I'm not a bot

In the previous article, I have described how poetry can be used to configure Python workspace and to create a new Python package project. Although poetry creates the structure of a package and adds some boilerplate code, in order to develop this package in VSCode we need to do some additional configurations. In this post, I describe how to start developing a new Python package project in VSCode.Table of ContentsPrerequisitesCurrently, I do Python development in (K)ubuntu 18.04 using VSCode as a enhanced text editor. If you use other IDE/text editor or other operating system, you may need to adapt these instructions.Before starting Python development in VSCode, you have to install Python extension. It is developed by Microsoft and provides you rich support of the Python language and tools including code completion and formatting, linting, code navigation, etc. In order to install this extension, open VSCode Quick Open (Ctrl+P), paste the following command, and press enter:ext install ms-python.pythonConfiguring VSCode for Python DevelopmentNow, lets start a new project and configure VSCode step-by-step. As I have mentioned in the previous article, I create a new project using the following poetry command:$ poetry new --src new_packageThis command creates new Python package project called new_package and puts the sources into the src directory. In addition, the command also creates tests directory where some pytest tests are added. After executing this command, you should get the following structure:$ tree new_package/new_package/ pyproject.toml README.rst src new_package __init__.py tests __init__.py test_new_package.pyLets add a new main.py module to our package with the following content:from loguru import loggerimport osdef adder(i1, i2): return (i1 + i2)def main(): print("Cool new_package project!") logger.info("We are here: {}".format(os.path.abspath(__file__))) logger.info("Adding integers {0} and {1}. Result: {2}".format(1, 2, adder(1, 2))) logger.info("Adding floats {0} and {1}. Result: {2}".format(1.0, 2.0, adder(1.0, 2.0))) logger.info("Adding strings {0} and {1}. Result: {2}".format(1, 2, adder("1", "2")))if __name__ == "__main__": main()If you try to run this module you will get some errors: we need to create a virtual environment and install there the loguru package. We can do this with the following command:Now, you should be able to run our main module. Run the code using poetry in the created virtual environment and you should get the following output:~/tmp/new_package$ poetry run python src/new_package/main.pyCool new_package project!2020-04-03 16:13:41.202 | INFO | __main__:main:10 - We are here: /home/yury/tmp/new_package/src/new_package/main.py2020-04-03 16:13:41.202 | INFO | __main__:main:11 - Adding integers 1 and 2. Result: 32020-04-03 16:13:41.202 | INFO | __main__:main:12 - Adding floats 1.0 and 2.0. Result: 3.02020-04-03 16:13:41.202 | INFO | __main__:main:13 - Adding strings 1 and 2. Result: 12IntelliSense and Code NavigationThe first thing every developer currently needs is IntelliSense code completion tool. In order to check if it works in VSCode for Python code, just start typing in the editor, e.g., pr and press Ctrl+Space. If IntelliSense is configured properly, VSCode should provide you with a list of possible options to complete the statement, for instance, print. Moreover, if put your cursor on this definition, VSCode should open a tooltip showing the signature and the description of this method.VSCode and Python extension developers make use several tools to provide IntelliSense capabilities. You can choose the provider according to your preferences. Currently, there are two options: either use Microsoft Python Analysis Engine (MPAE) or to use the jedi package. Most probably your Python extension will be configured by default to use MPAE, however I am not sure about this for all platforms. If you prefer to use jedi you can adjust your VSCode preferences: set the python.jediEnabled preference to true. On my Linux machine, this setting is set to true by default. If I set this preference to false, IntelliSense features stop working.So as jedi is crucial for the Python extension it is integrated into it, so you do not need to install additional Python packages. However, you can install this package into the system if you like, e.g., other version. In this case, in VSCode preferences you have to provide also the path to this package directory (see python.jediPath setting for details). However, I do not recommend to do this because the Python extension is regularly updated and therefore, supplied with the latest jedi version, which is tested with this extension.In order to navigate over the code, e.g., in order to be able to run Go to definition, Find references and other commands, you have to configure Language Server Protocol (LSP) provider. Currently, there are three options available (see preference python.languageServer): Jedi, Microsoft (default) and None. If you choose None code navigation and other features provided by LSP will not be available. However, I have not noticed any difference if you use either Microsoft or Jedi on my machine. Just for unification purposes, I set it to Jedi. [Update: 19/04/2020]It seems that now Microsoft has started to enforce usage of MPAE. If Jedi is selected as a IntelliSense engine (python.jediEnabled is set to true), VSCode offers you to enable MPAE and restart your editor. If you agree it sets python.jediEnabled to false and python.languageServer to Microsoft. Good news is that in my case MPAE is working ok as an IntelliSense engine. However, if you do not agree next time you work on a Python project it will show the notification again.When I was looking information about language server protocol (LSP), I have also found other implementations of the protocol, e.g., by Palantir. Although I do not understand the purpose of providing another implementation (it is implemented in Python, similarly to jedi, therefore, speed improvement most probably is not the reason), however I assume that there is background behind this choice, it is just not obvious to me.Sorting ImportsThe power of Python is in the libraries developed for this language. In order to use them in your code, you need to import in your code their definitions and modules. Besides third-party libraries, you may also import modules and definitions from the standard library, which is supplied with the language interpretator and therefore, does not require to be installed), and from your package.When you open Python sources the first thing you usually see is the list of imports. So as there can be lots of imports it may take you some time to understand where are they imported from and what they influence on. However, the same order of the imports in every Python file may facilitate this process and help you to understand key components and the functionality of the code much faster. Therefore, it is very useful to sort the imports according to the same criteria. The isort tool enables us to do this. This package is also integrated into the Python extension, therefore its functionality is available by default.You can organize imports in the current module by pressing Shift+Alt+O. If we do this for our example code, isort will rearrange the modules in the following way:import osfrom loguru import loggerWithin VSCodes Python extension, the isort tool uses the default settings how to organize imports. However, it is possible to change this default behavior for your package. In order to do this, you have to create setup.cfg file in the root of your project. VSCodes Python extension (actually, the corresponding tools) automatically reads configuration from this file and apply it. In order to change the isort behavior, add the [isort] section to setup.cfg and modify the settings you would like to adjust. For instance, in my packages this section looks in the following way:[isort]# isort configuration:# See 3 - one module per line in parenthesismulti_line_output = 3# if the trailing comma should be included for the last modulesinclude_trailing_comma = true# where to put unrecognized importsdefault_section = FIRSTPARTY# Should be: max_string_length - 1line_length = 79In order to understand what is changed I have added the comments to each changed setting. More isort settings and their description you can find here. However, I would like to talk a little bit more about the line length setting. There are a lot of argues in the development community what line length should be. Some people suggest 80 characters (because it is the length of a string in a terminal), other developers prefer 100 characters (because this is the maximum length of a string displayed without being wrapped on popular version control system websites like Github or Gitlab), third group of coders votes for 120 characters (because sometimes there are names of the functions that may exceed 100 characters). All these people have rationale, therefore you should agreed on that with your colleagues and select the value you like if you develop your code solely. Personally, I have decided to use the default value equal to 80 characters.Code RefactoringThe code development is an iterative process. From time to time, developers review and improve their code. This process is called refactoring. The most common refactoring operations are variable or method renaming and method extraction. VSCode also enables you to perform these operations on the Python code. To achieve this, it relies on the Python tool called rope that is developed to facilitate refactoring operations. Although rope provides many different refactoring operations, currently VSCode supports only the two mentioned previously.Although the rope tool is necessary for Python code refactoring in VSCode, it is not supplied with the Python extension. Therefore, you have to install it. So as this tool is required only for development, lets install it as a development dependency.You might need to restart your VSCode in order the features to be activated.If you would try to rename a symbol or to extract a method without rope being installed, VSCode would not be able to do this. Instead, it would offer to install the tool. In order to check that this functionality is working, try to create a variable in your code and rename it by selecting it and pressing F2.You may also improve the speed of this tool by making some changes to its configuration. I would also suggest to make the following changes in the tool preference file (.vscode/.ropeproject/config.py):Uncomment the line prefs['python_files'] = ['*.py']. With this preference, rope would consider only .py files.Add the following directories (.venv, .pytest_cache, .vscode) to the list of the ignored resources prefs['ignored_resources'] = ['*.pyc', '*~', '.ropeproject', '.hg', '.svn', '_svn', '.git', '.tox', '.venv', '.pytest_cache', '.vscode']Style CheckingUsually, there is a number of developers working on the same code. Before starting new project, it is worth to agree on the rules how would you format the code. Otherwise, you might spend a lot of time arguing during the code review. It would be nice if these rules are automatically checked and inconsistencies are reported. The Python community has developed a number of such tools called linters. Besides style checking, linters can also detect some logical errors. Therefore, they are recommended even if you develop your project alone.Among Python linters, pylint and flake8 are the most popular. Both these linters can be integrated with VSCode and the Python extension (pylint is enabled by default). However, I have decided to use flake8. The reason for choosing this linter is that there is a very strict styleguide (a set of flake8 extensions and custom strict rules) developed on top of this linter called wemake-python-styleguide. Moreover, pylint has some drawbacks.In order to start using this styleguide, we have to make some configuration changes in VSCode and the Python extension. First of all, we have to enable flake8 as the Python linter: set python.linting.flake8Enabled to true; and disable pylint: set python.linting.pylintEnabled to false. Check that the python.linting.flake8Path value is equal to flake8. Wemake-python-styleguide depends on flake8 so it will be automatically installed. Moreover, I would also suggest to disable linting for some directories. Add .venv/**/*.py and .pytest_cache/**/*.py to python.linting.ignorePatterns:"python.linting.ignorePatterns": [".vscode/*.py", "**/site-packages/**/*.py", ".venv/**/*.py", ".pytest_cache/**/*.py"]So as there are many different linters developed by the Python community, the Python extension has many settings regarding them. For instance, there you can enable bandit, pylama, pydocstyle, pycodestyle and other linters to check your project. However, I do not recommend you to do this. Many of them are already integrated into wemake-python-styleguide and some of them are outdated, therefore you have to understand exactly why you enable them.Now, lets install the wemake-python-styleguide package:$ poetry add --dev wemake-python-styleguideSo as this package depends on many packages, it may take some time to install it. After it is installed, we need to make some configuration adjustments. So as there are many different plugins and thus, many different configurations, I have taken as a template the settings from wemake-python-package and modified them according to my preferences. Add the following lines to setup.cfg (I have added comments to some settings):[flake8]# Base flake8 configuration:# = wemakeshow-source = Truestatistics = Falsedoctests = True# Plugins:max-complexity = 6max-line-length = 80# strings are in single or double quotes# inline-quotes = double# wemake-python-styleguide settings:i-control-code = True# Disable some pydocstyle checks:# Exclude some pydoctest checks globally:ignore = # Missing docstring in public module # D100 # Missing docstring in public package # D104 # Missing docstring in public nested class # D106 # First line should be in imperative mood D401 # line break after binary operator W504 # per-file ignoring (better to live) X100 # Unknown directive type "XXX". RST303 # Unknown interpreted text role "XXX". RST304 # Darglint configuration # The docstring parameter type doesn't match function. # DAR103 # The docstring parameter type doesn't match function. # DAR203# Excluding some directories:exclude = .git __pycache__ .venv .eggs *.egg # add the following directories .venv .mypy_cache .vscode# Ignoring some errors in some files:per-file-ignores = # Enable `assert` keyword and magic numbers for tests: tests/*.py: S101, WPS226, WPS432[darglint]# darglint configuration:# = longYou may need to restart your VSCode for the settings to take effect. Now, for our test module you should get the following list of errors and warnings: 1:1 D100 Missing docstring in public module 4:1 I003 isort expected 1 blank line in imports, found 0 5:1 E302 expected 2 blank lines, found 1 5:1 D103 Missing docstring in public function 8:1 E302 expected 2 blank lines, found 1 8:1 D103 Missing docstring in public function 9:5 WPS421 Found wrong function call: print 9:11 Q000 Remove bad quotes 10:17 P101 format string does contain unindexed parameters 10:17 Q000 Remove bad quotes 11:17 Q000 Remove bad quotes 11:81 E501 line too long (85 > 80 characters) 12:17 Q000 Remove bad quotes 12:70 WPS432 Found magic number: 2.0 12:81 E501 line too long (91 > 80 characters) 12:86 WPS432 Found magic number: 2.0 13:17 Q000 Remove bad quotes 13:78 Q000 Remove bad quotes 13:81 E501 line too long (88 > 80 characters) 13:83 Q000 Remove bad quotes 16:16 Q000 Remove bad quotesLets remove some errors and warnings according to the recommendations. If you do not understand why the linter generates an error I recommend to refer to the list of the wemake-python-styleguide violations. There you will find the links to the violations generated by different integrated plugins. For flake8 violations, I would recommend to check the www.flake8rules.com website. There you can read about different flake8 violations. In order to do this, you need just modify the URL by adding the name of the error to the end of the path. For instance, if you would like to know what W503 error means you have to open the following link: modifying the code according to the recommendations, it should look like the following:# -*- coding: UTF-8 -*-"""Test Module.Author: Yury ZhauniarovichDate created: 04/04/2020"""import osfrom loguru import loggerdef adder(i1, i2): """This function adds two numbers.""" return (i1 + i2)def main(): """Main function.""" logger.info('Cool new_package project!') logger.info('We are here: {0}'.format(os.path.abspath(__file__))) logger.info('Adding integers {0} and {1}. Result: {2}'.format(1, 2, adder(1, 2),)) logger.info('Adding floats {0} and {1}. Result: {2}'.format(1.0, 2.0, adder(1.0, 2.0),)) logger.info('Adding strings {0} and {1}. Result: {2}'.format('1', '2', adder('1', '2'),))if __name__ == '__main__': main()Additional flake8 PluginsAlthough the wemake-python-styleguide package comes with a number of good flake8 plugins there is always a room for improvement. In particular, the authors of the wemake-python-styleguide package recommend to use the following flake8 plugins:cohesion to measure code cohesion.dlint to ensure Python code is secure and ensure best coding practices.Besides these plugins, personally I would recommend also to check the following ones:If you want to use some of these plugins in your project, just install them as development dependencies to your virtual environment using poetry (example for cohesion and dlint plugins):$ poetry add --dev cohesion dlintThe wemake-python-styleguide package is under active development. Therefore, its updates may change the list of the integrated flake8 plugins. I recommend to check the changelog of this styleguide regularly.I would like also to recommend the awesome-flake8-extensions repository where you can find an huge list of different flake8 plugins that can be added as well.Code FormattingSome of the errors found by a linter could be automatically corrected: so-called auto-formatters can take this task. Personally, as a Python auto-formatter I use autopep8. There are two reasons for this. First, this auto-formatter is fully compatible with the wemake-python-styleguide. Second, it allows to format a part of the code (select the code you would like to format and press Ctrl+K Ctrl+F). Other auto-formatters, e.g., black, do not allow you to do that.In order to use this auto-formatter, you need to set the auto-formatter provider (python.formatting.provider) to autopep8. Also, you must add it as a development dependency to your project:$ poetry add --dev autopep8Type CheckingIn our example, there is a function adder that adds two numbers. However, so as Python is a dynamically typed language this function also works if we provide two strings as parameters. In this case, the function will concatenate these two strings. This behaviour is unusual and may lead to a bug. In order to add more patency, Python developers have started to use type annotations.To check these annotations the mypy tool is used. To enable mypy checks in VSCode you need to do the following configuration. First, in the settings (Ctrl+,), change python.linting.mypyEnabled to True and provide the path to the mypy executable (mypy) in the python.linting.mypyPath. Second, install mypy as a development dependency:Now, if you save a file mypy will check if all contracts hold. However, for our example it would not generate any error. The issue is that the default mypy settings are quite relaxed. To make them more rigid, add the following lines to your setup.cfg (this configuration is partially taken from the settings of the wemake-python-package):[mypy]# mypy configurations: = src/, tests/allow_redefinition = Falsecheck_untyped_defs = Truedisallow_any_explicit = Truedisallow_any_generics = Truedisallow_untyped_calls = Trueignore_errors = Falseignore_missing_imports = Trueimplicit_reexport = Falselocal_partial_types = Truestrict_optional = Truestrict_equality = Trueno_implicit_optional = Truewarn_no_return = Truewarn_unused_ignores = Truewarn_redundant_casts = Truewarn_unused_configs = Truewarn_unreachable = TrueNote that storing the code of a package in the src directory allows me to copy the same setup.cfg from one project to another. Now, when you save the file you should get the list of the following mypy errors:src/new_package/main.py:27: error: Call to untyped function "adder" in typed contextsrc/new_package/main.py:32: error: Call to untyped function "adder" in typed contextsrc/new_package/main.py:37: error: Call to untyped function "adder" in typed contextsrc/new_package/main.py:42: error: Call to untyped function "main" in typed contextLets annotate our adder function to accept only numbers:def adder(i1: Union[int, float], i2: Union[int, float],) -> Union[int, float]: """This function adds two numbers.""" return (i1 + i2)Now, if you run mypy it will generate an error for the line where we try to add two strings. Lets remove this line from our example.Testing Python CodeTesting your code is essential if you plan to update your package regularly. Although, I am only in the beginning of my Test Driven Development (TDD) journey I have decided to describe in this