X	



18-ene.-25


Curso 2024/2025

página 49

estatico: se puede pero difícil o costoso

vector es un array ligeramente dinamico

Cadena/String: suele ser un array de caracteres y suelen ser constantes. En algunos no son constantes y usan vectores.



Propiedades de los tipos de datos

- » Propiedades de los tipos de datos compuestos:

Propiedad	Tabla	Vector	Lista (pila / cola)	Conjunto	Diccionario	Árbol	Grafo
tamaño	fijo	estático	dinámico	dinámico	dinámico	dinámico	dinámico
orden	si	si	si	no	no	complejo	complejo
homogéneo	si	si	no (si)	no (si)	¿claves?	no	no
acceso secuencial (recorrido)	no	si	variado (no)	si	claves, valores y parejas	múltiple	complejo
acceso aleatorio	si	si	complejo	si	si	no	no
inserción	no	complejo	si (fin/fin)	si	si	caro	si
borrado	no	complejo	si (ini/fin)	si	si	caro	si
sustitución	si	si	si	no	si	si	si
búsqueda	no	si	lineal	pertene- ncia	clave → valor	rápida	compleja
concatenar	no	compleja	si	si	si	complejo	complejo
operación básica	acceso aleatorio	acceso aleatorio	recorrido	pertene- ncia	búsqueda	recorrido	camino


- » Cadena: normalmente tabla o vector de caracteres

18-ene.-25

Curso 2024/2025

página 50

50



Sistema de Tipado


Type systems

La relacion entre tipos y sistema de tipados es como un arbol y el bosque

Que cosas podemos hacer entre diferentes tipos en un lenguaje

Como se pueden relacionar

51



Introduction to type systems

- » Assigning datatypes ("typing") gives **meaning** to collections of bits
- » A **type system** stipulates the ways typed programs may **behave** and makes behaviour outside these rules illegal

In computer science, a type system defines how a programming language classifies values and variables into types, how it can manipulate those types and how they interact.

» From Wikipedia, the free encyclopedia

18-ene.-25 Curso 2024/2025 página 52

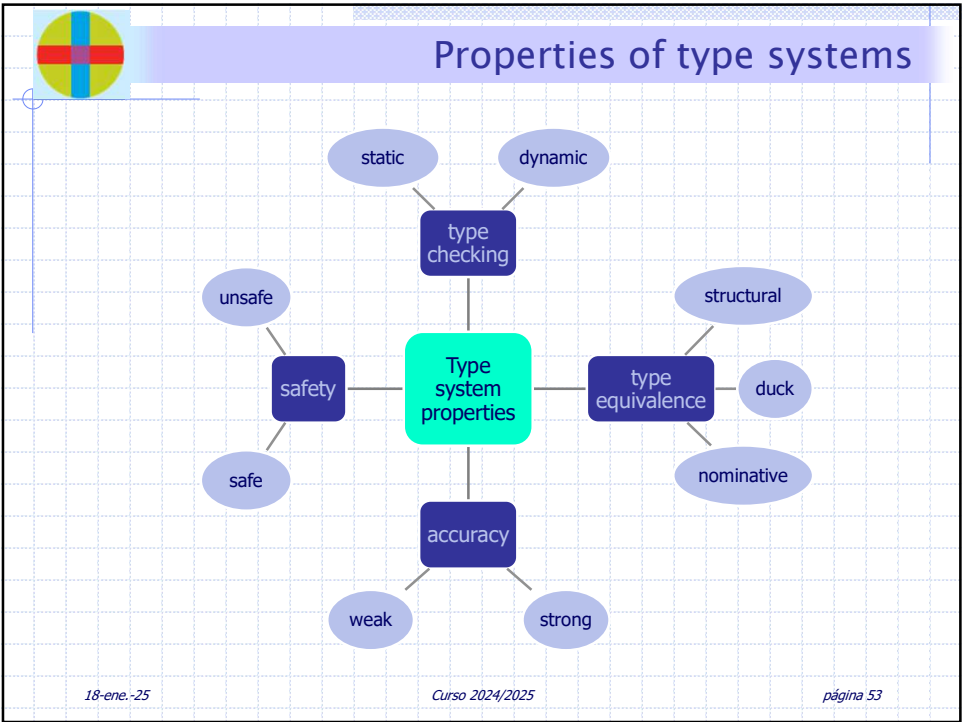
Saber que una cadena de bits significa o representa algo.

