Beginner's Python Workshop with Thonny

Welcome to Python programming! This workshop will teach you the fundamentals of Python using the Thonny editor. Python is a powerful yet beginner-friendly programming language used for web development, data analysis, automation, artificial intelligence, and much more.

By the end of this workshop, you'll be able to write simple Python programs, understand basic programming concepts, and have the foundation to continue learning on your own.

Introduction: Setting Up Your Environment

What is Thonny?

Thonny is a beginner-friendly Python editor specifically designed for learning. Unlike complex development environments, Thonny keeps things simple while providing everything you need to write and run Python code.

Installing Thonny

Step 1: Download Thonny

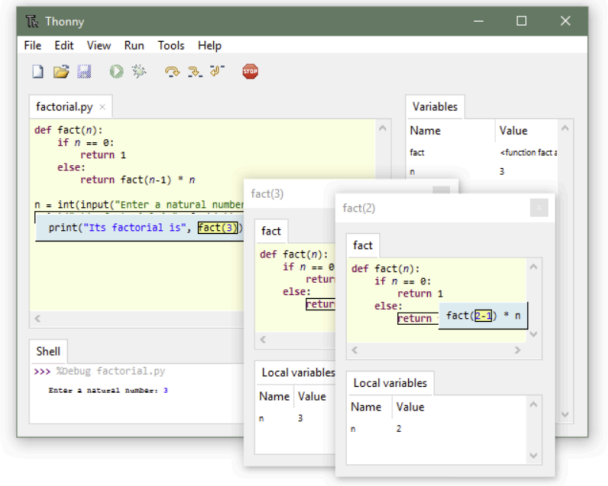
1. Go to https://thonny.org/
2. Click the download button for your operating system (Windows, Mac, or Linux)
3. The website automatically detects your system

Thonny 4 is dedicated to Ukraine fighting the Russian invasion.
🇺🇦 Please support Ukraine! 🇺🇦

