Manejo de errores

Manejo de errores

- A veces puedes prever que tu código puede provocar un error ya sea humano (la entrada de datos no es correcta) o por el propio algoritmo elegido (nos salimos del rango de una lista, dividimos entre 0, etc.).
- En Python existe una herramienta para manejar los errores y prevenir el fallo del programa y su abrupta parada.

try, except

- Cuando creamos que en un trozo de código puede haber un error lo metemos dentro de un bloque try.

```
try:
    a = 3/0
except:
    print(‘¡Error!’)
```

- except se encarga de manejar el error cuando se produce.

Tipos de error

- Cuando queremos manejar solo un tipo de error podemos indicarlo en except.

```
try:
    num1 = int(input())
    num2 = int(input())
    print(num1/num2)
except ValueError:
    print('¡Error, introduce un número!')
except ZeroDivisionError:
    print('¡No puedes dividir entre 0!')
```

- Existen muchos tipos de error predefinidos.

else

- Podemos indicar que se realice una acción si no se lanza ningún error.

```
try:
    num1 = int(input())
    num2 = int(input())
    out = num1/num2
except ValueError:
    print('¡Error, introduce un número!')
except ZeroDivisionError:
    print('¡No puedes dividir entre 0!')
else:
    print('Resultado:', out)
```

finally

- También podemos realizar una acción ‘de limpieza’, que se ejecutará se lance o no un error.

```
try:
    num1 = int(input())
    num2 = int(input())
    out = num1/num2
except ValueError:
    print(‘¡Error, introduce un número!’)
except ZeroDivisionError:
    print(‘¡No puedes dividir entre 0!’)
else:
    print(‘Resultado:’, out)
finally:
    print(‘Fin del programa”)
```

finally

- También podemos realizar una acción ‘de limpieza’, que se ejecutará se lance o no un error.

```
try:
    num1 = int(input())
    num2 = int(input())
    out = num1/num2
except ValueError:
    print(‘¡Error, introduce un número!’)
except ZeroDivisionError:
    print(‘¡No puedes dividir entre 0!’)
else:
    print(‘Resultado:’, out)
finally:
    print(‘Fin del programa”)
```

Lanzar errores

- Podemos lanzar errores a voluntad:

```
try:
  if edad < 18:
    raise ValueError
except ValueError:
  print(‘Es demasiado Joven!”)
```

- Además, podemos lanzar mensajes personalizados:

```
try:
  if edad < 18:
    raise ValueError(‘Es demasiado Joven!’)
except ValueError as err:
  print(err) → Es demasiado Joven!
```

Lanzar errores

- Otra opción es usar assert para comprobar condiciones. Assert lanza una excepción si no se cumple la condición:

```
def area_rectangulo(a: float, b: float):
    assert isinstance(a, (float, int))
    assert isinstance(b, (float, int))
    assert a > 0 and b > 0
    return a*b
```

- Si no se cumplen las condiciones se lanzará AssertionError.

Errores personalizados

- Podemos crear errores personalizados.

```
class GranError(Exception):
    pass

raise GranError('Qué gran error!!!')
```

- La clase Exception sirve de base para todos los errores.
- La personalización de errores puede ser mucho más complicada.

Cazar errores.

- Los errores no conocen de ámbito:

```
def foo1(a):
    assert a > 0
    (...)
def foo2(a, b):
    assert a * b < 100
    (...)
try:
    foo1(x)
    foo2(x,y)
except AssertionError:
    print('Error en las entradas')
```