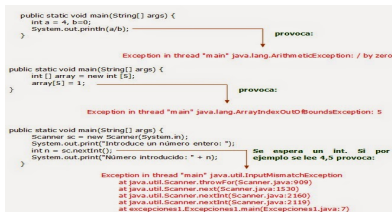


# Control de errores. Entrada y salida de datos

## Excepciones y E/S

L. Daniel Hernández <ldaniel@um.es>

Dpto. Ingeniería de la Información y las Comunicaciones  
Universidad de Murcia  
16 de noviembre de 2023



```
public static void main(String[] args) {
    int a = 4, b=0;
    System.out.println(a/b);
}
provo-ca:
Exception in thread "main" java.lang.ArithmeticException: / by zero

public static void main(String[] args) {
    int [] array = new int [5];
    array[5] = 1;
}
provo-ca:
Exception in thread "main" java.lang.ArrayIndexOutOfBoundsException: 5

public static void main(String[] args) {
    Scanner sc = new Scanner(System.in);
    System.out.print("Introduce un número entero: ");
    int n = sc.nextInt();
    System.out.print("Número introducido: " + n);
}
Se espera un int. Si por ejemplo se lee 4,5 provo-ca:
Exception in thread "main" java.util.InputMismatchException
at java.util.Scanner.throwFor(Scanner.java:909)
at java.util.Scanner.next(Scanner.java:1530)
at java.util.Scanner.nextInt(Scanner.java:2160)
at java.util.Scanner.nextInt(Scanner.java:2118)
at excepciones1.Excepciones1.main(Excepciones1.java:7)
```

1

<sup>1</sup> Imagen: <http://puntoscomnoesunlenguaje.blogspot.com/2014/04/java-excepciones.html>

# Índice de Contenidos

## 1 Excepciones

Capturar Excepciones

Lanzar Excepciones

Definir Excepciones del Usuario

## 2 Entrada y Salida de Datos

Entrada y Salida Estándar

Entrada y Salida a Ficheros

Ficheros con JSON

Ficheros con pickle

Librerías

Más sobre Ficheros Locales. El módulo `os`

Ficheros Lejanos. El módulo `urllib`

## 3 Ejercicios



## 1 Excepciones

- Capturar Excepciones

- Lanzar Excepciones

- Definir Excepciones del Usuario

## 2 Entrada y Salida de Datos

- Entrada y Salida Estándar

- Entrada y Salida a Ficheros

- Ficheros con JSON

- Ficheros con pickle

- Librerías

- Más sobre Ficheros Locales. El módulo `os`

- Ficheros Lejanos. El módulo `urllib`

## 3 Ejercicios



# Excepción = Error

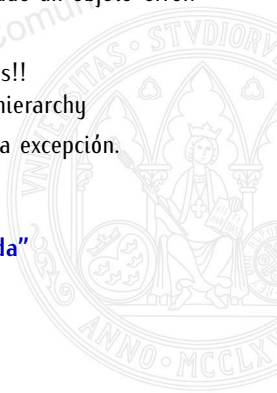
- Lo que menos le gusta a un programador son los **errores**
- Considera los siguientes códigos y observa las excepciones que genera:

```
>>> print(a)
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
NameError: name 'a' is not defined
>>> print(55/0)
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
ZeroDivisionError: division by zero
>>> print('2' + 2)
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
TypeError: can only concatenate str (not "int") to str
>>> lista = None
>>> for i in range(2):
...     print(lista[i])
...
Traceback (most recent call last):
  File "<stdin>", line 2, in <module>
TypeError: 'NoneType' object is not subscriptable
>>> lista = []
>>> for i in range(2):
...     print(lista[i])
...
Traceback (most recent call last):
  File "<stdin>", line 2, in <module>
IndexError: list index out of range
```

- Las excepciones incorporadas en Python en <https://docs.python.org/es/3/library/exceptions.html>

