

---

# **f2format**

***Release 0.8.7rc1***

**Python Backport Compiler Project**

**Nov 21, 2020**



# CONTENTS

<b>1</b>	<b>Usage</b>	<b>3</b>
1.1	Specifying Files And Directories To Convert . . . . .	3
1.2	Archiving And Recovering Files . . . . .	4
1.3	Conversion Options . . . . .	5
1.4	Runtime Options . . . . .	5
1.5	API Usage . . . . .	6
1.6	Disutils/Setuptools Integration . . . . .	6
<b>2</b>	<b>Algorithms</b>	<b>9</b>
2.1	Basic Concepts . . . . .	9
2.2	Concatenable Strings . . . . .	9
2.3	Debug F-Strings . . . . .	10
<b>3</b>	<b>API Reference</b>	<b>11</b>
3.1	Public Interface . . . . .	11
3.2	Conversion Implementation . . . . .	13
3.2.1	Data Structures . . . . .	13
3.2.2	Conversion Contexts . . . . .	13
3.3	Internal Auxiliaries . . . . .	17
3.3.1	Options & Defaults . . . . .	17
3.3.2	CLI Utilities . . . . .	19
<b>4</b>	<b>Installation</b>	<b>21</b>
<b>5</b>	<b>Usage</b>	<b>23</b>
<b>6</b>	<b>Indices and tables</b>	<b>25</b>
	<b>Python Module Index</b>	<b>27</b>
	<b>Index</b>	<b>29</b>



Write *formatted string literals* in Python 3.8 flavour, and let `f2format` worry about back-port issues

Since [PEP 498](#), Python introduced *formatted string literals* syntax in version **3.6**. For those who wish to use *formatted string literals* in their code, `f2format` provides an intelligent, yet imperfect, solution of a **backport compiler** by replacing *formatted string literals* syntax with old-fashioned syntax, which guarantees you to always write *formatted string literals* in Python 3.8 flavour then compile for compatibility later.



## 1.1 Specifying Files And Directories To Convert

To convert a single file:

```
$ cat myscript.py
print(hello := 'world')
$ f2format myscript.py
Now converting: '/path/to/project/myscript.py'
$ cat myscript.py # file overwritten with conversion result
if False:
    hello = NotImplemented

def __f2format_wrapper_hello_5adbf5ee911449cba75e35b9ef97ea80(expr):
    """Wrapper function for assignment expression."""
    global hello
    hello = expr
    return hello

print(__f2format_wrapper_hello_5adbf5ee911449cba75e35b9ef97ea80('world'))
```

To convert the whole project at the current working directory (overwrites all Python source files inside):

```
$ f2format .
```

Multiple files and directories may be supplied at the same time:

```
$ f2format script_without_py_extension /path/to/another/project
```

---

**Note:** When converting a directory, `f2format` will recursively find all the Python source files in the directory (and its subdirectories, if any). Whether a file is a Python source file is determined by its file extension (`.py` or `.pyw`). If you want to convert a file without a Python extension, you will need to explicitly specify it in the argument list.

---

If you prefer a side-effect free behavior (do not overwrite files), you can use the **simple mode**.

Simple mode with no arguments (read from stdin, write to stdout):

```
$ printf 'print(hello := "world")\n' | python3 f2format.py -s
if False:
    hello = NotImplemented
```

(continues on next page)

(continued from previous page)

```
def __f2format_wrapper_hello_fbf3a9dabd2b40348815e3f2b22a1683(expr):
    """Wrapper function for assignment expression."""
    global hello
    hello = expr
    return hello

print(__f2format_wrapper_hello_fbf3a9dabd2b40348815e3f2b22a1683("world"))
```

Simple mode with a file name argument (read from file, write to stdout):

```
$ cat myscript.py
print(hello := 'world')
$ f2format -s myscript.py
if False:
    hello = NotImplemented

def __f2format_wrapper_hello_d1e6c2a11a76400aa9745bd90b3fb52a(expr):
    """Wrapper function for assignment expression."""
    global hello
    hello = expr
    return hello

print(__f2format_wrapper_hello_d1e6c2a11a76400aa9745bd90b3fb52a('world'))
$ cat myscript.py
print(hello := 'world')
```

In simple mode, no file names shall be provided via positional arguments.

## 1.2 Archiving And Recovering Files

If you are not using the simple mode, `f2format` overwrites Python source files, which could potentially cause data loss. Therefore, a built-in archiving functionality is enabled by default. The original copies of the Python source files to be converted will be packed into an archive file and placed under the `archive` subdirectory of the current working directory.

