3. Configure Python

3.1. Configure Options

List all `./configure` script options using:

```
./configure --help
```

See also the `Misc/SpecialBuilds.txt` in the Python source distribution.

3.1.1. General Options

`--enable-loadable-sqlite-extensions`

Support loadable extensions in the `_sqlite` extension module (default is no).

See the `sqlite3.Connection.enable_load_extension()` method of the `sqlite3` module.

New in version 3.6.

`--disable-ipv6`

Disable IPv6 support (enabled by default if supported), see the `socket` module.

`--enable-big-digits``=[15|30]`

Define the size in bits of Python `int` digits: 15 or 30 bits.

By default, the number of bits is selected depending on `sizeof(void*)`: 30 bits if `void*` size is 64-bit or larger, 15 bits otherwise.

Define the `PYLONG_BITS_IN_DIGIT` to `15` or `30`.

See `sys.int_info.bits_per_digit`.

`--with-cxx-main`

`--with-cxx-main``=COMPILER`

Compile the Python `main()` function and link Python executable with C++ compiler: `$CXX`, or *COMPILER* if specified.

`--with-suffix``=SUFFIX`

Set the Python executable suffix to *SUFFIX*.

The default suffix is `.exe` on Windows and macOS (`python.exe` executable), and an empty string on other platforms (`python` executable).

`--with-tzpath``=<list of absolute paths separated by pathsep>`

Select the default time zone search path for `zoneinfo.TZPATH`. See the Compile-time configuration of the `zoneinfo` module.

Default: `/usr/share/zoneinfo:/usr/lib/zoneinfo:/usr/share/lib/zoneinfo:/etc/zoneinfo`.

See `os.pathsep` path separator.

New in version 3.9.

`--without-decimal-contextvar`

Build the `_decimal` extension module using a thread-local context rather than a coroutine-local context (default), see the `decimal` module.

See `decimal.HAVE_CONTEXTVAR` and the `contextvars` module.

New in version 3.9.

--with-dbmliborder=db1:db2:...
Override order to check db backends for the `dbm` module

A valid value is a colon (`:`) separated string with the backend names:

- `ndbm`;
- `gdbm`;
- `bdb`.

--without-c-locale-coercion
Disable C locale coercion to a UTF-8 based locale (enabled by default).

Don't define the `PY_COERCE_C_LOCALE` macro.

See `PYTHONCOERCECLOCALE` and the **PEP 538**.

--with-platlibdir=DIRNAME
Python library directory name (default is `lib`).

Fedora and SuSE use `lib64` on 64-bit platforms.

See `sys.platlibdir`.

New in version 3.9.

--with-wheel-pkg-dir=PATH
Directory of wheel packages used by the `ensurepip` module (none by default).

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.

New in version 3.10.

3.1.2. Install Options

--disable-test-modules
Don't build nor install test modules, like the `test` package or the `_testcapi` extension module (built and installed by default).

New in version 3.10.

--with-ensurepip=[upgrade|install|no]
Select the `ensurepip` command run on Python installation:

- `upgrade` (default): run `python -m ensurepip --altinstall --upgrade` command.
- `install`: run `python -m ensurepip --altinstall` command;
- `no`: don't run ensurepip;

New in version 3.6.

3.1.3. Performance options

Configuring Python using `--enable-optimizations --with-lto` (PGO + LTO) is recommended for best performance.

`--enable-optimizations`
Enable Profile Guided Optimization (PGO) using `PROFILE_TASK` (disabled by default).

The C compiler Clang requires `llvm-profdata` program for PGO. On macOS, GCC also requires it: GCC is just an alias to Clang on macOS.

Disable also semantic interposition in libpython if `--enable-shared` and GCC is used: add `-fno-semantic-interposition` to the compiler and linker flags.

New in version 3.6.

Changed in version 3.10: Use `-fno-semantic-interposition` on GCC.

`PROFILE_TASK`
Environment variable used in the Makefile: Python command line arguments for the PGO generation task.

Default: `-m test --pgo --timeout=$(TESTTIMEOUT)`.

New in version 3.8.

`--with-lto`
Enable Link Time Optimization (LTO) in any build (disabled by default).

