TEMA IV

DESARROLLO DE APLICACIONES DE PROPÓSITO GENERAL

"*Siempre parece imposible,*
hasta que se hace"
Nelson Mandela

2. MANEJO DE EXCEPCIONES

La sintaxis para el manejo de excepciones en python

```
try:
    ……..
    ……..
except TipoExcepcion:
    ……..
except TipoExcepción:
else:
```

Se ejecuta el bloque de instrucciones de try.

Si no se produce la excepción, el bloque de except no se ejecuta y continúa la ejecución secuencia.

Si se produce una excepción, el resto del bloque try no se ejecuta,

Si la excepción que se ha produce corresponde con la indicada en except se salta a ejecutar el bloque de instrucciones except.

Si la excepción producida no se corresponde con las indicadas en except se pasa al bloque else si lo hemos codificado. Si finalmente no hay un manejador nos dará un error y el programa terminará.

Un bloque except puede manejar varios tipos de excepciones:

```
... except (RuntimeError, TypeError, NameError):
```

Si quiero controlar varios tipos de excepciones puedo poner varios bloques except. Teniendo en cuenta que en el último, si quiero no indico el tipo de excepción:

```
>>> try:
...   print (10/int(cad))
... except ValueError:
...   print("No se puede converir a entero")
... except ZeroDivisionError:
...   print("No se puede divir por cero")
... except:
...   print("Otro error")
```

Se puede utilizar también la clausula else:

```
>>> try:
...   print (10/int(cad))
... except ValueError:
...   print("No se puede converir a entero")
... except ZeroDivisionError:
...   print("No se puede divir por cero")
... else:
...   print("Otro error")
```

Por último indicar que podemos indicar una clausula finally para indicar un bloque de instrucciones que siempre se debe ejecutar, independientemente de la excepción se haya producido o no.

```
>>> try:
...     result = x / y
... except ZeroDivisionError:
...     print("División por cero!")
... else:
...     print("El resultado es", result)
... finally:
...     print("Terminamos el programa")
```

Disponemos de funciones que nos permiten obtener información sobre las excepciones:

```
>>> cad = "a"
>>> try:
...   i = int(cad)
... except ValueError as error:
...   print(type(error))
...   print(error.args)
...   print(error)
...
<class 'ValueError'>
("invalid literal for int() with base 10: 'a'",)
invalid literal for int() with base 10: '
```

```
num1 = '5'
num2 = 0

try:
   x = num1 / num2
except ZeroDivisionError as err:
   print('No puedes dividir un número entre 0')
except TypeError as err:
   print('Los dos valores deben ser númericos')
except Exception as err:
   params = {
      'num1': num1,
      'num2': num2
   }
   # Utilizamos la función type para saber que tipo
de error ha ocurrido y name para obtener su nombre
   save_log(type(err).__name__, params=params)
   print('Error desconocido')
finally:
   print('Fin del proceso')
```

Para lanzar una excepción disponemos del método raise

Si construimos una función donde se maneje una excepción podemos hacer que la excepción se envía a la función desde la que la hemos llamado. Para ello utilizamos la instrucción raise. Veamos algunos ejemplos:

```
def dividir(x,y):
        try:
                return x/y
        except ZeroDivisionError:
                raise
```

También podemos propagar una excepción en concreto:

```
def nivel(numero):
        if numero<0:
                raise ValueError("El número debe ser
positivo:"+str(numero))
        else:
                return numero
```

Podemos crear nuestras propias excepciones y lanzarlas. Para ello creamos una clase que hereda la clase Exception

```
class CustomException(Exception):
```

```
try:
   raise CustomException('Hola', 'esto es un error')
except CustomException as err:
   print(err)
```

Podemos pasar tantos argumentos como queramos.

Podemos añadir atributos de objeto a nuestra clase

```
class MyHttpException(Exception):
   def __init__(self, code=500, message='Internal
server error'):
      self.code = code
      self.message = message

try:
   raise MyHttpException
except MyHttpException as err:
   print(err.code)
   print(err.message)

try:
   raise MyHttpException(code=404, message='Not
found')
except MyHttpException as err:
   print(err.code)
   print(err.message)
```