

Sidebar

Programming · VS Code Pylance Cheat Sheet

Pylance, the Python language server, in depth: type checking, search paths, rule severities, auto-import, hints and commands. Checked against Pylance 2026.4.1 installed here; the language sheet covers the most common Python settings and this one goes through the rest. Sheet 20 in the VS Code series.

SETUP & COMMANDS

- Extension IDms-python.vscode-pylance (version 2026.4.1 here); it works with the Python extension
- python.languageServerDefault (uses Pylance when installed), Pylance, Jedi, None
- Python: Select InterpreterCommand Palette: Pylance reads packages from the interpreter you pick
- Pylance: Go to output channelOpen the log: first place to look for import and indexing problems
- Pylance: Start Logging / Stop LoggingCollect detailed logs for a bug report
- Pylance: Clear All Persisted IndicesDelete the saved library indices if completions go stale
- Pylance: Fold All Docstrings / UnfoldCollapse or expand every docstring in the file
- Install Matching Type StubsQuick fix: install a types-... package that supplies missing type information
- Generate Dependency FileQuick fix: create a requirements.txt -style file
- Pylance: Report Issue...Start a bug report with logs attached

TYPE CHECKING

- python.analysis.typeC
- offMostly syntax errors
- basic / standardAdd nar
- strictAdds missing-an
- python.analysis.diagn
- python.analysis.enabl
- python.analysis.disab
- python.analysis.displ

TYPE EVALUATION (PYTHON.ANALYSIS.TYPEEVALUATION.*)

- python.analysis.typeEvaluation.analyzeUnannotatedFunctionsCheck functions that have no annotations (default true)
- python.analysis.typeEvaluation.enableTypeIgnoreCommentsHonour # type: ignore comments (default true)
- python.analysis.typeEvaluation.strictListInferenceInfer precise list element types, e.g. list[int | str | float] (default false)
- python.analysis.typeEvaluation.strictDictionaryInferenceThe same for dictionaries (default false)
- python.analysis.typeEvaluation.strictSetInferenceThe same for sets (default false)
- python.analysis.typeEvaluation.strictParameterNoneValueA parameter defaulting to None is not implicitly optional (default true)
- python.analysis.typeEvaluation.deprecateTypingAliasesFlag typing.List -style aliases (default false)
- python.analysis.typeEvaluation.enableReachabilityAnalysisMark code that can never run with a hint (default false)
- python.analysis.typeEvaluation.disableBytesTypePromotionsbytearray and memoryview are not treated as bytes (default true)
- python.analysis.typeEvaluation.enableExperimentalFeaturesTurn on experimental type features (default false)

SCOPE & SEARCH PATH

- python.analysis.inclu
- python.analysis.useDe
- python.analysis.ignor
- python.analysis.extra
- python.analysis.autoS
- python.analysis.stubP
- python.analysis.enabl
- python.analysis.useli
- python.analysis.exclu
- python.analysis.ignor
- python.analysis.extra

RULE SEVERITIES

`python.analysis.diagnosticSeverityOverrides` An object of rule: level, e.g. `{ "reportMissingImports": "warning" }`

Levels `none`, `information`, `warning`, `error` (also `true` / `false`)

`reportMissingImports` Import cannot be resolved (default `warning` in the schema)

`reportMissingModuleSource` Stub found but the real package is not installed (default `warning`)

`reportUndefinedVariable` Name is not defined (default `warning`)

`reportAttributeAccessIssue` Attribute does not exist on that type (default `error`)

`reportPossiblyUnboundVariable` Variable might not be assigned yet (default `error`)

`reportArgumentType` Wrong argument type in a call (default `error`)

`reportUnusedImport` / `reportUnusedVariable` Unused imports and variables (default `none`)

`reportOptionalMemberAccess` Attribute use on a value that may be `None` (default `none`)

`reportPrivateUsage` Use of `_private` names from outside (default `none`)

The `mode` matters These are the schema's defaults; the `typeCheckingMode` you choose changes many of them

AUTO-IMPORT & INDEX

`python.analysis.autoI`

`python.analysis.showO`

`python.analysis.impor`

`python.analysis.compl`

`python.analysis.extra`

`python.analysis.index`

`python.analysis.userF`

`python.analysis.persi`

`python.analysis.userF`

`python.analysis.inclu`

`python.analysis.enabl`

TYPING HELP & DOCSTRINGS

`python.analysis.autoIndent` Adjust indentation as you type Python (default `true`)

