:mod:`runpy` --- Locating and executing Python modules

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\library\(cpython-main)(Doc)(library)runpy.rst, line 1); *backlink*

Unknown interpreted text role "mod".

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\library\(cpython-main)(Doc)(library)runpy.rst, line 4)

Unknown directive type "module".

```
.. module:: runpy
   :synopsis: Locate and run Python modules without importing them first.
```

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\library\(cpython-main)(Doc)(library)runpy.rst, line 7)

Unknown directive type "moduleauthor".

```
.. moduleauthor:: Nick Coghlan <ncoghlan@gmail.com>
```

Source code: :source:`Lib/runpy.py`

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\library\(cpython-main)(Doc)(library)runpy.rst, line 9); *backlink*

Unknown interpreted text role "source".

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

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\library\(cpython-main)(Doc)(library)runpy.rst, line 13); *backlink*

Unknown interpreted text role "mod".

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\library\(cpython-main)(Doc)(library)runpy.rst, line 13); *backlink*

Unknown interpreted text role "option".

Note that this is *not* a sandbox module - all code is executed in the current process, and any side effects (such as cached imports of other modules) will remain in place after the functions have returned.

Furthermore, any functions and classes defined by the executed code are not guaranteed to work correctly after a :mod:`runpy` function has returned. If that limitation is not acceptable for a given use case, :mod:`importlib` is likely to be a more suitable choice than this module.

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\library\(cpython-main)(Doc)(library)runpy.rst, line 22); *backlink*

Unknown interpreted text role "mod".

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\library\(cpython-main)(Doc)(library)runpy.rst, line 22); *backlink*

Unknown interpreted text role "mod".

The :mod:`runpy` module provides two functions:

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-

main\Doc\library\(cpython-main)(Doc)(library)runpy.rst, line 27); *backlink*

Unknown interpreted text role "mod".

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\library\(cpython-main)(Doc)(library)runpy.rst, line 30)

Unknown directive type "function".

```
.. function:: run_module(mod_name, init_globals=None, run_name=None, alter_sys=False)

   .. index::
      module: __main__

   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 *mod_name* argument should be an absolute module name.
   If the module name refers to a package rather than a normal
   module, then that package is imported and the ``__main__`` submodule within
   that package is then executed and the resulting module globals dictionary
   returned.

   The optional dictionary argument *init_globals* may be used to pre-populate
   the module's 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 :func:`run_module`.

   The special global variables ``__name__``, ``__spec__``, ``__file__``,
   ``__cached__``, ``__loader__`` and ``__package__`` are set in the globals
   dictionary before the module code is executed (Note that this is a
   minimal set of variables - other variables may be set implicitly as an
   interpreter implementation detail).

   ``__name__`` is set to *run_name* if this optional argument is not
   :const:`None`, to ``mod_name + '.__main__'`` if the named module is a
   package and to the *mod_name* argument otherwise.

   ``__spec__`` will be set appropriately for the *actually* imported
   module (that is, ``__spec__.name`` will always be *mod_name* or
   ``mod_name + '.__main__'``, never *run_name*).

   ``__file__``, ``__cached__``, ``__loader__`` and ``__package__`` are
   :ref:`set as normal <import-mod-attrs>` based on the module spec.

   If the argument *alter_sys* is supplied and evaluates to :const:`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 :mod:`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 :mod:`sys` module be left alone when
   invoking this function from threaded code.

   .. seealso::
      The :option:`-m` option offering equivalent functionality from the
      command line.

   .. versionchanged:: 3.1
      Added ability to execute packages by looking for a ``__main__`` submodule.

   .. versionchanged:: 3.2
      Added ``__cached__`` global variable (see :pep:`3147`).

   .. versionchanged:: 3.4
      Updated to take advantage of the module spec feature added by
      :pep:`451`. This allows ``__cached__`` to be set correctly for modules
      run this way, as well as ensuring the real module name is always
      accessible as ``__spec__.name``.
```

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\library\(cpython-main)(Doc)(library)runpy.rst, line 96)

Unknown directive type "function".

```
.. function:: run_path(file_path, init_globals=None, run_name=None)

   .. index::
      module: __main__

   Execute the code at the named filesystem location and return the resulting
   module globals dictionary. As with a script name supplied to the CPython
   command line, the supplied path may refer to a Python source file, a
   compiled bytecode file or a valid sys.path entry containing a ``__main__``
   module (e.g. a zipfile containing a top-level ``__main__.py`` file).

   For a simple script, the specified code is simply executed in a fresh
   module namespace. For a valid sys.path entry (typically a zipfile or
   directory), the entry is first added to the beginning of ``sys.path``. The
   function then looks for and executes a :mod:`__main__` module using the
   updated path. Note that there is no special protection against invoking
   an existing :mod:`__main__` entry located elsewhere on ``sys.path`` if
   there is no such module at the specified location.

   The optional dictionary argument *init_globals* may be used to pre-populate
   the module's 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 :func:`run_path`.

   The special global variables ``__name__``, ``__spec__``, ``__file__``,
   ``__cached__``, ``__loader__`` and ``__package__`` are set in the globals
   dictionary before the module code is executed (Note that this is a
   minimal set of variables - other variables may be set implicitly as an
   interpreter implementation detail).

   ``__name__`` is set to *run_name* if this optional argument is not
   :const:`None` and to ``'<run_path>'`` otherwise.

   If the supplied path directly references a script file (whether as source
   or as precompiled byte code), then ``__file__`` will be set to the
   supplied path, and ``__spec__``, ``__cached__``, ``__loader__`` and
   ``__package__`` will all be set to :const:`None`.

   If the supplied path is a reference to a valid sys.path entry, then
   ``__spec__`` will be set appropriately for the imported ``__main__``
   module (that is, ``__spec__.name`` will always be ``__main__``).
   ``__file__``, ``__cached__``, ``__loader__`` and ``__package__`` will be
   :ref:`set as normal <import-mod-attrs>` based on the module spec.

   A number of alterations are also made to the :mod:`sys` module. Firstly,
   ``sys.path`` may be altered as described above. ``sys.argv[0]`` is updated
   with the value of ``file_path`` and ``sys.modules[__name__]`` is updated
   with a temporary module object for the module being executed. All
   modifications to items in :mod:`sys` are reverted before the function
   returns.

   Note that, unlike :func:`run_module`, the alterations made to :mod:`sys`
   are not optional in this function as these adjustments are essential to
   allowing the execution of sys.path entries. As the thread-safety
   limitations still apply, use of this function in threaded code should be
   either serialised with the import lock or delegated to a separate process.

   .. seealso::
      :ref:`using-on-interface-options` for equivalent functionality on the
      command line (``python path/to/script``).

   .. versionadded:: 3.2

   .. versionchanged:: 3.4
      Updated to take advantage of the module spec feature added by
      :pep:`451`. This allows ``__cached__`` to be set correctly in the
      case where ``__main__`` is imported from a valid sys.path entry rather
      than being executed directly.
```

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\library\(cpython-main)(Doc)(library)runpy.rst, line 166)

Unknown directive type "seealso".

```
.. seealso::
```

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

   :pep:`366` -- Main module explicit relative imports
      PEP written and implemented by Nick Coghlan.

   :pep:`451` -- A ModuleSpec Type for the Import System
      PEP written and implemented by Eric Snow

   :ref:`using-on-general` - CPython command line details

   The :func:`importlib.import_module` function
```