article how to configure VSCode to test your modules. By default, when you create a project using poetry it adds pytest as a test runner and creates a boilerplate code of a test example. Moreover, pytest assertions seem to me more natural therefore I prefer this framework. So, lets configure VSCode to use this test runner.At first, lets configure VSCode to work with pytest. In order to do this, open settings (Ctrl+,) and enable pytest: set python.testing.pytestEnabled to true.If you try now to run your tests in Test Explorer, VSCode would generate an error because it cannot discover them. Similarly, if you try to run test discovery through command line using the poetry run pytest --collect-only command, pytest would generate an error that it cannot import a module. This issue is because we store our sources in the src/ directory. In order to fix this, you need to install the package at first in order to test it. Luckily, you can do this just with one command:If you store the code in the src/ directory as I do and as some Python developers recommend, in order to run the tests you need to install this package at first.If you store your code in src-less layout pytest would be able to discover the tests without the package being installed.In order to speed up test execution, lets install the pytest-xdist package. This tool is able to distribute tests between multiple test executors thus, they can be run on multiple CPU cores in parallel:$ poetry add --dev pytest-xdistNow, lets add some settings to our setup.cfg:[tool:pytest]# search for tests only in tests/ directorytestpaths = tests# make XPASS (unexpectedly passing) result to fail the test suitexfail_strict = trueaddopts = # report details (verbose) -v # xdist - number of parallel test executors (auto - detect automatically) -n auto # report the local variables for every failure with the stacktrace -l # report the reasons for all tests that skipped, xfailed, or xpassed -rsxX # treat unregistered markers as errors allowing to avoid typos --strict # short traceback format --tb=short # execute doctests directly from docstrings of your classes and functions --doctest-modulesAfter that, you should be able to run your tests from Test Explorer and see the results of their execution.Collecting Code CoverageLast but not least, lets add code coverage data collection. In order to do this, we have to install the pytest-cov package (as a dependency it has the coverage.py package that collects coverage information):$ poetry add --dev pytest-covLets modify our setup.cfg to enable pytest to collect coverage information during test execution. In particular, we need to modify the [tool:pytest] and add two additional sections:[tool:pytest]# search for tests only in tests/ directorytestpaths = tests# make XPASS (unexpectedly passing) result to fail the test suitexfail_strict = trueaddopts = # report details (verbose) -v # xdist - number of parallel test executors -n auto # report the local variables for every failure with the stacktrace -l # report the reasons for all tests that skipped, xfailed, or xpassed -rsxX # treat unregistered markers as errors allowing to avoid typos --strict # short traceback format --tb=short # execute doctests directly from docstrings of your classes and functions --doctest-modules # coverage --cov # generate html coverage report and store it into htmlcov dir --cov-report=html:htmlcov[coverage:run]# directory to run coverage onsource = src/[coverage:report]# do not consider the following lines during coverage calculationexclude_lines = # Have to re-enable the standard pragma pragma: no cover # Don't complain about missing debug-only code: def __repr__ if self\.debug # Don't complain if tests don't hit defensive assertion code: raise AssertionError raise NotImplementedError # Don't complain if non-runnable code isn't run: if 0: if __name__ == .__main__.:Now, if you run your tests pytest will also collect code coverage information and put it in the html format to the htmlcov directory. If you need reports in other formats you can modify the settings accordingly.Final ConfigurationFinally, in the end of this tutorial your setup.cfg and pyproject.toml should look in the following way:pyproject.toml[tool.poetry]name = ""version = "0.1.0"description = ""authors = ["Author Name "][tool.poetry.dependencies]python = "^3.8"[tool.poetry.dev-dependencies]pytest = "^5.2"rope = "^0.16.0"wemake-python-styleguide = "^0.14.0"cohesion = "^1.0.0"dlint = "^0.10.3"flake8-pytest = "^1.3"flake8-pytest-style = "^1.0.0"flake8-coding = "^1.3.2"flake8-class-attributes-order = "^0.1.0"flake8-builtins = "^1.5.2"flake8-mutable = "^1.2.0"autopep8 = "^1.5"mypy = "^0.770"pytest-xdist = "^1.31.0"pytest-cov = "^2.8.1"[build-system]requires = ["poetry>=0.12"]build-backend = "poetry.masonry.api"setup.cfg[isort]# isort configuration:# See 3 - one module per line in parenthesismulti_line_output = 3# if the trailing comma should be included for the last modulesinclude_trailing_comma = true# where to put unrecognized importsdefault_section = FIRSTPARTY# Should be: max_string_length - 1line_length = 79[flake8]# Base flake8 configuration:# = wemakeshow-source = Truestatistics = Falsedoctests = True# Plugins:max-complexity = 6max-line-length = 80# strings are in single or double quotes# inline-quotes = double# wemake-python-styleguide settings:i-control-code = True# Disable some pydocstyle checks:# Exclude some pydoctest checks globally:ignore = # Missing docstring in public module # D100 # Missing docstring in public package # D104 # Missing docstring in public nested class # D106 # First line should be in imperative mood D401 # line break after binary operator W504 # per-file ignoring (better to live) X100 # Unknown directive type "XXX". RST303 # Unknown interpreted text role "XXX". RST304 # Darglint configuration # The docstring parameter type doesn't match function. # DAR103 # The docstring parameter type doesn't match function. # DAR203# Excluding some directories:exclude = .git __pycache__ .venv .eggs *.egg # add the following directories .venv .mypy_cache .vscode# Ignoring some errors in some files:per-file-ignores = # Enable `assert` keyword and magic numbers for tests: tests/*.py: S101, WPS226, WPS432[darglint]# darglint configuration:# = long[mypy]# mypy configurations: = src/, tests/allow_redefinition = Falsecheck_untyped_defs = Truedisallow_any_explicit = Truedisallow_any_generics = Truedisallow_untyped_calls = Trueignore_errors = Falseignore_missing_imports = Trueimplicit_reexport = Falselocal_partial_types = Truestrict_optional = Truestrict_equality = Trueno_implicit_optional = Truewarn_no_return = Truewarn_unused_ignores = Truewarn_redundant_casts = Truewarn_unused_configs = Truewarn_unreachable = True[tool:pytest]# search for tests only in tests/ directorytestpaths = tests# make XPASS (unexpectedly passing) result to fail the test suitexfail_strict = trueaddopts = # report details (verbose) -v # xdist - number of parallel test executors -n auto # report the local variables for every failure with the stacktrace -l # report the reasons for all tests that skipped, xfailed, or xpassed -rsxX # treat unregistered markers as errors allowing to avoid typos --strict # short traceback format --tb=short # execute doctests directly from docstrings of your classes and functions --doctest-modules # coverage --cov # generate html coverage report and store it into htmlcov dir --cov-report=html:htmlcov[coverage:run]# directory to run coverage onsource = src/[coverage:report]# do not consider the following lines during coverage calculationexclude_lines = # Have to re-enable the standard pragma pragma: no cover # Don't complain about missing debug-only code: def __repr__ if self\.debug # Don't complain if tests don't hit defensive assertion code: raise AssertionError raise NotImplementedError # Don't complain if non-runnable code isn't run: if 0: if __name__ == .__main__.:Of course, with the lapse of time the versions of the dependencies would expire. You would need to update them from time to time. You could also replace the version number with a wildcard so that, when you copy the pyproject.toml file to a new package, poetry automatically picks the latest available dependencies versions and lock them for your project.Besides the tools we have considered so far that can be integrated with VSCode, I recommend you to check the following that can also improve the quality of your Python package:vulture to find the dead code.import-linter to enforce rules for the internal and external imports.safety checks if there is a vulnerability in a dependency.They all can be run from command line and report on potential issues. Besides them, I would like to highlight the pre-commit tool staying aside. It is used to develop hook scripts that are run before a commit or a push to projects git repository. This functionality can be used to run all the checks considered so far automatically before a commit or a push. This can be very useful, for instance, if your project is big, and running all the checks after every save operation may cause unnecessary delays. You can read this article in order to understand how to integrate this tool.ConclusionIn this article, I have considered how to configure your VSCode for Python development. So as Python world is quite diverse, you may use other tools in your projects. However, I hope that using the information from this article would help you to integrate them as well. Watch Now This tutorial has a related video course created by the Real Python team. Watch it together with the written tutorial to deepen your understanding: Python Development in Visual Studio Code (Setup Guide) One of the coolest code editors available to programmers, Visual Studio Code, is an open-source, extensible, light-weight editor available on all platforms. Its these qualities that make Visual Studio Code from Microsoft very popular, and a great platform for Python development.In this article, youll learn about Python development in Visual Studio Code, including how to:Install Visual Studio CodeDiscover and install extensions that make Python development easyWrite a straightforward Python applicationLearn how to run and debug existing Python programs in VS CodeConnect Visual Studio Code to Git and GitHub to share your code with the worldWe assume you are familiar with Python development and already have some form of Python installed on your system (Python 2.7, Python 3.6/3.7, Anaconda, or others). Screenshots and demos for Ubuntu and Windows are provided. Because Visual Studio Code runs on all major platforms, you may see slightly different UI elements and may need to modify certain commands.If you already have a basic VS