Brookhaven™
National Laboratory

Foundations of Scientific Computing

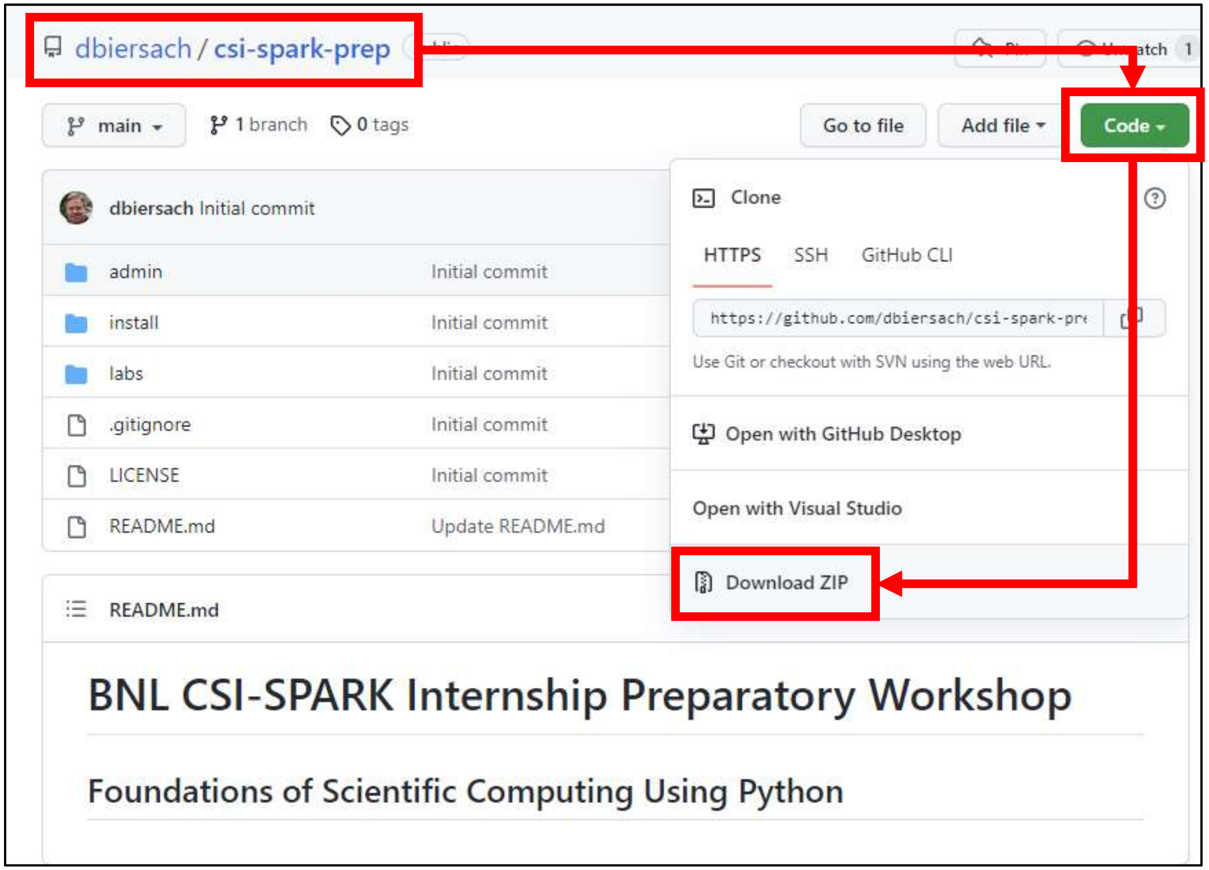
Dave Biersach
dbiersach@bnl.gov

elif _operation == "MIRROR_Z":
mirror_mod.use_x = False
mirror_mod.use_y = False
mirror_mod.use_z = True
#selection at the end -add back the deselected mirror
mirror_ob.select= 1
modifier_ob.select=1
bpy.context.scene.objects.active = modifier_ob
print("Selected" + str(modifier_ob)) # modifier ob is the active

Session 01
Python
Fundamentals

Downloading the Courseware

The courseware can be downloaded as a large single **ZIP** file from https://github.com/dbiersach/csi-spark-prep



dbiersach Initial commit

admin	Initial commit
install	Initial commit
labs	Initial commit
.gitignore	Initial commit
LICENSE	Initial commit
README.md	Update README.md

https://github.com/dbiersach/csi-spark-pre

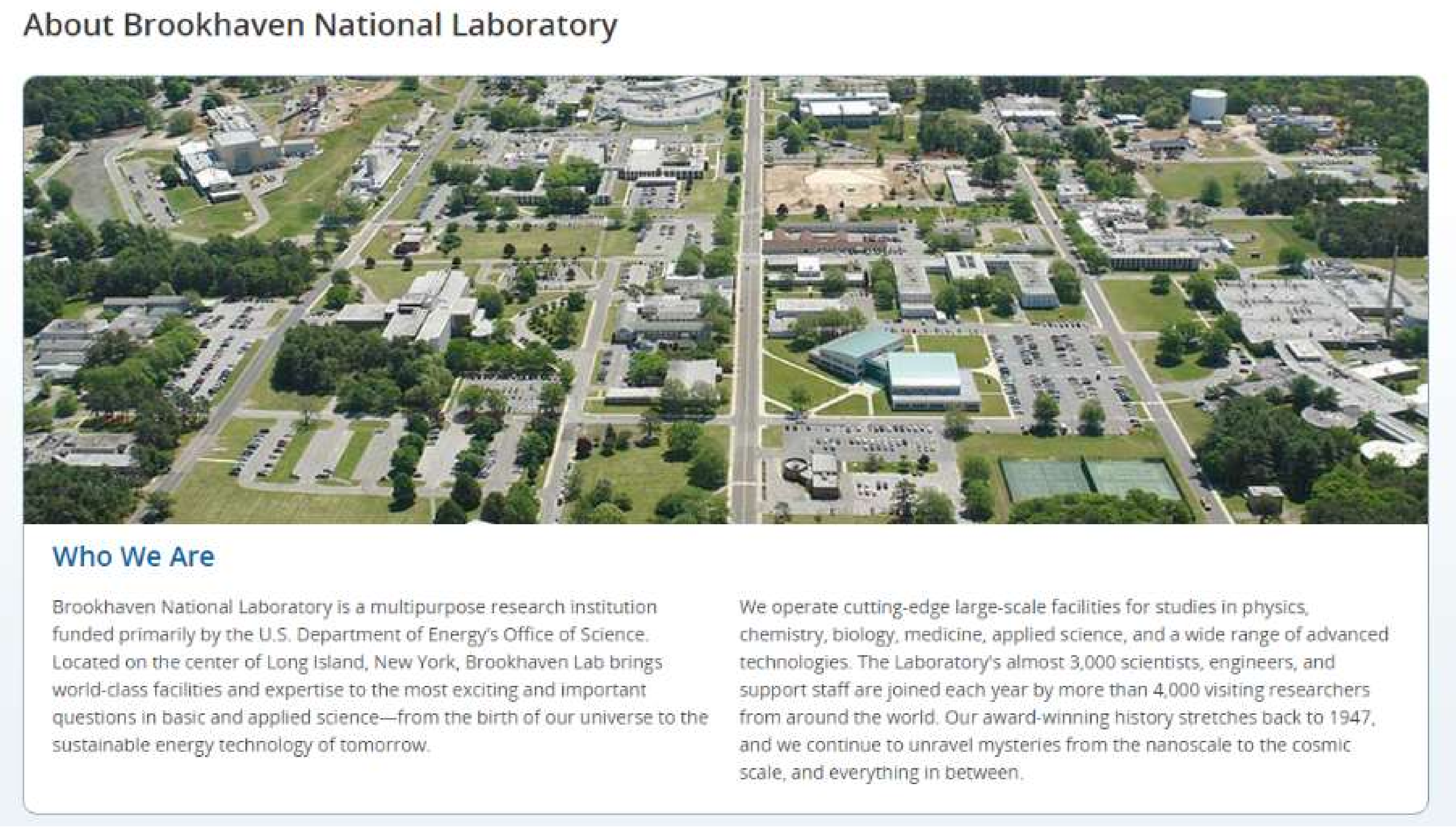
Use Git or checkout with SVN using the web URL.

BNL CSI-SPARK Internship Preparatory Workshop

Foundations of Scientific Computing Using Python

Welcome!

- My name is **Dave Biersach**
- I am a Senior Technology Architect at BNL
- I am a 1989 graduate of the United States Military Academy at West Point, and I served in the Persian Gulf War as a Combat Engineer
- I worked for DARPA developing satellite counter-reconnaissance search algorithms
- I have worked at **Microsoft** & **Pfizer**
- I have been married 30 years, have three adult children, and have mentored students for the past 15 years



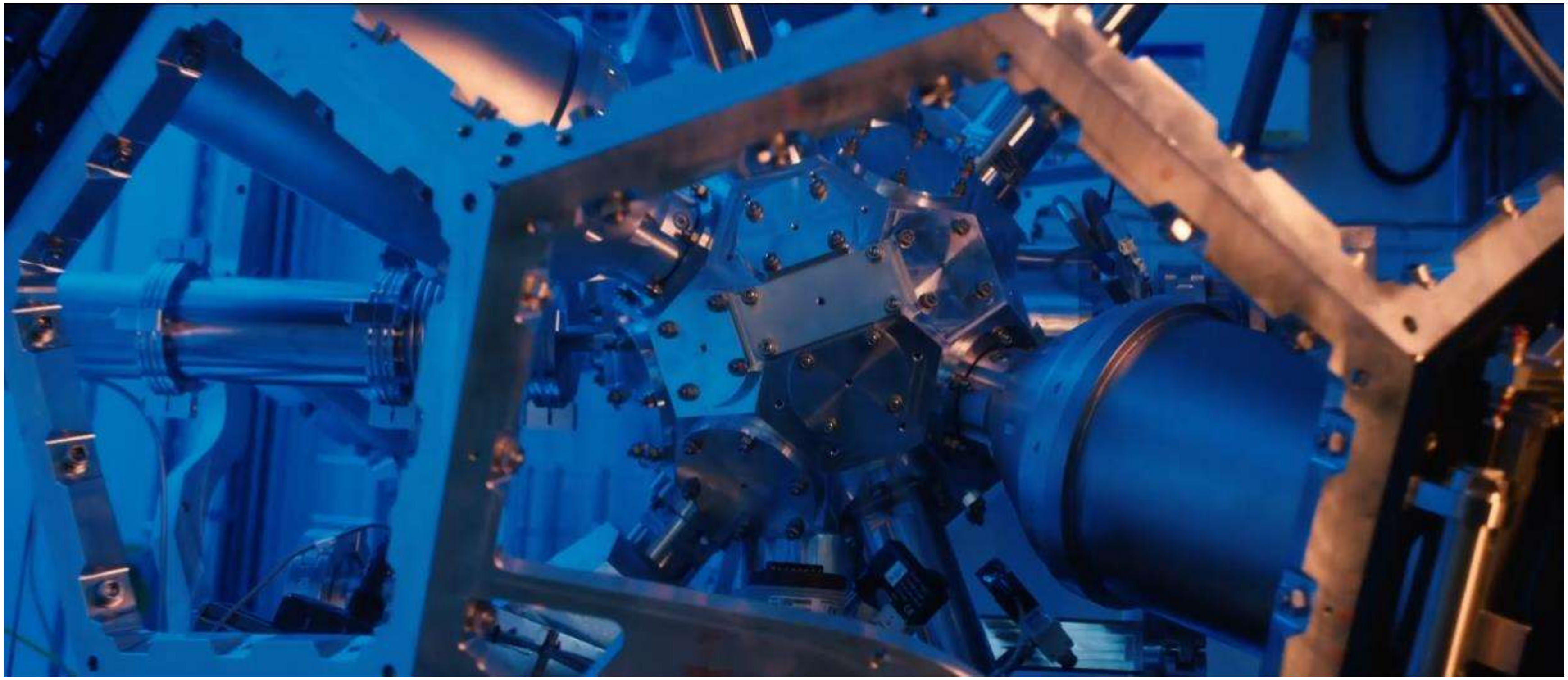
About Brookhaven National Laboratory

Who We Are

Brookhaven National Laboratory is a multipurpose research institution funded primarily by the U.S. Department of Energy's Office of Science. Located on the center of Long Island, New York, Brookhaven Lab brings world-class facilities and expertise to the most exciting and important questions in basic and applied science—from the birth of our universe to the sustainable energy technology of tomorrow.

We operate cutting-edge large-scale facilities for studies in physics, chemistry, biology, medicine, applied science, and a wide range of advanced technologies. The Laboratory's almost 3,000 scientists, engineers, and support staff are joined each year by more than 4,000 visiting researchers from around the world. Our award-winning history stretches back to 1947, and we continue to unravel mysteries from the nanoscale to the cosmic scale, and everything in between.

About Brookhaven National Laboratory



https://www.youtube.com/watch?v=csgXRPV0R3A

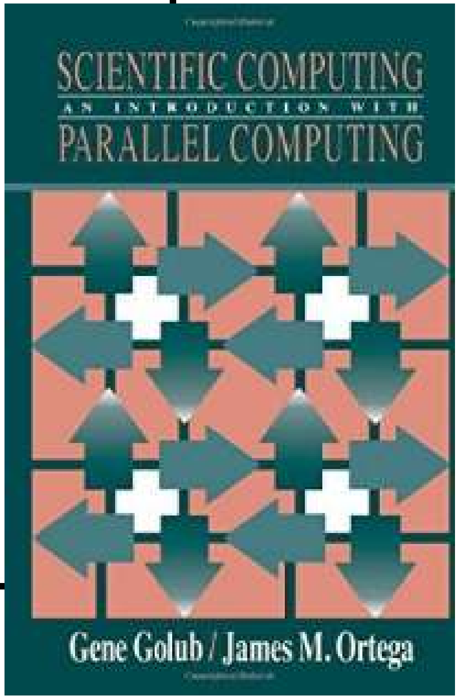
What is Scientific Computing?

Golub and Ortega: *"Scientific computing is the collection of tools, techniques and theories required to solve on a computer mathematical models of problems in science and engineering."*

Or a more narrow definition: *"Development and use of numerical methods and mathematical models to solve real-world problems efficiently on computers."*

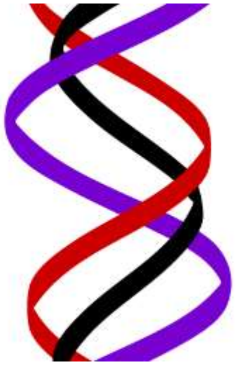
Interdisciplinary field requiring:

- knowledge about the underlying (physical) problem,
- ability to formulate a mathematical model,
- stable & accurate numerical schemes,
- efficient implementation on high performance computers.



What is Scientific Computing?

- Scientific computing problems **cannot be solved** using just a graphing calculator or a spreadsheet program
  - A computer should not be viewed as just another closed-form benchtop instrument with fixed functionality
  - SciComp does not require writing thousands of lines of code to answer problems – complete code usually fits on **one** slide!
- SciComp is **applied** computer science
  - The first name of CompSci is *computer*
  - The first name of SciComp is **science**
  - A **triple helix** of math, science, and computing



SciComp vs CompSci

Scientific Computing

- Probability and Statistics
- Simulation and Modelling
- Data Visualization
- Storing and Analyzing Very Large Datasets
- Parallel & Distributed Algorithms
- Speed and Accuracy Paramount
- Functional Languages
- Open-Ended Problems with Unknown Solutions

Computer Science

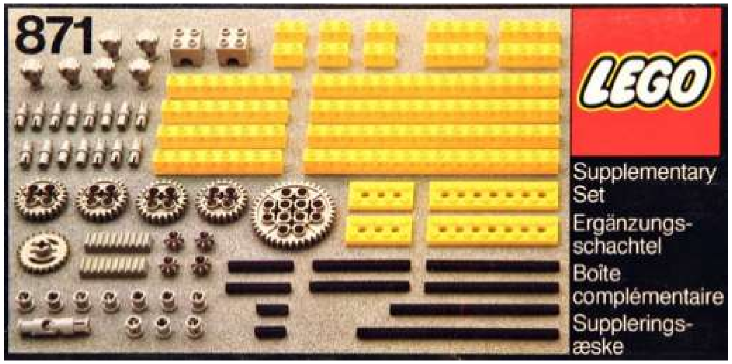
- General Data Structures
- Design Methodologies
- Procedural Languages
- Stand-Alone Programs
- Emphasis on Object-Orientation
- Simple Data Models
- Sequential Algorithms
- Less Graphics Intensive
- Directed Closed-Form Problems with Known Solutions

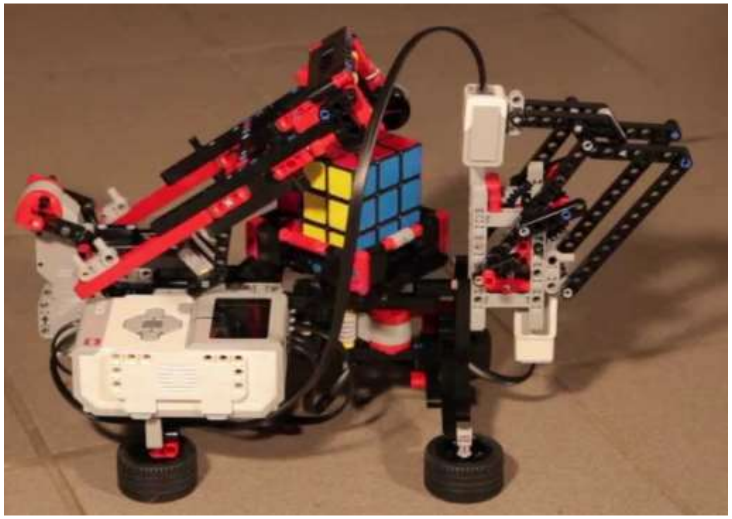
Scientific Computing with Python

- Python is quickly becoming one of the **most heavily used languages** in science projects
- Python runs on all major modern **operating systems** and is completely free and open-source (not vendor controlled)
- Python makes it easy for your code to directly integrate with a large spectrum of available 3rd party software
- Python code runs **consistently** on different platforms and scales well from small IoT devices to large server clusters
- Python benefits from a very active and growing user community that continues to enhance the language

