Python Development Documentation

Release 1.0

Victor Stinner

Oct 29, 2020

Contents

by Victor Stinner.

Python:

CHAPTER 1

Python 3

Python 3.0 was released in December 2008. Python 2.7 was released in July 2010.

1.1 Python 2 or Python 3?

- pythonclock.org: "Python 2.7 will retire in..."
- Python 3 Statement

My article Why should OpenStack move to Python 3 right now? explains why you should move to Python 3 right now.

Python 2 will reach its end of life in 2020: see Python 2.7 Release Schedule. At Pycon US 2014, the support was extended from 2015 to 2020 to give more time to companies to port their applications to Python 3.

1.2 Port Python 2 code to Python 3

- six module
- 2to6
- Language differences and workarounds (python3porting book)
- Porting Python 2 Code to Python 3 (docs.python.org)
- python-porting mailing list
- getpython3.com
- Porting code to Python 3 (Python.org wiki)
- python-incompatibility
- 2to3c
- py3to2

- Python 3 Wall of Superpowers
- Can I Use Python 3?

1.3 Python 3 is better than Python 2

- 10 awesome features of Python that you can't use because you refuse to upgrade to Python 3

1.3.1 Comparison on unequal types compares the type names

Python 2 implements a strange comparison operator. For the instruction `a < b`, if `a.__cmp__(b)` and `b.__cmp__(a)` raise `NotImplementedError` or return `NotImplemented`, Python 2 compares the *name* of the types. Basically, `a < b` becomes `type(a).__name__ < type(b).__name__`. It's more complex than that, if a type is a number, the comparison uses an empty string as the type name.

Extract of the `default_3way_compare()` function of Python 2 `Objects/object.c`:

```
/* different type: compare type names; numbers are smaller */
if (PyNumber_Check(v))
    vname = "";
else
    vname = v->ob_type->tp_name;
if (PyNumber_Check(w))
    wname = "";
else
    wname = w->ob_type->tp_name;
c = strcmp(vname, wname);
```

Example in Python 2:

```
>>> [1, 2, 3] < "abc"
True
>>> type([1, 2, 3]).__name__, type("abc").__name__
('list', 'str')
>>> type([1, 2, 3]).__name__ < type("abc").__name__
True
```

As a proof of the behaviour, it's possible to use type subclasses to modify the type names:

```
>>> class z(list): pass
...
>>> class a(str): pass
...
>>> [1, 2, 3] < "abc"
True
>>> z([1, 2, 3]) < a("abc")
False
>>> type(z([1, 2, 3])).__name__, type(a("abc")).__name__
('z', 'a')
>>> type(z([1, 2, 3])).__name__ < type(a("abc")).__name__
False
```

Python 3 doesn't have this strange fallback in comparison. It now raises TypeError on this case:

```
>>> [1, 2, 3] < "abc"
TypeError: unorderable types: list() < str()
```

As a consequence, the builtin `cmp()` function was removed from Python 3. To sort a list, the `key` parameter of `list.sort()` must be used. By the way, on Python 2, using a *key* function (`list.sort(key=func)`) is more efficient than using a *cmp* function (`list.sort(cmp=func)`).

On Python 2.7, it's possible to enable Python 3 comparison using `-3 -Werror` command line options:

```
$ python2 -3 -Werror
>>> [1, 2, 3] < "abc"
DeprecationWarning: comparing unequal types not supported in 3.x
```

1.3.2 Bugs already fixed in Python 3

Some race conditions are already fixed in Python 3. The fix may be backported to Python 2, but it takes more time because the Python 3 branch diverged from the Python 2 branch, and Python core developer focus on Python 3.

- python RLock implementation unsafe with signals
- Locks cannot be interrupted by signals in Python 2: Condition.wait() doesn't raise KeyboardInterrupt
- subprocess is not thread-safe in Python 2:
  - file descriptor issue (see above)
  - subprocess.Popen hangs when child writes to stderr
  - Doc: subprocess should warn uses on race conditions when multiple threads spawn child processes

In Python 2, file descriptors are inherited by default in the subprocess module, close_fds must be set explicitly to True. A race condition causes two child processes to inherit a file descriptor, whereas only one specific child process was supposed to inherit it. Python 3.2 fixed this issue by closing all file descriptors by default. Python 3.4 is even better: now all file descriptors are not inheritable by default (PEP 446: Make newly created file descriptors non-inheritable).

1.4 Bugs that won't be fixed in Python 2 anymore

1.4.1 Unicode

The Unicode support of Python 3 is much much better than in Python 2. Many Unicode issues were closed as "won't fix" in Python 2, especially issues opened after the release of Python 3.0. Some examples:

- Outputting unicode crushes when printing to file on Linux
- stdout.encoding not set when redirecting windows command line output

1.4.2 Bugs in the C stdio (used by the Python I/O)

Python 2 uses the buffer API of the C standard library: `fopen()`, `fread()`, `fseek()`, etcThis API has many bugs. Python works around some bugs, but some others cannot be fixed (in Python). Examples:

- Issue #20866: Crash in the libc fwrite() on SIGPIPE (segfault with os.popen and SIGPIPE)
- Issue #21638: Seeking to EOF is too inefficient!
- Issue #1744752: end-of-line issue on Windows on file larger than 4 GB

- Issue #683160: Reading while writing-only permissions on Windows
- Issue #2730: file readline w+ memory dumps
- Issue #22651: Open file in a+ mode reads from end of file in Python 3.4
- Issue #228210: Threads using same stream blow up (Windows)

Python 3 has a much better I/O library: the `io` module which uses directly system calls like `open()`, `read()` and `lseek()`.

1.4.3 Hash DoS

The hash function of Python 2 has a "worst complexity" issue which can be exploited for a denial of service (DoS). It's called the "hash DoS" vulnerability. Python 3.3 randomizes the hash function by default, Python 2.7 can use randomized hash if enabled explicitly. But the real fix is in Python 3.4 with the PEP 456 which now uses the new SipHash hash function which is much safer.

1.4.4 subprocess

The subprocess module is written in pure Python in Python 2.7. There are complex race conditions. The correct fix was to reimplement the critical part in C, fix implemented in Python 3.

- subprocess.Popen hangs when child writes to stderr

See also the PEP 446: Make newly created file descriptors non-inheritable which also fixes a complex issues related to subprocesses, PEP implemented in Python 3.4.

Workaround: install the subprocess32 module from PyPI (and use it instead of subprocess).

1.4.5 No more polling (busy loop) in Lock.acquire(timeout)

In Python 3.2, locks got a new optional timeout parameter which uses the native OS function.

Extract of `threading._Condition.wait(timeout)` of Python 2.7:

```
def wait(self, timeout=None):
    ...
    # Balancing act:  We can't afford a pure busy loop, so we
    # have to sleep; but if we sleep the whole timeout time,
    # we'll be unresponsive.  The scheme here sleeps very
    # little at first, longer as time goes on, but never longer
    # than 20 times per second (or the timeout time remaining).
    endtime = _time() + timeout
    delay = 0.0005 # 500 us -> initial delay of 1 ms
    while True:
        gotit = waiter.acquire(0)
        if gotit:
            break
        remaining = endtime - _time()
        if remaining <= 0:
            break
        delay = min(delay * 2, remaining, .05)
        _sleep(delay)
    ...
```

Moreover, `subprocess.Popen.communicate()` also got a timeout parameter.

1.4.6 Monotonic clocks

Timeouts must not use the system clocks but a monotonic clock. It is explained in the PEP 418 which has been implemented in Python 3.3.

Example of issue with system clock changes: threading.Timer/timeouts break on change of win32 local time.

See also the PEP 418 for a list of issues related to the system clock.

1.4.7 Other bugs

Misc bugs:

- Destructor of ElementTree.Element is recursive
- Ctrl-C doesn't interrupt simple loop: require the new GIL introduced in Python 3.2

1.5 Python 2 is slower

- The C code base doesn't respect strict aliasing and so must be compiled with `-fno-strict-aliasing` (to avoid bugs when the compiler optimizes the code) which is inefficient. The structure of Python C type has been deeply rewritten to fix the root cause.
- Python 3 uses less memory for Unicode text thanks to the PEP 393: Flexible String Representation. Many operations on "ASCII" strings are faster on Python 3 than Python 2.

1.6 Port Python 3 code to Python 2

Notes based on my experience of porting Tulip to Python 2 (Trollius project).

- Remove keyword-only parameter: replace `def func(*, loop=None): ...` with `def func(loop=None): ...`
- `super()` requires the class and self, *and* the class must inherit from object
- A class must inherit explicitly from object to use properties and `super()`, otherwise `super()` fails with a cryptic "TypeError: must be type, not classobj" message.
- Python 2.6: `str.format()` doesn't support `{}`. For example, `"{} {}".format("Hello", "World")` must be written `"{0} {1}".format("Hello", "World")`.
- Replace `list.clear()` with `del list[:]`
- Replace `list2 = list.copy()` with `list2 = list[:]`
- Python 3.3 has new specialized `OSError` exceptions: `BlockingIOError`, `InterruptedError`, `TimeoutError`, etc. Python 2 has `IOError`, `OSError`, `EnvironmentError`, `WindowsError`, `VMSError`, `mmap.error`, `select.error`, etc.
- `raise ValueError("error") from None` should be replaced with `raise ValueError("error")`
- `memoryview` should be replaced with `buffer`

Major changes in between Python 2.6 and 3.3:

- `threading.Lock.acquire()` and `subprocess.Popen.communicate()` support timeout. A busy loop can be used for `threading.Lock.acquire()` (non-blocking call + sleep) in Python 2.

- `time.monotonic()` (3.3)
- set and dict literals
- `memoryview` object
- `collections.OrderedDict` (2.7, 3.1)
- `weakref.WeakSet` (2.7, 3.0)
- `argparse`
- Python 2 doesn't support `ssl.SSLContext` nor certificate validation
- `ssl` module: `SSLContext`, `SSLWantReadError`, `SSLWantWriteError`, `SSLError`
- Python 2 does not support `yield from` and does not support `return` in generators (3.3)
- Python 2 doesn't support the `nonlocal` keyword: use mutable types like list or dict instead (3.0)

New modules in the standard library between Python 2.6 and Python 3.3:

- concurrent.futures (3.2)
- faulthandler (3.3)
- importlib (3.1)
- ipaddress (3.3)
- lzma (3.3)
- tkinter.ttk (3.1)
- unittest.mock (3.3)
- venv (3.3)

Python 3.4 has even more modules:

- asyncio
- enum
- ensurepip
- pathlib
- selectors
- statistics
- tracemalloc

CHAPTER 2

Python Packaging

2.1 Python packaging

- Python Packaging User Guide (URL on readthedocs.io)
- pip documentation
- Python Wheels
- https://hynek.me/articles/conditional-python-dependencies/

2.2 Compile Python extensions on Windows

- Install Windows SDK
- Run "Windows SDK Command Prompt"
- Type:

```
setenv /x64 /release
set MSSDK=1
set DISTUTILS_USE_SDK=1
```

See also *Windows*.

2.3 Build a Python Wheel package on Windows

For Python 2.7, you need: Download Visual C++ Compiler for Python 2.7.

Steps:

- Install pip

- Install wheel using pip:

```
\python27\python.exe -m pip install wheel
```

- Run "Windows SDK Command Prompt"
- Setup the environment to build code in 64-bit mode (replace `/x64` with `/x86` for 32bit):

```
setenv /x64 /release
set MSSDK=1
set DISTUTILS_USE_SDK=1
```

- Go to your project
- Cleanup the project (is it really needed?):

```
del build\*
del dist\*
```

- Build the wheel and upload it:

```
\python27\python.exe setup.py bdist_wheel upload
```

Notes:

- To build a 32-bit wheel, you need 32-bit Python and configure the SDK using `/x86`.
- Python 2.7 requires the Windows SDK v7.0 because Python 2.7 is built using Visual Studio 2008 (MSVCR90). Python 3.3 is built using Visual Studio 2010.
- It looks like Python 3.3 doesn't need `MSSDK` and `DISTUTILS_USE_SDK` environment variables anymore.

See also *Windows*.

2.4 Python environment markers

https://wiki.openstack.org/wiki/Python3#Environment_markers

pip supports environment markers in requirements since pip 6.0, example of requirement:

```
six
futures; python_version < '3.2'
```

pip uses ";" (colon) separator but requires "; " (colon, space) if the requirement uses an URL. A space is added for readability (spaces are ignored).

Environment markers in extra requirements of setup.cfg:

```
[extras]
test =
    six
    futures :python_version < '3.2'
```

The separator is the ":" (colon), space is only used for readability (spaces are ignored).

Environment markers in extra requirements of setup.py:

```
expected_requirements = {
    "test:python_version < '3.2'": ['futures'],
    "test": ['six']
}
```

2.5 pip issues

- Upgrading pip3 replaces /usr/bin/pip with the Python 3 pip
- Once, I got two dist-info directories for pip (`ls /usr/lib*/python3.4/site-packages/pip-*.dist-info -d`) which broke `python3 -m venv`: `ensurepip` was unable to find the system pip and Fedora doesn't include bundled wheel packages of `ensurepip` in the `python3-libs` package
- With pip 7.0 and newer, `pip3 install Routes; pip2 install Routes` installs the Python 3 version of Routes on Python 2. pip3 creates a wheel package using 2to3 but Routes 2.1 announces universal wheel support which is wrong.
- Wheel caching doesn't work on pip 7.0, 7.0.1 and 7.0.2. It was fixed in pip 7.0.3.

2.6 ensurepip

- Debian doesn't provide ensurepip: https://bugs.debian.org/cgi-bin/bugreport.cgi?bug=732703
  - Random workaround: https://gist.github.com/uranusjr/d03a49767c7c307be5ed
- Fedora, random links:
  - https://github.com/fedora-python/rewheel/issues/2
  - https://github.com/fedora-python/rewheel/blob/master/python2-ensurepip-rewheel.patch

CHAPTER 3

Python Date and Time

See also the Datetime-SIG mailing list.

3.1 datetime module

- datetime documentation
- PEP 495 – Local Time Disambiguation: New **fold** attribute added by Python 3.6

3.2 Other modules

- ago
- Arrow
- Chronyk
- dateutil
- Delorean
- FreezeGun
- Maya
- Moment
- Pendulum
- PyTime
- pytz
- When.py

3.3 Articles

- Semantics of timezone-aware datetime arithmetic by Paul Ganssle
- A curious case of non-transitive datetime comparison by Paul Ganssle
- pytz: The Fastest Footgun in the West by Paul Ganssle

CHAPTER 4

Python Numbers

4.1 Number Types

- `int`: integral number
- `float`: floating point number, usually IEEE 754 (64 bits, base 2)
- `complex`: complex number, implemented as two floats
- `decimal.Decimal`: floating point number stored in base 10, arbitrary precision
- `fractions.Fraction`: rational, numerator/denominator; automatically compute the greatest common divisor (GCD) to simplify the fraction

4.2 Number Tower

The numbers module is specified by the PEP 3141 – A Type Hierarchy for Numbers.

- numbers.Number: base class
- numbers.Complex: Add `real`, `imag`, `conjugate()`
- numbers.Real: subclass of Complex; add many float operations.
- numbers.Rational: subclass of Real; add `numerator` and `denominator` attributes
- numbers.Integral: subclass of Rational

Subclasses:

- numbers.Number: int, float, complex, decimal.Decimal, fractions.Fraction
- numbers.Complex: int, float, complex, fractions.Fraction
- numbers.Real: int, float, fractions.Fraction
- numbers.Rational: int, fractions.Fraction

- numbers.Integral: int

int, float, complex methods and attributes:

- `conjugate()`
- `imag`
- `real`

int and Fraction attributes:

- `denominator`
- `numerator`

4.3 Conversions in Python

`int(obj)` and `float(obj)` accept:

- `int`
- `float`
- `complex`
- `decimal.Decimal`
- `fractions.Fraction`
- `bytes`
- `str`

`int(obj)` rounds towards zero (ROUND_DOWN, ex: `int(0.9) == 0` and `int(-0.9) == 0`).

But `int(obj)` and `float(obj)` reject:

- `complex`

`complex(obj)` accepts:

- `int`
- `float`
- `complex`
- `decimal.Decimal`
- `fractions.Fraction`
- `bytes`
- `str`

But `complex(obj)` rejects:

- `bytes`

`decimal.Decimal(obj)` accepts:

- `int`
- `float`
- `complex`

- `decimal.Decimal`
- `fractions.Fraction`
- `bytes`
- `str`

But `decimal.Decimal(obj)` rejects:

- `complex`
- `fractions.Fraction`
- `bytes`

`fractions.Fraction(obj)` accepts:

- `int`
- `float`
- `complex`
- `decimal.Decimal`
- `fractions.Fraction`
- `bytes`
- `str` (ex: `"1"` and `"1/2"`)

But `fractions.Fraction(obj)` rejects:

- `complex`
- `bytes`

4.4 int type

Examples:

```
>>> (123).bit_length()
7
>>> sys.int_info
sys.int_info(bits_per_digit=30,
             sizeof_digit=4)
```

Serialization as bytes:

```
>>> (123).to_bytes(4, 'little')
b'{\x00\x00\x00'
>>> int.from_bytes(b'{\x00\x00\x00', 'little')
123
```

Rounding:

- `int(float)` calls `float.__trunc__()`, so rounds towards zero (ROUND_DOWN)

4.5 float type

Examples:

```
>>> sys.float_info
sys.float_info(max=1.7976931348623157e+308,
               max_exp=1024,
               max_10_exp=308,
               min=2.2250738585072014e-308,
               min_exp=-1021,
               min_10_exp=-307,
               dig=15,
               mant_dig=53,
               epsilon=2.220446049250313e-16,
               radix=2,
               rounds=1)
>>> sys.float_repr_style
'short'
>>> (1.2).as_integer_ratio()
(5404319552844595, 4503599627370496)
```

Formatting as hexadecimal (base 16):

```
>>> (1.1).hex()
'0x1.199999999999ap+0'
>>> float.fromhex('0x1.199999999999ap+0')
1.1
```

Rounding:

- float.__trunc__(): Round towards zero (ROUND_DOWN)
- float.__round__(): Round to nearest with ties going to nearest even integer (ROUND_HALF_EVEN).
- float.__int__() is an alias to float.__trunc__() (ROUND_DOWN)

4.6 Fraction

```
>>> fractions.Fraction(5, 10)    # int / int
Fraction(1, 2)
>>> fractions.Fraction(1.2)    # float
Fraction(5404319552844595, 4503599627370496)
```

4.7 C API

Convert a Python object to an integer. Type methods:

- `__int__()`: slot `type->tp_as_number->nb_int`
- `__index__()`: slot `type->tp_as_number->nb_index`
- `__trunc__()` (no slot)

`PyNumber_Long(obj)`:

- Call `obj.__trunc__()` (Round towards zero, ROUND_DOWN)

- or: If `obj` type is bytes or str, parse the string in decimal (base 10)
- or: error!