# Objetos de Excepciones

- Los errores **en tiempo de ejecución** se manejan usando **excepciones**.
- **Las excepciones son objetos** que se crean cuando el programa hace algo que se considera irregular.
- En los ejemplos anteriores cada **xxxxError** indica que se ha creado un objeto-error.
- Como objetos, siguen una jerarquía de clases.
- En Python, los usuales son las subclases de **Exception**: ¡¡Leelos!!  
<https://docs.python.org/es/3/library/exceptions.html#exception-hierarchy>
- Cuando se crea una excepción, se dice que se ha **“lanzado”** una excepción.
  - En Java lo llaman **throw** (lanzar, arrojar).
  - En Python lo llaman **raise** (elevar, levantar).
- Para tratar conveniente el error, la excepción debe ser **“atrapada”**
- **Estudiaremos cómo**
  - Capturar excepciones
  - Lanzar/Elevar excepciones
  - Crear nuestros propios objetos excepción



# Profundizamos en ...

## 1 Excepciones

- Capturar Excepciones

- Lanzar Excepciones

- Definir Excepciones del Usuario

## 2 Entrada y Salida de Datos

- Entrada y Salida Estándar

- Entrada y Salida a Ficheros

- Ficheros con JSON

- Ficheros con pickle

- Librerías

- Más sobre Ficheros Locales. El módulo `os`

- Ficheros Lejanos. El módulo `urllib`

## 3 Ejercicios



# Capturar excepciones

- Para manejar excepciones se usan la sentencias `try/except/else/finally`

```
try:
    // Código que se desea ejecutar

except [zzzError [as nombre]]:
    // Código a ejecutar si se produce un error en try
    // Código que controla la situación de error. Puede lanzar otro error.

else:
    // Bloque a ejecutar si try no tuvo errores

finally:
    // Código que se ejecutará siempre. Ocurra lo que ocurra en el programa.
    // P.e. cerrar los recursos
```

# Ejemplo de Captura de Excepciones

try/except/finally

## Ejemplo:

- Veamos antes el error sin capturas.

```
>>> print(un_mensaje)
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
NameError: name 'un_mensaje' is not defined
```

- Con las 3 componentes; pero a **except** le da el igual el error.

```
>>> try:
...     print(un_mensaje) # Código que se desea ejecutar
... except:               # Salvo que se genere una excepción. Observa la salida !!
...     print('Opppss ... hubo un error') # No importa el error
... finally:             # Siempre se puede obligar a cierto código
...     print('Siempre muestro este mensaje')
...
Opppss ... hubo un error
Siempre muestro este mensaje
```



# Ejemplo de Captura de Excepciones

try/except

Ejemplo:

- **except** solo se activa si el error es **NameError**.

```
>>> try:
...     print(un_mensaje) # Código que se desea ejecutar
... except NameError:    # Salvo que se genere una excepción
...     print('Opppss ... hubo un error')
...
Opppss ... hubo un error
```

- **except** asigna el error a una variable para poder ser manipulado.

```
>>> try:
...     print(un_mensaje) # Código que se desea ejecutar
... except NameError as error: # Salvo que se genere una excepción
...     print(f'Opppss ... hubo un "{error}")')
...
Opppss ... hubo un "name 'un_mensaje' is not defined"
```

# Ejemplo de Captura de Excepciones con else

<https://www.programiz.com/python-programming/exception-handling>

```
>>> def funcion(num):  
...     try:  
...         print(f"Número introducido {num}")  
...         assert num % 2 == 0  
...     except: # Si try SÍ ha generado excepciones  
...         print("No es un número par")  
...     else:   # Si try NO ha generado excepciones  
...         reciproco = 1/num  
...         print(reciproco)  
... 
```

- Se activa **except**

```
>>> funcion(1)  
Número introducido 1  
No es un número par
```

- Se activa **else**

```
>>> funcion(4)  
Número introducido 4  
0.25
```

# Varios except

- Si se conoce la jerarquía de clases de las excepciones se pueden usar varios "except".
- Las capturas "except" se ordenarán desde las subclases a las superclases
- Código correcto.

```
>>> try:
...     print(100/0)
... except ZeroDivisionError: # ZeroDivisionError es SUBclase de ArithmeticError
...     print("Error: divides por cero")
... except ArithmeticError:
...     print("Error aritmético")
...
Error: divides por cero
```

- Código incorrecto

```
>>> try:
...     print(100/0)
... except ArithmeticError: # ArithmeticError es una SUPERclase de ZeroDivisionError
...     print("Error aritmético")
... except ZeroDivisionError:
...     print("Error: divides por cero")
...
Error aritmético
```

Nunca se capturará un objeto de `ZeroDivisionError`.

