bpc-utils

Release 0.7.0

Python Backport Compiler Project

Sep 08, 2020

CONTENTS

Utility library for the Python `bpc` backport compiler.

Currently, the three individual tools (`f2format`, `poseur`, `walrus`) depend on this repo. The `bpc` compiler is a work in progress.

CHAPTER

ONE

MODULE CONTENTS

exception bpc_utils.**BPCSyntaxError**
Bases: SyntaxError

Syntax error detected when parsing code.

class bpc_utils.**BaseContext**(*node*, *config*, *, *indent_level=0*, *raw=False*)
Bases: abc.ABC

Abstract base class for general conversion context.

__iadd__(*code*)
Support of the += operator.

If `self._prefix_or_suffix` is `True`, then the `code` will be appended to `self._prefix`; else it will be appended to `self._suffix`.

Parameters **code** (*str*) – code string

Returns self

Return type *BaseContext*

__init__(*node*, *config*, *, *indent_level=0*, *raw=False*)
Initialize BaseContext.

Parameters

- **node** (*parso.tree.NodeOrLeaf*) – parso AST
- **config** (Config) – conversion configurations

Keyword Arguments

- **indent_level** (*int*) – current indentation level
- **raw** (*bool*) – raw processing flag

__str__()
Returns a *stripped* version of `self._buffer`.

abstract _concat()
Concatenate final string.

_process(*node*)
Recursively process parso AST.

All processing methods for a specific `node` type are defined as `_process_{type}`. This method first checks if such processing method exists. If so, it will call such method on the `node`; otherwise it will traverse through all *children* of `node`, and perform the same logic on each child.

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

_walk(*node*)
Start traversing the AST module.

The method traverses through all *children* of `node`. It first checks if such child has the target expression. If so, it will toggle `self._prefix_or_suffix` (set to `False`) and save the last previous child as `self._node_before_expr`. Then it processes the child with *`self._process`*.

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

static extract_whitespaces(*node*)
Extract preceding and succeeding whitespaces from the node given.

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

Returns a tuple of *preceding* and *succeeding* whitespaces in `node`

Return type Tuple[str, str]

abstract has_expr(*node*)
Check if node has the target expression.

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

Returns if `node` has the target expression

Return type bool

static missing_newlines(*prefix*, *suffix*, *expected*, *linesep*)
Count missing blank lines for code insertion given surrounding code.

Parameters

- **prefix** (*str*) – preceding source code
- **suffix** (*str*) – succeeding source code
- **expected** (*int*) – number of expected blank lines
- **linesep** (*str*) – line seperator

Returns number of blank lines to add

Return type int

static split_comments(*code*, *linesep*)
Separates prefixing comments from code.

This method separates *prefixing* comments and *suffixing* code. It is rather useful when inserting code might break shebang and encoding cookies (**PEP 263**), etc.

Parameters

- **code** (*str*) – the code to split comments
- **linesep** (*str*) – line seperator

Returns a tuple of *prefix comments* and *suffix code*

Return type Tuple[str, str]

config
Internal configurations.

Type *Config*

property string
Returns conversion buffer (`self._buffer`).

class bpc_utils.**Config**(***kwargs*)
Bases: collections.abc.MutableMapping

Configuration namespace.

This class is inspired from argparse.Namespace for storing internal attributes and/or configuration variables.

class bpc_utils.**UUID4Generator**(*dash=True*)
Bases: object

UUID 4 generator wrapper to prevent UUID collisions.

__init__(*dash=True*)
Constructor of UUID 4 generator wrapper.

Parameters **dash** (*bool*) – whether the generated UUID string has dashes or not

gen()
Generate a new UUID 4 string that is guaranteed not to collide with used UUIDs.

Returns a new UUID 4 string

Return type str

bpc_utils.**TaskLock**()
Function that returns a lock for possibly concurrent tasks.

Returns a lock for possibly concurrent tasks

Return type Union[contextlib.nullcontext, multiprocessing.synchronize.Lock]

bpc_utils.**archive_files**(*files*, *archive_dir*)
Archive the list of files into a *tar* file.

Parameters

- **files** (*List[str]*) – a list of files to be archived (should be *absolute path*)
- **archive_dir** (*os.PathLike*) – the directory to save the archive

Returns path to the generated *tar* archive

Return type str

bpc_utils.**detect_encoding**(*code*)
Detect encoding of Python source code as specified in **PEP 263**.

Parameters **code** (*bytes*) – the code to detect encoding

Returns the detected encoding, or the default encoding (utf-8)

Return type str

Raises **TypeError** – if code is not a bytes string

bpc_utils.**detect_files**(*files*)
Get a list of Python files to be processed according to user input.

This will perform *glob* expansion on Windows, make all paths absolute, resolve symbolic links and remove duplicates.