SciComp = **Just Enough** CompSci

- Data **types**: int, float, bool, string
- Data **structures**: lists, arrays, dictionaries, classes, modules
- **Functions**: def, return
- **Statements**: if, for, while, break
- **Patterns**: vectorization, divide & conquer, map-reduce
- **Algorithms**: GCD, mean/variance, sorting, searching
- **Modules**: numpy, matplotlib, numba, scipy, sympy, scikit-learn

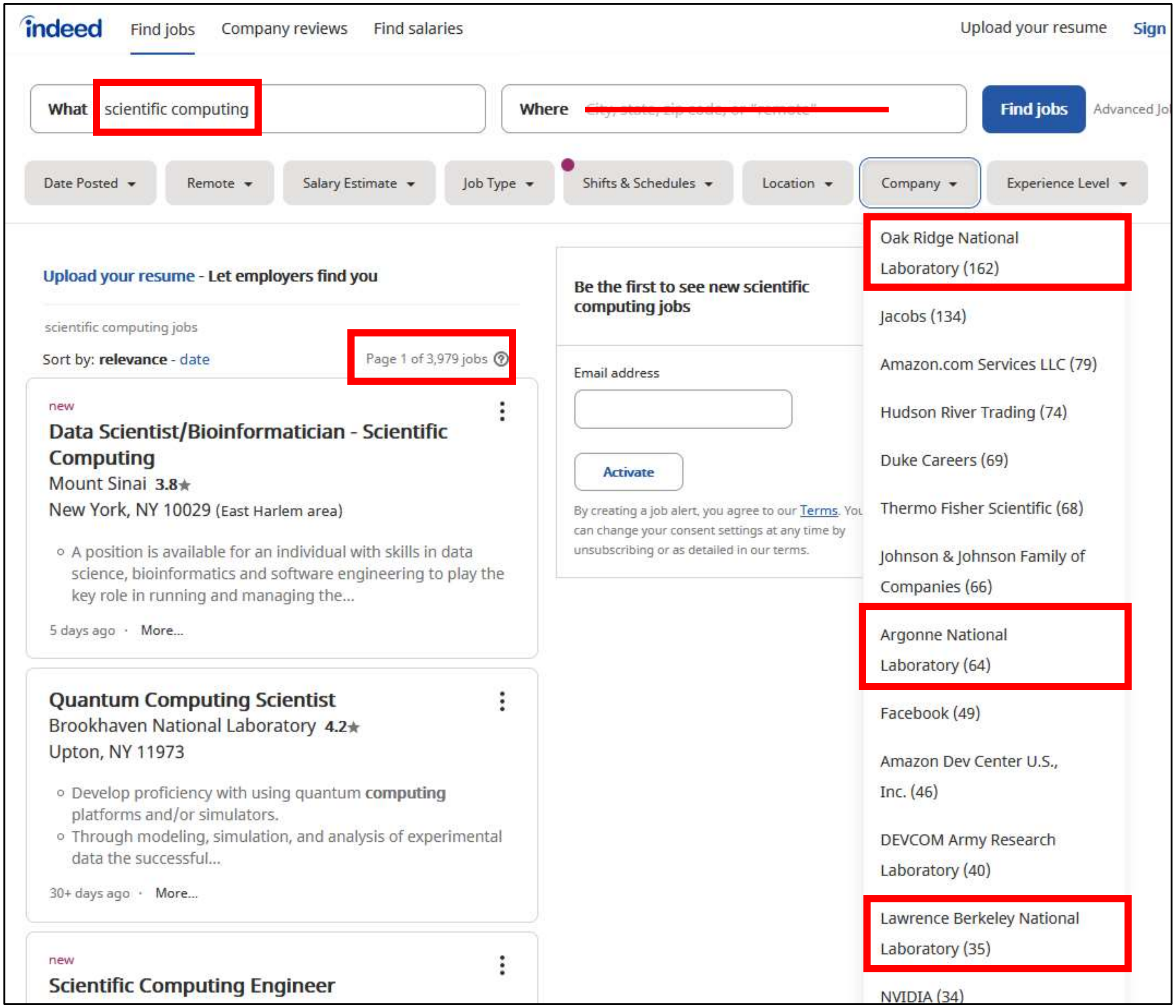




Useful Python Reference Sites

- https://docs.python.org/3/tutorial
- https://www.learnpython.org
- https://realpython.com
- https://www.w3schools.com/python
- https://www.fullstackpython.com

SciComp = The Pathway to Internships



indeed Find jobs Company reviews Find salaries Upload your resume Sign

What scientific computing

Where ~~City, state, zip code, or "remote"~~

Find jobs Advanced Jo

Date Posted ▾ Remote ▾ Salary Estimate ▾ Job Type ▾ Shifts & Schedules ▾ Location ▾ Company ▾ Experience Level ▾

Upload your resume - Let employers find you

scientific computing jobs

Sort by: **relevance** - date

Page 1 of 3,979 jobs ?

new

Data Scientist/Bioinformatician - Scientific Computing

Mount Sinai 3.8★

New York, NY 10029 (East Harlem area)

- A position is available for an individual with skills in data science, bioinformatics and software engineering to play the key role in running and managing the...

5 days ago · More...

Quantum Computing Scientist

Brookhaven National Laboratory 4.2★

Upton, NY 11973

- Develop proficiency with using quantum **computing** platforms and/or simulators.
- Through modeling, simulation, and analysis of experimental data the successful...

30+ days ago · More...

new

Scientific Computing Engineer

Be the first to see new scientific computing jobs

Email address

Activate

By creating a job alert, you agree to our Terms. You can change your consent settings at any time by unsubscribing or as detailed in our terms.

Oak Ridge National Laboratory (162)

Jacobs (134)

Amazon.com Services LLC (79)

Hudson River Trading (74)

Duke Careers (69)

Thermo Fisher Scientific (68)

Johnson & Johnson Family of Companies (66)

Argonne National Laboratory (64)

Facebook (49)

Amazon Dev Center U.S., Inc. (46)

DEVCOM Army Research Laboratory (40)

Lawrence Berkeley National Laboratory (35)

NVIDIA (34)

Session **01** – Goals

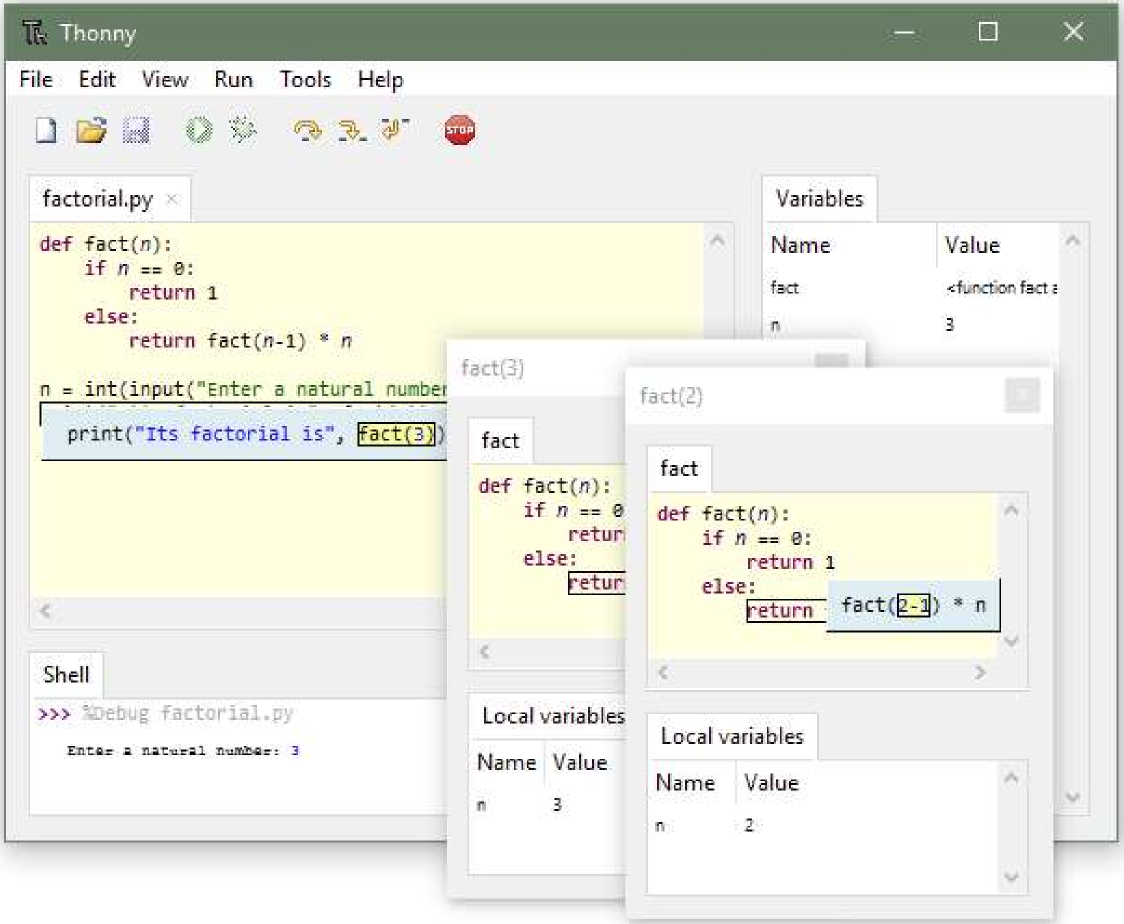
- Learn how to use the **Thonny** integrated development environment (IDE) and to install packages and plug-ins
- Review the rules for declaring **identifiers** in Python
- Declare **variables** and use built-in data **types**
- Display the values of variables in the output "Shell" window
- Review mathematical operators (PEMDAS) in Python
- Understand **statements** and **scopes**
- Create simple loops with the **for** statement
- Introduce **range**() as a "lazy" list definition

About the Thonny IDE

Thonny

Python IDE for beginners

Download version 4.0.1 for
Windows • Mac • Linux



```
def fact(n):
    if n == 0:
        return 1
    else:
        return fact(n-1) * n

n = int(input("Enter a natural number
print("Its factorial is", fact(3))
```

Name	Value
fact	<function fact a
n	3

```
>>> %Debug factorial.py
  Enter a natural number: 3
```

fact(3)

```
def fact(n):
    if n == 0
        retur
    else:
        retur
```

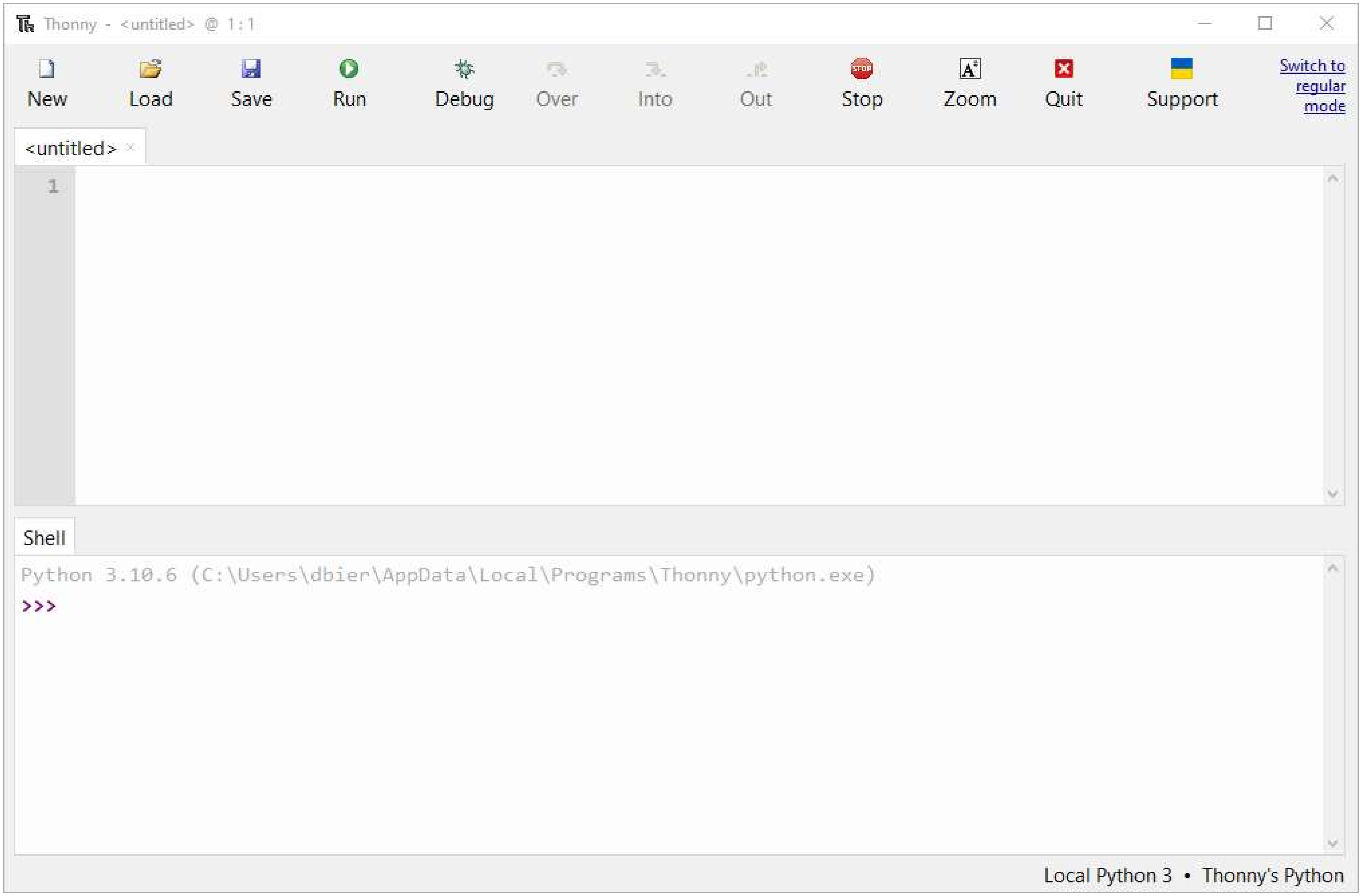
Name	Value
n	3

fact(2)

```
def fact(n):
    if n == 0:
        return 1
    else:
        return fact(2-1) * n
```

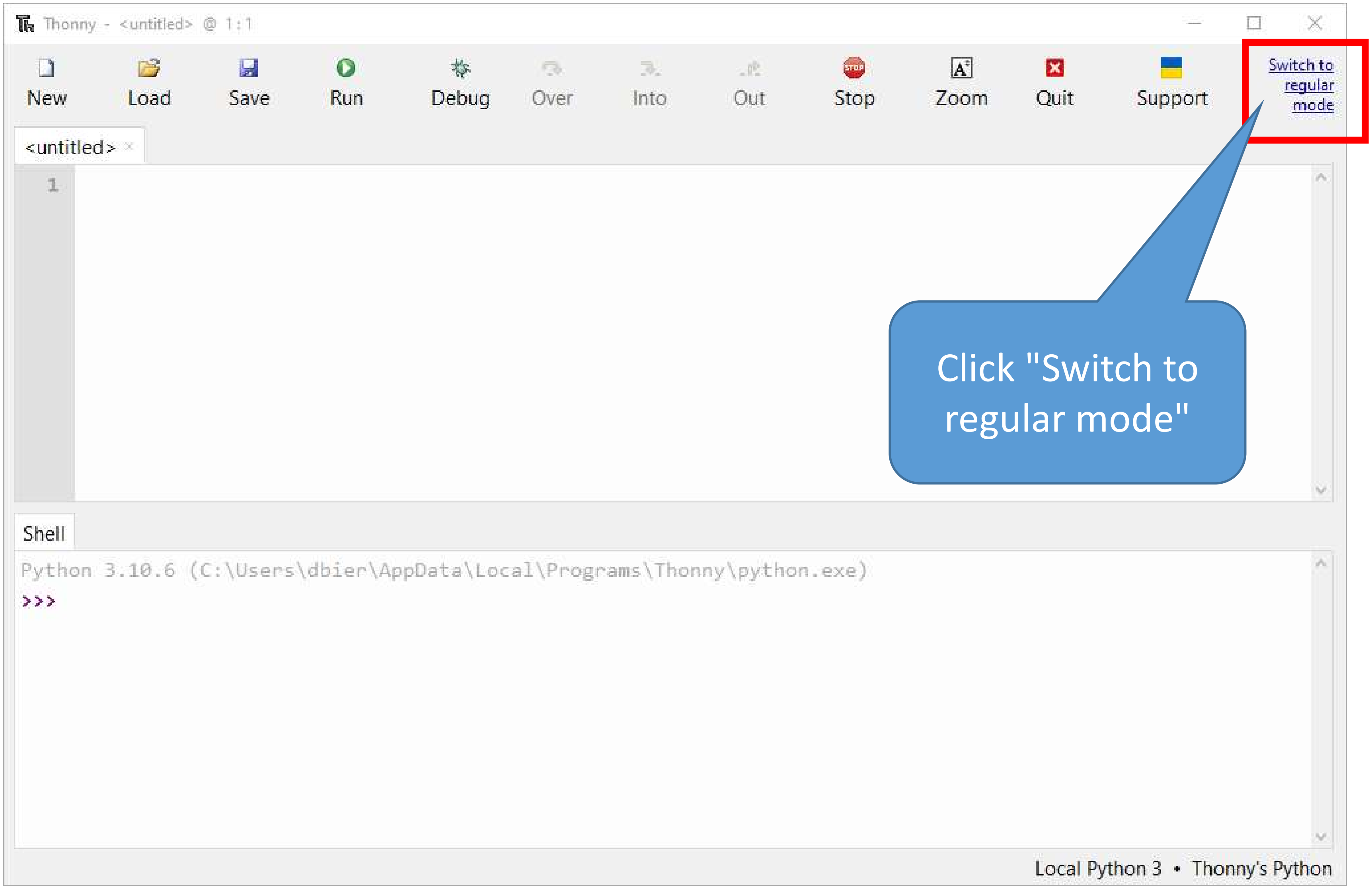
Name	Value
n	2

Run the Thonny IDE



```
Python 3.10.6 (C:\Users\dbier\AppData\Local\Programs\Thonny\python.exe)
>>>
```

