MordinezNLP

Release 0.1.0a1

Marcin Borzymowski

Mar 19, 2022

CONTENTS:

CHAPTER

ONE

DOWNLOADERS - BASIC

class MordinezNLP.downloaders.Basic.**BasicDownloader**

Class helps to download multiple files from list of provided links using multithreading.

static download_to_bytes_io(*url: str*, *temp_path: str*, *use_memory: bool*, *custom_headers: dict = {}*, *streamable: bool = False*, *sleep_time: float = 0*, *max_retries: int = 10*) → Union[_io.BytesIO, pathlib.Path]

Function defines how to download single URL. It is used by *download_urls* function to use Threading for multithread downloading.

Function makes GET request to a specified URL. If there is an exception, the function will try to download file untill it will be successful or until it reaches 10 unsucessful downloads

Parameters

- **max_retries** (*int*) – How many retries function will make until it marks a file undownloadable
- **sleep_time** (*int*) – A sleep time in seconds that is used to prevent sites from detecting downloading as a DDoS attack
- **streamable** (*bool*) – Sets a *request*'s *stream* parameter. More info https://2.python-requests.org/en/v2.8.1/user/advanced/#body-content-workflow
- **custom_headers** (*bool*) – Custom headers used in each request
- **url** (*str*) – valid HTTP/HTTPS URL
- **temp_path** (*str*) – A temporary path where downloaded files will be saved (argument used only during in memory download)
- **use_memory** (*bool*) – When set to *True* script will download all data to the memory. Otherwise it will save
- **data as temporary files on a disk.** (*downloaded*) –

Returns downloaded file as a bytes

Return type io.BytesIO

static download_urls(*urls: List[str]*, *file_type_handler: Callable*, *threads: int = 8*, *sleep_time: float = 0*, *custom_headers: Iterable = repeat({})*, *streamable: Iterable = repeat(False)*, *max_retries: int = 10*, *use_memory: bool = True*, *temp_dir: Optional[pathlib.Path] = None*) → list

Function allows user to download files from provided URLs in list. Each file is downloaded as BytesIO using specified number of threads and then *file_type_handler* is used to convert file from BytesIO to specified format. Each file type should have its own *file_type_handler*.

Sleep_time is used to prevent sites from detecting DDoS attacks. Before downloading file specified thread is going to sleep for specified amount of time.

Function for each thread (and for each single URL) uses *download_to_bytes_io* function.

Using *use_memory* user can decide if files will be downloaded to the memory or saved as a temporary files on a disk.

Parameters

- **urls** (*List[str]*) – List of URLs of files to download
- **file_type_handler** (*Callable*) – Function used to convert downloaded file to a specified format
- **threads** (*int*) – Number of threads to download files
- **sleep_time** (*int*) – Time used to prevent file downloads from being detected as DDoS attack
- **max_retries** (*int*) – Refer to *download_to_bytes_io* function documentation.
- **streamable** (*bool*) – Refer to *download_to_bytes_io* function documentation.
- **custom_headers** (*int*) – Refer to *download_to_bytes_io* function documentation.
- **use_memory** (*bool*) – Downloader can download files to memory or to the binary files. Use memory downloader
- **You know that You will download small amout of data, otherwise set this value to False.** (*when*) –
- **temp_dir** (*Path*) – Path to directory where temporary files will be saved.

Returns A list of downloaded and processed by *file_type_handler* function files.

Return type list

Example usage for TXT files:

```
from MordinezNLP.downloaders import BasicDownloader
from MordinezNLP.downloaders.Processors import text_data_processor

downloaded_elements = BasicDownloader.download_urls(
    [
        "https://raw.githubusercontent.com/BMarcin/MordinezNLP/main/requirements.txt",
        "https://raw.githubusercontent.com/BMarcin/MordinezNLP/main/LICENSE"
    ],
    lambda x: text_data_processor(x),
)

print(downloaded_elements) # <- will display a list with elements where each is a␣
↪content of a downloaded files
```

Example usage for PDF files:

```
from MordinezNLP.downloaders import BasicDownloader
from MordinezNLP.downloaders.Processors import pdf_data_processor

downloaded_pdfs = BasicDownloader.download_urls(
    [
        "https://docs.whirlpool.eu/_doc/19514904100_PL.pdf",
        "https://mpm.pl/docs/_instrukcje/WA-6040S_instrukcja.pdf",
    ],
    lambda x: pdf_data_processor(x)
)
```

