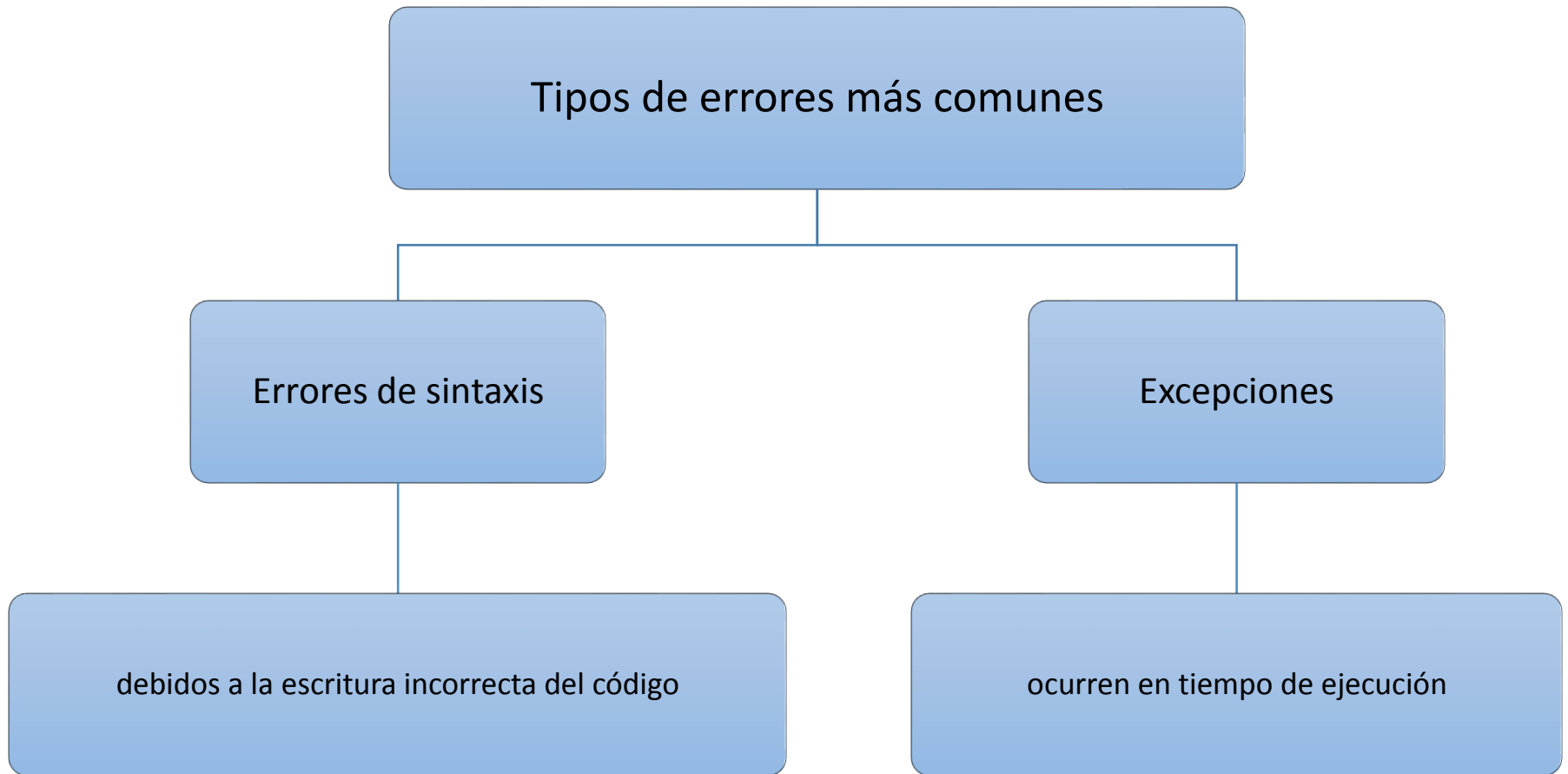


# ERRORES y EXCEPCIONES



Error en tiempo de ejecución  
(excepción)