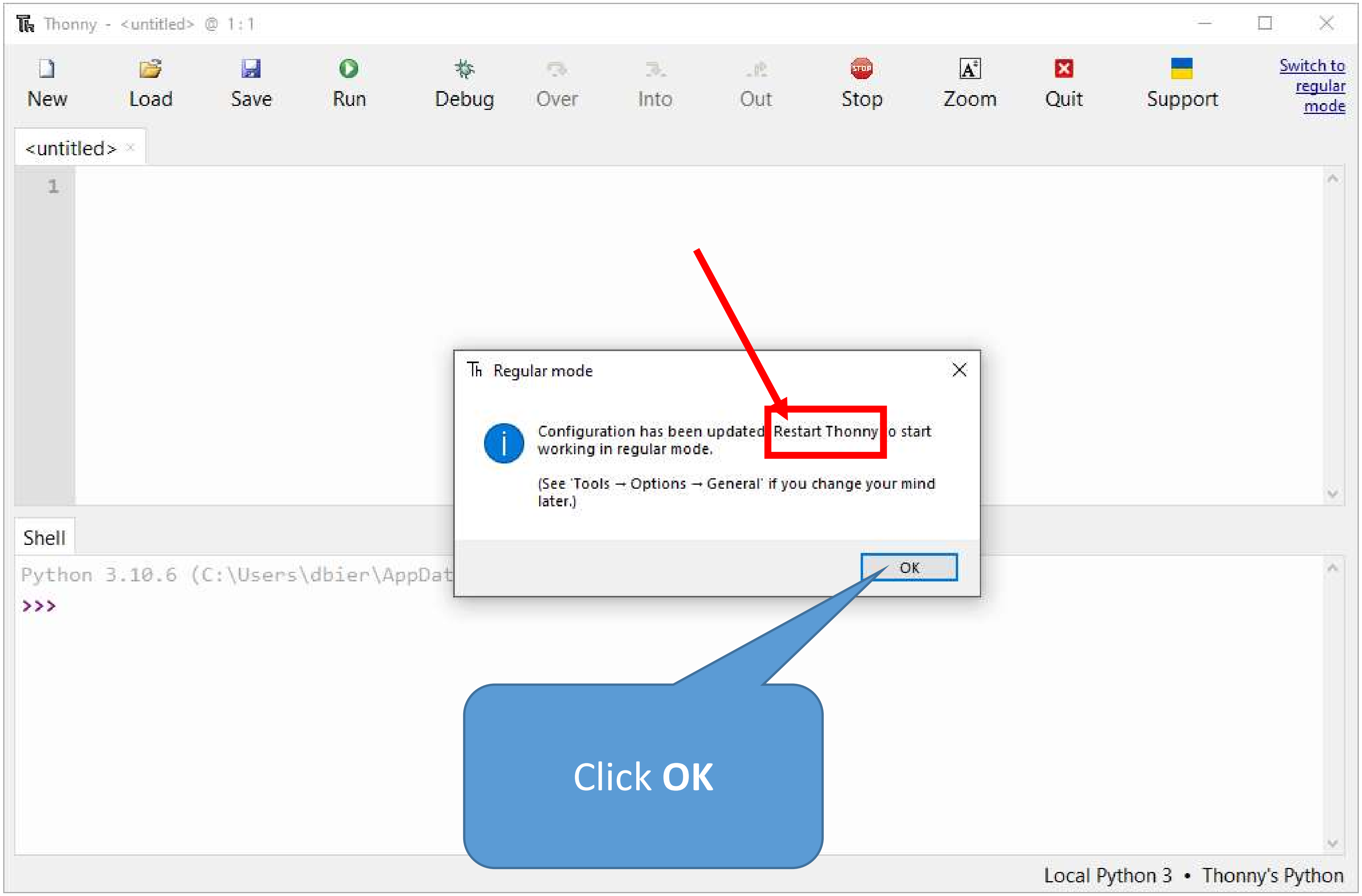
Switch the Thonny IDE into Regular Mode



```
Python 3.10.6 (C:\Users\dbier\AppData\Local\Programs\Thonny\python.exe)
>>>
```

Click "Switch to regular mode"

Restart the Thonny IDE



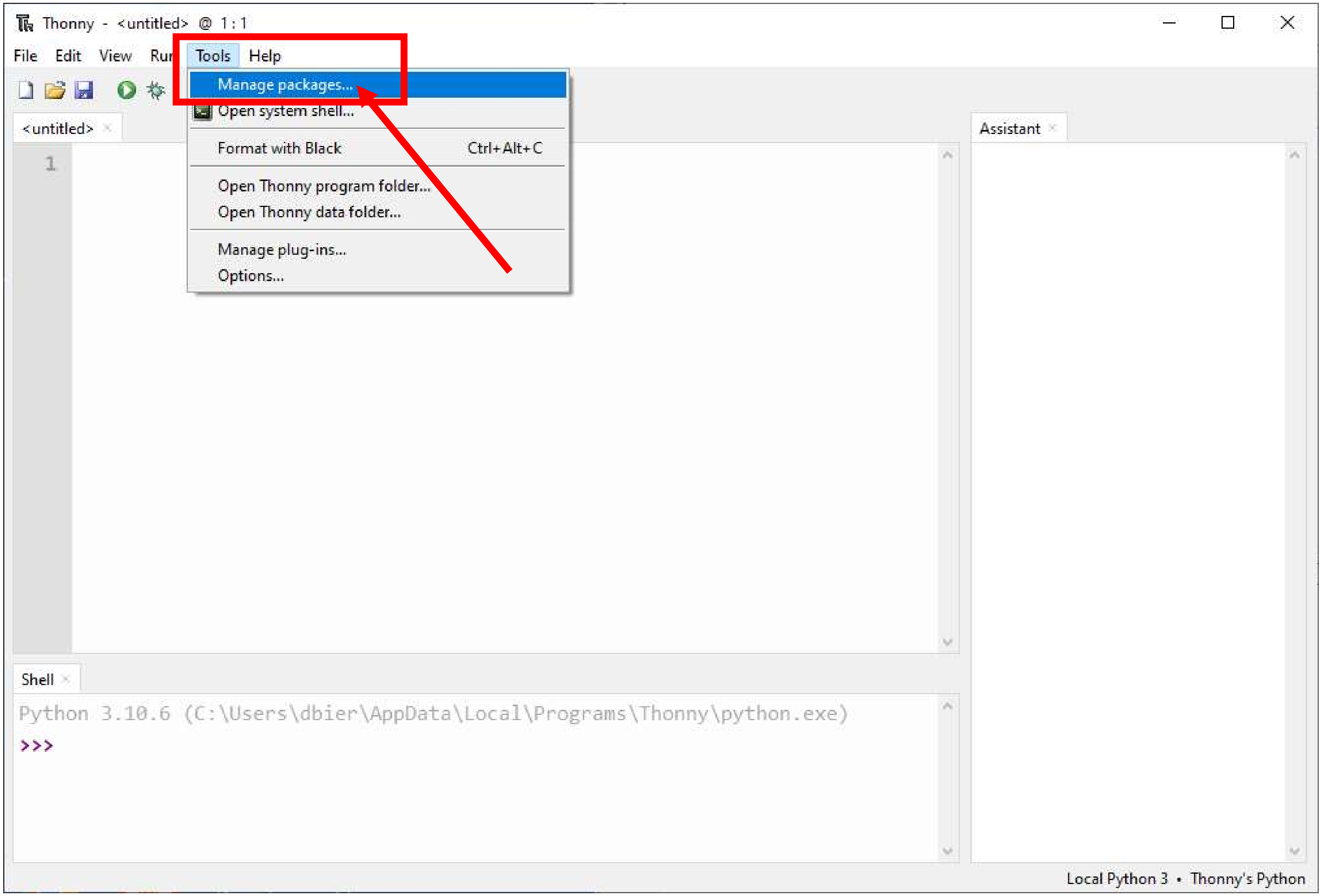
Configuration has been updated. Restart Thonny to start working in regular mode.

(See 'Tools → Options → General' if you change your mind later.)

```
Python 3.10.6 (C:\Users\dbier\AppDat
>>>
```

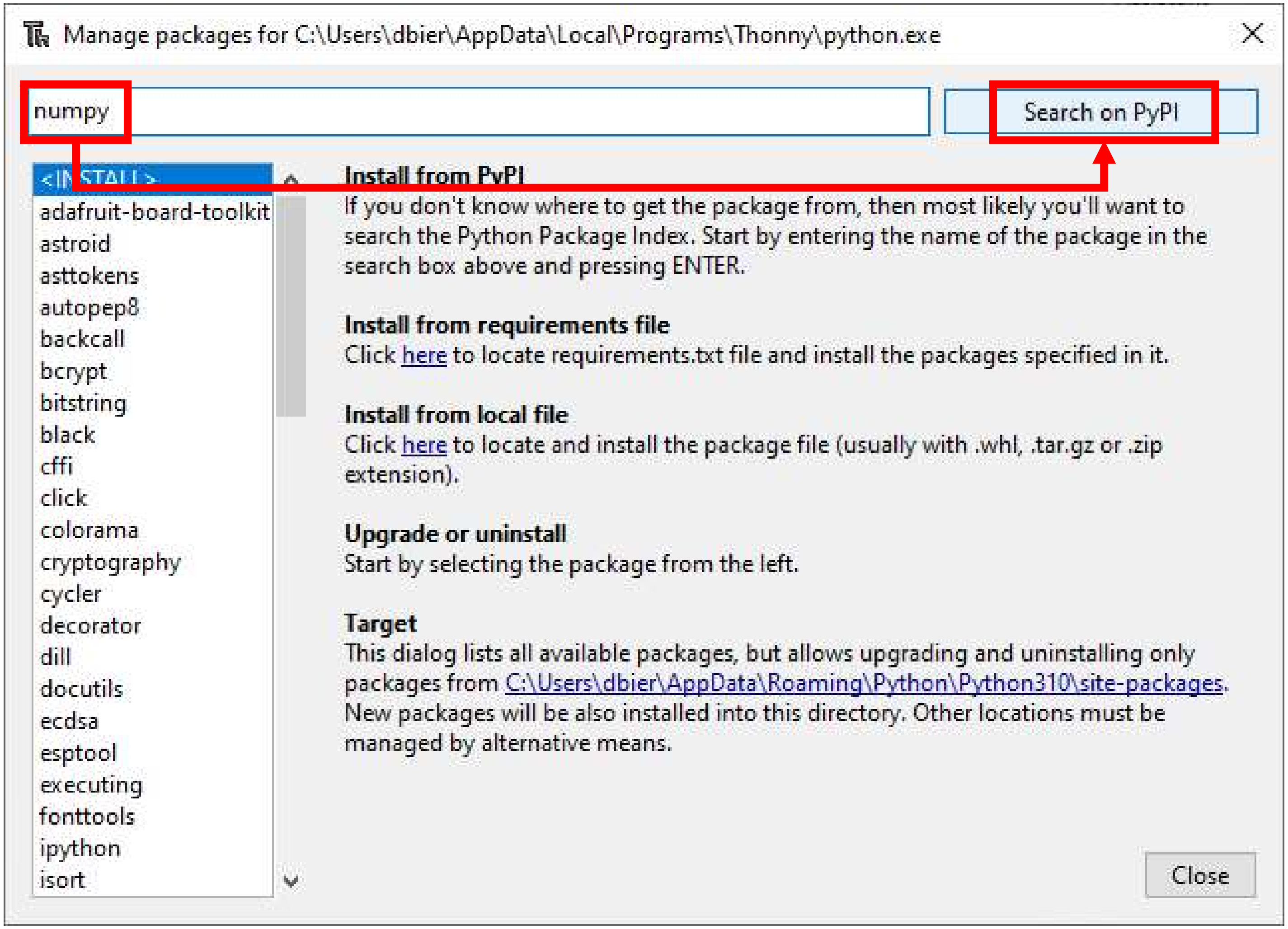
Click **OK**

Installing Python Packages into Thonny



```
Python 3.10.6 (C:\Users\dbier\AppData\Local\Programs\Thonny\python.exe)
>>>
```

Find the Package Name on **PyPI**



Manage packages for C:\Users\dbier\AppData\Local\Programs\Thonny\python.exe

Install from PyPI
If you don't know where to get the package from, then most likely you'll want to search the Python Package Index. Start by entering the name of the package in the search box above and pressing ENTER.

Install from requirements file
Click here to locate requirements.txt file and install the packages specified in it.

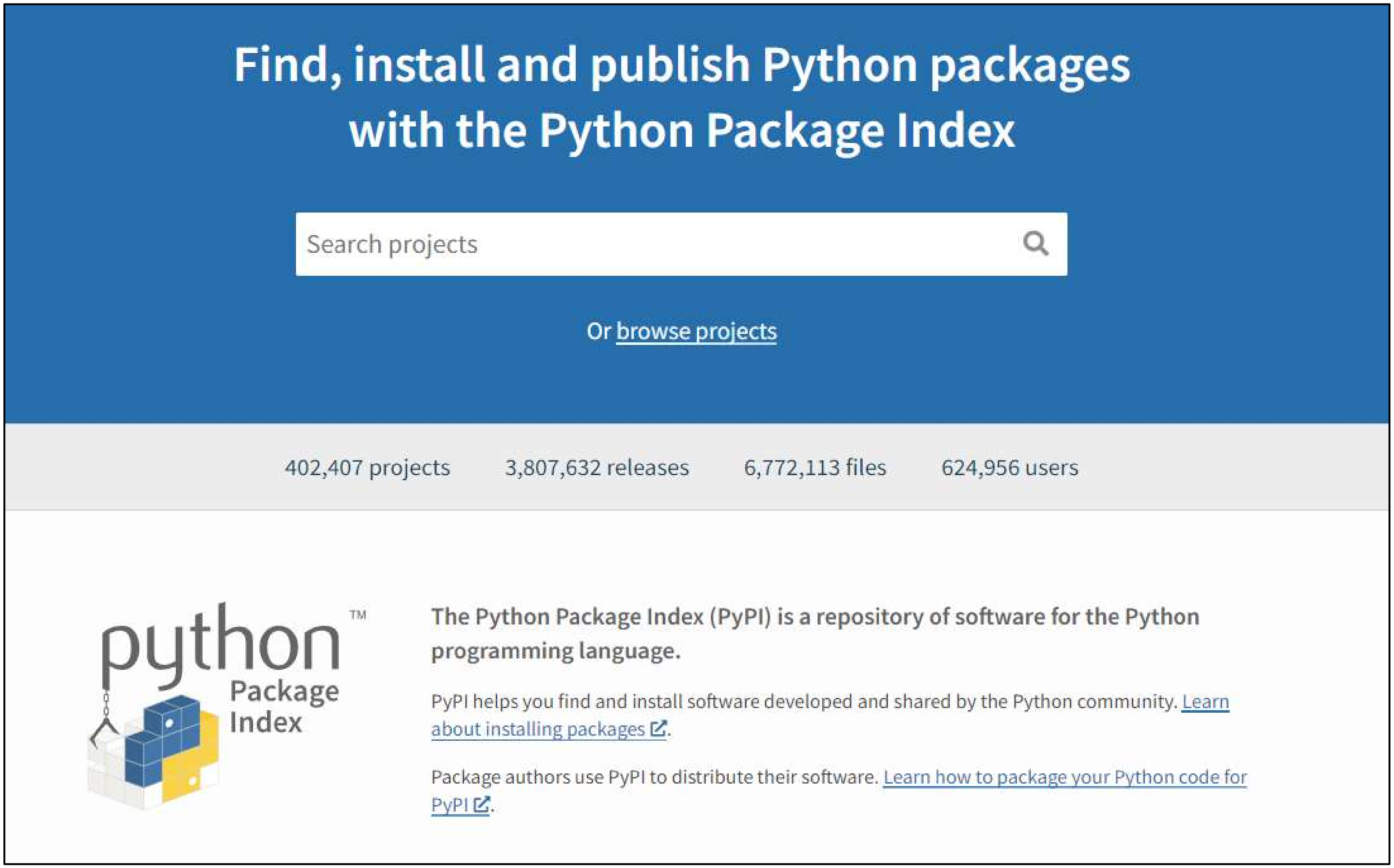
Install from local file
Click here to locate and install the package file (usually with .whl, .tar.gz or .zip extension).

Upgrade or uninstall
Start by selecting the package from the left.

Target
This dialog lists all available packages, but allows upgrading and uninstalling only packages from C:\Users\dbier\AppData\Roaming\Python\Python310\site-packages. New packages will be also installed into this directory. Other locations must be managed by alternative means.

About **PyPI** (Python Package Index)

https://pypi.org



Find, install and publish Python packages with the Python Package Index

Or browse projects

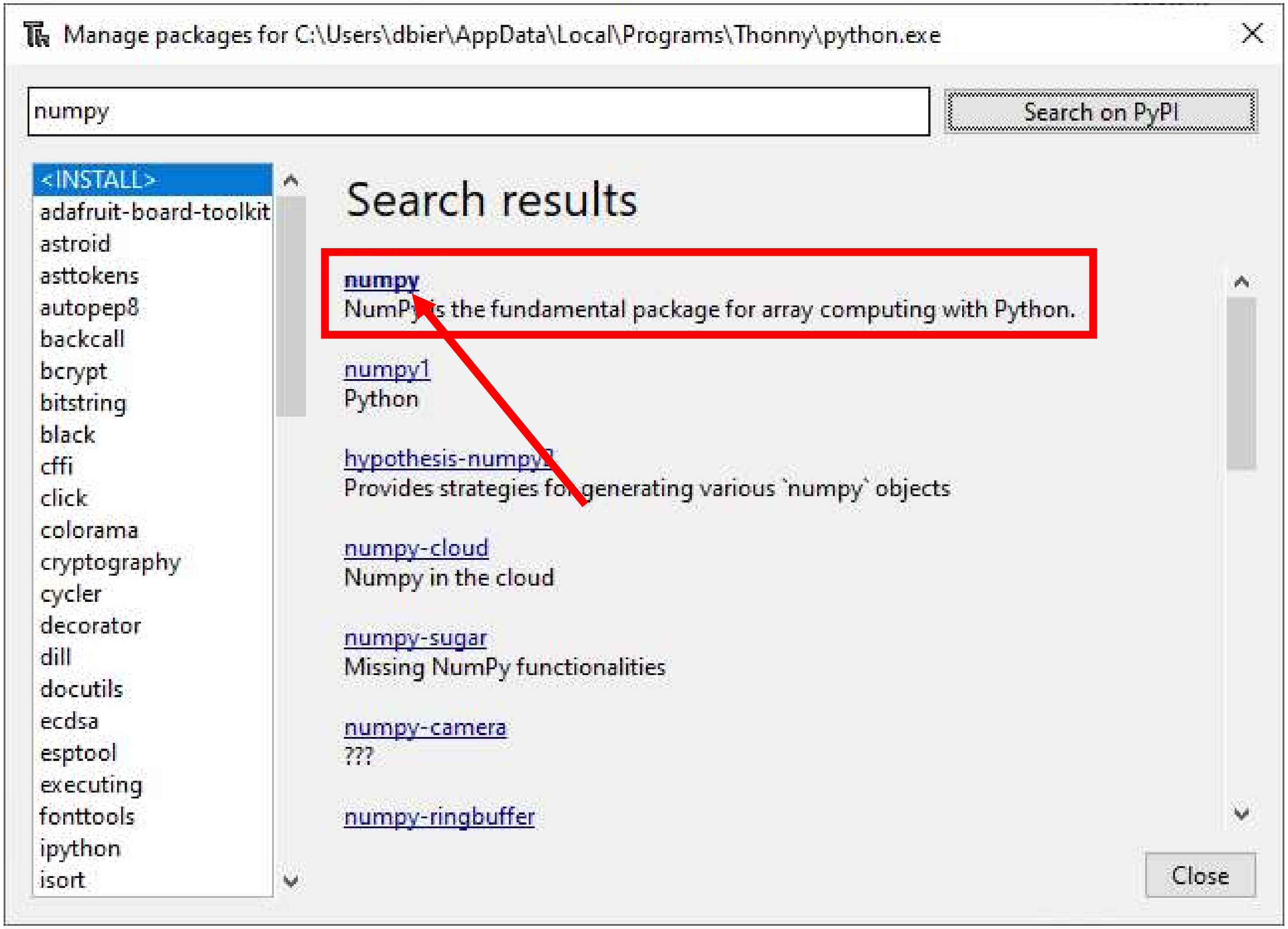
402,407 projects 3,807,632 releases 6,772,113 files 624,956 users

The Python Package Index (PyPI) is a repository of software for the Python programming language.