The C compiler Clang requires `llvm-ar` for LTO (`ar` on macOS), as well as an LTO-aware linker (`ld.gold` or `lld`).

New in version 3.6.

`--with-computed-gotos`
Enable computed gotos in evaluation loop (enabled by default on supported compilers).

`--without-pymalloc`
Disable the specialized Python memory allocator pymalloc (enabled by default).

See also `PYTHONMALLOC` environment variable.

`--without-doc-strings`
Disable static documentation strings to reduce the memory footprint (enabled by default). Documentation strings defined in Python are not affected.

Don't define the `WITH_DOC_STRINGS` macro.

See the `PyDoc_STRVAR()` macro.

`--enable-profiling`
Enable C-level code profiling with `gprof` (disabled by default).

3.1.4. Python Debug Build

A debug build is Python built with the `--with-pydebug` configure option.

Effects of a debug build:

- Display all warnings by default: the list of default warning filters is empty in the `warnings` module.

- Add `d` to `sys.abiflags`.
- Add `sys.gettotalrefcount()` function.
- Add `-X showrefcount` command line option.
- Add `PYTHONTHREADDEBUG` environment variable.
- Add support for the `__ltrace__` variable: enable low-level tracing in the bytecode evaluation loop if the variable is defined.
- Install debug hooks on memory allocators to detect buffer overflow and other memory errors.
- Define `Py_DEBUG` and `Py_REF_DEBUG` macros.
- Add runtime checks: code surroundeded by `#ifdef Py_DEBUG` and `#endif`. Enable `assert(...)` and `_PyObject_ASSERT(...)` assertions: don't set the `NDEBUG` macro (see also the `--with-assertions` configure option). Main runtime checks:
  - Add sanity checks on the function arguments.
  - Unicode and int objects are created with their memory filled with a pattern to detect usage of uninitialized objects.
  - Ensure that functions which can clear or replace the current exception are not called with an exception raised.
  - The garbage collector (`gc.collect()` function) runs some basic checks on objects consistency.
  - The `Py_SAFE_DOWNCAST()` macro checks for integer underflow and overflow when downcasting from wide types to narrow types.

See also the Python Development Mode and the `--with-trace-refs` configure option.

Changed in version 3.8: Release builds and debug builds are now ABI compatible: defining the `Py_DEBUG` macro no longer implies the `Py_TRACE_REFS` macro (see the `--with-trace-refs` option), which introduces the only ABI incompatibility.

3.1.5. Debug options

`--with-pydebug`

Build Python in debug mode: define the `Py_DEBUG` macro (disabled by default).

`--with-trace-refs`

Enable tracing references for debugging purpose (disabled by default).

Effects:

- Define the `Py_TRACE_REFS` macro.
- Add `sys.getobjects()` function.
- Add `PYTHONDUMPREFS` environment variable.

This build is not ABI compatible with release build (default build) or debug build (`Py_DEBUG` and `Py_REF_DEBUG` macros).

New in version 3.8.

`--with-assertions`

Build with C assertions enabled (default is no): `assert(...);` and `_PyObject_ASSERT(...);`.

If set, the `NDEBUG` macro is not defined in the `OPT` compiler variable.

See also the `--with-pydebug` option (debug build) which also enables assertions.

New in version 3.6.

`--with-valgrind`

Enable Valgrind support (default is no).

--with-dtrace

Enable DTrace support (default is no).

See Instrumenting CPython with DTrace and SystemTap.

New in version 3.6.

--with-address-sanitizer

Enable AddressSanitizer memory error detector, `asan` (default is no).

New in version 3.6.

--with-memory-sanitizer

Enable MemorySanitizer allocation error detector, `msan` (default is no).

New in version 3.6.

--with-undefined-behavior-sanitizer

Enable UndefinedBehaviorSanitizer undefined behaviour detector, `ubsan` (default is no).

New in version 3.6.

3.1.6. Linker options

--enable-shared

Enable building a shared Python library: `libpython` (default is no).

--without-static-libpython

Do not build `libpythonMAJOR.MINOR.a` and do not install `python.o` (built and enabled by default).

New in version 3.10.