Code setup and youre hoping to dig deeper than the goals in this tutorial, you might want to explore some advanced features in VS Code.Free Bonus: 5 Thoughts On Python Mastery, a free course for Python developers that shows you the roadmap and the mindset youll need to take your Python skills to the next level.Installing Visual Studio CodeInstalling Visual Studio Code is very accessible on any platform. Full instructions for Windows, Mac, and Linux are available, and the editor is updated monthly with new features and bug fixes. You can find everything at the Visual Studio Code website: In case you were wondering, Visual Studio Code (or VS Code for short) shares almost nothing other than a name with its larger Windows-based namesake, Visual Studio.Note: To learn how to set up VS Code as part of a full Python coding environment on a Windows machine, check out this comprehensive guide.Visual Studio Code has built-in support for multiple languages and an extension model with a rich ecosystem of support for others. VS Code is updated monthly, and you can keep up to date at the Microsoft Python blog. Microsoft even makes the VS Code GitHub repo available for anyone to clone and contribute. (Cue the PR flood.)The VS Code UI is well documented, so I wont rehash it here:As stated above, VS Code supports development in multiple programming languages through a well-documented extension model. The Python extension enables Python development in Visual Studio Code, with the following features: Visual Studio Code extensions cover more than just programming language capabilities: Here are some other extensions and settings I find useful:GitLens provides tons of useful Git features directly in your editing window, including blame annotations and repository exploration features.Auto save is easily turned on by selecting File, Auto Save from the menu. The default delay time is 1000 milliseconds, which is also configurable.Settings Sync allows you to synchronize your VS Code settings across different installations using GitHub. If you work on different machines, this helps keep your environment consistent across them.Docker lets you quickly and easily work with Docker, helping author Dockerfile and docker-compose.yml, package and deploy your projects, and even generate the proper Docker files for your project.Of course, you may discover other useful extensions as you use VS Code. Please share your discoveries and settings in the comments!Discovering and installing new extensions and themes is accessible by clicking on the Extensions icon on the Activity Bar. You can search for extensions using keywords, sort the results numerous ways, and install extensions quickly and easily. For this article, install the Python extension by typing python in the Extensions item on the Activity Bar, and clicking Install: You can find and install any of the extensions mentioned above in the same manner.One important thing to mention is that Visual Studio Code is highly configurable through user and workspace settings.User settings are global across all Visual Studio Code instances, while workspace settings are local to the specific folder or project workspace. Workspace settings give VS Code tons of flexibility, and I call out workspace settings throughout this article. Workspace settings are stored as .json files in a folder local to the project workspace called .vscode.Lets start our exploration of Python development in Visual Studio Code with a new Python program. In VS Code, type Ctrl+N to open a new File. (You can also select File, New from the menu.)Note: The Visual Studio Code UI provides the Command Palette, from which you can search and execute any command without leaving the keyboard. Open the Command Palette using Ctrl+Shift+P, type File: New File, and hit Enter to open a new file.No matter how you get there, you should see a VS Code window that looks similar to the following: Once a new file is opened, you can begin entering code.For our test code, lets quickly code up the Sieve of Eratosthenes (which finds all primes less than a given number). Begin typing the following code in the new tab you just opened: You should see something similar to this: Wait, whats going on? Why isnt Visual Studio Code doing any keyword highlighting, any auto-formatting, or anything really helpful? What gives?The answer is that, right now, VS Code doesnt know what kind of file its dealing with. The buffer is called Untitled-1, and if you look in the lower right corner of the window, youll see the words Plain Text.To activate the Python extension, save the file (by selecting File, Save from the menu, File:Save File from the Command Palette, or just using Ctrl+S) as sieve.py. VS Code will see the .py extension and correctly interpret the file as Python code. Now your window should look like this: Thats much better! VS Code automatically reformats the file as Python, which you can verify by inspecting the language mode in the lower left corner.If you have multiple Python installations (like Python 2.7, Python 3.x, or Anaconda), you can change which Python interpreter VS Code uses by clicking the language mode indicator, or selecting Python: Select Interpreter from the Command Palette. VS Code supports formatting using pep8 by default, but you can select black or yapf if you wish. Lets add the rest of the Sieve code now. To see IntelliSense at work, type this code directly rather than cut and paste, and you should see something like this: Heres the full code for a basic Sieve of Eratosthenes: As you type this code, VS Code automatically indents the lines under for and if statements for you properly, adds closing parentheses, and makes suggestions for you. Thats the power of IntelliSense working for you.Now that the code is complete, you can run it. There is no need to leave the editor to do this: Visual Studio Code can run this program directly in the editor. Save the file (using Ctrl+S), then right-click in the editor window and select Run Python File in Terminal: You should see the Terminal pane appear at the bottom of the window, with your code output showing.You may have seen a pop up appear while you were typing, stating that linting was not available. You can quickly install linting support from that pop up, which defaults to PyLint. VS Code also supports other linters. Heres the complete list at the time of this writing:pylintflake8mypypydocstylepep8prospectorpyllamabanditThe Python linting page has complete details on how to setup each linter.Note: The choice of linter is a project workspace setting, and not a global user setting.In the Sieve of Eratosthenes example, you created a single Python file. Thats great as an example, but many times, youll create larger projects and work on them over a longer period of time. A typical new project work flow might look like this:Create a folder to hold the project (which may include a new GitHub project)Change to the new folderCreate the initial Python code using the command code filename.pyUsing Visual Studio Code on a Python project (as opposed to a single Python file) opens up tons more functionality that lets VS Code truly shine. Lets take a look at how it works with a larger project.Late in the previous millennium, when I was a much younger programmer, I wrote a calculator program that parsed equations written in infix notation, using an adaptation of Edsger Dijkstras shunting yard algorithm.To demonstrate the project-focused features of Visual Studio Code, I began recreating the shunting yard algorithm as an equation evaluation library in Python. To continue following along, feel free to clone the repo locally.Once the folder is created locally, you can open the entire folder in VS Code quickly. My preferred method (as mentioned above) is modified as follows, since I already have the folder and basic files created: VS Code understands, and will use, any virtualenv, pipenv, or conda environments it sees when opened this way. You dont even need to start the virtual environment first! You can even open a folder from the UI, using File, Open Folder from the menu, Ctrl+K, Ctrl+O from the keyboard, or File:Open Folder from the Command Palette.For my equation eval library project, heres what I see: When Visual Studio Code opens the folder, it also opens the files you last had opened. (This is configurable.) You can open, edit, run, and debug any file listed. The Explorer view in the Activity Bar on the left gives you a view of all the files in the folder and shows how many unsaved files exist in the current set of tabs.VS Code can automatically recognize existing Python tests written in the unittest framework, or the pytest or Nose frameworks if those frameworks are installed in the current environment. I have a unit test written in unittest for the equation eval library, which you can use for this example.To run your existing unit tests, from any Python file in the project, right-click and select Run Current Unit Test File. Youll be prompted to specify the test framework, where in the project to search for tests, and the filename pattern your tests utilize.All of these are saved as workspace settings in your local .vscode/settings.json file and can be modified there. For this equation project, you select unittest, the current folder, and the pattern *_test.py.Once the test framework is set up and the tests have been discovered, you can run all your tests by clicking Run Tests on the Status Bar and selecting an option from the Command Palette: You can even run individual tests by opening the test file in VS Code, clicking Run Tests on the Status Bar, and selecting the Run Unit Test Method and the specific test to run. This makes it trivial to address individual test failures and re-run only failed tests, which is a huge time-saver! Test results are shown in the Output pane under Python Test Log.Even though VS Code is a code editor, debugging Python directly within VS Code is possible. VS Code offers many of the features you would expect from a good code debugger, including:Automatic variable trackingWatch expressionsBreakpointsCall stack inspectionYou can see them all as part of the Debug view on the Activity Bar: The debugger can control Python apps running in the built-in terminal or an external terminal instance. It can attach to an already running Python instances, and can even debug Django and Flask apps.Debugging code in a single Python file is as simple as starting the debugger using F5. You use F10 and F11 to step over and into functions respectively, and Shift+F5 to exit the debugger. Breakpoints are set using F9, or using the mouse by clicking in the left margin in the editor window.Before you start debugging more complicated