(continues on next page)

(continued from previous page)

```
print(downloaded_pdfs) # <- will display a list with elements where each is a content␣
↪of a downloaded files
```

CHAPTER

TWO

DOWNLOADERS - COMMONCRAWL

class MordinezNLP.downloaders.CommonCrawlDownloader.**CommonCrawlDownloader**(*links_to_search: List[str]*, *index_name: str = 'CC-MAIN-2020-24'*, *base_index_url: str = 'http://index.commoncra*
search_for_mime: str = 'text/html', *search_for_language: str = 'eng'*, *threads: int = 8*)

Class used to download common crawl data using Basic multithreaded downloader.

download(*save_to: str*, *base_url: str = 'https://commoncrawl.s3.amazonaws.com'*, *sleep_time: float = 0*)

Main function used to download CC data using multithreaded Base Downloader.

Parameters

- **save_to** (*str*) – path to a folder where the data will be downloaded. Each file is a HTML document downloaded from CC.
- **base_url** (*str*) – base CC URL for example: https://commoncrawl.s3.amazonaws.com
- **sleep_time** (*int*) – A sleep time in seconds that is used to prevent sites from detecting downloading as a DDoS attack

Example usage:

```
from MordinezNLP.downloaders import CommonCrawlDownloader

ccd = CommonCrawlDownloader(
    [
        "reddit.com/r/space/*",
        "reddit.com/r/spacex/*",
    ]
)
ccd.download('./test_data')
```

CHAPTER

THREE

DOWNLOADERS - ELASTIC SEARCH

class MordinezNLP.downloaders.ElasticSearchDownloader.**ElasticSearchDownloader**(*ip: str*, *port: int*, *api_key_1: Optional[str] = None*, *api_key_2: Optional[str] = None*, *timeout: int = 100*)

Class used to download elastic search data from specified index using multithreading todo make tests

get_all_available_indexes() → List[str]

Get all available elastic search indexes.

Returns a list of string where each element is a elastic search index name

Return type List[str]

scroll_data(*index_name: str*, *query: dict*, *processing_function: Callable[[dict], Any]*, *threads: int = 6*, *scroll: str = '2m'*, *scroll_size: int = 100*) → List[any]

A function that scrolls through an elastic search data from index and returns a multithreaded data processed with *processing_function*. It returns a List of types returned by a *processing_function*.

Parameters

- **index_name** (*str*) – An index name to scroll/download the data
- **query** (*dict*) – An elastic search query
- **processing_function** (*Callable[[dict], Any]*) – A function that processes single item from elastic search index
- **threads** (*int*) – A number of threads to run processing on
- **scroll** (*str*) – A scroll value
- **scroll_size** (*int*) – A size of scrolling items at once

Returns Returns a list of processed items with type according to a *processing_function* or empty list if index doesn't exists.

Return type List[any]

Example usage:

```
from MordinezNLP.downloaders import ElasticSearchDownloader

es = ElasticSearchDownloader(
    ip='',
    port=9200,
    timeout=10
)

body = {}  # <- use your own elastic search query

' Your own processing function for a single element '
def processing_func(data: dict) -> str:
    return data['my_key']['my_next_key'].replace("\r\n", "\n")


' Scroll the data '
downloaded_elastic_search_data = es.scroll_data(
    'my_index_name',
    body,
    processing_func,
    threads=8
)

print(len(downloaded_elastic_search_data))
```

CHAPTER

FOUR

DOWNLOADERS - PROCESSORS

MordinezNLP.downloaders.Processors.**gzip_to_text_data_processor**(*data: _io.BytesIO*) → str

Function can be used together with downloaders to covnert BytesIO to GZIP and unpack it to str.

Parameters **data** (*BytesIO*) – input data which comes from downlaoder class/function

Returns parsed input

Return type str

MordinezNLP.downloaders.Processors.**pdf_data_processor**(*data: _io.BytesIO*) → str

Function can be used together with downloaders to convert BytesIO from PDF files to str.

Parameters **data** (*BytesIO*) – input data which comes from downloader class/function

Returns

parsed input, more informations about parsing PDFs can be found in method MordinezNLP.parsers.process_pdf

Return type str

MordinezNLP.downloaders.Processors.**text_data_processor**(*data: _io.BytesIO*) → str

Function can be used together with downloaders to convert BytesIO from text data to str.

