6.2.9. Yield expressions

```
yield_atom       ::=  "(" yield_expression ")"
yield_expression ::=  "yield" [expression_list | "from" expression]
```

The yield expression is only used when defining a generator function and thus can only be used in the body of a function definition. Using a yield expression in a function's body causes that function to be a generator.

When a generator function is called, it returns an iterator known as a generator. That generator then controls the execution of the generator function. The execution starts when one of the generator's methods is called. At that time, the execution proceeds to the first yield expression, where it is suspended again, returning the value of`expression_list` to the generator's caller. By suspended, we mean that all local state is retained, including the current bindings of local variables, the instruction pointer, the internal evaluation stack, and the state of any exception handling. When the execution is resumed by calling one of the generator's methods, the function can proceed exactly as if the yield expression were just another external call. The value of the yield expression after resuming depends on the method which resumed the execution. If`__next__()` is used (typically via either a `for` or the `next()` builtin) then the result is `None`. Otherwise, if`send()` is used, then the result will be the value passed in to that method.

All of this makes generator functions quite similar to coroutines; they yield multiple times, they have more than one entry point and their execution can be suspended. The only difference is that a generator function cannot control where the execution should continue after it yields; the control is always transferred to the generator's caller.

Yield expressions are allowed anywhere in a `try` construct. If the generator is not resumed before it is finalized (by reaching a zero reference count or by being garbage collected), the generator-iterator's `close()` method will be called, allowing any pending`finally` clauses to execute.

When `yield from <expr>` is used, it treats the supplied expression as a subiterator. All values produced by that subiterator are passed directly to the caller of the current generator's methods. Any values passed in with `send()` and any exceptions passed in with `throw()`are passed to the underlying iterator if it has the appropriate methods. If this is not the case, then `send()` will raise `AttributeError` or`TypeError`, while `throw()` will just raise the passed in exception immediately.

When the underlying iterator is complete, the `value` attribute of the raised `StopIteration` instance becomes the value of the yield expression. It can be either set explicitly when raising`StopIteration`, or automatically when the sub-iterator is a generator (by returning a value from the sub-generator).

*Changed in version 3.3:*Added `yield from<expr>` to delegate control flow to a subiterator.

The parentheses may be omitted when the yield expression is the sole expression on the right hand side of an assignment statement.