

TEMA 1

Introducción a la programación en Python

Prof. Alejandro Baldominos

¿Qué es la programación informática?

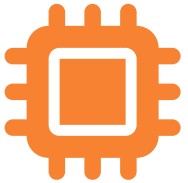
Conjunto de procesos que nos permiten construir programas de ordenador.

Idea fundamental:



Estructura básica de un ordenador

Un ordenador comúnmente posee los siguientes componentes:



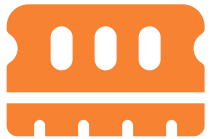
Procesador

Es el componente responsable de ejecutar instrucciones y realizar cálculos.



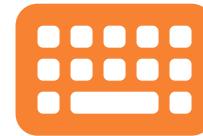
Dispositivos de almacenamiento

Más allá del disco duro, permiten almacenar datos en una ubicación externa al ordenador.



Memoria

Almacena las instrucciones y los datos de los programas en ejecución.



Dispositivos de entrada

Permiten comunicarse con el ordenador, por ejemplo, introduciendo datos.



Disco duro

Almacena los datos del ordenador (programas y documentos) de forma persistente.



Pantalla


Muestra de forma gráfica el estado del ordenador y los resultados de los programas ejecutados.

Estructura básica de un programa

El programa consta de los siguientes elementos fundamentales:



Instrucciones


Son los comandos que le indican al  procesador lo que debe hacer durante la ejecución del programa.

Algunos ejemplos de instrucciones son:

- Lee el valor de una posición de memoria.
- Suma dos números y almacena el resultado en memoria.
- Muestra el resultado de una operación por pantalla.
- Si un determinado dato es menor que cero, sal del programa.



Datos

Son valores e información que indican el estado del programa y permiten también almacenar resultados. Los datos pueden almacenarse en  memoria, si solo se van a emplear durante la ejecución del programa.

También pueden almacenarse en  el disco duro o  un dispositivo de almacenamiento, si deben persistir tras la finalización del programa.

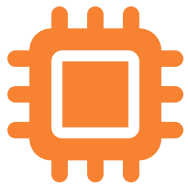
Además, los datos pueden tener varios orígenes:

- Puede definirlos el programador en el código del programa.
- Pueden ser introducidos por el usuario (ej.: mediante el teclado).
- Pueden ser obtenidos de fuentes externas (por ejemplo, Internet o un dispositivo de almacenamiento).
- Puede ser calculado durante una ejecución del programa.

Funcionamiento básico de un procesador

Instrucciones

El procesador tiene un conjunto de instrucciones.



Instrucción

```
add  r1 r2 r3
addi r1 r2 10
mul  r1 r2 r3
lw   r1 100(r2)
sw   r1 100(r2)
beq  r1 r2 10
slt  r1 r2 r3
j    1200
jr   r1
```

Descripción

Suma los valores de los registros r2 y r3 y almacena el resultado en el registro r1.
Suma 10 al valor del registro r2 y almacena el resultado en el registro r1.
Multiplica los valores de los registros r2 y r3 y almacena el resultado en r1.
Carga en el registro r1 el valor de la posición de memoria almacenada en r2+100.
Carga en la posición de memoria almacenada en r2+100 el valor del registro r1.
Si los valores de r1 y r2 coinciden, avanza el programa 10 instrucciones.
Si r2 < r3, almacena el valor 1 en r1, en caso contrario, almacena el valor 0.
Mueve el puntero actual a la posición de memoria 1200.
Mueve el puntero actual a la posición de memoria almacenada en el registro r1.

Un registro es un pequeño espacio de almacenamiento del procesador (normalmente capaz de almacenar un número o una dirección de memoria)

Funcionamiento básico de un procesador

Código binario

El procesador funciona empleando código binario (0 / 1).

Estos valores se representan mediante señales electrónicas de un determinado voltaje.

Cada instrucción puede representarse por un determinado código binario.

