31.6. `runpy` — Locating and executing Python modules¶

New in version 2.5.

The `runpy` module is used to locate and run Python modules without importing them first. Its main use is to implement the *-m* command line switch that allows scripts to be located using the Python module namespace rather than the filesystem.

When executed as a script, the module effectively operates as follows:

```
del sys.argv[0]  # Remove the runpy module from the arguments
run_module(sys.argv[0], run_name="__main__", alter_sys=True)
```

The `runpy` module provides a single function:

`runpy.run_module`(*mod_name*[, *init_globals*][, *run_name*][, *alter_sys*])¶

Execute the code of the specified module and return the resulting module globals dictionary. The module's code is first located using the standard import mechanism (refer to PEP 302 for details) and then executed in a fresh module namespace.

The optional dictionary argument *init_globals* may be used to pre-populate the globals dictionary before the code is executed. The supplied dictionary will not be modified. If any of the special global variables below are defined in the supplied dictionary, those definitions are overridden by the `run_module` function.

The special global variables `__name__`, `__file__`, `__loader__` and `__builtins__` are set in the globals dictionary before the module code is executed.

`__name__` is set to *run_name* if this optional argument is supplied, and the *mod_name* argument otherwise.

`__loader__` is set to the PEP 302 module loader used to retrieve the code for the module (This loader may be a wrapper around the standard import mechanism).

`__file__` is set to the name provided by the module loader. If the loader does not make filename information available, this variable is set to `None`.

`__builtins__` is automatically initialised with a reference to the top level namespace of the `__builtin__` module.

If the argument *alter_sys* is supplied and evaluates to `True`, then `sys.argv[0]` is updated with the value of `__file__` and `sys.modules[__name__]` is updated with a temporary module object for the module being executed. Both `sys.argv[0]` and `sys.modules[__name__]` are restored to their original values before the function returns.

Note that this manipulation of `sys` is not thread-safe. Other threads may see the partially initialised module, as well as the altered list of arguments. It is recommended that the `sys` module be left alone when invoking this function from threaded code.

See also

PEP 338 - Executing modules as scripts
PEP written and implemented by Nick Coghlan.

Previous topic

31.5. `modulefinder` — Find modules used by a script

Next topic

32. Python Language Services

This Page

- Show Source

Navigation

- index