# Python notes

## [Setting up VSCode for Python programming](https://www.youtube.com/watch?v=W--_EOzdTHk)

https://code.visualstudio.com/docs/python/python-tutorial

Extension:

AREPL automatically evaluates python code in real-time as you type. (Little orange guy top right)

KITE autocomplete while coding in Python & JavaScript

Requires install of copilot and linters post install. Pip install-U pylint

Note: [See Prettier does not work with Python](https://dev.to/eegli/quick-guide-to-python-formatting-in-vs-code-2040)

[**Setting Up PEP8 and Pylint on VS Code**](https://dev.to/j0nimost/setting-up-pep8-and-pylint-on-vs-code-34h)

[PEP8](https://www.python.org/dev/peps/pep-0008/) defines Python coding standards; from variable declaration to formatting of classes. It has it all, this allows you to nicely format your python code.

## What Is PEP8?

PEP 8 exists to improve the readability of Python code, helps you collaborate well with others and makes it easier for them to understand the code you write.

[Pylint](https://www.pylint.org/) checks whether we follow PEP8 standards and returns errors where we fail to follow. Furthermore, this tool also does error checking due to syntax errors.

## What Is Linting?

Linting is the automated checking of your source code for programmatic and stylistic errors. This is done by using a lint tool (otherwise known as linter). A lint tool is a basic static code analyzer.

Settings: **<ctrl> <,>**

Editor:Format On Type

Editor:Format On Save

Python:formatting:Provider [autopep8 | black | yapf]

Command Palette: **<ctrl> <shift> <p>**

Pyhton: Select Linter: >>>> pylint

Python: Enable Linting

Python: Run Linter

Also see: [Editing Python in Visual Studio Code](https://code.visualstudio.com/docs/python/editing)

[Start using Kite](https://www.kite.com/welcome/?id=a8a8b915-9462-4892-bb3f-19c0d056c6ed)

[REST Client](https://www.youtube.com/watch?v=ezhugY8TJDU)

## [Getting Started with Python in Visual Studio Code](https://www.youtube.com/watch?v=7EXd4_ttIuw)

The **Python Interactive** window can be used as a standalone console with arbitrary code (with or without code cells #%%). To use the window as a console, open it with the **Jupyter: Create Interactive Window** command from the Command Palette. You can then type in code, using Enter to go to a new line and Shift+Enter to run the code.

To use the window with a file, use the **Jupyter: Create Interactive Window** command from the Command Palette.

https://code.visualstudio.com/docs/python/jupyter-support-py

Create a Python virtual environment:

py -3 -m venv folder\_name

python3 -m venv folder\_name

Note: to put terminal in virtual environment execute appropriate activate shell script e.g., activate.bat

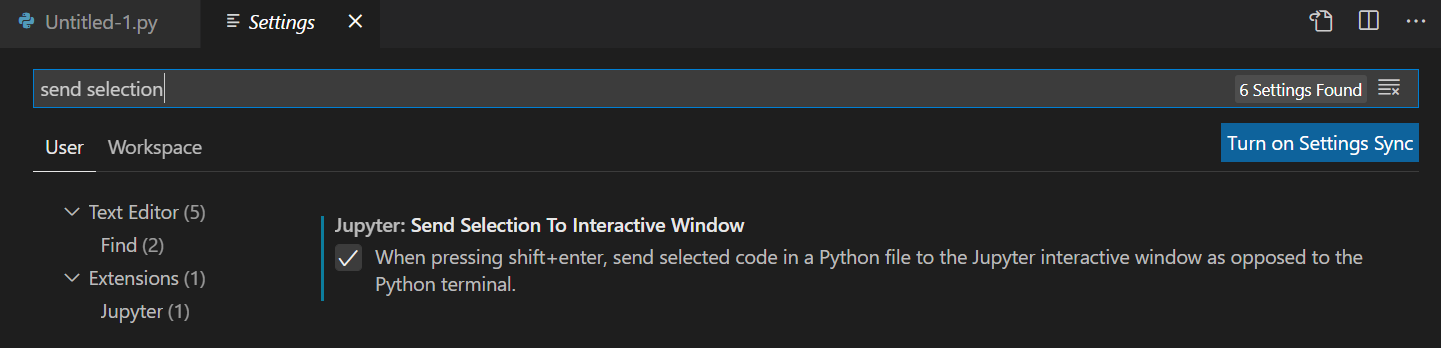
Command palette

From the Command Palette (Ctrl+Shift+P), select the **Python: Start REPL** command to open a REPL terminal (default, see \*Global settings below to change to run in interactive window rather than a terminal) for the currently selected Python interpreter. In the REPL, you can then enter and run lines of code one at a time.