# Profundizamos en ...

## 1 Excepciones

Capturar Excepciones

**Lanzar Excepciones**

Definir Excepciones del Usuario

## 2 Entrada y Salida de Datos

Entrada y Salida Estándar

Entrada y Salida a Ficheros

Ficheros con JSON

Ficheros con pickle

Librerías

Más sobre Ficheros Locales. El módulo `os`

Ficheros Lejanos. El módulo `urllib`

## 3 Ejercicios



# Lanzar excepciones

- El programador también puede crear excepciones
- El programador puede **lanzar una excepcion** utilizando **raise** o **assert**.

**Ejemplo:** Para indicar que un método aún no ha sido implementado.

```
>>> def metodo():
...     raise NotImplementedError("Not supported yet.")
...
>>> metodo()
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
  File "<stdin>", line 2, in metodo
NotImplementedError: Not supported yet.
```

**Ejemplo:** Para indicar que un parámetro no es el adecuado.

```
>>> def metodo(param):
...     assert param > 0, "Deber ser positivo"
...
>>> metodo(-1)
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
  File "<stdin>", line 2, in metodo
AssertionError: Deber ser positivo
```

```
>>> def metodo(param):
...     if param <= 0:
...         raise ValueError("Deber ser positivo")
...
>>> metodo(-1)
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
  File "<stdin>", line 3, in metodo
ValueError: Deber ser positivo
```

# Excepciones en Setter/Propiedades

- La interface Setter permite asignar valores a los atributos de instancia de una clase.
- Los métodos Setter de una atributo deben lanzar errores/excepciones cuando los valores a asignar sean incorrectos.
- Alternativamente, se puede usar propiedades.

Mira <https://realpython.com/python-property/#putting-pythons-property-into-action> para usar property con raise.

**Ejemplo:** Para indicar que un valor debe ser entero positivo.

```
class Dia:
    def __init__(self, d: int):
        self._dia = d
    @property
    def dia(self):
        return self._dia
    @dia.setter
    def dia(self, valor: int):
        try:
            self._dia = int(valor)
            assert self._dia > 0
        except Exception as e:
            print(f"#ERROR {e.__class__}")
```

# Profundizamos en ...

## 1 Excepciones

Capturar Excepciones

Lanzar Excepciones

Definir Excepciones del Usuario

## 2 Entrada y Salida de Datos

Entrada y Salida Estándar

Entrada y Salida a Ficheros

Ficheros con JSON

Ficheros con pickle

Librerías

Más sobre Ficheros Locales. El módulo `os`

Ficheros Lejanos. El módulo `urllib`

## 3 Ejercicios



# Creando Excepciones Nuevas

- Pueden crearse subclases a partir de la clase `Exception`
- Usa la notación usual de creación: `class MiError(Exception)`

```
>>> class BaseError(Exception):  
...     """Mi clase base para mis excepciones"""  
...     pass # Lo correcto es definir __init__(self, *args)  
...  
>>> raise BaseError # Alza un error sin texto  
Traceback (most recent call last):  
  File "<stdin>", line 1, in <module>  
__console__.BaseError
```

- Se pueden construir clases hijas: `class ConcretoError(MiError)`

```
>>> class ConcretoError(BaseError):  
...     """Una clase concreta para mis excepciones"""  
...     pass # Lo correcto es definir __init__(self, *args)  
...  
>>> raise ConcretoError("Esto es un error concreto") # Con texto  
Traceback (most recent call last):  
  File "<stdin>", line 1, in <module>  
__console__.ConcretoError: Esto es un error concreto
```

- Es una buena práctica colocar todas las excepciones en algún fichero separado como `exceptions.py` o `errors.py`. Pero esto no es obligatorio.