To opt out of archiving, use the CLI option `-na` (`--no-archive`), or set environment variable `F2FORMAT_DO_ARCHIVE=0`.

To use an alternative name for the archive directory (other than `archive`), use the CLI option `-k` (`--archive-path`), or set the environment variable `F2FORMAT_ARCHIVE_PATH`.

To recover files from an archive file, use the CLI option `-r` (`--recover`):

```
$ f2format -r archive/archive-20200814222751-f3a514d40d69c6d5.tar.gz
Recovered files from archive: 'archive/archive-20200814222751-f3a514d40d69c6d5.tar.gz'
$ ls archive/
archive-20200814222751-f3a514d40d69c6d5.tar.gz
```

By default, the archive file is still retained after recovering from it. If you would like it to be removed after recovery, specify the CLI option `-r2`:



```
$ f2format -r archive/archive-20200814222751-f3a514d40d69c6d5.tar.gz -r2
Recovered files from archive: 'archive/archive-20200814222751-f3a514d40d69c6d5.tar.gz'
$ ls archive/
```

If you also want to remove the archive directory if it becomes empty, specify the CLI option `-r3`:

```
$ f2format -r archive/archive-20200814222751-f3a514d40d69c6d5.tar.gz -r3
Recovered files from archive: 'archive/archive-20200814222751-f3a514d40d69c6d5.tar.gz'
$ ls archive/
ls: cannot access 'archive/': No such file or directory
```

**Warning:** To improve stability of file recovery, the archive file records the original absolute paths of the Python source files. Thus, file recovery is only guaranteed to work correctly on **the same machine** where the archive file was created. Never perform the recovery operation on an arbitrary untrusted archive file. Doing so may allow attackers to overwrite any files in the system.

## 1.3 Conversion Options

By default, `f2format` automatically detects file line endings and use the same line ending for code insertion. If you want to manually specify the line ending to be used, use the CLI option `-l` (`--linesep`) or the `F2FORMAT_LINESEP` environment variable.

By default, `f2format` automatically detects file indentations and use the same indentation for code insertion. If you want to manually specify the indentation to be used, use the CLI option `-t` (`--indentation`) or the `F2FORMAT_INDENTATION` environment variable.

By default, `f2format` parse Python source files as the latest version. If you want to manually specify a version for parsing, use the CLI option `-vs` (`-vf`, `--source-version`, `--from-version`) or the `F2FORMAT_SOURCE_VERSION` environment variable.

By default, code insertion of `f2format` conforms to [PEP 8](#). To opt out and get a more compact result, specify the CLI option `-n8` (`--no-pep8`) or set environment variable `F2FORMAT_PEP8=0`.

## 1.4 Runtime Options

Specify the CLI option `-q` (`--quiet`) or set environment variable `F2FORMAT_QUIET=1` to run in quiet mode.

Specify the CLI option `-C` (`--concurrency`) or set environment variable `F2FORMAT_CONCURRENCY` to specify the number of concurrent worker processes for conversion.

Use the `--dry-run` CLI option to list the files to be converted without actually performing conversion and archiving.

By running `f2format --help`, you can see the current values of all the options, based on their default values and your environment variables.

## 1.5 API Usage

If you want to programmatically invoke `f2format`, you may want to look at [API Reference](#). The `f2format.convert()` and `f2format.f2format()` functions should be most commonly used.

## 1.6 Disutils/Setuptools Integration

`f2format` can also be directly integrated within your `setup.py` script to dynamically convert *assignment expressions* upon installation:

```
import subprocess
import sys

try:
    from setuptools import setup
    from setuptools.command.build_py import build_py
except ImportError:
    from distutils.core import setup
    from distutils.command.build_py import build_py

version_info = sys.version_info[:2]

class build(build_py):
    """Add on-build backport code conversion."""

    def run(self):
        if version_info < (3, 8):
            try:
                subprocess.check_call( # nosec
                    [sys.executable, '-m', 'f2format', '--no-archive', 'PACKAGENAME']
                )
            except subprocess.CalledProcessError as error:
                print('Failed to perform assignment expression backport compiling.'
                    'Please consider manually install `bpc-f2format` and try again.'
                    '\n', file=sys.stderr)
                sys.exit(error.returncode)
            build_py.run(self)

setup(
    ...
    setup_requires=[
        'bpc-f2format; python_version < "3.8"',
    ],
    cmdclass={
        'build_py': build,
    },
)
```

Or, as [PEP 518](#) proposed, you may simply add `bpc-f2format` to the `requires` list from the `[build-system]` section in the `pyproject.toml` file:

```
[build-system]
# Minimum requirements for the build system to execute.
```

(continues on next page)

(continued from previous page)

```
requires = ["setuptools", "wheel", "bpc-f2format"] # PEP 508 specifications.  
...
```



## ALGORITHMS

As discussed in [PEP 498](#), *formatted string literals (f-string)* is a way to interpolate evaluated expression values into regular string literals, using the syntax `f ' <text> { <expression> <optional !s, !r, or !a> <optional : format specifier> } <text> ... '`. It is roughly equivalent to first evaluate the value of `expression` then interpolate its value into the string literal with provided format specifiers and conversion.