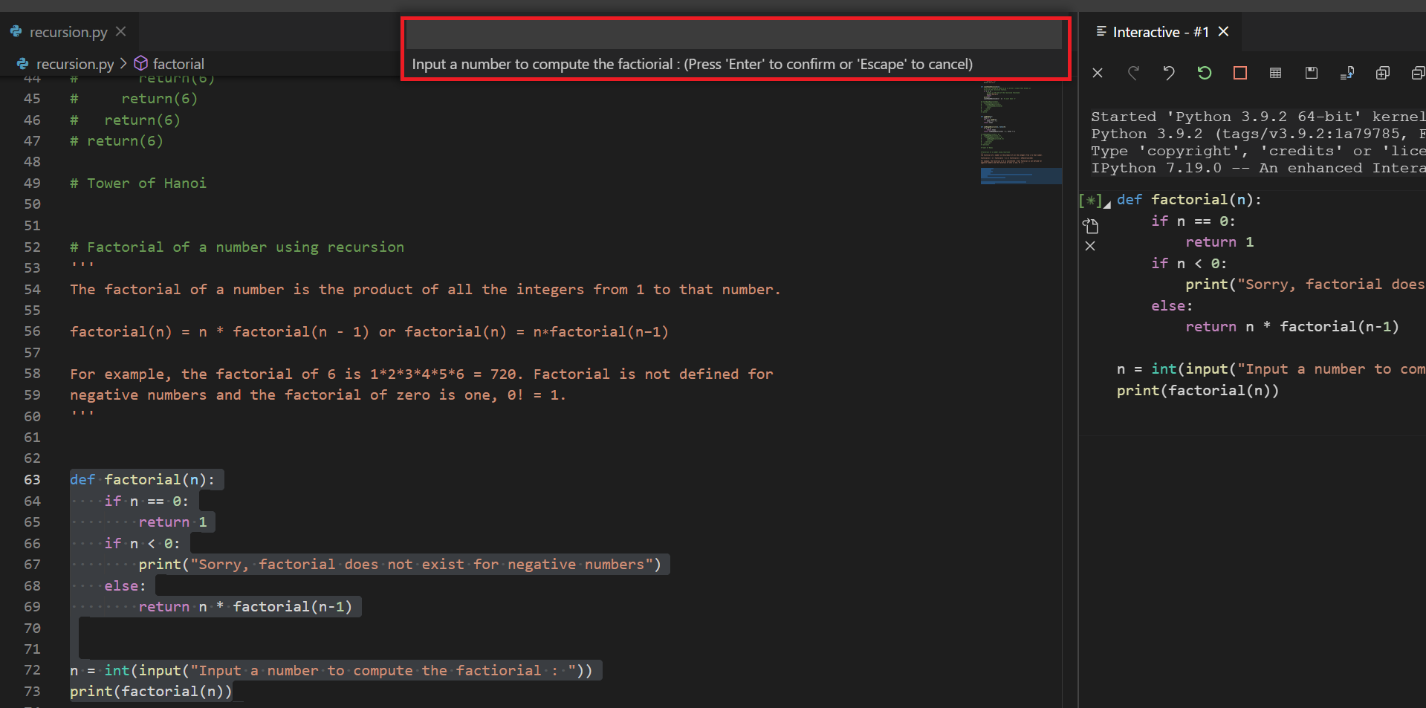
To run selected code <shift><enter>. The code will either run in the terminal or Jupyter interactive window; dependent on the global settings.