PyPI helps you find and install software developed and shared by the Python community. Learn about installing packages.

Package authors use PyPI to distribute their software. Learn how to package your Python code for PyPI.

Select the Package Name



Manage packages for C:\Users\dbier\AppData\Local\Programs\Thonny\python.exe

numpy

Search results

numpy
NumPy is the fundamental package for array computing with Python.

numpy1
Python

hypothesis-numpy2
Provides strategies for generating various `numpy` objects

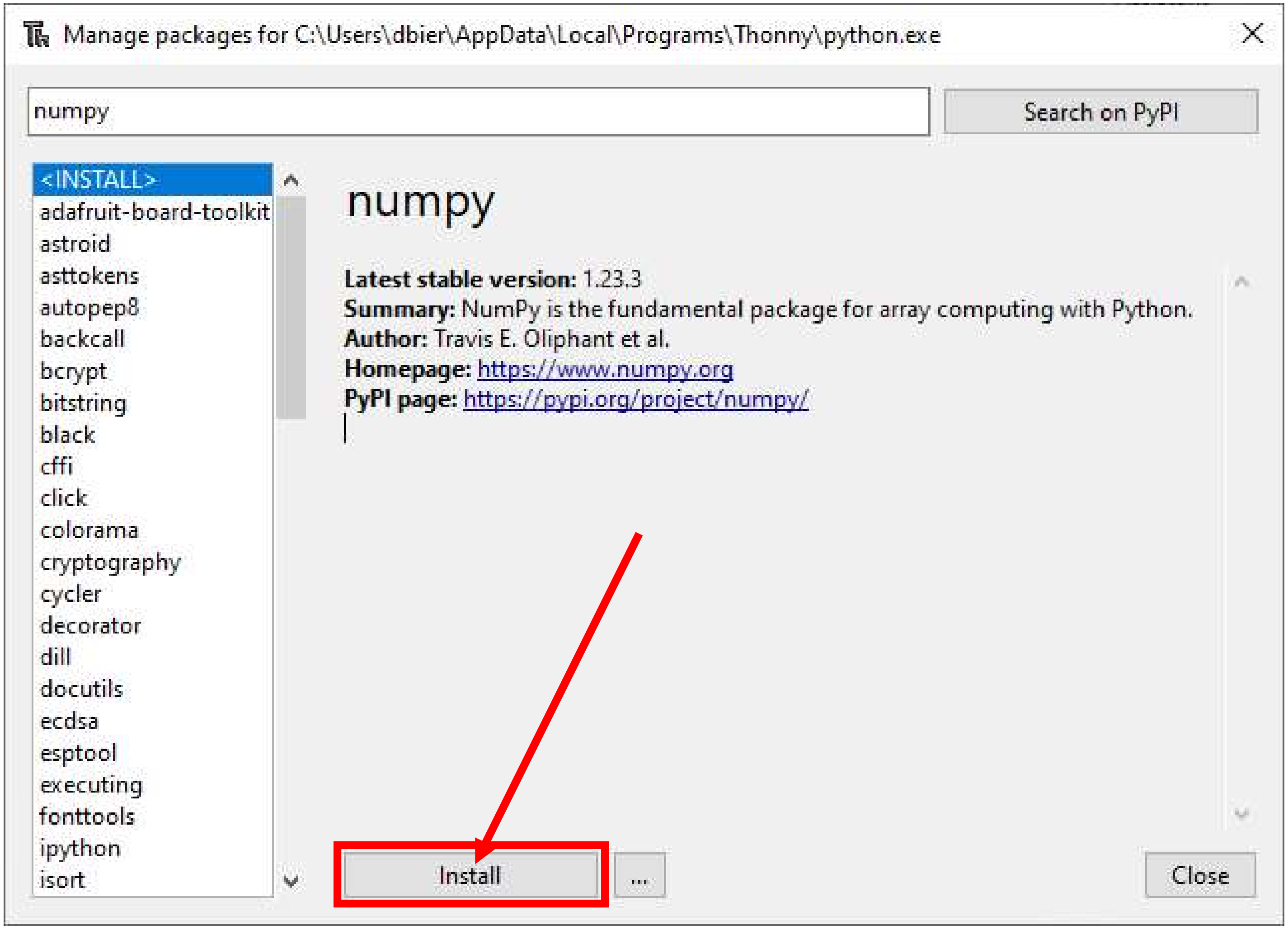
numpy-cloud
Numpy in the cloud

numpy-sugar
Missing NumPy functionalities

numpy-camera
???

numpy-ringbuffer

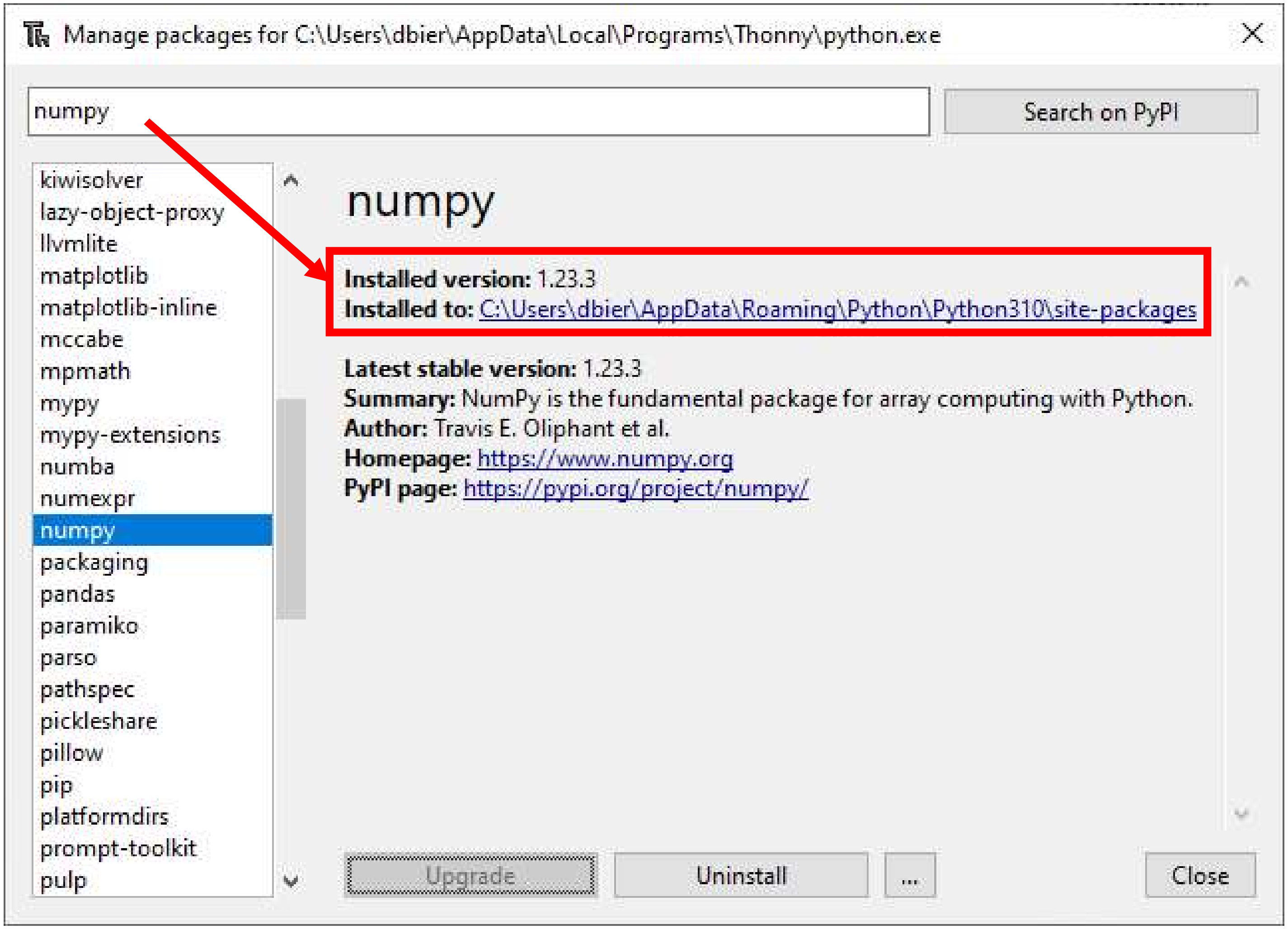
Install the Package



numpy

Latest stable version: 1.23.3
Summary: NumPy is the fundamental package for array computing with Python.
Author: Travis E. Oliphant et al.
Homepage: https://www.numpy.org
PyPI page: https://pypi.org/project/numpy/

Verify the Package Installation

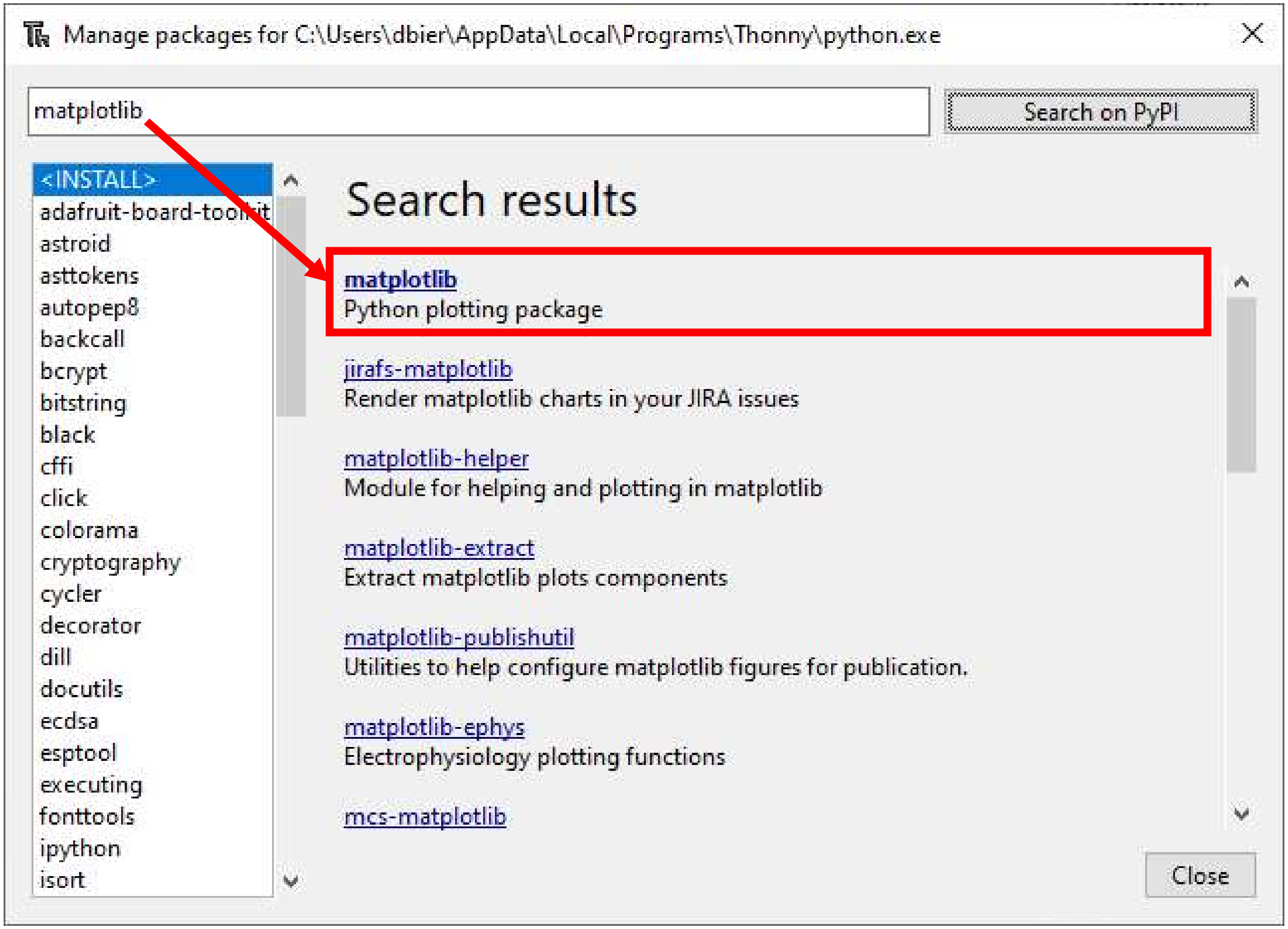


numpy

Installed version: 1.23.3
Installed to: C:\Users\dbier\AppData\Roaming\Python\Python310\site-packages

Latest stable version: 1.23.3
Summary: NumPy is the fundamental package for array computing with Python.
Author: Travis E. Oliphant et al.
Homepage: https://www.numpy.org
PyPI page: https://pypi.org/project/numpy/

Install the **matplotlib** Package



Manage packages for C:\Users\dbier\AppData\Local\Programs\Thonny\python.exe

matplotlib

<INSTALL>
adafruit-board-toolkit
astroid
asttokens
autopep8
backcall
bcrypt
bitstring
black
cffi
click
colorama
cryptography
cycler
decorator
dill
docutils
ecdsa
esptool
executing
fonttools
ipython
isort

Search results

matplotlib
Python plotting package

jirafs-matplotlib
Render matplotlib charts in your JIRA issues

matplotlib-helper
Module for helping and plotting in matplotlib

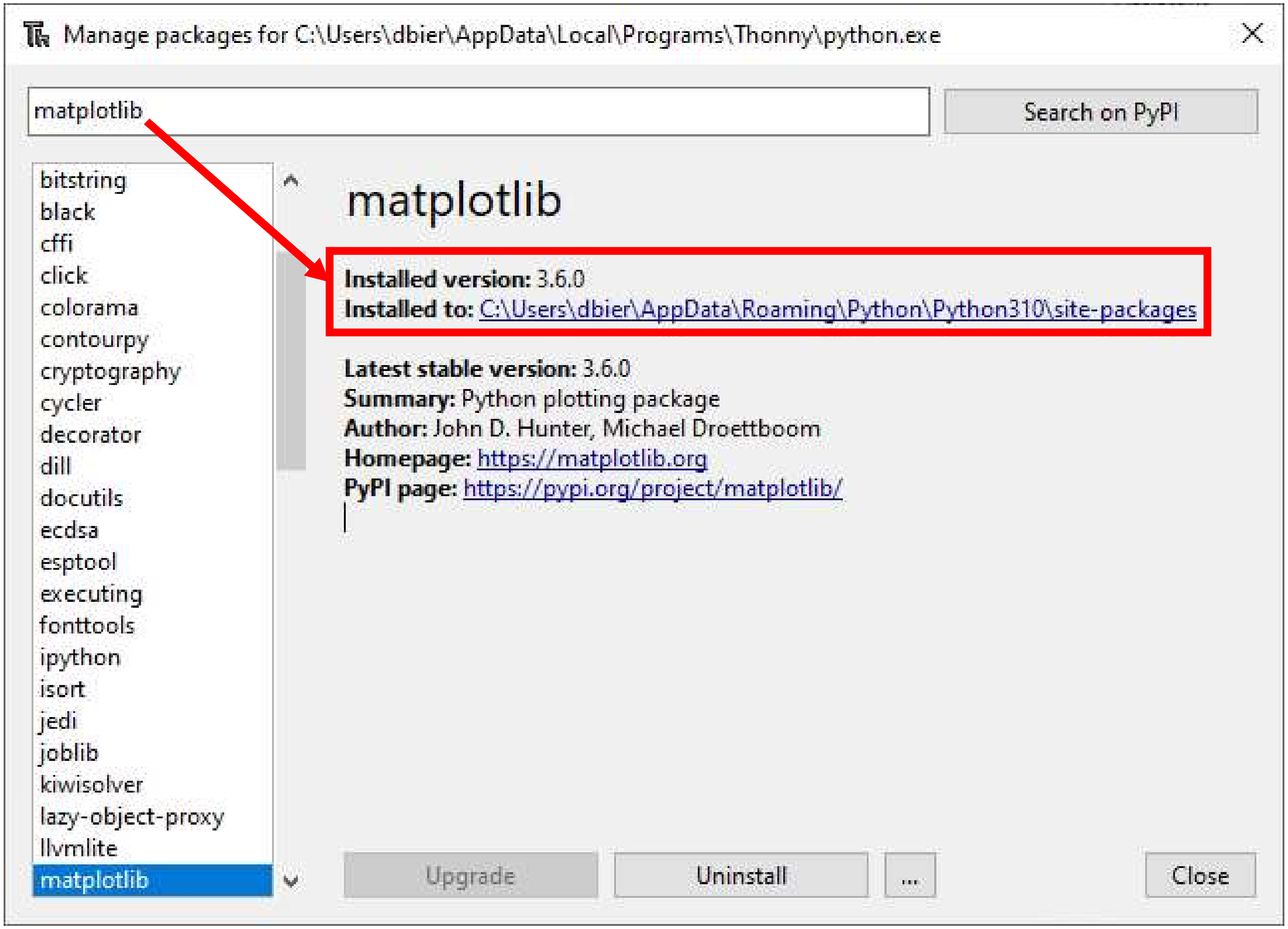
matplotlib-extract
Extract matplotlib plots components

matplotlib-publishutil
Utilities to help configure matplotlib figures for publication.