Ejemplo

	SPECIAL	r2	r3	r1	0	ADD
add r1 r2 r3	000000	00010	00011	00001	00000	100000

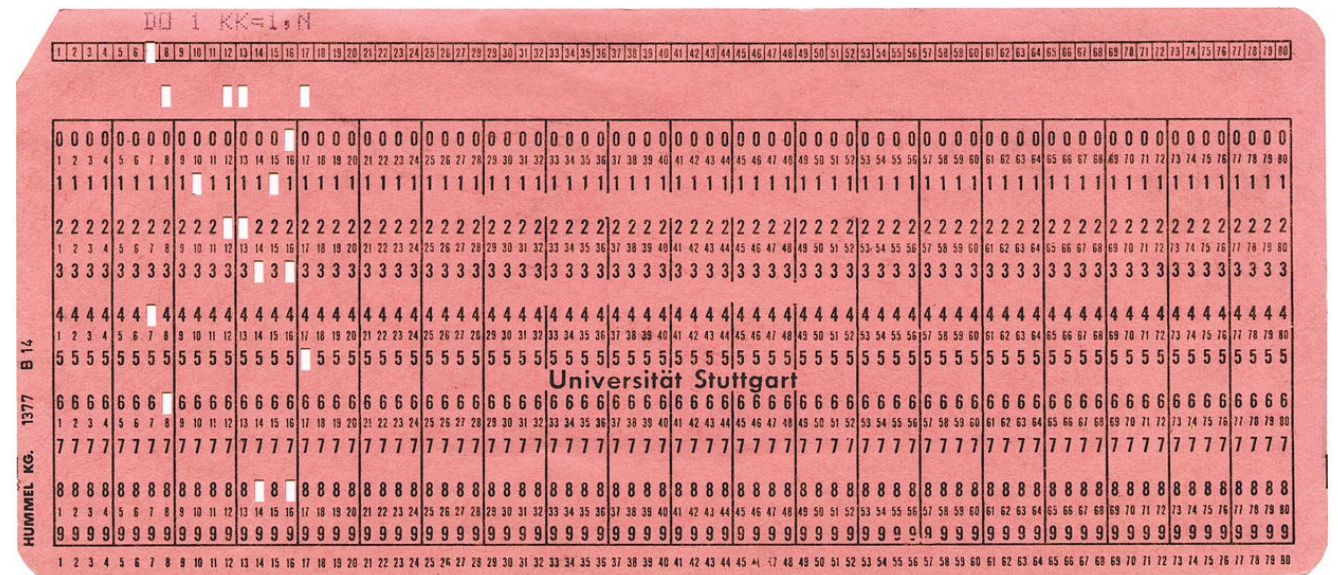
Fundamentos de la programación

Código binario

¿Para programar, tenemos que escribir código binario?

¡No!

(al menos, desde hace medio siglo)



Tarjeta perforada empleada a principios de los años 70 en la Universidad de Stuttgart (Alemania), una forma de representar instrucciones y datos empleando el código binario. Imagen reproducida bajo licencia Creative Commons, publicada originalmente por Harke para Wikimedia Commons.

Fundamentos de la programación

Lenguajes de bajo nivel

Tradicionalmente, los ordenadores solían programarse empleando un lenguaje ensamblador, en el que se emplean directamente las instrucciones del procesador.

Existe un programa, también denominado «ensamblador», que traduce dicho lenguaje directamente a código máquina (binario) que el ordenador puede procesar.

El problema del lenguaje ensamblador es que varía de un procesador a otro, por lo que dificulta construir programas portables.

Fundamentos de la programación

Lenguajes de alto nivel

Desde los años 50 del siglo XX, surgen lenguajes de programación más genéricos, que no requieren conocer las instrucciones concretas que proporciona el procesador.