# Personalizando las Excepciones Nuevas

- Las nuevas clases, como cualquier otra, pueden tener su propio constructor y métodos.
- Esto nos permite crear excepciones más personalizadas.

```
>>> class NotInRangeError(Exception):
...     def __init__(self, valor: int, mensaje="No está en el rango (10, 40)":
...         self._valor = valor
...         self._mensaje = str(valor) + " -> " + mensaje
...         super().__init__(self._mensaje)
...
>>> valor = 50
>>> if valor < 10 or 40 < valor:
...     raise NotInRangeError(valor)
...
Traceback (most recent call last):
  File "<stdin>", line 2, in <module>
__console__.NotInRangeError: 50 -> No está en el rango (10, 40)
```

## Ejercicio.

- Crea la clase `SintaxisError` y sus subclases `SinArrobaSintaxisError` y `CadenaVacíaSintaxisError`
- Sobre la clase `Persona` se tienen los atributos `eMail` y `nombre`.
- Se quieren detectar los errores de que el correo electrónico no tiene arroba `@` y que el nombre dado no es ni nulo ni tiene una longitud inferior a 3 caracteres.
- Escribe las clases indicadas con sus constructores, que deberán de lanzar las correspondientes excepciones si fuera necesario.
- Escribe el programa para una prueba unitaria

## 1 Excepciones

Capturar Excepciones

Lanzar Excepciones

Definir Excepciones del Usuario

## 2 Entrada y Salida de Datos

Entrada y Salida Estándar

Entrada y Salida a Ficheros

Ficheros con JSON

Ficheros con pickle

Librerías

Más sobre Ficheros Locales. El módulo `os`

Ficheros Lejanos. El módulo `urllib`

## 3 Ejercicios



# Entrada y Salida

- Un **stream** (secuencia o flujo) es una representación abstracta de un dispositivo físico de entrada o salida.
- Un stream **puede imaginarse** como un tubo por el que fluyen bytes de datos y dónde una vez transmitido el dato éste no se volverá a enviar.
- Hay **dos tipos de flujos** atendiendo a “su dirección”:
  - Flujos de **entrada**: teclado, archivo de disco, ...
  - Flujos de **salida**: archivo de disco, una consola, impresora ...
- En Python se distinguen los siguientes tipos:
  - **E/S de texto**. En el flujo los datos serán objetos de la clase **str**. Representa los objetos como cadena de caracteres.
  - **E/S binario** (o *buffered I/O*). En el flujo se esperan objetos de tipo binario y produce objetos de tipo **bytes**.
    - Algunos objetos de tipo binarios son **bytes**, **bytearray**, **array.array**, etc ...
    - El tratamiento básico de los objetos **bytes** es como una secuencia de números enteros y cada uno restringido a valores  $x$  con  $0 \leq x < 256$ .
    - Los valores literales de **bytes** son igual que la cadenas de texto anteponiendo la letra **b**.  
Ejemplo: **"hola"** es un str, **b"hola"** es un bytes.
  - **E/S sin formato** (o *unbuffered I/O*). Trata los datos en crudo, a bajo nivel, no hace ningún tipo de encapsulación (agrupamiento).

# Profundizamos en ...

## 1 Excepciones

- Capturar Excepciones

- Lanzar Excepciones

- Definir Excepciones del Usuario

## 2 Entrada y Salida de Datos

- Entrada y Salida Estándar

- Entrada y Salida a Ficheros

- Ficheros con JSON

- Ficheros con pickle

- Librerías

- Más sobre Ficheros Locales. El módulo `os`

- Ficheros Lejanos. El módulo `urllib`

## 3 Ejercicios



# Entrada y Salida Estándar.

- Se trabaja con **E/S de texto**.
- **Mostrar objetos en la pantalla.**

```
print(*objects, sep=' ', end='\n', file=sys.stdout, flush=False)
```

- Todos los argumentos de **\*objects** se convierten a cadenas de caracteres.
- Los objetos se mandan al flujo separados por **sep**.
- Se termina el envío con **end**.
- **file** debe ser un objeto que implemente un método **write(string)**. Si no se indica se usará **sys.stdout**. También puede usar **sys.stderr** (donde se envían las indicaciones del intérprete y mensajes de error).
- **flush** indica si se obliga a vaciar el buffer del stream.