### 2.1 Basic Concepts

To convert, `f2format` will first extract all *expressions* from the *f-strings*, then rewrite the literal with a `str.format` function using anonymous positional expression sequence from the extracted expressions.

For example, with the samples from [PEP 498](#):

```
f'abc{expr1:spec1}{expr2!r:spec2}def{expr3}ghi'
```

it should be converted to

```
'abc{:spec1}{:spec2}def{}ghi'.format(expr1, expr2, expr3)
```

### 2.2 Concatenable Strings

As mentioned in the [Python documentation](#), multiple adjacent string or bytes literals (delimited by whitespace), possibly using different quoting conventions, are allowed, and their meaning is the same as their concatenation.

In cases where a *f-string* can be found in such sequence of concatenable strings, directly converting the *f-string* to `str.format` syntax may cause the concatenation to be broken. Therefore, `f2format` will rather insert the converted `.format` call at the end of the string sequence.

For example, a sequence of concatenable strings may be as follows:

```
(('/usr/local/opt/python/bin/python3.7 -c "'
'import re, sys\n'
'for line in sys.stdin:\n'
'    data = line.rstrip().replace('^D\x08\x08', '')\n'
'    temp = re.sub(r'\x1b\\[[0-9][0-9;]*m', r'', data, flags=re.IGNORECASE)\n'
f'    text = temp.replace('Password:', 'Password:\\r\\n') (_replace(password) )\n'
'    if text:\n'
'        print(text, end='\\r\\n')\n'
'    )
')
```

then `f2format` will convert the code above as

```
('/usr/local/opt/python/bin/python3.7 -c "'
'import re, sys\n'
'for line in sys.stdin:\n'
"    data = line.rstrip().replace('^D\x08\x08', '')\n"
"    temp = re.sub(r'\x1b\\[[0-9][0-9;]*m', r'', data, flags=re.IGNORECASE)\n"
"    text = temp.replace('Password:', 'Password:\\r\\n'){}\n"
"    if text:\n"
"        print(text, end='\\r\\n')\n"
"'.format(_replace(password)))
```

## 2.3 Debug F-Strings

Since Python 3.8, `=` was introduced to *f-strings* in addition to the acceptance of [bpo-36817](#). As discussed in the [Python documentation](#), when the equal sign `'='` is provided, the output will have the expression text, the `'='` and the evaluated value, therefore `f2format` tend to keep an original copy of the expressions in the converted strings then append `str.format` with corresponding expressions.

For a *f-string* as below:

```
>>> foo = "bar"
>>> f"{ foo = }" # preserves whitespace
" foo = 'bar'"
```

`f2format` will convert it as

```
" foo = {!r}".format(foo)
```

## API REFERENCE

### 3.1 Public Interface

`f2format.convert` (*code*, *filename=None*, \*, *source\_version=None*, *linesep=None*, *indentation=None*,  
*pep8=None*)

Convert the given Python source code string.

#### Parameters

- **code** (*Union[str, bytes]*) – the source code to be converted
- **filename** (*Optional[str]*) – an optional source file name to provide a context in case of error
- **source\_version** (*Optional[str]*) –
- **linesep** (*Optional[Literal[ , , ]]*) –
- **indentation** (*Optional[Union[str, int]]*) –
- **pep8** (*Optional[bool]*) –

**Keyword Arguments** **source\_version** (*Optional[str]*) – parse the code as this Python version (uses the latest version by default)

#### Environment Variables

- **F2FORMAT\_SOURCE\_VERSION** – same as the **source\_version** argument and the **--source-version** option in CLI
- **F2FORMAT\_LINESEP** – same as the **linesep** argument and the **--linesep** option in CLI
- **F2FORMAT\_INDENTATION** – same as the **indentation** argument and the **--indentation** option in CLI
- **F2FORMAT\_PEP8** – same as the **pep8** argument and the **--no-pep8** option in CLI (logical negation)

**Returns** converted source code

**Return type** *str*

`f2format.f2format` (*filename*, \*, *source\_version=None*, *linesep=None*, *indentation=None*, *pep8=None*,  
*quiet=None*, *dry\_run=False*)

Convert the given Python source code file. The file will be overwritten.