3.1.7. Libraries options

--with-libs`='lib1 ...'`

Link against additional libraries (default is no).

--with-system-expat

Build the `pyexpat` module using an installed `expat` library (default is no).

--with-system-ffi

Build the `_ctypes` extension module using an installed `ffi` library, see the `ctypes` module (default is system-dependent).

--with-system-libmpdec

Build the `_decimal` extension module using an installed `mpdec` library, see the `decimal` module (default is no).

New in version 3.3.

--with-readline`=editline`

Use `editline` library for backend of the `readline` module.

Define the `WITH_EDITLINE` macro.

New in version 3.10.

--without-readline

Don't build the `readline` module (built by default).

Don't define the `HAVE_LIBREADLINE` macro.

New in version 3.10.

--with-tcltk-includes`='-I...'`
Override search for Tcl and Tk include files.

--with-tcltk-libs`='-L...'`
Override search for Tcl and Tk libraries.

--with-libm`=STRING`
Override `libm` math library to *STRING* (default is system-dependent).

--with-libc`=STRING`
Override `libc` C library to *STRING* (default is system-dependent).

--with-openssl`=DIR`
Root of the OpenSSL directory.

New in version 3.7.

--with-openssl-rpath`=[no|auto|DIR]`
Set runtime library directory (rpath) for OpenSSL libraries:

- `no` (default): don't set rpath;
- `auto`: auto-detect rpath from `--with-openssl` and `pkg-config`;
- *DIR*: set an explicit rpath.

New in version 3.10.

3.1.8. Security Options

--with-hash-algorithm`=[fnv|siphash24]`
Select hash algorithm for use in `Python/pyhash.c`:

- `siphash24` (default).
- `fnv`;

New in version 3.4.

--with-builtin-hashlib-hashes`=md5,sha1,sha256,sha512,sha3,blake2`
Built-in hash modules:

- `md5`;
- `sha1`;
- `sha256`;
- `sha512`;
- `sha3` (with shake);
- `blake2`.

New in version 3.9.

--with-ssl-default-suites`=[python|openssl|STRING]`
Override the OpenSSL default cipher suites string:

- `python` (default): use Python's preferred selection;

- `openssl`: leave OpenSSL's defaults untouched;
- *STRING*: use a custom string

See the `ssl` module.

New in version 3.7.

Changed in version 3.10: The settings `python` and *STRING* also set TLS 1.2 as minimum protocol version.

3.1.9. macOS Options

See `Mac/README.rst`.

`--enable-universalsdk`

`--enable-universalsdk``=SDKDIR`

Create a universal binary build. *SDKDIR* specifies which macOS SDK should be used to perform the build (default is no).

`--enable-framework`

`--enable-framework``=INSTALLDIR`

Create a Python.framework rather than a traditional Unix install. Optional *INSTALLDIR* specifies the installation path (default is no).

`--with-universal-archs``=ARCH`

Specify the kind of universal binary that should be created. This option is only valid when `--enable-universalsdk` is set.

Options:

- `universal2`;
- `32-bit`;
- `64-bit`;
- `3-way`;
- `intel`;
- `intel-32`;
- `intel-64`;
- `all`.

`--with-framework-name``=FRAMEWORK`

Specify the name for the python framework on macOS only valid when `--enable-framework` is set (default: `Python`).

3.2. Python Build System

3.2.1. Main files of the build system

- `configure.ac` => `configure`;
- `Makefile.pre.in` => `Makefile` (created by `configure`);
- `pyconfig.h` (created by `configure`);
- `Modules/Setup`: C extensions built by the Makefile using `Module/makesetup` shell script;
- `setup.py`: C extensions built using the `distutils` module.

3.2.2. Main build steps

- C files (`.c`) are built as object files (`.o`).
- A static `libpython` library (`.a`) is created from objects files.
- `python.o` and the static `libpython` library are linked into the final `python` program.
- C extensions are built by the Makefile (see `Modules/Setup`) and `python setup.py build`.

