Inline Markup

`*emphasis* and :emphasis:`txt``	*emphasis* and *txt*	` ``literal``, :literal:`x \ y` `	`literal, x \ y`
`**strong** and :strong:`txt``	**strong** and **txt**	`\*escape\*`	*escape*
`H\ :sub:`2`\ O`	H_2O	`E = mc\ :sup:`2``	$E = mc^2$
`Latex $$: :math:`(\pi/4) d^2``	Latex $$: $(\pi/4) d^2$	`See :RFC:`2822``	See *RFC 2822*

Lists

```
- This is item 1. A blank line before the first
  and last items is required.

  - This is nested list

* Item 3: blank lines between items are optional.
* Item 4: Bullets are "-", "*" or "+".
  Continuing text must be aligned after the
  bullet and whitespace.
```

- This is item 1. A blank line before the first and last items is required.
  - This is nested list
- Item 3: blank lines between items are optional.
- Item 4: Bullets are "-", "*" or "+". Continuing text must be aligned after the bullet and whitespace.

```
3. Enumerators are arabic numbers, single
   letters, or roman numerals
4. List items should be sequentially numbered,
   but need not start at 1 (although not all
   formatters will honour the first index).
#. This item is auto-enumerated
```

3. Enumerators are arabic numbers, single letters, or roman numerals
4. List items should be sequentially numbered, but need not start at 1 (although not all formatters will honour the first index).
5. This item is auto-enumerated

```
Term definition
  The term is a one-line phrase, and the
  definition is one or more paragraphs or body
  elements, indented relative to the term. Blank
  lines are not allowed between term and definition.
```

Term definition
    The term is a one-line phrase, and the definition is one or more paragraphs or body elements, indented relative to the term. Blank lines are not allowed between term and definition.

```
-a            command-line option "a"
-b file       options can have arguments and long
              descriptions
--long        options can be long also
--input=file  long options can also have arguments
/V            DOS/VMS-style options too
```

`-a`	command-line option "a"
`-b` *`file`*	options can have arguments and long descriptions
`--long`	options can be long also
`--input=file`	long options can also have arguments
`/V`	DOS/VMS-style options too

Sections structure

`Parts` `#####`	**Parts**	`Chapters` `********`	**Chapters**

`Sections`
`========`
	Sections	`Subsections`
		`-----------`
			Subsections

`Subsubsection` `^^^^^^^^^^^^^`	**Subsubsection**	`Paragraphs` `""""""""""`	**Paragraphs**

Internal and external links

```
Footnote [1]_. Use * for symbols. Reference [CIT]

.. [1] You can use # (autonumbered) as well.
.. [CIT] Any combination of letters and numbers.
```

Footnote example [1]. Use * for symbols. Reference [CIT]

| [1]   You can use # (autonumbered) as well.
| [CIT]   Any combination of letters and numbers.

```
Titles are hyperlinks
=====================

`Titles are hyperlinks`
```

Titles are hyperlinks

Titles are hyperlinks

```
External hyperlinks, like `Python
<http://www.python.org/>`_.
```

External hyperlinks, like Python.

```
External hyperlinks, like Python_.

.. _Python: http://www.python.org/
```

External hyperlinks, like Python.

```
General URI, such as http://www.python.org or
mailto:me@somewhere.com is acceptable. Standalone
email addresses (me@test.com) as well.
```

General URI, such as *http://www.python.org* or *mailto:me@somewhere.com* is acceptable as well. Standalone email address (*me@test.com*) as well.

```
Internal crossreferences, like example_.

.. _example: This is an example crossref. target.
```

Internal crossreferences, like *example*
This is an example crossreference target.

```
This is an example of an `annonymous hyperlink
reference`__. The order of the hyperlinks must match
the order of `the targets`__.

.. __: http://www.python.org/
__ http://www.python.org/
```

This is an example of an annonymous hyperlink reference. The order of the hyperlinks must match the order of the targets.

Official reference: https://docs.python.org/devguide/documenting.html   **Playground:** https://www.tele3.cz/jbar/rest/rest.html

Created by: Radek Lát (radek.lat@gmail.com)   **https://lat.sk**   **Revised:** 2016-08-09 15:23, v1.5

Substitutions (general syntax is `.. |<sub>| <directive>:: <data>`)

`The **|BH|** symbol must be used on containers. **|RST|** replacement example, **|RST|_** documentation.`
 	
`.. **|BH|** image:: biohazard.png`[1]
`.. **|RST|** replace:: reStructuredText`
`.. **_RST**: http://docutils.sourceforge.net/rst.html`
	The ☣ symbol must be used on containers. reStructuredText replacement example, *reStructuredText* documentation.

`.. include myfile.rst`	Will 'inline' the given file. A common convention is to create a global .rst file called *global.rst* and include that at the top of every page. Very useful for links to common images or common files links, etc.