Parameters **data** (*BytesIO*) – input data which comes from downloader class/function

Returns parsed input

Return type str

CHAPTER

FIVE

PARSERS - PDF PARSER

MordinezNLP.parsers.process_pdf.**process_pdf**(*pdf_input: _io.BytesIO*) → List[str]

A function can read strings from PDF docs handled in the BytesIO object. It extracts whole text and removes text that occurs in tables. The reason for that is that tables have mainly messy data for NLP tasks.

Function is divided into two parts. First removes tokens by exact match and the same number of occurences in text and tables. First part uses list of tokens, second uses tokens joined with space.

Parameters **pdf_input** (*BytesIO*) – A PDF as a BytesIO object

Returns Parsed text without texts found in tables

Return type List[str]

Example usage for TXT files:

```
from io import BytesIO
from MordinezNLP.parsers import process_pdf

with open("my_pdf_doc.pdf", "rb") as f:
       pdf = BytesIO(f.read())
   output = process_pdf(pdf)
   print(output)
```

CHAPTER

SIX

PARSERS - HTML PARSER

MordinezNLP.parsers.HTML_Parser.**HTML_Parser**(*html_doc: str*, *separator: str = ' '*) → str

Function which removes not vaulable text and tags from HTML docs. It is based on research https://rushter.com/blog/python-fast-html-parser/

IMPORTANT If You must be 100% sure, that text You want to process is a HTML doc. Otherwise some parts of the source text can be deleted because of misunderstanding text as a tags.

Parameters

- **separator** – Separator used to join HTML nodes in *selectolax* package
- **html_doc** (*str*) – a HTML doc

Returns String which is a vaulable text parsed from HTML doc.

Return type str

Example usage for HTML files:

```
from MordinezNLP.parsers import HTML_Parser

with open("my_html_file.html", "r") as f:
    html_content = HTML_Parser(f.read())
    print(html_content)
```

CHAPTER

SEVEN

PROCESSORS - BASIC

class MordinezNLP.processors.Basic.**BasicProcessor**(*language: str = 'en'*)

The aim of the class is to make use of NLP-dirty texts

get_special_tokens() → List[str]

Function can return all of the special tokens used by *process* function. It can be needed when training SentencePiece tokenizer.

Returns all of the special tokens used in *process* function

Return type List[str]

static load_language_days(*language: str*) → List[str]

Return language specific names of days of the week :param language: a language in which return name of days of the week :type language: str

Returns a list of day names in specified language

Return type List[str]

static load_language_months(*language: str*) → List[str]

Function returns language specific names of months

Parameters **language** (*str*) – language in which return names

Returns a list of months in specified language

Return type List[str]

static load_numerals(*language: str*) → List[str]

Build language specific numerals. Currently supported numerals are from 1 to 99.

Parameters **language** (*str*) – a language in which function will return numerals

Returns a list of numerals in specified language

Return type List[str]

static load_ordinals(*language: str*) → List[str]

Build a language specific ordinals. Currently supported ordinals from 1 to 99

Parameters **language** (*str*) – a language in which function will return ordinals

Returns a list of ordinals in specified language

Return type List[str]

pos_tag_data(*post_processed_data: List[str]*, *replace_with_number: str*, *tokenizer_threads: int*, *tokenizer_batch_size: int*, *pos_batch_size: int*) → List[str]

A helper function to postprocess numbers tags and replace according tokens with special token. It also uses SpaCy tokenization to return "normal" form of tokens.

Long story short: This function will parse input "There wasn't six apples" to "There was not <number> apples".

Parameters

- **post_processed_data** (*List(str)*) – a postprocessed texts list
- **replace_with_number** (*str*) – a special token to replace numbers with
- **tokenizer_threads** (*int*) – How many threads to use for tokenization
- **tokenizer_batch_size** (*int*) – Batch size for tokenization
- **pos_batch_size** (*int*) – POS tagging batch size, be careful when CUDA is availabe in Your system!

