HINTS

¿CÓMO FUNCIONA LA SENTENCIA TRY?

Primero, se ejecuta la cláusula ***try*** (la(s) línea(s) entre las palabras reservadas ***try*** y la ***except***).

Si no ocurre ninguna excepción, la cláusula ***except*** se omite, y la ejecución de la cláusula ***try*** finaliza.

Si ocurre una excepción durante la ejecución de la cláusula ***try***, se omite el resto de esta. Luego, si su tipo coincide con la excepción nombrada después de la palabra clave ***except***, se ejecuta la cláusula ***except***, y luego la ejecución continúa después del bloque ***try***/***except***.

Si ocurre una excepción que no coincide con la indicada en la cláusula ***except***, se pasa a los ***try*** más externos; si no se encuentra un gestor, se genera una ***unhandled exception*** (excepción no gestionada).

Una declaración ***try*** puede tener más de una cláusula ***except***, para especificar gestores de diferentes excepciones. Como máximo, se ejecutará un gestor. Los gestores solo manejan las excepciones que ocurren en la cláusula ***try*** correspondiente, no en otros gestores de la misma declaración ***try***. Una cláusula ***except*** puede nombrar múltiples excepciones como una tupla entre paréntesis. Por ejemplo:

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

Una clase en una cláusula ***except*** es compatible con una excepción si es de la misma clase, o de una clase derivada de la misma (pero no de la otra manera — una cláusula ***except*** listando una clase derivada no es compatible con una clase base). Por ejemplo, el siguiente código imprimirá B, C y D, en ese orden:

```
class B(Exception):
    pass

class C(B):
    pass

class D(C):
    pass

for cls in [B, C, D]:
    try:
        raise cls()
    except D:
        print("D")
```

```
    except C:
        print("C")
    except B:
        print("B")
```

Nótese que si las cláusulas ***except*** estuvieran invertidas (con ***except*** B primero), habría impreso B, B, B — se usa la primera cláusula ***except*** coincidente.

Todas las excepciones heredan de **BaseException**, por lo que se puede utilizar como comodín. Use esto con extrema precaución, ya que es fácil enmascarar un error de programación real de esta manera. También se puede usar para imprimir un mensaje de error, y luego volver a generar la excepción (permitiendo que la función que llama también maneje la excepción):

```
import sys
try:
    f = open('myfile.txt')
    s = f.readline()
    i = int(s.strip())
except OSError as err:
    print("OS error: {0}".format(err))
except ValueError:
    print("Could not convert data to an integer.")
except BaseException as err:
    print(f"Unexpected {err=}, {type(err)=}")
    raise
```

Alternativamente, la última cláusula ***except*** puede omitir el(los) nombre(s) de excepción, sin embargo, el valor de la excepción debe recuperarse de `sys.exc_info()[1]`.

La declaración ***try*** … ***except*** tiene una cláusula `else` opcional, que, cuando está presente, debe seguir todas las cláusulas ***except***. Es útil para el código que debe ejecutarse si la cláusula ***try*** no lanza una excepción. Por ejemplo:

```
for arg in sys.argv[1:]:
    try:
        f = open(arg, 'r')
    except OSError:
        print('cannot open', arg)
    else:
        print(arg, 'has', len(f.readlines()), 'lines')
        f.close()
```

El uso de la cláusula `else` es mejor que agregar código adicional en la cláusula ***try***, pues evita capturar accidentalmente una excepción que no fue generada por el código que está protegido por la declaración ***try*** … ***except***.

Cuando ocurre una excepción, puede tener un valor asociado, también conocido como el

argumento de la excepción. La presencia y el tipo de argumento depende del tipo de excepción.

La cláusula ***except*** puede especificar una variable después del nombre de la excepción. La variable está vinculada a una instancia de excepción con los argumentos almacenados en `instance.args`. Por conveniencia, la instancia de excepción define `__str__()` para que los argumentos se puedan imprimir directamente sin tener que hacer referencia a `.args`. También se puede crear una instancia de una excepción antes de generarla, y agregarle los atributos que desee.

```
>>>
>>> try:
...     raise Exception('spam', 'eggs')
... except Exception as inst:
...     print(type(inst))    # the exception instance
...     print(inst.args)     # arguments stored in .args
...     print(inst)          # __str__ allows args to be printed
directly,
...                          # but may be overridden in exception
subclasses
...     x, y = inst.args     # unpack args
...     print('x =', x)
...     print('y =', y)
...
<class 'Exception'>
('spam', 'eggs')
('spam', 'eggs')
x = spam
y = eggs
```

Si una excepción tiene argumentos, estos se imprimen como en la parte final (el "detalle") del mensaje para las excepciones no gestionadas (" ***Unhandled exception***").

Los gestores de excepciones no solo las gestionan si ocurren inmediatamente en la cláusula ***try***, sino también si ocurren dentro de funciones que son llamadas (incluso indirectamente) en la cláusula ***try***. Por ejemplo:

```
>>>
>>> def this_fails():
...     x = 1/0
...
>>> try:
...     this_fails()
... except ZeroDivisionError as err:
...     print('Handling run-time error:', err)
...
Handling run-time error: division by zero
```

LA SENTENCIA RAISE

La declaración ***raise*** permite al programador forzar a que ocurra una excepción específica. Por ejemplo:

```
>>>
>>> raise NameError('HiThere')
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
NameError: HiThere
```

El único argumento de ***raise*** indica la excepción a generarse. Tiene que ser o una instancia de excepción, o una clase de excepción (una clase que hereda de **Exception**). Si se pasa una clase de excepción, la misma será instanciada implícitamente llamando a su constructor sin argumentos:

```
raise ValueError  # shorthand for 'raise ValueError()'
```

Si es necesario determinar si una excepción fue lanzada, pero sin intención de gestionarla, una versión simplificada de la instrucción ***raise*** te permite relanzarla:

```
>>>
>>> try:
...     raise NameError('HiThere')
... except NameError:
...     print('An exception flew by!')
...     raise
...
An exception flew by!
Traceback (most recent call last):
  File "<stdin>", line 2, in <module>
NameError: HiThere
```

La declaración ***raise*** permite una palabra clave opcional ***from*** que habilita el encadenamiento de excepciones. Por ejemplo:

```
# exc must be exception instance or None.
raise RuntimeError from exc
```

Esto puede resultar útil cuando está transformando excepciones. Por ejemplo:

```
>>>
>>> def func():
...     raise ConnectionError
...
>>> try:
...     func()
... except ConnectionError as exc:
...     raise RuntimeError('Failed to open database') from exc
...
Traceback (most recent call last):
  File "<stdin>", line 2, in <module>
  File "<stdin>", line 2, in func
ConnectionError

The above exception was the direct cause of the following
exception:

Traceback (most recent call last):
  File "<stdin>", line 4, in <module>
RuntimeError: Failed to open database
```