#### Parameters

- **filename** (*str*) – the file to convert

- **source\_version** (*Optional[str]*) –
- **linesep** (*Optional[Literal[ , , ]]*) –
- **indentation** (*Optional[Union[str, int]]*) –
- **pep8** (*Optional[bool]*) –
- **quiet** (*Optional[bool]*) –
- **dry\_run** (*bool*) –

#### Keyword Arguments

- **source\_version** (*Optional[str]*) – parse the code as this Python version (uses the latest version by default)
- **linesep** (*Optional[str]*) – line separator of code (LF, CRLF, CR) (auto detect by default)
- **indentation** (*Optional[Union[int, str]]*) – code indentation style, specify an integer for the number of spaces, or 't'/'tab' for tabs (auto detect by default)
- **pep8** (*Optional[bool]*) – whether to make code insertion **PEP 8** compliant
- **quiet** (*Optional[bool]*) – whether to run in quiet mode
- **dry\_run** (*bool*) – if **True**, only print the name of the file to convert but do not perform any conversion

#### Environment Variables

- **F2FORMAT\_SOURCE\_VERSION** – same as the **source-version** argument and the **--source-version** in CLI
- **F2FORMAT\_LINESEP** – same as the **linesep** argument and the **--linesep** option in CLI
- **F2FORMAT\_INDENTATION** – same as the **indentation** argument and the **--indentation** option in CLI
- **F2FORMAT\_PEP8** – same as the **pep8** argument and the **--no-pep8** option in CLI (logical negation)
- **F2FORMAT\_QUIET** – same as the **quiet** argument and the **--quiet** option in CLI

**Return type** `None`

`f2format.main` (*argv=None*)

Entry point for f2format.

**Parameters** **argv** (*Optional[List[str]]*) – CLI arguments

#### Environment Variables

- **F2FORMAT\_QUIET** – same as the **--quiet** option in CLI
- **F2FORMAT\_CONCURRENCY** – same as the **--concurrency** option in CLI
- **F2FORMAT\_DO\_ARCHIVE** – same as the **--no-archive** option in CLI (logical negation)
- **F2FORMAT\_ARCHIVE\_PATH** – same as the **--archive-path** option in CLI
- **F2FORMAT\_SOURCE\_VERSION** – same as the **--source-version** option in CLI
- **F2FORMAT\_LINESEP** – same as the **--linesep** option in CLI



- `F2FORMAT_INDENTATION` – same as the `--indentation` option in CLI
- `F2FORMAT_PEP8` – same as the `--no-pep8` option in CLI (logical negation)

Return type `int`

## 3.2 Conversion Implementation

The main logic of the `f2format` conversion is to extract the expressions part from *formatted string literals* and rewrite the original *f-string* using `str.format` syntax.

For conversion algorithms and details, please refer to *Algorithms*.

### 3.2.1 Data Structures

During conversion, we utilised `bpc_utils.Config` to store and deliver the configurations over the conversion `Context` instances, which should be as following:

```
class f2format.Config
    Configuration object shared over the conversion process of a single source file.

    indentation: str
        Indentation sequence.

    linesep: Literal['\n', '\r\n', '\r']
        Line separator.

    pep8: bool
        PEP 8 compliant conversion flag.

    filename: Optional[str]
        An optional source file name to provide a context in case of error.

    source_version: Optional[str]
        Parse the code as this Python version (uses the latest version by default).
```

### 3.2.2 Conversion Contexts

```
class f2format.Context (node, config, *, indent_level=0, raw=False)
    Bases: bpc_utils.context.BaseContext
```

General conversion context.

#### Parameters

- **node** (`parso.tree.NodeOrLeaf`) – parso AST
- **config** (`Config`) – conversion configurations
- **indent\_level** (`int`) –
- **raw** (`bool`) –

**Keyword Arguments** **raw** (`bool`) – raw processing flag

Return type `None`

---

**Important:** `raw` should be `True` only if the node is in the clause of another *context*, where the converted wrapper functions should be inserted.

---

For the *Context* class of *f2format* module, it will process nodes with following methods:

- `stringliteral`
  - `Context._process_strings()`
  - `Context._process_string_context()`
- `f_string`
  - `Context._process_fstring()`

Initialize BaseContext.

**Parameters**

- **node** (*NodeOrLeaf*) – parso AST
- **config** (*Config*) – conversion configurations
- **indent\_level** (*int*) – current indentation level
- **raw** (*bool*) – raw processing flag

**Return type** `None`

**`_concat()`**

Concatenate final string.