Thonny

Python IDE for beginners

Download version 4.1.7 for Windows • Mac • Linux



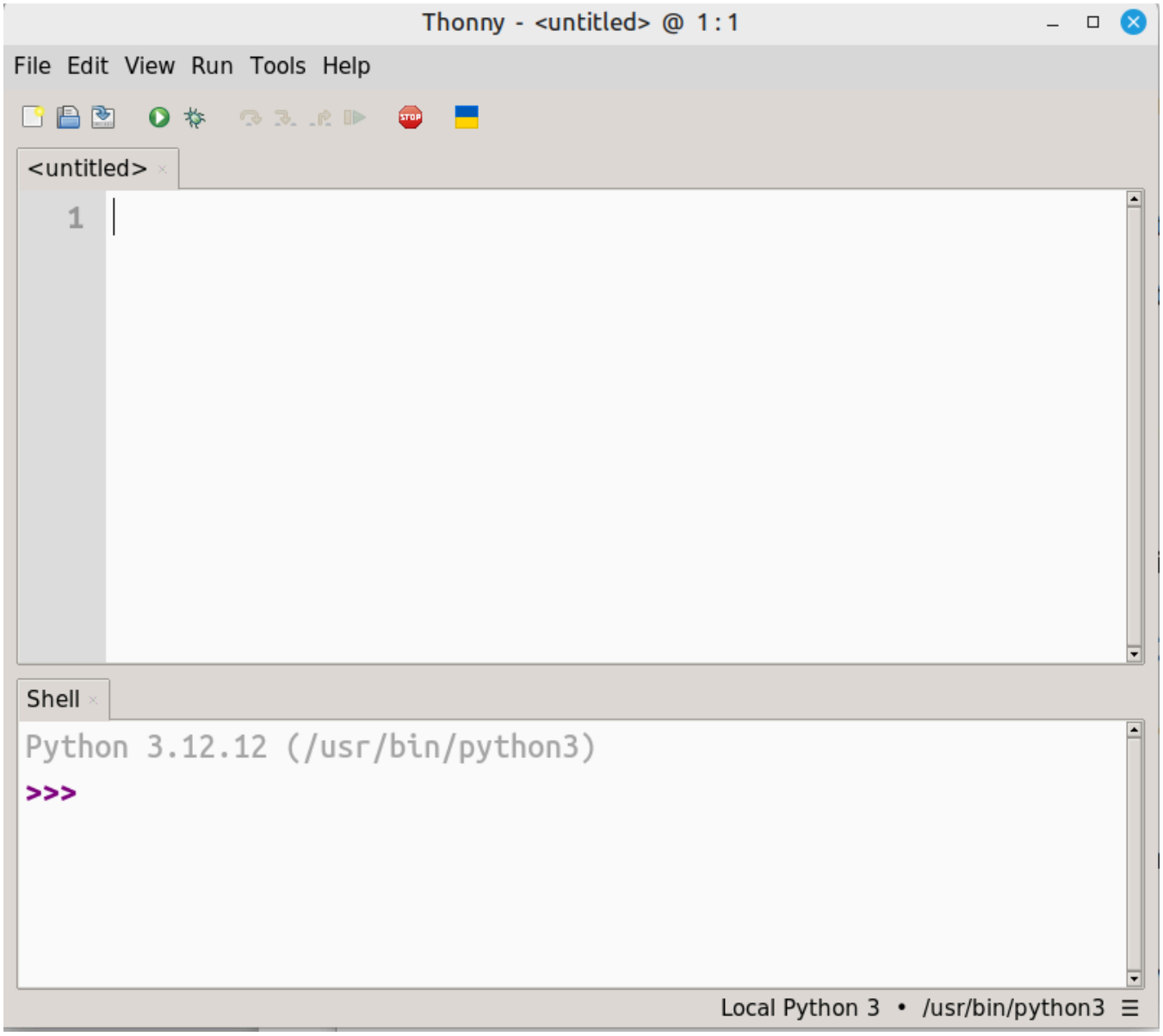
Step 2: Install Thonny

1. Run the downloaded installer
2. Follow the installation wizard
3. Accept the default settings (Thonny will install Python automatically if you don't have it)

[Screenshot: Thonny installation wizard]

Step 3: Launch Thonny

1. Open Thonny from your applications menu or desktop shortcut
2. You should see the Thonny interface



Understanding the Thonny Interface

When you open Thonny, you'll see:

- **Editor Area (top)**: Where you write your code
- **Shell/Console (bottom)**: Where you see results and interact with Python
- **Toolbar**: Contains the Run button (green arrow) and other tools
- **Menu Bar**: File, Edit, View, Run, Tools, Help

Your First Program

Let's make sure everything works!

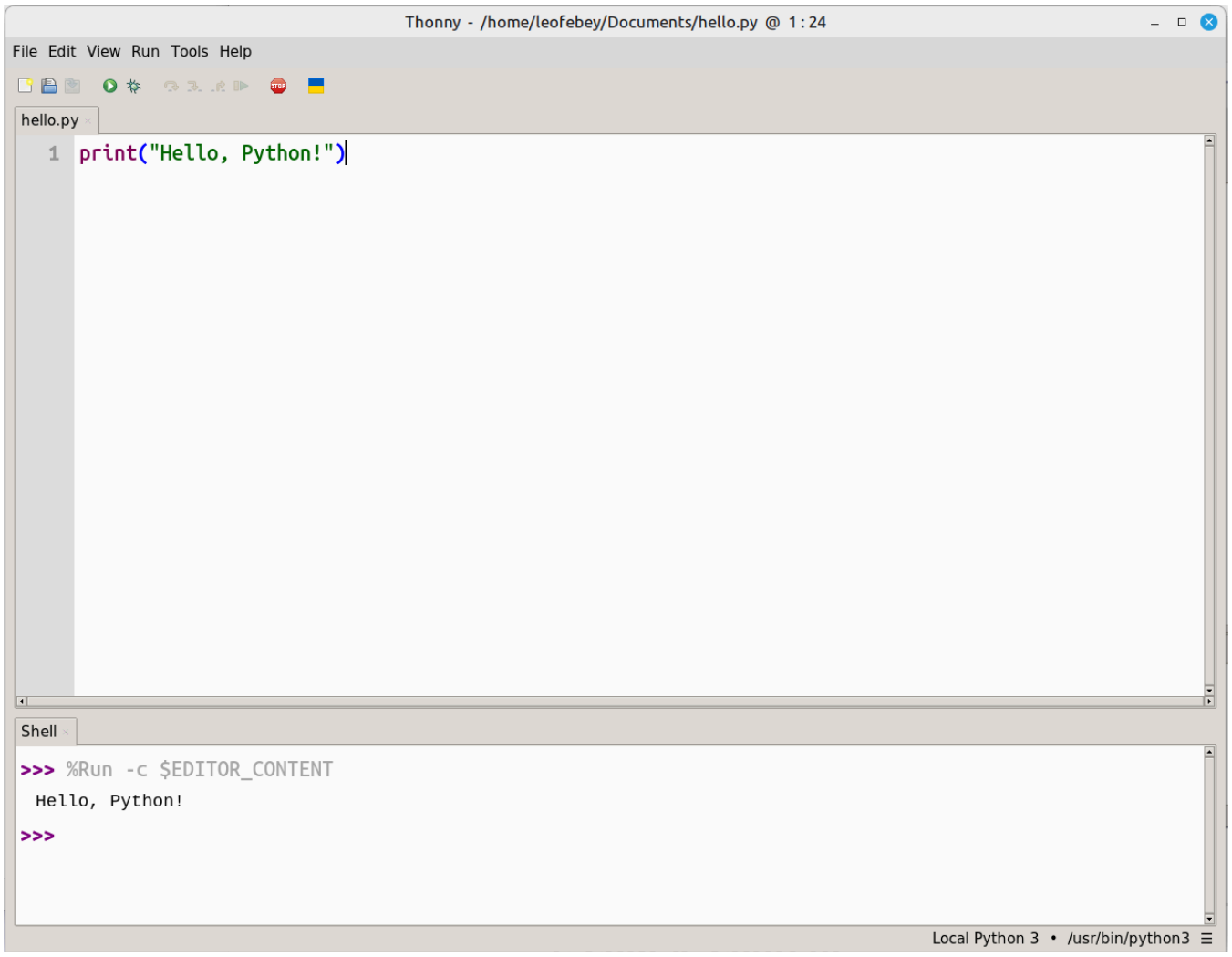
1. In the editor area (top), type:

```
print("Hello, Python!")
```

