# Parte I: Programación en un lenguaje orientado a objetos

1. Introducción a los lenguajes de programación
2. Datos y expresiones
3. Clases
4. Estructuras algorítmicas
5. Estructuras de Datos
6. Tratamiento de errores
7. *Entrada/salida*
   - Escritura de texto con formato. Lectura de números con formato. Ficheros. Lectura de ficheros de texto. Escritura de ficheros de texto.
8. Herencia y Polimorfismo

# 7.1. Escritura de texto con formato

Hemos usado los `f-strings` para generar texto con formato

- por ejemplo, podemos usar `f-strings` al escribir datos con `print()`
- tienen texto libre y plantillas para datos delimitadas con `{}`
- el formato se indica tras el dato separado por `":"`


Ejemplo

```
print (f"Soy {nombre} de {edad:4d} años")
```

Produce la salida (suponiendo nombre="Pedro", edad=18)

```
Soy Pedro de   18 años
```

# Datos en columna

## Ejemplo

```
Santander
Datos meteorológicos promedio

Mes Máx / Mín Lluvias
Septiembre 22° / 15° 9 días
Octubre 19° / 13° 11 días
Noviembre 16° / 10° 12 días
Diciembre 13° / 8° 11 días
```

a) Datos sin formatear

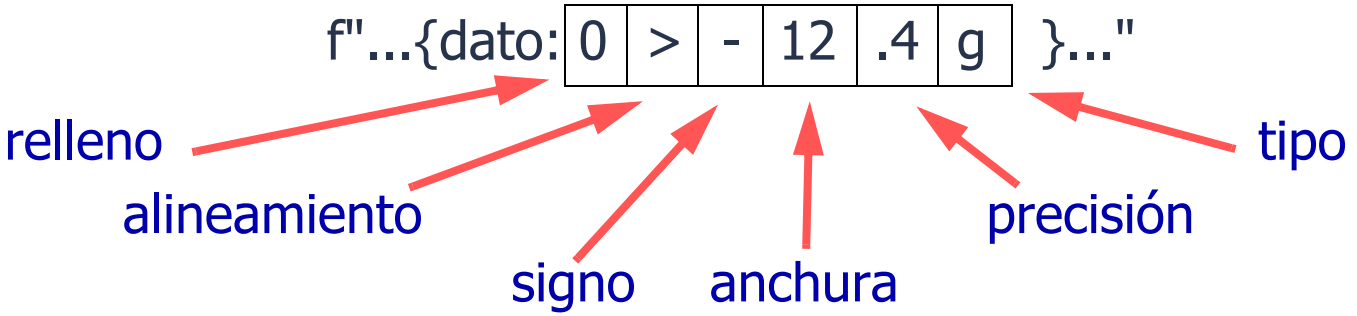
```
Santander
Datos meteorológicos promedio

Mes          Máx / Mín    Lluvias
Septiembre   22° / 15°     9 días
Octubre      19° / 13°    11 días
Noviembre    16° / 10°    12 días
Diciembre    13° /  8°    11 días
```

b) Los mismos datos, formateados en columnas

Los datos en columna son más fáciles de entender y comparar

# Especificaciones de formato habituales

Se componen de los siguientes campos opcionales, en este orden:




- *relleno*: cualquier carácter, que servirá para rellenar los caracteres del dato; por defecto: " "
- *alineamiento*: carácter que indica cómo alinear el dato en su columna

| | | |
|---|---|---|
| > | derecha | por defecto para números |
| < | izquierda | por defecto para strings |
| ^ | centrado | |

# Especificaciones de formato habituales (cont.)

- *tipo*: una letra que indica cómo presentar el dato

| Dato | Tipo | Descripción | Comentario |
|---|---|---|---|
| str | s | string | por defecto para strings |
| int | d | entero decimal | por defecto para enteros |
| | n | número en el formato local | |
| float | e o E | real con notación *exponencial* | |
| | f o F | real con coma *fija* | |
| | g o G | real con formato *general* (exponencial o coma fija según el tamaño) | por defecto para reales |
| | n | número general (como la 'g') pero en el formato local | |

# Especificaciones de formato habituales (cont.)

- *signo*:

| | | |
|---|---|---|
| + | siempre, sea + o - | |
| - | solo si - | por defecto |