matplotlib-ephys
Electrophysiology plotting functions

mcs-matplotlib

Verify the **matplotlib** Package Installation

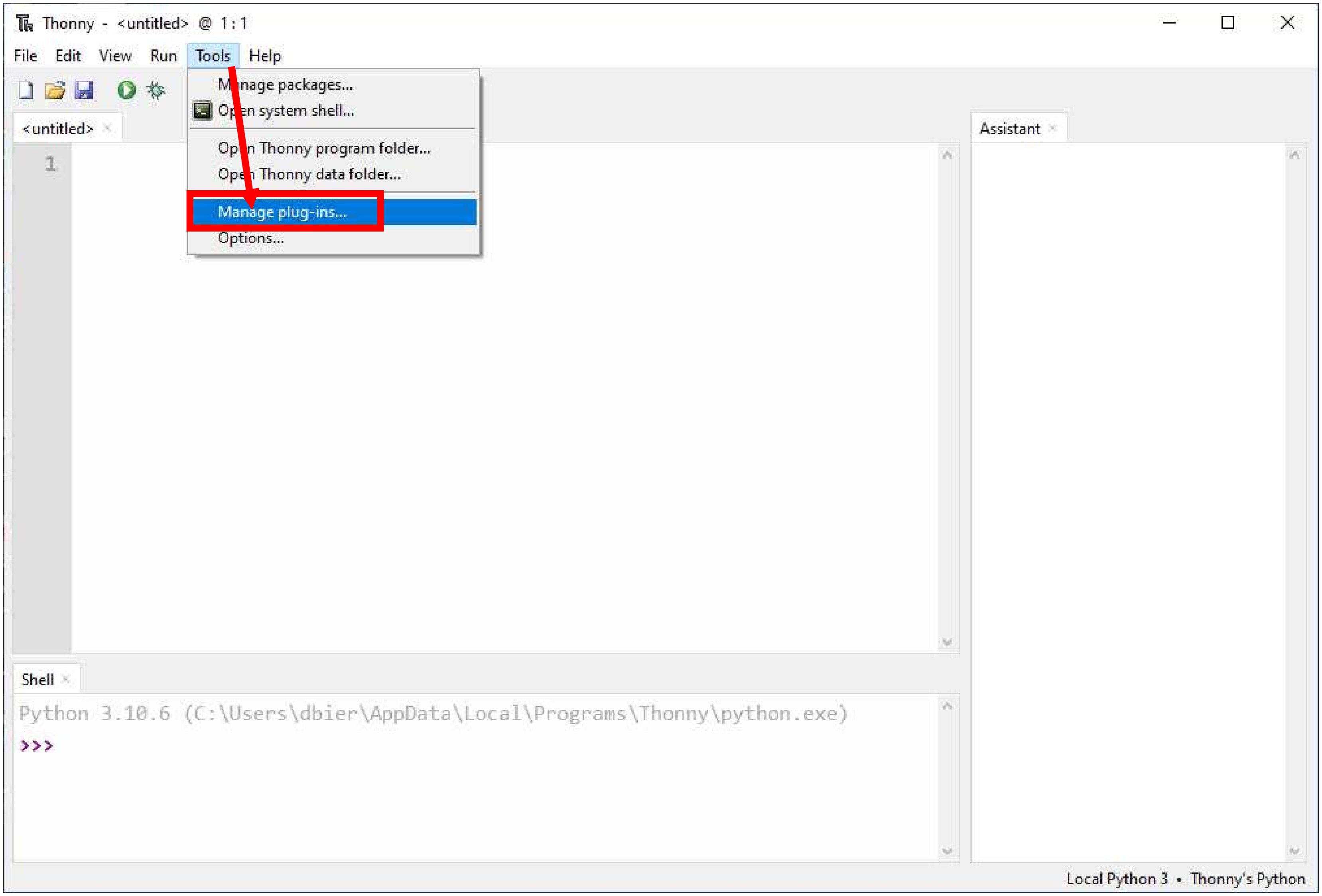


matplotlib

Installed version: 3.6.0
Installed to: C:\Users\dbier\AppData\Roaming\Python\Python310\site-packages

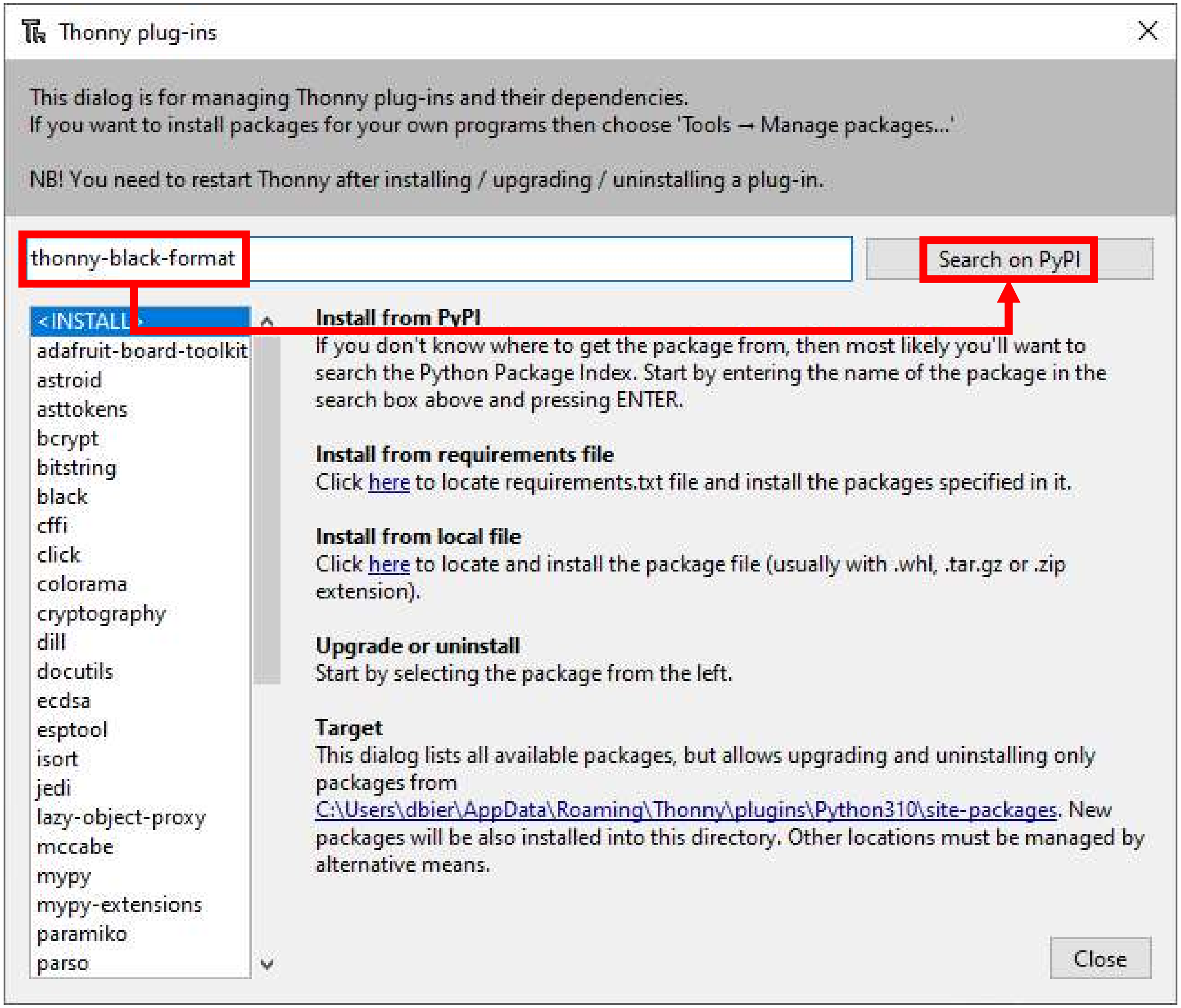
Latest stable version: 3.6.0
Summary: Python plotting package
Author: John D. Hunter, Michael Droettboom
Homepage: https://matplotlib.org
PyPI page: https://pypi.org/project/matplotlib/

Installing Thonny **plug-ins**


Thonny - <untitled> @ 1 : 1
File
Edit
View
Run
Tools
Help
Manage packages...
Open system shell...
Open Thonny program folder...
Open Thonny data folder...
Manage plug-ins...
Options...
<untitled>
Assistant
Shell
Local Python 3 • Thonny's Python

```
Python 3.10.6 (C:\Users\dbier\AppData\Local\Programs\Thonny\python.exe)
>>>
```

Find the plug-in name on **PyPI**



This dialog is for managing Thonny plug-ins and their dependencies.
If you want to install packages for your own programs then choose 'Tools → Manage packages...'

NB! You need to restart Thonny after installing / upgrading / uninstalling a plug-in.

thonny-black-format

Install from PyPI
If you don't know where to get the package from, then most likely you'll want to search the Python Package Index. Start by entering the name of the package in the search box above and pressing ENTER.

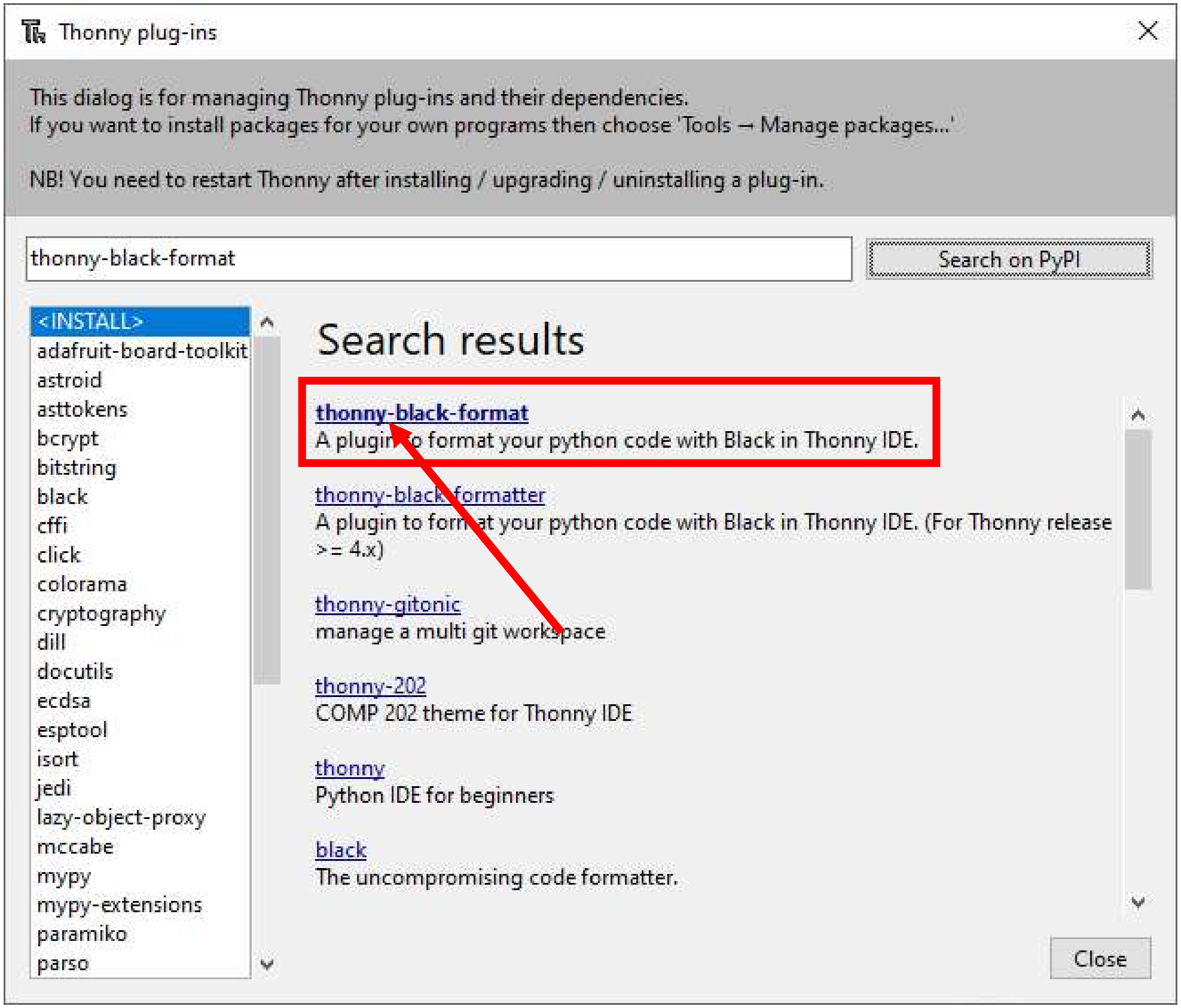
Install from requirements file
Click here to locate requirements.txt file and install the packages specified in it.

Install from local file
Click here to locate and install the package file (usually with .whl, .tar.gz or .zip extension).

Upgrade or uninstall
Start by selecting the package from the left.

Target
This dialog lists all available packages, but allows upgrading and uninstalling only packages from C:\Users\dbier\AppData\Roaming\Thonny\plugins\Python310\site-packages. New packages will be also installed into this directory. Other locations must be managed by alternative means.

Select plug-in name



This dialog is for managing Thonny plug-ins and their dependencies.
If you want to install packages for your own programs then choose 'Tools → Manage packages...'

NB! You need to restart Thonny after installing / upgrading / uninstalling a plug-in.

thonny-black-format

<INSTALL>
adafruit-board-toolkit
astroid
asttokens
bcrypt
bitstring
black
cffi
click
colorama
cryptography
dill
docutils
ecdsa
esptool
isort
jedi
lazy-object-proxy
mccabe
mypy
mypy-extensions
paramiko
parso

Search results

thonny-black-format
A plugin to format your python code with Black in Thonny IDE.

thonny-black-formatter
A plugin to format your python code with Black in Thonny IDE. (For Thonny release >= 4.x)

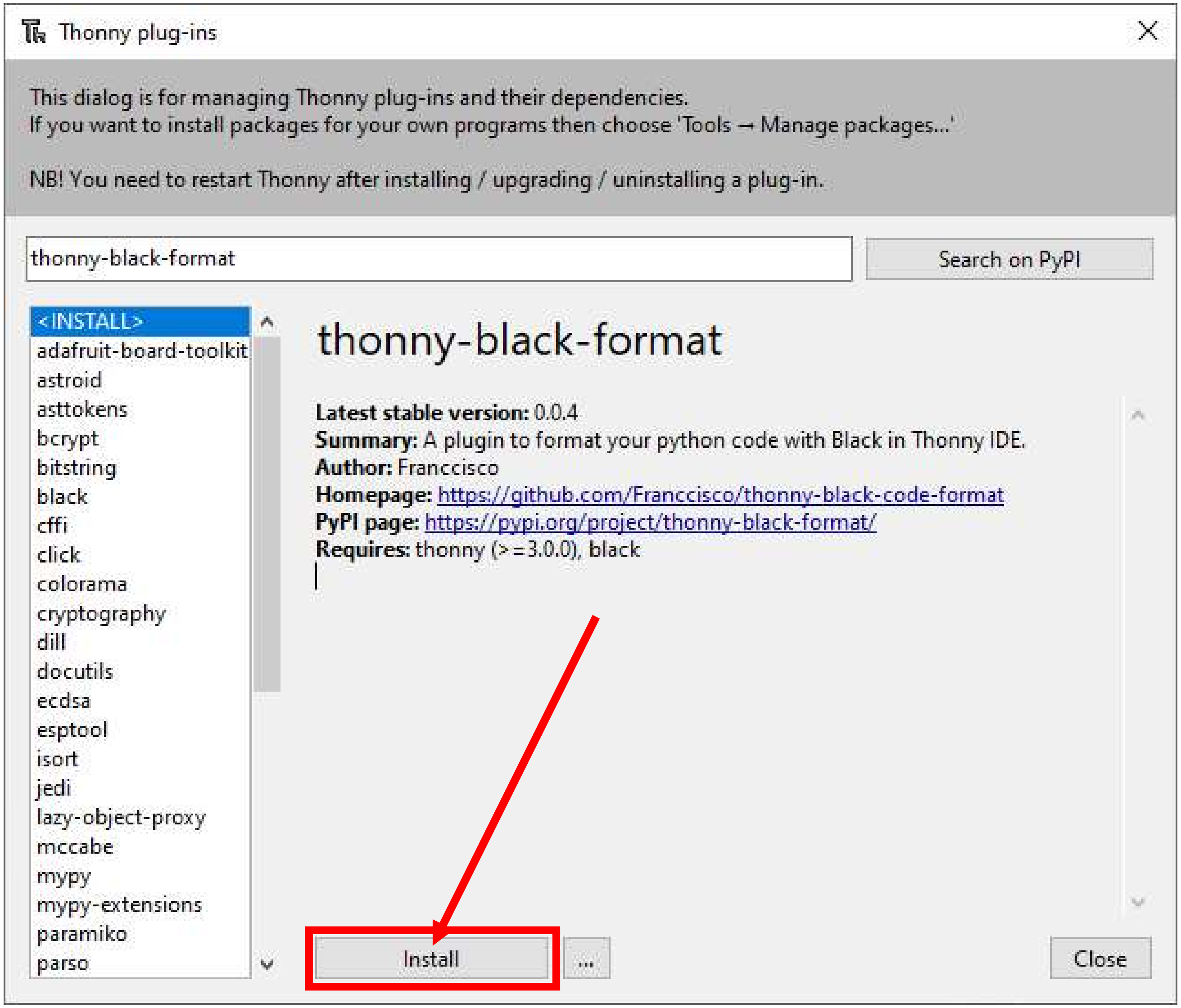
thonny-gitonic
manage a multi git workspace

thonny-202
COMP 202 theme for Thonny IDE

thonny
Python IDE for beginners

black
The uncompromising code formatter.

