Optimizations

- Constructors `str()`, `bytes()` and `bytearray()` are now faster (around 30–40% for small objects). (Contributed by Serhiy Storchaka in bpo-41334.)
- The `runpy` module now imports fewer modules. The `python3 -m module-name` command startup time is 1.4x faster in average. On Linux, `python3 -I -m module-name` imports 69 modules on Python 3.9, whereas it only imports 51 modules (-18) on Python 3.10. (Contributed by Victor Stinner in bpo-41006 and bpo-41718.)
- The `LOAD_ATTR` instruction now uses new "per opcode cache" mechanism. It is about 36% faster now for regular attributes and 44% faster for slots. (Contributed by Pablo Galindo and Yury Selivanov in bpo-42093 and Guido van Rossum in bpo-42927, based on ideas implemented originally in PyPy and MicroPython.)
- When building Python with `--enable-optimizations` now `-fno-semantic-interposition` is added to both the compile and link line. This speeds builds of the Python interpreter created with `--enable-shared` with `gcc` by up to 30%. See this article for more details. (Contributed by Victor Stinner and Pablo Galindo in bpo-38980.)
- Use a new output buffer management code for `bz2` / `lzma` / `zlib` modules, and add `.readall()` function to `_compression.DecompressReader` class. bz2 decompression is now 1.09x ~ 1.17x faster, lzma decompression 1.20x ~ 1.32x faster, `GzipFile.read(-1)` 1.11x ~ 1.18x faster. (Contributed by Ma Lin, reviewed by Gregory P. Smith, in bpo-41486)
- When using stringized annotations, annotations dicts for functions are no longer created when the function is created. Instead, they are stored as a tuple of strings, and the function object lazily converts this into the annotations dict on demand. This optimization cuts the CPU time needed to define an annotated function by half. (Contributed by Yurii Karabas and Inada Naoki in bpo-42202.)
- Substring search functions such as `str1 in str2` and `str2.find(str1)` now sometimes use Crochemore & Perrin's "Two-Way" string searching algorithm to avoid quadratic behavior on long strings. (Contributed by Dennis Sweeney in bpo-41972)
- Add micro-optimizations to `_PyType_Lookup()` to improve type attribute cache lookup performance in the common case of cache hits. This makes the interpreter 1.04 times faster on average. (Contributed by Dino Viehland in bpo-43452.)
- The following built-in functions now support the faster **PEP 590** vectorcall calling convention: `map()`, `filter()`, `reversed()`, `bool()` and `float()`. (Contributed by Dong-hee Na and Jeroen Demeyer in bpo-43575, bpo-43287, bpo-41922, bpo-41873 and bpo-41870.)
- `BZ2File` performance is improved by removing internal `RLock`. This makes `BZ2File` thread unsafe in the face of multiple simultaneous readers or writers, just like its equivalent classes in `gzip` and `lzma` have always been. (Contributed by Inada Naoki in bpo-43785.)

Deprecated

- Currently Python accepts numeric literals immediately followed by keywords, for example `0in x`, `1or x`, `0if 1else 2`. It allows confusing and ambiguous expressions like `[0x1for x in y]` (which can be interpreted as `[0x1 for x in y]` or `[0x1f or x in y]`). Starting in this release, a deprecation warning is raised if the numeric literal is immediately followed by one of

keywords and, else, for, if, in, is and or. In future releases it will be changed to syntax warning, and finally to syntax error. (Contributed by Serhiy Storchaka in bpo-43833.)