Finalización  
súbita del  
programa

**Solución**

Control de  
Excepciones

# ERRORES y EXCEPCIONES

Existen muchos tipos de excepciones (*integradas*), que nos permiten controlar multitud de situaciones:

- División por cero (*ZeroDivisionError*)

```
>>> 1.5 * (2 / 0)
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
ZeroDivisionError: division by zero
```

- Error de tipo de datos (*TypeError*)

```
>>> '3' + 5
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
TypeError: Can't convert 'int' object to str implicitly
```

- Fichero inexistente (*FileNotFoundError*)

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

- Índice fuera de rango (*IndexError*)

```
>>> lista = [1, 2, 3, 4]
>>> lista[4]
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
IndexError: list index out of range
```

- *OverflowError, ImportError, ...*

# ERRORES y EXCEPCIONES

Builtin Exceptions: <https://docs.python.org/3/library/exceptions.html>

```
BaseException
+-- SystemExit
+-- KeyboardInterrupt
+-- GeneratorExit
+-- Exception
    +-- StopIteration
    +-- StopAsyncIteration
    +-- ArithmeticError
        |   +-- FloatingPointError
        |   +-- OverflowError
        |   +-- ZeroDivisionError
    +-- AssertionError
    +-- AttributeError
    +-- BufferError
    +-- EOFError
    +-- ImportError
        |   +-- ModuleNotFoundError
    +-- LookupError
        |   +-- IndexError
        |   +-- KeyError
    +-- MemoryError
    +-- NameError
        |   +-- UnboundLocalError
    +-- OSError
        |   +-- BlockingIOError
        |   +-- ChildProcessError
        |   +-- ConnectionError
        |       |   +-- BrokenPipeError
        |       |   +-- ConnectionAbortedError
        |       |   +-- ConnectionRefusedError
        |       +-- ConnectionResetError
        |   +-- FileExistsError
        |   +-- FileNotFoundError
        |   +-- InterruptedError
        |   +-- IsADirectoryError
        |   +-- NotADirectoryError
        |   +-- PermissionError
        |   +-- ProcessLookupError
        |   +-- TimeoutError
    +-- ReferenceError
    +-- RuntimeError
        |   +-- NotImplementedError
        |   +-- RecursionError
    +-- SyntaxError
        |   +-- IndentationError
        |   +-- TabError
    +-- SystemError
    +-- TypeError
    +-- ValueError
        |   +-- UnicodeError
        |       |   +-- UnicodeDecodeError
        |       |   +-- UnicodeEncodeError
        |       +-- UnicodeTranslateError
    +-- Warning
        +-- DeprecationWarning
        +-- PendingDeprecationWarning
        +-- RuntimeWarning
        +-- SyntaxWarning
        +-- UserWarning
        +-- FutureWarning
        +-- ImportWarning
        +-- UnicodeWarning
        +-- BytesWarning
        +-- ResourceWarning
```

Podemos capturar y manejar las excepciones con la cláusula:

```
try:  
    código que puede producir la excepción  
except [tipo_excepción]:  
    código que gestionará el error
```

*# Ejemplo 1*

```
y = 3  
try:  
    x = 10.2 / y  
    print('El resultado es: ', x)  
except:  
    print('Se ha producido un error inesperado')  
...
```

- Primero se ejecuta el bloque *try* (el código entre *try:* y *except:*)
- Si ocurre una excepción, se ejecuta el bloque *except:* inmediatamente. En caso contrario el programa continúa después del *except:*
- Si no se especifica *tipo\_excepción* la cláusula *except:* se ejecutará sea cual sea el error que se produzca.

# ERRORES y EXCEPCIONES

Muchas veces es conveniente especificar el tipo de excepción para tener mayor control sobre los errores.

*# Ejemplo 2*

```
y = 0
try:
    x = 10.2 / y
    print('El resultado es: ', x)
except ZeroDivisionError:
    print('La variable y no puede ser 0')
```

- Si  $y = 0$  entonces se producirá una excepción del tipo *ZeroDivisionError* y se ejecutará el *except:* correspondiente.