- *anchura*: un número entero, que indica la anchura mínima total, incluyendo punto decimal, signo, exponente si lo hay, etc.
- *.precisión*: un punto seguido de un número entero, que indica
  - el número de decimales para números reales: tipos fijo (`f`, `F`) y exponencial (`e`, `E`)
  - o el número de dígitos significativos: tipos general (`g`, `G`) y número (`n`)

# Ejemplos de especificación de anchura y precisión

Para números puede ponerse la especificación de anchura mínima y (si corresponde) el número de decimales; ejemplos con la constante `math.pi` y el valor `i=18`

| Dato | Parámetro del print() | Salida |
|---|---|---|
| `float` | `f"Pi= {math.pi:4.0f}"` | `Pi=    3` |
| | `f"Pi= {math.pi:4.2f}"` | `Pi= 3.14` |
| | `f"Pi= {math.pi:12.4f}"` | `Pi=       3.1416` |
| | `f"Pi= {math.pi:<12.4f};"` | `Pi= 3.1416      ;` |
| | `f"Pi= {math.pi:12.8f}"` | `Pi=   3.14159265` |
| `int` | `f"I= {i:8d}"` | `I=       18` |
| | `f"I= {i:+4d}"` | `I=  +18` |
| | `f"I= {i:04d}"` | `I= 0018` |

# Ejemplos de especificación de tipos

En los formatos g y n la precisión no indica el número de decimales, sino el número de dígitos significativos

Ejemplos con la constante `math.pi`

| Dato | Parámetro del print() | Salida |
|---|---|---|
| `float` | `f"Pi= {math.pi:10.2e}"` | `Pi=   3.14e+00` |
| | `f"Pi= {math.pi:10.2E}"` | `Pi=   3.14E+00` |
| | `f"Pi= {math.pi:10.2g}"` | `Pi=         3.1` |
| | `f"Pi= {math.pi:10.4g}"` | `Pi=       3.142` |
| | `f"Pi= {math.pi:10.6g}"` | `Pi=     3.14159` |
| | `f"Pi= {math.pi:10.9g}"` | `Pi= 3.14159265` |

# Uso de formatos locales (español)

Con el formato n es posible definir el uso de localismos, como la "*coma*" decimal y el separador de *miles* en español. Para usarlo, debemos invocar a `setlocale()`:

```
import locale
locale.setlocale(locale.LC_ALL, "es_ES.UTF-8")
```

Ejemplos de formateo de números en español, con la constante `math.pi` y el valor `j=18000`

| Dato | Parámetro del print() | Salida |
|---|---|---|
| float | f"Pi= {math.pi:10.4n}" | Pi=  3,142 |
| | f"Pi= {math.pi:10.6n}" | Pi=  3,14159 |
| | f"Pi*1000= {math.pi*1000:10.6n}" | Pi*1000=  3.141,59 |
| int | f"I= {j:6n}" | I= 18.000 |
| | f"I= {j:8n}" | I=  18.000 |

# 7.2 Lectura de números con formato

Habitualmente las funciones de entrada, como `input()`, leen un string y luego lo convertimos a número real o entero con `float()` o `int()`

- Estas conversiones solo funcionan para números en notación inglesa

Para hacer conversión de números en notación española, después de especificar los localismos con `locale.setlocale()`, podemos usar

| Función | Descripción |
|---|---|
| `locale.atof(s: str) -> float` | Convierte el string `s` a número real |
| `locale.atoi(s: str) -> int` | Convierte el string `s` a número entero |

Ejemplos

| Código | Resultado |
|---|---|
| `locale.atof("12.345,56")` | 12345.56 |
| `locale.atoi("10.000")` | 10000 |

# 7.3. Ficheros

***Fichero***:

- es una secuencia de bytes en un dispositivo de almacenamiento habitualmente *persistente*: disco duro, DVD, memoria USB, ...
- se puede leer y/o escribir
- se identifica mediante un nombre
  - absoluto (*pathname*), que comienza por "/"
    ```
    /home/pepe/documentos/un_fichero
    ```
  - o relativo a la carpeta del proyecto
    ```
    un_fichero
    ```