Install the plug-in



This dialog is for managing Thonny plug-ins and their dependencies.
If you want to install packages for your own programs then choose 'Tools → Manage packages...'

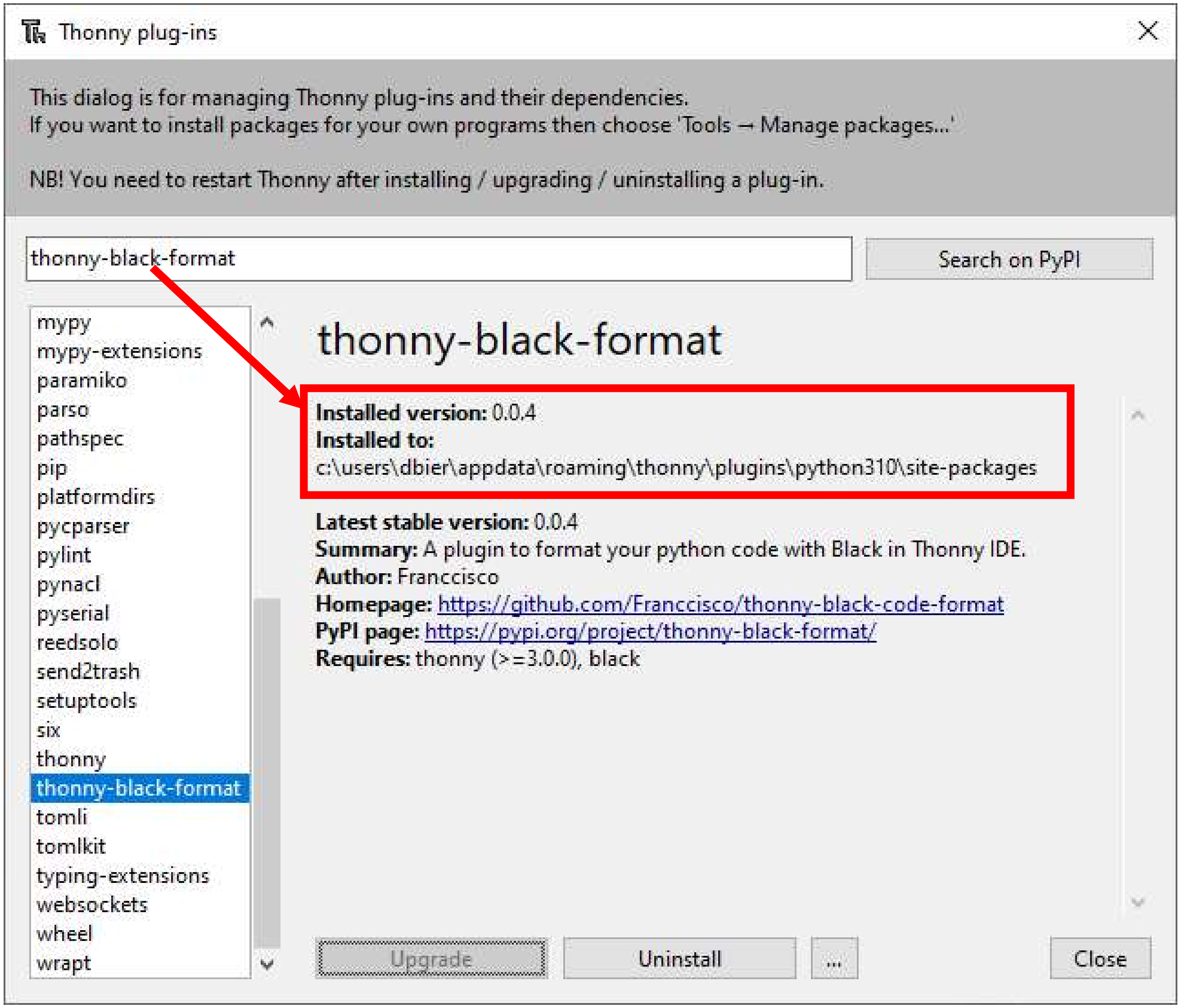
NB! You need to restart Thonny after installing / upgrading / uninstalling a plug-in.

thonny-black-format

thonny-black-format

Latest stable version: 0.0.4
Summary: A plugin to format your python code with Black in Thonny IDE.
Author: Franccisco
Homepage: https://github.com/Franccisco/thonny-black-code-format
PyPI page: https://pypi.org/project/thonny-black-format/
Requires: thonny (>=3.0.0), black

Verify the plug-in installation



This dialog is for managing Thonny plug-ins and their dependencies.
If you want to install packages for your own programs then choose 'Tools → Manage packages...'

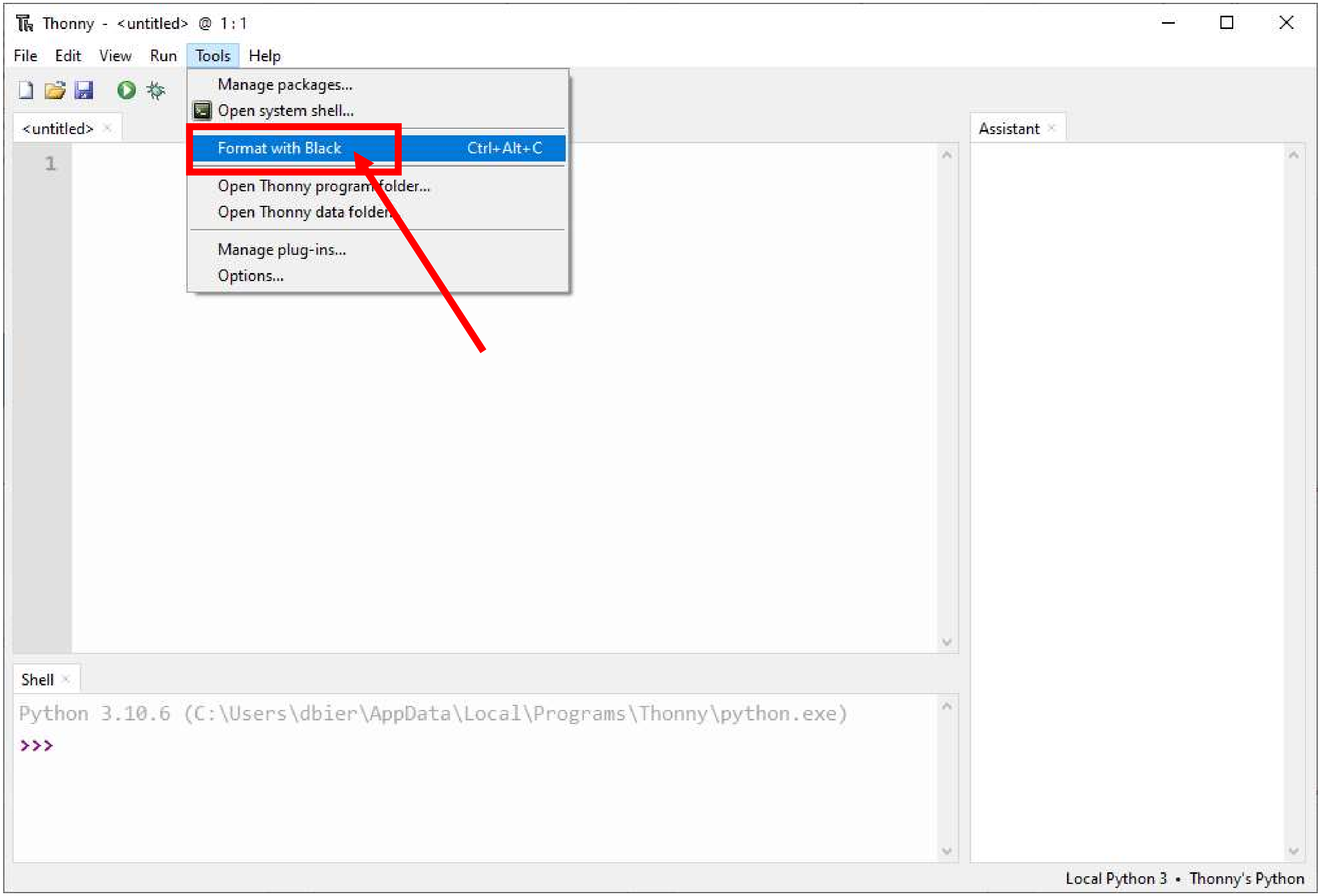
NB! You need to restart Thonny after installing / upgrading / uninstalling a plug-in.

thonny-black-format

Installed version: 0.0.4
Installed to:
c:\users\dbier\appdata\roaming\thonny\plugins\python310\site-packages

Latest stable version: 0.0.4
Summary: A plugin to format your python code with Black in Thonny IDE.
Author: Franccisco
Homepage: https://github.com/Franccisco/thonny-black-code-format
PyPI page: https://pypi.org/project/thonny-black-format/
Requires: thonny (>=3.0.0), black

Restart the Thonny IDE to use a plug-in



```
Python 3.10.6 (C:\Users\dbier\AppData\Local\Programs\Thonny\python.exe)
>>>
```

Identifiers

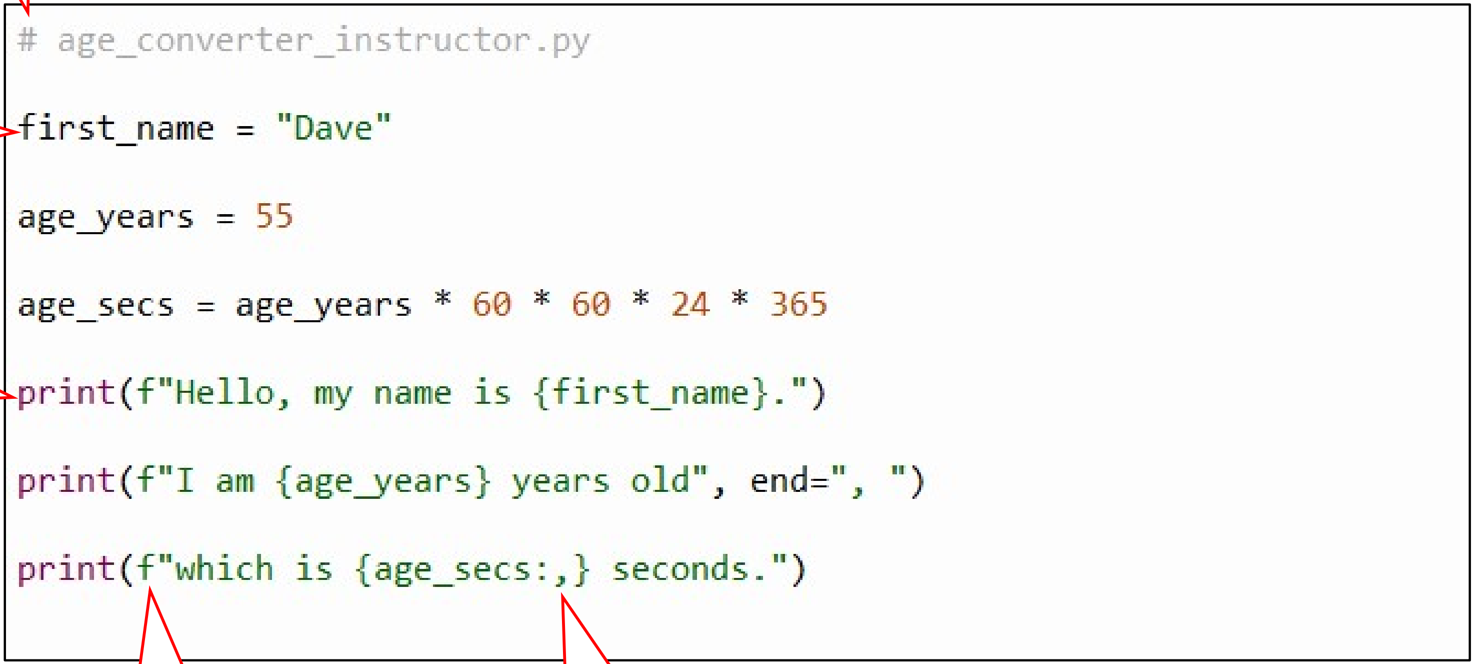
- Identifiers are just **names** – everything in code has a name
  - Names must be < 64 chars in length
  - They can include upper- or lower-case letters & numbers
  - Identifiers must start with a letter and **cannot contain spaces!**
- Three types of identifier "casing"
  1. CamelCaseEachWord (first letter is Capitalized)
  2. camelCaseEachWord (first letter is not capitalized)
  3. all **lower_case** with underscores **(Snake case in Python!)** ✓
- Identifiers in Python are **case sensitive!!**
  - ***x*** *is not the same as* ***X***
  - Use ALLCAPS to define global constants (very rare)

Identifiers

Source code comments start with a #

Variable Name

Function Name



```
# age_converter_instructor.py

first_name = "Dave"

age_years = 55

age_secs = age_years * 60 * 60 * 24 * 365

print(f"Hello, my name is {first_name}.")

print(f"I am {age_years} years old", end=", ")

print(f"which is {age_secs:,} seconds.")
```

An **f** string

A **formatter**

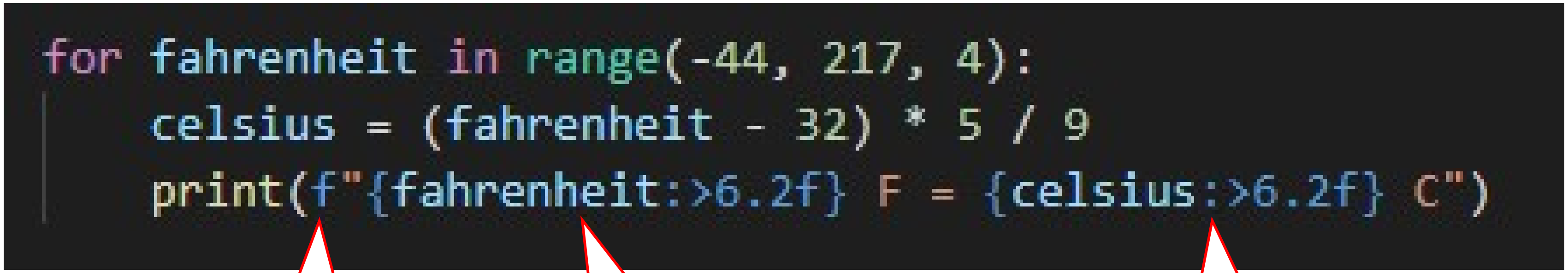
Variable Types

- **Variables** store data in memory to be used later
  - Variables can be called whatever you want
  - Pick variable names that mean something to a human
  - Use **snake_case** (all lower case, underscores to break words)
- Python supports many built-in **data types** for variables:
  - **int** = Stores integers only
  - **float** = Stores real numbers with 15 digits of precision
  - **bool** = Stores **True** or **False** (Boolean logic)
  - **str** = Stores zero or more letters & numbers
- Python mostly “infers” the type of a variable

Displaying Variables

- The **print**() function is used to display the value of variables
  - When running inside Thonny, the **print**() output will show up in the "Shell" terminal window at the bottom of screen
  - String *literals* must be enclosed in quotation marks (single or double)
- Python can substitute a variable's value into a **placeholder**
  - To make a **placeholder** (aka a *replacement field*) you enclose the variable name between curly braces **{}**
  - Substituting a variable's actual value into its *replacement field* is called *string interpolation*
- Placeholders can also contain **format specifiers**
  - You can specify number of digits to the right of the decimal, etc.
  - You can specify left/right/center justification, column width, etc.

print() and **f-strings**



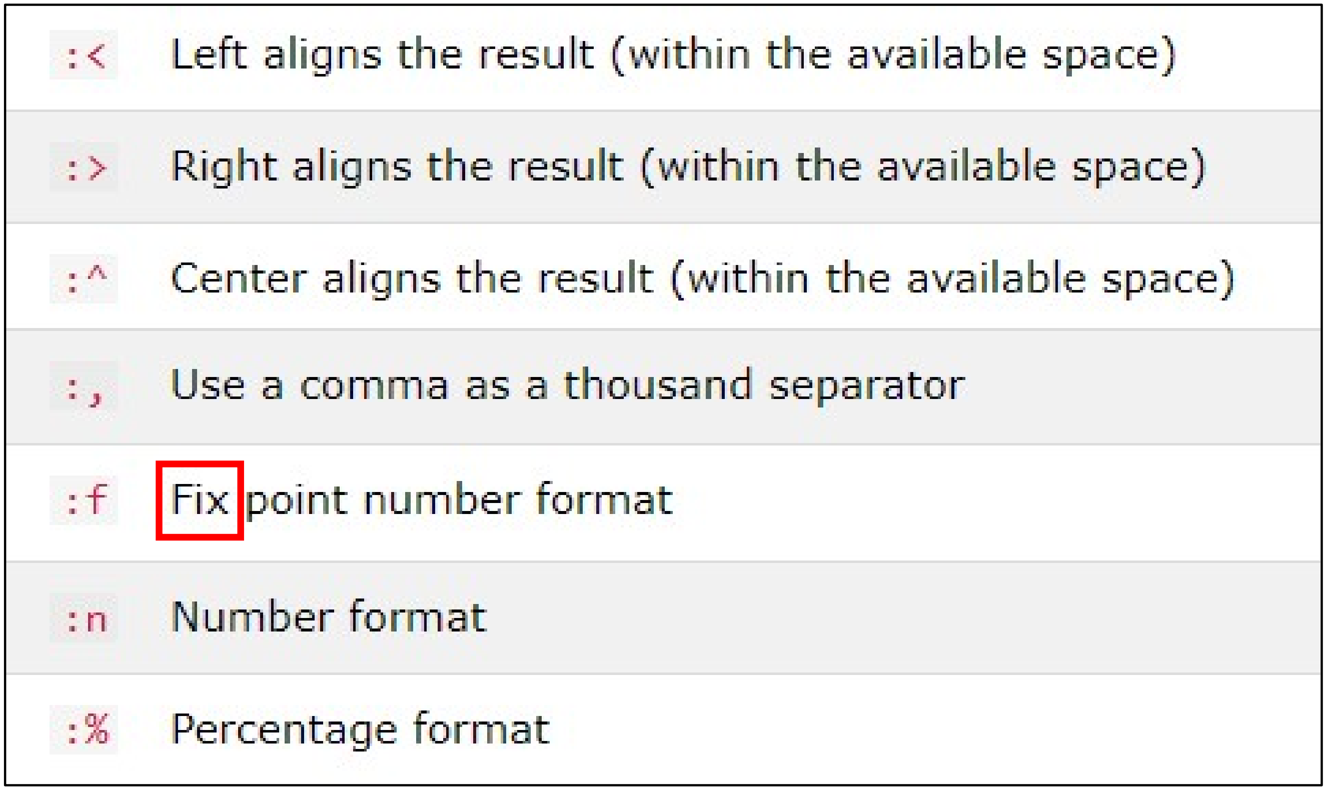
```
for fahrenheit in range(-44, 217, 4):
    celsius = (fahrenheit - 32) * 5 / 9
    print(f"{fahrenheit:>6.2f} F = {celsius:>6.2f} C")
```