2. Click the green **Run** button (or press **F5**)

3. You'll be asked to save the file - save it as `hello.py`
4. Look at the Shell area - you should see:

```
Hello, Python!
```



Congratulations! You've just written and run your first Python program!

---

1. Course Content

Getting Started with Thonny

To run your code: Press **F5** or click the green **Run** button.

Tips:

- Save your work frequently (Ctrl+S or Cmd+S)
- Thonny will prompt you to save before running
- Use descriptive filenames like `calculator.py` or `shopping_list.py`

Chapter 1: Your First Python Program

Printing Text

Let's start with displaying text on the screen:

```
print("Hello, World!")
```

Run this code. You should see `Hello, World!` appear in the Shell below.

Try printing your own message:

```
print("My name is Sam")
print("I am learning Python!")
```

Complete code for this section:

```
print("Hello, World!")
print("My name is Sam")
print("I am learning Python!")
```

Chapter 2: Variables and Numbers

Storing Values

Variables are like labeled boxes that store information:

```
age = 25
```

This creates a variable called `age` and stores the number 25 in it.

```
name = "Alex"
```

This stores text (called a "string") in a variable called `name`.

Basic Math

Python can do calculations:

```
result = 10 + 5
print(result)
```

This will display `15`.

Try different operations:

```
addition = 10 + 5
subtraction = 20 - 7
multiplication = 6 * 4
division = 15 / 3

print(addition)
print(subtraction)
print(multiplication)
print(division)
```

Combining Text and Variables

```
name = "Jordan"
age = 30
print("My name is " + name)
print("I am " + str(age) + " years old")
```

Note: We use `str()` to convert numbers to text so we can combine them.

Complete code for this section:

```
age = 25
name = "Alex"
result = 10 + 5
print(result)

addition = 10 + 5
subtraction = 20 - 7
multiplication = 6 * 4
division = 15 / 3

print(addition)
print(subtraction)
print(multiplication)
print(division)

name = "Jordan"
age = 30
print("My name is " + name)
print("I am " + str(age) + " years old")
```

Chapter 3: Getting Input from Users

The input() Function

You can ask users to type something:

```
name = input("What is your name? ")
print("Hello, " + name + "!")
```

When you run this, Python will wait for you to type something and press Enter.

Working with Number Input

```
age_text = input("How old are you? ")
age = int(age_text)
next_year = age + 1
print("Next year you will be " + str(next_year))
```

We use `int()` to convert text input into a number we can do math with.