This method tries to concatenate final result based on the very location where starts to contain formatted string literals, i.e. between the converted code as `self._prefix` and `self._suffix`.

**Return type** `None`

**`_process_fstring(node)`**

Process formatted strings (*f\_string*).

**Parameters** **node** (*parso.python.tree.PythonNode*) – formatted strings node

**Return type** `None`

**`_process_strings(node)`**

Process concatenable strings (*stringliteral*).

**Parameters** **node** (*parso.python.tree.PythonNode*) – concatentable strings node

**Return type** `None`

As in Python, adjacent string literals can be concatenated in certain cases, as described in the **documentation**. Such concatenable strings may contain formatted string literals (*f-string*) within its scope.

`_documentation`: [https://docs.python.org/3/reference/lexical\\_analysis.html#string-literal-concatenation](https://docs.python.org/3/reference/lexical_analysis.html#string-literal-concatenation)

**Return type** `None`

**Parameters** **node** (*parso.python.tree.PythonNode*) –

**`classmethod has_debug_fstring(node)`**

Check if node has *debug* formatted string literals.

**Parameters** **node** (*parso.tree.NodeOrLeaf*) – parso AST

**Returns** if node has debug formatted string literals

**Return type** `bool`

**classmethod** `has_expr (node)`

Check if node has formatted string literals.

**Parameters** `node` (`parso.tree.NodeOrLeaf`) – parso AST

**Returns** if node has formatted string literals

**Return type** `bool`

**classmethod** `has_f2format (node)`

Check if node has formatted string literals.

**Parameters** `node` (`parso.tree.NodeOrLeaf`) – parso AST

**Returns** if node has formatted string literals

**Return type** `bool`

`_abc_impl = <_abc_data object>`

**class** `f2format.StringContext (node, config, *, has_fstring=None, indent_level=0, raw=False)`

Bases: `f2format.Context`

String (f-string) conversion context.

This class is mainly used for converting **formatted string literals**.

#### Parameters

- `node` (`parso.python.tree.PythonNode`) – parso AST
- `config` (`Config`) – conversion configurations
- `has_fstring` (`Optional[bool]`) –
- `indent_level` (`int`) –
- `raw` (`bool`) –

#### Keyword Arguments

- `has_fstring` (`bool`) – flag if contains actual formatted string literals (with expressions)
- `indent_level` (`int`) – current indentation level
- `raw` (`bool`) – raw processing flag

Initialize BaseContext.

#### Parameters

- `node` (`NodeOrLeaf`) – parso AST
- `config` (`Config`) – conversion configurations
- `indent_level` (`int`) – current indentation level
- `raw` (`bool`) – raw processing flag
- `has_fstring` (`Optional[bool]`) –

`_concat ()`

Concatenate final string.

This method tries to concatenate final result based on the very location where starts to contain formatted string literals, i.e. between the converted code as `self._prefix` and `self._suffix`.

**Return type** `None`

**`_process_fstring`** (*node*)  
Process formatted strings (*f\_string*).  
**Parameters** *node* (*parso.python.tree.PythonNode*) – formatted strings node  
**Return type** `None`

**`_process_fstring_expr`** (*node*)  
Process formatted string literal expression node (*f\_expression*).  
**Parameters** *node* (*parso.python.tree.PythonNode*) – formatted literal expression node  
**Return type** `None`

**`_process_fstring_start`** (*node*)  
Process formatted string literal starting node (*stringprefix*).  
**Parameters** *node* (*parso.python.tree.FStringStart*) – formatted literal starting node  
**Return type** `None`

**`_process_fstring_string`** (*node*)  
Process formatted string literal string node (*stringliteral*).  
**Parameters** *node* (*parso.python.tree.FStringString*) – formatted string literal string node  
**Return type** `None`

**`_process_string`** (*node*)  
Process string node (*stringliteral*).  
**Parameters** *node* (*parso.python.tree.PythonNode*) – string node  
**Return type** `None`

**`classmethod has_fstring`** (*node*)  
Check if node has actual formatted string literals.  
**Parameters** *node* (*parso.tree.NodeOrLeaf*) – parso AST  
**Returns**  
    **if node has actual formatted string literals** (with expressions in the literals)  
**Return type** `bool`

**`_abc_impl`** = `<_abc_data object>`

**`_expr`**: `List[str]`  
Expressions extracted from the formatted string literal.  
**Type** `List[str]`

**`_flag`**: `bool`  
Flag if contains actual formatted string literals (with expressions).  
**Type** `bool`

**`property expr`**  
Expressions extracted from the formatted string literal.  
**Return type** `List[str]`

**fstring\_bracket:** `Final[ClassVar[re.Pattern]] = re.compile('([{}])', re.ASCII)`  
 Pattern matches single brackets in the formatted string literal (`{}`).