`PyNumber_Index(x)` calls the `__index__()` method: raises an exception if the type has no `__index__()` method or if method does not return exactly an int type (*).

`_PyLong_FromNbInt(x)`:

- Call `type(x).__int__(x)`: the result type must be exactly the int type (*)
- or: error!

`_PyLong_FromNbIndexOrNbInt(x)`: `__index__()` or `__int__()`:

- return `x` if `type(x) == int` (`PyLong_CheckExact()`)
- call `type(x).__index__(x)` is defined: the result type must be exactly the int type (*)
- call `type(x).__int__(x)` (call `_PyLong_FromNbInt()`): the result type must be exactly the int type (*)
- error if it is not a int subclass, and the type don't define `__index__()` nor `__int__()` method
- New in Python 3.8

`PyLong_AsLong()` converts a Python `int` to a C `long`:

- call `_PyLong_FromNbIndexOrNbInt()`
- raise OverflowError if the result does not fit into a C long
- Python 3.7 and older only calls `__int__()`, not `__index__()`.

`PyLong_AsUnsignedLongMask()` converts a Python object to a C `unsigned long`:

- call `_PyLong_FromNbIndexOrNbInt(x)` and then `_PyLong_AsUnsignedLongMask()` on the result
- error if the object cannot be converted to an int by `_PyLong_FromNbIndexOrNbInt(x)`
- **mask integer overflow**
- Python 3.7 and older only calls `__int__()`, not `__index__()`.

(*) Special case: `__int__()` or `__index__()` return a int subclass. This feature is deprecated since Python 3.3 (see commit 6a44f6ee). This feature may be removed from Python 3.9: see bpo-17576.

4.8 PyArg_ParseTuple and Py_BuildValue

Reference documentation: Parsing arguments and building values.

- PyArg_ParseTuple is implemented in `Python/getargs.c`, core function: `convertsimple()`.
- Py_BuildValue is implemented in `Python/modsupport.c`, core function: `do_mkvalue()`.
- Functions behave differently if `PY_SSIZE_T_CLEAN` is defined:

  For all `#` variants of formats (`s#`, `y#`, etc.), the type of the length argument (int or `Py_ssize_t`) is controlled by defining the macro `PY_SSIZE_T_CLEAN` before including `Python.h`.

PyArg_ParseTuple formats:

- `"i"` (C `int`): `__index__()` or `__int__()`; call `PyLong_AsLong(obj)`, but explicitly rejects float using `PyFloat_Check(arg)`.

- `"l"` (C `long`): `__index__()` or `__int__()`; call `PyLong_AsLong()`, but explicitly rejects float using `PyFloat_Check(arg)`
- `"n"` (C `ssize_t`): `__index__()`; call `PyNumber_Index()` and then `PyLong_AsSsize_t()`. Exception on overflow.
- `"k"`: call `PyLong_AsUnsignedLongMask()` if it's an int or int subclass, error otherwise. **Mask integer overflow**.

Note: In Python 3.7 and older, `PyLong_AsLong()` only calls `__int__()`, not `__index__()`.

4.9 Script to update this page

number_tower.py:

```
from __future__ import print_function
import numbers
import warnings
import fractions
import decimal

warnings.simplefilter("error", DeprecationWarning)

VALUES = (
    ("int", 123),
    ("float", 1.5),
    ("complex", complex(0, 1.5)),
    ("decimal.Decimal", decimal.Decimal("1.1")),
    ("fractions.Fraction", fractions.Fraction(1, 7)),
    ("bytes", b"123"),
    ("str", "123"),
)

TYPES = (
    ("int", int),
    ("float", float),
    ("complex", complex),
    ("decimal.Decimal", decimal.Decimal),
    ("fractions.Fraction", fractions.Fraction),
)

for type_descr, num_type in TYPES:
    accepted = []
    rejected = []
    for value_descr, value in VALUES:
        try:
            num_type(value)
        except TypeError:
            rejected.append(value_descr)
        accepted.append(value_descr)
    if accepted:
        print("``%s`` accept:" % type_descr)
        print()
        for name in accepted:
            print("* ``%s``" % name)
        print()
    if rejected:
```

(continues on next page)

(continued from previous page)

```
        print("``%s`` reject:" % type_descr)
        print()
        for name in rejected:
            print("* ``%s``" % name)
        print()
print()

for tower_name, tower_type in (
    ("Number", numbers.Number),
    ("Complex", numbers.Complex),
    ("Real", numbers.Real),
    ("Rational", numbers.Rational),
    ("Integral", numbers.Integral),
):
    subclasses = []
    for type_descr, num_type in TYPES:
        if issubclass(num_type, tower_type):
            subclasses.append(type_descr)
    print("%s subclasses: %s" % (tower_name, ", ".join(subclasses)))
```

Development:

CHAPTER 5

Python Continuous Integration: Travis CI, AppVeyor, VSTS, Buildbots

5.1 The Night's Watch is Fixing the CIs in the Darkness for You

Buildbot Watch: The watchers of CPython's Continuous Integration

Members:

- Pablo Galindo Salgado
- Victor Stinner

The Night's Watch is Fixing the CIs in the Darkness for You by Pablo Galindo Salgado.

5.2 Python Test Suite

Python Test Suite can be found in Lib/test/ directory.

Python uses libregrtest test runner which is based on unittest with additional features:

- `-R 3:3` command to check for reference and memory leaks (don't detect PyMem_RawMalloc leaks, only detect PyMem_Malloc and PyObject_Malloc leaks)
- `-u` resources to enable extra tests. Examples:
- `-u cpu` enables tests which use more CPU
- `-u largefile` enables tests which use a lot of disk space

- `-m 4G` enables tests which use a lot of memory and specify that we allow Python tests to allocate up to 4 GiB.
- `-w` to re-run failed tests in verbose mode
- `-m`, `--fromfile`, `--matchfile` to select tests

See devguide documentation: https://devguide.python.org/runtests/

5.3 Buildbots links

CPython buildbots:

- Builders
- buildmaster-config (GitHub): Buildbot configuration
- Devguide: buildbots

5.4 Travis CI

- Travis CI: Build History
- CPython: Travis CI configuration (.travis.yml)
- https://docs.travis-ci.com/user/running-build-in-debug-mode/
- Travis CI Status

5.5 GitHub Actions

- Configuration lives in .github/workflows/ directory

5.6 Azure Pipelines PR

- Restart a job: close/reopen the PR
- Azure Pipelines configuration (.azure-pipelines/ directory)

5.7 Buildbots notifications

- IRC: #python-dev on Freenode
- Email: buildbot-status mailing list

5.8 Articles

- Work on Python buildbots, 2017 Q2

5.9 How to watch buildbots?

Email: [Python-Dev] How to watch buildbots?.

5.9.1 Report a failure

When a buildbot fails, I look at tests logs and I try to check if an issue has already been reported. For example, search for the test method in title (ex: "test_complex" for test_complex() method). If no result, search using the test filename (ex: "test_os" for Lib/test/test_os.py). If there is no result, repeat with full text searchs ("All Text"). If you cannot find any open bug, create a new one:

- The title should contain the test name, test method and the buildbot name. Example: " test_posix: TestPosixSpawn fails on PPC64 Fedora 3.x".
- The description should contain the link to the buildbot failure. Try to identify useful parts of tests log and copy them in the description.
- Fill the Python version field (ex: "3.8" for 3.x buildbots)
- Select at least the "Tests" Component. You may select additional Components depending on the bug.

If a bug was already open, you may add a comment to mention that there is a new failure: add at least a link to buildbot name and a link to the failure.

And that's all! Simple, isn't it? At this stage, there is no need to investigate the test failure.

To finish, reply to the failure notification on the mailing list with a very short email: add a link to the existing or the freshly created issue, maybe copy one line of the failure and/or the issue title.

Bug example: issue33630.

5.9.2 Analyze a failure

Later, you may want to analyze these failures, but I consider that it's a different job (different "maintenance task"). If you don't feel able to analyze the bug, you may try to find someone who knows more than you about the failure.

For better bug reports, you can look at the [Changes] tab of a build failure, and try to identify which recent change introduced the regression. This task requires to follow recent commits, since sometimes the failure is old, it's just that the test fails randomly depending on network issues, system load, or anything else. Sometimes, previous tests have side effects. Or the buildbot owner made a change on the system. There are many different explanation, it's hard to write a complete list. It's really on a case by case basis.

Hopefully, it's now more common that a buildbot failure is obvious and caused by a very specific recent changes which can be found in the [Changes] tab.

5.10 Revert on fail

- Policy to revert commits on buildbot failure
- [Python-Dev] Keeping an eye on Travis CI, AppVeyor and buildbots: revert on regression (May, 2018)
- [python-committers] Revert changes which break too many buildbots (June, 2017)

5.11 OLD: AppVeyor

It is no longer used by Python.

- AppVeyor: CPython build history
- CPython: AppVeyor configuration (.github/appveyor.yml)
- AppVeyor status page

CHAPTER 6

Python Debug Tools

Spoiler: Python 3.6 and newer provide a much better debugging experience!

See also *Debug CPython with gdb*.

6.1 Take Away

Command to enable debug tools:

- Python 3.7 and newer: `python3 -X dev`
- Python 3.6 and newer: `PYTHONMALLOC=debug python3 -Wd -X faulthandler`

Tools:

- faulthandler
- PYTHONMALLOC=debug: builtin memory debugger
- *tracemalloc*
- *importtime*

6.2 importtime

See *importtime*.

6.3 Get a traceback on a crash

Example of crash, `crash.py`:

```
import ctypes

def bug():
    ctypes.string_at(0)

bug()
```

Output:

```
$ python3 crash.py
Segmentation fault (core dumped)
```

... not very helpful :-(Enable faulthandler to get the Python traceback where the crash occurred. `python3 -X dev` (Python 3.7 and newer) enables automatically faulthandler.

faulthandler provides a traceback on a crash:

```
$ python3 -X faulthandler crash.py
Fatal Python error: Segmentation fault

Current thread 0x00007f3bb3998700 (most recent call first):
  File "/usr/lib64/python3.6/ctypes/__init__.py", line 487 in string_at
  File "crash.py", line 4 in bug
  File "crash.py", line 6 in <module>
Segmentation fault (core dumped)
```

To debug a deadlock, `faulthandler.dump_traceback_later()` can be implemented to implement a "watchdog": dump the traceback where Python is stuck if Python main code is blocked for longer than N seconds, and exit Python.

See also: Crash reporting in desktop Python applications by Nikhil Marathe and Max Bélanger (November 2018).

6.4 ResourceWarning

Example which doesn't close explicitly a file:

```
def func():
    f = open(__file__)
    f = None

func()
```

Output (or lack of output):

```
$ python3 filebug.py
```

... ResourceWarning warnings are hidden by default:

```
$ python3.6 -c 'import pprint, warnings; pprint.pprint(warnings.filters)'
[('ignore', None, <class 'DeprecationWarning'>, None, 0),
 ...
 ('ignore', None, <class 'ResourceWarning'>, None, 0)]
```

Use `python3 -X dev` (Python 3.7 and newer) or `python3 -Wd` (Python 3.6 and older) to display `ResourceWarning`:

```
$ python3 -Wd filebug.py
filebug.py:3: ResourceWarning: unclosed file <_io.TextIOWrapper name='filebug.py'␣
↪mode='r' encoding='UTF-8'>
  f = None
```

On Python 3.6 and newer, enabling *tracemalloc* shows where the resource (file in this example) has been created:

```
$ python3 -Wd -X tracemalloc=5 filebug.py
filebug.py:3: ResourceWarning: unclosed file <_io.TextIOWrapper name='filebug.py'␣
↪mode='r' encoding='UTF-8'>
  f = None
Object allocated at (most recent call first):
  File "filebug.py", lineno 2
    f = open(__file__)
  File "filebug.py", lineno 5
    func()
```

6.5 Debug memory errors

6.5.1 PYTHONMALLOC=debug

Memory managment in C is complex and error-prone.

Python has multiple allocators which are more or less compatible, but not always. For example, PyMem_Malloc() uses `malloc()` in Python 3.5 and older, but `pymalloc` in Python 3.6 and newer. Releasing memory allocated by `PyMem_Malloc()` using `PyObject_Free()` worked until Python 3.5, but "can" crash on Python 3.6 (depending if the memory block is longer than 512 bytes or not...).

Since Python 3.6, the new PYTHONMALLOC environment variable allows to change the memory allocator at runtime (when starting Python).

`PYTHONMALLOC=debug` enables Python builtin memory debugger: PyMem_SetupDebugHooks(). `python3 -X dev` (Python 3.7 and newer) enables automatically `PYTHONMALLOC=debug`.

Example `membug.py`:

```
import _testcapi

def main():
    _testcapi.pymem_buffer_overflow()

main()
```

Output:

```
$ PYTHONMALLOC=debug ./python membug.py
Debug memory block at address p=0x7f7c0ed9f160: API 'm'
    16 bytes originally requested
    The 7 pad bytes at p-7 are FORBIDDENBYTE, as expected.
    The 8 pad bytes at tail=0x7f7c0ed9f170 are not all FORBIDDENBYTE (0xfb):
        at tail+0: 0x78 *** OUCH
        at tail+1: 0xfb
        at tail+2: 0xfb
        at tail+3: 0xfb
        at tail+4: 0xfb
```

(continues on next page)

(continued from previous page)

```
        at tail+5: 0xfb
        at tail+6: 0xfb
        at tail+7: 0xfb
    The block was made by call #28431 to debug malloc/realloc.
    Data at p: cb cb cb cb cb cb cb cb cb cb cb cb cb cb cb cb

Fatal Python error: bad trailing pad byte

Current thread 0x00007f7c0ee875c0 (most recent call first):
  File "membug.py", line 4 in main
  File "membug.py", line 6 in <module>
Aborted (core dumped)
```

Python dumps the current traceback where the bug has been allocated, but it can be "too late".

On Python 3.6 and newer, enabling *tracemalloc* allows to find where the memory block has been allocated which can help to investigate the bug (truncated output to highlight the difference):

```
$ PYTHONMALLOC=debug ./python -X tracemalloc=5 membug.py
(...)
Memory block allocated at (most recent call first):
  File "membug.py", line 4
  File "membug.py", line 6
(...)
```

Traceback with source code recreated manually:

```
Memory block allocated at (most recent call first):
  File "membug.py", line 4
    _testcapi.pymem_buffer_overflow()
  File "membug.py", line 6
    main()
```

On this artificial example, the current Python traceback and memory block allocation traceback are the same, but usually they are different.

Sadly, on Python 3.5 and older, the only way to get the Python builtin memory allocator is to recompile Python (ex: using `./configure --with-pydebug` which changes the ABI...).

6.5.2 Valgrind

`PYTHONMALLOC=malloc valgrind python3 script.py` can also be used to debug C extensions which use directly `malloc()/free()`, and not `PyMem_Malloc()/PyMem_Free()`.

Use suppression file which can be found in Misc/valgrind.suppr

Link Misc/valgrind-python.supp of the master (development) branch.

6.6 gc.set_threshold(5)

https://mail.python.org/pipermail/python-dev/2018-June/153857.html

6.7 Debug functions

You might want to call these functions in a running process from gdb:

- _PyObject_Dump(obj)
- _PyUnicode_Dump(obj): dump properties of the Unicode object, not it's content
- PyErr_Occurred(): get the current exception, NULL if no exception has been raised
- if py-bt command is broken, try to call:
  - `_Py_DumpTraceback(2, tstate)`
  - `_Py_DumpTracebackThreads(2, interp, tstate)` where `tstate` can be `NULL`
  - Python 3.8: get `tstate` from `_PyRuntime.gilstate.tstate_current` and `interp` from `_PyRuntime.gilstate.autoInterpreterState`
  - `2` is the file descriptor 2: `stderr`

6.8 Core dump

Write core dumps on the current directory:

```
$ ulimit -c
unlimited
$ sudo bash -c 'echo "coredump-%e.%p" > /proc/sys/kernel/core_pattern'
```

Check that it works:

```
$ ./python -c 'import _testcapi, signal; _testcapi.raise_signal(signal.SIGABRT)'
Aborted (core dumped)
$ ls coredump*
coredump-python.23861

$ gdb ./python -c coredump-python.23861
GNU gdb (GDB) Fedora 8.0.1-36.fc27
(...)
Core was generated by `./python -c import _testcapi, signal; _testcapi.raise_
↪signal(signal.SIGABRT)'.
Program terminated with signal SIGABRT, Aborted.
(gdb) where
#0  0x00007fb0cb3ad050 in raise () from /lib64/libpthread.so.0
#1  0x00007fb0c3a53006 in test_raise_signal (self=<module at remote 0x7fb0cb624758>,
```

Ok, Python crashes generate coredump files and gdb is able to load them.

6.9 Windows

https://bugs.python.org/issue35418#msg331195

6.10 pudb

Put a breakpoint:

- hit 'm', search 'test_api' to open glance.tests.unit.test_api

6.11 tracemalloc

The tracemalloc module traces Python memory allocations. It can be used to find memory leaks, or just to have an accurate measure of the memory allocated by Python.

Usage:

- Write a scenario to reproduce the memory leak. The ideal is a scenario taking only a few minutes
- Enable tracemalloc and replay the scenario
- Take regulary tracemalloc snapshots
- Compare snapshots
- Enjoy!

If your application only uses Python memory allocators, tracemalloc must show your the exact memory usage counting every single bytes.

If a C extensions uses other memory allocators like `malloc()`, tracemalloc is unable to trace these allocations.

If the application allocates a lot of memory to process some data (memory peak) and then releases almost all memory, except a few small objects, the memory may become fragmented. For example, the application only uses 20 MB whereas the operating system see 24 or 30 MB.

See pytracemalloc: backport to Python 2.7 (need to patch and compile Python manually).

6.12 Debug crash in garbage collection (visit_decref)

It's really hard to investigate such crash. Usually a crash in the GC is only the symptom that something corrupted a Python object, and the crash can occur very late after the object has been corrupted.

You might attempt:

- Try python3 -X dev.
- Try Python compiled in debug mode.
- Try a more recent Python version. Maybe it's a bug in Python which is already fixed?
- List third party C extensions and look first at them. Usually, if you are the only one to see a crash, it comes from your bug. Maybe a C extension doesn't update reference counters correctly.
- Change GC thresholds: `gc.set_threshold(5)`. See [Python-Dev] Idea: reduce GC threshold in development mode (-X dev).
- Disable completely the GC: `gc.disable()`. It helped the reporter of bpo-2546 to find his bug.