projects, including Django or Flask applications, you need to setup and then select a debug configuration. Setting up the debug configuration is relatively straightforward. From the Debug view, select the Configuration drop-down, then Add Configuration, and select Python: Visual Studio Code will create a debug configuration file under the current folder called .vscode/launch.json, which allows you to setup specific Python configurations as well as settings for debugging specific apps, like Django and Flask.You can even perform remote debugging, and debug Jinja and Django templates. Close the launch.json file in the editor and select the proper configuration for your application from the Configuration drop-down.VS Code has built-in support for source control management, and ships with support for Git and GitHub right out of the box. You can install support for other SCMs in VS Code, and use them side by side. Source control is accessible from the Source Control view: If your project folder contains a .git folder, VS Code automatically turns on the full range of Git/GitHub functionality. Here are some of the many tasks you can perform: All of this functionality is available directly from the VS Code UI: VS Code will also recognize changes made outside the editor and behave appropriately.Committing your recent changes within VS Code is a fairly straightforward process. Modified files are shown in the Source Control view with an M marker, while new untracked files are marked with a U. Stage your changes by hovering over the file and then clicking the plus sign (+). Add a commit message at the top of the view, and then click the check mark to commit the changes: You can push local commits to GitHub from within VS Code as well. Select Sync from the Source Control view menu, or click Synchronize Changes on the status bar next to the branch indicator.Visual Studio Code is one of the coolest general purpose editors and a great candidate for Python development. In this article, you learned:How to install VS Code on any platformHow to find and install extensions to enable Python-specific featuresHow VS Code makes writing a simple Python application easierHow to run and debug existing Python programs within VS CodeHow to work with Git and GitHub repositories from VS CodeVisual Studio Code has become my default editor for Python and other tasks, and I hope you give it a chance to become yours as well.If you have questions or comments, please reach out in the comments below. There is also a lot more information at the Visual Studio Code website than we could cover here.The author sends thanks to Dan Taylor from the Visual Studio Code team at Microsoft for his time and invaluable input in this article. Watch Now This tutorial has a related video course created by the Real Python team. Watch it together with the written tutorial to deepen your understanding: Python Development in Visual Studio Code (Setup Guide) This post is a guide to help you create a new Python project with its own virtual environment using VSCode/VSCodium and GitHub. Introduction This guide aims to assist you in setting up a Python project with its own virtual environment using VSCode/VSCodium and GitHub.The goal is not to go into exhaustive details of the tools, but to provide a quick start to create a new Python project and manage it with Git.I strongly recommend to use Git even when working alone, as it helps in tracking changes and provides a backup of your code.You wouldnt want to lose weeks of work due to a computer crash! First, ensure you have the following installed: Git Python VSCode or VSCodium Alternatives exist for each of these tools. For the code editor, PyCharm, Vim, Neovim, Emacs, or many others can be used.However, tools with features like syntax highlighting, code completion, linting, git integration, and debugging capabilities are preferred.PyCharm comes with a lot of features out of the box, VSCode/VSCodium will require some extensions, Vim/NeoVim and Emacs will require a lot of configuration and time to learn the basics and to obtain the same features as the other tools.Personally, I use VSCode because I can easily switch between Python, C++, and other languages without having to spend a lot of time configuring it. For version control, GitLab, Bitbucket, or other alternatives can also be used. For managing Python environments, this guide utilizes the built-in venv module. Alternatives like virtualenv, conda, or poetry exist.Personally, I use venv because it is built-in and easy to use. Other resources such as The Hitchhikers Guide to Python, Real Python, or Python.org to learn more about Python, its ecosystem, and best practices (how to structure your project, how to write tests, how to document your code, etc.).There are also a lot of tools to help you organize your project and write better and cleaner code : Black for code formatting. Mypy for static type checking. Pylint for linting. Isort for import sorting. Ruff for linting and code formatting. CookieCutter for project templating. and many others Installing VSCode Python extensions Open VSCode. On the left, click on the Extensions icon (four squares icon). Search for Python and install the Python Extension Pack. You can also install other extensions like Black Formatter, Python Type Hint, or Pylint. You can search for other extensions that you might find useful. Creating a new repository on github Log in to GitHub. Click on the New button (green on the top left). Choose a name for your repository (e.g. my_project), add a description, and configure privacy settings (public or private, you can change it later). Check Add a README file and add a gitignore file for Python. Select the license you want to use (e.g. MIT). Click on Create repository. Your new repository is now set up on GitHub. Cloning the repository on your computer through VSCode In VSCode, open the command palette (Ctrl+Shift+P on Linux), type git clone, and select Git: Clone. Choose Clone from github. Enter your github credentials. Select the repository you want to clone (e.g. your_github_name/my_project, it should be the first one in the list). Choose a local folder for cloning (this will create a new folder with the name of your repository inside the selected folder). You have now cloned the repository on your computer. VSCode will ask you if you want to open the repository, click on Open. You have now opened the repository in VSCode.You can see the README.md, .gitignore, and LICENSE files in the Explorer on the left. You can then create files and add send them to the repository with git.Look at this video tutorial to learn how to use git in VSCode (staging, committing, pushing, pulling, etc.).It is also possible to create the repository locally then publish it on github.I proposed to do it from github because it comes directly with a README.md, a gitignore file, and a license file. Creating a Python environment Open the terminal in VSCode (On the top menu, click on Terminal -> New Terminal, the shortcut should be indicated here). Check the path to the python interpreter path you are currently using with which python3, probably /usr/bin/python3.xx. Create a virtual environment in a venv folder with python3 -m venv venv, you can specify the exact python version by using python3.11 or python3.12 depending on the version installed on your computer. Activate the virtual environment with the command : source venv/bin/activate (on Windows, the command should be venv\Scripts\activate). You should see (venv) at the beginning of the command line, this means you are now using the python interpreter from the virtual environment. Check the path to the python interpreter path you are currently using with which python3, the result should be /path_to_current_directory/venv/bin/python3. This means you are using the python interpreter from the virtual environment, installing packages with pip will install them in this virtual environment and not in the system python. You can upgrade pip with the command : pip install --upgrade pip. You can now install the dependencies of your project with pip.If you want to install a package, for example black, you can do it with the command pip install black. The package will be installed in the virtual environment.But if you only use pip install package and someone else clones your repository, they will not have the package installed as the virtual environment is not committed to the repository (it is in the .gitignore file, see this stackoverflow question for more information on this topic).To avoid this, you can create a requirements.txt file with the command pip freeze > requirements.txt and commit it to your repository.This file will list all the dependencies of your project and their versions, for tools like black or mypy that you normally dont directly use in your code, you can remove the version number to install the latest version, but for libraries that you use in your code, it is better to specify the version number.You can then install the dependencies with the command pip install -r requirements.txt. In the README.md file, you can add a section Install dependencies with the commands to create the virtual environment and install the dependencies. Coding! You can now start coding your project.You can take a look at the section on the structure of a project on the hitchhikers guide to python to have an idea on how to organize your project. After few minutes of coding (or seconds if you are very efficient!), you will probably have some bugs in your code.You can use the debugger of VSCode to help you find and fix them.The docs on debugging python in VSCode should help you to get started with the debugger. Visual Studio Code (VS Code) is a powerful, open-source code editor that supports multiple programming languages, including Python. In this guide, we will walk through the steps to create your first Python project in VS Code, covering everything from installation to running your code. Lets dive in!Setting Up Your EnvironmentThe first step to creating a Python project in Visual Studio Code is to ensure you have the right environment set up. This involves installing Python and the necessary extensions in VS Code.Installing PythonTo run Python code, you need to have Python installed on your machine. Follow these steps:Visit the official Python website.Download the latest version of Python for your operating system.Follow the installation instructions provided on the website.Installing Visual Studio CodeIf you havent installed Visual Studio Code yet, you can download it from the official VS Code website. Follow the installation prompts for your operating system.Creating Your Project FolderOnce you have Python