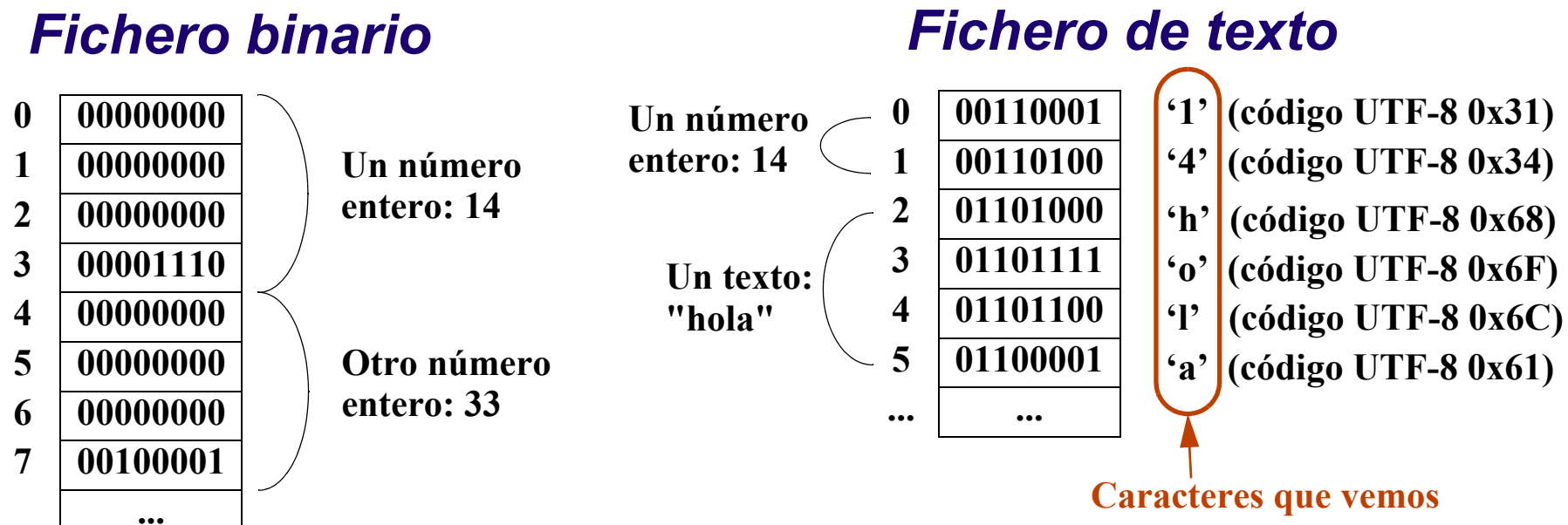
Tipos de ficheros:

- ***programas***: contienen instrucciones
- ***datos***: contienen información, como números (enteros o reales), secuencias de caracteres, ...

# Ficheros de texto y binarios

Tipos de ficheros de datos:

- ***de bytes*** (binarios): pensados para ser leídos por un programa
- ***de caracteres*** (de texto): pueden ser leídos y/o creados por una persona



# 7.4. Lectura de ficheros de texto

La operación `open` abre un fichero y retorna un objeto para usarlo

```
fich = open("datos.txt", "r")
```

nombre del fichero

modo (read)

Sobre el objeto fichero `fich` disponemos entre otras de las operaciones

- `read()`: lee el fichero completo y lo retorna como string
- `readline()`: lee una línea del fichero y la retorna como string
  - se empieza por la primera línea
  - cada vez que la invocamos obtenemos la línea siguiente
  - cuando el fichero se acaba, obtenemos un string vacío: `""`
  - cada línea leída (aunque esté vacía) conserva el salto de línea (carácter `\n`) como último carácter
  - después de leer la línea podemos partirla en trozos con `split()` y convertir alguno de ellos a número con `float()` o `int()`

# Ejemplos de lectura

Leer todo el fichero y mostrarlo

```
fich = open("datos.txt", "r", encoding="utf-8")
contenido: str = fich.read()
print(contenido)
```

Leer cada línea del fichero y mostrarla. Lo haremos iterando en un bucle

```
fich = open("datos.txt", "r", encoding="utf-8")
for linea in fich:
    print(linea, end="") # línea ya tiene el \n
```

El parámetro opcional `encoding` permite indicar la codificación del fichero de entrada y es útil si se quieren leer caracteres internacionales como letras acentuadas o "ñ"