\*Global settngs:

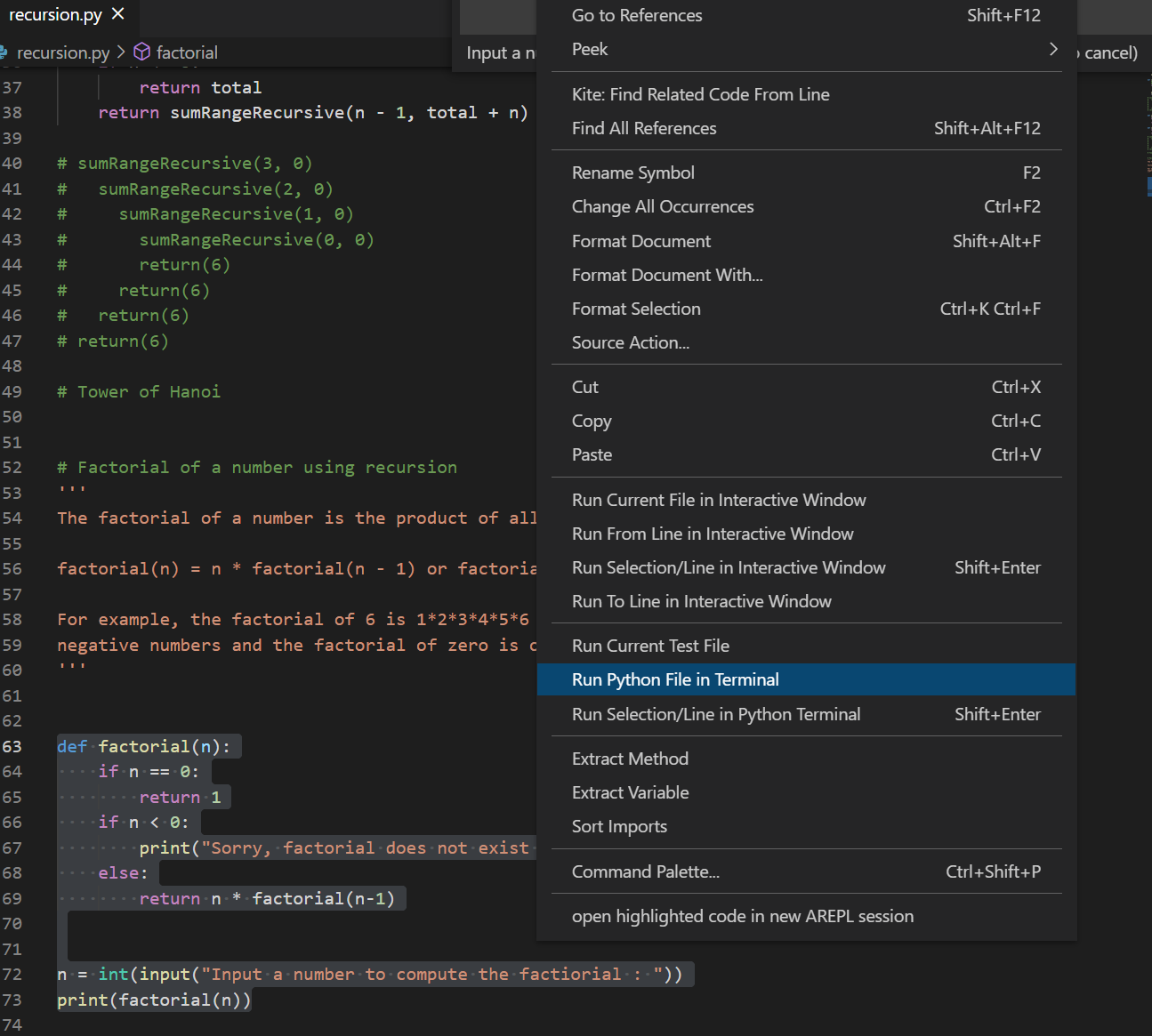
open <CRTL> <,>



Note: If your program requires input whilst in Jupyter interactive mode, make sure you look in the right place:



You can temp override the above setting by highlighting code and then mouse right click and select ‘Run Ppython File in Terminal’.



What is a **SCALAR**

The term "scalar" comes from [linear algebra](http://en.wikipedia.org/wiki/Scalar_%28mathematics%29), where it is used to differentiate a single number from a vector or matrix. The meaning in computing is similar. It distinguishes a single value like an integer or float from a data structure like an array.

# **[Introducing Native Notebooks](https://jupyter-notebook-beginner-guide.readthedocs.io/en/latest/what_is_jupyter.html)**

<https://github.com/microsoft/vscode-jupyter/wiki/Introducing-Native-Notebooks>

# [1. What is the Jupyter Notebook?](https://jupyter-notebook-beginner-guide.readthedocs.io/en/latest/what_is_jupyter.html#id4)

In this page briefly introduce the main components of the Jupyter Notebook environment. For a more complete overview see [References](https://jupyter-notebook-beginner-guide.readthedocs.io/en/latest/what_is_jupyter.html#references).