Cross-referencing syntax (general syntax is `:domain:role:`[!~][<title>]<target>``)

`Prefix ! will not generate any hyperlink:` **`:py:meth:`!Queue.Queue.get``** `Prefix ~ will generate only the last part:` **`:py:meth:`~Queue.Queue.get``** `Custom title:` **`:meth:`getFunc <Queue.Queue.get>``**`,` `py is the default domain.`	Prefix ! will not generate any hyperlink: Queue.Queue.get() Prefix ~ will generate only the last part: *get()* Custom title: *getFunc*, py is the default domain.

Python domain references

`:paramref:`	Reference a method parameter.
`:mod:`	Reference a module; a dotted name may be used. This should also be used for package names.
`:func:`	Reference a Python function; dotted names may be used. Trailing parentheses to enhance readability are added automatically.
`:data:`	Reference a module-level variable. Works also as **`:data:`True``**, **`:data:`False``**, and **`:data:`None``**.
`:const:`	Reference a "defined" constant. This may be a Python variable that is not intended to be changed.
`:class:`	Reference a class; a dotted name may be used. Works also as **`:class:`int``, `:class:`bool``, and `:class:`float``.**
`:meth:`	Reference a method of an object. The role text can include the type name and the method name; if it occurs within the description of a type, the type name can be omitted. A dotted name may be used.
`:attr:`	Reference a data attribute of an object.
`:exc:`	Reference an exception. A dotted name may be used.

Info fields lists (separated by a blank line from other text)

`:param` `[type] name:`	Description of a parameter *name*, with optional *type*. Alternatives are **`parameter`**, **`arg`**, **`argument`**, **`key`**, and **`keyword`**.
`:type` `name: type`	Type of a parameter *name*.
`:raises` `ExClass:`	That (and when) a specific exception is raised. Alternatives are **`raise`**, **`except`**, and **`exception`**.
`:var` `name:`	Description of a variable (useful for class and instance variables). Alternatives are **`ivar`** and **`cvar`**.
`:vartype` `name:`	Type of a variable name.
`:returns``:`	Description of the return value. Alternative is **`return`**.
`:rtype``: type`	Return type.

Paragraph-level markup (general syntax is `.. <type>:: [<param>]\n<text>`)

`..` **`note`**`::`	An especially important bit of information about an API that a user should be aware of when using whatever bit of this API. New line is optional.
`..` **`warning`**`::`	The same as *note*, but it is recommended over *note* for information regarding security.
`..` **`todo`**`::`	The same as *note*, but for to-do notes.
`versionadded` *`version`*	Documents the version of the project which added the described feature to the library or C API. When this applies to an entire module, it should be placed at the top of the module section before any prose. The first argument must be given and is the version in question; you can add a second argument consisting of a *brief* explanation of the change.
`versionchanged` *`version`*	Similar to above, but describes when and what changed in some way (new parameters, changed side effects, etc.).
`..` **`deprecated`**`::` *`version`*	Similar to above, but describes when the feature was deprecated. An explanation can also be given, for example to inform the reader what should be used instead.
`..` **`seealso`**`::`	Many sections include a list of references to module documentation or external documents.
`..` **`rubric`**`::` *`title`*	This directive creates a paragraph heading that is not used to create a table of contents node.

`..` **`hlist`**`::`
`   :columns: 3`
 	
`   * First item`
	This directive must contain a bullet list. It will transform it into a more compact list by either distributing more than one item horizontally, or reducing spacing between items, depending on the builder. For builders that support the horizontal distribution, there is a `:columns:` option that specifies the number of columns; it defaults to 2.

`Source code example::`
 	
`  separated by a blank`
`  line and indented`
	Source code example:
	
	`  separated by a blank`
	`  line and indented`

Documentation of module data members and class attributes

```
#: One or more lines like this before the data member or the class attribute
class_attribute = 1  #: single line next to the data member or the class attribute
""" One or more lines after the data member or the class attribute. """
```

1 http://docutils.sourceforge.net/docs/ref/rst/directives.html#image

PyCharm - Type Hinting[2]

Type of parameter	`def f(param: int):`
Return type	`def f() -> int:`
Type of local variables and attributes	see code below

```
class C:
    foo = None  # type: List[str]

    def __init__(self, bar):
        self.bar = bar  # type: Optional[str]

    def f2():
        return 'foo'

    def f1():
        x = f2()  # type: str
        return x.upper()
```

PyCharm - Legacy type syntax for doctrings (use in info fields lists `:param` or `:type`)

`Foo`	Class `Foo` visible in the current scope	**`T`**	Generic type (T-Z are reserved for generics)
`x.y.Bar`	Class `Bar` from x.y module	**`T <= Foo`**	Generic type with upper bound `Foo`
`Foo | Bar`	Classes `Foo` or `Bar`	**`Foo[T]`**	`Foo` parametrized with `T`
`(Foo, Bar)`	Tuple of `Foo` and `Bar`	**`(Foo, Bar) -> Baz`**	Function of `Foo` and `Bar` that returns Baz
`list[Foo]`	`List` of `Foo` elements	**`list[dict[str, int]]`**	`List` of `dicts` from `str` to `int` (nested arguments)
`dict[Foo, Bar]`	Dict from `Foo` to `Bar`		

2 https://www.jetbrains.com/pycharm/help/type-hinting-in-pycharm.html