and VS Code installed, you can start creating your Python project.Open Visual Studio Code.Click on the Open Folder button or go to File > Open Folder.Create a new folder for your project, for example, my_first_py_project, and select it.Creating Your First Python FileNow that your project folder is set up, lets create a Python file.In the Explorer view, right-click on your project folder and select New File.Name your file main.py.Writing Your First Python CodeOpen your main.py file and type the following code:print("Hello, World!")To save your file, press Ctrl + S.Installing the Python ExtensionTo effectively run and debug Python code in VS Code, you need to install the Python extension provided by Microsoft.Click on the Extensions icon in the Activity Bar on the side of the window or press Ctrl + Shift + X.Search for Python in the Extensions Marketplace.Click on the Python extension by Microsoft and then click Install.Running Your Python CodeOnce the extension is installed, you can run your Python code directly from VS Code.Open the terminal by going to Terminal > New Terminal.In the terminal, type the following command to run your Python file:python main.pyThis command will execute your script, and you should see Hello, World! printed in the terminal.Using the Run ButtonTo make running your code even easier, you can use the run button that appears in the top right corner of the editor once the Python extension is installed.Simply click the play button (run button) to execute your code without opening the terminal manually.Creating Additional Python FilesAs you expand your project, you may want to create additional Python files. You can repeat the steps to create a new file, for example, main2.py, and write a simple print statement:print("Hello from main2!")Run it the same way you did with main.py.ConclusionCongratulations! Youve successfully set up your first Python project in Visual Studio Code. You learned how to create a project folder, write Python code, install necessary extensions, and run your code efficiently. If you have any questions, feel free to leave them in the comments below!For more tutorials and resources, you can check out supporting my work or visit my YouTube channel In this tutorial, you will learn how to use Python 3 in Visual Studio Code to create, run, and debug a Python "Roll a dice!" application, work with virtual environments, use packages, and more! By using the Python extension, you turn VS Code into a great, lightweight Python editor.If you are new to programming, check out the Visual Studio Code for Education - Introduction to Python course. This course offers a comprehensive introduction to Python, featuring structured modules in a ready-to-code browser-based development environment.To gain a deeper understanding of the Python language, you can explore any of the programming tutorials listed on python.org within the context of VS Code.For a Data Science focused tutorial with Python, check out our Data Science section.PrerequisitesTo successfully complete this tutorial, you need to first set up your Python development environment. Specifically, this tutorial requires: Install a Python interpreterAlong with the Python extension, you need to install a Python interpreter. Which interpreter you use is dependent on your specific needs, but some guidance is provided below.WindowsInstall Python from python.org. Use the Download Python button that appears first on the page to download the latest version.Note: If you don't have admin access, an additional option for installing Python on Windows is to use the Microsoft Store. The Microsoft Store provides installs of supported Python versions.For additional information about using Python on Windows, see Using Python on Windows at Python.orgmacOSThe system install of Python on macOS is not supported. Instead, a package management system like Homebrew is recommended. To install Python using Homebrew on macOS use brew install python3 at the Terminal prompt.Note: On macOS, make sure the location of your VS Code installation is included in your PATH environment variable. See these setup instructions for more information.LinuxThe built-in Python 3 installation on Linux works well, but to install other Python packages you must install pip with get-pip.py.Other optionsData Science: If your primary purpose for using Python is Data Science, then you might consider a download from Anaconda. Anaconda provides not just a Python interpreter, but many useful libraries and tools for data science.Windows Subsystem for Linux: If you are working on Windows and want a Linux environment for working with Python, the Windows Subsystem for Linux (WSL) is an option for you. If you choose this option, you'll also want to install the WSL extension. For more information about using WSL with VS Code, see VS Code Remote Development or try the Working in WSL tutorial, which will walk you through setting up WSL, installing Python, and creating a Hello World application running in WSL.Note: To verify that you've installed Python successfully on your machine, run one of the following commands (depending on your operating system):Linux/macOS: open a Terminal Window and type the following command:python3 --versionWindows: open a command prompt and run the following command:py -3 --versionIf the installation was successful, the output window should show the version of Python that you installed.Alternatively, you can use the py -0 command in the VS Code integrated terminal to view the versions of python installed on your machine. The default interpreter is identified by an asterisk (*).Start VS Code in a workspace folderBy starting VS Code in a folder, that folder becomes your "workspace".Using a command prompt or terminal, create an empty folder called "hello", navigate into it, and open VS Code (code) in that folder (.) by entering the following commands:mkdir hellocd hellocode .Note: If you're using an Anaconda distribution, be sure to use an Anaconda command prompt.Alternately, you can create a folder through the operating system UI, then use VS Code's File > Open Folder to open the project folder.Create a virtual environmentA best practice among Python developers is to use a project-specific virtual environment. Once you activate that environment, any packages you then install are isolated from other environments, including the global interpreter environment, reducing many complications that can arise from conflicting package versions. You can create non-global environments in VS Code using Venv or Anaconda with Python: Create Environment.Open the Command Palette (P (Windows, Linux Ctrl+Shift+P)), start typing the Python: Create Environment command to search, and then select the command.The command presents a list of environment types, Venv or Conda. For this example, select Venv.The command then presents a list of interpreters that can be used for your project. Select the interpreter you installed at the beginning of the tutorial.After selecting the interpreter, a notification will show the progress of the environment creation and the environment folder (/.venv) will appear in your workspace.Ensure your new environment is selected by using the Python: Select Interpreter command from the Command Palette.Note: For additional information about virtual environments, or if you run into an error in the environment creation process, see Environments.Create a Python source code fileFrom the File Explorer toolbar, select the New File button on the hello folder:Name the file hello.py, and VS Code will automatically open it in the editor:By using the .py file extension, you tell VS Code to interpret this file as a Python program, so that it evaluates the contents with the Python extension and the selected interpreter.Note: The File Explorer toolbar also allows you to create folders within your workspace to better organize your code. You can use the New folder button to quickly create a folder.Now that you have a code file in your Workspace, enter the following source code in hello.py:msg = "Roll a dice!"print(msg)When you start typing print, notice how IntelliSense presents auto-completion options.IntelliSense and auto-completions work for standard Python modules as well as other packages you've installed into the environment of the selected Python interpreter. It also provides completions for methods available on object types. For example, because the msg variable contains a string, IntelliSense provides string methods when you type msg.:Finally, save the file (S (Windows, Linux Ctrl+S)). At this point, you're ready to run your first Python file in VS Code.For full details on editing, formatting, and refactoring, see Editing code. The Python extension also has full support for Linting.Run Python codeClick the Run Python File play button in the top-right side of the editor.The button opens a terminal panel in which your Python interpreter is automatically activated, then runs python3 hello.py (macOS/Linux) or python hello.py (Windows):There are three other ways you can run Python code within VS Code:Right-click anywhere in the editor window and select Run Python > Run Python File in Terminal (which saves the file automatically):Select one or more lines, then press Shift+Enter or right-click and select Run Python > Run Selection/Line in Python Terminal. Alternatively, you can activate Smart Send using Shift+Enter without a selection and the Python extension will send the smallest runnable block of code near where your cursor is placed to the terminal. This command is convenient for testing just a part of a file.Note: If you prefer to send code at the particular line your cursor is placed, you can turn off Smart Send by setting python.REPL.enableREPLSmartSend : "false" in your User settings.From the Command Palette (P (Windows, Linux Ctrl+Shift+P)), select the Python: Start Terminal REPL command to open a REPL terminal (notated by >>>) for the currently selected Python interpreter. In the REPL, you can then enter and run lines of code one at a time.Congrats, you just ran your first Python code in Visual Studio Code!Configure and run the debuggerLet's now try debugging our Python program. Debugging support is provided by the Python Debugger extension, which is automatically installed with the Python extension. To ensure it has been installed correctly, open the Extensions view (X (Windows, Linux Ctrl+Shift+X)) and search for @installed python debugger. You should see the Python Debugger extension listed in the results.Next, set a breakpoint on line 2 of hello.py by placing the cursor on the print call and pressing F9. Alternately, click in the editor's left gutter, next to the line numbers. When you