Parameters **files** (*List[str]*) – a list of files and directories to process (usually provided by users on command-line)

Returns a list of Python files to be processed

Return type List[str]

See also:

See *expand_glob_iter()* for more information.

bpc_utils.**detect_indentation**(*code*)
Detect indentation of Python source code.

Parameters **code** (*Union[str, bytes, TextIO, parso.tree.NodeOrLeaf]*) – the code to detect indentation

Returns the detected indentation sequence

Return type str

Notes

In case of mixed indentation, try voting by the number of occurrences of each indentation value (*spaces* and *tabs*).

When there is a tie between *spaces* and *tabs*, prefer **4 spaces** for **PEP 8**.

---

bpc_utils.**detect_linesep**(*code*)
Detect linesep of Python source code.

Parameters **code** (*Union[str, bytes, TextIO, parso.tree.NodeOrLeaf]*) – the code to detect linesep

Returns the detected linesep (one of `'\n'`, `'\r\n'` and `'\r'`)

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

Notes

In case of mixed linesep, try voting by the number of occurrences of each linesep value.

When there is a tie, prefer `LF` to `CRLF`, prefer `CRLF` to `CR`.

---

bpc_utils.**first_non_none**(**args*)
Return the first non-`None` value from a list of values.

Parameters ***args** – variable length argument list

- If one positional argument is provided, it should be an iterable of the values.
- If two or more positional arguments are provided, then the value list is the positional argument list.

Returns the first non-`None` value, if all values are `None` or sequence is empty, return `None`

Return type Any

Raises **TypeError** – if no arguments provided

bpc_utils.**first_truthy**(**args*)
Return the first *truthy* value from a list of values.

Parameters ***args** – variable length argument list

- If one positional argument is provided, it should be an iterable of the values.
- If two or more positional arguments are provided, then the value list is the positional argument list.

Returns the first *truthy* value, if no *truthy* values found or sequence is empty, return `None`

Return type Any

Raises **`TypeError`** – if no arguments provided

`bpc_utils.`**`get_parso_grammar_versions`**(*minimum=None*)
Get Python versions that parso supports to parse grammar.

Parameters **`minimum`** (*`str`*) – filter result by this minimum version

Returns a list of Python versions that parso supports to parse grammar

Return type List[str]

Raises **`ValueError`** – if `minimum` is invalid

`bpc_utils.`**`map_tasks`**(*func*, *iterable*, *posargs=None*, *kwargs=None*, ***, *processes=None*, *chunksize=None*)
Execute tasks in parallel if `multiprocessing` is available, otherwise execute them sequentially.

Parameters

- **`func`** (*`Callable`*) – the task function to execute
- **`iterable`** (*`Iterable[Any]`*) – the items to process
- **`posargs`** (*`Optional[Iterable[Any]]`*) – additional positional arguments to pass to `func`
- **`kwargs`** (*`Optional[Mapping[str, Any]]`*) – keyword arguments to pass to `func`
- **`processes`** (*`Optional[int]`*) – the number of worker processes (default: auto determine)
- **`chunksize`** (*`Optional[int]`*) – chunk size for multiprocessing

Returns the return values of the task function applied on the input items and additional arguments

Return type List[Any]

`bpc_utils.`**`parse_boolean_state`**(*s*)
Parse a boolean state from a string representation.

- These values are regarded as `True`: `'1'`, `'yes'`, `'y'`, `'true'`, `'on'`
- These values are regarded as `False`: `'0'`, `'no'`, `'n'`, `'false'`, `'off'`

Value matching is case **insensitive**.

Parameters **`s`** (*`Optional[str]`*) – string representation of a boolean state

Returns the parsed boolean result, return `None` if input is `None`

Return type Optional[bool]

Raises **`ValueError`** – if `s` is an invalid boolean state value

See also:

See *`_boolean_state_lookup`* for default lookup mapping values.

`bpc_utils.`**`parse_indentation`**(*s*)
Parse indentation from a string representation.

- If an integer or a string of positive integer `n` is specified, then indentation is `n` spaces.
- If `'t'` or `'tab'` is specified, then indentation is tab.

- **If `'\t'` (the tab character itself) or a string consisting only of the space character (U+0020) is** specified, it is returned directly.

Value matching is **case insensitive**.

Parameters **s** (*Optional[Union[str, int]]*) – string representation of indentation

Returns the parsed indentation result, return `None` if input is `None` or empty string

Return type Optional[str]

Raises

- **TypeError** – if `s` is not `str` or `int`
- **ValueError** – if `s` is an invalid indentation value

bpc_utils.**parse_linesep**(*s*)

Parse linesep from a string representation.