Alto nivel (C++)

```
int main () {  
    int sum = 0;  
    for (int i=0; i<10; i++)  
        sum += i;  
}
```

*Ejemplo de código escrito en C++
(lenguaje de alto nivel) para calcular
la suma de los 10 primeros números.*

Ensamblador

```
main:  
    daddiu    $sp, $sp, -32  
    sd        $fp, 24($sp)  
    move      $fp, $sp  
    sw        $0, 0($fp)  
    sw        $0, 4($fp)
```

```
.L3:  
    lw        $2, 4($fp)  
    slt       $2, $2, 10  
    beq       $2, $0, .L2  
    nop  
    lw        $3, 0($fp)  
    lw        $2, 4($fp)  
    addu      $2, $3, $2  
    sw        $2, 0($fp)  
    lw        $2, 4($fp)  
    addiu     $2, $2, 1  
    sw        $2, 4($fp)  
    b         .L3  
    nop
```

```
.L2:  
    move      $2, $0  
    move      $sp, $fp  
    ld        $fp, 24($sp)  
    daddiu    $sp, $sp, 32  
    j         $31  
    nop
```

Fundamentos de la programación

Compiladores e intérpretes

Un compilador es un programa que permite «convertir» código escrito en un lenguaje de programación de alto nivel en otro lenguaje (normalmente, de más bajo nivel).

Esto permite convertir código escrito en un lenguaje de alto nivel en ensamblador, que posteriormente puede ser ensamblado a código máquina para ser ejecutado.

Algunos lenguajes son interpretados: en este caso no hay un compilador, sino que las instrucciones se van «interpretando» (y ejecutando) una a una.

Python

Introducción

Python es un lenguaje de programación interpretado y multiplataforma:

- Interpretado: los programas se ejecutan instrucción a instrucción, sin compilar.
- Multiplataforma: el mismo código permite desarrollar programas para varios sistemas operativos (Windows, Linux, macOS, ...) y procesadores.

Nace en 1991, pero adquiere gran relevancia en el mundo de la programación, particularmente para *data science* y *machine learning*.



Python

¿Por qué Python?

El lenguaje más popular

De acuerdo con PYPL, un proyecto que estima la popularidad de los lenguajes de programación en función de las búsquedas realizadas en Google, Python es el lenguaje más popular en la actualidad, seguido de lejos por Java.

	Lenguaje	Share
1	Python	28%
2	Java	15,9%
3	JavaScript	9,4%
4	C#	6,7%
5	C / C++	6,5%
6	PHP	4,9%
7	R	4,4%
8	TypeScript	3%
9	Swift	2,6%
10	Objective-C	2,2%

*Datos de septiembre de 2023 del índice PYPL.
Fuente: <https://pypl.github.io/>*

Python

¿Por qué Python?

Aplicaciones comunes: *data science*

En la actualidad, Python es el lenguaje de programación más empleado para ciencia de datos.

Esto se debe a la sencillez que proporciona para realizar análisis complejos, otorgando un buen compromiso entre eficiencia y velocidad de desarrollo.

Además, existe una comunidad muy amplia de desarrolladores, y librerías muy bien documentadas para resolver numerosos problemas de análisis de datos.



Python

¿Por qué Python?

Aplicaciones comunes: inteligencia artificial

Python es el lenguaje más popular a la hora de desarrollar aplicaciones de inteligencia artificial y, en particular, de entrenar modelos de *machine learning*.

Esto se debe a la cantidad de librerías disponibles, que proporcionan implementaciones de numerosas técnicas para abordar problemas de todo tipo: visión por computador, lenguaje natural, modelado de series temporales, etc.

La existencia de estas librerías permite aprender modelos y ponerlos a funcionar en tiempos récord.



Python

¿Por qué Python?

Aplicaciones comunes: desarrollo web

A la hora de desarrollar el *backend* (es decir, el código que se ejecuta en el servidor), Python es una opción que está ganando adeptos, adelantando a algunos competidores como Java o PHP.

Esto se debe a la existencia de numerosos *frameworks* que permiten simplificar notablemente el desarrollo de aplicaciones web, facilitando la conexión con bases de datos, librerías de terceros, etc.



Python

¿Por qué Python?

Software desarrollado en Python



Instagram

La conocida red social está desarrollada empleando Django, que es un *framework* web escrito enteramente en Python.