set a breakpoint, a red circle appears in the gutter.Next, to initialize the debugger, press F5. Since this is your first time debugging this file, a configuration menu will open from the Command Palette allowing you to select the type of debug configuration you would like for the opened file.Note: VS Code uses JSON files for all of its various configurations; launch.json is the standard name for a file containing debugging configurations.Select Python File, which is the configuration that runs the current file shown in the editor using the currently selected Python interpreter.The debugger will start, and then stop at the first line of the file breakpoint. The current line is indicated with a yellow arrow in the left margin. If you examine the Local variables window at this point, you can see that the msg variable appears in the Local pane.A debug toolbar appears along the top with the following commands from left to right: continue (F5), step over (F10), step into (F11), step out (F11 (Windows, Linux Shift+F11)), restart (F5 (Windows, Linux Ctrl+Shift+F5)), and stop (F5 (Windows, Linux Shift+F5)).The Status Bar also changes color (orange in many themes) to indicate that you're in debug mode. The Python Debug Console also appears automatically in the lower right panel to show the commands being run, along with the program output.To continue running the program, select the continue command on the debug toolbar (F5). The debugger runs the program to the end.Tip Debugging information can also be seen by hovering over code, such as variables. In the case of msg, hovering over the variable will display the string Roll a dice! in a box above the variable.You can also work with variables in the Debug Console (If you don't see it, select Debug Console in the lower right area of VS Code, or select it from the ... menu.) Then try entering the following lines, one by one, at the > prompt at the bottom of the console:msgmsg.capitalize()msg.split()Select the blue Continue button on the toolbar again (or press F5) to run the program to completion. "Roll a dice!" appears in the Python Debug Console if you switch back to it, and VS Code exits debugging mode once the program is complete.If you restart the debugger, the debugger again stops on the first breakpoint.To stop running a program before it's complete, use the red square stop button on the debug toolbar (F5 (Windows, Linux Shift+F5)), or use the Run > Stop debugging menu command.For full details, see Debugging configurations, which includes notes on how to use a specific Python interpreter for debugging.Tip: Use Logpoints instead of print statements: Developers often litter source code with print statements to quickly inspect variables without necessarily stepping through each line of code in a debugger. In VS Code, you can instead use Logpoints. A Logpoint is like a breakpoint except that it logs a message to the console and doesn't stop the program. For more information, see Logpoints in the main VS Code debugging article.Install and use packagesLet's build upon the previous example by using packages.In Python, packages are how you obtain any number of useful code libraries, typically from PyPI, that provide additional functionality to your program. For this example, you use the numpy package to generate a random number.Return to the Explorer view (the top-most icon on the left side, which shows files), open hello.py, and paste in the following source code:import numpy as npmsg = "Roll a dice!"print(msg)print(np.random.randint(1,9))Tip: If you enter the above code by hand, you may find that auto-completions change the names after the as keywords when you press Enter at the end of a line. To avoid this, type a space, then Enter.Next, run the file in the debugger using the "Python: Current file" configuration as described in the last section.You should see the message, "ModuleNotFoundError: No module named 'numpy'". This message indicates that the required package isn't available in your interpreter. If you're using an Anaconda distribution or have previously installed the numpy package you may not see this message.To install the numpy package, stop the debugger and use the Command Palette to run Terminal: Create New Terminal (` (Windows, Linux Ctrl+Shift+`)). This command opens a command prompt for your selected interpreter.To install the required packages in your virtual environment, enter the following commands as appropriate for your operating system:Install the packages# Don't use with Anaconda distributions because they include matplotlib already.# macOSpython3 -m pip install numpy# Windows (may require elevation)py -m pip install numpy# Linux (Debian)apt-get install python3-tkpython3 -m pip install numpyNow, rerun the program, with or without the debugger, to view the output!Managing dependencies across environmentsWhen working on Python projects, its essential to manage your dependencies effectively. One useful tip is to use the pip freeze > requirements.txt command. This command helps you create a requirements.txt file

that lists all the packages installed in your virtual environment. This file can then be used to recreate the same environment elsewhere.Follow these steps to create a requirements.txt file:Activate your virtual environment, if you havent already.source venv/bin/activate # On macOS/Linux.\venv\Scripts\activate # On WindowsGenerate the requirements.txt file.pip freeze > requirements.txtYou can now use the newly generated requirements.txt file to install dependencies in another environment. Furthermore, you can continue to add dependencies to it as your project may grow in complexity.pip install -r requirements.txtBy following these steps, you ensure that your project dependencies are consistent across different environments, making it easier to collaborate with others and deploy your project.Congrats on completing the Python tutorial! During the course of this tutorial, you learned how to create a Python project, create a virtual environment, run and debug your Python code, and install Python packages. Explore additional resources to learn how to get the most out of Python in Visual Studio Code!Next stepsTo learn how to build web apps with popular Python web frameworks, see the following tutorials: There is then much more to explore with Python in Visual Studio Code: 07/09/2025 Creating a Python project in Visual Studio Code (VS Code) is a straightforward process. Heres a step-by-step guide to help you get started: Install VS Code:If you havent already, download and install Visual Studio Code from the official website (Download Visual Studio Code Mac, Linux, Windows). Install Python:Ensure you have Python installed on your system. You can download it from the official Python website. Make sure to add Python to your PATH during installation (Download Python | Python.org). Install Python Extension for VS Code:Open VS Code and go to the Extensions view by clicking on the Extensions icon in the Activity Bar on the side of the window or by pressing Ctrl+Shift+X. Search for Python and install the extension provided by Microsoft. Create a New Project Folder:Create a new folder on your computer where you want to store your Python project. You can name it something relevant to your project. Open the Project Folder in VS Code:Open VS Code, then go to File > Open Folder and select the folder you just created. Create a Virtual Environment:It is recommended to create virtual environment to isolate project dependencies. Open the terminal in VS Code by going to View > Terminal or pressing Ctrl+` . In the terminal, navigate to your project folder and create a virtual environment by running: python -m venv venv Activate the virtual environment: On Windows:.\venv\Scripts\activate On macOS/Linux:source venv/bin/activate Install dependencies:If you need any Python packages for your project, you can install them using pippip install package-name Create Requirement file:Pip freeze is a command used in Python tofreeze the current state of a virtual environment. This command creates a list of all the installed packages in the virtual environment, along with their versions. This list can be used later to recreate the same virtual environment on another machine. Always activate your virtual environment before using pip freeze. Remember to update therequirements.txtfile when new packages are installedpip freeze > requirements.txt To install requirements in new virtual environment, use command pip install -r requirements.txt Create a Python File:In your project folder, create a new Python file by going to File > New File or pressing Ctrl+N. Save the file with a .py extension, for example, main.py. Select the Python Interpreter:Click on the Python version displayed in the bottom-left corner of VS Code. Select the interpreter from the list that corresponds to your virtual environment (if you created one) or your system Python installation. Write Your Python Code:Open your main.py file and start writing your Python code. For example:print(Hello, VS Code!) Run Your Python Code:You can run your Python code by opening the terminal and typing:python main.pyAlternatively, you can run the code directly in VS Code by right-clicking inside the editor and selecting Run Python File in Terminal. Conclusion:You now have a basic Python project set up in VS Code. As you continue to develop your project, you can take advantage of VS Codes features like IntelliSense, linting, and version control integration to enhance your development experience. Setting up Python and its environment can be confusing for beginners, but its simpler than it looks! This guide will walk you through the steps needed to install Python, set up a virtual environment, and install the necessary libraries. By the end, youll be ready to run any Python script easily. First, go to the official Python website and download the latest Python version (3.x) for your operating system. Then, run the installer and follow the instructions. Make sure to check the box that says Add Python to PATH during the installation process. This will allow you to run Python from the command line and make it much easier to manage later. Open a terminal on your PC (Command Prompt, PowerShell, or VS Code terminal). Type python --version and press Enter. If Python is installed, the version number will be displayed like this: You may have a newer version, so it will display your version. However, if it doesnt display the version, it means you did something wrong, so go over the steps again. Why VS Code? VS Code is a popular and lightweight code editor that is perfect for writing and running Python scripts. To install VS code, go to the VS Code website, and download