- Starting in this release, there will be a concerted effort to begin cleaning up old import semantics that were kept for Python 2.7 compatibility. Specifically, find_loader()/find_module() (superseded by find_spec()), load_module() (superseded by exec_module()), module_repr() (which the import system takes care of for you), the __package__ attribute (superseded by __spec__.parent), the __loader__ attribute (superseded by __spec__.loader), and the __cached__ attribute (superseded by __spec__.cached) will slowly be removed (as well as other classes and methods in importlib). ImportWarning and/or DeprecationWarning will be raised as appropriate to help identify code which needs updating during this transition.
- The entire distutils namespace is deprecated, to be removed in Python 3.12. Refer to the module changes section for more information.
- Non-integer arguments to random.randrange() are deprecated. The ValueError is deprecated in favor of a TypeError. (Contributed by Serhiy Storchaka and Raymond Hettinger in bpo-37319.)
- The various load_module() methods of importlib have been documented as deprecated since Python 3.6, but will now also trigger a DeprecationWarning. Use exec_module() instead. (Contributed by Brett Cannon in bpo-26131.)
- zimport.zipimporter.load_module() has been deprecated in preference for exec_module(). (Contributed by Brett Cannon in bpo-26131.)
- The use of load_module() by the import system now triggers an ImportWarning as exec_module() is preferred. (Contributed by Brett Cannon in bpo-26131.)
- The use of importlib.abc.MetaPathFinder.find_module() and importlib.abc.PathEntryFinder.find_module() by the import system now trigger an ImportWarning as importlib.abc.MetaPathFinder.find_spec() and importlib.abc.PathEntryFinder.find_spec() are preferred, respectively. You can use importlib.util.spec_from_loader() to help in porting. (Contributed by Brett Cannon in bpo-42134.)
- The use of importlib.abc.PathEntryFinder.find_loader() by the import system now triggers an ImportWarning as importlib.abc.PathEntryFinder.find_spec() is preferred. You can use importlib.util.spec_from_loader() to help in porting. (Contributed by Brett Cannon in bpo-43672.)
- The various implementations of importlib.abc.MetaPathFinder.find_module() (importlib.machinery.BuiltinImporter.find_module(), importlib.machinery.FrozenImporter.find_module(), importlib.machinery.WindowsRegistryFinder.find_module(), importlib.machinery.PathFinder.find_module(), importlib.abc.MetaPathFinder.find_module()), importlib.abc.PathEntryFinder.find_module() (importlib.machinery.FileFinder.find_module()),

and `importlib.abc.PathEntryFinder.find_loader()` (`importlib.machinery.FileFinder.find_loader()`) now raise `DeprecationWarning` and are slated for removal in Python 3.12 (previously they were documented as deprecated in Python 3.4). (Contributed by Brett Cannon in bpo-42135.)

- `importlib.abc.Finder` is deprecated (including its sole method, `find_module()`). Both `importlib.abc.MetaPathFinder` and `importlib.abc.PathEntryFinder` no longer inherit from the class. Users should inherit from one of these two classes as appropriate instead. (Contributed by Brett Cannon in bpo-42135.)
- The deprecations of `imp`, `importlib.find_loader()`, `importlib.util.set_package_wrapper()`, `importlib.util.set_loader_wrapper()`, `importlib.util.module_for_loader()`, `pkgutil.ImpImporter`, and `pkgutil.ImpLoader` have all been updated to list Python 3.12 as the slated version of removal (they began raising `DeprecationWarning` in previous versions of Python). (Contributed by Brett Cannon in bpo-43720.)
- The import system now uses the `__spec__` attribute on modules before falling back on `module_repr()` for a module's `__repr__()` method. Removal of the use of `module_repr()` is scheduled for Python 3.12. (Contributed by Brett Cannon in bpo-42137.)
- `importlib.abc.Loader.module_repr()`, `importlib.machinery.FrozenLoader.module_repr()`, and `importlib.machinery.BuiltinLoader.module_repr()` are deprecated and slated for removal in Python 3.12. (Contributed by Brett Cannon in bpo-42136.)
- `sqlite3.OptimizedUnicode` has been undocumented and obsolete since Python 3.3, when it was made an alias to `str`. It is now deprecated, scheduled for removal in Python 3.12. (Contributed by Erlend E. Aasland in bpo-42264.)
- The undocumented built-in function `sqlite3.enable_shared_cache` is now deprecated, scheduled for removal in Python 3.12. Its use is strongly discouraged by the SQLite3 documentation. See the SQLite3 docs for more details. If a shared cache must be used, open the database in URI mode using the `cache=shared` query parameter. (Contributed by Erlend E. Aasland in bpo-24464.)
- The following `threading` methods are now deprecated:
  - `threading.currentThread` => `threading.current_thread()`
  - `threading.activeCount` => `threading.active_count()`
  - `threading.Condition.notifyAll` => `threading.Condition.notify_all()`
  - `threading.Event.isSet` => `threading.Event.is_set()`
  - `threading.Thread.setName` => `threading.Thread.name`
  - `threading.thread.getName` => `threading.Thread.name`
  - `threading.Thread.isDaemon` => `threading.Thread.daemon`
  - `threading.Thread.setDaemon` => `threading.Thread.daemon`

  (Contributed by Jelle Zijlstra in gh-87889.)