Que comportamientos entre tipos se considera legales o ilegales.

52

Un lenguaje tiene un sistema de tipado en respecto a 4 propiedades normalmente

Type Checking: estatico comprueba tipos en tiempo de compilacion y dinamico en tiempo de ejecucion



Ningun lenguaje es 100% de algo, es mas bien que se aproxima

Como el +inf -inf se aproxima

Java tienen tipado estatico asi que obliga a hacer una declaracion

```
int x = 0;
```

En Python no:
x = 0

18-ene.-25

Curso 2024/2025

página 54

» Every name is bound both to a type (at **compile time**, via a **data declaration**) to an object

- The binding to an object is **optional** — if a name is not bound to an object, the name is said to be *null*
- Once a variable name has been bound to a type (declared) it can be bound (via an **assignment statement**) only to objects of that type; it cannot ever be bound to an object of a different type. An attempt to bind the name to an object of the wrong type will raise a **type exception**
- All checks are done in **compiling time**
- Can only check if the data processing is right but not the data itself

Una vez engancha a un tipo, no cambia ni deja engancharlo a un tipo diferente.

Solo mira si el procesamiento del dato es correcto pero no si el dato lo es.

```
int a = 3;  
int b = 0;  
int c = a/b;
```

NA


a = 3
a = 'hola'

Los identificadores no tienen tipo, se enganchan a lo que le asignes pero se le puede cambiar.

Los problemas se detectan en tiempo de ejecucion, no en compilacion

a = 'hola'
b = 3
c = a/b

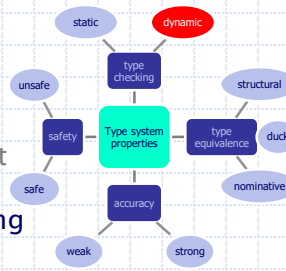
MPL



Dynamic type checking

» Every name is (unless it is *null*) bound only to an object

- Names are bound to the objects at **execution time** by means of **assignment statements**, and it is possible to bind a name to objects of **different types** during the execution of the program
- Type **errors** not detected until a piece of code is **executed**
- Allows **more kinds of code constructs** (expression evaluation, introspection)
- Shortens the **edit-compile-test-debug** cycle (**REPL**: Read-Eval-Print Loop)
- Makes *metaprogramming* more powerful and easier to use



The diagram shows 'Type system properties' at the center. It is connected to 'static' and 'dynamic' (highlighted in red) under 'type checking'. It is also connected to 'unsafe' and 'safe' under 'safety', and 'weak' and 'strong' under 'accuracy'. To the right, it connects to 'structural' and 'type equivalence' (which further connects to 'duck' and 'nominative').

18-ene.-25

Curso 2024/2025

página 55


metaprogramar, un programa que escribe programas, se puede modificar en tiempo de ejecucion

55

Si hay 'match' no hay problema de los tipos

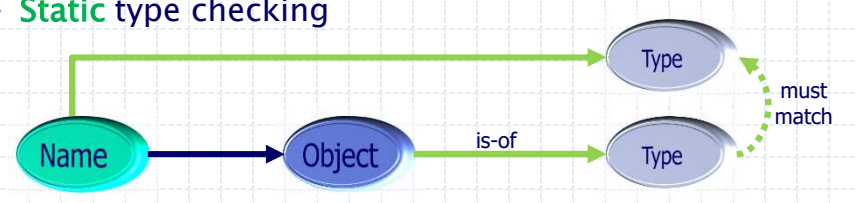
El tipo se comprueba el la interaccion con otros objetos

a = 'hola'
b = 3
c = a/b




Static *versus* Dynamic type checking

» Static type checking



The diagram shows a 'Name' node pointing to an 'Object' node. The 'Object' node points to a 'Type' node via an 'is-of' relationship. Another 'Type' node is shown above, with a dashed arrow pointing to the 'Object' node and a label 'must match'.