the installer for your operating system (Windows, macOS, or Linux). Then, run the installer and follow the instructions. Make sure the Add to PATH option is selected during installation, as we did when installing Python. Finally, open VS Code, go to Extensions on the left sidebar and install Microsofts Python Extension. This one: A virtual environment helps you manage dependencies for Python projects without impacting your systems setup or causing conflicts between installed libraries. In VS Code, open the folder where your Python script will be saved and open a new terminal by clicking ctrl + shift + ` Then, run the command: python -m venv venv This creates a folder named venv in your project directory. Now, youll have to activate it to use it by running this command: Windows: venv\Scripts\activate macOS/Linux: source venv/bin/activate Once activated, the terminal will show (venv) before the command prompt. If you want to deactivate the virtual environment just type deactivate and press Enter. Libraries are collections of pre-written code that provide additional functionality, allowing you to perform specific tasks or solve problems without writing the code from scratch. For example, you can use the requests library for web requests or pandas for data analysis. Installing any libraries is very simple; all you have to do is use the pip command in this way: pip install requests Here are some other functionalities you can do with libraries: See all installed libraries: pip list Save the list of installed libraries to a file: pip freeze > requirements.txt Install a list of libraries from a text file: pip install -r requirements.txtOpen VS Code and create a new file (e.g., script.py). Write or paste your Python code into the file and save it. Open the terminal ctrl + shift + ` Ensure the virtual environment is activated. Run the script: python script.py By following these steps, youll be ready to set up Python, manage dependencies, and execute Python scripts. Create Python Project in VS Code: Step-by-Step Guide for Beginners ProjectHey there, aspiring tech wizards and Python enthusiasts! Today, we are diving into the exciting world of creating a Python project in the ever-popular Visual Studio Code (VS Code) environment. Buckle up, grab your coding hats, and lets embark on this adventure together! So, first things first! You gotta have VS Code on your system to rock this Python party. If you havent downloaded this gem yet, what on earth are you waiting for? Go ahead, hit that download button, and lets get this show on the road! Next up, its time to make VS Code Python-friendly. Install the Python extension in VS Code so it can understand all your Pythonic dreams and turn them into reality. A few clicks here and there, and voil, youre all set to conquer the Python universe! Ah, the wonderful world of virtual environments! Trust me, these are your best buddies when it comes to keeping your Python projects neat and tidy. Set up a virtual environment like a pro and watch how it works its magic to keep things organized. Say goodbye to dependency chaos! Now, lets work our magic in VS Code by creating some Python files and folders to house our brilliant code creations. Organize your project structure in a way that even Marie Kondo would approve of! Clean code sparks joy, after all! Here comes the thrilling part writing your very first Python program! The excitement, the nerves, the endless possibilities its all part of the coding journey. Let your creativity flow as you craft lines of code that will soon come to life. Embrace the Pythonic vibes, my friend! Once your masterpiece is ready, its showtime! Hit that run button in VS Code and watch your Python code come alive. Whether its a simple Hello, World! or a complex algorithm, seeing your code run successfully is always a satisfying moment. Enjoy the magic of coding in action! Ah, dependencies the silent heroes of every Python project. Learn the art of using pip to install essential packages and libraries to supercharge your Python projects. Need a package? Say no more pip has your back! To keep your project consistent and hassle-free, dive into the world of requirements.txt files. Listing down all your project dependencies here will ensure smooth sailing when sharing your project with others or deploying it. Think of it as your projects recipe book! Bugs beware! Its time to unleash the power of the debugger in VS Code. Track down those pesky bugs, set breakpoints, and step through your code like a detective solving a mystery. Debugging might just become your new favorite pastime! Last but not least, testing the unsung hero of reliable code. Dive into the world of writing tests in Python to ensure your code behaves as expected. From unit tests to integration tests, testing is your safety net in the wild world of coding. Embrace the process and code fearlessly! In closing, creating a Python project in VS Code is not just about writing code; its about embracing the journey of learning, exploring, and pushing your boundaries. So, grab that cup of coffee, don your coding cape, and embark on this Python adventure with zest and zeal! Thank you for joining me on this coding escapade! Until next time, happy coding and may your Python projects shine brighter than a supernova in the coding galaxy! #HappyCoding Overall, creating a Python project in VS Code is an exhilarating journey filled with coding triumphs, debugging dramas, and testing adventures. Dive in, explore, and let your Pythonic creativity soar to new heights! Copy Code Copied Use a different Browser import os# Step 1: Create a folder where your project will resideproject_folder = 'MyPythonProject'if not os.path.exists(project_folder): os.makedirs(project_folder)# Step 2: Create a Python file in that folderpython_filename = 'main.py'with open(os.path.join(project_folder, python_filename), 'w') as file: file.write('print('Hello, VS Code!')')# Step 3: Create a virtual environmentos.system(f'python -m venv {os.path.join(project_folder, 'venv')}')# Step 4: Write a requirements.txt (though empty here, typically would include needed packages)requirements_file = 'requirements.txt'with open(os.path.join(project_folder, requirements_file), 'w') as file: file.write('# Add your dependencies here, e.g.,# numpy==1.18.5')# Step 5: Add a .gitignore filegitignore_content = '''# Byte-compiled / optimized / DLL files__pycache__/*.py[cod]*$py.class'''with open(os.path.join(project_folder, '.gitignore'), 'w') as file: file.write(gitignore_content) No standard output is expected as this script sets up a project structure without producing console output unless an error occurs. The provided Python script automates the setup of a new Python project in Visual Studio Code for beginners. Heres a breakdown of its operations:Create Project Directory: The script starts by defining a project folder (MyPythonProject). It checks whether this folder exists, and if not, it creates it using os.makedirs.Add Main Python file: Within the newly created project folder, the script generates a main Python file (main.py). It opens this file in write mode and inserts a simple print statement to exemplify a basic operation.Setup Virtual Environment: To encapsulate project dependencies, a virtual environment named venv is created inside the project folder using the Python venv module.Create requirements.txt: Its common practice to list project dependencies in a requirements.txt file. The script adds this file with a placeholder for dependencies, ready to be filled as needed.Add .gitignore File: Finally, the script creates a .gitignore file tailored for Python projects (ignoring bytecode and similar files). This file is crucial for version control, where it ensures that unnecessary files are not tracked by Git.Overall, this script forms a foundational template that can be expanded based on the specific needs of any Python project. Creating Python projects in VS Code is essential for beginners as it provides a user-friendly interface, powerful tools, and seamless integration with the Python programming language. It helps in organizing code, debugging, and collaborating effectively on projects. To start a new Python project in VS Code, you can open the editor, create a new folder for your project, set up a virtual environment, install the necessary Python extensions, and begin writing your Python code. VS Code offers a wide range of features for Python development, including IntelliSense for code completion, debugging capabilities, version control integration, built-in terminal, and an extensive marketplace for extensions to enhance your workflow. You can debug your Python code in VS Code by setting breakpoints in your code, running the debugger, stepping through code, inspecting variables, and monitoring the execution flow to identify and fix issues effectively. Yes, VS Code allows for seamless collaboration on Python projects through features like Live Share, which enables real-time collaboration, sharing of code snippets, and joint debugging sessions with team members, regardless of their physical location. Several popular extensions are recommended for Python projects in VS Code, such as Python, Pylance, Python Docstring Generator, GitLens, and Python Test Explorer, to boost productivity, streamline workflows, and enhance the development experience. You can manage dependencies in your Python project within VS Code by utilizing tools like pip, virtual environments, requirements.txt files, and the Python extensions package management capabilities to install, update, and organize project dependencies efficiently. Yes, you can deploy Python projects from VS Code to production environments by configuring deployment pipelines, integrating with cloud services, containerizing applications, and leveraging extensions like Azure App Service, Docker, or Heroku for seamless deployment and scaling. There are numerous online resources, tutorials, documentation, and community forums available to help you enhance your skills in creating Python projects in VS Code. Websites like Real Python, Stack Overflow, VS Code documentation, and YouTube tutorials offer valuable insights, tips, and guidance for improving your Python development proficiency. Ensuring the security of your Python projects in VS Code involves practices like using secure coding principles, keeping dependencies up to date, implementing authentication mechanisms, encrypting sensitive data, performing regular security audits, and following best practices for secure development and deployment of applications.

Exercícios sobre as cruzadas 7o ano. Exercícios sobre as cruzadas 7o ano com gabarito. As cruzadas historia 7 ano. Cruzadas 7 ano.