- `pathlib.Path.link_to()` is deprecated and slated for removal in Python 3.12. Use `pathlib.Path.hardlink_to()` instead. (Contributed by Barney Gale in bpo-39950.)
- `cgi.log()` is deprecated and slated for removal in Python 3.12. (Contributed by Inada Naoki in bpo-41139.)
- The following `ssl` features have been deprecated since Python 3.6, Python 3.7, or OpenSSL 1.1.0 and will be removed in 3.11:
  - `OP_NO_SSLv2`, `OP_NO_SSLv3`, `OP_NO_TLSv1`, `OP_NO_TLSv1_1`, `OP_NO_TLSv1_2`, and `OP_NO_TLSv1_3` are replaced by `sslSSLContext.minimum_version` and `sslSSLContext.maximum_version`.
  - `PROTOCOL_SSLv2`, `PROTOCOL_SSLv3`, `PROTOCOL_SSLv23`, `PROTOCOL_TLSv1`, `PROTOCOL_TLSv1_1`, `PROTOCOL_TLSv1_2`, and `PROTOCOL_TLS` are deprecated in favor of `PROTOCOL_TLS_CLIENT` and `PROTOCOL_TLS_SERVER`
  - `wrap_socket()` is replaced by `ssl.SSLContext.wrap_socket()`
  - `match_hostname()`
  - `RAND_pseudo_bytes()`, `RAND_egd()`
  - NPN features like `ssl.SSLSocket.selected_npn_protocol()` and `ssl.SSLContext.set_npn_protocols()` are replaced by ALPN.
- The threading debug (`PYTHONTHREADDEBUG` environment variable) is deprecated in Python 3.10 and will be removed in Python 3.12. This feature requires a debug build of Python. (Contributed by Victor Stinner in bpo-44584.)
- Importing from the `typing.io` and `typing.re` submodules will now emit `DeprecationWarning`. These submodules will be removed in a future version of Python. Anything belonging to these submodules should be imported directly from `typing` instead. (Contributed by Sebastian Rittau in bpo-38291.)

Removed

- Removed special methods `__int__`, `__float__`, `__floordiv__`, `__mod__`, `__divmod__`, `__rfloordiv__`, `__rmod__` and `__rdivmod__` of the `complex` class. They always raised a `TypeError`. (Contributed by Serhiy Storchaka in bpo-41974.)
- The `ParserBase.error()` method from the private and undocumented `_markupbase` module has been removed. `html.parser.HTMLParser` is the only subclass of `ParserBase` and its `error()` implementation was already removed in Python 3.5. (Contributed by Berker Peksag in bpo-31844.)
- Removed the `unicodedata.ucnhash_CAPI` attribute which was an internal PyCapsule object. The related private `_PyUnicode_Name_CAPI` structure was moved to the internal C API. (Contributed by Victor Stinner in bpo-42157.)
- Removed the `parser` module, which was deprecated in 3.9 due to the switch to the new PEG parser, as well as all the C source and header files that were only being used by the old parser, including `node.h`, `parser.h`, `graminit.h` and `grammar.h`.
- Removed the Public C API functions `PyParser_SimpleParseStringFlags`, `PyParser_SimpleParseStringFl`

`agsFilename`, `PyParser_SimpleParseFileFlags` and `PyNode_Compile` that were deprecated in 3.9 due to the switch to the new PEG parser.