Type `re.Pattern`

**fstring\_start:** `Final[ClassVar[re.Pattern]] = re.compile('[fF]', re.ASCII)`  
 Pattern matches the formatted string literal prefix (`f`).

Type `re.Pattern`

## 3.3 Internal Auxiliaries

### 3.3.1 Options & Defaults

Below are option getter utility functions. Option value precedence is:

explicit value (CLI/API arguments) > environment variable > default value

`f2format._get_quiet_option(explicit=None)`

Get the value for the `quiet` option.

**Parameters** `explicit` (`Optional[bool]`) – the value explicitly specified by user, `None` if not specified

**Returns** the value for the `quiet` option

**Return type** `bool`

**Environment Variables** `F2FORMAT_QUIET` – the value in environment variable

See also:

`_default_quiet`

`f2format._get_concurrency_option(explicit=None)`

Get the value for the `concurrency` option.

**Parameters** `explicit` (`Optional[int]`) – the value explicitly specified by user, `None` if not specified

**Returns** the value for the `concurrency` option; `None` means *auto detection* at runtime

**Return type** `Optional[int]`

**Environment Variables** `F2FORMAT_CONCURRENCY` – the value in environment variable

See also:

`_default_concurrency`

`f2format._get_do_archive_option(explicit=None)`

Get the value for the `do_archive` option.

**Parameters** `explicit` (`Optional[bool]`) – the value explicitly specified by user, `None` if not specified

**Returns** the value for the `do_archive` option

**Return type** `bool`

**Environment Variables** `F2FORMAT_DO_ARCHIVE` – the value in environment variable

See also:

`_default_do_archive`

`f2format._get_archive_path_option (explicit=None)`

Get the value for the `archive_path` option.

**Parameters** `explicit` (*Optional*[*str*]) – the value explicitly specified by user, `None` if not specified

**Returns** the value for the `archive_path` option

**Return type** *str*

**Environment Variables** `F2FORMAT_ARCHIVE_PATH` – the value in environment variable

See also:

`_default_archive_path`

`f2format._get_source_version_option (explicit=None)`

Get the value for the `source_version` option.

**Parameters** `explicit` (*Optional*[*str*]) – the value explicitly specified by user, `None` if not specified

**Returns** the value for the `source_version` option

**Return type** *str*

**Environment Variables** `F2FORMAT_SOURCE_VERSION` – the value in environment variable

See also:

`_default_source_version`

`f2format._get_linesep_option (explicit=None)`

Get the value for the `linesep` option.

**Parameters** `explicit` (*Optional*[*str*]) – the value explicitly specified by user, `None` if not specified

**Returns** the value for the `linesep` option; `None` means *auto detection* at runtime

**Return type** *Optional*[*Literal*['\n', '\r\n', '\r']]

**Environment Variables** `F2FORMAT_LINESEP` – the value in environment variable

See also:

`_default_linesep`

`f2format._get_indentation_option (explicit=None)`

Get the value for the `indentation` option.

**Parameters** `explicit` (*Optional*[*Union*[*str*, *int*]]) – the value explicitly specified by user, `None` if not specified

**Returns** the value for the `indentation` option; `None` means *auto detection* at runtime

**Return type** *Optional*[*str*]

**Environment Variables** `F2FORMAT_INDENTATION` – the value in environment variable

See also:

`_default_indentation`

`f2format._get_pep8_option` (*explicit=None*)

Get the value for the pep8 option.

**Parameters** `explicit` (*Optional[bool]*) – the value explicitly specified by user, `None` if not specified

**Returns** the value for the pep8 option

**Return type** `bool`

**Environment Variables** `F2FORMAT_PEP8` – the value in environment variable

**See also:**

`_default_pep8`

The following variables are used for fallback default values of options.

`f2format._default_quiet = False`

Default value for the `quiet` option.

`f2format._default_concurrency = None`

Default value for the `concurrency` option.

`f2format._default_do_archive = True`

Default value for the `do_archive` option.

`f2format._default_archive_path = 'archive'`

Default value for the `archive_path` option.

`f2format._default_source_version = '3.10'`

Default value for the `source_version` option.

`f2format._default_linesep = None`

Default value for the `linesep` option.

`f2format._default_indentation = None`

Default value for the `indentation` option.

`f2format._default_pep8 = True`

Default value for the `pep8` option.