» Dynamic type checking



The diagram shows a 'Name' node pointing to an 'Object' node. The 'Object' node points to a 'Type' node via an 'is-of' relationship.

18-ene.-25

Curso 2024/2025

página 56


Equivalencia es cuando se hace el match de los tipos

56

Genericos: tipo plantillas
ArrayList<float> = ...

Se escribe menos.
Pero los identificadores al
no tener tipo pueden
cambiar de tipo, que
puede ser liante para
muchos

Mas redundante al
hacerte hacer declaracion
de datos, pero obliga a
escribir mas.



Static *versus* Dynamic type checking

» Benefits of **static** type checking

- Errors / Warnings in your editor (now also on dynamic)
- Generics (but this can be also a negative)
- Ability to follow a chain and figure out what type of object is required at each step
- Refactoring
- Common in compiled languages, considered “safer”

» Benefits of **dynamic** type checking

- More concise and precise, less typing
- No badly implemented generics
- Closures / Functions
- REPL: Read-Eval-Print Loop
- Common in interpreted languages

18-ene.-25

Curso 2024/2025

página 57

57

No permite que se haga
operaciones con
diferentes tipados.

No permite que se
haga operaciones con
diferentes tipados.


Cuanto mas debil es mas
tienes que tener en la
cabeza, pero mas facil y
comodo.

Los lenguajes de tipo
debil son mas peligrosos

C si hace conversion
implicita en ambos la
suma y lo que es 'y'

Java no te deja porque son
de dos tipos deferentes.
Pero no porque no deje
sumarlos, sino porque y es
un entero.

Java convierte el entero
en real, el resultado es un
float 29.6, y como 'y' es
entero da error



Strong / Weak typing

» A **strongly typed** language doesn't allow an operation to succeed on arguments which have the wrong type

» A **weakly typed** language does

```
int x = 5, y;  
float c = 24.6;  
  
y = x + c;
```

Example

Tipado Debil

```
y = 29
```

Result in C

```
java.lang.Error:  
Unresolved compilation problems:  
Type mismatch:  
cannot convert from float to int
```

Result in Java

Tipado Fuerte

18-ene.-25

Curso 2024/2025

página 58

58

Java si es entero
pero no se le
bindea valor, le da
0.

Un lenguaje fuertemente tipado
no hace conversiones sin que
se lo pidas (implicitamente)

Los lenguajes con tipado
dinamico como no se declara el
tipo, son de tipado fuerte (como
python)

Conversion Implicita => Lo hace el lenguaje
Conversion Explicita => Lo haces tu

No unicamente, pero principalmente. Ensamblador tampoco es que sea inseguro porque su uso es para estar muy cercano a la maquina, trabajando directamente por bits.

No te va a hacer conversiones que puedan joder y que sean peligrosos

Java realmente no hace cast, sino que conversiones

Un cast (molde) coge patron de bits y lo reinterpreta con otro tipo.

El la conversion se modifican los bits de la mem, en cast no

Safe / Unsafe typing

staticdynamic

unsafe

safe

type checking

Type system properties

accuracy

weakstrong

structural

type equivalence

duck

nominative

» A language is **type-safe** if it does not allow operations or conversions which lead to erroneous conditions

» A language is **type-unsafe** if allows these conversions (type coercion)

var x = 5;
var y = "hi";
var z = x + y;

Example in Visual Basic

z = "5hi";

Result in Visual Basic

int x = 5;
char y[] = "hi";
char* z = x + y;

Example in C

z = " 1 00 00E-14 "

Result in C

18-ene.-25

tipado debil seguro

Curso 2024/2025

inseguro

página 59

C tiene como ultimo caracter como 0, asi que son tres posiciones en mem, no dos

Puntero + entero, desplazo del puntero. Te vas a una zona de mem que no esta definida

De aqui vienen muchas vulnerabilidades de los programas de C.