- Removed the `formatter` module, which was deprecated in Python 3.4. It is somewhat obsolete, little used, and not tested. It was originally scheduled to be removed in Python 3.6, but such removals were delayed until after Python 2.7 EOL. Existing users should copy whatever classes they use into their code. (Contributed by Dong-hee Na and Terry J. Reedy in bpo-42299.)
- Removed the `PyModule_GetWarningsModule()` function that was useless now due to the _warnings module was converted to a builtin module in 2.6. (Contributed by Hai Shi in bpo-42599.)
- Remove deprecated aliases to Collections Abstract Base Classes from the `collections` module. (Contributed by Victor Stinner in bpo-37324.)
- The `loop` parameter has been removed from most of `asyncio`'s high-level API following deprecation in Python 3.8. The motivation behind this change is multifold:
    1. This simplifies the high-level API.
    2. The functions in the high-level API have been implicitly getting the current thread's running event loop since Python 3.7. There isn't a need to pass the event loop to the API in most normal use cases.
    3. Event loop passing is error-prone especially when dealing with loops running in different threads.

    Note that the low-level API will still accept `loop`. See Changes in the Python API for examples of how to replace existing code.

    (Contributed by Yurii Karabas, Andrew Svetlov, Yury Selivanov and Kyle Stanley in bpo-42392.)

Porting to Python 3.10

This section lists previously described changes and other bugfixes that may require changes to your code.

Changes in the Python syntax

- Deprecation warning is now emitted when compiling previously valid syntax if the numeric literal is immediately followed by a keyword (like in `0in x`). In future releases it will be changed to syntax warning, and finally to a syntax error. To get rid of the warning and make the code compatible with future releases just add a space between the numeric literal and the following keyword. (Contributed by Serhiy Storchaka in bpo-43833.)

Changes in the Python API

- The *etype* parameters of the `format_exception()`, `format_exception_only()`, and `print_exception()` functions in the `traceback` module have been renamed to *exc*. (Contributed by Zackery Spytz and Matthias Bussonnier in bpo-26389.)
- `atexit`: At Python exit, if a callback registered with `atexit.register()` fails, its exception is now logged. Previously, only some exceptions were logged, and the

last exception was always silently ignored. (Contributed by Victor Stinner in bpo-42639.)

- `collections.abc.Callable` generic now flattens type parameters, similar to what `typing.Callable` currently does. This means that `collections.abc.Callable[[int, str], str]` will have `__args__` of `(int, str, str)`; previously this was `([int, str], str)`. Code which accesses the arguments via `typing.get_args()` or `__args__` need to account for this change. Furthermore, `TypeError` may be raised for invalid forms of parameterizing `collections.abc.Callable` which may have passed silently in Python 3.9. (Contributed by Ken Jin in bpo-42195.)
- `socket.htons()` and `socket.ntohs()` now raise `OverflowError` instead of `DeprecationWarning` if the given parameter will not fit in a 16-bit unsigned integer. (Contributed by Erlend E. Aasland in bpo-42393.)
- The `loop` parameter has been removed from most of `asyncio`'s high-level API following deprecation in Python 3.8.

  A coroutine that currently looks like this:

  ```
  async def foo(loop):
      await asyncio.sleep(1, loop=loop)
  ```

  Should be replaced with this:

  ```
  async def foo():
      await asyncio.sleep(1)
  ```

  If `foo()` was specifically designed *not* to run in the current thread's running event loop (e.g. running in another thread's event loop), consider using `asyncio.run_coroutine_threadsafe()` instead.

  (Contributed by Yurii Karabas, Andrew Svetlov, Yury Selivanov and Kyle Stanley in bpo-42392.)

- The `types.FunctionType` constructor now inherits the current builtins if the *globals* dictionary has no `"__builtins__"` key, rather than using `{"None": None}` as builtins: same behavior as `eval()` and `exec()` functions. Defining a function with `def function(...): ...` in Python is not affected, globals cannot be overridden with this syntax: it also inherits the current builtins. (Contributed by Victor Stinner in bpo-42990.)

Changes in the C API

- The C API functions `PyParser_SimpleParseStringFlags`, `PyParser_SimpleParseStringFlagsFilename`, `PyParser_SimpleParseFileFlags`, `PyNode_Compile` and the type

used by these functions, `struct _node`, were removed due to the switch to the new PEG parser.