- These values are regarded as `'\n'`: `'\n'`, `'lf'`
- These values are regarded as `'\r\n'`: `'\r\n'`, `'crlf'`
- These values are regarded as `'\r'`: `'\r'`, `'cr'`

Value matching is **case insensitive**.

Parameters **s** (*Optional[str]*) – string representation of linesep

Returns the parsed linesep result, return `None` if input is `None` or empty string

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

Raises **ValueError** – if `s` is an invalid linesep value

See also:

See *_linesep_lookup* for default lookup mapping values.

bpc_utils.**parse_positive_integer**(*s*)

Parse a positive integer from a string representation.

Parameters **s** (*Optional[Union[str, int]]*) – string representation of a positive integer, or just an integer

Returns the parsed integer result, return `None` if input is `None` or empty string

Return type Optional[int]

Raises

- **TypeError** – if `s` is not `str` or `int`
- **ValueError** – if `s` is an invalid positive integer value

bpc_utils.**parso_parse**(*code*, *filename=None*, ***, *version=None*)

Parse Python source code with parso.

Parameters

- **code** (*Union[str, bytes]*) – the code to be parsed
- **filename** (*str*) – an optional source file name to provide a context in case of error
- **version** (*str*) – parse the code as this version (uses the latest version by default)

Returns parso AST

Return type parso.python.tree.Module

Raises ***BPCSyntaxError*** – when source code contains syntax errors

bpc_utils.**recover_files**(*archive_file*)
Recover files from a *tar* archive.

Parameters **archive_file** (*os.PathLike*) – path to the *tar* archive file

CHAPTER

TWO

INTERNAL UTILITIES

class bpc_utils.**MakeTextIO**(*obj*)

Bases: object

Context wrapper class to handle str and *file* objects together.

Variables

- ***obj*** (*Union[str, TextIO]*) – the object to manage in the context
- ***sio*** (*Optional[StringIO]*) – the I/O object to manage in the context only if *self.obj* is str
- ***pos*** (*Optional[int]*) – the original offset of *self.obj*, only if *self.obj* is a seekable *file* object

obj: Union[str, TextIO]

The object to manage in the context.

sio: StringIO

The I/O object to manage in the context only if *self.obj* is str.

pos: int

The original offset of *self.obj*, if only *self.obj* is a seekable TextIO.

__enter__()

Enter context.

- If *self.obj* is str, a StringIO will be created and returned.
- If *self.obj* is a seekable *file* object, it will be seeked to the beginning and returned.
- If *self.obj* is an unseekable *file* object, it will be returned directly.

__exit__(*exc_type*, *exc_value*, *traceback*)

Exit context.

- If *self.obj* is str, the StringIO (*self.sio*) will be closed.
- If *self.obj* is a seekable *file* object, its stream position (*self.pos*) will be recovered.

__init__(*obj*)

Initialize context.

Parameters **obj** (*Union[str, TextIO]*) – the object to manage in the context

bpc_utils.**_mp_map_wrapper**(*args*)

Map wrapper function for multiprocessing.

Parameters **args** (*Tuple[Callable, Iterable[Any], Mapping[str, Any]]*) – the function to execute, the positional arguments and the keyword arguments packed into a tuple

Returns the function execution result

Return type Any

bpc_utils.**_mp_init_lock**(*lock*)
Initialize lock for multiprocessing.

Parameters **lock** (*multiprocessing.synchronize.Lock*) – the lock to be shared among tasks

bpc_utils.**expand_glob_iter**(*pathname*)
Wrapper function to perform glob expansion.

Parameters **pathname** (*str*) – pathname pattern

Returns an iterator which yields the paths matching a pathname pattern

Return type Iterator[str]

bpc_utils.**_boolean_state_lookup = {'0': False, '1': True, 'false': False, 'n': False,**
A mapping from string representation to boolean states. The values are used for *parse_boolean_state()*.

Type Dict[str, bool]

bpc_utils.**_linesep_lookup = {'\n': '\n', '\r': '\r', '\r\n': '\r\n', 'cr': '\r', 'crlf**
A mapping from string representation to linesep. The values are used for *parse_linesep()*.

Type Dict[str, str]

bpc_utils.**CPU_CNT: int**
Number of CPUs for multiprocessing support.

bpc_utils.**mp: Optional[ModuleType] = <module 'multiprocessing'>**
An alias of the Python builtin multiprocessing module if available.

bpc_utils.**parallel_available: bool**
Whether parallel execution is available.

bpc_utils.**task_lock: Union[contextlib.nullcontext, multiprocessing.synchronize.Lock]**
A lock for possibly concurrent tasks.

CHAPTER

THREE

INDICES AND TABLES

- genindex
- modindex
- search

PYTHON MODULE INDEX

b

INDEX

Symbols

A

B

C

D

E

F

G

H

M

O

P

R

S

T

U