*# Ejemplo 3*

```
y = 'a'
try:
    x = 10.2 / y
    print('El resultado es: ', x)
except ZeroDivisionError:
    print('La variable y no puede ser 0')
```

- En este caso el programa abortará su ejecución al intentar hacer la división ya que la variable *y* contiene una cadena en lugar de un número y no hemos puesto ninguna excepción que maneje esta situación (*TypeError*)

## Manejando varios tipos de excepciones de manera individual:

*# Ejemplo 4*

```
y = 'a'  
try:  
    x = 10.2 / y  
    print('El resultado es: ', x)  
except ZeroDivisionError:  
    print('La variable y no puede ser 0')  
except TypeError:  
    print('La variable y no puede contener una cadena de texto')
```

- En este caso se contemplan o se diferencian dos tipos de excepciones que podrían ocurrir en la división y se tratan de forma individualizada, es decir, se realizan diferentes acciones para cada uno de ellos.
- Podemos tratar de manera individualizada tantos tipos de error como queramos.

## Manejando varios tipos de excepciones de manera múltiple:

*# Ejemplo 5*

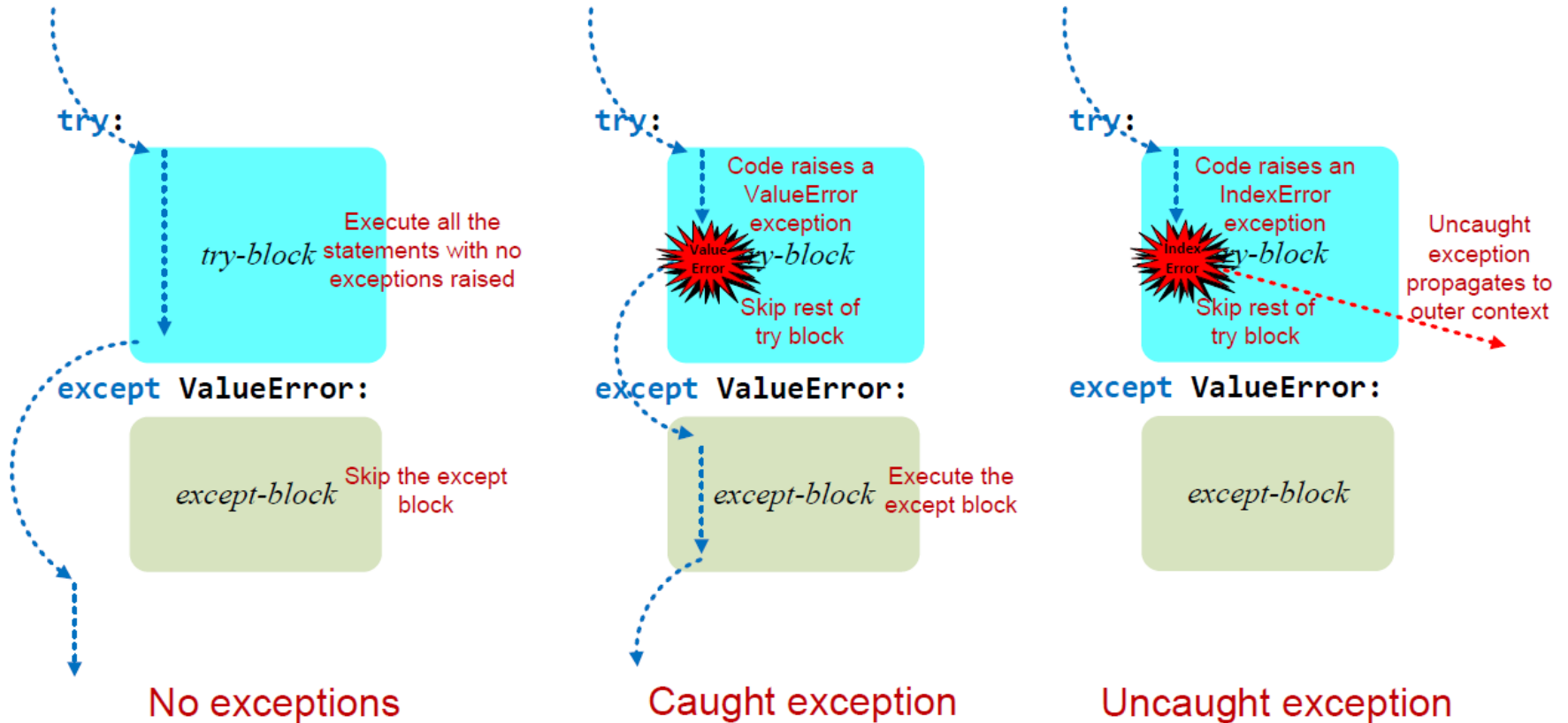
```
y = 'a'  
try:  
    x = 10.2 / y  
    print('El resultado es: ', x)  
except (ZeroDivisionError, TypeError):  
    print('El valor de la variable y es incorrecto')
```

