

29.10. `__future__` — Future statement definitions

Source code: [Lib/__future__.py](#)

`__future__` is a real module, and serves three purposes:

- To avoid confusing existing tools that analyze import statements and expect to find the modules they're importing.
- To ensure that [future statements](#) run under releases prior to 2.1 at least yield runtime exceptions (the import of `__future__` will fail, because there was no module of that name prior to 2.1).
- To document when incompatible changes were introduced, and when they will be — or were — made mandatory. This is a form of executable documentation, and can be inspected programmatically via importing `__future__` and examining its contents.

Each statement in `__future__.py` is of the form:

```
FeatureName = _Feature(OptionalRelease, MandatoryRelease,  
                        CompilerFlag)
```

where, normally, *OptionalRelease* is less than *MandatoryRelease*, and both are 5-tuples of the same form as [sys.version_info](#):

```
(PY_MAJOR_VERSION, # the 2 in 2.1.0a3; an int  
PY_MINOR_VERSION, # the 1; an int  
PY_MICRO_VERSION, # the 0; an int  
PY_RELEASE_LEVEL, # "alpha", "beta", "candidate" or "final"; string  
PY_RELEASE_SERIAL # the 3; an int  
)
```

OptionalRelease records the first release in which the feature was accepted.

In the case of a *MandatoryRelease* that has not yet occurred, *MandatoryRelease* predicts the release in which the feature will become part of the language.

Else *MandatoryRelease* records when the feature became part of the language; in releases at or after that, modules no longer need a future statement to use the feature in question, but may continue to use such imports.

MandatoryRelease may also be `None`, meaning that a planned feature got dropped.

Instances of class `_Feature` have two corresponding methods, `getOptionalRelease()` and `getMandatoryRelease()`.

CompilerFlag is the (bitfield) flag that should be passed in the fourth argument to the built-in function `compile()` to enable the feature in dynamically compiled code. This flag is stored in the `compiler_flag` attribute on `_Feature` instances.

No feature description will ever be deleted from `__future__`. Since its introduction in Python 2.1 the following features have found their way into the language using this mechanism:

feature	optional in	mandatory in	effect
nested_scopes	2.1.0b1	2.2	PEP 227 : <i>Statically Nested Scopes</i>
generators	2.2.0a1	2.3	PEP 255 : <i>Simple Generators</i>
division	2.2.0a2	3.0	PEP 238 : <i>Changing the Division Operator</i>
absolute_import	2.5.0a1	3.0	PEP 328 : <i>Imports: Multi-Line and Absolute/Relative</i>
with_statement	2.5.0a1	2.6	PEP 343 : <i>The “with” Statement</i>
print_function	2.6.0a2	3.0	PEP 3105 : <i>Make print a function</i>
unicode_literals	2.6.0a2	3.0	PEP 3112 : <i>Bytes literals in Python 3000</i>
generator_stop	3.5.0b1	3.7	PEP 479 : <i>StopIteration handling inside generators</i>

See also:

Future statements

How the compiler treats future imports.