Returns postprocessed texts

Return type str

process(*text_to_process: Union[str, List[str]]*, *pre_rules: List[Callable] = []*, *post_rules: List[Callable] = []*, *language: str = 'en'*, *fix_unicode: bool = True*, *lower: bool = False*, *no_line_breaks: bool = False*, *no_urls: bool = True*, *no_emails: bool = True*, *no_phone_numbers: bool = True*, *no_numbers: bool = True*, *no_digits: bool = False*, *no_currency_symbols: bool = True*, *no_punct: bool = False*, *no_math: bool = True*, *no_dates: bool = True*, *no_multiple_chars: bool = True*, *no_lists: bool = True*, *no_brackets: bool = True*, *replace_with_url: str = '<url>'*, *replace_with_email: str = '<email>'*, *replace_with_phone_number: str = '<phone>'*, *replace_with_number: str = '<number>'*, *replace_with_digit: str = '0'*, *replace_with_currency_symbol: str = '<currency>'*, *replace_with_date: str = '<date>'*, *replace_with_bracket: str = '<bracket>'*, *replace_more: str = '<more>'*, *replace_less: str = '<less>'*, *use_pos_tagging: bool = True*, *list_processing_threads: int = 8*, *tokenizer_threads: int = 8*, *tokenizer_batch_size: int = 60*, *pos_batch_size: int = 7000*) → Union[str, List[str]]

Main text processing function. It mainly uses regexes to find specified patterns in texts and replace them by a defined custom token or fixes parts that are not valuable for humans and machines.

Function also enables users to set *pre_rules* and *post_rules*. You can use those lists of Callables to add pre and post processing rules. A good use case is processing CommonCrawl reddit data, where each page has the same schema (headers, navigation bars etc.). In such case You can use *pre_rules* to filter them and then pass such text into the *process* function pipeline. Also feel free to add *post_rules* to match other cases which are not used here.

Depending on parameters function can replace a specified type of data.

Currently supported entities:

- dates,
- brackets,
- simple math strings,
- phone numbers,
- emails,
- urls,
- numbers and digits
- multiple characters in single words

Dates

Examples of dates matching in strings for english:

- 1.02.2030
- 1st of December 3990
- first of DecEmber 1233
- first december 2020
- early 20s
- 01.03.4223
- 11-33-3222
- 2020s
- Friday 23 October
- late 90s
- in 20s

Brackets

Examples of brackets matching in strings for english:

- [tryrty]
- (other text)

Simple math strings

Examples of simple math strings for english:

- 2 > 3
- 4<6
- 4>=4
- 5<= 4

If You decide to use *no_math=False* than such cases will be processed with other functions. It meas that one function will remove math operator (<,>,<=,>=) and another will replace numbers with special token.

Multiple characters in single words

Table below show string *before* and *after* using a *multiple characters in single word* processing function

Before	After
'EEEEEEEEEEE'	''
'supeeeeeer'	'super'
'EEEE<number>!'	''
'supppppprrrrpper'	'suprpper'

Processing multiple characters is extremely useful in processing CommonCrawl reddit data.

Lists replacement

Lists in a text with leading "-" or ">" for each item can be parsed to simple and more understandable text. For example list:

```
My_list:
- item 1
- item 2,
-item 3
```

Will be parsed to:

```
My_list: item 1, item 2, item 3.
```

Use *no_lists* argument to enable this feature.

Supported languages

Fully supported languages	Partially supported languages
English	German

Be careful

Please don't replace special tokens in function, because it can wrongly process strings. It will be fixed in feature releases # todo fix regexes in __init__ for special tokens

Parameters

- **text_to_process** (*Union[str, List[str]]*) – An input text or a list of texts (for multiprocess processing) to process by a function
- **pre_rules** (*List[Callable]*) – A list of lambdas that are applied before main pre-processing rules
- **post_rules** (*List[Callable]*) – A list of lambdas that are applied after *pre_rules* and function processing rules
- **language** (*str*) – A input text language
- **fix_unicode** (*bool*) – replace all non unicode characters to unicode
- **lower** (*bool*) – lowercase all characters
- **no_line_breaks** (*bool*) – fully strip line breaks as opposed to only normalizing them
- **no_urls** (*bool*) – replace all URLs with a special token
- **no_emails** (*bool*) – replace all email addresses with a special token
- **no_phone_numbers** (*bool*) – replace all phone numbers with a special token
- **no_numbers** (*bool*) – replace all numbers with a special token
- **no_digits** (*bool*) – replace all digits with a special token
- **no_currency_symbols** (*bool*) – replace all currency symbols with a special token
- **no_punct** (*bool*) – remove punctuations
- **no_math** (*bool*) – remove >= <= in math strings
- **no_dates** (*bool*) – remove dates strings in input text 'early 80s' -> '<date>'
- **no_lists** (*bool*) – replace all texts lists
- **no_brackets** (*bool*) – replace brackets: '[', ']', '(', ')'
- **no_multiple_chars** (*bool*) – reduce multiple characters in string into a single ones 'supeeeeeer' -> 'super'