---

**Important:** For `_default_concurrency`, `_default_linesep` and `_default_indentation`, `None` means *auto detection* during runtime.

---

### 3.3.2 CLI Utilities

`f2format.get_parser()`

Generate CLI parser.

**Returns** CLI parser for f2format

**Return type** `argparse.ArgumentParser`

The following variables are used for help messages in the argument parser.

`f2format.__cwd__`: `str`

Current working directory returned by `os.getcwd()`.

`f2format.__f2format_quiet__`: `Literal['quiet mode', 'non-quiet mode']`

Default value for the `--quiet` option.

**See also:**

*f2format.\_get\_quiet\_option()*

`f2format.__f2format_concurrency__`: `Union[int, Literal['auto detect']]`

Default value for the `--concurrency` option.

**See also:**

*f2format.\_get\_concurrency\_option()*

`f2format.__f2format_do_archive__`: `Literal['will do archive', 'will not do archive']`

Default value for the `--no-archive` option.

**See also:**

*f2format.\_get\_do\_archive\_option()*

`f2format.__f2format_archive_path__`: `str`

Default value for the `--archive-path` option.

**See also:**

*f2format.\_get\_archive\_path\_option()*

`f2format.__f2format_source_version__`: `str`

Default value for the `--source-version` option.

**See also:**

*f2format.\_get\_source\_version\_option()*

`f2format.__f2format_linesep__`: `Literal['LF', 'CRLF', 'CR', 'auto detect']`

Default value for the `--linesep` option.

**See also:**

*f2format.\_get\_linesep\_option()*

`f2format.__f2format_indentation__`: `str`

Default value for the `--indentation` option.

**See also:**

*f2format.\_get\_indentation\_option()*

`f2format.__f2format_pep8__`: `Literal['will conform to PEP 8', 'will not conform to PEP 8']`

Default value for the `--no-pep8` option.

**See also:**

*f2format.\_get\_pep8\_option()*



## INSTALLATION

---

**Note:** `f2format` only supports Python versions **since 3.4**

---

For macOS users, `f2format` is available through [Homebrew](#):

```
brew tap jarryshaw/tap
brew install f2format
# or simply, a one-liner
brew install jarryshaw/tap/f2format
```

You can also install from [PyPI](#) for any OS:

```
pip install bpc-f2format
```

Or install the latest version from the [Git repository](#):

```
git clone https://github.com/pybpc/f2format.git
cd f2format
pip install -e .
# and to update at any time
git pull
```

---

**Note:** Installation from [Homebrew](#) will also automatically install the man page and [Bash Completion](#) script for you. If you are installing from [PyPI](#) or the [Git repository](#), you can install the completion script manually.

---



---

CHAPTER  
**FIVE**

---

**USAGE**

See *Usage*.



## INDICES AND TABLES

- `genindex`
- `modindex`
- `search`



## PYTHON MODULE INDEX

### f

f2format, [11](#)





## Symbols

[\\_abc\\_impl \(f2format.Context attribute\), 15](#)  
[\\_abc\\_impl \(f2format.StringContext attribute\), 16](#)  
[\\_concat \(\) \(f2format.Context method\), 14](#)  
[\\_concat \(\) \(f2format.StringContext method\), 15](#)  
[\\_default\\_archive\\_path \(in module f2format\), 19](#)  
[\\_default\\_concurrency \(in module f2format\), 19](#)  
[\\_default\\_do\\_archive \(in module f2format\), 19](#)  
[\\_default\\_indentation \(in module f2format\), 19](#)  
[\\_default\\_linesep \(in module f2format\), 19](#)  
[\\_default\\_pep8 \(in module f2format\), 19](#)  
[\\_default\\_quiet \(in module f2format\), 19](#)  
[\\_default\\_source\\_version \(in module f2format\), 19](#)  
[\\_expr \(f2format.StringContext attribute\), 16](#)  
[\\_flag \(f2format.StringContext attribute\), 16](#)  
[\\_get\\_archive\\_path\\_option \(\) \(in module f2format\), 18](#)  
[\\_get\\_concurrency\\_option \(\) \(in module f2format\), 17](#)  
[\\_get\\_do\\_archive\\_option \(\) \(in module f2format\), 17](#)  
[\\_get\\_indentation\\_option \(\) \(in module f2format\), 18](#)  
[\\_get\\_linesep\\_option \(\) \(in module f2format\), 18](#)  
[\\_get\\_pep8\\_option \(\) \(in module f2format\), 18](#)  
[\\_get\\_quiet\\_option \(\) \(in module f2format\), 17](#)  
[\\_get\\_source\\_version\\_option \(\) \(in module f2format\), 18](#)  
[\\_process\\_fstring \(\) \(f2format.Context method\), 14](#)  
[\\_process\\_fstring \(\) \(f2format.StringContext method\), 15](#)  
[\\_process\\_fstring\\_expr \(\) \(f2format.StringContext method\), 16](#)  
[\\_process\\_fstring\\_start \(\) \(f2format.StringContext method\), 16](#)  
[\\_process\\_fstring\\_string \(\) \(f2format.StringContext method\), 16](#)  
[\\_process\\_string \(\) \(f2format.StringContext method\), 16](#)  
[\\_process\\_strings \(\) \(f2format.Context method\), 14](#)