`python.analysis.autoFormatStrings` Typing `{` in a string adds the `f` prefix (default `false`)

`python.analysis.autoSplitStrings` Splitting a string with Enter adds quotes and a continuation (default `true`)

`python.analysis.formatOnTypeStyle` `pep8` (default) or `black`; indentation inside wrapped brackets

`python.analysis.supportRestructuredText` Render reStructuredText in docstrings (default `true`)

`python.analysis.supportDocstringTemplate` Docstring templates after typing `"""` (default `false`)

`python.analysis.enableColorPicker` Colour picker for `#RRGGBB` strings (default `true`)

`python.analysis.gotoDefinitionInStringLiteral` Go to definition on strings that look like module names (default `true`)

`python.analysis.renameInDocstrings` Rename symbols inside docstrings too (default `false`)

`python.analysis.generateWithTypeAnnotation` Add annotations to generated code (default `false` in `off` mode)

`python.analysis.fixAll` Source fixes to run in *fix all* (default `[]`)

`[python] defaults shipped` Pylance sets `editor.formatOnType` to `true` and `editor.wordBasedSuggestions` to `off` for Python

INLAY HINTS, CODE LE

`python.analysis.inlay`

`python.analysis.inlay`

`python.analysis.inlay`

`python.analysis.inlay`

`python.analysis.refer`

`python.analysis.goto0`

`python.analysis.inclu`

`python.analysis.inclu`

FRAMEWORKS & OPTIONAL EXTRAS

<code>python.analysis.enablePytestSupport</code>	Understand pytest fixtures (default <code>true</code>)
<code>python.analysis.enableDjangoSupport</code>	Understand Django ORM-generated members (default <code>false</code>)
<code>python.analysis.supportHtmlEmbeddedPython</code>	Language support for Python embedded in HTML (default <code>false</code>)
<code>python.analysis.supportAllPythonDocuments</code>	Also work in non-file Python documents such as the terminal (default <code>false</code>)
<code>python.analysis.disableBundledStubs</code>	Analyse installed packages directly instead of Pylance's stubs (default <code>false</code>)
<code>python.analysis.cellMagicLanguages</code>	Map IPython <code>%%</code> magics to languages in notebooks (default <code>{}</code>)
<code>python.analysis.aiHoverSummaries</code> / <code>python.analysis.aiCodeActions</code> / <code>python.analysis.autoTranslateDocstrings</code>	Optional Copilot-powered features (all need Copilot; mostly off by default)
<code>python.analysis.useNearestConfiguration</code>	Experimental: use the nearest <code>pyrightconfig.json</code> / <code>pyproject.toml</code> in subfolders (default <code>false</code>)

PYRIGHT CONFIG FILE

```
pyrightconfig.json or  
  
# type: ignore Silenc  
# pyright: ignore[rep  
# pyright: basic / #  
# pyright: reportMiss  
settings.json vs conf
```

A REASONABLE STARTER SET (MY SUGGESTIONS, NOT DEFAULTS)

<code>"python.analysis.typeCheckingMode": "basic"</code>	A useful middle ground
<code>"python.analysis.autoImportCompletions": true</code>	Import names as you complete them
<code>"python.analysis.inlayHints.functionReturnTypes": true</code>	See inferred return types
<code>"python.analysis.diagnosticMode": "openFilesOnly"</code>	Keep it fast (the default)
<code>"python.analysis.exclude": ["**/venv", "**/.venv"]</code>	Make sure virtual environments are left out
<code>"python.analysis.diagnosticSeverityOverrides": { "reportUnusedImport": "information" }</code>	Gentle reminder, not a warning

"IMPORT COULD NOT BE RESOLVED": WORK DOWN THIS LIST

1) Is the right interpreter selected (*Python: Select Interpreter*)? 2) Is the package installed *in that environment* (`pip list`)? 3) For your own modules, is the folder in `python.analysis.extraPaths`, or is the code in a `src` folder that `autoSearchPaths` will find? 4) Check *Pylance: Go to output channel*. Note the difference between `reportMissingImports` (the module cannot be found) and `reportMissingModuleSource` (type information exists but the real package is not installed, so install it).

PYLANCE IS SLOW OR USES LOTS OF CPU

Keep `python.analysis.diagnosticMode` on `openFilesOnly`, make sure virtual environments and big data folders are in `python.analysis.exclude`, lower `python.analysis.userFileIndexingLimit` for huge workspaces, and run *Pylance: Clear All Persisted Indices* if indexing seems stuck. Type checking in strict mode on a large project needs more time too.