Python issues related to visit_decref():

- 2019-10-07: Python segfaults when configured with –with-pydebug –with-trace-refs. `_PyObject_IsFreed()` regression, it detected `PyObject._ob_prev=NULL` or `PyObject._ob_next=NULL` as a bug (when Python is built using `./configure --with-pydebug --with-trace-refs`). For example, the `Py_None` object is not tracked by `sys.getobjects()` circular list. Parent object type: `dict`.
- 2019-10-07: Ensure that objects entering the GC are valid
- 2019-09-09: visit_decref(): add an assertion to check that the object is not freed
- 2019-09-05: reference counter issue in signal module. Missing `Py_INCREF()` in the initialization of the `_signal` module. Crash occurs while `_PyImport_Cleanup()` is clearing the `_signal` module. Parent object type: `dict` (`_signal` module namespace).
- 2019-03-21: Add gc.enable_object_debugger(): detect corrupted Python objects in the GC
- 2018-07-10: int(s), float(s) and others may cause segmentation fault. Buffer overflow in an `int` object. Parent object type: `list`.
- 2018-06-07: contextvars: hamt_alloc() must initialize h_root and h_count fields. `hamt_alloc()` tracks an object in the GC before it is fully initialized: `hamt_tp_traverse()` visits `h_root` which is not initialized yet. Parent object type: `hamt`.
- 2017-08-10: Segfault in gcmodule.c:360 visit_decref (PyObject_IS_GC(op)). Crash occurs in 3rd party project pyETL: no reproducer has been provided.
- 2016-07-17: Segfault in gcmodule.c:360 visit_decref. Related to pip, wheel and cffi. Parent object type: `dict`.
- 2014-02-06: python: Modules/gcmodule.c:379: visit_decref: Assertion '((gc)->gc.gc_refs >> (1)) != 0' failed. Crash at Python exit related to daemon threads spawned by asyncio. Also someone reported a bug in cx_Oracle, likely a corrupted exception: crash in visit_decref() called by BaseException_traverse(). Parent object type: `traceback`, visited object type: `Frame`.
- 2013-02-19: python-2.7.3-r3: crash in visit_decref(). Application using numpy, matplotlib, expat, and cElementTree. Parent object type: `tuple`.
- 2012-07-01: SEGFAULT in visit_decref. Reference counting issue. Parent object type: `tuple`.
- 2008-04-04: Python-2.5.2: crash in visit_decref () at Modules/gcmodule.c:270. Bug in a C extension, `char*` string passed as a `PyObject*`

CHAPTER 7

Debug CPython with gdb

7.1 Current Python thread state

Read `tstate_current`, if the GIL is held:

```
p ((PyThreadState*)_PyRuntime.gilstate.tstate_current)
```

Thread Local Storage (TLS):

```
p (PyThreadState *)PyThread_tss_get(&_PyRuntime.gilstate.autoTSSkey)
```

See also *Python Thread State*.

7.2 Put a breakpoint on the next exception

Put a breakpoint on the next exception: `break _PyErr_SetObject` (or `break PyErr_SetObject` in Python 3.7 and older).

Then use `condition` to only break at the expected exception.

7.3 Watch when reference count changes

Use a memory breakpoint like:

```
watch ((PyObject*)MEMORY_ADDRESS)->ob_refcnt
```

where `MEMORY_ADDRESS` is the address of a Python object.

7.4 Analyze a coredump

Current Python thread state, like `PyThreadState_GET()`:

```
p ((PyThreadState*)_PyRuntime.gilstate.tstate_current)
```

Current error, like `PyErr_Occurred()`:

```
p ((PyThreadState*)_PyRuntime.gilstate.tstate_current)->curexc_type
```

7.5 Load python-gdb.py

Load it manually:

```
(gdb) source /path/to/python-gdb.py
```

Add directory containing Python source code to "safe path", to automatically load python-gdb.py when debugging Python. Add the following line to your `~/.gdbinit`:

```
add-auto-load-safe-path ~/prog/
```

In my case, Python is in `~/prog/python/master`, but I chose to allow to load any Python script from `~/prog/`.

On Fedora, the script is provided as:

```
/usr/lib/debug/usr/lib64/libpython3.6m.so.1.0-3.6.6-1.fc28.x86_64.debug-gdb.py
```

CHAPTER 8

Unstable tests

The multiprocessing tests leaked a lot of resources. Victor Stinner and others fixed dozens of bugs in these tests.

See also: *Enable tracemalloc to get ResourceWarning traceback*.

8.1 How to write reliable tests

8.1.1 Don't use sleep as synchronization

Don't use a sleep as a synchronization primitive between two threads or two processes. It will later, soon or later.

- Threads: use threading.Event
- Processes: use a pipe (os.pipe()), write a byte when read, read to wait

8.1.2 Don't limit the maximum duration

Don't make a test fail if it takes longer than a specified number of seconds. Example:

```
t1 = time.monotonic()
func()
t2 = time.monotonic()
self.assertLess(t2 - t1, 60.0)  # cannot happen
```

Python has buildbot workers which are very slow where "cannot happen" does happen. In most cases, the maximum duration is not a bug in Python and so the test must not fail.

For example, test_time had a test to ensure that time.sleep(0.5) takes less than 0.7 seconds. The test started to fail on slow buildbots where it took 0.8 seconds: maximum extended to 1 second. The test has been modified later to no longer check the maximum duration.

- https://bugs.python.org/issue19999#msg206344

Another example, a sleep of 100 ms took 2 seconds on "AMD64 OpenIndiana 3.x" buildbot: https://bugs.python.org/issue20336

8.2 Debug race conditions

8.2.1 Debug test relying on time.sleep() or asyncio.sleep()

For example, test_asyncio: test_run_coroutine_threadsafe_with_timeout() has a race condition issue is caused by `await asyncio.sleep(0.05)` used in a test.

To reproduce the race condition, just use the smallest possible sleep of 1 nanosecond:

```
diff --git a/Lib/test/test_asyncio/test_tasks.py b/Lib/test/test_asyncio/test_tasks.py
index dde84b84b1..c94113712a 100644
--- a/Lib/test/test_asyncio/test_tasks.py
+++ b/Lib/test/test_asyncio/test_tasks.py
@@ -3160,7 +3160,7 @@ class RunCoroutineThreadsafeTests(test_utils.TestCase):

     async def add(self, a, b, fail=False, cancel=False):
         """Wait 0.05 second and return a + b."""
-        await asyncio.sleep(0.05)
+        await asyncio.sleep(1e-9)
         if fail:
             raise RuntimeError("Fail!")
         if cancel:
```

And run the test in a loop until it fails:

```
./python -m test test_asyncio -m test_run_coroutine_threadsafe_with_timeout -v -F
```

8.2.2 Debug Dangling process

For example, debug test_multiprocessing_spawn which logs:

```
Warning -- Dangling processes: {<SpawnProcess(QueueManager-1576, stopped)>}
```

https://bugs.python.org/issue38447

Get cases:

```
./python -m test test_multiprocessing_spawn  --list-cases > cases
```

Bisect:

```
./python -m test.bisect_cmd -i cases -o bisect1 -n 5 -N 500 test_multiprocessing_
↪spawn -R 3:3 --fail-env-changed
```

8.2.3 Debug reap_children() warning

For example, test_concurrent_futures logs such warning:

```
0:27:13 load avg: 4.88 [416/419/1] test_concurrent_futures failed (env changed) (17␣
↪min 11 sec) -- running: test_capi (7 min 28 sec), test_gdb (8 min 49 sec), test_
↪asyncio (23 min 23 sec)
beginning 6 repetitions
123456
.Warning -- reap_children() reaped child process 26487
.....
Warning -- multiprocessing.process._dangling was modified by test_concurrent_futures
  Before: set()
  After:  {<weakref at 0x7fdc08f44e30; to 'SpawnProcess' at 0x7fdc0a467c30>}
```

https://bugs.python.org/issue38448

Run the test in a loop until it fails?

```
./python -m test test_concurrent_futures --fail-env-changed -F
```

If it's not enough, spawn more jobs in parallel, example with 10 processes:

```
./python -m test test_concurrent_futures --fail-env-changed -F -j10
```

If it's not enough, use the previous commands, but also inject some workload. For example, run a different terminal:

```
./python -m test -u all -r -F -j4
```

Hack reap_children() to detect more issues, sleep 100 ms before calling waitpid(WNOHANG):

```
diff --git a/Lib/test/support/__init__.py b/Lib/test/support/__init__.py
index 0f294c5b0f..d938ae6b16 100644
--- a/Lib/test/support/__init__.py
+++ b/Lib/test/support/__init__.py
@@ -2320,6 +2320,8 @@ def reap_children():
     if not (hasattr(os, 'waitpid') and hasattr(os, 'WNOHANG')):
         return

+    time.sleep(0.1)
+
     # Reap all our dead child processes so we don't leave zombies around.
     # These hog resources and might be causing some of the buildbots to die.
     while True:
```

Untested function which might help, count the number of child processes of a process on Linux: Add support.get_child_processes().

8.2.4 Coredump in multiprocessing

FreeBSD buildbot workers were useful to detect crashes at Python exit, bugs related to dangling threads. It helps to add a random sleep at Python exit, in `Modules/main.c`.

8.3 Multiprocessing issues

8.3.1 Open

- 2018-07-20: multiprocessing.Pool and ThreadPool leak resources after being deleted

- 2017-07-19: Missing multiprocessing.queues.SimpleQueue.close() method (OPEN).

8.3.2 Fixed, rejected, out of date

- 2018-12-05, **multiprocessing**: test_multiprocessing_fork: test_del_pool() leaks dangling threads and processes on AMD64 FreeBSD CURRENT Shared 3.x
- 2018-07-18: test_multiprocessing_spawn: Dangling processes leaked on AMD64 FreeBSD 10.x Shared 3.x
- 2018-07-03: asyncio: BaseEventLoop.close() shutdowns the executor without waiting causing leak of dangling threads (FIXED in Python 3.9).
- 2018-05-28, **test_multiprocessing**: test_multiprocessing_fork: dangling threads warning (commit: call Pool.join)
- 2017-07-28: test_multiprocessing_spawn and test_multiprocessing_forkserver leak dangling processes (commit: remove Process.daemon=True, call Process.join)
- 2017-07-24, **multiprocessing**: multiprocessing.Pool should join "dead" processes (commit)
- 2017-07-09, **multiprocessing**: multiprocessing.Queue.join_thread() does nothing if created and use in the same process (commit)
- 2017-06-08, **multiprocessing**: Add close() to multiprocessing.Process
- 2017-05-03: Emit a ResourceWarning in concurrent.futures executor destructors (OUT OF DATE).
- 2017-04-26: Emit ResourceWarning in multiprocessing Queue destructor (REJECTED).
- 2016-04-15, **multiprocessing**: test_multiprocessing_spawn leaves processes running in background. **Add more checks** to _test_multiprocessing to **detect dangling processes and threads**.
- 2015-11-18, **multiprocessing**: test_multiprocessing_spawn ResourceWarning with -Werror (commit: use closefd=False)
- 2011-08-18: Warning – multiprocessing.process._dangling was modified by test_multiprocessing (commit: test_multiprocessing.py calls the terminate() method of all classes).

8.4 Python issues

8.4.1 Open issues

Search for `test_asyncio`, `multiprocessing` tests.

- 2019-06-11: test__xxsubinterpreters fails randomly

8.4.2 Fixed issues

- 2018-05-16, **socketserver**: socketserver: Add an opt-in option to get Python 3.6 behavior on server_close()
- 2017-08-18, **support**: Make support.threading_cleanup() stricter (**big issue with many fixes**)
- 2017-08-18, **test_logging**: test_logging: ResourceWarning: unclosed socket
- 2017-08-18, **socketserver**: socketserver.ThreadingMixIn leaks running threads after server_close()
- 2017-08-09, **socketserver**: socketserver.ForkingMixIn.server_close() leaks zombie processes

8.4.3 Rejected, Not a Bug, Out of Date

- 2016-03-25: Replace stdout and stderr with simple standard printers at Python exit

8.4.4 Windows handles

Abandonned attempt to hunt for leak of Windows handles:

- https://github.com/python/cpython/pull/7827 from https://bugs.python.org/issue18174
- https://github.com/python/cpython/pull/7966 from https://bugs.python.org/issue33966

8.5 Unlimited recursion

Some specific unit tests rely on the exact C stack size and how Python detects stack overflow. These tests are fragile because each platform uses a different stack size and behaves differently on stack overflow. For example, the stack size can depend if Python is compiled using PGO or not (depend on functions inlining).

`_Py_CheckRecursiveCall()` is a portable but not reliable test: basic counter using `sys.getrecursionlimit()`.

MSVC allows to implement `PyOS_CheckStack()` (`USE_STACKCHECK` macro is defined) using `alloca()` and catching `STATUS_STACK_OVERFLOW` error. If uses `_resetstkoflw()` to reset the stack overflow flag.

8.5.1 Tests

- test_pickle: test_bad_getattr()
- test_marshal: test_recursion_limit()

8.5.2 History

- 2019-04-29: macOS no longer specify stack size. Previously, it was set to 8 MiB (`-Wl,-stack_size,1000000`).
  - https://github.com/python/cpython/commit/883dfc668f9730b00928730035b5dbd24b9da2a0
  - https://bugs.python.org/issue34602
- 2018-07-05: test_marshal: "Improve tests for the stack overflow in marshal.loads()"
  - https://bugs.python.org/issue33720
  - https://github.com/python/cpython/commit/fc05e68d8fac70349b7ea17ec14e7e0cfa956121
- 2018-06-04: test_marshal: "Reduces maximum marshal recursion depth on release builds" on Windows
  - https://github.com/python/cpython/commit/2a4a62ba4ae770bbc7b7fdec0760031c83fe1f7b
  - https://bugs.python.org/issue33720
- 2014-11-01: MAX_MARSHAL_STACK_DEPTH sets to 1000 instead of 1500 on Windows
  - https://github.com/python/cpython/commit/f6c69e6cc9aac35564a2a2a7ecc43fa8db6da975
  - https://bugs.python.org/issue22734
- 2013-07-07: Visual Studio project (PCbuild) now uses 4.2 MiB stack, instead of 2 MiB

  - https://github.com/python/cpython/commit/24e33acf8c422f6b8f84387242ff7874012f7291
  - https://bugs.python.org/issue17206
- 2013-05-30: macOS sets the stack size to 8 MiB
  - https://github.com/python/cpython/commit/335ab5b66f432ae3713840ed2403a11c368f5406
  - https://bugs.python.org/issue18075
- 2007-08-29: test_marshal: MAX_MARSHAL_STACK_DEPTH set to 1500 instead of 2000 on Windows for debug build
  - https://github.com/python/cpython/commit/991bf5d8c8fdd94c3b9238d7111c0dfb41973804
  - https://bugs.python.org/issue1050

8.6 Notes

On FreeBSD, `sudo sysctl -w 'kern.corefile =%N.%P.core'` command can be used to include the pid in coredump filenames, since 2 processes can crash at the same time.

CHAPTER 9

External libraries and Generated files

9.1 Add a new C file

To add a C file (`.c`) or an header file (`.h`), you have to:

- Create `Python/config.c`
- Add it to `Makefile.pre.in`: in PYTHON_OBJS for example
- Add it to `PCbuild/pythoncore.vcxproj`
- Add it to `PCbuild/pythoncore.vcxproj.filters`

When creating a new directory, see also:

- For header files, see also `Tools/msi/dev/dev.wixproj` (read Steve Dower's comment).
- "make tags" and "make TAGS" in Makefile.pre.in
- The Windows installer copies Lib/test/ and subdirectories: see `<InstallFiles Include="$(PySourcePath)Lib\test\**\*" ...>` in `Tools/msi/test/test.wixproj`.

New subdirectories created in `Lib/` and `Lib/test/` must be added to `LIBSUBDIRS` of `Makefile.pre.in` (example).

9.2 Add a new subdirectory

If the subdirectory contains C code (`.c` files), it may be interesting to add it to `SUBDIRS` in `configure.ac`, so `make tags` will also parse these files to build the index used by some text editors.

9.3 Add a new C extension

- Update setup.py

- Update Modules/Setup

9.4 Generated files

- make regen-all
- autoconf: Regenerate `configure` from `configure.ac`
- `install-sh`: `rm install-sh; automake --add-missing --copy`, https://bugs.python.org/issue34765
- `Lib/distutils/command/wininst-*.exe` files are binaries of the `PC/bdist_wininst/` program (in Python source code).
- The frozen `__phello` module comes from `M____hello__` constant in `Python/frozen.c`. The bytecode is generated by `Tools/freeze/freeze.py Tools/freeze/flag.py`.
- `Python/dtoa.c` is based on http://www.netlib.org/fp/dtoa.c
- Unicode: Update the Unicode version in the `Tools/unicode/makeunicodedata.py` script, run it, and fix what fails (msg318935).
- `Lib/keyword.py`: https://bugs.python.org/issue36143

9.5 Vendored external libraries

Update dependencies: https://github.com/python/cpython-source-deps/blob/master/README.rst

See my external_versions.py script: external version of embedded libraries from CPython source code (locally).

On security branches, some dependencies are outdated because no more macOS nor Windows installer is built. It was decided to not upgrade outdated zlib 1.2.5 in Python 3.3.7, since it's specific to Windows, and no Windows user is expected to build his/her own Python 3.3 anymore.

- `Modules/_ctypes/libffi/`: copy of libffi
  - Removed from Python 3.7: https://bugs.python.org/issue27979
- `Modules/_ctypes/libffi_osx/`: libffi for macOS?
  - Version: `grep PACKAGE_VERSION Modules/_ctypes/libffi_osx/include/fficonfig.h`
  - Python 2.7-3.6 uses libffi 2.1
- `Modules/_ctypes/libffi_msvc/`: libffi for Windows (for Microsoft Visual Studio)?
  - Version: second line of `Modules/_ctypes/libffi_msvc/ffi.h`
  - Python 2.7-3.6 use libffi 2.0 beta, copied from ctypes-0.9.9.4 in 2006
- `Modules/expat/`: copy of libexpat
  - `./configure --with-system-expat`
  - Rationale: https://mail.python.org/pipermail/python-dev/2017-June/148287.html
  - Used on Windows and macOS, Linux distributions use system libexpat
  - Version: search for `XML_MAJOR_VERSION` in `Modules/expat/expat.h`
  - Script to update it: see attached script to https://bugs.python.org/issue30947

  - Recent update: https://bugs.python.org/issue30947
  - Python 2.7, 3.3-3.6 use libexpat 2.2.1
- `Modules/zlib/`: copy of zlib
  - Version: `ZLIB_VERSION` in `Modules/zlib/zlib.h`
  - Only used on Windows (system zlib is used on macOS and Linux)
  - Python zlib module not built if system zlib is older than 1.1.3
  - Script to update it: XXX
  - Recent update: https://bugs.python.org/issue29169
  - Python 2.7, 3.4 and 3.5, 3.6 use zlib 1.2.11
  - Python 3.3 uses zlib 1.2.5: https://github.com/python/cpython/pull/3108
- `Modules/_decimal/libmpdec/`: copy of libmpdec
  - Option: `./configure --with-system-libmpdec`
  - Included since Python 3.3 for _decimal
  - Maintained by Stefan Krah
  - Version: `MPD_VERSION` in `Modules/_decimal/libmpdec/mpdecimal.h`
  - Used on all platforms
  - Script to update: XXX
  - Recent update: https://bugs.python.org/issue26621
  - Python 3.6 uses libmpdec 2.4.2 (released at february 2016)
  - Python 3.4 and 3.5 uses libmpdec 2.4.1
  - Python 3.3 uses libmpdec 2.4.0
- Windows and macOS installers include OpenSSL (binary library)
  - Windows version: search for `openssl-` in `PCbuild/get_externals.bat`
  - macOS version: search for `openssl-` in `Mac/BuildScript/build-installer.py`
  - See: http://python-security.readthedocs.io/ssl.html#openssl-versions
  - See: https://www.python.org/dev/peps/pep-0543/
- Windows and macOS installers include SQLite
  - Recent update: https://bugs.python.org/issue28791
  - macOS: search for `SQLite` in `Mac/BuildScript/build-installer.py`
  - Windows: search for `sqlite-` in `PCbuild/get_externals.bat`

See also cpython-bin-deps and cpython-source-deps.

