TEXT CLASS REVIEW

TEMAS A TRATAR EN EL CUE

- Excepciones definidas por el usuario.
- Acciones de limpieza con finally.
- Acciones de limpieza predefinidas.

EXCEPCIONES DEFINIDAS POR EL USUARIO

Los programas pueden nombrar sus propias excepciones creando una nueva clase excepción. Las excepciones, típicamente, deberán derivar de la clase **Exception**, directa o indirectamente.

La mayoría de las excepciones se definen con nombres acabados en «Error», de manera similar a la nomenclatura de las excepciones estándar.

Muchos módulos estándar definen sus propias excepciones para reportar errores que pueden ocurrir en funciones propias.

```
class MyError(Exception):
     def __init__(self, value):
        self.value = value
     def __str__(self):
        return(repr(self.value))
try:
    raise(MyError(3*2))
except MyError as error:
    print('Ocurrió una nueva excepción: ',error.value)
```

Salida:

```
('Ocurrió una nueva excepción:', 6)
```

ACCIONES DE LIMPIEZA CON FINALLY

La declaración ***try*** tiene otra cláusula opcional, cuyo propósito es definir acciones de limpieza que serán ejecutadas bajo ciertas circunstancias. Por ejemplo:

```
>>>
>>> try:
```

```
...     raise KeyboardInterrupt
... finally:
...     print('Goodbye, world!')
...
Goodbye, world!
KeyboardInterrupt
Traceback (most recent call last):
  File "<stdin>", line 2, in <module>
```

Si una cláusula ***finally*** está presente, el bloque ***finally*** se ejecutará al final antes de que todo el bloque ***try*** se complete. La cláusula ***finally*** se ejecuta independientemente de que la cláusula ***try*** produzca o no una excepción. Los siguientes puntos explican casos más complejos en los que se produce una excepción:

- Si ocurre una excepción durante la ejecución de la cláusula ***try***, ésta podría ser gestionada por una cláusula ***except***. Si la excepción no es gestionada por una cláusula ***except***, es relanzada después de que se ejecute el bloque de la cláusula ***finally***.
- Podría aparecer una excepción durante la ejecución de una cláusula ***except*** o ***else***. De nuevo, la excepción será relanzada después de que el bloque de la cláusula ***finally*** se ejecuta.
- Si la cláusula ***finally*** ejecuta una declaración ***break***, ***continue*** o ***return***, las excepciones no se vuelven a lanzar.
- Si el bloque ***try*** llega a una sentencia ***break***, ***continue*** o ***return***, la cláusula ***finally*** se ejecutará justo antes de la ejecución de dicha sentencia.
- Si una cláusula ***finally*** incluye una sentencia ***return***, el valor retornado será el de la cláusula ***finally***, no la de la sentencia ***return*** de la cláusula ***try***.

Por ejemplo:

```
>>>
>>> def bool_return():
...     try:
...         return True
...     finally:
...         return False
...
>>> bool_return()
False
```

Un ejemplo más complicado:

```
>>>
>>> def divide(x, y):
...     try:
...         result = x / y
...     except ZeroDivisionError:
...         print("division by zero!")
...     else:
...         print("result is", result)
...     finally:
...         print("executing finally clause")
...
>>> divide(2, 1)
result is 2.0
executing finally clause
>>> divide(2, 0)
division by zero!
executing finally clause
>>> divide("2", "1")
executing finally clause
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
  File "<stdin>", line 3, in divide
TypeError: unsupported operand type(s) for /: 'str' and 'str'
```

Como se puede observar, la cláusula ***finally*** siempre se ejecuta. La excepción ***TypeError*** lanzada al dividir dos cadenas de texto no es gestionado por la cláusula ***except***, y por lo tanto es relanzada luego de que se ejecuta la cláusula ***finally***.

En aplicaciones reales, la cláusula ***finally*** es útil para liberar recursos externos (como archivos o conexiones de red), sin importar si el uso del recurso fue exitoso.

ACCIONES DE LIMPIEZA PREDEFINIDAS

Algunos objetos definen acciones de limpieza estándar para llevar a cabo cuando el objeto ya no es necesario, independientemente de que las operaciones sobre el objeto hayan sido exitosas o no. Véase el siguiente ejemplo, que intenta abrir un archivo e imprimir su contenido en la pantalla.

```
for line in open("myfile.txt"):
    print(line, end="")
```

El problema con este código es que deja el archivo abierto por un periodo de tiempo indeterminado luego de que esta parte termine de ejecutarse. Esto no es un problema en scripts simples, pero puede serlo en aplicaciones más grandes. La declaración ***with*** permite que los objetos como archivos sean usados de una forma que asegure que siempre se los libera rápido, y en forma correcta:

```
1 with open("myfile.txt") as f:
2     for line in f:
3         print(line, end="")
```

Una vez que la declaración se ejecuta, el fichero **f** siempre se cierra, incluso si aparece algún error durante el procesado de las líneas. Los objetos que, como los ficheros, posean acciones predefinidas de limpieza, lo indicarán en su documentación.