8.4. The `try`statement

The `try` statement specifies exception handlers and/or cleanup code for a group of statements:

```
try_stmt  ::=  try1_stmt | try2_stmt
try1_stmt ::=  "try" ":" suite
               ("except" [expression ["as" identifier]] ":" suite)+
               ["else" ":" suite]
               ["finally" ":" suite]
try2_stmt ::=  "try" ":" suite
               "finally" ":" suite
```

The `except` clause(s) specify one or more exception handlers. When no exception occurs in the `try`clause, no exception handler is executed. When an exception occurs in the `try` suite, a search for an exception handler is started. This search inspects the except clauses in turn until one is found that matches the exception. An expression-less except clause, if present, must be last; it matches any exception. For an except clause with an expression, that expression is evaluated, and the clause matches the exception if the resulting object is “compatible” with the exception. An object is compatible with an exception if it is the class or a base class of the exception object or a tuple containing an item compatible with the exception.

If no except clause matches the exception, the search for an exception handler continues in the surrounding code and on the invocation stack. [1]

If the evaluation of an expression in the header of an except clause raises an exception, the original search for a handler is canceled and a search starts for the new exception in the surrounding code and on the call stack (it is treated as if the entire `try` statement raised the exception).

When a matching except clause is found, the exception is assigned to the target specified after the `as`keyword in that except clause, if present, and the except clause’s suite is executed. All except clauses must have an executable block. When the end of this block is reached, execution continues normally after the entire try statement. (This means that if two nested handlers exist for the same exception, and the exception occurs in the try clause of the inner handler, the outer handler will not handle the exception.)

When an exception has been assigned using `as target`, it is cleared at the end of the except clause. This is as if

```
except E as N:
    foo
```

was translated to

```
except E as N:
    try:
        foo
    finally:
        del N
```

This means the exception must be assigned to a different name to be able to refer to it after the except clause. Exceptions are cleared because with the traceback attached to them, they form a reference cycle with the stack frame, keeping all locals in that frame alive until the next garbage collection occurs.

Before an except clause's suite is executed, details about the exception are stored in the `sys`module and can be accessed via`sys.exc_info()`. `sys.exc_info()`returns a 3-tuple consisting of the exception class, the exception instance and a traceback object (see section The standard type hierarchy) identifying the point in the program where the exception occurred. `sys.exc_info()` values are restored to their previous values (before the call) when returning from a function that handled an exception.

The optional `else` clause is executed if and when control flows off the end of the `try` clause. [2]Exceptions in the `else` clause are not handled by the preceding`except` clauses.

If `finally` is present, it specifies a 'cleanup' handler. The `try` clause is executed, including any `except` and`else` clauses. If an exception occurs in any of the clauses and is not handled, the exception is temporarily saved. The `finally`clause is executed. If there is a saved exception it is re-raised at the end of the `finally` clause. If the`finally` clause raises another exception, the saved exception is set as the context of the new exception. If the `finally` clause executes a `return` or `break`statement, the saved exception is discarded:

```
>>> def f():
...     try:
...         1/0
...     finally:
...         return 42
...
>>> f()
42
```

The exception information is not available to the program during execution of the `finally` clause.

When a `return`, `break` or`continue` statement is executed in the `try` suite of a `try`...`finally`statement, the `finally` clause is also executed 'on the way out.' A`continue` statement is illegal in the`finally` clause. (The reason is a problem with the current implementation — this restriction may be lifted in the future).

The return value of a function is determined by the last `return`statement executed. Since the`finally` clause always executes, a`return` statement executed in the`finally` clause will always be the last one executed:

```
>>> def foo():
...     try:
...         return 'try'
...     finally:
...         return 'finally'
...
>>> foo()
'finally'
```

Additional information on exceptions can be found in sectionExceptions, and information on using the `raise` statement to generate exceptions may be found in section The raise statement.