- En este caso se contemplan dos tipos de excepciones que se gestionan de manera múltiple, realizando la misma acción ante cualquiera de las dos situaciones.
- *Dentro del paréntesis del except podemos especificar tantos tipos de error como queramos para tratarlos de forma conjunta.*



# ERRORES y EXCEPCIONES

## Posibles escenarios en el control de excepciones individualizadas



Podemos mezclar *except* con errores específicos y *except* genérico:

*# Ejemplo 6*

```
y = 'a'
try:
    x = 10.2 / y
    print('El resultado es: ', x)
except ZeroDivisionError:
    print('La variable y no puede ser 0')
except TypeError:
    print('La variable y no puede contener una cadena de texto')
except:
    print('Error inesperado')
```

- Podemos tratar ciertos tipos de excepciones de manera individualizada y para el resto poner un *except* genérico.

**Importante:** el *except* genérico ha de ser el último de los *except*.

# ERRORES y EXCEPCIONES

- Es conveniente poner en la parte *try*: únicamente las instrucciones que puedan ser conflictivas, para un mayor control y una mejor lectura y/o comprensión del código.
- De forma opcional, podemos incorporar en la parte *else*: las acciones a hacer en caso de no producirse algún error en las instrucciones conflictivas, es decir, la parte *else*: se ejecuta después de ejecutarse la parte *try*: cuando no hay errores.

*# Ejemplo 7*

```
y = 7
try:
    x = 10.2 / y
except ZeroDivisionError:
    print('La variable y no puede ser 0')
except TypeError:
    print('La variable y no puede contener una cadena de texto')
else:
    print('El resultado es: ', x)
```

En caso de no producirse ninguna excepción en la división, se ejecutaría la parte *else*:

También podemos usar la parte *finally*: para realizar alguna acción tanto si se produce una excepción como si no, es decir, siempre se va a ejecutar.