Placing a lowercase **f** before the first quote in a **print**() statement indicates you will use some **placeholders**

A placeholder contains the variable's name sandwiched between curly **braces** {}

A **colon** after the variable's name starts a **format specifier**

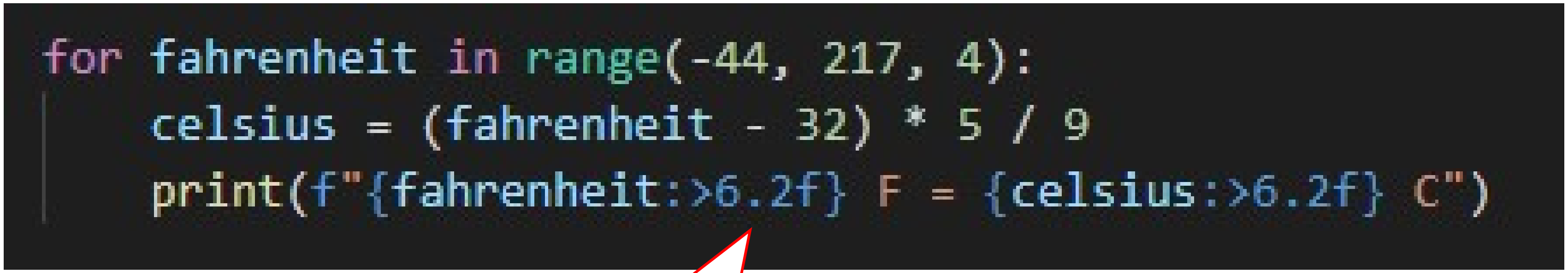
Some Common Format Specifiers



`:<`	Left aligns the result (within the available space)
`:>`	Right aligns the result (within the available space)
`:^`	Center aligns the result (within the available space)
`:,`	Use a comma as a thousand separator
`:f`	Fix point number format
`:n`	Number format
`:%`	Percentage format

Using format specifiers is optional but makes your output more professional

print() and **f-strings**

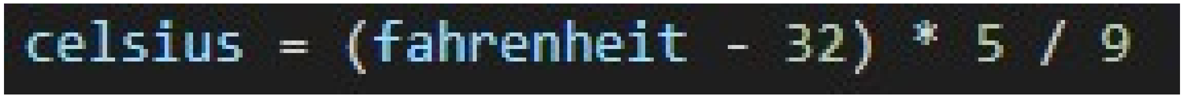


```python
for fahrenheit in range(-44, 217, 4):
    celsius = (fahrenheit - 32) * 5 / 9
    print(f"{fahrenheit:>6.2f} F = {celsius:>6.2f} C")
```

Print the current value of the variable **fahrenheit** right-justified (**>**) in a column six characters wide (**6**) and round to two digits (**.2f**) to the right of the decimal

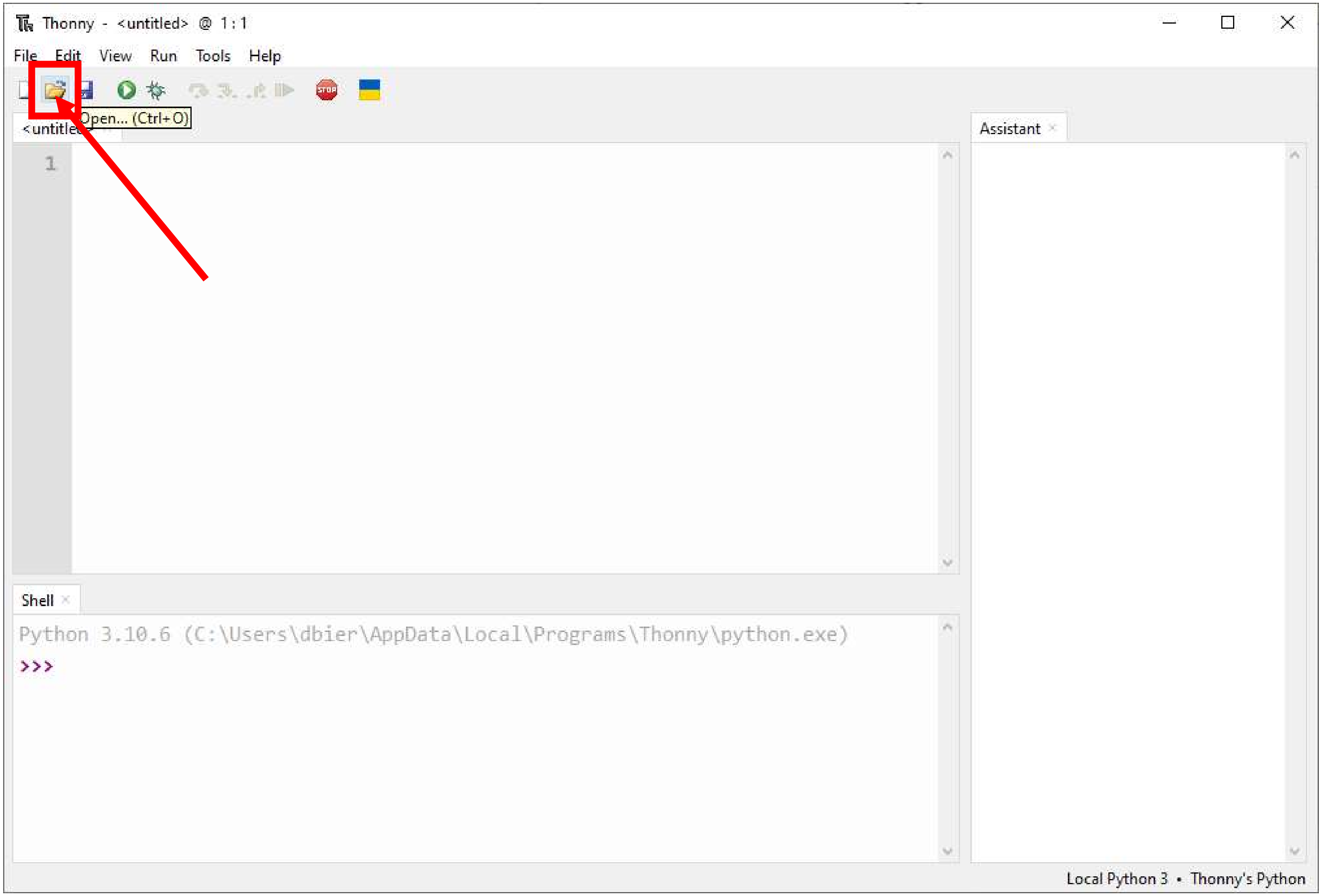
Mathematical Operators

- Python operators obey normal **PEMDAS** precedence
  - Expressions are evaluated left to right in your source code
  - Use **=** to assign a value to a variable
  - Use ***** for multiplication and **/** for division operators
  - Use parenthesis to explicit specify the order of operations
  - The “greater than or equal to” operator is **>=**
  - The “less than or equal to” operator is **<=**



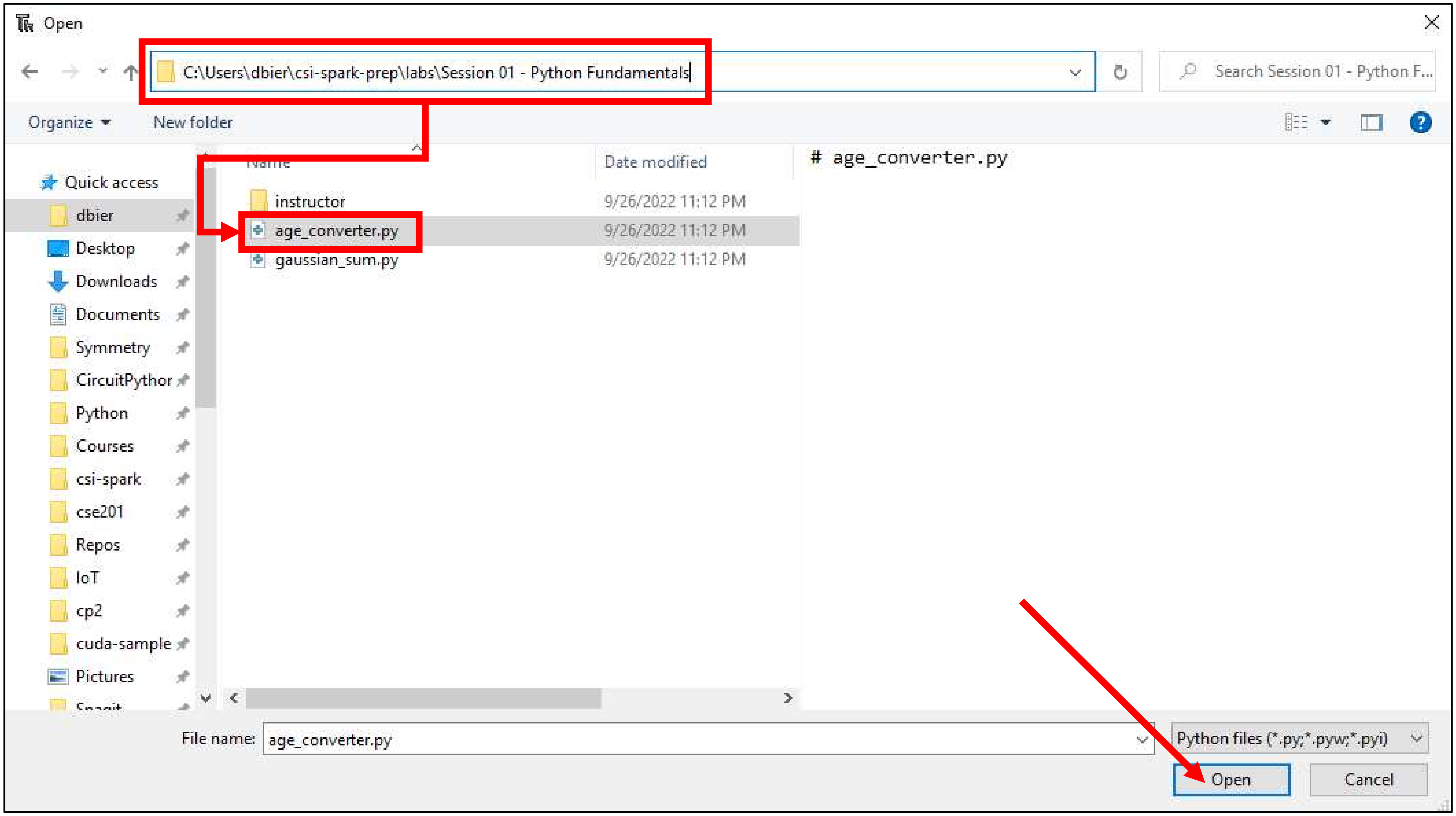
```
celsius = (fahrenheit - 32) * 5 / 9
```

Open age_converter.py



```
Python 3.10.6 (C:\Users\dbier\AppData\Local\Programs\Thonny\python.exe)
>>>
```

Open age_converter.py



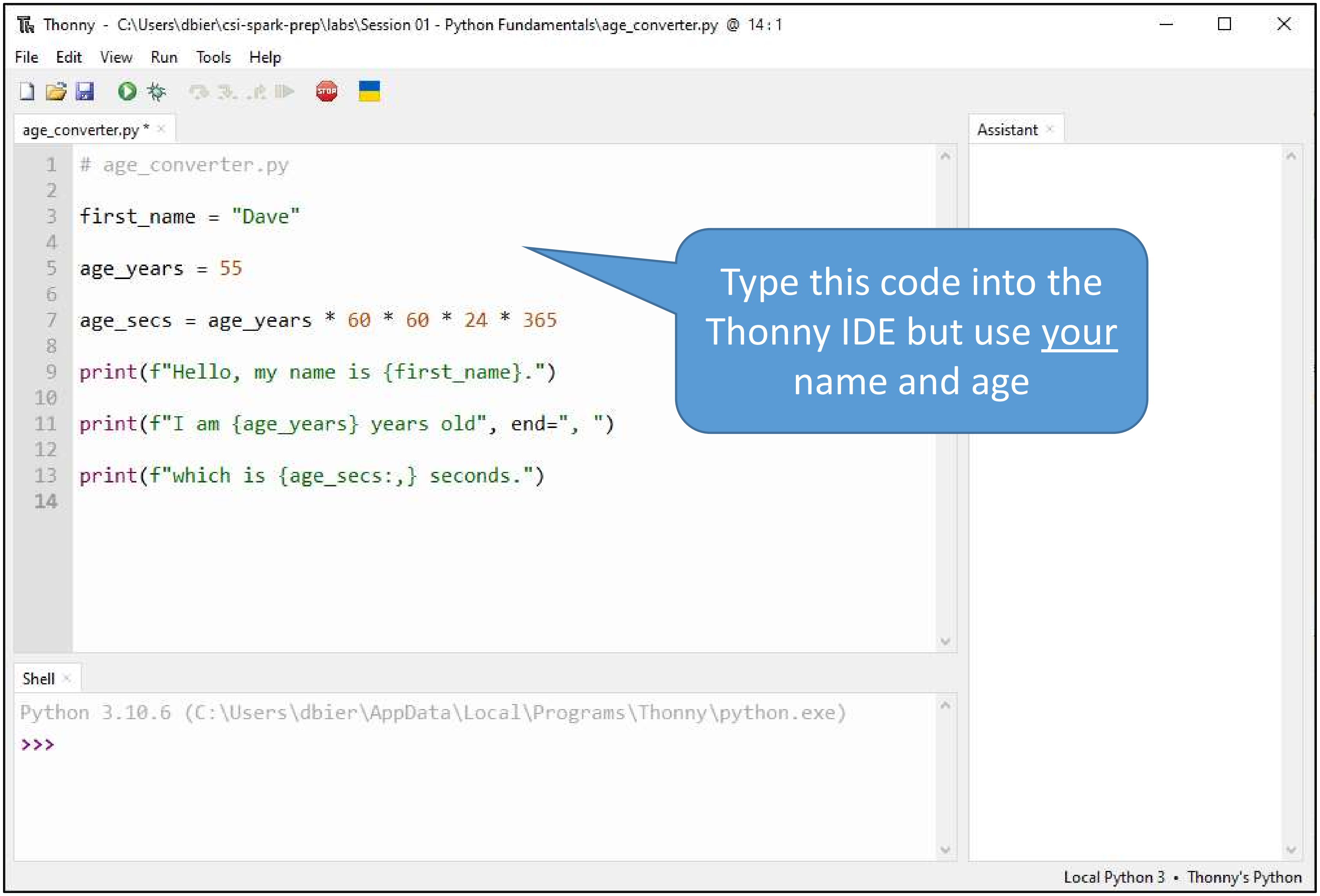
Name	Date modified
instructor	9/26/2022 11:12 PM
age_converter.py	9/26/2022 11:12 PM
gaussian_sum.py	9/26/2022 11:12 PM

```
# age_converter.py
```

File name: age_converter.py

Python files (*.py;*.pyw;*.pyi)

Edit age_converter.py

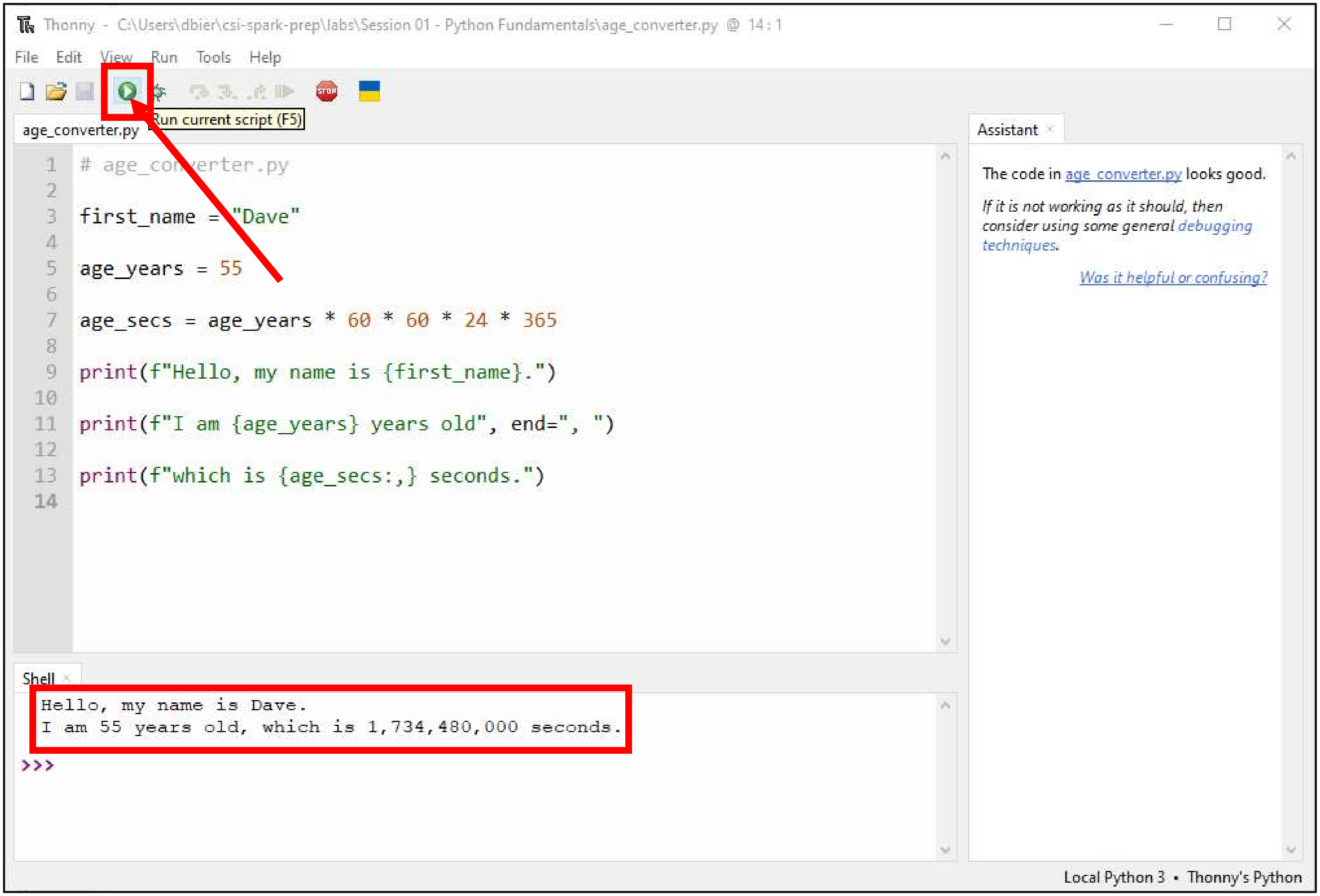

Thonny - C:\Users\dbier\csi-spark-prep\labs\Session 01 - Python Fundamentals\age_converter.py @ 14 : 1
File Edit View Run Tools Help
age_converter.py *
Assistant
Shell
Local Python 3 • Thonny's Python

```
# age_converter.py

first_name = "Dave"

age_years = 55

age_secs = age_years * 60 * 60 * 24 * 365

print(f"Hello, my name is {first_name}.")

print(f"I am {age_years} years old", end=", ")

print(f"which is {age_secs:,} seconds.")