3.2.3. Main Makefile targets

- `make`: Build Python with the standard library.
- `make platform:`: build the `python` program, but don't build the standard library extension modules.
- `make profile-opt`: build Python using Profile Guided Optimization (PGO). You can use the configure `--enable-optimizations` option to make this the default target of the `make` command (`make all` or just `make`).
- `make buildbottest`: Build Python and run the Python test suite, the same way than buildbots test Python. Set `TESTTIMEOUT` variable (in seconds) to change the test timeout (1200 by default: 20 minutes).
- `make install`: Build and install Python.
- `make regen-all`: Regenerate (almost) all generated files; `make regen-stdlib-module-names` and `autoconf` must be run separately for the remaining generated files.
- `make clean`: Remove built files.
- `make distclean`: Same than `make clean`, but remove also files created by the configure script.

3.2.4. C extensions

Some C extensions are built as built-in modules, like the `sys` module. They are built with the `Py_BUILD_CORE_BUILTIN` macro defined. Built-in modules have no `__file__` attribute:

```
>>> import sys
>>> sys
<module 'sys' (built-in)>
>>> sys.__file__
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
AttributeError: module 'sys' has no attribute '__file__'
```

Other C extensins are built as dynamic libraries, like the `_asyncio` module. They are built with the `Py_BUILD_CORE_MODULE` macro defined. Example on Linux x86-64:

```
>>> import _asyncio
>>> _asyncio
<module '_asyncio' from '/usr/lib64/python3.9/lib-dynload/_asyncio.cpython-39-x86_64
>>> _asyncio.__file__
'/usr/lib64/python3.9/lib-dynload/_asyncio.cpython-39-x86_64-linux-gnu.so'
```

`Modules/Setup` is used to generate Makefile targets to build C extensions. At the beginning of the files, C extensions are built as built-in modules. Extensions defined after the `*shared*` marker are built as dynamic libraries.

The `setup.py` script only builds C extensions as shared libraries using the `distutils` module.

The `PyAPI_FUNC()`, `PyAPI_API()` and `PyMODINIT_FUNC()` macros of `Include/pyport.h` are defined differently depending if the `Py_BUILD_CORE_MODULE` macro is defined:

- Use `Py_EXPORTED_SYMBOL` if the `Py_BUILD_CORE_MODULE` is defined
- Use `Py_IMPORTED_SYMBOL` otherwise.

If the `Py_BUILD_CORE_BUILTIN` macro is used by mistake on a C extension built as a shared library, its `PyInit_xxx()` function is not exported, causing an `ImportError` on import.

3.3. Compiler and linker flags

Options set by the `./configure` script and environment variables and used by `Makefile`.

3.3.1. Preprocessor flags

CONFIGURE_CPPFLAGS

Value of `CPPFLAGS` variable passed to the `./configure` script.

New in version 3.6.

CPPFLAGS

(Objective) C/C++ preprocessor flags, e.g. `-I<include dir>` if you have headers in a nonstandard directory `<include dir>`.

Both `CPPFLAGS` and `LDFLAGS` need to contain the shell's value for setup.py to be able to build extension modules using the directories specified in the environment variables.

BASECPPFLAGS

New in version 3.4.

PY_CPPFLAGS

Extra preprocessor flags added for building the interpreter object files.

Default: `$(BASECPPFLAGS) -I. -I$(srcdir)/Include $(CONFIGURE_CPPFLAGS) $(CPPFLAGS)`.

New in version 3.2.

3.3.2. Compiler flags

CC

C compiler command.

Example: `gcc -pthread`.

MAINCC

C compiler command used to build the `main()` function of programs like `python`.

Variable set by the `--with-cxx-main` option of the configure script.

Default: `$(CC)`.

CXX

C++ compiler command.

Used if the `--with-cxx-main` option is used.

Example: `g++ -pthread`.

CFLAGS

C compiler flags.

CFLAGS_NODIST

`CFLAGS_NODIST` is used for building the interpreter and stdlib C extensions. Use it when a compiler flag should *not* be part of the distutils `CFLAGS` once Python is installed (bpo-21121).

In particular, `CFLAGS` should not contain:

- the compiler flag *-I* (for setting the search path for include files). The *-I* flags are processed from left to right, and any flags in `CFLAGS` would take precedence over user- and package-supplied *-I* flags.
- hardening flags such as *-Werror* because distributions cannot control whether packages installed by users conform to such heightened standards.