59

Si reinerpreta los bits de un 36 como si fuera un string, entonces lo interpreta como ASCII y sale el codigo 36 => \$

Un cast nunca te dara error porque coge el chorro de bits y los interpreta como funcione el otro tipo

Genericos hacen hacer ""casts"" (realmente conversiones) que hace perder informacion

int != integer
Dos tipos son iguales si se llaman igual

Dos tipos son iguales si sus estructuras son iguales
dos de 32bits C-2, aun con nombre diferente se considera igual

En haskell los tipos los calcula por inferencia, las declaraciones no son obligatorias (a veces hasta las ignora lol)

Cuando dos tipos pueden operar juntos

Type equivalence

staticdynamic

unsafe

safe

type checking

Type system properties

accuracy

weakstrong

structural

type equivalence

duck

nominative

» A type system has always an own definition of **equivalence**

» Two extreme cases are

- **Nominative** type systems
 - > (e.g. Java, C, Fortran)
 - > Types must have the same **name** in order to be equal
- **Structural** type systems
 - > (e.g. Haskell, Lisp)
 - > Types must have the same **structure** in order to be equal

» A special form of dynamic typing is "**duck typing**"

- The language "guesses" which type is meant (e.g. Perl, Python, Ruby, Smalltalk)

18-ene.-25

Curso 2024/2025

página 60

En OOP tiran mas del arbol genealogico, porque depende de quien heredes (persona: alumno, profesor)

Crean unas jerarquias de clases muy complejas y jodidas.

No mira si la herencia ni estructura, mira lo que puede hacer el objeto. Lo que mira para que operen entre objetos es lo que tiene esa clase.

60

La mayor parte de lenguajes son nominativos, pero los modernos tiran a ser mas 'duck type'.

JavaScript convierte de todo implicitamente porque se hizo para buscadores y que no se rallara mucho

primer ensamblador concatenativo (super ligero) se uso en astronomia

Programming languages & type systems				
Language	type checking	accuracy	safety	equivalence
Ada	static	strong	safe	nominative
assembler	none	weak	unsafe	structural
BASIC	static	weak	safe	nominative
C	static	weak	unsafe	nominative
C++	static	strong	unsafe	nominative
C#	static	strong	both	nominative
Eiffel	static	strong	safe	nominative
Erlang	dynamic	strong	safe	nominative
FORTH	none	weak	unsafe	structural
Haskell	static	strong	safe	nominative
Java	static	strong	safe	nominative
Javascript	dynamic	weak	safe	duck
Lisp	dynamic	strong	safe	structural
ML	static	strong	safe	structural
Objective-C	dynamic	strong	unsafe	duck
Pascal	static	strong	safe	nominative
Perl	dynamic	weak	safe	nominative
PHP	dynamic	weak	safe	nominative
Python	dynamic	strong	safe	duck
Ruby	dynamic	strong	safe	duck
Scheme	dynamic	strong	safe	nominative
Smalltalk	dynamic	strong	safe	duck