Complete code for this section:

```
name = input("What is your name? ")
print("Hello, " + name + "!")

age_text = input("How old are you? ")
age = int(age_text)
next_year = age + 1
print("Next year you will be " + str(next_year))
```

Chapter 4: Making Decisions with if Statements

Basic if Statement

Programs can make decisions:

```
temperature = 25

if temperature > 20:
    print("It's warm outside!")
```

Notice the colon `:` at the end and the indentation (spaces) before `print`. This is very important in Python!

if-else Statements

```
age = 16

if age >= 18:
    print("You are an adult")
else:
    print("You are a minor")
```

if-elif-else for Multiple Conditions

```
score = 75

if score >= 90:
    print("Grade: A")
elif score >= 80:
    print("Grade: B")
elif score >= 70:
    print("Grade: C")
else:
    print("Grade: F")
```

Complete code for this section:

```
temperature = 25

if temperature > 20:
    print("It's warm outside!")

age = 16

if age >= 18:
    print("You are an adult")
else:
    print("You are a minor")

score = 75

if score >= 90:
    print("Grade: A")
elif score >= 80:
    print("Grade: B")
elif score >= 70:
    print("Grade: C")
else:
    print("Grade: F")
```

Chapter 5: Repeating Actions with Loops

The while Loop

Repeats actions while a condition is true:

```
count = 1

while count <= 5:
    print("Count is: " + str(count))
    count = count + 1

print("Loop finished!")
```

This prints numbers 1 through 5.

The for Loop

Used to repeat a specific number of times:

```
for number in range(5):
    print("Number: " + str(number))
```

This prints 0, 1, 2, 3, 4 (5 numbers starting from 0).

To start from 1:

```
for number in range(1, 6):
    print("Number: " + str(number))
```

Complete code for this section:

```
count = 1

while count <= 5:
    print("Count is: " + str(count))
    count = count + 1

print("Loop finished!")

for number in range(5):
    print("Number: " + str(number))

for number in range(1, 6):
    print("Number: " + str(number))
```

Chapter 6: Lists - Storing Multiple Values

Creating Lists

Lists store multiple items in one variable:

```
fruits = ["apple", "banana", "orange"]
print(fruits)
```

Accessing List Items

```
fruits = ["apple", "banana", "orange"]

print(fruits[0])  # First item: apple
print(fruits[1])  # Second item: banana
print(fruits[2])  # Third item: orange
```

Python counts from 0!

Adding to Lists

```
fruits = ["apple", "banana"]
fruits.append("orange")
print(fruits)
```

Looping Through Lists

```
fruits = ["apple", "banana", "orange"]

for fruit in fruits:
    print("I like " + fruit)
```

Complete code for this section:

```
fruits = ["apple", "banana", "orange"]
print(fruits)

print(fruits[0])
print(fruits[1])
print(fruits[2])

fruits = ["apple", "banana"]
fruits.append("orange")
print(fruits)

fruits = ["apple", "banana", "orange"]

for fruit in fruits:
    print("I like " + fruit)
```

Chapter 7: Functions - Reusable Code

Creating Simple Functions

Functions are blocks of code you can reuse:

```
def greet():
    print("Hello!")
    print("Welcome to Python!")

greet()
greet()
```

This prints the greeting twice.

Functions with Parameters

```
def greet_person(name):
    print("Hello, " + name + "!")

greet_person("Alice")
greet_person("Bob")
```

Functions that Return Values

```
def add_numbers(a, b):
    result = a + b
    return result

answer = add_numbers(5, 3)
print("The sum is: " + str(answer))
```

Complete code for this section:

```
def greet():
    print("Hello!")
    print("Welcome to Python!")

greet()
greet()

def greet_person(name):
    print("Hello, " + name + "!")

greet_person("Alice")
greet_person("Bob")

def add_numbers(a, b):
    result = a + b
    return result

answer = add_numbers(5, 3)
print("The sum is: " + str(answer))
```