New in version 3.5.

EXTRA_CFLAGS
Extra C compiler flags.

CONFIGURE_CFLAGS
Value of `CFLAGS` variable passed to the `./configure` script.

New in version 3.2.

CONFIGURE_CFLAGS_NODIST
Value of `CFLAGS_NODIST` variable passed to the `./configure` script.

New in version 3.5.

BASECFLAGS
Base compiler flags.

OPT
Optimization flags.

CFLAGS_ALIASING
Strict or non-strict aliasing flags used to compile `Python/dtoa.c`.

New in version 3.7.

CCSHARED
Compiler flags used to build a shared library.

For example, `-fPIC` is used on Linux and on BSD.

CFLAGSFORSHARED
Extra C flags added for building the interpreter object files.

Default: `$(CCSHARED)` when `--enable-shared` is used, or an empty string otherwise.

PY_CFLAGS
Default: `$(BASECFLAGS) $(OPT) $(CONFIGURE_CFLAGS) $(CFLAGS) $(EXTRA_CFLAGS)`.

PY_CFLAGS_NODIST
Default: `$(CONFIGURE_CFLAGS_NODIST) $(CFLAGS_NODIST) -I$(srcdir)/Include/internal`.

New in version 3.5.

PY_STDMODULE_CFLAGS
C flags used for building the interpreter object files.

Default: `$(PY_CFLAGS) $(PY_CFLAGS_NODIST) $(PY_CPPFLAGS) $(CFLAGSFORSHARED)`.

New in version 3.7.

PY_CORE_CFLAGS

Default: `$(PY_STDMODULE_CFLAGS) -DPy_BUILD_CORE`.

New in version 3.2.

PY_BUILTIN_MODULE_CFLAGS

Compiler flags to build a standard library extension module as a built-in module, like the `posix` module.

Default: `$(PY_STDMODULE_CFLAGS) -DPy_BUILD_CORE_BUILTIN`.

New in version 3.8.

PURIFY

Purify command. Purify is a memory debugger program.

Default: empty string (not used).

3.3.3. Linker flags

LINKCC

Linker command used to build programs like `python` and `_testembed`.

Default: `$(PURIFY) $(MAINCC)`.

CONFIGURE_LDFLAGS

Value of `LDFLAGS` variable passed to the `./configure` script.

Avoid assigning `CFLAGS`, `LDFLAGS`, etc. so users can use them on the command line to append to these values without stomping the pre-set values.

New in version 3.2.

LDFLAGS_NODIST

`LDFLAGS_NODIST` is used in the same manner as `CFLAGS_NODIST`. Use it when a linker flag should *not* be part of the distutils `LDFLAGS` once Python is installed (bpo-35257).

In particular, `LDFLAGS` should not contain:

- the compiler flag *-L* (for setting the search path for libraries). The *-L* flags are processed from left to right, and any flags in `LDFLAGS` would take precedence over user- and package-supplied *-L* flags.

CONFIGURE_LDFLAGS_NODIST

Value of `LDFLAGS_NODIST` variable passed to the `./configure` script.

New in version 3.8.

LDFLAGS

Linker flags, e.g. `-L<lib dir>` if you have libraries in a nonstandard directory `<lib dir>`.

Both `CPPFLAGS` and `LDFLAGS` need to contain the shell's value for setup.py to be able to build extension modules using the directories specified in the environment variables.

LIBS

Linker flags to pass libraries to the linker when linking the Python executable.

Example: `-lrt`.

LDSHARED

Command to build a shared library.

Default: `@LDSHARED@ $(PY_LDFLAGS)`.

BLDSHARED

Command to build `libpython` shared library.

Default: `@BLDSHARED@ $(PY_CORE_LDFLAGS)`.

PY_LDFLAGS

Default: `$(CONFIGURE_LDFLAGS) $(LDFLAGS)`.

PY_LDFLAGS_NODIST

Default: `$(CONFIGURE_LDFLAGS_NODIST) $(LDFLAGS_NODIST)`.

New in version 3.8.

PY_CORE_LDFLAGS

Linker flags used for building the interpreter object files.

New in version 3.8.