Pinterest

El motor web de la conocida red social de imágenes está programado en Python.



Spotify

Aproximadamente el 80% de los servicios de *backend* y de análisis de datos de la compañía están programados en Python.



BitTorrent

El conocido software para el intercambio de archivos está escrito enteramente en Python.



Dropbox

La mayoría de los servicios de la compañía, y la aplicación de escritorio, están programados empleando Python.



Los Sims 4

El popular juego permite el uso de Python para programar algunas modificaciones y automatizar algunas tareas.

Python

Cómo ejecutar programas

Para ejecutar programas en Python, debemos escribir el código fuente del programa y luego introducirlo a un intérprete

Para facilitar la tarea, existen «cuadernos» que permiten escribir código e incorporan un intérprete integrado

- Por ejemplo: Google Colab (entorno online con intérprete de Python)

Python

Google Colab

En este curso emplearemos Python con Google Colab para aprender a programar



El programa «hola mundo»

Introducción

El programa «hola mundo» es un ejemplo clásico de un programa sencillo para ilustrar un lenguaje de programación

Consiste únicamente en mostrar por pantalla el texto «Hola mundo»

El programa «hola mundo»

Comparativa de algunos lenguajes de programación



C

```
#include <stdio.h>

int main(void) {
    printf("Hola, mundo");
}
```



Java

```
class HolaMundoApp {
    public static void main(String[] args) {
        System.out.println("Hola, mundo");
    }
}
```

El programa «hola mundo»

Comparativa de algunos lenguajes de programación



Fortran

```
program holamundo  
    print *, "Hola, mundo"  
end program holamundo
```



Pascal

```
program HolaMundo(output);  
begin  
    Write('Hola, mundo')  
end.
```



Go

```
package main  
import "fmt"  
func main() {  
    fmt.Println("Hola, mundo")  
}
```

El programa «hola mundo»

Python



```
print("Hola, mundo")
```

¿Qué son las variables?

Una variable es un nombre que le asignamos a un valor, para utilizarlo posteriormente.

```
>> a = 3
```

declaramos una variable «a» y le asignamos el valor 3

```
>> print(a + 5 )
```

accedemos al valor de la variable «a», que es 3

8

Empleando variables

Declaración y asignación

Decimos que «declaramos» una variable cuando la creamos. «Asignamos» un valor a una variable mediante el símbolo «=».

En Python, la declaración se realiza en la primera asignación.

```
>> a = 3      # Declaración y asignación
>> print(a)   # Muestra por pantalla el valor «3»
>> a = 5      # Asignación (no es declaración, pues «a» ya existía)
```

⚠ ¡Cuidado!

Al asignar un valor a una variable, el nombre de la variable va a la izquierda del «=»

```
>> a = 3  ✓
>> 3 = a  ✗
```

ℹ Curiosidad

En Python, la declaración se realiza en la primera asignación. En otros lenguajes, la declaración y la asignación pueden realizarse por separado.

```
>> int a;  // Declaración
>> a = 3;  // Asignación
```



Empleando variables

Asignación múltiple

Podemos asignar valor a varias variables al mismo tiempo.

Para ello, en el lado izquierdo incluiremos varios nombres de variables, y en el derecho los valores para cada una de ellas:

```
>> a, b = 3, "Hola"           # Asignación de 2 variables a la vez
>> a, b, c = 1, 2, 3          # Asignación de 3 variables a la vez
```

Es fundamental que el número de variables coincida con el de valores:

```
>> a, b = 1, 2, 3             # Esto provoca un error
```

Empleando variables

Acceso

Decimos que se «accede» a una variable cuando se recupera su valor.
«Asignamos» un valor a una variable mediante el símbolo «=».

```
>> a = 3          # Declaración y asignación  
>> print(a+2)    # Acceso
```

Una misma instrucción puede realizar asignación y acceso a una misma variable:

```
>> a = a + 1      # Sintaxis alternativa: a += 1
```

Paso 1: se accede al valor de «a»

Paso 2: se le suma 1

Paso 3: se asigna el resultado a la variable «a»

Solo se puede acceder a una variable declarada anteriormente

Empleando variables

Nombres de variables

Los nombres de variables deben cumplir algunos requisitos:

- Solo deben contener letras (mayúsculas o minúsculas), números y guiones bajos «_».
- No pueden comenzar por un número.