**Contents**

* [What is the Jupyter Notebook?](https://jupyter-notebook-beginner-guide.readthedocs.io/en/latest/what_is_jupyter.html#what-is-the-jupyter-notebook)
  + [Notebook document](https://jupyter-notebook-beginner-guide.readthedocs.io/en/latest/what_is_jupyter.html#notebook-document)
  + [Jupyter Notebook App](https://jupyter-notebook-beginner-guide.readthedocs.io/en/latest/what_is_jupyter.html#jupyter-notebook-app)
  + [kernel](https://jupyter-notebook-beginner-guide.readthedocs.io/en/latest/what_is_jupyter.html#kernel)
  + [Notebook Dashboard](https://jupyter-notebook-beginner-guide.readthedocs.io/en/latest/what_is_jupyter.html#notebook-dashboard)
  + [References](https://jupyter-notebook-beginner-guide.readthedocs.io/en/latest/what_is_jupyter.html#references)

## [1.1. Notebook document](https://jupyter-notebook-beginner-guide.readthedocs.io/en/latest/what_is_jupyter.html#id5)

Notebook documents (or “notebooks”, all lower case) are documents produced by the [Jupyter Notebook App](https://jupyter-notebook-beginner-guide.readthedocs.io/en/latest/what_is_jupyter.html#notebook-app), which contain both computer code (e.g. python) and rich text elements (paragraph, equations, figures, links, etc…). Notebook documents are both human-readable documents containing the analysis description and the results (figures, tables, etc..) as well as executable documents which can be run to perform data analysis.

**References**: Notebook documents [in the project homepage](http://ipython.org/notebook.html#notebook-documents) and [in the official docs](http://jupyter-notebook.readthedocs.org/en/latest/notebook.html#notebook-documents).

## [1.2. Jupyter Notebook App](https://jupyter-notebook-beginner-guide.readthedocs.io/en/latest/what_is_jupyter.html#id6)

The Jupyter Notebook App is a server-client application that allows editing and running [notebook documents](https://jupyter-notebook-beginner-guide.readthedocs.io/en/latest/what_is_jupyter.html#notebook-document) via a web browser. The Jupyter Notebook App can be executed on a local desktop requiring no internet access (as described in this document) or can be installed on a remote server and accessed through the internet.

In addition to displaying/editing/running notebook documents, the Jupyter Notebook App has a “Dashboard” ([Notebook Dashboard](https://jupyter-notebook-beginner-guide.readthedocs.io/en/latest/what_is_jupyter.html#dashboard)), a “control panel” showing local files and allowing to open notebook documents or shutting down their [kernels](https://jupyter-notebook-beginner-guide.readthedocs.io/en/latest/what_is_jupyter.html#kernel).

**References**: Jupyter Notebook App [in the project homepage](http://ipython.org/notebook.html) and [in the official docs](http://jupyter-notebook.readthedocs.org/).

# [**6. Expressions**](https://docs.python.org/3/reference/expressions.html)

## **6.8. Shifting operations**

The shifting operations have lower priority than the arithmetic operations:

**shift\_expr** ::= [a\_expr](https://docs.python.org/3/reference/expressions.html?highlight=ternary#grammar-token-a-expr) | [shift\_expr](https://docs.python.org/3/reference/expressions.html?highlight=ternary#grammar-token-shift-expr) ("<<" | ">>") [a\_expr](https://docs.python.org/3/reference/expressions.html?highlight=ternary#grammar-token-a-expr)

These operators accept integers as arguments. They shift the first argument to the left or right by the number of bits given by the second argument.

A right shift by n bits is defined as floor division by pow(2,n). A left shift by n bits is defined as multiplication with pow(2,n).