# Ejemplo 8

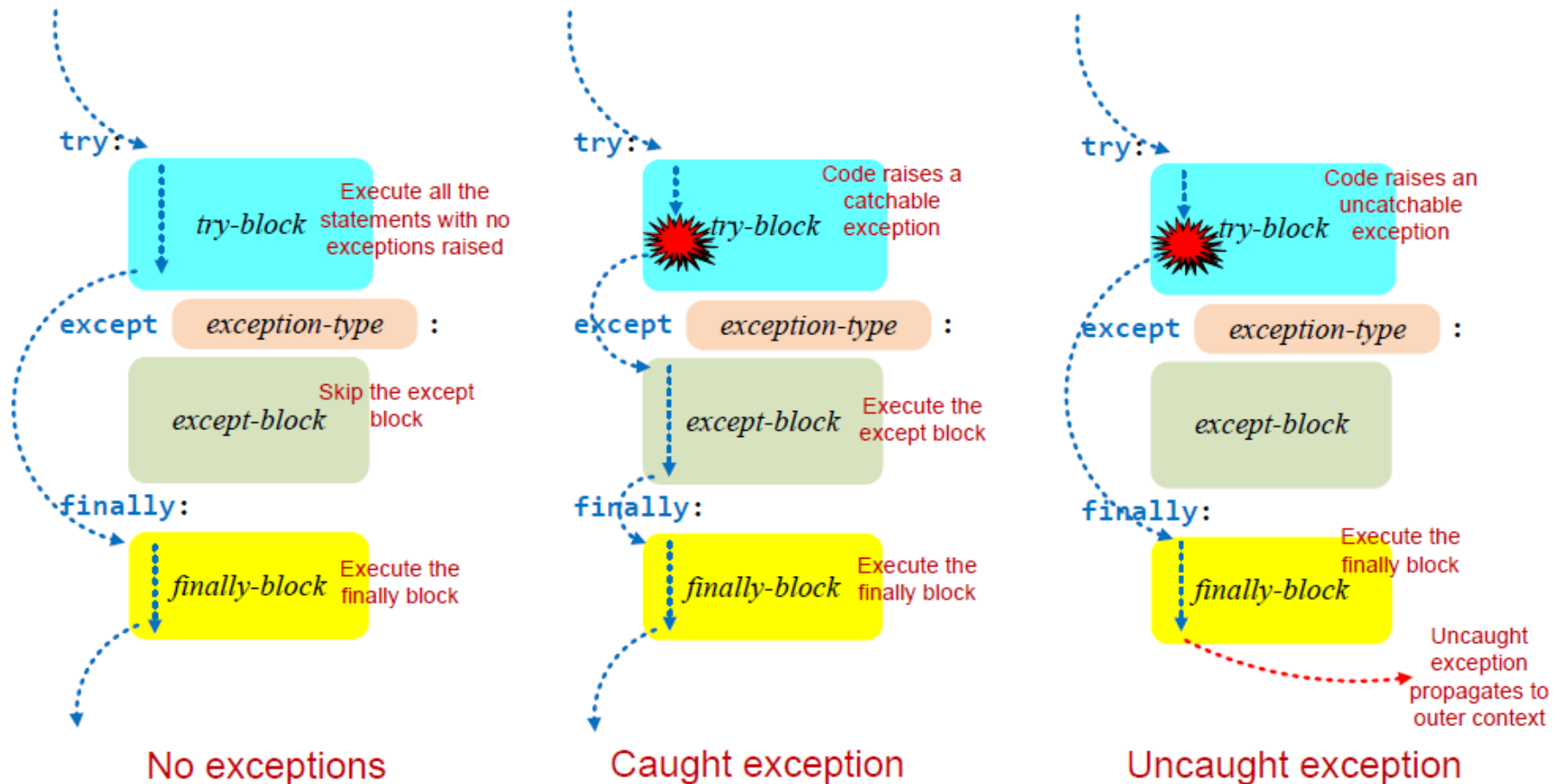
```
y = 5
try:
    x = 10.2 / y
except ZeroDivisionError:
    print('La variable y no puede ser 0')
except TypeError:
    print('La variable y no puede contener una cadena de texto')
else:
    print('El resultado es: ', x)
finally:
    print('Finalizando el programa')
```

- La parte *finally*: se va a ejecutar siempre

En caso de que se haya producido una excepción sin capturar, es decir, una excepción no contemplada en un *except*, se ejecutará la parte *finally* y luego se abortará el programa.

# ERRORES y EXCEPCIONES

Posibles escenarios en la ejecución de la cláusula *finally*:



Normalmente *finally*: se utiliza para liberar recursos externos tales como archivos, conexiones a Bases de Datos, conexiones de red, etc., sin importar si el uso del recurso fue exitoso o no.

# ERRORES y EXCEPCIONES

Podemos definir una excepción para establecer precondiciones personalizadas con *assert*.

*# Ejemplo 9*

```
y = -2.1
try:
    assert(y >= -2 and y <=2) # y tiene que estar en el rango [-2,2]
    x = 10.2 / y
except ZeroDivisionError:
    print('La variable y no puede ser 0')
except TypeError:
    print('La variable y no puede contener una cadena de texto')
except AssertionError: # excepcion para la parte assert
    print('La variable y debe estar en el rango [-2,2]')
else:
    print('El resultado es: ', x)
finally:
    print('Finalizando el programa')
```

Si no se cumple la condición establecida en *assert* se ejecutará la parte *except AssertionError*:

## Diferencia entre *TypeError* y *ValueError*.

- *TypeError*: ocurre cuando una función recibe un argumento con un tipo de datos erróneo.
- *ValueError*: ocurre cuando una función recibe un argumento con un tipo de datos adecuado para su ejecución pero con un valor inapropiado.