Source: http://wikipedia.org/wiki/Template:Type_system_cross_reference_list

18-ene.-25

Curso 2024/2025

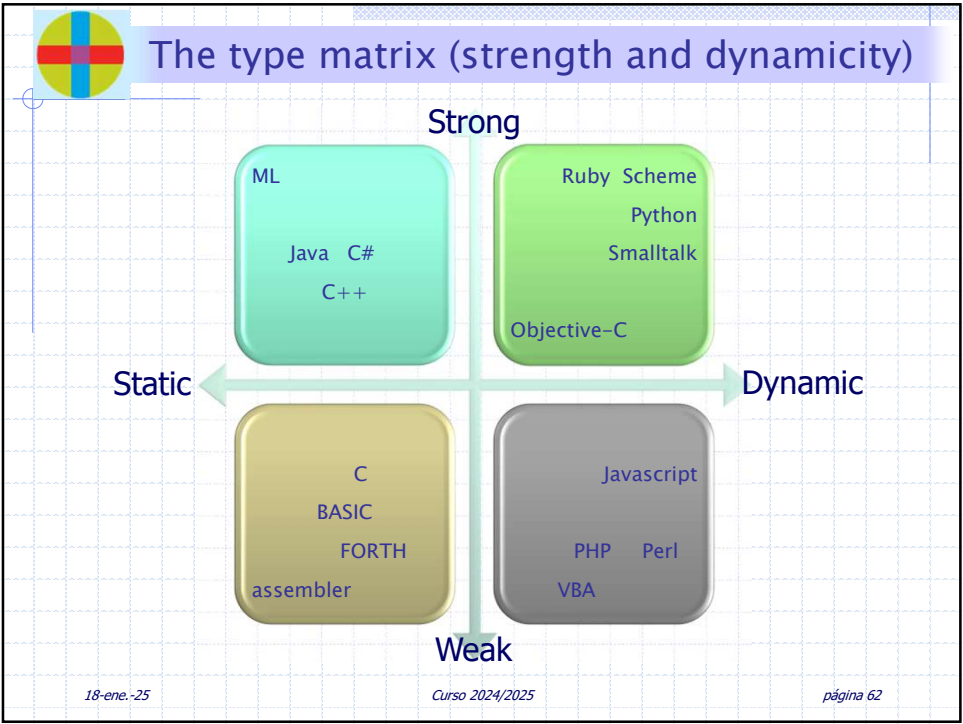
página 61

Java compila el .class que hace un codigo intermedio, y una VM (JVM) lo interpreta

Las apps de los moviles (android) van sobre una JVM

Python tambien usa una JVM


Historicamente los lenguajes bien vistos eran los estaticos, actualmente los que se ven bien son los fuertes



El peligro de las implicitas es que dependes de como se decidio el lenguaje

algo como:
>"1" + "50"
>"51"
imagina que hace eso lol

PHP y Pearl si no sabe lo que hacer lo convierte todo a strings




References

- » Paul Hudak, “*Conception, Evolution, and Application of Functional Programming Languages*”, ACM Computing Surveys 21/3, Sept. 1989, pp 359–411
- » L. Cardelli and P. Wegner, “*On Understanding Types, Data Abstraction, and Polymorphism*”, ACM Computing Surveys, 17/4, Dec. 1985, pp. 471–522
- » D. Watt, “*Programming Language Concepts and Paradigms*”, Prentice Hall, 1990


18-ene.-25 Curso 2024/2025 página 63

63



Object-Oriented Programming (OOP)

64



Imperative programming vs. OOP

- » So, what are the paradigms of programming?
- » Imperative programming is the “traditional” model of computation.
 - State
 - Variables
 - Assignment
 - Loops
- » A processing unit is separate from memory, and “acts” upon memory
- » What is the paradigm known as **Object–Oriented Programming (OOP)**?

18-ene.-25

Curso 2024/2025

página 65

65

La OOP es tener muchos objetos colaborando. Y de usar lo que ya esta hecho.

Si una vez hecha la interfaz de metodos , si hay que arreglar algo, entonces ya nos metemos en el cacho de codigo, pero la interfaz y llamada sera la misma

Hay lenguajes como JavaScript (1.5) que tiene objetos sin clases. Hace copias de otros objetos, y despues cambiarlos.


No sale de una clase principal, salen de copiar y alterar objetos. JS ya tiene objetos base de los que copiar.

Modelo: reducido a la informacion que interesa.

Cuando se hereda te comes las carateristicas que a lo mejor no te interesan

Los diferentes objetos se distinguen por sus diuferentes valores de sus atributos.

Los metodos (lo que saben hacer)



Basic concepts

- » A **Class**:
 - Is a description used to instantiate objects
 - Identifies properties (attributes) that belong to all objects of the class and behaviors (methods)
- » An **Object**:
 - Is an instance of a class, it has a name, attributes and their values, and methods
 - Models an idea found in reality (tangible or abstract)
- » **Attributes**: properties of the object
- » **Methods (Services, Messages)**: behaviors
- » **Information hiding and Encapsulation**:
 - A technique in which an object reveals as little as possible about its inner workings
- » Other concepts: no tan importante
 - **Inheritance, Polymorphism, Overloading, Templates**

18-ene.-25

Curso 2024/2025

página 66

Como el DNA de un objeto

Diferencia:
Los objetos estan vivos (tienen ciclo de vida) las clases no. Los que funcionan son los objetos no las clases.