```

```
Python 3.10.6 (C:\Users\dbier\AppData\Local\Programs\Thonny\python.exe)
>>>
```

Type this code into the Thonny IDE but use your name and age

Run age_converter.py


Thonny - C:\Users\dbier\csi-spark-prep\labs\Session 01 - Python Fundamentals\age_converter.py @ 14 : 1
File
Edit
View
Run
Tools
Help
Run current script (F5)
age_converter.py
Shell
Assistant
Local Python 3 • Thonny's Python

```python
# age_converter.py

first_name = "Dave"

age_years = 55

age_secs = age_years * 60 * 60 * 24 * 365

print(f"Hello, my name is {first_name}.")

print(f"I am {age_years} years old", end=", ")

print(f"which is {age_secs:,} seconds.")
```

```
Hello, my name is Dave.
I am 55 years old, which is 1,734,480,000 seconds.
>>>
```

The code in age_converter.py looks good.

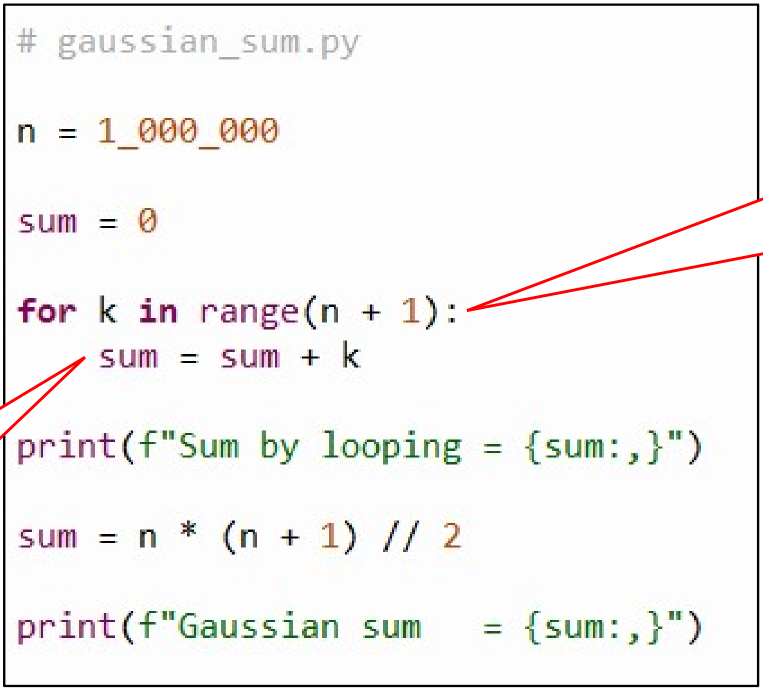
If it is not working as it should, then consider using some general debugging techniques.

Was it helpful or confusing?

Statements & Scopes

- A **statement** does something (think: *sentence*)
  - A statement is either a declaration, a keyword, or a function
  - Statements are executed from top to bottom of a scope
- A **scope** contains one or more statements (think: *paragraph*)
  - A scope in Python begins with a **colon :**
  - Scopes are also denoted via **indentation**
  - All the statements in a scope **must** start at the same **column**
  - Scopes can be *nested* – each inner scope is further indented
  - Certain Python statements "introduce" (**require**) a new scope
  - In Python, white space is significant – indentation matters!

Statements and Scopes



```
# gaussian_sum.py

n = 1_000_000

sum = 0

for k in range(n + 1):
    sum = sum + k

print(f"Sum by looping = {sum:,}")

sum = n * (n + 1) // 2

print(f"Gaussian sum   = {sum:,}")
```

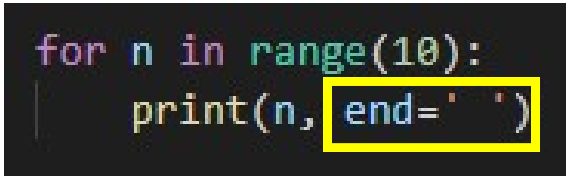
A **for** loop introduces a scope with a **colon :**

The statements in a **for** loop **scope** are all indented by the *same* amount

Identifiers *live* only within the scope in which they are defined
(except for globals & closures)

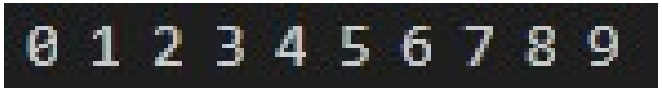
for loops

- A **for** loop executes all the statements within its scope for **each** item in the **list** passed *into* the **for**
- It is common to use the **range()** **function** to describe the list of numbers passed into the **for** statement
- The **range()** function takes three parameters: (**start**, **stop**, **step**)
  - The **stop** value is required but the **start** and **step** values are *optional*
  - The *default* value (if unspecified) for **start** is **0** and for **step** is **1**
  - The range is inclusive, *exclusive*: **[start, stop)** ←



```
for n in range(10):
    print(n, end=' ')
```

⇒



```
0 1 2 3 4 5 6 7 8 9
```

A Shortcut

Carl Friedrich Gauss

(1777 – 1855)

Sum the integers from 1 to 100

1
2
3
4
5
6
7
8
9
10

1	9
2	8
3	7
4	6
5	
10	

n = 10

4 matched rows that each sum to 10

1 row that is = 10 / 2 = 5

1 row that is = n = 10

$$n\left(\frac{n}{2}-1\right)+\frac{n}{2}+n \qquad = \frac{n*(n+1)}{2}$$

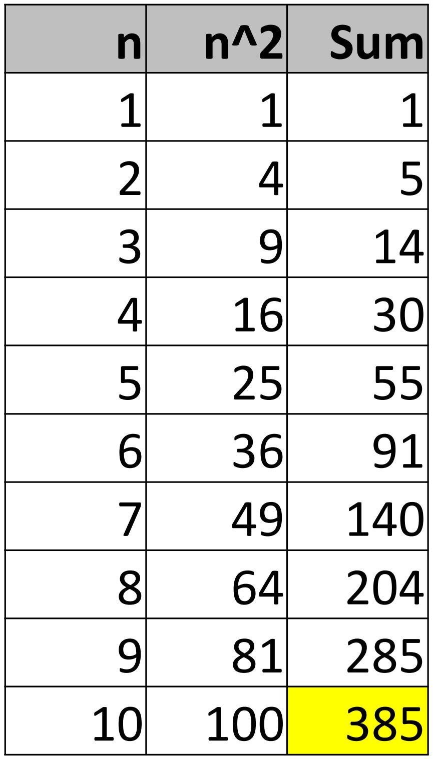
= 55

Another Shortcut

Sum of first **n** *natural* numbers:

$$\sum_{k=1}^{n} k = \frac{n(n+1)}{2},$$

Sum of squares of first **n** *natural* numbers:

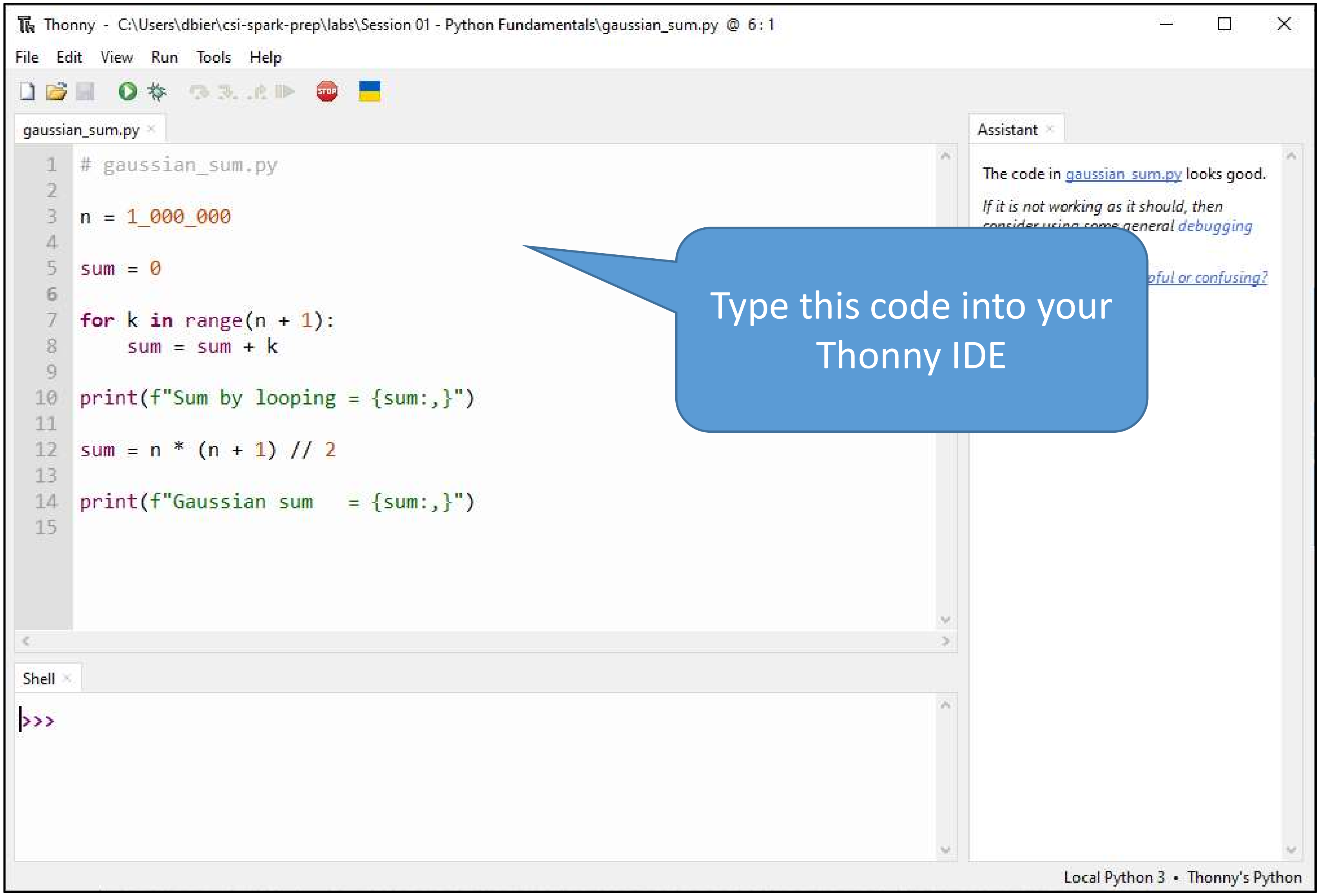


n	n^2	Sum
1	1	1
2	4	5
3	9	14
4	16	30
5	25	55
6	36	91
7	49	140
8	64	204
9	81	285
10	100	385

$$P_n = \sum_{k=1}^{n} k^2 = \frac{n(n+1)(2n+1)}{6} = \boxed{\frac{2n^3+3n^2+n}{6}.}$$

These are functional equations - we can now calculate the sums *immediately* without having to loop over every element!

Edit gaussian_sum.py



Thonny - C:\Users\dbier\csi-spark-prep\labs\Session 01 - Python Fundamentals\gaussian_sum.py @ 6 : 1

```
# gaussian_sum.py

n = 1_000_000

sum = 0

for k in range(n + 1):
    sum = sum + k

print(f"Sum by looping = {sum:,}")

sum = n * (n + 1) // 2

print(f"Gaussian sum   = {sum:,}")
```

The code in gaussian_sum.py looks good.

If it is not working as it should, then consider using some general debugging

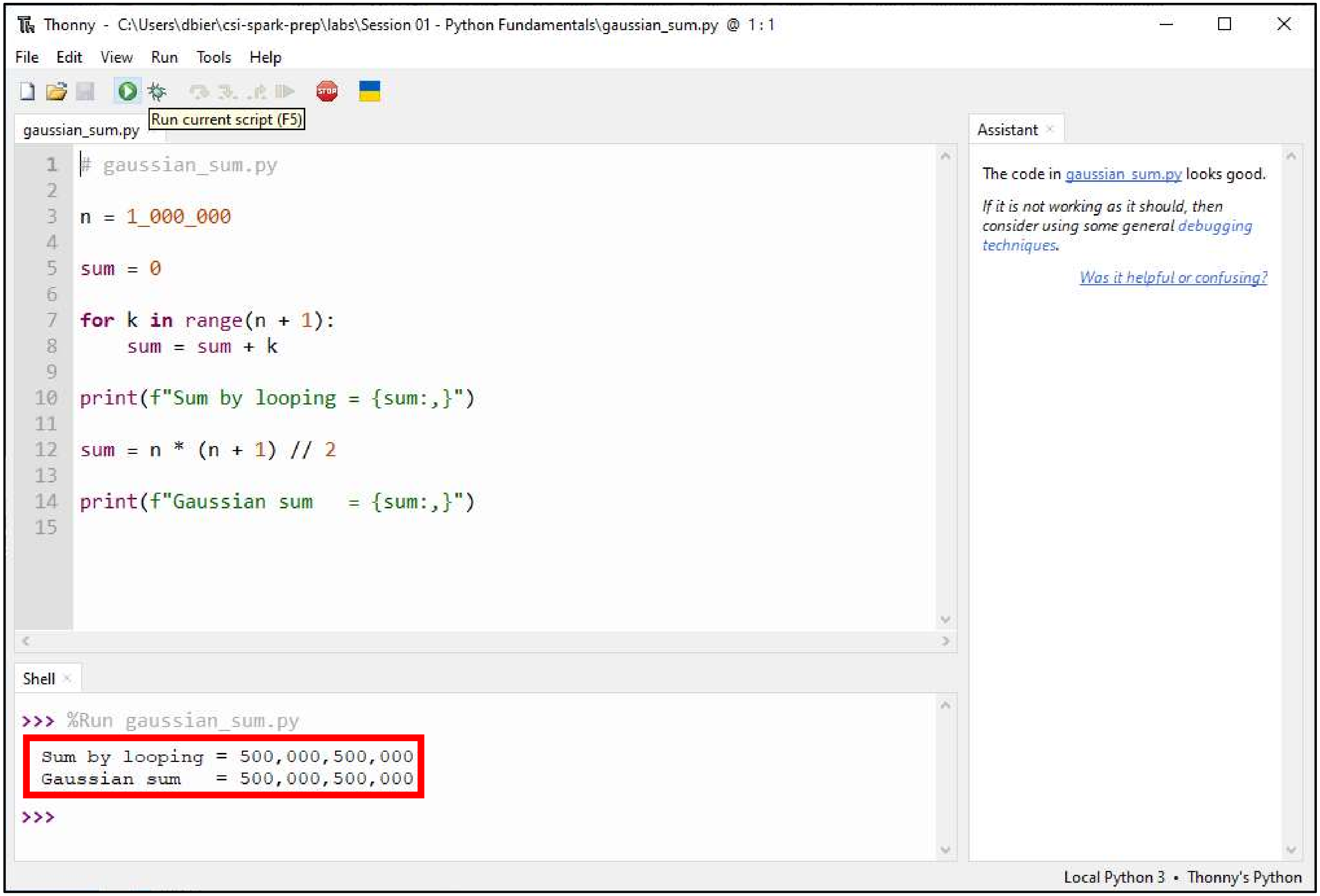
pful or confusing?

```
>>>
```

Local Python 3 • Thonny's Python

Type this code into your Thonny IDE

Run gaussian_sum.py


Thonny - C:\Users\dbier\csi-spark-prep\labs\Session 01 - Python Fundamentals\gaussian_sum.py @ 1 : 1
File
Edit
View
Run
Tools
Help
Run current script (F5)
gaussian_sum.py
Assistant
Shell
Local Python 3 • Thonny's Python

```python
# gaussian_sum.py

n = 1_000_000

sum = 0

for k in range(n + 1):
    sum = sum + k

print(f"Sum by looping = {sum:,}")

sum = n * (n + 1) // 2

print(f"Gaussian sum   = {sum:,}")
```

The code in gaussian_sum.py looks good.

If it is not working as it should, then consider using some general debugging techniques.

Was it helpful or confusing?

```
>>> %Run gaussian_sum.py
 Sum by looping = 500,000,500,000
 Gaussian sum   = 500,000,500,000
>>>
```

Session **01** – Now You Know…

- How to install packages and plug-ins into the Thonny IDE
- How to define variables in Python (**snake_case**)
- How to use **print**() to show variable values on screen
- How to use **f-strings** with **{}** placeholders (replacement fields) for variable names
- How to indicate **format specifiers** in order to make your code output more human readable and professional looking
- How to pass the lazy list generated by **range()** into the **for** statement to enumerate each value in that sequence

TASK 01-01

- Create a console (terminal/text mode) Python program called **celsius_to_fahrenheit.py** that converts a range of temperatures in the Celsius scale to the equivalent temperatures in the Fahrenheit scale
- The program should display all temperatures between -44°C and 106°C inclusive in steps of 4°C and the corresponding temperature in Fahrenheit
- Each C/F pair should be displayed on its own output line, with two digits to the right of the decimal for each temperature scale
- Verify the correctness of your program by checking your values for -40°C, 0°C, and 100°C