Source should be now be compiled directly to a code object using, for example, `Py_CompileString()`. The resulting code object can then be evaluated using, for example, `PyEval_EvalCode()`.

Specifically:

- A call to `PyParser_SimpleParseStringFlags` followed by `PyNode_Compile` can be replaced by calling `Py_CompileString()`.
- There is no direct replacement for `PyParser_SimpleParseFileFlags`. To compile code from a `FILE *` argument, you will need to read the file in C and pass the resulting buffer to `Py_CompileString()`.
- To compile a file given a `char *` filename, explicitly open the file, read it and compile the result. One way to do this is using the `io` module with `PyImport_ImportModule()`, `PyObject_CallMethod()`, `PyBytes_AsString()` and `Py_CompileString()`, as sketched below. (Declarations and error handling are omitted.)

```
io_module = Import_ImportModule("io");
fileobject = PyObject_CallMethod(io_module, "open", "ss", filename, "rb");
source_bytes_object = PyObject_CallMethod(fileobject, "read", "");
result = PyObject_CallMethod(fileobject, "close", "");
source_buf = PyBytes_AsString(source_bytes_object);
code = Py_CompileString(source_buf, filename, Py_file_input);
```

- For `FrameObject` objects, the `f_lasti` member now represents a wordcode offset instead of a simple offset into the bytecode string. This means that this number needs to be multiplied by 2 to be used with APIs that expect a byte offset instead (like `PyCode_Addr2Line()` for example). Notice as well that the `f_lasti` member of `FrameObject` objects is not considered stable: please use `PyFrame_GetLineNumber()` instead.

CPython bytecode changes

- The `MAKE_FUNCTION` instruction now accepts either a dict or a tuple of strings as the function's annotations. (Contributed by Yurii Karabas and Inada Naoki in bpo-42202.)

Build Changes

- **PEP 644**: Python now requires OpenSSL 1.1.1 or newer. OpenSSL 1.0.2 is no longer supported. (Contributed by Christian Heimes in bpo-43669.)
- The C99 functions `snprintf()` and `vsnprintf()` are now required to build Python. (Contributed by Victor Stinner in bpo-36020.)

- `sqlite3` requires SQLite 3.7.15 or higher. (Contributed by Sergey Fedoseev and Erlend E. Aasland in bpo-40744 and bpo-40810.)
- The `atexit` module must now always be built as a built-in module. (Contributed by Victor Stinner in bpo-42639.)
- Add `--disable-test-modules` option to the `configure` script: don't build nor install test modules. (Contributed by Xavier de Gaye, Thomas Petazzoni and Peixing Xin in bpo-27640.)
- Add `--with-wheel-pkg-dir=PATH option` to the `./configure` script. If specified, the `ensurepip` module looks for `setuptools` and `pip` wheel packages in this directory: if both are present, these wheel packages are used instead of ensurepip bundled wheel packages.

  Some Linux distribution packaging policies recommend against bundling dependencies. For example, Fedora installs wheel packages in the `/usr/share/python-wheels/` directory and don't install the `ensurepip._bundled` package.

  (Contributed by Victor Stinner in bpo-42856.)

- Add a new `configure --without-static-libpython option` to not build the `libpythonMAJOR.MINOR.a` static library and not install the `python.o` object file.

  (Contributed by Victor Stinner in bpo-43103.)

- The `configure` script now uses the `pkg-config` utility, if available, to detect the location of Tcl/Tk headers and libraries. As before, those locations can be explicitly specified with the `--with-tcltk-includes` and `--with-tcltk-libs` configuration options. (Contributed by Manolis Stamatogiannakis in bpo-42603.)
- Add `--with-openssl-rpath` option to `configure` script. The option simplifies building Python with a custom OpenSSL installation, e.g. `./configure --with-openssl=/path/to/openssl --with-openssl-rpath=auto`. (Contributed by Christian Heimes in bpo-43466.)

C API Changes

PEP 652: Maintaining the Stable ABI

The Stable ABI (Application Binary Interface) for extension modules or embedding Python is now explicitly defined. C API Stability describes C API and ABI stability guarantees along with best practices for using the Stable ABI.