Ejemplos:

```
>> CuneF = 3 ✓ Es un nombre válido
>> a151 = 3 ✓ Es un nombre válido
>> b_12 = 3 ✓ Es un nombre válido
>> _aB1 = 3 ✓ Es un nombre válido
>> a!17 = 3 ✗ Es un nombre no válido: contiene el carácter «!»
>> 1abc = 3 ✗ Es un nombre no válido: comienza por un número
```

¿Qué puede almacenar una variable?

Hasta ahora, hemos visto cómo asignar un número entero (sin decimales) a una variable.
¿Qué más podemos asignar?

```
>> i = 42  
>> f = 3.1416  
>> s = "Hola, mundo"  
>> b = False
```

En Python existen algunos «tipos básicos de datos». Vamos a verlos...

Tipos de datos

Visión general

Podemos considerar cuatro tipos fundamentales:

Tipo	Descripción	Ejemplos
int	Número entero, sin parte decimal.	42 15324
float	Número de coma flotante, es decir, decimal. Puede estar expresado en notación científica.	3.1416 6.67E-11
str	Cadena de texto, que debe especificarse entre comillas dobles (") o simples (').	"Hola mundo" 'Buenas tardes, señora'
bool	Valor binario, solo puede tomar dos valores.	True False

Tipos de datos

Enteros y decimales

Podemos emplear cualquier valor entero, por grande que sea:

```
>> i = -1234567890123456789012345678901234567890
```

En los números enteros, nunca debemos escribir el separador de miles:

```
>> i = 15.800
```

✗ Esto es «quince con ocho», no «quince mil ochocientos»

Los números decimales irán separados por punto, no por coma:

```
>> pi = 3.1416
```

✓ Se está asignando un número decimal correctamente

```
>> pi = 3,1416
```

✗ El valor asignado no es un número decimal

Podemos emplear notación científica:

```
>> G = 6.67E-11
```

✓ El valor asignado es $6,67 \cdot 10^{-11}$

Tipos de datos

Cadenas de texto

Podemos crear una cadena de texto empleando comillas dobles:

```
>> s1 = "Hola, mundo"
```

También podemos crearla empleando comillas simples:

```
>> s2 = 'Hola, mundo'
```

¿Y si queremos que la cadena tenga comillas en su contenido?

```
>> c1 = 'Hola, "mundo"'
```

```
>> c2 = "Hola, 'mundo'"
```

```
>> c2 = "Hola, \"mundo\"" # El símbolo \ permite caracteres especiales
```

```
>> c2 = 'Hola, \'mundo\''
```


Tipos de datos

Valores binarios

Existen dos posibles valores binarios:

```
>> t = True
```

```
>> f = False
```

Pueden interpretarse como sigue:

- «True» es un valor verdadero, por ejemplo, indica que se cumple una condición
- «False» es un valor falso, por ejemplo, indica que no se cumple una condición

Tipos de datos

Transformación de datos

Algunos datos pueden convertirse de un tipo a otro:

```
>> int(7.5)           # 7
>> str(12.1)          # "12.1"
>> int(True)          # 1
>> float("7.5")       # 7.5
```

Para conocer el tipo de un valor (o variable) podemos emplear `type`:

```
>> type("7.5")        # str
>> type(7.5)           # float
>> type(7)              # int
>> type(False)         # bool
```

Operadores

Visión general

Todos estos tipos de datos pueden emplearse para operar.
Existen varios tipos de operadores:

Tipo	Descripción	Ejemplos
Aritméticos	Permite realizar cálculos aritméticos. Retornan un valor numérico.	Suma (+), resta (-), producto (*), etc.
Relacionales	Permiten comparar valores. Retornan un valor booleano.	Igual (==), mayor o igual (>=), distinto (!=)
Booleanos	Permiten operar con valores binarios. Retornan un valor booleano.	AND (and), OR (or), NOT (not), XOR (^)
Cadenas de texto	Permiten modificar cadenas de texto. Retornan una nueva cadena de texto.	Concatenación (+), acceso a caracteres ([])

Operadores

Aritméticos

Se emplean con valores enteros y decimales.