CHAPTER 10

Supported platforms and architectures

See also *Python on Android*.

10.1 Supported architectures

Well supported architectures:

- Intel x86 (32-bit) and x86_64 (64-bit, aka AMD64)

Best effort support architectures:

- ppc64le: should be well supported in practice
- AArch64
- ARMv7: should be well supported in practice
- s390x

10.2 Well supported platforms

Well supported platforms on Python 3.7 and 2.7:

- Linux
- Windows Vista and newer for Python 3.7, Windows XP and newer for Python 2.7
- FreeBSD 10 and newer
- macOS Leopard (macOS 10.6, 2008) and newer

It took 9 years to fix all compiler warnings on Windows 64-bit! Usually, the fix was to use a larger type to avoid a downcast. For example replace `int` with `Py_ssize_t`.

10.2.1 Linux

CPython still has compatibility code for Linux 2.6, whereas the support of Linux 2.6.x ended in August 2011, longer than 6 years ago.

Proposition in January 2018 to drop support for old Linux kernels: https://mail.python.org/pipermail/python-dev/2018-January/151821.html

10.2.2 Windows

- Windows Supported Versions in Python (in the Python development `master` branch)
- Windows Vista support dropped in Python 3.7
- Windows XP support dropped in Python 3.5
- Windows 2000 support dropped in Python 3.4
- bpo-23451, 2015-03: "Python 3.5 now requires Windows Vista or newer". See change1 and change2.
- Python 2.7 supports Windows XP and newer
- PEP 11 on Windows:

  > CPython's Windows support now follows [Microsoft product support lifecycle]. A new feature release X.Y.0 will support all Windows releases whose extended support phase is not yet expired. Subsequent bug fix releases will support the same Windows releases as the original feature release (even if the extended support phase has ended).

10.2.3 FreeBSD

- Python 3.7 dropped support for FreeBSD 9 and older.
- FreeBSD 9 buildbot wokers have been removed in 2017

10.2.4 macOS

- 2018-05-28: macOS Leopard (macOS 10.6, 2008) is currently the minimum officially supported macOS version.
- February 2018: Tiger (macOS 10.4, 2004) buildbots removed, which indirectly means that Tiger is no longer officially supported.

Tested by Travis CI and buildbots.

10.3 Best effort and unofficial platforms

Supported platform with best effort support:

- Android API 24
- OpenBSD
- NetBSD
- AIX 6 and newer (see bpo-42087 for AIX 5)

Platforms not supported officially:

- Cygwin
- HP-UX
  - HP-UX 11i v3 was first released in 2007. Latest HP-UX release: May 2020.
  - test_gdb fix in 2020: commit b2dca49ca3769cb60713f5c2b43e5d5bbdc1f9c7
  - os.chroot fix in 2020: commit a9edf44a2de9b23a1690b36cdfeed7b41ab763bd
- MinGW
- Solaris, OpenIndiana

Unofficial projects:

- Python for OpenVMS
- PythonD: PythonD is a 32-bit, multi-threaded, networking- and OpenGL-enabled Python interpreter for DOS and Windows.

10.4 Removed platforms

PEP 11 lists removal of supported platforms:

- Platforms without threading support removed in Python 3.7
  - https://bugs.python.org/issue31370
  - https://mail.python.org/pipermail/python-dev/2017-September/149156.html
  - https://github.com/python/cpython/commit/a6a4dc816d68df04a7d592e0b6af8c7ecc4d4344
- BSD/OS, 2017:
  - https://bugs.python.org/issue31624
  - https://github.com/python/cpython/commit/288d1daadaddf6ae35cf666138ba4b5d07449657
- MS-DOS: 2014: bpo-22591: Drop support of MS-DOS (DJGPP compiler), commit b71c7dc9
- Python 3.4: VMS, OS/2, Windows 2000
  - VMS: bpo-16136: Removal of VMS support, main removal commit (remove VMSError doc commit)
- IRIX ("The last major version of IRIX is IRIX 6.5, which was released in May 1998")
  - Python 3.7: Remove dynload_dl (2020)

10.5 I want CPython to support my platform!

In short, there are 2 conditions:

- the full test suite have to pass (`./python -m test` succeess)
- a CPython core developer has to be responsible of the platform to fix issues specific to this platform on *CIs*.

If it's not possible, the best option is to maintain a fork of CPython (fork of the Git repository) to maintain patches to top of the master branch (and maybe also patches on other branches).

More detail in the **PEP 11**.

10.6 C compilers

Python has a good support for:

- GCC
- clang
- Visual Studio MSC

Best effort:

- XLC on AIX 7

Compiler flags:

- Debug build uses -Og
- Release build uses -O3
- clang with LTO
- clang with LTO+PGO
- GCC with LTO
- GCC with LTO+PGO

See *Python Continuous Integration* to see exactly which C compilers and which compiler and linker flags are actually tested.

See also *Python builds*.

See PEP 7 for the minimum C standard version. In short, it's a subset of C99 with static line functions and `<stdint.h>`.

10.7 sys.platform versus os.name

Example of `sys.platform` and `os.name` values:

Platform	sys.platform	os.name
AIX	`aix` on Python3.8+, (**)	`posix`
FreeBSD	`freebsd5`, `freebsd6`, ...	`posix`
Linux	`linux` on Python 3, `linux2` on Python 2 (*)	`posix`
macOS	`darwin`	`posix`
NetBSD	`netbsd` (with a suffix?)	`posix`
OpenBSD	`openbsd5`	`posix`
Solaris	`sunos5`	`posix`
Windows	`win32`	`nt`

`sys.platform` comes from the `MACHDEP` variable which is built by the configure script using:

- `uname -s` command output converted to lowercase, with some special rules (ex: `linux3` is replaced with `linux` on Python 3)
- `uname -r` command output (or `uname -v` UnixWare or OpenUNIX)
- `$host` variable (`./configure --host=xxx` parameter) when cross-compiling

(*) `sys.platform` was also `linux3` on old versions of Python 2.6 and Python 2.7 with Linux kernel 3.x.

(**) On AIX `sys.platform` included a release digit, `aix3`, ..., `aix7` on all versions of Python through version Python 3.7.

CHAPTER 11

Downstream patches

Linux distributions use downstream patches to adapt Python for their needs, backport bugfixes and security fixes, etc.

- Debian
  - python3: debian/patches/ (2019-08-12: 41 files; Python 3.8.0b2)
- Fedora
  - python38 (2019-08-12: 7 patches; Python 3.8.0b3)
  - python3
  - python2
- Guix
  - http://git.savannah.gnu.org/cgit/guix.git/tree/gnu/packages/patches/python-3-deterministic-build-info.patch
  - http://git.savannah.gnu.org/cgit/guix.git/tree/gnu/packages/patches/python-3-fix-tests.patch
  - http://git.savannah.gnu.org/cgit/guix.git/tree/gnu/packages/patches/python-3-search-paths.patch
- FreeBSD
  - https://svnweb.freebsd.org/ports/head/lang/python38/files/
  - https://svnweb.freebsd.org/ports/head/lang/python37/files/
  - https://svnweb.freebsd.org/ports/head/lang/python36/files/
  - https://svnweb.freebsd.org/ports/head/lang/python27/files/

See also *Python on Android*.

CHAPTER 12

Python on Android

Summary:

- Python 2.7 and 3.5 can be used on Android using CrystaX NDK. It uses patches on Python.
- Python 3.6 (from python.org) mostly work on Android API 24 without further patches, but need recipes to be built and these recipes are not standardized yet.
- The goal is to get a full Android API 24 support for Python 3.7 using a standard build recipe; Android buildbot using qemu.

Xavier's "Android support" document (Decembre 2017): https://github.com/xdegaye/cagibi/blob/master/doc/android_support.rst

12.1 BeeWare VOC

A transpiler that converts Python bytecode into Java bytecode.

- http://beeware.org/voc
- https://github.com/beeware/voc

In 2019, BeeWare was awarded by the PSF a $50,000 grant to help improve Python on Android. Check out their blog and call for contractors here: https://beeware.org/news/buzz/beeware-project-awarded-a-psf-education-grant/.

See also https://github.com/beeware/ouroboros/

12.2 Android CI for Python

- Xavier's abifa
- https://mail.python.org/pipermail/python-dev/2017-December/151171.html
- Guido wants a PEP?

- https://bugs.python.org/issue30386
- https://github.com/python/cpython/pull/1629 (closed)

pmpp's plan: "Test on VMs with Android 4.4 (arm/i386) and Android 7 or 8 (all platforms)".

12.3 People

- Xavier de Gaye: was given push privileges on June 3, 2016 on the recommendation of Victor Stinner to pursue its work on porting Python on Android.
- Chih-Hsuan Yen aka *yan12125*
- Paul Peny aka *pmpp*: Panda3D contributor, wants to "port" Panda3D on Android (especially Python binding of Panda3D)

12.4 Android?

Android is not the common "Linux" system, only the Linux kernel is shared, everything else is different:

- Kernel: Linux
- libc: Bionic
- Android SDK
- NDK: toolchain used for cross-compilation, compiler, tools, headers and libraries
- Different filesystem layout and device mapper:
  - /system/bin/sh: Shell
  - /system/lib/libc.so: Bionic libc

The Android API version combines Linux kernel and bionic versions.

12.5 Bionic

- https://en.wikipedia.org/wiki/Bionic_(software)
- Broken locales in libc (but not in libstdc++)
- No shmem – see https://github.com/pelya/android-shmem

C++ stdlib has a working localeconv()!? Example:

```
#include <locale>

using namespace std;

static char decimal_point;
static char thousands_sep;
static lconv lc_cache ;

extern "C" {
    struct lconv *localeconv(void){
        decimal_point = std::use_facet<std::numpunct<char> >
↪(std::locale(std::setlocale(LC_ALL, NULL))).decimal_point();
```

(continues on next page)

(continued from previous page)

```
        lc_cache.decimal_point = &decimal_point;

        thousands_sep = std::use_facet<std::numpunct<char> >
↪(std::locale(std::setlocale(LC_ALL, NULL))).thousands_sep();
        lc_cache.thousands_sep = &thousands_sep ;

        return &lc_cache;
    }
}
```

12.6 Android, NDK and API versions

12.6.1 Android versions

https://en.wikipedia.org/wiki/Android_version_history :

Android version	Release	API versions
Android 7 (Nougat)	2016-08	24-25
Android 6 (Marshmallow)	2015-10	23
Android 5 (Lollipop)	2014-11	21-22
Android 4.4 (Kitkat)	2013-10	19-20

See also https://developer.android.com/about/dashboards

12.6.2 Supported API

- Python 3.7 currently targets API 21+ (but may have a partial support of API 19+)
- Unity supports API 16+
- Kivy supports API 19+
- Termux layer support python3.6+ and API 21+

Cheap devices only support old versions of Android. Cheap devices allow developers to work on Android without having to buy expensive devices just for this platforms.

XXX Python 3.7 should have a basic API 19+ support: fix compilation, but it's ok if some tests fail. Just push upstream existing patches for API 19.

12.6.3 API 19

- Basically, the full locale API is broken
- mmap() works but is not exported in libc headers

12.6.4 NDK

XXX what is NDK? :-)

NDK 14b is the first release to use "Unified headers".

12.7 Python on Android

- A lot of changes merged since 2016
- Python uses UTF-8 as its "filesystem encoding" and uses directly Python's codec rather than mbstowcs() and wcstombs()
- Python 3.7 added sys.getandroidapilevel(): API level used to *build* Python, not the runtime API version. `sys.getandroidapilevel()` mostly exists to implement the test "is Python running on Android?".

XXX should we change sys.platform from "linux" to "android" on Android?

Patches for API 19:

- https://github.com/pmp-p/droid-pydk/tree/master/sources.32

Build system and patches for API 21:

- https://github.com/yan12125/python3-android

12.8 Build Python for Android

Stdlib packed into a ZIP file.

12.8.1 Cross-compilation

Xavier's favorite option.

Drawback: pip cannot be used to install C extensions (see *pip*).

12.8.2 Build Python on Android

pmpp's favorite option.

12.8.3 Hackish option

pmpp's second choice.

- Link Python to a static libc on Linux using Android linker
- Extract object files from libpython.a and link again on Android

Drawback: broken DNS resolution.

12.9 Devices to develop Python on Android?

Devices:

- Raspberry PI 3: arm64

Software (Android):

- Lineage (ex-cyanogen)
- Android TV

12.10 APK package

Problem to solve for in apk use (standard android application, main is java based but embed libpython via Java native interface and a thread):

- libpython3.x.so must be unversionned and be located in <apkroot>/lib
- for cross-compiling and linking against libpython3.x.so, its soname must be set or set(IMPORTED_NO_SONAME ON) must be used in cmake, not ndk-build, linking would be broken.

Possible fix, apply patchelf after compilation: https://github.com/pmp-p/pydk/blob/3e87331d62fb80549b61cff561d192a594efec70/sources.aosp/python3.aosp.sh#L409

12.11 TTY on Android?

- Python REPL
- ncurses

See Terminal Emulator for Android (Google Play).

12.12 dlopen() RTLD_BIND_NOW

Bionic dlopen() doesn't support RTDL_LAZY. Dependencies must be loaded explicitly!

12.13 pip, MACHDEP, sysconfig

- https://bugs.python.org/issue32637 proposes to change sys.platform from "linux" to "android", but keep MACHDEP="linux".
- sysconfig: sysconfig data filename generated by Makefile using MACHDEP. Issue on cross-compilation. sysconfig uses sys.platform to recreate the module name at runtime.

If Python was cross-compiled, pip fails to build C extensions. The C compiler fails to locate Python header files.

12.14 SELinux

SELinux is enforced on arm64 since Android 5 (Lollipop).

12.15 CrystaX NDK

In short, CrystaX NDK is closer to a regular Linux glibc.

CrystaX NDK is a drop-in replacement for Google's NDK. Following are the main goals of CrystaX NDK:

- Better standard compatibility
- Easy porting of existing code to Android
- New features for Android native development

It provides Python 2.7 and Python 3.5, Python compiled with patches to support Android.

The Kivy project uses the Python of CrystaX. Kivy updated CrystaX Python get to Python 3.6:

- python-for-android (aka "p4a"): Turn your Python application into an Android APK. (GitHub python-for-android)
- buildozer, Generic Python packager for Android and iOS.

12.16 Cross-compilation

- Documentation: yan12125's comment on issue28833
- Xavier's abandonned PR: bpo-28833: Fix cross-compilation of third-party extension modules

Cross-compilation is used to target:

- Android on ARM and ARM64
- Android x86 and amd64.
- Android mips, but deprecated on newer toolchains.
- Intel 32-bit for Ubuntu multiarch: compilation done from x86-64 (64 bit) to x86 (32 bit)
- can be used for WASM (emscripten), and could be used for WASI (clang) targets.

Python stdlib C extensions are by default cross compiled as dynamic libraries, but they can't be garanteed to load because:

- their filenames don't start by "lib".
- their SONAME is not set.
- their default location is not <apkroot>/lib/ (only location allowed for most Android api) but is instead <prefix>/lib/python3.x/lib-dynload/
- on earlier python they were not linked to libpython (fixed since 3.7).
- LD_LIBRARY_PATH / LD_PRELOAD are not allowed from java user space.
- See: https://android.googlesource.com/platform/bionic/+/master/android-changes-for-ndk-developers.md

Third party C extensions have the same problem. Possible fix: apply patchelf after compilation, change filenames (avoiding collisions!), set correct soname, move files in same folder for android sdk builder (gradle) to pick libpython.so.

Cross-compiling third party C extension with setuptools/pip may need to set _PYTHON_SYSCONFIGDATA_NAME environment variable.

12.17 See also

- Python Everywhere? talk by Russell Keith-magee about **embedding Python** at PyCon Thailand 2019

CHAPTER 13

Python Development Workflow

See also *CPython infrastructure <infra>*.

13.1 GitHub bots

13.1.1 miss-islington

- Bug reports: https://github.com/python/miss-islington/issues
- Code: https://github.com/python/miss-islington
- https://github.com/miss-islington

Mariatta is the primary maintainer. The bot runs in Heroku.

The bot runs cherry-picker to backport changes in CPython.

CHAPTER 14

CPython infrastructure

See also *Python Development Workflow <workflow>*.