- **Lee datos del teclado.** **input([prompt])**

- Si se indica **prompt** se mostrará en la salida estándar sin nueva línea.
- Lee una línea de la entrada **sys.stdin**, la convierte en una cadena (eliminando la nueva línea), y retorna eso.

```
>>> s=input("Dime tu edad: ")
Dime tu edad: 22
>>> print(f'{'Sí' if int(s)>17 else 'No'} eres mayor de edad')
Sí eres mayor de edad
```

# Formatear Cadenas

- **Formatear cadenas de texto**: cambiar la salida (texto) de un programa para que sea legible por humanos, incluyendo texto y datos.
- Python ofrece **dos formas** para realizar el formato.
- Usar el **método** `str.format()`
  - **str** contienen campos de reemplazo, que son expresiones delimitadas por llaves `{}`

```
>>> for x in range(1, 3):  
...     print("{} {} {}".format(repr(x), x, x*x))  
...  
'1'      1      1.00  
'2'      4      2.00
```

- **Literales de cadena formateados** (formatted string literal o **f-string**).
  - Es un literal de cadena que prefijo 'f' o 'F'.
  - Contienen campos de reemplazo, que son expresiones delimitadas por llaves `{}`

```
>>> for x in range(1, 3):  
...     print(f"{str(x)!r:10} {x*x: 8d} {x: 10.2f}")  
...  
'1'      1      1.00  
'2'      4      2.00
```

[https://docs.python.org/es/3/reference/lexical\\_analysis.html#f-strings](https://docs.python.org/es/3/reference/lexical_analysis.html#f-strings)

# Profundizamos en ...

## 1 Excepciones

- Capturar Excepciones

- Lanzar Excepciones

- Definir Excepciones del Usuario

## 2 Entrada y Salida de Datos

- Entrada y Salida Estándar

- Entrada y Salida a Ficheros**

- Ficheros con JSON

- Ficheros con pickle

- Librerías

- Más sobre Ficheros Locales. El módulo `os`

- Ficheros Lejanos. El módulo `urllib`

## 3 Ejercicios





# Esquema General

Pasos a seguir siempre

## 1. Abrir un fichero. `f = open(file, mode='r')`

- **file** es un string indicando la ubicación del fichero.
- **mode** es el modo de E/S. Por defecto es de lectura con flujo de texto.
- Retorna un **file object**, que tiene una interface para tratar archivos (pasos siguientes) junto con algunos atributos:
  - `f.closed`: True si el fichero está cerrado. Falso en otro caso.
  - `f.mode`: Indica el modo de acceso de cómo se abrió el fichero.
  - `f.name`: Indica el nombre del fichero.

## 2. Leer/Escribir en el fichero.

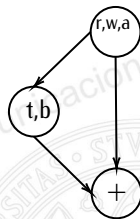
- `f.write(string)`: Escribe un string en el fichero abierto.
- `f.read([n])`: Vuelca en un string el contenido del fichero abierto. Opcionalmente se puede indicar el número de bytes que se quieran leer.
- `f.readline()`: Lee una sola línea del archivo. Deja `\n` al final de la cadena.
- `f.readlines()`: Vuelca en una lista de strings el contenido del fichero
- `f.writelines(lista)`: Escribe una lista de líneas en el archivo.
- `f.tell()`: Indica la posición actual del puntero.
- `f.seek(offset[, from])`: Cambia la posición según offset desde una posición dada. Si `from` es 0 indica el inicio, 1 indica la posición actual y 2 indica la posición final.

## 3. Cerrar el fichero.

- `f.close()`: Cierra el fichero

# Modos de Apertura

- Un fichero se puede **leer**, se puede **sobreescribir** sobre él o hacer **las dos cosas** simultáneamente.
- A la vez, la E/S puede ser de **texto** o **binaria**.
- Python usa varios indicadores para eso.
- **r**: Abre el fichero para **lectura**. El fichero debe existir. El puntero del fichero se coloca al **principio**. Este modo es por defecto.
- **w**: Abre el fichero para **escritura**. Sobreescribe el fichero si existe. El puntero del fichero se coloca al **principio**.
- **a**: Abre el fichero para **añadir**. Si no existe el fichero lo crea. Si existe, coloca el puntero del fichero al **final**.
- **b**: Si se especifica se tratará el fichero como **binario**.
- **t**: Será una E/S de texto. Valor por defecto.
- **+**: Se puede usar el fichero para **leer y escribir**.
- **x**: Abierto para **creación** en exclusiva, falla si el fichero ya existe.