## **6.9. Binary bitwise operations**

Each of the three bitwise operations has a different priority level:

**and\_expr** ::= [shift\_expr](https://docs.python.org/3/reference/expressions.html?highlight=ternary#grammar-token-shift-expr) | [and\_expr](https://docs.python.org/3/reference/expressions.html?highlight=ternary#grammar-token-and-expr) "&" [shift\_expr](https://docs.python.org/3/reference/expressions.html?highlight=ternary#grammar-token-shift-expr)

**xor\_expr** ::= [and\_expr](https://docs.python.org/3/reference/expressions.html?highlight=ternary#grammar-token-and-expr) | [xor\_expr](https://docs.python.org/3/reference/expressions.html?highlight=ternary#grammar-token-xor-expr) "^" [and\_expr](https://docs.python.org/3/reference/expressions.html?highlight=ternary#grammar-token-and-expr)

**or\_expr**  ::= [xor\_expr](https://docs.python.org/3/reference/expressions.html?highlight=ternary#grammar-token-xor-expr) | [or\_expr](https://docs.python.org/3/reference/expressions.html?highlight=ternary#grammar-token-or-expr) "|" [xor\_expr](https://docs.python.org/3/reference/expressions.html?highlight=ternary#grammar-token-xor-expr)

The & operator yields the bitwise AND of its arguments, which must be integers.

The ^ operator yields the bitwise XOR (exclusive OR) of its arguments, which must be integers.

The | operator yields the bitwise (inclusive) OR of its arguments, which must be integers.

## **6.12. Assignment expressions**

**assignment\_expression** ::= [[identifier](https://docs.python.org/3/reference/lexical_analysis.html#grammar-token-identifier) ":="] [expression](https://docs.python.org/3/reference/expressions.html?highlight=ternary#grammar-token-expression)

An assignment expression (sometimes also called a “named expression” or “walrus”) assigns an [expression](https://docs.python.org/3/reference/expressions.html?highlight=ternary#grammar-token-expression) to an [identifier](https://docs.python.org/3/reference/lexical_analysis.html#grammar-token-identifier), while also returning the value of the [expression](https://docs.python.org/3/reference/expressions.html?highlight=ternary#grammar-token-expression).

One common use case is when handling matched regular expressions:

**if** matching := pattern.search(data):

do\_something(matching)

Or, when processing a file stream in chunks:

**while** chunk := file.read(9000):

process(chunk)

*New in version 3.8:*See [**PEP 572**](https://www.python.org/dev/peps/pep-0572) for more details about assignment expressions.

## **6.13. Conditional expressions**

## **6.13. Conditional expressions**

**conditional\_expression** ::= [or\_test](https://docs.python.org/3/reference/expressions.html?highlight=ternary#grammar-token-or-test) ["if" [or\_test](https://docs.python.org/3/reference/expressions.html?highlight=ternary#grammar-token-or-test) "else" [expression](https://docs.python.org/3/reference/expressions.html?highlight=ternary#grammar-token-expression)]

**expression**  ::= [conditional\_expression](https://docs.python.org/3/reference/expressions.html?highlight=ternary#grammar-token-conditional-expression) | [lambda\_expr](https://docs.python.org/3/reference/expressions.html?highlight=ternary#grammar-token-lambda-expr)

**expression\_nocond**  ::= [or\_test](https://docs.python.org/3/reference/expressions.html?highlight=ternary#grammar-token-or-test) | [lambda\_expr\_nocond](https://docs.python.org/3/reference/expressions.html?highlight=ternary#grammar-token-lambda-expr-nocond)

Conditional expressions (sometimes called a “ternary operator”) have the lowest priority of all Python operations.

The expression x if C else y first evaluates the condition, C rather than x. If C is true, x is evaluated and its value is returned; otherwise, y is evaluated and its value is returned.

See [**PEP 308**](https://www.python.org/dev/peps/pep-0308) for more details about conditional expression

## **6.14. Lambdas**

**lambda\_expr**  ::= "lambda" [[parameter\_list](https://docs.python.org/3/reference/compound_stmts.html#grammar-token-parameter-list)] ":" [expression](https://docs.python.org/3/reference/expressions.html?highlight=ternary#grammar-token-expression)

**lambda\_expr\_nocond** ::= "lambda" [[parameter\_list](https://docs.python.org/3/reference/compound_stmts.html#grammar-token-parameter-list)] ":" [expression\_nocond](https://docs.python.org/3/reference/expressions.html?highlight=ternary#grammar-token-expression-nocond)

Lambda expressions (sometimes called lambda forms) are used to create anonymous functions. The expression lambda parameters: expression yields a function object. The unnamed object behaves like a function object defined with:

def <lambda>(parameters):

return expression

See section [Function definitions](https://docs.python.org/3/reference/compound_stmts.html#function) for the syntax of parameter lists. Note that functions created with lambda expressions cannot contain statements or annotations