8.5 The `with` statement

The *with* statement is used to wrap the execution of a block with methods defined by a context manager (see section *With Statement Context Managers*). This allows common *try*…*except*…*finally* usage patterns to be encapsulated for convenient reuse.

```
with_stmt   ::=   "with" with_item (","  with_item)* ":" suite
with_item   ::=   expression ["as" target]
```

The execution of the *with* statement with one "item" proceeds as follows:

1. The context expression (the expression given in the *with_item*) is evaluated to obtain a context manager.
2. The context manager's *__enter__()* is loaded for later use.
3. The context manager's *__exit__()* is loaded for later use.
4. The context manager's *__enter__()* method is invoked.
5. If a target was included in the *with* statement, the return value from *__enter__()* is assigned to it.

   Note: The *with* statement guarantees that if the *__enter__()* method returns without an error, then *__exit__()* will always be called. Thus, if an error occurs during the assignment to the target list, it will be treated the same as an error occurring within the suite would be. See step 6 below.

6. The suite is executed.
7. The context manager's *__exit__()* method is invoked. If an exception caused the suite to be exited, its type, value, and traceback are passed as arguments to *__exit__()*. Otherwise, three `None` arguments are supplied.

   If the suite was exited due to an exception, and the return value from the *__exit__()* method was false, the exception is reraised. If the return value was true, the exception is suppressed, and execution continues with the statement following the *with* statement.

   If the suite was exited for any reason other than an exception, the return value from *__exit__()* is ignored, and execution proceeds at the normal location for the kind of exit that was taken.

The following code:

```
with EXPRESSION as TARGET:
    SUITE
```

is semantically equivalent to:

```
manager = (EXPRESSION)
enter = type(manager).__enter__
exit = type(manager).__exit__
value = enter(manager)
hit_except = False

try:
    TARGET = value
    SUITE
except:
    hit_except = True
    if not exit(manager, *sys.exc_info()):
        raise
finally:
    if not hit_except:
        exit(manager, None, None, None)
```