- Current organization owners
- Current repository administrators

14.1 Python infrastructure

- http://infra.psf.io/
- https://status.python.org/ Status of services maintained by the Python infra team
- https://github.com/python/psf-chef/
  - doc/nodes.rst
  - doc/roles.rst
- https://github.com/python/psf-salt/
- PSF pays a full-time sysadmin to maintain the Python infra: XXX
- https://www.python.org/psf/league/
- Managed services: http://infra.psf.io/overview/#details-of-various-services
- http://www.pythontest.net/ used by the test suite, see https://github.com/python/pythontestdotnet/

14.2 Services used by unit tests

- pythontest.net services:
  - pythontestdotnet: source of pythontest.net (resources used for Python test suite).
  - test_urllib2net: http://www.pythontest.net/index.html#frag, tcp/80 (HTTP)

  - FTP: ftp://www.pythontest.net/README
  - Copies of unicode text files like http://www.pythontest.net/unicode/EUC-CN.TXT
  - test_hashlib test files like http://www.pythontest.net/hashlib/blake2b.txt
  - test_httplib: self-signed.pythontest.net, tcp/443 (HTTPS)
  - test_robotparser: http://www.pythontest.net/elsewhere/robots.txt
  - test_socket: .pythontest.net
- snakebite.net:

```
# Testing connect timeout is tricky: we need to have IP connectivity
# to a host that silently drops our packets.  We can't simulate this
# from Python because it's a function of the underlying TCP/IP stack.
# So, the following Snakebite host has been defined:
blackhole = resolve_address('blackhole.snakebite.net', 56666)

# Blackhole has been configured to silently drop any incoming packets.
# No RSTs (for TCP) or ICMP UNREACH (for UDP/ICMP) will be sent back
# to hosts that attempt to connect to this address: which is exactly
# what we need to confidently test connect timeout.

# However, we want to prevent false positives.  It's not unreasonable
# to expect certain hosts may not be able to reach the blackhole, due
# to firewalling or general network configuration.  In order to improve
# our confidence in testing the blackhole, a corresponding 'whitehole'
# has also been set up using one port higher:
whitehole = resolve_address('whitehole.snakebite.net', 56667)
```

- news.trigofacile.com:
  - Used by test_nntplib
  - NNTP (tcp/119) and and NNTP/SSL (tcp/563)
  - Server administrator: julien@trigofacile.com
- ipv6.google.com:
  - test_ssl uses it to test IPv6: HTTP (tcp/80) and HTTPS (tcp/443)
- sha256.tbs-internet.com:
  - test_ssl uses it to test x509 certificate signed by SHA256: HTTPS (tcp/443)

14.3 python.org

- https://github.com/python/pythondotorg/ Source code of python.org
- wiki.python.org runs MoinMoin

14.4 Package Index (PyPI)

- https://pypi.org/ "Warehouse", the new Python Package Index,
  - https://github.com/pypa/warehouse

- https://pypi.python.org/ "Python Cheeseshop", the old Python Package Index
- Python CDN: http://infra.psf.io/services/cdn/

14.5 cpython GitHub project

- https://github.com/python/cpython/
- GitHub uses mention-bot: https://github.com/facebook/mention-bot
  - https://github.com/mention-bot/how-to-unsubscribe
  - userBlacklist, userBlacklistForPR in CPython .mention-bot
  - Adding you GitHub login to userBlacklistForPR stops the mention bot from mentioning anyone on your PRs.
- https://github.com/python/core-workflow/tree/master/cherry_picker/
- Check Python CLA (service run by Mariatta Wyjaya)

14.6 Misc

- http://bugs.python.org/ Bug tracker (modified instance of Roundup)
  - https://pypi.python.org/pypi/roundup
  - Report Tracker Problem: https://github.com/python/psf-infra-meta/issues
  - Source code: https://bitbucket.org/python/tracker-cpython/ (fork of Roundup)
  - Special links:
    - edit Components
    - edit Versions
- Mailing lists: https://mail.python.org/mailman/listinfo
  - python-dev
  - python-ideas
  - python-list
  - lot of Special Interest Groups (SIG)
  - etc.
- http://buildbot.python.org/
  - CPython 3.6
  - CPython 3.x (master)
  - Custom builders
  - buildmaster-config (configuration)
  - Fork of BuildBot running on buildbot.python.org
- GitHub CLA bot: XXX

14.7 Documentation

- https://docs.python.org/ Python online documentation
- https://github.com/python/docsbuild-scripts/
- Mirror: http://python.readthedocs.io/en/latest/ Still use the old Mercurial repository.
- https://www.python.org/dev/peps/pep-0545/ i18n doc

14.8 IRC bots

IRC bots on #python-dev:

- `github`: bot run by GitHub. Source code: github-services: lib/services/irc.rb. GitHub services are deprecated since April 2018.
- `py-bb`: buildbot IRC bot, see buildmaster-config (buildbot configuration).
- `irker007`: Roundup bot.
  - https://hg.python.org/tracker/python-dev/file/tip/detectors/irker.py
  - http://www.catb.org/esr/irker/

14.9 Roundup: bugs.python.org

- https://github.com/python/bugs.python.org/
- https://hg.python.org/tracker/roundup/
- https://hg.python.org/tracker/python-dev/

CHAPTER 15

Python Memory

See also *tracemalloc*.

15.1 TODO

- Compute fragmentation of pymalloc
- Compute fragmentation of the glibc heap
- Tool to visualize the fragmentation?

15.2 Benchmarks

Issue #13483: http://bugs.python.org/file26069/tuples.py

15.3 Memory Fragmentation

- Improving Python's Memory Allocator, Evan Jones: http://www.evanjones.ca/memoryallocator/
- MemoryError when using highlight in hgweb: http://bz.selenic.com/show_bug.cgi?id=3005

15.4 Python Memory Allocators

- PyMem_RawMalloc(), PyMem_RawRealloc(), PyMem_RawFree(): new in Python 3.4
- PyMem_Malloc(), PyMem_Realloc(), PyMem_Free()
- PyObject_Malloc(), PyObject_Realloc(), PyObject_Free()

Current allocators:

- PyMem_RawMalloc(): system malloc()
- PyMem_Malloc(): system malloc()
- PyObject_Malloc(): pymalloc

http://www.python.org/dev/peps/pep-0445/

15.5 Windows

- VirtualAlloc
- Windows Low Fragementation Heap (LFH)
- Python: http://bugs.python.org/issue13483

Links:

- http://smallvoid.com/article/winnt-memory-decommit.html

15.6 pymalloc

- Used by PyObject_Malloc()
- Threshold of 512 bytes
- Arenas of 256 KB

Arena allocator:

- Windows: VirtualAlloc(NULL, size, MEM_COMMIT | MEM_RESERVE, PAGE_READWRITE);
- UNIX: mmap(NULL, size, PROT_READ|PROT_WRITE, MAP_PRIVATE|MAP_ANONYMOUS, -1, 0);
- Other: malloc(size);

15.7 Linux

/proc/self/statm:

```
size       total program size
           (same as VmSize in /proc/[pid]/status)
resident   resident set size
           (same as VmRSS in /proc/[pid]/status)
share      shared pages (from shared mappings)
text       text (code)
data       data + stack
```

http://linux-mm.org/Low_On_Memory

/proc, https://www.kernel.org/doc/Documentation/filesystems/proc.txt:

```
/proc/buddyinfo
/proc/pagetypeifo
/proc/slabinfo => slabtop program
/proc/meminfo
/proc/vmallocinfo
```

CHAPTER 16

Compile CPython on Windows

http://bugs.python.org/issue30350

16.1 People

People who understands these things.

- Steve Dower: employed by Microsoft, he is the maintainer of the Windows installer for 2.7, 3.6, 3.7 and master branches
- Zachary Ware
- Jeremy Kloth knows the `PC\VS9.0\` directory of Python 2.7!

16.2 Build a Windows VM

- Windows 10 or newer is recommended. Get a "multi-version" of Windows 10 (no N, KN or VL variant) and use a "Pro - Retail" product key.
- At least 60 GB of disk space is needed: Windows needs 15-20 GB, Visual Studio needs 10 GB. Having at least 60 GB is recommended to be able to compile multiple Python versions, install other tools, upgrade Windows, etc.

16.3 Python and Visual Studio version matrix

For Python 3.7 and later, VS 2017 is recommended (Community Edition is plenty), minimum installer options:

- Workload: only [x] "Desktop development with C++"
- Language pack: [x] "English"

Python version	Visual Studio
Python 2.7	VS 2008 **and** VS 2010
Python 3.4	VS 2010
Python 3.5 and 3.6	VS 2015 (or newer)
Python 3.7	VS 2017
Python 3.8 (master)	VS 2017

python.exe binaries delivered by python.org:

- Python 2.7.x (64-bit): [MSC v.1500 64 bit (AMD64)] on win32 – VS 2008
- Python 3.4.x (64-bit): [MSC v.1600 64 bit (AMD64)] on win32 – VS 2010
- Python 3.5.x (64-bit): [MSC v.1900 64 bit (AMD64)] on win32 – VS 2015
- Python 3.6.x (64-bit): [MSC v.1900 64 bit (AMD64)] on win32 – VS 2015
- Python 3.7.x (64-bit): [MSC v.1914 64 bit (AMD64)] on win32 – VS 2017

16.4 Dependencies

Python master needs binary dependencies from github.com/python/cpython-bin-deps and source dependencies from github.com/python/cpython-source-deps. `PCbuild\get_externals.bat`, which is called automatically by `PCbuild\build.bat`, will take care of fetching these for you if you have any of Git, Python, NuGet, or PowerShell available.

See `PCbuild\get_externals.bat` to learn what to put where if you aren't using `PCbuild\build.bat`.

16.5 Compile the master branch

To build the Python interpreter and all extension modules, including the ssl extension:

Requirements:

- Visual Studio 2015 or newer (VS 2017 recommended)
- CPython source code: get it using Git, or download a ZIP on GitHub.com

Compile 64-bit Debug Python in the command line:

```
PCBuild\build.bat -p x64 -d
```

Compile Python in the IDE: open the `PCbuild\pcbuild.sln` solution in Visual Studio.

See also: `PCbuild\readme.txt`.

16.6 Compile CPython 2.7

Python 2.7 is stuck forever on Visual Studio 2008 to not break the ABI, to keep the backward compatibility with all built extensions on the Python cheeseshop (PyPI). Obtaining VS 2008 is not nearly as simple or straightforward as it used to be and Python 2.7 is rapidly approaching the end of its support period. If you don't absolutely have to, we recommend not bothering to set things up to build 2.7!

16.6.1 Compile CPython 2.7 on Windows using Visual Studio 2008 and 2010

While Visual Studio 2008 alone is enough to build a full Python 2.7 binary with all standard extension modules, the standard method used to build 2.7 now requires installing both Visual Studio 2008 **and** Visual Studio 2010.

Requirements:

- MSDN account to get Visual Studio 2008. Maybe it's possible to build Python using the Express edition of VS 2008 and 2010, but in 2017, it became difficult to get VS 2008 and 2010 Express.
- Windows 10 or newer is recommended, even if Python 2.7 is supposed to support Windows XP!
- Visual Studio 2008 Professional. Visual Studio 2008 Express works too, but doesn't provide a 64-bit compiler.
- Visual Studio 2010 Professional. Maybe a lighter flavor works, I didn't try.

Compile 64-bit Debug Python in the command line:

```
PCBuild\build.bat -p x64 -d
```

Compile Python in the IDE: open the `PCbuild\pcbuild.sln` solution in Visual Studio.

See also: `PCbuild\readme.txt`.

16.6.2 Compile CPython 2.7 on Windows using only Visual Studio 2008

Similar to the previous section, but don't install Visual Studio 2010: only install Visual Studio 2008.

Compile 64-bit Debug Python in the command line:

```
PC\VS9.0\build.bat -p x64 -d -e
```

Compile Python in the IDE: open the `PC\VS9.0\pcbuild.sln` solution in Visual Studio.

See also: `PC\VS9.0\readme.txt`.

Note that this configuration is not well tested. Everything *should* work, but if it does not, please feel free to submit a patch! Also, if you use both methods and notice significant differences between them, we'd like to hear about those as well.

16.7 Windows Subsystem for Linux: WSL

Ubuntu running on Windows 10 using a thin layer to emulate the Linux kernel on top of the Windows kernel. Building Python in this environment is just the same as building on any other UNIXy system.

CHAPTER 17

Price of the portability

17.1 Misc

- Timers: PEP 418
- Inheritance of file descriptors and handles: PEP 446 (and PEP 433)
- Timezone
- SSL/TLS x509 root certificates
- use recent OS features: detected in configure, detected at runtime
- doc: “Availability: xxx”
- os.confstr(), os.sysconf()
- os.sysconf(“SC_PAGESIZE”)
- os.cpucount()
- OSError, VMSError, WindowsError, IOError => OSError
- os.closerange()
- os.sendfile(): different API depending on the OS!
- time.strftime()
- tzdata

17.2 POSIX vs real world

- Linux adds many new features not part of the POSIX standards
- (Systemd uses new Linux-only features like cgroups, not portable on FreeBSD, OpenBSD, etc.)
- _GNU_SOURCE: asprintf(), canonicalize_file_name()

- realpath(NULL)
- “Undefined” parts of POSIX standards: kept for efficiency?
- AIX

17.3 Windows

- POSIX API
- POSIX API has fewer features
- Windows console
- Windows ANSI, OEM and console code pages
- Atomic rename file, os.replace()
- One binary for all Windows versions: use LoadLibrary()
- os.fsencode() error handler and Windows versions
- Socket: no file descriptor, SOCKET_T

17.4 UNIX

- accept4() returns ENOSYS
- open(O_CLOEXEC) and socket(SOCK_CLOEXEC) on old Linux kernels

17.5 select, pipes

- test_signal: “OS doesn’t report write() error on the read end of a pipe”
- select() limited to sockets on Windows
- select(), poll(), epoll(), devpoll(), kqueue(), etc. => selectors => asyncio
- async I/O operations: no Python API yet?
- select.poll()
  - ifdef HAVE_BROKEN_POLL
  - #ifdef __APPLE__ select_have_broken_poll()

17.6 Threads and signals

- Hard
  - low-level OS functions, syscalls
  - threads
  - signals
  - fork, exec

- many tests skipped on old versions of operating systems
- OpenBSD older than 5.2 implemented threads in user-space
- FreeBSD older than 7.0
  - "Issue #12392 and #12469: send a signal to the main thread doesn't work before the creation of the first thread on FreeBSD 6"
- "Issue #18238: sigwaitinfo() can be interrupted on Linux (raises InterruptedError), but not on AIX"
- Signal orders
- HP-UX11
- depending on the OS, a signal sent to the pid is received by the mainthread or a random thread
- timeout on locks: drop support for thread APIs different than pthread and nt
- pthread_sigmask()
- pthread_kill()

CHAPTER 18

C hacks in CPython source code

18.1 Detect integer overflows

Before memory allocation.

In overallocate:

```
if (newlen <= (PY_SSIZE_T_MAX - newlen / 4))
    newlen += newlen / 4;
```

18.2 Avoid undefined behaviour

Issue #7406:

```
-    x = a + b;
+    /* casts in the line below avoid undefined behaviour on overflow */
+    x = (long)((unsigned long)a + b);
```

18.3 Usage of goto

Something like try/finally.

18.4 Optimisations

Unicode.

18.4.1 stringlib

- fastsearch.h: Bloom filter

18.5 Computed goto

Python/ceval.c

18.6 Python types

- (size_t,) PY_SIZE_MAX
- Py_ssize_t, PY_SSIZE_T_MAX
- PY_LONG_LONG (#ifdef HAVE_LONG_LONG)

18.7 Type aliasing

PyCompactUnicodeObject:

```
typedef struct {
    PyASCIIObject _base;
    Py_ssize_t utf8_length;     /* Number of bytes in utf8, excluding the
                                 * terminating \0. */
    char *utf8;                 /* UTF-8 representation (null-terminated) */
    Py_ssize_t wstr_length;     /* Number of code points in wstr, possible
                                 * surrogates count as two code points. */
} PyCompactUnicodeObject;
```

18.8 Macros

- Add assertions using “a,b” syntax. Ugly example:

```
#define PyUnicode_READY(op)                     \
    (assert(PyUnicode_Check(op)),               \
     (PyUnicode_IS_READY(op) ?                  \
      0 : _PyUnicode_Ready((PyObject *)(op))))
```

- Py_SAFE_DOWNCAST
- Py_ARRAY_LENGTH(array): check at compile time if the argument is an array
- do { ... } while (0), ex: Py_DECREF()

Examples:

```
#define Py_RETURN_TRUE return Py_INCREF(Py_True), Py_True
#define PyBool_Check(x) (Py_TYPE(x) == &PyBool_Type)
```

18.9 Memory allocation

- pymalloc
- *free list*
- overallocate
  - list: 12.5% new_allocated = (newsize >> 3) + (newsize < 9 ? 3 : 6);
  - string formatting: 25% newlen += newlen / 4;
  - depend on the performance of realloc(): fast on Linux, slow on Windows (older than Windows 7)
- Unicode: PyUnicodeWriter, PyAccu
- preallocated object: MemoryError instances

18.10 dict

- hash()
- hash vulnerability

CHAPTER 19

Bugs found by the Python project

CPython has an extensive test suite which sometimes spot bugs in third party code.

19.1 Kernel bugs

- Crash with mmap and sparse files on Mac OS X
- test_os on Linux 2.6.?35? return code
- Many issues with threads+signals on OpenBSD and old versions of FreeBSD: test disabled on these platforms
- FreeBSD: when close(fd) on a fifo fails with EINTR, the file descriptor is not really closed

19.2 Compiler bugs

- ppc64le double to float cast + memcpy bug
  - https://bugs.python.org/issue35752
  - https://gcc.gnu.org/bugzilla/show_bug.cgi?id=88892
- Visual Studio: PGO on 64 bits
- Visual Studio PGupdate duplicated functions:
  - "Disable COMDAT folding in Windows PGO builds."
  - http://bugs.python.org/issue8847#msg166935
  - http://hg.python.org/cpython/rev/029cde4e58c5
- Error with gcc-4.6 -O1 -ftree-vectorize
  - "Python 3.2 doesn't compile correctly with -O3"
  - http://gcc.gnu.org/ml/gcc-help/2011-01/msg00136.html

  - http://gcc.gnu.org/bugzilla/show_bug.cgi?id=47271
  - http://gcc.gnu.org/viewcvs?view=revision&revision=169233
- Apple Clang bug
  - llvm-gcc-4.2 miscompiles Python (XCode 4.1 on Mac OS 10.7)

CHAPTER 20

Python Finalization

At exit, Python calls `Py_Finalize()` which is responsible to stop Python and cleanly all states, variables, modules. etc. This code is very fragile.

20.1 Issues with clearing memory at exit

Since Python 3.6, a big refactoring started in CPython to clear more and more variables and states at Python exit: in `Py_FinalizeEx()`, but also in Python `main()` (for variables which cannot be cleared in `Py_FinalizeEx()`.

Sadly, it caused some nasty issues:

- The os module should unset() environment variable at exit
- Make struct module PEP-384 compatible

20.2 Prevent deadlock in io.BufferedWriter

https://bugs.python.org/issue23309

Commit:

```
commit 25f85d4bd58d86d3e6ce99cb9f270e96bf5ba08f
Author: Antoine Pitrou <solipsis@pitrou.net>
Date:   Mon Apr 13 19:41:47 2015 +0200

    Issue #23309: Avoid a deadlock at shutdown if a daemon thread is aborted
    while it is holding a lock to a buffered I/O object, and the main thread
    tries to use the same I/O object (typically stdout or stderr).  A fatal
    error is emitted instead.
```

Code:

```
relax_locking = _Py_IsFinalizing();
Py_BEGIN_ALLOW_THREADS
if (!relax_locking)
    st = PyThread_acquire_lock(self->lock, 1);
else {
    /* When finalizing, we don't want a deadlock to happen with daemon
     * threads abruptly shut down while they owned the lock.
     * Therefore, only wait for a grace period (1 s.).
     * Note that non-daemon threads have already exited here, so this
     * shouldn't affect carefully written threaded I/O code.
     */
    st = PyThread_acquire_lock_timed(self->lock, (PY_TIMEOUT_T)1e6, 0);
}
Py_END_ALLOW_THREADS
if (relax_locking && st != PY_LOCK_ACQUIRED) {
    PyObject *msgobj = PyUnicode_FromFormat(
        "could not acquire lock for %A at interpreter "
        "shutdown, possibly due to daemon threads",
        (PyObject *) self);
    const char *msg = PyUnicode_AsUTF8(msgobj);
    Py_FatalError(msg);
}
```

20.3 Notes

To workaround bpo-19565 on Windows, multiprocessing crash at exit, `_winapi.Overlapped` deallocator leaves the overlapped handle open if Python is exiting, see the commit:

```
commit 633db6f6a69fd44b4a27e7e216ff7a138f69aaf3
Author: Richard Oudkerk <shibturn@gmail.com>
Date:   Sun Nov 17 13:15:51 2013 +0000

    Issue #19565: Prevent warnings at shutdown about pending overlapped ops.
```

20.4 Python issues

- 2013-10-31: Clear state of threads earlier in Python shutdown. Call `_PyThreadState_DeleteExcept(tstate)` in `Py_Finalize()`. This issue introduced corrupted a Python frame of an asyncio daemon thread which leaded to a crash: bpo-20526. I had to revert the `_PyThreadState_DeleteExcept(tstate)` change.