- **replace_with_url** (*str*) – a special token used to replace urls
- **replace_with_email** (*str*) – a special token used to replace emails
- **replace_with_phone_number** (*str*) – a special token used to replace phone numbers
- **replace_with_number** (*str*) – a special token used to replace numbers
- **replace_with_digit** (*str*) – a special token used to replace digits
- **replace_with_currency_symbol** (*str*) – a special token used to replace currency symbol
- **replace_with_date** (*str*) – a special token used to replace dates
- **replace_with_bracket** (*str*) – a special token used to replace brackets
- **replace_more** (*str*) – a special token used to replace more '>' and more or equal '>=' symbols in math texts
- **replace_less** (*str*) – a special token used to replace less '<' and less or equal '<=' symbols in math texts
- **use_pos_tagging** (*bool*) – if True function will use StanzaNLP & SpaCy for POS tagging and token normalization
- **list_processing_threads** (*int*) – How many threads You want to use to process List(str) which is on a input for
- **function.** (*this*) –
- **tokenizer_threads** (*int*) – How many threads to use during tokenization, this value is passed to the SpaCy pipeline.
- **tokenizer_batch_size** (*int*) –
- **pos_batch_size** (*int*) –

Returns Post-processed text

Return type Union[str, List[str]]

process_multiple_characters(*text_to_process: str*) → str

Function can detect multiplied characters in a word and replace them by a single one.

Before	After
'EEEEEEEEEEEE!'	''
'supeeeeeer'	'super'
'EEEE<number>!'	''
'supppppprrrrrpper'	'suprpper'

Parameters **text_to_process** (*str*) – An input text to process

Returns Text with removed duplicated characters in each word

Return type str

Example usage:

```
from MordinezNLP.processors import BasicProcessor

bp = BasicProcessor()
post_process = bp.process("this is my text to process by a funcion", language='en')
print(post_process)
```

CHAPTER

EIGHT

TOKENIZERS - SPACYTOKENIZER

A custom SpaCy tokenizer ready for tokenizing special tokens which comes from *BasicProcessor*.

Out of the box SpaCy tokenizer will parse special tokens (tags) separately for example: "<date>" to "< date >", so that function changes such behavior.

param nlp A Language object from SpaCy

type nlp spacy.language.Language

returns A SpaCy tokenizer

rtype spacy.tokenizer.Tokenizer

Example usage:

```
from MordinezNLP.tokenizers import spacy_tokenizer
import spacy

nlp: Language = spacy.load("en_core_web_sm")
nlp.tokenizer = spacy_tokenizer(nlp)

test_doc = nlp('Hello today is <date>, tomorrow it will be <number> degrees of␣
↪celcius.')

for token in test_doc:
    print(token)

# output
# Hello
# today
# is
# <date>
# ,
# tomorrow
# it
# will
# be
# <number>
# degrees
# of
# celcius
# .
```

CHAPTER

NINE

PIPELINES - PARTOFSPEECH

class MordinezNLP.pipelines.PartOfSpeech.**PartOfSpeech**(*nlp: spacy.language.Language*, *language: str = 'en'*)

The aim of the class is to tag each token (which comes from MordinezNLP processors) with its POS tag.

process(*texts: List[str]*, *tokenizer_threads: int = 8*, *tokenizer_batch_size: int = 50*, *pos_batch_size: int = 3000*, *pos_replacement_list: Optional[Dict[str, str]] = None*, *token_replacement_list: Optional[Dict[str, str]] = None*, *return_docs: bool = False*, *return_string_tokens: bool = False*) → Union[Generator[Tuple[List[Union[spacy.tokens.token.Token, str]], List[str]], None, None], Generator[Tuple[List[List[Union[spacy.tokens.token.Token, str]]], List[List[str]]], None, None]]

Main processing function. First step is to tokenize a list of input texts to sentences and then to the tokens. Then such input goes to the StanzaNLP.