Las clases no son lo importante

Un metodo es algo que sabe hacer un objeto

Se le invoca con un mensaje, luego da la respuesta

La gracia de OPP es tener muchos objetos muy sencillos, lo mejor es que un objeto solo tenga una funcion en su vida.

Siendo muy pequeños la programacion imperativa se hace sencilla

Information Hiding: lo que tienes que saber de un objeto es que puede hacer, no el como. La gracia es que lo que tienes que aprender es como usarlo. Aunque lo hayas implemetado tu, ya no tienes que editarlo

de aui viene el Principio de la Delegacion

Encapsulacion: se le llama a la interfaz de los metodos (en java con private,...). El objeto tiene la capacidad de responder lo que quiere

66

C lo hace todo por valor las cosas de 'verdad' cuando le pasas un array, copia la dir de mem del array

En OOP es el propio objeto el que tiene el poder de hacer copias de el mismo

Lo que todo el mundo llama variables suelen ser referencias.

en Java cuando llega a la {}, la referencia se borra, el objeto no.

Los objeyos se manipulan por referencias, el objeto sigue ahí. Cuando ya desaparecen todos los nombres/referencias se queda inalcanzable y se borra (el recolector de basera)

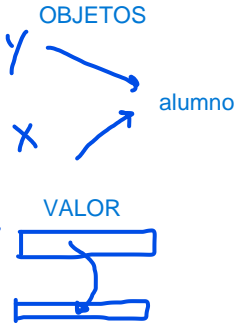
Sabes cuando se crean, pero no cuando se destruyen, porque lo hace el recolector de basura cuando sea.

En Java el 'new' es el que crea un objeto.
En Python con poner el n de la clase vale

Porque los metodos son siempre parte de un objeto, siempre tienen el parametro 'self'.

Por valor:
int a;
int b;

a= 3;
b=a;



No se crean copias, son dos nombres para el mismo elemento/objeto.

Si se quiere hacer una copia de verdad:
`b = a.copy()`
invoca un metodo sobre el objeto (crea nueva lista con la copia del anterior)

Comprobar identidad (python):
`a is b;`

Comprobar igualdad:
`a==b;`

metodo '___eq___' es equivalente a ==. (Overloading)

Java no tiene overloading, nunca se podran sumar directamente objetos

En Java el '==' (igualdad) es el .equals, la identidad 'is' es el ==.



Assignment, equality and identity

» **Assignments** create/manipulate object references

- `x = y` **does not make a copy** of `y`
- `x = y` makes `x` **reference** the object `y` references

» **Very useful; but beware!**

```
>>> a = [1, 2, 3] a es una referencia a la lista
>>> b = a ahora la lista tiene 2 nombres
>>> a.append(4)
>>> print(b)
[1, 2, 3, 4]
```

apunta a misma dir de mem

» A test for **identity** asks whether two references refer to **exactly the same object**.

dos referencias apuntan al mismo objeto

» A test for **equality** asks whether two references refer to **values that are equivalent**.

dos objetos tienen mismo valor

- The meaning of **equivalent** is **inherently domain specific**.
pero depende de quien haya disenado los objetos

18-ene.-25

Curso 2024/2025

página 67

67

C no tiene realmente recolector de basura, así que hay que tener en cuenta mas cosas

Para objetos en Python, si no le pasa criterio para comparar la igualdad, usa el siguiente criterio, el de identidad (dir de mem)

Los problemas grandes son igual a los problemas pequeños. Aparecen las mismas cosas en menores escalas.

Todos los problemas se pueden reducir a objetos.

Se le delega el trabajo a los objetos.

Se le dan diferentes trabajos a diferentes objetos.

Las computaciones se hacen cuando se envían mensajes. Una vez aceptan el mensaje, se dispara la acción (método), el cual suele tener el nombre del mensaje (ej: `QueHoraEs()`).

Importante para programación en red porque no tienen que estar los objetos en la misma máquina para ser invocados.