### # Ejemplo 10

```
print(len(1827.43))  
print(float("1827.43"))  
print(float("abc123"))
```

- El primer *print* arrojará la siguiente excepción: “*TypeError: object of type 'float' has no len()*”, debido a que la función *len()* no puede ejecutarse correctamente con un tipo de datos flotante.
- El segundo *print* se ejecutará correctamente e imprimirá el mensaje *1827.43* en pantalla. Esto es así porque la función *float()* puede convertir correctamente una cadena de texto si ésta contiene números exclusivamente.
- El tercer *print* arrojará la siguiente excepción: “*ValueError: could not convert string to float: 'abc123'*”, debido a que, a pesar de que la función *float()* puede recibir una cadena de texto para su conversión, en este caso el valor de la cadena pasada como argumento es tiene un valor erróneo.

- Ejercicios.

1. Calcular el mínimo de una lista de valores utilizando la función integrada en la librería estándar  $\min(x)$ . Tener en cuenta que todos los elementos de la lista sean valores numéricos.
2. Calcular el volumen de una esfera ( $v = 4/3 \pi r^3$ ). El radio se introducirá por teclado. Tener en cuenta el tipo de datos introducido.
3. Crear una lista de 10 números devolver la suma de los n primeros. El parámetro n lo introducirá el usuario por teclado. Tener en cuenta el tipo de datos introducido y forzar como condición que n pertenezca al rango [1,10].



# ERRORES y EXCEPCIONES

1. Calcular el mínimo de una lista de valores utilizando la función integrada en la librería estándar *min(x)*. Tener en cuenta que todos los elementos de la lista sean valores numéricos.

```
# Ejercicio 1
```

```
"""
```

```
Ejercicio 1: Imprime el mínimo de una lista de elementos.
```

```
"""
```

```
lista = [3, 7, 4, 'a', 1, 2]
```

```
try:
```

```
    minimo = min(lista)
```

```
except TypeError:
```

```
    print("Error - Todos los elementos de la lista han de ser numeros")
```

```
else:
```

```
    print("El minimo de la lista es: ", minimo)
```

2. Calcular el volumen de una esfera ( $v = 4/3 \pi r^3$ ). El radio se introducirá por teclado. Tener en cuenta que el valor introducido para r sea un valor positivo.

```
# Ejercicio 2

"""
Calcula el volumen de una esfera (v = 4/3 π r3). El radio se introducirá por teclado.
"""

pi = 3.1416

try:
    r = float(input('Introduzca el radio (en metros): '))
    assert (r > 0)
except ValueError:
    print('El valor introducido para el radio es incorrecto')
except AssertionError:
    print('El valor del radio tiene que ser positivo')
else:
    v = 4 / 3 * pi * r ** 3
    print ('El volumen de una esfera de radio', r, 'metros es:', v, 'metros cubicos')
```

3. Crear una lista de 10 números y devolver la suma de los n primeros. El parámetro n lo introducirá el usuario por teclado. Tener en cuenta el tipo de datos introducido y forzar como condición que n pertenezca al rango [1,10].

```
# Ejercicio 3
```

```
"""
```

```
    Crea una lista de 10 números y devuelve la suma de los n primeros. El parámetro n se introducirá por teclado.
```

```
"""
```

```
suma = 0
```

```
lista = range(1,11)
```

```
try:
```

```
    n = int(input('introduzca el numero de elementos a sumar: '))
```

```
    assert(n in range(1,11))
```

```
except (ValueError, AssertionError):
```

```
    print('Indique un numero de elementos en el rango [1,', len(lista), ']')
```

```
else:
```

```
    suma = sum(lista[:n])
```

```
    print('La suma de los', n, 'primeros numeros es:', suma)
```