---

2. Demos

Demo 1: Installing and Using External Packages from PyPI

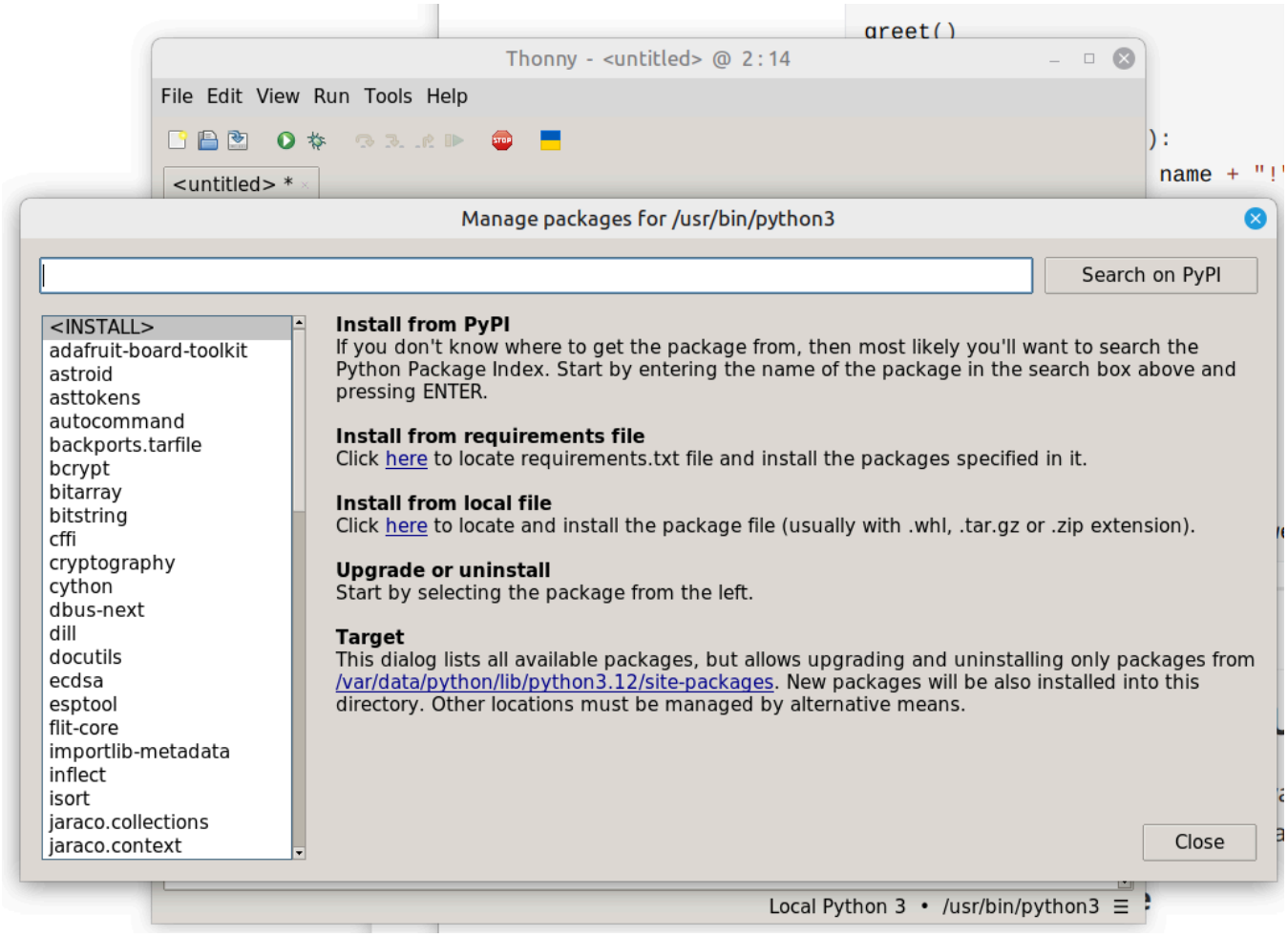
Python has thousands of packages available on PyPI (Python Package Index) that you can install to add extra features. Let's learn how to install a package and use it. On Windows and MacOS this is quite straightforward, however on Linux it is more complex and you might need assistance.

Installing a Package

We'll install the `emoji` package, which lets us easily work with emojis in our code.