For the function List[str] object which comes as an input is a list of docs to process. Each item in a list is a document (SpaCy logic in pipelines). In such case You can specify if You want to return texts in structure documents[sentences[tokens]] or sentences[tokens] (removing the documents layer).

Sometimes You want to force POS tagger to assign POS tag to a specified token or instead of other POS tag. For such cases You can use *pos_replacement_list* and *token_replacement_list*. You can import sample token and POS replacement lists from MordinezNLP.utils.pos_replacement_list and MordinezNLP.utils.token_replacement_list.

If You want to use a special attributes for each tokens from SpaCy please pass *False* as a value of *return_string_tokens* argument.

Each token parsed by SpaCy tokenizer will by converted to its normal version. For example each *n't* will be replaced by *not*.

Parameters

- **texts** (*List[str]*) – an input texts, each item in a list is a document (SpaCy logic in pipelines)
- **tokenizer_threads** (*int*) – How many threads You want to use in SpaCy tokenization
- **tokenizer_batch_size** (*int*) – Batch size for SpaCy tokenizer
- **pos_batch_size** (*int*) = Batch size for Stanza POS tagger (if enabled) –
- **pos_replacement_list** (*Union[Dict[str, str], None]*) – If not None function will replace each POS tag
- **value set in value field of the dict. Each key is a POS tag to be replaced by its value.** (*with*) –

- **token_replacement_list** – If not None function will replace each POS tag with the value set in value field of
- **dict. Each key is token, which will be replaced by its value.** (*the*) –
- **return_docs** (*bool*) – If True function will keep a "documents" layer on output.
- **return_string_tokens** (*bool*) – Function can return tokens as SpaCy Token object (if You need to access token
- **data such as norm) or can return tokens as a string object. If True returns a string tokens.** (*specified*) –

Returns Union[Generator[Tuple[List[Union[Token, str]], List[str]], None, None], Generator[Tuple[List[List[Union[Token, str]]], List[List[str]]], None, None]]: a list of doc(if return docs is set) with list of sentences with list of tokens and its pos tags.

Example usage:

```
from MordinezNLP.pipelines import PartOfSpeech
from MordinezNLP.tokenizers import spacy_tokenizer
import spacy

nlp: Language = spacy.load("en_core_web_sm")
nlp.tokenizer = spacy_tokenizer(nlp)

docs_to_tag = [
    'Hello today is <date>, tomorrow it will be <number> degrees of celcius.'
]

pos_tagger = PartOfSpeech(
    nlp,
    'en'
)

pos_output = pos_tagger.process(
    docs_to_tag,
    4,
    30,
    return_docs=True
)
```

CHAPTER

TEN

UTILS

MordinezNLP.utils.ngram_iterator.**ngram_iterator**(*string: str*, *ngram_len: int = 3*) → list

Returns an iterator that yeilds the given string and its ngrams. Each subsequent list element has got lenght set to *ngram_len*. The differences between each of following elements in list is a one letter forward in a context. For example for string "hello" and ngram_len set to 3 it will output ["hel", "ell", "llo"]

Parameters

- **string** (*str*) – string to iterate on
- **ngram_len** (*int*) – lenght of each ngram

Returns ngram - list of ngram_len characters of input string

Return type list

Example usage:

```
from MordinezNLP.utils import ngram_iterator

print(list(ngram_iterator("<hello>", 3))) # <- will print ['<he', 'hel', 'ell', 'llo',
↪ 'lo>']
```

MordinezNLP.utils.random_string.**random_string**(*length: int = 64*, *choices_list: List[str] = 'ABCDEFGHIJKLMNOPQRSTUVWXYZ0123456789'*) → str

Generate random string which contains characters from *choices_list* arg.

Parameters

- **length** (*int*) – length of generated string
- **choices_list** (*List[str]*) – List of characters from which random string should be generated

Returns Randomly generated string

Return type str

Example usage:

```
from MordinezNLP.utils import random_string
import string

rs = random_string(32)
print(rs)

rs = random_string(10, string.digits)
print(rs)
```

CHAPTER

ELEVEN

INDICES AND TABLES

- genindex
- modindex
- search

PYTHON MODULE INDEX

m

INDEX

N

P

R

S

T