20.5 Cython

__dealloc__():

> By the time your `__dealloc__()` method is called, the object may already have been partially destroyed and may not be in a valid state as far as Python is concerned, so you should avoid invoking any Python operations which might touch the object. In particular, don't call any other methods of the object or do anything which might cause the object to be resurrected. It's best if you stick to just deallocating C data.

20.6 Daemon threads

- Subinterpreters cannot spawn daemon threads anymore, since Python 3.9: https://bugs.python.org/issue37266
- In Python 3.8, daemon threads now exit immediately when they attempt to acquire the GIL, after Py_Finalize() has been called:
  - https://bugs.python.org/issue36475 with https://github.com/python/cpython/commit/f781d202a2382731b43bade845a58d28a02e9ea1
  - https://bugs.python.org/issue39877 with https://github.com/python/cpython/commit/eb4e2ae2b8486e8ee4249218b95d94a9f0cc513e
- change of bpo-19466 caused bpo-20526 regression

CHAPTER 21

Python builds

See also *C compilers*.

21.1 Release build

Python is built with `-O3 -DNDEBUG` compiler flags.

21.1.1 LTO

Use `gcc -flto`

Maybe also: `-fuse-linker-plugin -ffat-lto-objects -flto-partition=none`.

21.1.2 PGO

- Build Python with `gcc -fprofile-generate`
- Run the test suite: `make run_profile_task` uses `PROFILE_TASK` variable of Makefile. configure uses `./python -m test --pgo` by default. List of test used with `--pgo`: see libregrtest/pgo.py.
- Rebuild Python with `gcc -fprofile-use`

21.1.3 PGO + LTO

Use PGO and add LTO flags.

21.2 Debug build

- `./configure --with-pydebug`
- Python built using `-Og` if available, or `-O0` otherwise.
- Define the `Py_DEBUG` macro which enables many runtime checks to ease debug.

21.3 Special builds

Read Misc/SpecialBuilds.txt.

CHAPTER 22

Python Subinterpreters

See also *Python Finalization*.

- https://github.com/ericsnowcurrently/multi-core-python/
- bpo-36476: Runtime finalization assumes all other threads have exited
- bpo-1635741: Py_Finalize() doesn't clear all Python objects at exit created in 2007
- LWN: Subinterpreters for Python (May 13, 2020) By Jake Edge

22.1 Milestones

- May 2020: PoC: Subinterpreters 4x faster than sequential execution or threads on CPU-bound workaround

22.2 TODO list for per-interpreter GIL

Search for Subinterpreters issues at bugs.python.org.

Meta issue: per-interpreter GIL. Issues:

- parser_init(): _PyArg_Parser
- Disble lzma, bz2
  - lzma: https://github.com/python/cpython/pull/19382
  - bz2: https://github.com/python/cpython/commit/5d38517aa1836542a5417b724c093bcb245f0f47 (this fix is not enough)
- Workaround: __bases__
- Free lists:
  - float: DONE

  - slice: DONE
  - frame: DONE
  - list: DONE
  - async gen: DONE
  - context: DONE
  - dict: DONE
- _PyUnicode_FromId(): https://bugs.python.org/issue39465
- Unicode interned strings: https://github.com/python/cpython/pull/20085
- Singletons
  - None, True, False, Ellipsis: https://bugs.python.org/issue39511
  - bytes: empty string and single character singletons: DONE
  - str: empty string and single latin1 character singletons: DONE
  - empty frozenset singleton: DONE (removed)
- Type method cache
- Heap types
  - https://bugs.python.org/issue40077
  - https://bugs.python.org/issue40601
- pymalloc
- Workarounds
  - bpo-40533: Make PyObject.ob_refcnt atomic in subinterpreters: https://github.com/python/cpython/pull/19958
  - _dictkeysobject.dk_refcnt made _Atomic
  - Disable GC: https://github.com/python/cpython/commit/d8135e913ab7c694db247c86d0a84c450c32d86e
- tstate: get/set TSS: https://bugs.python.org/issue40522
- Per-interpreter GIL: https://bugs.python.org/issue40512

Enhancements:

- Debug: ensure that an object is not accessed by two interpreters: https://bugs.python.org/issue33607
- _xxsubinterpreters.run_string(): release the GIL: https://github.com/python/cpython/commit/fb2c7c4afbab0514352ab0246b0c0cc85d1bba53
- subprocess: close_fds=False, posix_spawn() is safe in subinterpreters

22.3 Limitations

Not supported in subinterpreter:

- os.fork(): it may be possible to fix it.
- signal.signal()
- static types

Current workarounds:

- Disable GC
- Disable many caches like frame free list
- etc.

22.4 Convert static type to heap type

See: Convert static types to heap types: use PyType_FromSpec().

Example: Modules/_abcmodule.c.

Decrement the type reference counter in the dealloc function. Something like:

```
static void
my_dealloc(my_data *self)
{
    (...)
    PyTypeObject *tp = Py_TYPE(self);
    tp->tp_free(self);
    Py_DECREF(tp);
}
```

22.5 Add a module state to a module

Example: Modules/_abcmodule.c.

Add traverse, clear and free functions to the module to better collaborate with the garbage collector. Otherwise, the GC fails to break reference cycles.

22.6 Heap allocated types

Modules/_randommodule.c:

```
PyObject *Random_Type = PyType_FromSpec(&Random_Type_spec);
```

Example:

```
$ ./python
Python 3.9.0a6+ (heads/frame_getback:6bde4d96c7, Apr 29 2020, 03:02:24)
>>> import _random as mod1
>>> import sys; del sys.modules['_random']
>>> import _random as mod2
>>> mod2.Random is mod1.Random
False
>>> mod1.Random.x=1
>>> mod2.Random.x
AttributeError: type object '_random.Random' has no attribute 'x'
```

22.7 Multiphase initialization (PEP 489)

See _abc module.

- PyInit__abc() calls PyModuleDef_Init
- PyModuleDef has slots, at least Py_mod_exec.

22.8 Get module

Create module:

```
_PyModule_CreateInitialized(struct PyModuleDef* module, int module_api_version)
```

Members:

- PyModuleDef.m_base.m_index: int
- PyInterpreterState.modules_by_index: list

PyModuleDef_Init() assigns an unique index to a PyModuleDef. It is called by _PyModule_CreateInitialized().

_PyImport_FixupExtensionObject() and import_find_extension() call:

```
_PyState_AddModule(PyThreadState *tstate, PyObject* module, struct PyModuleDef* def)
```

Modules with slots must not be added to PyInterpreterState.modules_by_index.

22.9 Module State

Find a module:

```
m = PyState_FindModule(&posixmodule);
```

From a module:

```
void *state = PyModule_GetState(module);
```

CHAPTER 23

Add a field to PyConfig

23.1 Documentation

- https://docs.python.org/dev/c-api/init_config.html
- https://www.python.org/dev/peps/pep-0587/

23.2 Main files

- `Python/preconfig.c`: _PyPreConfig_Read()
- `Python/initconfig.c`: PyConfig_Read()
- `Include/cpython/initconfig.h`: PyConfig structure
- `Include/internal/pycore_initconfig.h`

Python initialization and finalization:

- `Modules/main.c`:
  - pymain_main()
  - pymain_free()
  - Py_BytesMain()
  - Py_RunMain()
- `Python/pylifecycle.c`:
  - Py_InitializeFromConfig()
  - Py_PreInitialize()
  - Py_FinalizeEx()

23.3 Add a field to PyConfig

- `Include/cpython/initconfig.h`: Add a new field into `PyConfig` structure
- `Python/initconfig.c`:
  - Initialize the field in `_PyConfig_InitCompatConfig()` if the default is not zero
  - `_PyConfig_Copy()`: add `COPY_ATTR(field)`
  - `config_as_dict()`: add `SET_ITEM_INT(field)`
  - `PyConfig_Read()`: if needed, add an assertion at the function exit to validate that the field is valid.
- `Lib/test/test_embed.py`: Add field default value to `InitConfigTests.CONFIG_COMPAT`
- `Doc/c-api/init_config.rst`: Document the new field.

23.4 Add a field to PyPreConfig

- `Include/cpython/initconfig.h`: Add a new field into `PyPreConfig` structure
- `Python/preconfig.c`:
  - Initialize the field in `_PyPreConfig_InitCompatConfig()` if the default is not zero
  - `preconfig_copy()`: add `COPY_ATTR(field)`
  - `_PyPreConfig_AsDict()`: add `SET_ITEM_INT(field)`
  - `preconfig_read()`: if needed, add an assertion at the function exit to validate that the field is valid.
- `Lib/test/test_embed.py`: Add field default value to `InitConfigTests.PRE_CONFIG_COMPAT`
- `Doc/c-api/init_config.rst`: Document the new field.

23.5 Add an unit test for new PyPreConfig or PyConfig field

- `Programs/_testembed.c`, `test_init_from_config()`: Set the field to a value different than the default.
- `Lib/test/test_embed.py`: Update `test_init_from_config()` of `InitConfigTests`.

23.6 Add a new command line option

- `Python/getopt.c`: Update `SHORT_OPTS` or `_PyOS_LongOption`
- `Python/initconfig.c`: Modify config_parse_cmdline()
- `Lib/subprocess.py`: Update `_args_from_interpreter_flags()` and `_optim_args_from_interpreter_flags()` if needed.
- Document the new env option
  - `Python/initconfig.c`: Update one of the `usage_xxx` variables
  - `Doc/using/cmdline.rst`
  - `Misc/python.man`

Example of `Python/initconfig.c` usage:

```
-B     : don't write .pyc files on import; also PYTHONDONTWRITEBYTECODE=x\n\
```

Example of `Doc/using/cmdline.rst` entry:

```
.. cmdoption:: -d

   Turn on parser debugging output (for expert only, depending on compilation
   options).  See also :envvar:`PYTHONDEBUG`.
```

- Add an unit test. `Lib/test/test_cmd_line.py`, `Lib/test/test_embed.py` (`InitConfigTests`) or another test depending on how the value is exposed in Python.

23.7 Add an environment variable

- `Python/initconfig.c`:
  - Modify the appropriate helper function of `PyConfig_Read()` to read the environment variable.
  - Update `usage_xxx` to document the new env var

Example:

```
_Py_get_env_flag(use_env, &config->parser_debug, "PYTHONDEBUG");
```

- `Doc/using/cmdline.rst`: Document the new env var
- `Misc/python.man`: Document the new env var
- `Programs/_testembed.c`: Update `test_init_from_config()` and `set_most_env_vars()` to set the env var, `PyConfig` must have the priority.
- `Lib/test/test_embed.py`: Update `InitConfigTests`.

Note: environment variable with a name starting with `PYTHON` are ignored when using `-E` or `-I` command line options.

CHAPTER 24

Python C API

- https://docs.python.org/dev/c-api/
- https://pythoncapi.readthedocs.io/
- https://github.com/pyhandle/hpy

24.1 Add a new function to the limited C API

Don't forget to add it to `PC/python3.def`.

CHAPTER 25

Python Thread State

25.1 Get the current Python Thread State (tstate)

- `_PyRuntimeState_GetThreadState(runtime)`: read `runtime->gilstate.tstate_current`
- `_PyThreadState_GET()`: internal C API, call `_PyRuntimeState_GetThreadState(&_PyRuntime)`, new in Python 3.8.
- `PyThreadState_Get()`: opaque function call
- `_PyThreadState_UncheckedGet()`: added to Python 3.5.2 (bpo-26154)
- `PyGILState_GetThisThreadState()`
- `PyThreadState_GET()`: macro, alias to PyThreadState_Get(). When pycore_pystate.h is included: macro redefined as an alias to _PyThreadState_GET().

There was also `_PyThreadState_Current`: removed from Python 3.5.1.

History:

- Python 3.7: `PyThreadState_GET()` reads `_PyThreadState_Current` (atomic variable).
- Python 3.8: `_PyThreadState_Current` becomes `_PyRuntime.gilstate.tstate_current`

25.2 Get the current interpreter (interp)

- `PyInterpreterState_Get()`: limited C API. new in Python 3.9. Known as `_PyInterpreterState_Get()` in Python 3.8. Implemented as `_PyThreadState_GET()->interp`.
- `_PyInterpreterState_GET()`: internal C API, added to Python 3.8. Previously called `_PyInterpreterState_GET_UNSAFE()` in Python 3.8.

- `_PyGILState_GetInterpreterStateUnsafe()`: read `_PyRuntime.gilstate.autoInterpreterState`, new in Python 3.6.

Python 3.9 also defines `_PyInterpreterState_Get()` as an alias to `PyInterpreterState_Get()` for backward compatibility.

25.3 Main thread and main interpreter

- `PyInterpreterState_Main()`: get `_PyRuntime.interpreters.main`
- `_PyOS_IsMainThread()`: call `_Py_ThreadCanHandleSignals(_PyInterpreterState_GET())`
- `_Py_IsMainThread()`: Check if the current thread is the main thread.
- `_Py_IsMainInterpreter(tstate)`
- `_Py_ThreadCanHandleSignals(interp)`: added by https://bugs.python.org/issue40010
- `_Py_ThreadCanHandlePendingCalls()`: see https://bugs.python.org/issue40231

Identifier of the main thread (of the main interpreter): `_PyRuntime.main_thread`. Identifier read by `PyThread_get_thread_ident()`, it's not a Python thread state. Used by `_Py_IsMainThread()`.

Main interpreter: `_PyRuntime.interpreters.main`.

25.4 GIL

`take_gil()` and `drop_gil()` stores the Python thread state into `_PyRuntime.ceval.gil.last_holder`.

25.5 Signal handler

25.5.1 trip_signal() of signalmodule.c

Python 3.9 now always uses the main interpreter (`_PyRuntime.interpreters.main`), it no longer tries to get the current Python thread state: bpo-40082.

`_PyEval_SignalReceived(interp)` sets `signals_pending` and `eval_breaker` to 1.

Call `_PyEval_AddPendingCall(interp, ...)` if writing into the wakeup fd fails.

Python 3.9 made `eval_breaker` and pending calls per interpreter.

On Windows, `SIGINT` (CTRL+C) signal handler is called from a different thread at each call. MSDN signal documentation:

> When a CTRL+C interrupt occurs, Win32 operating systems generate a new thread to specifically handle that interrupt.
>
> This can cause a single-thread application, such as one in UNIX, to become multithreaded and cause unexpected behavior.

25.5.2 faulthandler

`faulthandler_fatal_error()` and `faulthandler_user()` signal handlers use `PyGILState_GetThisThreadState()` to get the current Python thread state:

```
/* SIGSEGV, SIGFPE, SIGABRT, SIGBUS and SIGILL are synchronous signals and
   are thus delivered to the thread that caused the fault. Get the Python
   thread state of the current thread.

   PyThreadState_Get() doesn't give the state of the thread that caused the
   fault if the thread released the GIL, and so this function cannot be
   used. Read the thread specific storage (TSS) instead: call
   PyGILState_GetThisThreadState(). */
tstate = PyGILState_GetThisThreadState();
```

`enable()` and `register()` use `_PyThreadState_UncheckedGet()` and then store `PyThreadState_GetInterpreter(tstate)` to use it from signal handlers.

25.6 tracemalloc

Hooks on memory allocators use `PyGILState_GetThisThreadState()` to get the current Python thread state. `PyMem_RawMalloc()` can be called without holding the GIL.

`PyMem_RawMalloc()` hook uses `PyGILState_Ensure()` and `PyGILState_Release()` to hold the GIL.

25.7 PyGILState and subinterpreters

- https://bugs.python.org/issue1021318
- https://bugs.python.org/issue10915
- https://bugs.python.org/issue15751

25.8 subinterpreters

It is currently possible for a single native thread to be associated to multiple Python thread states: one per interpreter.

Extract of `_testcapi.run_in_subinterp()` implementation:

```
PyThreadState *mainstate = PyThreadState_Get();

PyThreadState_Swap(NULL);

PyThreadState *substate = Py_NewInterpreter();
...
Py_EndInterpreter(substate);

PyThreadState_Swap(mainstate);
```

`Py_NewInterpreter()` creates a new Python thread state. Extract of its implementation:

```
PyInterpreterState *interp = PyInterpreterState_New();
...
PyThreadState *tstate = PyThreadState_New(interp);
..
PyThreadState_Swap(tstate);
```

25.9 Pass tstate explicitly

Pass the Python thread state explicitly (January 2020) by Victor Stinner.

25.10 Move global variables into PyInterpreterState

`PyLong_FromLong()` now requires to get the current interpreter to access `PyInterpreterState.small_ints` singletons.

25.11 Thread-local storage

Mark Shannon: experiment to moving the Python thread state to thread-local storage (TLS):

- https://mail.python.org/archives/list/python-dev@python.org/thread/RPSTDB6AEMIACJFZKCKIRFTVLAJQLAS2/
- https://github.com/python/cpython/compare/master...markshannon:threadstate_in_tls

CHAPTER 26

Fork in Python

The os.fork() function causes many implementation issues. It is supposed on most platforms, but Windows and VxWorks.

26.1 posix_spawn() (fork+exec)

First of all, it's important to know how to avoid fork, especially to spawn child processes which execute a new program :-)

On Windows, spawning a process can be done using `CreateProcess()` which doesn't use fork. On Unix, `posix_spawn()` can avoid fork on some platforms, and handles the dirty work for us on other platforms which implement it in userland.

Python 3.8 provides the os.posix_spawn() function. Not only `posix_spawn()` is safer than calling manully `fork()` + `exec()`, it can also be way faster in some cases.

The `subprocess` module can use `os.posix_spawn()` under some conditions:

- *close_fds* is false;
- *preexec_fn*, *pass_fds*, *cwd* and *start_new_session* parameters are not set;
- the *executable* path contains a directory.

26.2 fork() (fork without exec)

The POSIX standard says only calling exec() is after after calling fork(). Calling any function different than exec() after fork() is unsafe.

Forking a process creates a child process which only has 1 thread: all other threads are destroyed. The whole memory is duplicated. For performance, usually memory is copied using "copy-on-write": physical memory pages are marked as read-only and shared between the parent and the child process. When one process modify a memory page, the page is really duplicated and is no longer read-only.

See os.fork() function documentation.

26.3 Reinitialize all locks after fork

When fork is called, all threads are destroyed immediately except of the thread which called fork(). Locks can be in an inconsistent state. Using a lock after a fork can lead to a hang or to a crash.