(Contributed by Petr Viktorin in **PEP 652** and bpo-43795.)

New Features

- The result of `PyNumber_Index()` now always has exact type `int`. Previously, the result could have been an instance of a subclass of `int`. (Contributed by Serhiy Storchaka in bpo-40792.)
- Add a new `orig_argv` member to the `PyConfig` structure: the list of the original command line arguments passed to the Python executable. (Contributed by Victor Stinner in bpo-23427.)
- The `PyDateTime_DATE_GET_TZINFO()` and `PyDateTime_TIME_GET_TZINFO()` macros have been added for accessing the `tzinfo` attributes of `datetime.datetime` and `datetime.time` objects. (Contributed by Zackery Spytz in bpo-30155.)
- Add a `PyCodec_Unregister()` function to unregister a codec search function. (Contributed by Hai Shi in bpo-41842.)
- The `PyIter_Send()` function was added to allow sending value into iterator without raising `StopIteration` exception. (Contributed by Vladimir Matveev in bpo-41756.)
- Add `PyUnicode_AsUTF8AndSize()` to the limited C API. (Contributed by Alex Gaynor in bpo-41784.)
- Add `PyModule_AddObjectRef()` function: similar to `PyModule_AddObject()` but don't steal a reference to the value on success. (Contributed by Victor Stinner in bpo-1635741.)
- Add `Py_NewRef()` and `Py_XNewRef()` functions to increment the reference count of an object and return the object. (Contributed by Victor Stinner in bpo-42262.)
- The `PyType_FromSpecWithBases()` and `PyType_FromModuleAndSpec()` functions now accept a single class as the *bases* argument. (Contributed by Serhiy Storchaka in bpo-42423.)
- The `PyType_FromModuleAndSpec()` function now accepts NULL `tp_doc` slot. (Contributed by Hai Shi in bpo-41832.)
- The `PyType_GetSlot()` function can accept static types. (Contributed by Hai Shi and Petr Viktorin in bpo-41073.)
- Add a new `PySet_CheckExact()` function to the C-API to check if an object is an instance of `set` but not an instance of a subtype. (Contributed by Pablo Galindo in bpo-43277.)
- Add `PyErr_SetInterruptEx()` which allows passing a signal number to simulate. (Contributed by Antoine Pitrou in bpo-43356.)
- The limited C API is now supported if Python is built in debug mode (if the `Py_DEBUG` macro is defined). In the limited C API, the `Py_INCREF()` and `Py_DECREF()` functions are now implemented as opaque function calls, rather than accessing directly the `PyObject.ob_refcnt` member, if Python is built in debug mode and the `Py_LIMITED_API` macro targets Python 3.10 or newer. It became possible to support the limited C API in debug mode because the `PyObject` structure is the same in release and debug mode since Python 3.8 (see bpo-36465).

The limited C API is still not supported in the `--with-trace-refs` special build (`Py_TRACE_REFS` macro). (Contributed by Victor Stinner in bpo-43688.)

- Add the `Py_Is(x, y)` function to test if the *x* object is the *y* object, the same as `x is y` in Python. Add also the `Py_IsNone()`, `Py_IsTrue()`, `Py_IsFalse()` functions to test if an object is, respectively, the `None` singleton, the `True` singleton or the `False` singleton. (Contributed by Victor Stinner in bpo-43753.)
- Add new functions to control the garbage collector from C code: `PyGC_Enable()`, `PyGC_Disable()`, `PyGC_IsEnabled()`. These functions allow to activate, deactivate and query the state of the garbage collector from C code without having to import the `gc` module.
- Add a new `Py_TPFLAGS_DISALLOW_INSTANTIATION` type flag to disallow creating type instances. (Contributed by Victor Stinner in bpo-43916.)
- Add a new `Py_TPFLAGS_IMMUTABLETYPE` type flag for creating immutable type objects: type attributes cannot be set nor deleted. (Contributed by Victor Stinner and Erlend E. Aasland in bpo-43908.)