14

## C

[Config \(class in f2format\), 13](#)  
[Context \(class in f2format\), 13](#)  
[convert \(\) \(in module f2format\), 11](#)

## E

[environment variable](#)  
[F2FORMAT\\_SOURCE\\_VERSION, 11, 12](#)  
[environment variable](#)  
[F2FORMAT\\_ARCHIVE\\_PATH, 12, 18](#)  
[F2FORMAT\\_CONCURRENCY, 12, 17](#)  
[F2FORMAT\\_DO\\_ARCHIVE, 12, 17](#)  
[F2FORMAT\\_INDENTATION, 11–13, 18](#)  
[F2FORMAT\\_LINESEP, 11, 12, 18](#)  
[F2FORMAT\\_PEP8, 11–13, 19](#)  
[F2FORMAT\\_QUIET, 12, 17](#)  
[F2FORMAT\\_SOURCE\\_VERSION, 12, 18](#)  
[expr \(\) \(f2format.StringContext property\), 16](#)

## F

[f2format](#)  
[module, 11](#)  
[f2format \(\) \(in module f2format\), 11](#)  
[f2format.\\_\\_cwd\\_\\_ \(in module f2format\), 19](#)  
[f2format.\\_\\_f2format\\_archive\\_path\\_\\_ \(in module f2format\), 20](#)  
[f2format.\\_\\_f2format\\_concurrency\\_\\_ \(in module f2format\), 20](#)  
[f2format.\\_\\_f2format\\_do\\_archive\\_\\_ \(in module f2format\), 20](#)  
[f2format.\\_\\_f2format\\_indentation\\_\\_ \(in module f2format\), 20](#)  
[f2format.\\_\\_f2format\\_linesep\\_\\_ \(in module f2format\), 20](#)  
[f2format.\\_\\_f2format\\_pep8\\_\\_ \(in module f2format\), 20](#)  
[f2format.\\_\\_f2format\\_quiet\\_\\_ \(in module f2format\), 19](#)  
[f2format.\\_\\_f2format\\_source\\_version\\_\\_ \(in module f2format\), 20](#)

F2FORMAT\_ARCHIVE\_PATH, [12](#), [18](#)  
F2FORMAT\_CONCURRENCY, [12](#), [17](#)  
F2FORMAT\_DO\_ARCHIVE, [12](#), [17](#)  
F2FORMAT\_INDENTATION, [11–13](#), [18](#)  
F2FORMAT\_LINESEP, [11](#), [12](#), [18](#)  
F2FORMAT\_PEP8, [11–13](#), [19](#)  
F2FORMAT\_QUIET, [12](#), [17](#)  
F2FORMAT\_SOURCE\_VERSION, [11](#), [12](#)  
F2FORMAT\_SOURCE\_VERSION, [12](#), [18](#)  
filename (*f2format.Config* attribute), [13](#)  
fstring\_bracket (*f2format.StringContext* attribute),  
[16](#)  
fstring\_start (*f2format.StringContext* attribute), [17](#)

## G

get\_parser() (*in module f2format*), [19](#)

## H

has\_debug\_fstring() (*f2format.Context* class  
method), [14](#)  
has\_expr() (*f2format.Context* class method), [15](#)  
has\_f2format() (*f2format.Context* class method), [15](#)  
has\_fstring() (*f2format.StringContext* class  
method), [16](#)

## I

indentation (*f2format.Config* attribute), [13](#)

## L

linesep (*f2format.Config* attribute), [13](#)

## M

main() (*in module f2format*), [12](#)  
module  
    f2format, [11](#)

## P

pep8 (*f2format.Config* attribute), [13](#)  
Python Enhancement Proposals  
    PEP 498, [1](#), [9](#)  
    PEP 518, [6](#)  
    PEP 8, [5](#), [12](#), [13](#)

## S

source\_version (*f2format.Config* attribute), [13](#)  
StringContext (*class in f2format*), [15](#)