The meta-issue bpo-6721 "Locks in the standard library should be sanitized on fork" tracks this problem.

The bpo-40089 added `_PyThread_at_fork_reinit()` function to reinitialize a lock to the unlocked state. It can be called after a fork to prevent the bug.

26.4 PyOS_AfterFork_Child()

Python has multiple functions called before and after fork:

- `PyOS_BeforeFork()`
- `PyOS_AfterFork_Parent()`
- `PyOS_AfterFork_Child()`

The importlib lock is acquired before fork and released after fork.

The `PyOS_AfterFork_Child()` function is the most important: update Python internal states after fork in the child process.

- Reset the GIL state
- Reset threads
- Reset importlib lock
- Clear pending signals to prevent to handle the same signal "twice" (once in the parent, once in the child)
- Delete Python thread states of other threads which have been destroyed.
- etc.

26.5 os.register_at_fork()

The os.register_at_fork() added to Python 3.7 allows to register functions which will be called at fork:

- before fork (in the parent process)
- after fork in the parent process
- after fork in the child process

26.6 macOS 10.14 (Mojave)

The multiprocessing started to crash randomly on macOS 10.14 (Mojave). The multiprocessing default start method changed from fork to spawn in Python 3.8: see bpo-33725.

26.7 getaddrinfo() and gethostbyname() locks

The C library provides getaddrinfo() and gethostbyname() functions which are not thread-safe on some platforms. Python uses an internal lock on platforms where these functions are known to not be thread-safe.

There are numerous articles about bugs caused by threads or caused by the lock added to make the function thread-safe. For example, Python didn't reinitialize the getaddrinfo() lock at fork in the child process (the lock has been removed).

A thread-safe gethostbyname_r() function was added to avoid this issue.

The bpo-25920 removed the getaddrinfo() lock.

26.8 ssl.RAND_bytes()

"Applications must change the PRNG state of the parent process if they use any SSL feature with os.fork()." https://docs.python.org/dev/library/ssl.html#multi-processing

CHAPTER 27

Test the next Python

To evolve and remain relevant, incompatible changes must happen in Python. The problem is how to migrate existing code to the "next" Python with these incompatible changes.

27.1 Deprecation warnings

DeprecationWarning and PendingDeprecationWarning warnings are hidden by default, to not bother users. Developers can use `-Wd` (`-Wdefault`) command line option on Python which displays these warnings.

Python Development Mode.

What's New in Python 3.9: You should check for DeprecationWarning in your code section.

27.2 Fedora COPR

Test Rawhide (future Fedora 33) with Python 3.9 as the "system Python": https://copr.fedorainfracloud.org/coprs/g/python/python3.9/

27.3 pythonci project

https://github.com/vstinner/pythonci

27.4 Projects having "Python nightly in their CI

XXX

numpy?

Cython?

27.5 Searching deprecated API usage

27.5.1 Sourcegraph

GitHub code search is not powerful enough and there is a lot of noise. (e.g. many people copy CPython source code). On the other hand, Sourcegraph only searches from major repositories, and has powerful filtering. Example: search PyEval_ReleaseLock.

27.5.2 Top 4000 packages

You can download a list of top 4000 PyPI packages in JSON format from hugovk site.

INADA-san's script to download sdist packages from the JSON file: download_sdist.py.

Note that this script doesn't download packages without sdist (e.g. only universal wheel). It is because INADA-san has searched Python/C API. The pain of the removal can be reduced by fixing most of top 4000 packages.

27.6 Docker images

Barry Warsaw maintains CI Images for Python. Example: .gitlab-ci.yml of flufl.lock.

Performance:

CHAPTER 28

Python performance

28.1 Codespeed websites

- speed.python.org: CPython benchmarks, run performance and (tobami's) codespeed
- speed.pypy.org: PyPy benchmarks, run PyPy benchmarks
- speed.pyston.org: Pyston benchmarks, run pyston-perf

28.2 Projects

Python benchmarks:

- performance (GitHub project): Python benchmark suite
- perf (GitHub project): Python module to run, analyze and compare benchmarks
- performance_results: Results of the of performance benchmark suite
- pymicrobench: Collection of Python microbenchmarks written for CPython.
- PyPy benchmarks: PyPy benchmark suite
- Codespeed (tobami's flavor): Django application to explose benchmark results. Pages: Homepage, Changes, Timeline, Compare.
- pyston-perf: Pyston benchmark suite

Other Python Benchmarks:

- Numba benchmarks
- Cython Demos/benchmarks
- pythran numpy-benchmarks

Old deprecated projects:

- benchmarks: the old and deprecated Python benchmark suite

28.3 Mailing list

- Python speed mailing list

28.4 Old Python benchmarks

The following benchmarks were removed from pyperformance:

- bm_rietveld.py: use Google AppEngine and Rietveld application which are not available on PyPI
- bm_spitfire.py: Spitfire project is not available on PyPI
- bm_threading.py (`threading_iterative_count`, `threading_threaded_count`)
- gcbench.py
- pystone.py
- tuple_gc_hell.py

28.5 Zero copy

Python3:

```
offset = 0
view = memoryview(large_data)
while True:
    chunk = view[offset:offset + 4096]
    offset += file.write(chunk)
```

This copy creates views on `large_data` without copying bytes, no bytes is copied in memory.

CHAPTER 29

CPython implementation optimizations

29.1 Singletons

- Integers in range [-5; 256]
- Empty string
- Single Latin-1 letter

29.2 IO

- Efficient buffering: FileIO.read() and FileIO.readall()
- HTTP and socket buffering

29.3 Free lists

When a Python object is destroyed, its type can decided to keep the memory alive to optimize the allocation of future objects. Bultin types using a free list: see `clear_freelists()` in `Modules/gcmodule.c`.

Python 3.8 types using a free list:

- Common types:
  - dict: PyDictObject
  - float: PyFloatObject
  - list: PyListObject
  - set: PySetObject
  - str: PyUnicodeObject

    - tuple: PyTupleObject
- Other types:
    - async generator value: _PyAsyncGenWrappedValue
    - builtin function: PyCFunctionObject
    - contextvars.Context: PyContext
    - frame: PyFrameObject
    - method: PyMethodObject

Python 2.7 types using a free list:

- builtin function: PyCFunctionObject
- float: PyFloatObject
- frame: PyFrameObject
- method: PyMethodObject
- int (but not long): PyIntObject
- tuple: PyTupleObject
- unicode: PyUnicodeObject

29.4 Others

- peephole
    - x in {1, 2, 3} => frozenset (constant)
- Overalllocation
    - list
    - bytearray?
    - PyUnicodeWriter
- Free list
- method cache
- stringlib
    - process long per long, instead of byte per byte
    - use goto

29.5 ceval.c

- computed goto: label + goto
- fast locals: f_localsplus of a frame

29.6 Unicode

- PEP 393: efficient storage for ASCII
- ASCII strings only require to copy N bytes to copy N characters: 2 or 4 times faster than applications using UTF-16 or UCS-4
- findchar: use memchr(), even for UCS-2 and UCS-4

29.7 dict

- specialized for int key
- specialized for str key
- hash(str)

29.8 list

- timsort

29.9 implemented in C

- decimal of Python 3.3 is 120x faster than the Python implementation of Python 3.2

CHAPTER 30

Python Startup Time

CPython startup is "slow": takes between 8 ms and 100 ms depending on the Python version, the operating system, how many .pth files are installed, if Python runs into a virtual environment, etc.

See: https://github.com/bdrung/startup-time

30.1 Raw benchmark

Benchmark:

```
python3 -m perf command -o python2.json -- python2 -c pass
python3 -m perf command -o python3.json -- python3 -c pass
python3 -m perf command -o python2_nosite.json -- python2 -S -c pass
python3 -m perf command -o python3_nosite.json -- python3 -S -c pass
```

Result speed.python.org (July 2017), fatest to slowest:

- `git --version`: 974 us +- 7 us
- `perl -e ' '`: 1.18 ms +- 0.01 ms
- Python 2.7: 6.4 ms with site; 3.0 ms without site (-S)
- `php -r ' '`: 8.57 ms +- 0.05 ms
- Python 3.7: 14.5 ms with site; 8.4 ms without site (-S)
- `ruby -e ' '`: 32.8 ms +- 0.1 ms
- `hg version`: 44.6 ms +- 0.2 ms

Reference: benchmark run at July 2017.

30.2 importtime

To analyze the startup time of your application, Python 3.7 and newer have a builtin profiler on import time: `-X importtime` option.

Truncated example:

```
$ python3.7 -X importtime -c pass
import time: self [us] | cumulative | imported package
import time:       274 |        274 | zipimport
(...)
import time:       949 |       2055 | io
import time:       453 |        453 |       _stat
import time:       701 |       1154 |     stat
import time:       595 |        595 |       genericpath
import time:       886 |       1480 |     posixpath
import time:      3005 |       3005 |     _collections_abc
import time:      1922 |       7559 |   os
(...)
import time:      4325 |      22832 | site
```

An alternative is to set `PYTHONPROFILEIMPORTTIME=1` environment variable.

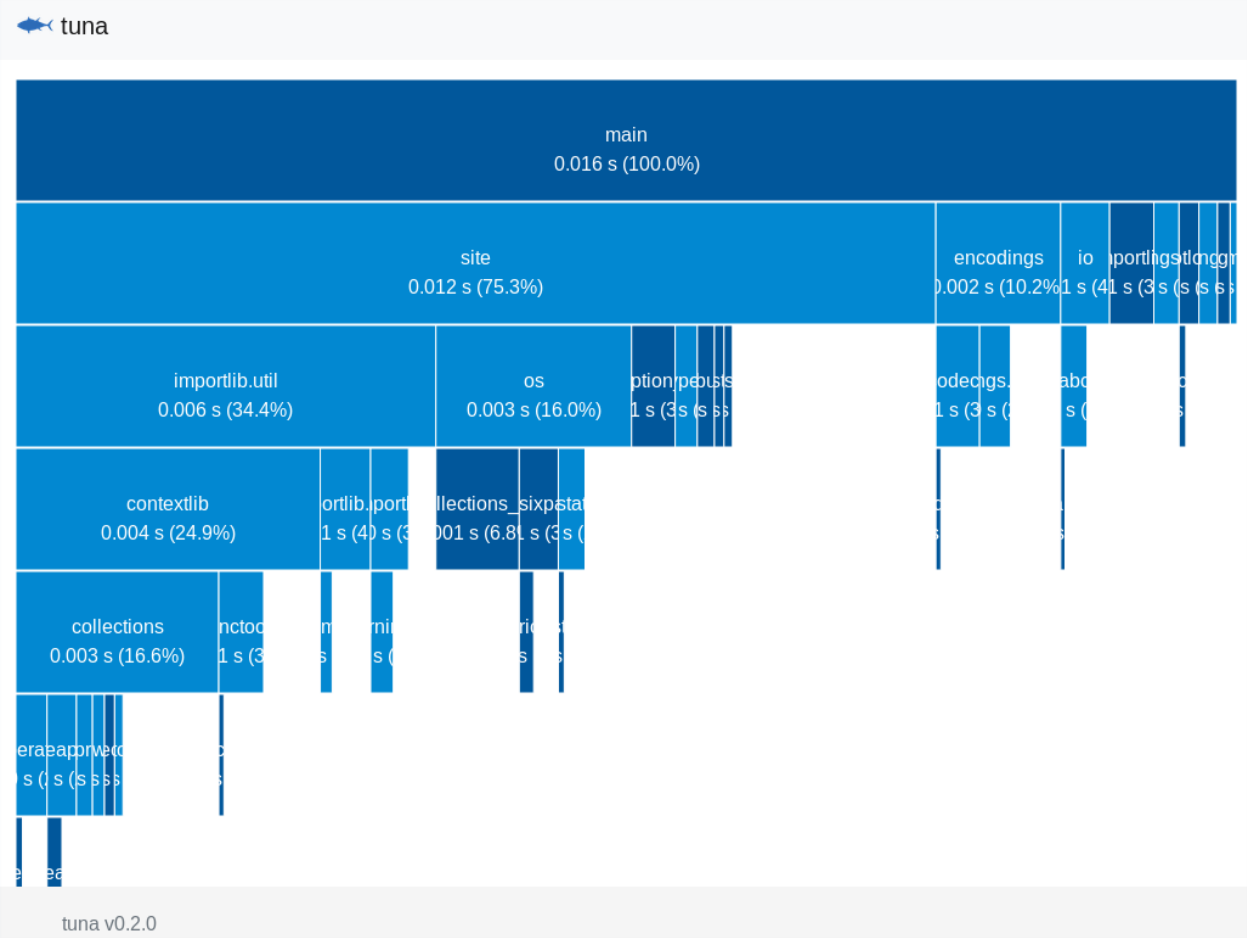
No surprise, `site` has the longest cumulative time. Disable `site` to speedup Python :-) Example:

```
$ python3.7 -X importtime -S -c pass
import time: self [us] | cumulative | imported package
import time:        88 |         88 | zipimport
import time:       580 |        580 | _frozen_importlib_external
import time:        66 |         66 |     _codecs
import time:       498 |        564 |   codecs
import time:       411 |        411 |   encodings.aliases
import time:       678 |       1653 | encodings
import time:       232 |        232 | encodings.utf_8
import time:       155 |        155 | _signal
import time:       341 |        341 | encodings.latin_1
import time:        56 |         56 |     _abc
import time:       273 |        329 |   abc
import time:       298 |        626 | io
```

tune is a Python profile viewer which accepts importtime logs as input, see Nico Schlömer's comment on bpo-31415. Extract:

```
python -X importprofile application.py 2>import.log
tuna import.log
```

Example with `python3.7 -X importtime -c pass`:



Links:

- How to speed up Python application startup time (Jan, 2018) by INADA Naoki.
- importtime-waterfall (GitHub project): Generate waterfalls from `-X importtime` tracing.

30.3 .pth files

See Deprecate and remove pth files created by Barry A. Warsaw at 2018-06-22.

30.4 Links

- python-dev: Python startup time (July 2017) by Victor Stinner
- LWN: Reducing Python's startup time (August 16, 2017) by Jake Edge.

CHAPTER 31

PEP 393 performances

Writing efficient code manipulating Unicode is harder since the PEP 393. The problem is to respect the canonical form.

If you preallocate an ASCII buffer but you need to write a Latin1 character, you have to convert the ASCII string to Latin1 which means copying all already written characters. It is inefficient especially if the Latin1 characters occurs at the end. If you preallocate an UCS4 buffer, but the result is UCS2, you have to "shrink" the buffer from UCS4 to UCS2, which means copying all characters.

To not having to widen or shrink your buffer, you can scan your input to compute the maximum character before allocating the buffer. In practice, processing the input twice may be slower.

Another problem is the length of the result. Getting the length of str%args and str.format(args) require to do the work twice: once to get the length, once to write characters. Both approaches were tested (*), and processing the output twice is too slow.

For efficient code, you should be optimistic and enlarge or widen the buffer on demand. When the output length is unknown, it is better to overallocate the buffer.

The _PyUnicodeWriter API helps to implement such function:

- Widen the buffer on demand
- Enlarge the buffer on demand
- Minimum length and overallocation of the buffer can be configured
- Avoid completly the need of a buffer when the output is only composed of one string
- Delay allocation of the buffer until the first write. It helps to compute the length and kind of the buffer, because the length and kind cannot always be computed before the first write. It avoids also allocating a buffer is no write is done at all (ex: error before writing the first characters).
- Give a direct access to the buffer for best performances

Community:

CHAPTER 32

Python community

Come for the Language, Stay for the Community – Brett Cannon (and Naomi Ceder)

Pull requests can be like someone trying to give you a puppy you didn't ask for; they mean well, but they can forget a puppy is a decade-or-more commitment and you just don't like the puppy. – Brett Cannon

Maintaining an open-source project is like being a Flight Attendant for an airline where all tickets are free and the majority of customer surveys offer suggestions on how to fly the airplane. – Kelsey Hightower

Having a great language is… great, but having a community around it gives you a sense of belonging, which is one of the most basic instincts and desires we have as animals. – Sawyer X (Perl 5 maintainer), Pragmatic Perl Interviews, May 2013

"When you choose a language, you're choosing more than a set of technical trade-offs-you're choosing a community." – Joshua Bloch. ("Coders at Work: Reflections on the Craft of Programming". Book by Peter Seibel, 2009.)

32.1 See also

- *Diversity*
- *Mentoring*
- *Communication Channels*
- *CPython Core Developers*

32.2 Talks

- Setting expectations for open source participation by Brett Cannon (September 2018)
- The give and take of open source by Brett Cannon, JupyterCon, August 2017
- Dial M For Mentor PyCon 2017 by Mariatta Wijaya, Pycon US, May 2017

- Come for the Language, Stay for the Community by Naomi Ceder, EuroPython 2016

32.3 Documentation

- https://devguide.python.org/
  - http://github.com/python/devguide/
  - https://github.com/python/devguide/issues/120
- http://cpython-core-tutorial.readthedocs.io/

32.4 Work In Progress

- Getting along in the Python community by Guido van Rossum and Brett Cannon (May 2018):

  When someone is having an emotional response to a post, they should wait to respond. Guido: *"Sitting on your hands is often a good response"*.
- Reserve easy issues to non-core developers: [python-committers] Please stop fixing easy issues right now! Leave them as exercices to newcomes
- RFC: Process to become a core developer
- Email templates:
  - Bug triage promoted
  - Core developer promoted

32.5 Ideas

- Organize mentoring? Make it public? List documentations for mentors?
- Django CoC: procedure to report abuse
- Thanks.Python.org - clone of https://thanks.rust-lang.org/ but with snakes and even cuter emojis!
- Tooling to detect active contributors: number of commits, emails, reviews, etc. Is it doable?
- Statistics on GitHub reviews:
  - https://developer.github.com/v3/pulls/reviews/#list-reviews-on-a-pull-request
  - https://github.com/joacim-boive/github-statistics # Chrome extension
- gamification: public top 5?
  - The idea is to motivate the contributors: one way is "self-motivation" (i.e. streak counter, daily/weekly/monthly goals, etc). Another is "multiplayer" (e.g. leaderboards and other "competitive" aspects).
  - Give badges depending on the number of posted bugs, PR, emails, etc.?
  - https://twistedmatrix.com/highscores/
- Bot to thanks automatically new contributors with cute emojis? "Congrats for your first PR merged into CPython !" (do email notifications like emojis?)

- Create subteams:
  - IDLE
  - asyncio
  - Documentation
  - Windows
  - Workgroup Community // core-workflow
  - XXX: need a bot on GitHub to restrict permissions to files/directories?
- Missing in action: drop core dev for inactive developers?

32.6 Issues when dealing with people

- https://geekfeminism.wikia.org/wiki/Five_Geek_Social_Fallacies
- https://geekfeminism.wikia.org/wiki/Ostracism_is_evil
- https://geekfeminism.wikia.org/wiki/Missing_stair
- https://en.wikipedia.org/wiki/Missing_stair
- https://en.wikipedia.org/wiki/Gaslighting

32.7 Links

- https://public.etherpad-mozilla.org/p/help-cpython-newcomers
- http://teachingopensource.org/

CHAPTER 33

Diversity

See also *CPython community*.