Kay's description of OOP

» Object-oriented programming is based on the principle of **recursive design**

- **Everything is an object**
 - › Actions are performed by agents, called instances or **objects**
- Objects perform computation by making **requests** of each other through the passing of **messages**
 - › Actions are produced in response to requests (messages)
 - › An instance may accept a message, and in return will perform an action (**method**) which use its data (**attributes**) and return a value (usually another object)
- Every object has its own memory, which consists of other objects (**encapsulation**)
 - › Each object is like a *miniature computer itself* – a specialized processor performing a specific task
- Every **object** is an **instance** of a **class**, so a class:
 - › Groups similar objects
 - › Is a **repository for behavior** associated with an object

No tiene que saber como un objeto hace su trabajo internamente (importante porque no siempre está en la misma máquina)

Un objeto es como un mini-ordenador que solo sabe hacer unas pocas cosas. Se suelen hacer de manera imperativa (más fácil cuando es poco).

18-ene.-25

Curso 2024/2025

página 68

68


Basado en prototipos (en vez de clases) es enseñar a un objeto hacer cosas y se hacen copias a ese objeto.

Para que aprenda un objeto, se le edita la clase.

Alumno es una Persona especializada. Hace lo que hace una Persona y mas.

Capaz de trabajar con diferentes objetos para hacer lo mismo.

como el __eq__ de antes



Elements of OOP

» **Inheritance** Mecanismo para escribir menos

- Ability to define classes that are extensions of other classes with new and/or specialized attributes and methods

» **Overriding**

- Subclasses can alter or override information inherited from parent classes

» **Polymorphism**

- Ability to use same name for methods
- Allows to manipulate objects without knowing their exact type but only their common property

» **Overloading**

- An existing operator, such as + or = is given the capability to operate on a new data type

18-ene.-25

Curso 2024/2025

página 69

Se puede liar mucho

4 + 7 => 11
3.3 + 7 => que hace?


69

En C++ se activa con 'virtual'

La ejecucion se hace una vez el objeto recibe el mensaje. Puede alterar el comportamiento en tiempo de ejecucion. Si fuera static binding, siempre daria el mismo resultado cuando se esta ejecutando.

LOS OBJETOS TIENEN CICLO DE VIDA

C++ tiene ambos dynamic y static binding y hay que activarlo si se quiere usar lol



Elements of OOP (II)

» **Dynamic binding** La mas importante, lo que lo hace OOP

- Process of identifying at *run time* what code should be executed as a result of a message
- The ability of invoke a derived class method via a base class reference is called *virtual method invocation* or *dynamic binding*

» This is the **one true characteristic of object-oriented languages**

- A language with classes, attributes, methods, inheritance, polymorphism, overloading, templates, etc. is said to be *object-based*, but **not** *object-oriented*.

18-ene.-25

Curso 2024/2025

página 70

Una funcion no es un metodo porque el 'self' no aparece por ningun lado, pero a un objeto se le puede asignar un atributo que apunte a una funcion. (pero eso no lo convierte en metodo):

a = A()
a.f = funcion

Pero cuando se le edita en la clase, si que se convierte en un metodo:

A.c = funcion

Para un solo objeto, no todos:

a.g = function.__get__(a)
get obtiene un metodo.

Todo esto se puede cuando hay dynamic binding, si no lo tuviera, entonces no se podria hacerle cambios mientras se esta exec.


70

Lo que hacen los plugins, los objetos aprenden durante la marcha (o olvidan si se borra)

En vez de usar herencia para todas las opciones que pueda tener un objeto, con dynamic binding un objeto puede crecer/aprender nuevos atributos/metodos.

En static tienes que tener el catalogo de todas las opciones antes de la exec.