Porting to Python 3.10

- The `PY_SSIZE_T_CLEAN` macro must now be defined to use `PyArg_ParseTuple()` and `Py_BuildValue()` formats which use `#`: `es#`, `et#`, `s#`, `u#`, `y#`, `z#`, `U#` and `Z#`. See Parsing arguments and building values and **PEP 353**. (Contributed by Victor Stinner in bpo-40943.)
- Since `Py_REFCNT()` is changed to the inline static function, `Py_REFCNT(obj) = new_refcnt` must be replaced with `Py_SET_REFCNT(obj, new_refcnt)`: see `Py_SET_REFCNT()` (available since Python 3.9). For backward compatibility, this macro can be used:

```
#if PY_VERSION_HEX < 0x030900A4
#  define Py_SET_REFCNT(obj, refcnt) ((Py_REFCNT(obj) = (refcnt)), (void)0)
#endif
```

(Contributed by Victor Stinner in bpo-39573.)

- Calling `PyDict_GetItem()` without GIL held had been allowed for historical reason. It is no longer allowed. (Contributed by Victor Stinner in bpo-40839.)
- `PyUnicode_FromUnicode(NULL, size)` and `PyUnicode_FromStringAndSize(NULL, size)` raise `DeprecationWarning` now. Use `PyUnicode_New()` to allocate Unicode object without initial data. (Contributed by Inada Naoki in bpo-36346.)
- The private `_PyUnicode_Name_CAPI` structure of the PyCapsule API `unicodedata.ucnhash_CAPI` has been moved to the internal C API. (Contributed by Victor Stinner in bpo-42157.)
- `Py_GetPath()`, `Py_GetPrefix()`, `Py_GetExecPrefix()`, `Py_GetProgramFullPath()`, `Py_GetPythonHome()` and `Py_GetProgramName()` functions now return `NULL` if called before `Py_Initialize()` (before Python is initialized). Use the new Python

Initialization Configuration API to get the Python Path Configuration. (Contributed by Victor Stinner in bpo-42260.)

- `PyList_SET_ITEM()`, `PyTuple_SET_ITEM()` and `PyCell_SET()` macros can no longer be used as l-value or r-value. For example, `x = PyList_SET_ITEM(a, b, c)` and `PyList_SET_ITEM(a, b, c) = x` now fail with a compiler error. It prevents bugs like `if (PyList_SET_ITEM (a, b, c) < 0) ...` test. (Contributed by Zackery Spytz and Victor Stinner in bpo-30459.)
- The non-limited API files `odictobject.h`, `parser_interface.h`, `picklebufobject.h`, `pyarena.h`, `pyctype.h`, `pydebug.h`, `pyfpe.h`, and `pytime.h` have been moved to the `Include/cpython` directory. These files must not be included directly, as they are already included in `Python.h`; see Include Files. If they have been included directly, consider including `Python.h` instead. (Contributed by Nicholas Sim in bpo-35134.)
- Use the `Py_TPFLAGS_IMMUTABLETYPE` type flag to create immutable type objects. Do not rely on `Py_TPFLAGS_HEAPTYPE` to decide if a type object is mutable or not; check if `Py_TPFLAGS_IMMUTABLETYPE` is set instead. (Contributed by Victor Stinner and Erlend E. Aasland in bpo-43908.)
- The undocumented function `Py_FrozenMain` has been removed from the limited API. The function is mainly useful for custom builds of Python. (Contributed by Petr Viktorin in bpo-26241.)

Deprecated

- The `PyUnicode_InternImmortal()` function is now deprecated and will be removed in Python 3.12: use `PyUnicode_InternInPlace()` instead. (Contributed by Victor Stinner in bpo-41692.)