# Cerrar el fichero

Una vez acabado de usar debemos cerrar el fichero

- `close()`: cierra el fichero y libera los recursos que usa

```
fich = open("datos.txt", "r", encoding="utf-8")
for linea in fich:
    print(linea, end="") # línea ya tiene el \n
fich.close()
```

Como esto es muy importante, hay una orden especial (`with`) para hacer el `close()` automático, incluso aunque salte una excepción:

```
with open("da.txt", "r", encoding="utf-8") as fich:
    for linea in fich:
        print(linea, end="")
```

Ya no ponemos el `close()`, pues se hace automático

# Excepciones y ficheros

Las operaciones con ficheros pueden fallar con una excepción del tipo `IOError`

- por ejemplo, si abrimos un fichero que no existe para leerlo
  - esto ocurre típicamente cuando nos equivocamos al dar el nombre del fichero
- otro ejemplo: si abrimos un fichero para escribirlo y está protegido frente a escritura

Por tanto hay que protegerse de esta excepción

En el siguiente ejemplo mostramos cómo hacerlo

# Ejemplo: leer un fichero con palabras seguidas de números

Para el fichero (`mis_datos.txt`):

```
azul 1.0 3.5 7.7
rojo 2
verde 10.0 11.1
```

- Se desea obtener la siguiente salida por consola:

```
Palabra: azul
Número: 1.0
Número: 3.5
Número: 7.7
Palabra: rojo
Número: 2.0
Palabra: verde
Número: 10.0
Número: 11.1
```

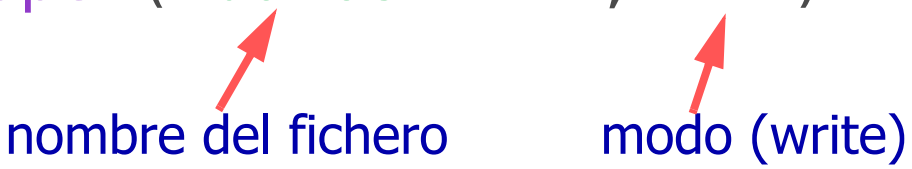
# Ejemplo (cont.)

```
nombre_fichero: str = "mis_datos.txt"
try:
    with open(nombre_fichero, "r",
              encoding="utf-8") as fich:
        for linea in fich:
            # separamos por los espacios en blanco
            palabras: list[str] = linea.split(" ")
            # Mostramos la palabra
            print(f"Palabra: {palabras[0]}")
            # Mostramos los números, de uno en uno
            for pal in palabras[1:]:
                num: float = float(pal)
                print(f"Número: {num}")
except IOError:
    print(f"El fichero {nombre_fichero} no existe")
```

# 7.5 Escritura de ficheros de texto

La operación `open` abre un fichero y retorna un objeto para usarlo

```
fich = open("datos.txt", "w")
```




Estos son los principales modos para ficheros de texto:

| Modo | | Descripción |
|---|---|---|
| `"r"` | `read` | leer (valor por defecto) |
| `"w"` | `write` | escribir, creando el fichero aunque exista |
| `"a"` | `append` | escribe añadiendo al final, aunque lo crea si no existe |
| `"x"` | `exclusive` | escribir creando el fichero, pero falla si ya existe |

Sobre el objeto fichero `fich` disponemos entre otras de la operación

- `write()`: escribe un string en el fichero

# Ejemplo: función que escribe tres datos en un fichero de texto

```
def escribe_fichero(nom_fichero: str, palabra: str,
                    i: int, x_0: float):
    """
    Escribe un string y dos números en un fichero de texto

    Args:
        nom_fichero: el nombre del fichero de texto
        palabra: una palabra que se escribirá en el fichero
        i: un número entero que se escribirá en el fichero
        x_0: un número real que se escribirá en el fichero
    """

    try:
        with open(nom_fichero, "w", encoding="utf-8") as fich:
            fich.write(f"Palabra: {palabra}\n")
            fich.write(f"Entero: {i}, Real: {x_0}\n")
    except IOError as err:
        print(f"Problema con el fichero {nom_fichero}.", err)
```

# Ejemplo: función que escribe en un fichero de texto (cont.)

Ejemplo de fichero generado con

```
escribe_fichero("datos.txt", "Pedro", 12, 34.56)
```

datos.txt:

```
Palabra: Pedro
Entero: 12, Real: 34.56
```