## Ejemplo:

- **rb+**: Para leer. Puntero al principio. Será una E/S binaria. También se podrá escribir.
- **a+**: Para añadir. Puntero al final. Será una E/S texto. También se podrá leer.

# Ejemplos Básicos

```
# Crear un fichero y escribe 3 líneas
f = open("test.txt", "w")
for num in range(3):
    f.write(str(num)+'\n') # \n para nueva línea
f.close()

# Lo abre de nuevo para añadir una cuarta línea
f = open("test.txt", "a")
f.write(str(4)+'\n')
f.close()

# Mostramos el contenido en pantalla
f = open("test.txt")
for linea in f:
    print(linea)
f.seek(0) # Para empezar desde el principio
lista= f.readlines()
f.close()
```



# Ficheros con try/excepts

- Si no existe el fichero para una lectura se generará un error.

```
>>> f = open("testR.txt")
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
FileNotFoundError: [Errno 2] No such file or directory: 'testR.txt'
```

- En estos casos se debe usar try/excepts

```
>>> try:
...     f = open("testR.txt")
... except:
...     print("Oppppps ! ... y no se cierra el fichero.")
... else:
...     f.read()
...     f.close()
...     print("Tarea realizada con éxito")
...
Oppppps ! ... y no se cierra el fichero.
```

# Ficheros con with

- Es importante que no se generen errores ANTES de cerrar el fichero

```
>>> f = open("testW.txt", "w")
>>> f.read()
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
io.UnsupportedOperation: not readable
>>> print(f"Está cerrado? {f.closed}")
Está cerrado? False
```

- Es fundamental cerrar el fichero `f.close()`
- Python garantiza el cierre usando `with`

```
>>> try:
...     with open("testW.txt", "w") as f:
...         f.read()                # Genera error, pero se cerrará
... except:
...     print(f"Está cerrado? {f.closed}")
...
Está cerrado? True
```

Notar que no se usa `f.close()` en el código anterior.

# Algo más sobre la sentencia with

- `with` genera un **Administrador de Contexto**.
- Un Administrador de Contexto **es un objeto que maneja la entrada y salida del contexto** para la ejecución del código que se indique en el bloque (BLOCK).
- `EXPR` es la expresión que **define** el Administrador de Contexto.
- `VAR` es un objeto de asignación.
- Ejemplos en distintos contextos

```
with EXPR as VAR:  
    BLOCK
```

```
# Abrir un fichero y manipularlo  
with open('text.txt', 'r') as f:  
    # Block
```

```
# Abrir una base de datos y dar una respuesta  
with sqlite3.connect('test.db') as connection:  
    # Block
```

- Se puede crear un Administrador de Contexto en una clase con los métodos `__enter__()` and `__exit__()`.

```
def __enter__(self):  
    self.file_handler = open(self.file_name, self.mode)  
    return self.file_handler
```

```
def __exit__(self, exc_type, exc_val, traceback):  
    if self.file_handler:  
        self.file_handler.close()  
    return True
```

Para saber más: <https://realpython.com/python-with-statement/>

# Profundizamos en ...

## 1 Excepciones

Capturar Excepciones

Lanzar Excepciones

Definir Excepciones del Usuario

## 2 Entrada y Salida de Datos

Entrada y Salida Estándar

Entrada y Salida a Ficheros

**Ficheros con JSON**

Ficheros con pickle

Librerías

Más sobre Ficheros Locales. El módulo `os`

Ficheros Lejanos. El módulo `urllib`

## 3 Ejercicios



# JSON

- JSON (JavaScript Object Notation) es un formato ligero de intercambio de datos.

```
{  
  "departamento": 8,  
  "nombredepto": "Ventas",  
  "director": "Juan Rodriguez",  
  "empleados": [  
    {  
      "nombre": "Pedro",  
      "apellido": "Fernández"  
    },  
    {  
      "nombre": "Jacinto",  
      "apellido": "Benavente"  
    }  
  ]  
}
```

- En lo esencial es una diccionario cuyos valores pueden ser: (1) un valor, (2) una lista de valores, (3) una lista de diccionarios.