Removed

- Removed `Py_UNICODE_str*` functions manipulating `Py_UNICODE*` strings. (Contributed by Inada Naoki in bpo-41123.)
  - `Py_UNICODE_strlen`: use `PyUnicode_GetLength()` or `PyUnicode_GET_LENGTH`
  - `Py_UNICODE_strcat`: use `PyUnicode_CopyCharacters()` or `PyUnicode_FromFormat()`
  - `Py_UNICODE_strcpy`, `Py_UNICODE_strncpy`: use `PyUnicode_CopyCharacters()` or `PyUnicode_Substring()`
  - `Py_UNICODE_strcmp`: use `PyUnicode_Compare()`
  - `Py_UNICODE_strncmp`: use `PyUnicode_Tailmatch()`
  - `Py_UNICODE_strchr`, `Py_UNICODE_strrchr`: use `PyUnicode_FindChar()`
- Removed `PyUnicode_GetMax()`. Please migrate to new (**PEP 393**) APIs. (Contributed by Inada Naoki in bpo-41103.)
- Removed `PyLong_FromUnicode()`. Please migrate to `PyLong_FromUnicodeObject()`. (Contributed by Inada Naoki in bpo-41103.)
- Removed `PyUnicode_AsUnicodeCopy()`. Please use `PyUnicode_AsUCS4Copy()` or `PyUnicode_AsWideCharString()` (Contributed by Inada Naoki in bpo-41103.)

- Removed `_Py_CheckRecursionLimit` variable: it has been replaced by `ceval.recursion_limit` of the `PyInterpreterState` structure. (Contributed by Victor Stinner in bpo-41834.)
- Removed undocumented macros `Py_ALLOW_RECURSION` and `Py_END_ALLOW_RECURSION` and the `recursion_critical` field of the `PyInterpreterState` structure. (Contributed by Serhiy Storchaka in bpo-41936.)
- Removed the undocumented `PyOS_InitInterrupts()` function. Initializing Python already implicitly installs signal handlers: see `PyConfig.install_signal_handlers`. (Contributed by Victor Stinner in bpo-41713.)
- Remove the `PyAST_Validate()` function. It is no longer possible to build a AST object (`mod_ty` type) with the public C API. The function was already excluded from the limited C API (**PEP 384**). (Contributed by Victor Stinner in bpo-43244.)
- Remove the `symtable.h` header file and the undocumented functions:
  - `PyST_GetScope()`
  - `PySymtable_Build()`
  - `PySymtable_BuildObject()`
  - `PySymtable_Free()`
  - `Py_SymtableString()`
  - `Py_SymtableStringObject()`

  The `Py_SymtableString()` function was part the stable ABI by mistake but it could not be used, because the `symtable.h` header file was excluded from the limited C API.

  Use Python `symtable` module instead. (Contributed by Victor Stinner in bpo-43244.)

- Remove `PyOS_ReadlineFunctionPointer()` from the limited C API headers and from `python3.dll`, the library that provides the stable ABI on Windows. Since the function takes a `FILE*` argument, its ABI stability cannot be guaranteed. (Contributed by Petr Viktorin in bpo-43868.)
- Remove `ast.h`, `asdl.h`, and `Python-ast.h` header files. These functions were undocumented and excluded from the limited C API. Most names defined by these header files were not prefixed by `Py` and so could create names conflicts. For example, `Python-ast.h` defined a `Yield` macro which was conflict with the `Yield` name used by the Windows `<winbase.h>` header. Use the Python `ast` module instead. (Contributed by Victor Stinner in bpo-43244.)
- Remove the compiler and parser functions using `struct _mod` type, because the public AST C API was removed:
  - `PyAST_Compile()`
  - `PyAST_CompileEx()`
  - `PyAST_CompileObject()`
  - `PyFuture_FromAST()`
  - `PyFuture_FromASTObject()`

    - `PyParser_ASTFromFile()`
    - `PyParser_ASTFromFileObject()`
    - `PyParser_ASTFromFilename()`
    - `PyParser_ASTFromString()`
    - `PyParser_ASTFromStringObject()`

    These functions were undocumented and excluded from the limited C API. (Contributed by Victor Stinner in bpo-43244.)

- Remove the `pyarena.h` header file with functions:
    - `PyArena_New()`
    - `PyArena_Free()`
    - `PyArena_Malloc()`
    - `PyArena_AddPyObject()`

    These functions were undocumented, excluded from the limited C API, and were only used internally by the compiler. (Contributed by Victor Stinner in bpo-43244.)

- The `PyThreadState.use_tracing` member has been removed to optimize Python. (Contributed by Mark Shannon in bpo-43760.)