> *"You can't solve people problems with software."* – Baldur Bjarnason (September 2015)
>
> *"Don't solve problems with software that should be solved with talking."* – Tanya Reilly (Feb. 2018)
>
> *"When women speak out about diversity and the community, they get penalized and branded as non-technical. Some women chose not to speak out because of this. We will not be silenced."* – Mariatta Wijaya (April 2018)
>
> *"Increasing the core's diversity is a very important goal to ensure the future health of Python."* – Guido van Rossum (May 2018)

33.1 First issue: become aware of diversity issues

Usually, people involved since a long time in a project and who are in the short priviledged group (ex: "core reviewers" or "core developers"), are not aware of diversity issue. If there is no issue, who should anyone care of diversity?

Unconscious Bias can be a first hint.

33.2 Discuss Diversity

33.2.1 Communities

- PyLadies
- Django Girls
- Geek Feminism Wiki

33.2.2 Mailing Lists

- OpenStreetMap Diversity-talk
- IETF diversity
- Fedora Diversity Team

Python had a diversity mailing list.

33.2.3 Code and emotions

- Kate Gregory's keynote at C++ on Sea: Keynote: Oh The Humanity!, photos on Twitter

33.3 Positive Action

Wikipedia: Positive Action.

- Mentor people of underrepresented groups
- Diversity Tickets
- Outreachy Internships. See Outreachy Eligibility Rules.
- Red Hat Women in Open Source.

33.4 Code of Conduct (CoC)

Codes of Conduct:

- Python Community Code of Conduct
  - Apply to python-ideas, python-dev and python-committers mailing lists
  - Apply to the bug tracker
- OpenStretMap Code of Conduct
- Django Code of Conduct

Articles:

- Code of Conduct Updates for PyCon [US] 2018
- The Code Of Conduct (December, 2012) by Jesse Noller:

  > RESOLVED, that the PSF will only sponsor conferences that have or agree to create and publish a Code of Conduct/Anti Harassment guide for their conference. A basic template to work from has been generated by the Ada Initiative at Conference anti-harassment/Policy.

33.5 Leaving LLVM because of the CoC

- Diversity and Discrimination in Open Source by Rafael Avila de Espindola
- [llvm-dev] I am leaving llvm (May 2, 2018) by Rafael Avila
- [llvm-dev] re: I am leaving llvm (May 4, 2018) by Renato Golin

- Lobsters: aphyr's comment
- http://nondot.org/sabre/2018-05-02-Rafael.html

33.6 NodeJS

Why I'm leaving the Node.js project (August, 2017) by Bryan Hughes.

33.7 Issues with speaker diversity in tech conference

- Mariatta Wijaya (April 20, 2018):

  > Trying not to do any community engagement this month, but this is such high priority, and I can't stay silent. I'm so utterly heartbroken and disappointed to find out that there is a PyCon someplace 2018, and that it has 22 all male speaker lineup.

- Twitter: snipeyhead's thread (April 22, 2018)
- Twitter: mariatta's thread (April 22, 2018)
- Postmorten on the lineup at PyCon Ukraine this year by Volodymyr Hotsyk (Apr 22, 2018)
- Geek Feminism Wiki: PyCon [US] 2013 forking and dongles incident.
- How could GitHub announce an all-male conference line up the same week it shares results from an OSS demographics survey with 3% women?

33.8 Python

Diversity Statement:

> The Python Software Foundation and the global Python community welcome and encourage participation by everyone. Our community is based on mutual respect, tolerance, and encouragement, and we are working to help each other live up to these principles. We want our community to be more diverse: whoever you are, and whatever your background, we welcome you.

- https://www.python.org/community/diversity/
- http://wiki.python.org/moin/DiversityInPython
- Mentoring and diversity for Python (June, 2018)

For example, don't say "hey *guys*!" or "fellas" but "hey **everyone**!", forks or "y'all": see heyguys.cc.

Comment template:

> In the future please use gender-neutral words such as "folks" and "y'all" instead of "guys". Thanks.

33.9 Being a Woman on the Internet

Stories unrelated to Python, or not directly related to the Internet. Its more to have an idea of the general idea of harassement on the Internet.

- What It's Like to Be A Woman On the Internet (January 2018) by Chloe Condon

- I Was The Victim Of A Deepfake Porn Plot Intended To Silence Me (November 2018) by Rana Ayyub

33.10 Links

- Wikipedia: Imposter syndrome
- Wikipedia: Underrepresented group
- Unconscious Bias:
  - Wikipedia: implicit bias or implicit stereotype
  - Wikipedia: Unconscious bias training
  - Companies are on the hook if their hiring algorithms are biased. "After an audit of the algorithm, the resume screening company found that the algorithm found **two factors to be most indicative of job performance**: their name was **Jared**, and whether they **played high school lacrosse**. Girouard's client did not use the tool." and "It's a really great representation of part of the problem with these systems, that **your results are only as good as your training data**, (. . .)"
- The Zero Theory (first edition) by Haïkel Guémar (May, 2015)
- Diversity and inclusion: Stop talking and do your homework by Emma Irwin (Sep 2017)
- https://github.com/opal/opal/issues/941

CHAPTER 34

Communication Channels

See also the Python community.

- Mailing lists:
  - python-ideas
  - python-dev
  - python-committers
  - see the list of all Python mailing lists
- IRC:
  - `#python` on FreeNode
  - `#python-dev` on FreeNode: only for development *of* Python, mostly to discuss bugs and pull requests
- discuss.python.org (Discourse) (exist since 2018)
- python.zulipchat.com (Zulip) (exist since 2017)
- Twitter, incomplete list of *core devs*:
  - Barry Warsaw
  - Brett Cannon
  - Guido van Rossum
  - Łukasz Langa
  - Mariatta Wijaya
  - Serhiy Storchaka
  - Yury Selivanov
  - ... the full list is very long, and I'm too lazy to complete it :-)

There are two physical meetings for *CPython core developers*:

- Language Summit during Pycon US (one day)

- Sprint in September (one week)

CHAPTER 35

CPython mentoring

See also *Community*.

35.1 Mentoring

- https://www.python.org/dev/core-mentorship/
- https://mail.python.org/mailman/listinfo/core-mentorship/
- https://treyhunner.com/mentoring/resources.html

35.2 Working Hours

- https://mail.python.org/pipermail/python-committers/2018-May/005396.html

35.3 Stop fixing easy issues

https://mail.python.org/pipermail/python-committers/2017-June/004564.html

CHAPTER 36

CPython Core Developers

36.1 What is a CPython Core Developer?

Video of Mariatta Wijaya's talk: "What is a Python Core Developer?" (Pycon US 2018): https://www.youtube.com/watch?v=hhj7eb6TrtI

36.2 How to increase the number of core developers?

Guido van Rossum, June 2018 about the low number of active core developers:

> The best course of action seems to be to take measures to acquire new committers (and contributors), not to try and reactivate old inactive committers.

- *Mentoring*
- Write less code, spend more time on reviews and mentoring

36.3 Lists of Core developers

- https://github.com/python/voters/
- https://github.com/orgs/python/teams/python-core/members
- https://discuss.python.org/groups/committers
- https://bugs.python.org/user?iscommitter=1&@action=search&@sort=username&@pagesize=300
- https://devguide.python.org/developers/
- (outdated) https://hg.python.org/committers.txt
- https://mail.python.org/mailman/listinfo/python-committers members

36.4 Statistics

Statistics on *new* CPython core developer per year using devguide as data:

- 2007: 15
- 2008: 19
- 2009: 11
- 2010: 20
- 2011: 12
- 2012: 9
- 2013: 4
- 2014: 10
- 2015: 2
- 2016: 5
- 2017: 4
- 2018: 6

Links:

- [python-committers] Statistics: growth of core dev number vs growth of the code size/complexity https://mail.python.org/pipermail/python-committers/2017-December/004983.html
- https://mail.python.org/pipermail/python-committers/2018-June/005517.html

October 2017:

- **6542** pull requests, **4966** merged
- **848** contributors (including core devs)
- **34** core developers active on GitHub (4% of all contributors)
- 24% of PR are written by core devs

36.5 Process to become a core developer

https://github.com/vstinner/misc/blob/master/cpython/pep-core_dev_process.rst

- [python-committers] Requirements to get the "bug triage" permission? https://mail.python.org/pipermail/python-committers/2017-December/004960.html
- [python-committers] What is a CPython core developer? https://mail.python.org/pipermail/python-committers/2017-September/004865.html

36.6 Misc

- PyTexas: Keynote: My Path to Becoming a Python Core Developer by Emily Morehouse-Valcarcel
- [python-committers] Should I merge a PR that I approved if it was written by a different core developer? https://mail.python.org/pipermail/python-committers/2017-September/004854.html

36.7 Steering Council

- https://github.com/python/steering-council
- PEP 13 – Python Language Governance
- PEP 8100 – January 2019 steering council election

36.8 Inactive core devs

Current discussion: devguide issue: Simplify developer log.

Links:

- 2019-02-21: Official list of core developers
- 2019-02-11: Remove Coordinator role of inactive coordinators on bugs.python.org
- 2018-11-02: devguide PR: Complete the core dev list
- 2018-09-18: Which list of core developers is authoritative?
- 2018-06-20: devguide issue: Simplify developer log
- 2018-06-15: [python-committers] Missing In Action
- 2018-06-02: [python-committers] number of active core devs [was: Comments on moving issues to GitHub]
- 2017-12-06: [python-committers] Statistics: growth of core dev number vs growth of the code size/complexity

CHAPTER 37

CPython tasks

CPython is a big project. Maintaining CPython require to frequently do different tasks. This page tries to list all "tasks" required to maintain Python. The goal is to make sure that each task has at least two maintainers, so one maintainer can easily take holiday or stop working on this task.

For the background, see Vicky's talk Passing the Baton: Succession planning for FOSS leadership.

37.1 Tasks

- Review and merge pull requests: done by around 34 core developers (Stephane Wirtel's stats at Pycon Italy 2018). The merge action is restricted to core developers. Maintainers: active core developers (June 2018: around 34 core devs).
- Bug triage: closing a bug needs the bug triage permission. Maintainers: active core developres.
- Check for buildbot failures: Read logs of each buildbot failure, check if the failure is known. If the failure is known, maybe mention the new failure in the existing bug. Otherwise, open a new bug. Then reply to the email with a link to the bug. Maintainers: Victor Stinner, Pablo Galindo Salgado.
- Run bugs.python.org: fix bugs, deploy new version. See the meta bug tracker for bugs of bugs.python.org itself (not for Python bugs). Roundup is going to be deployed in a Docker container on OpenShift. Maintainers: Ezio Melotti, Maciej Szulik.
- Run pythontest.net. Maintainers: ?
- Run GitHub bots. Maintainers: Brett Cannon and Mariatta Wijaya.
- Update *vendored external libraries* <*vendored-libs*>. Maintainers: ?
- Update unicodedata on new Unicode release. Latest update (Unicode 11.0): https://bugs.python.org/issue33778. Maintainer: Benajamin Peterson.

37.2 Administrators

Some actions require administrators who are the only ones allowed to do actions.

- Mailing lists: create a new mailing list. Maintainer: “postmaster” (who is the current postmaster?).
- Bug tracker: give “bug triage permission”. Roundup Coordinators:
  - Brett Cannon.
  - Ezio Melotti,
  - R. David Murray
  - Victor Stinner
- GitHub cpython: add new core developers. Administrators:
  - Brett Cannon
  - Release managers (ex: Ned Deily)

Misc:

CHAPTER 38

History of Python

Guido van Rossum started to write Python in December 1989.

38.1 History of Python releases

See also Status of Python branches.

- Python 3.8: October 2019
- Python 3.7: June 2018
- Python 3.6: December 2016
- Python 3.5: September 2015
- Python 3.4: March 2014
- Python 3.3: September 2012
- Python 3.2: February 2011
- Python 2.7: July 2010
- Python 3.1: June 2009
- Python 3.0: December 2008
- Python 2.6: October 2008
- Python 2.5: September 2006
- Python 2.4: March 2005
- Python 2.3: July 2003
- Python 2.2: December 2001
- Python 2.1: April 2001
- Python 2.0: October 2000

- Python 1.5: April 1999

38.2 History of the Python language (syntax)

- Python 3.8: `x := 1` assignment expression (PEP 572) and `/` in function for positional-only parameters (PEP 570)
- Python 3.7: `async` and `await` become keywords
- Python 3.6: f-strings (PEP 498 "Literal String Interpolation")
- Python 3.5:
  - Add `async` and `await` (not really keywords yet)
  - The `@` operator (PEP 465 "A dedicated infix operator for matrix multiplication")
  - PEP 448 "Additional Unpacking Generalization"
- *(Python 3.4: no change)*
- Python 3.3:
  - `yield from`: PEP 380 "Syntax for Delegating to a Subgenerator"
  - `u'unicode'` syntax is back: PEP 414 "Explicit Unicode literals"
- *(Python 3.2: no change)*
- Python 2.7:
  - all changes of Python 3.1
- Python 3.1:
  - dict/set comprehension
  - set literals
  - multiple context managers in a single with statement
- Python 3.0:
  - all changes of Python 2.6
  - new `nonlocal` keyword
  - `raise exc from exc2`: PEP 3134 "Exception Chaining and Embedded Tracebacks"
  - `print` and `exec` become a function
  - `True`, `False`, `None`, `as`, `with` are reserved words
  - Change from `except exc, var` to `except exc as var`: PEP 3110 "Catching Exceptions in Python 3000"
  - Removed syntax: `a <> b`, `` `a` ``, `123l`, `123L`, `u'unicode'`, `U'unicode'` and `def func(a, (b, c)): pass`
  - PEP 3132 "Extended Iterable Unpacking"
- Python 2.6:
  - with: PEP 343 "The "with" Statement"
  - `b'bytes'` syntax: PEP 3112 "Bytes literals in Python 3000"

38.3 Old Python Versions

- Python 0.9.1
- Python 1.5.2
- Python 0.9.1 .. 2.0 source tarballs
- David Beazley tests in 2017 (tweets)
- neopythonic by Guido van Rossum: Ramblings through technology, politics, culture and philosophy by the creator of the Python programming language.
- History of Python (Wikipedia article)
- The History of Python by Guido van Rossum: A series of articles on the history of the Python programming language and its community.

38.4 Development before GitHub

Buildbot was only running after changes were pushed upstream. It was common that a change broke the Windows support, and so core devs pushed an "attempt to fix Windows" commit, and then a second one, etc.

To propose a change, a contributor had to open an issue in the bug tracker (Roundup at bugs.python.org), and attach a patch file. A core developer had to

- Download the patch locally
- Apply the patch file
- Fix conflicts: when the day was older than 1 day, conflicts were very likely

Misc/NEWS was basically always in conflict, especially on merges.

Changes were first fixed in the oldest supported branch, and then forward-ported to newer branches. For example, fixed in 2.6, and ported to 2.7, 3.0 and 3.1.

When Subversion was used, a Subversion "property" (in practice, a text file tracked by Subversion) listed the revision number of all "merged" changes. For example, when a change made in the 2.6 branch was merged into the 2.7 branch, it was added to this list. It was likely that this property file was in conflict. Sadly, it was a text file made of a single line with thousands of revision numbers. Text editors are not convenient to edit such file. It was barely possible to fix a conflict in this property.

A new tool to review (comment) patches was linked to Roundup: Rietveld. It was possible to generate a patch from a fork the Mercurial repository, and then get a review page. Rietveld supported multiple revisions of the same change. Drawback: the tool was not well integrated with Roundup. For example, there was no way to unsubscribe from a review.

38.5 Python 3000

- https://mail.python.org/pipermail/python-3000/
- https://www.python.org/dev/peps/pep-3000/
- https://www.python.org/dev/peps/pep-3100/

38.6 CVS, Subversion, Mercurial

- CVS: Initially hosted on cvs.python.org, it migrated to Sourceforge
- Subversion: https://svn.python.org/projects/python/
- Mercurial: https://hg.python.org/cpython/
  - Map Subversion revision to Mercurial commit: Misc/svnmap.txt file in the Python code base.
- Lookup service. Examples:
  - Subversion revision 68121: https://hg.python.org/lookup/r68121 redirects to http://svn.python.org/view?view=revision&revision=68121 but sadly this service is down (tested in September 2020)

CHAPTER 39

Misc notes

- https://cpython-pulls.herokuapp.com/

39.1 datetime strptime

Documentation: https://docs.python.org/dev/library/datetime.html#strftime-and-strptime-behavior

Code:

```
date = datetime.datetime.strptime(date, '%Y-%m-%d %H:%M:%S %z')
```

year-month-day hour:minute:second timezone:

```
%Y-%m-%d %H:%M:%S %z
```

39.2 Python builtin types

Builtin scalar types:

- bool, int, float, complex, bytes, str

Builtin container types:

- Sequence: tuple, list
- Mapping: dict
- Set: frozenset, set

Singletons:

- None, Ellipsis (`...`), False (`bool(0)`), True (`bool(1)`)
- Small integers: [-5; 256]

- Empty bytes and Unicode strings: `b''` and `''`
- Latin-1 single letter, examples: `'\0'`, `'a'`, `'\xe9'`
- Empty tuple
- Empty frozenset: frozenset() (removed from Python 3.10)

39.3 Python developer mode

=> implementated in Python 3.7 as: "python3 -X dev" or PYTHONDEVMODE=1 !

https://mail.python.org/pipermail/python-ideas/2016-March/039314.html

Strict developer mode:

```
PYTHONMALLOC=debug python3.6 -Werror -bb -X faulthandler script.py
```

Developer mode:

```
PYTHONMALLOC=debug python3.6 -Wd -b -X faulthandler script.py
```

- Show `DeprecationWarning` and `ResourceWarning` warnings: `python -Wd`
- Show `BytesWarning` warning: `python -b`
- Enable `faulthandler` to get a Python traceback on segfault and fatal errors: `python -X faulthandler`
- Debug hooks on Python memory allocators: `PYTHONMALLOC=debug`
- Enable Python assertions (assert) and set `__debug__` to `True`: remove (or just ignore) -O or -OO command line arguments

See also `PYTHONASYNCIODEBUG=1` for asyncio.

39.4 pyupgrade

https://github.com/asottile/pyupgrade

39.5 AST changes

- Removing ast.Param in Python 3.9 broke chameleon

39.6 Things to do in Python 3.10

- Remove again collections.Mapping
- Remove again "U" mode of open()
- Make "from __future__ import annotations" the default https://bugs.python.org/issue38605
- Hide static types from the limited C API: https://bugs.python.org/issue40601

CHAPTER 40

See also

- Victor Stinner's Notes
- Tutorial to contribute to the CPython project's documentation!
- Victor Stinner's Blog 3

CHAPTER 41

Indices and tables

- genindex
- modindex
- search

Index

P