# Ejemplo - I

Definimos en **una variable** toda la información:

```
>>> empresa = {  
...     "departamento": 8,  
...     "nombredepto": "Ventas",  
...     "director": "Juan Rodríguez",  
...     "empleados": [  
...         {  
...             "nombre": "Pedro",  
...             "apellido": "Fernández"  
...         }, {  
...             "nombre": "Jacinto",  
...             "apellido": "Benavente"  
...         }  
...     ]  
... }
```

## Ejemplo - II

```
>>> import json    # Hay que importar esta librería

>>> with open("json.txt", "w") as f:          # Escritura texto
...     json.dump(empresa, f, indent=4)
...
>>> with open("json.txt", "r") as f:          # Lectura texto. Retorna un Diccionario
...     empresa = json.load(f)
...
>>> print(json.dumps(empresa, indent=4)) # Lo convierte a str con formato JSON
{
  "departamento": 8,
  "nombredepto": "Ventas",
  "director": "Juan Rodr\u00e9guez",
  "empleados": [
    {
      "nombre": "Pedro",
      "apellido": "Fern\u00e1ndez"
    },
    {
      "nombre": "Jacinto",
      "apellido": "Benavente"
    }
  ]
}
```

# Profundizamos en ...

## 1 Excepciones

- Capturar Excepciones

- Lanzar Excepciones

- Definir Excepciones del Usuario

## 2 Entrada y Salida de Datos

- Entrada y Salida Estándar

- Entrada y Salida a Ficheros

- Ficheros con JSON

- Ficheros con pickle**

- Librerías

- Más sobre Ficheros Locales. El módulo `os`

- Ficheros Lejanos. El módulo `urllib`

## 3 Ejercicios



# Serialización de objetos Python

- Los pickles de Python representan un objeto Python como una cadena de bytes.
- Se pueden transmitir por red o guardar en un archivo.

```
>>> import pickle # Hay que importar esta librería

>>> datos = [list().append(2), set().add(100)]

>>> with open("datos.pickle", "wb") as f: # Escritura *binario*
...     pickle.dump(datos, f)
...
>>> with open("datos.pickle", "rb") as f: # Lectura *binario*
...     datos_p = pickle.load(f)
...
>>> print(pickle.dumps(datos_p)) # Pickle como cadena binaria (bytes)
b'\x80\x04\x95\x15\x00\x00\x00\x00\x00\x00\x00\x00\x94[]\x94(K\x01K\x02e\x8f\x94('

>>> print(datos_r) # Lo convierte a str
[[1, 2], {3, 4}]
```

# Profundizamos en ...

## 1 Excepciones

- Capturar Excepciones

- Lanzar Excepciones

- Definir Excepciones del Usuario

## 2 Entrada y Salida de Datos

- Entrada y Salida Estándar

- Entrada y Salida a Ficheros

- Ficheros con JSON

- Ficheros con pickle

### Librerías

- Más sobre Ficheros Locales. El módulo `os`

- Ficheros Lejanos. El módulo `urllib`

## 3 Ejercicios



# No reinventes la rueda

Algunas librerías para manipular ficheros.

**csv:** Leer y escribir a ficheros .csv (Excel)

**xlwings:** Leer y escribir a ficheros .xls (Excel)

**wave:** Leer y escribir a ficheros WAV (audio Microsoft)

**aifc:** Leer y escribir a ficheros AIFF y AIFC (audio)

**tarfile:** Leer y escribir a ficheros tar (compresión)

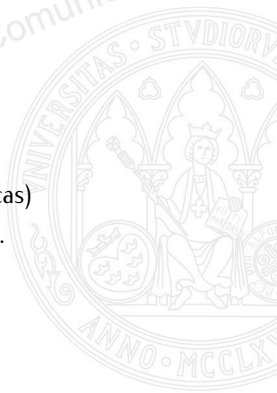
**zipfile:** Leer y escribir a ficheros ZIP (compresión)

**xml.etree.ElementTree:** Leer y escribir a ficheros XML (marcas)

**msilib:** Leer y escribir a ficheros de instalación de Microsoft.

**PyPDF2:** Manipulación de ficheros .pdf (Adobe)

**Pillow:** Manipulación de ficheros de imágenes.



# Profundizamos en ...

## 1 Excepciones

- Capturar Excepciones

- Lanzar Excepciones

- Definir Excepciones del Usuario

## 2 Entrada y Salida de Datos

- Entrada y Salida Estándar

- Entrada y Salida a Ficheros

- Ficheros con JSON

- Ficheros con pickle

- Librerías

- Más sobre Ficheros Locales. El módulo `os`

- Ficheros Lejanos. El módulo `urllib`

## 3 Ejercicios



# No reinventes la rueda

El módulo `os` proporciona métodos que ayudan a la gestión de ficheros.