Están disponibles los operadores clásicos: suma (+), resta (−), producto (*) y división (/):

```
>> a = 2.5      # Valor decimal      >> a - b      # -0.5
>> b = 3        # Valor entero       >> a * b      # 7.5
>> a + b        # 5.5                >> b / a      # 1.2
```

También está disponible la división entera o cociente (//), y el módulo o resto (%):

```
>> 50 // 6      # 8                  >> 50 % 6      # 2
```

También podemos calcular potencias (**):

```
>> 5 ** 3       # 125 = 53 = 5*5*5
```

Operadores

Relacionales

Sirven para comparar valores, y devuelven un valor booleano: True o False.

Los operadores habituales son:

- Igualdad (`==`). Devuelve True si ambos valores son iguales.
- Menor (`<`). Devuelve True si el primer valor es menor que el segundo.
- Mayor (`>`). Devuelve True si el primer valor es mayor que el segundo.
- Menor o igual (`<=`). Devuelve True si el primer valor es menor o igual que el segundo.
- Mayor o igual (`>=`). Devuelve True si el primer valor es mayor o igual que el segundo.
- Distinto (`!=`). Devuelve True si ambos valores no son iguales.

Con valores numéricos se emplea el orden usual (recta real).

Con cadenas de texto, se sigue un orden lexicográfico especial.

<code>>> -2 < 3.5</code>	<code># True</code>	<code>>> a = 2</code>
<code>>> 4 != 4</code>	<code># False</code>	<code>>> b = 3</code>
<code>>> "a" < "B"</code>	<code># False</code>	<code>>> c = 4</code>
<code>>> 5 == 5</code>	<code># True</code>	<code>>> a + b >= c # True</code>

Operadores

Booleanos

Permiten operar con valores binarios.

Los operadores habituales son:

- AND (**and**, **&**). Devuelve True si ambos valores son True.
- OR (**or**, **|**). Devuelve True si al menos uno de los valores es True.
- NOT (**not**). Invierte el valor: devuelve True si el valor es False y viceversa.
- XOR (**^**). Devuelve True si los dos valores binarios son distintos.

>> True and True	# True	>> False True	# True
>> False or False	# False	>> False ^ False	# False
>> not True	# False	>> True ^ False	# True
>> False & True	# False	>> not False	# True

Operadores

Cadenas de texto

Se emplean con cadenas de texto.

La concatenación (+) permite unir cadenas de texto:

```
>> a = "Hola "  
>> b = "Mundo"  
>> a + b          # "Hola Mundo"
```

Podemos conocer la longitud de una cadena de texto empleando `len`:

```
>> len("Hola mundo")    # 10
```

Podemos transformar una cadena a mayúsculas (`upper`) o minúsculas (`lower`):

```
>> "Hola Mundo".upper()    # "HOLA MUNDO"  
>> "Hola Mundo".lower()    # "hola mundo"
```

Comentarios

En este tema, hemos empleado muchas veces el símbolo `#` para hacer anotaciones.

Esto es debido a que este carácter inicia un comentario en Python. Los comentarios son fragmentos de texto que sirven para hacer anotaciones, pero que el intérprete ignora.

```
>> a = a + 1    # Incrementa la variable a en una unidad.
```

Hay dos tipos de comentarios:

- Comentario de una línea: comienza con el símbolo `#` y termina cuando finaliza la línea.
- Comentario de varias líneas: es un bloque de texto rodeado por tres comillas:

```
"""  
Esto es un ejemplo de un comentario  
que ocupa varias líneas  
"""  
  
a = a + 1
```