Pasa mucho que ya que un objeto hace una cosa, puede hacer otras ya que estamos. Esta MAL



OOP Design – **SOLID**

- » **Single Responsibility Principle (SRP)**
 - A class should have only one reason to change
- » **Open-Closed Principle (OCP)**
 - Software entities (classes, modules, functions, etc.) should be open for extension but closed for modification
- » **Liskov Substitution Principle (LSP)**
 - Subtypes must be substitutable for their base types
 - > Basic OOP discusses inheritance with "IS-A"
 - > LSP says that "IS-A" refers to behavior
 - > Behavior is what software is really all about
- » **Interface Segregation Principle (ISP)**
 - Clients should not be forced to depend on methods they do not use
- » **Dependency Inversion Principle (DIP)**
 - High-level modules should not depend on low-level modules. Both should depend on abstractions
 - Abstractions should not depend on details. Details should depend upon abstractions

18-ene.-25 Curso 2024/2025 página 71

Un objeto solo debería dedicarse a una sola cosa

Si hay un objeto con mucho 'cataogo', para hablar con él, no tienens que conocer todo su catalogo

o principio Hollywood (no llames tu, ya llamamos nosotros). Partes de abajo son los que llaman a partes de arriba, no al revés.

No se debe de llamar a las 'partes sucias' de los objetos, se debería de llamar a sus interfaces abstractas

Open for extension (ej ya que da la hora, que la de en otras zonas horarias) pero closed for modification (si tiene contratos con otros objetos, no se puede cambiar)


71

Llamadas a metodos abstractos, cada uno ya lo implementan. Ahi es donde se dan sus especificos detalles

Cuidado con los templates

Si no hay herencia no hay estos problemas

Los contenedores con template pueden ser un lio



Covariance and contravariance


- » **Formal definition:** within the type system of a programming language, a typing rule or a type constructor is:
 - **covariant** if it preserves the ordering of types (\leq), which orders types from more specific to more generic
 - **contravariant** if it reverses this ordering
 - **bivariant** if both of these apply (i.e., both $T\langle A \rangle \leq T\langle B \rangle$ and $T\langle B \rangle \leq T\langle A \rangle$ at the same time)
 - **invariant** or **nonvariant** if neither of these applies
- » **Example: arrays**
 - First consider the array type constructor: from the type `Animal` we can make the type `Animal[]` ("array of animals"). Should we treat this as
 - > **Covariant:** a `Cat[]` is an `Animal[]`
 - > **Contravariant:** an `Animal[]` is a `Cat[]`
 - > or neither (**invariant**)?

18-ene.-25 Curso 2024/2025 página 72

72

Si son de solo lectura si que son covariantes. Solo puedes sacar/leer. Como fuente son covariantes.

A la entrada son invariantes.



Covariance and contravariance

» **Answer:**

- If we wish to avoid type errors, and the array supports both reading and writing elements, then only the third choice is safe. Clearly, not every `Animal[]` can be treated as if it were a `Cat[]`, since a client reading from the array will expect a `Cat`, but an `Animal[]` may contain e.g. a `Dog`. So the contravariant rule is not safe.
- Conversely, a `Cat[]` can not be treated as an `Animal[]`. It should always be possible to put a `Dog` into an `Animal[]`. With covariant arrays this can not be guaranteed to be safe, since the backing store might actually be an array of cats. So the covariant rule is also not safe—the array constructor should be invariant. This is only an issue for mutable arrays; the covariant rule is safe for immutable (read-only) arrays.

» This illustrates a **general phenomenon**


- Read-only data types (**sources**) can be **covariant**
- Write-only data types (**sinks**) can be **contravariant**
- Mutable data types which act as **both sources and sinks** should be **invariant**

18-ene.-25 Curso 2024/2025 página 73

73

Con quien puedo hablar?

A quien conoce un objeto?
A si mismo, metodos de tus antecesoros.
Solamente hablas con los que conoces. A objetos que tengo declarados (como el circulo con Punto). Y objetos que vienen como parametros en los metodos.



Law of Demeter

» Also called **Don't Talk to Strangers**

- Each class should only use a limited set of other classes: only units “closely” related to the current unit
- “Each class should only talk (send messages) to its friends” or “Don’t talk to strangers”
- If objects traverse long object structure paths and send messages to distant, indirect (stranger) objects, the system is fragile with respect to changes in the object structures – a common point of instability in systems
- **LoD** helps us avoid creating such designs

18-ene.-25 Curso 2024/2025 página 74

74