- Relacionados con ficheros:

`os.rename('current_file_name', 'new_file_name')` Renombra un fichero.

`os.remove('file_name')` Borra un fichero.

- Relacionados con directorios:

`os.mkdir("newdir")` Construye un nuevo directorio.

`os.chdir("newdir")` Cambia el directorio actual.

`os.getcwd()` Muestra el directorio de trabajo actual.

`os.rmdir('dirname')` Borra el directorio actual





# Profundizamos en ...

## 1 Excepciones

- Capturar Excepciones

- Lanzar Excepciones

- Definir Excepciones del Usuario

## 2 Entrada y Salida de Datos

- Entrada y Salida Estándar

- Entrada y Salida a Ficheros

- Ficheros con JSON

- Ficheros con pickle

- Librerías

- Más sobre Ficheros Locales. El módulo `os`

- Ficheros Lejanos. El módulo `urllib`

## 3 Ejercicios



# Puedes acceder a ficheros externos

Proporciona métodos que ayudan a la gestión de ficheros externos vía URL. Es muy útil para análisis de datos (estadística, machine learning, ...)

```
>>> def words_file(url):
...     from urllib import request
...     from urllib.error import URLError
...     try:
...         file = request.urlopen(url)
...     except URLError:
...         # Observa que retorna, NO imprime
...         return('La url ' + url + ' no existe')
...     else:
...         content = file.read()
...         # split() divide un string por espacios.
...         # Construye una lista con cada una de l
...         return len(content.split())
...
>>> print(words_file('https://www.gutenberg.org/files/2000/2000-0.txt'))
389719
>>> print(words_file('https://no-existe.txt'))
La url https://no-existe.txt no existe
```

Fuente: <https://tinyurl.com/27wjb5td> ([visitar](#))

## 1 Excepciones

- Capturar Excepciones

- Lanzar Excepciones

- Definir Excepciones del Usuario

## 2 Entrada y Salida de Datos

- Entrada y Salida Estándar

- Entrada y Salida a Ficheros

- Ficheros con JSON

- Ficheros con pickle

- Librerías

- Más sobre Ficheros Locales. El módulo `os`

- Ficheros Lejanos. El módulo `urllib`

## 3 Ejercicios



## Ejercicio.

A veces querrás leer un fichero y escribir en el otro a la vez.

- Crea la función `def guarda(fichero: str, datos: list)` para guardar una lista de strings en un fichero.
- Crea la función `def backup(fichero_in: str, fichero_out: str)` para hacer un backup de un fichero en otro. Haz dos versiones de la función:
  1. En cada linea de respaldo añade el número de línea.
  2. En el respaldo, las líneas se colocan en el orden inverso al del fichero original.
  - Usa la función `reversed()` que retorna un iterador en el orden inverso del iterable dado.

```
>>> for num in reversed([1, 2]):  
...     print(num)  
...  
2  
1
```

- `.readlines()` usa un iterador para guardar las líneas.

## Ejercicio.

Se tiene un fichero .csv con las calificaciones de los alumnos. Es un fichero de texto donde cada fila constará de 3 datos separados por comas. Puedes generar el fichero con una hoja de cálculo o directamente con un editor que guarde el contenido sin formato.

,Teoría,Prácticas

Alumno A,6,4

Alumno B,3,5

....

Haz un programa que lea el fichero y muestre en pantalla (1) el nombre del alumno junto con el promedio de las dos notas, (2) el número de alumnos aprobados.

## Ejercicio.

Realiza un gestor telefónico con funciones. Guarda y borrar números de teléfono en un fichero de texto usando un menú en la consola. Es decir, con `print()` se mostrará en la consola un texto con las opciones de guardar, borrar y salir, y se usará un `input()` para que el usuario introduzca su opción.