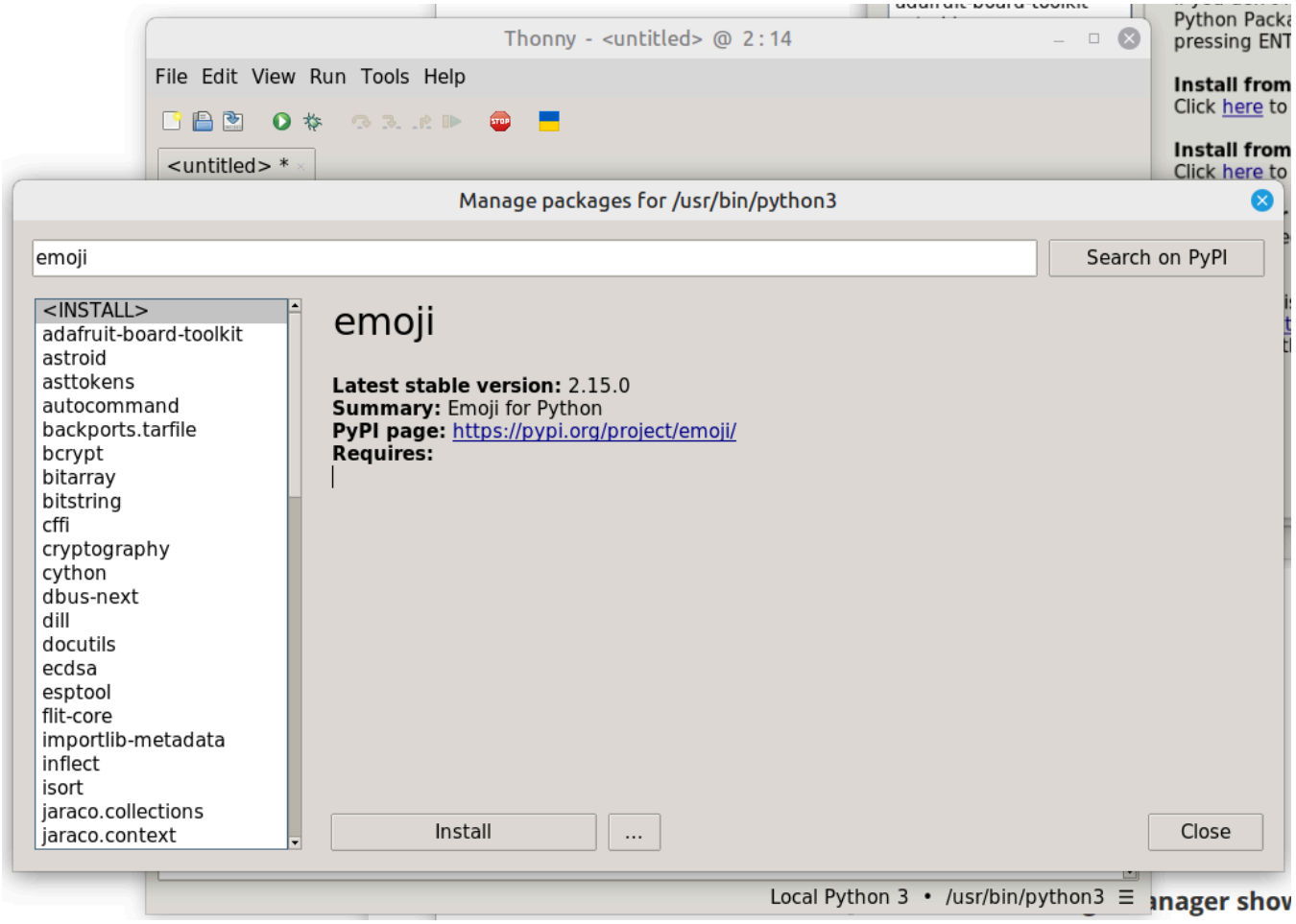
Step 1: Open the Package Manager

1. In Thonny, click **Tools** in the menu bar
2. Select **Manage packages...**



Step 2: Search and Install

1. In the search box, type: `emoji`
2. Click **Find package from PyPI**
3. Click the **Install** button
4. Wait for installation to complete



Step 3: Close the Package Manager

Using the Installed Package

Now let's use the emoji package in our code:

```
import emoji

# Print text with emojis
print(emoji.emojize("Hello! :snake: Welcome to Python! :rocket:"))
print(emoji.emojize("Python is fun! :fire: :star: :thumbs_up:"))

# Get user input and add emojis
name = input("What's your name? ")
print(emoji.emojize("Nice to meet you, " + name + "! :waving_hand:"))
```

[Screenshot: Code running with emoji output displayed]

What's happening:

- `import emoji` loads the emoji package we installed
- `emoji.emojize()` converts text like `:snake:` into actual emoji symbols
- You can find emoji codes at: https://carpedm20.github.io/emoji/

Try modifying:

- Use different emoji codes

- Create a mood tracker that prints different emojis based on user input
- Make a fun greeting card with multiple emojis

Other Useful Packages from PyPI

Once you're comfortable, try exploring:

- `requests` - For downloading data from websites
- `pillow` - For working with images
- `matplotlib` - For creating graphs and charts
- `pygame` - For making simple games

Demo 2: FractPy

We will use a basic fractal art generator package to generate fractals with code!

Install the package

Using the same steps as in Demo 1, install the `fractpy` package.

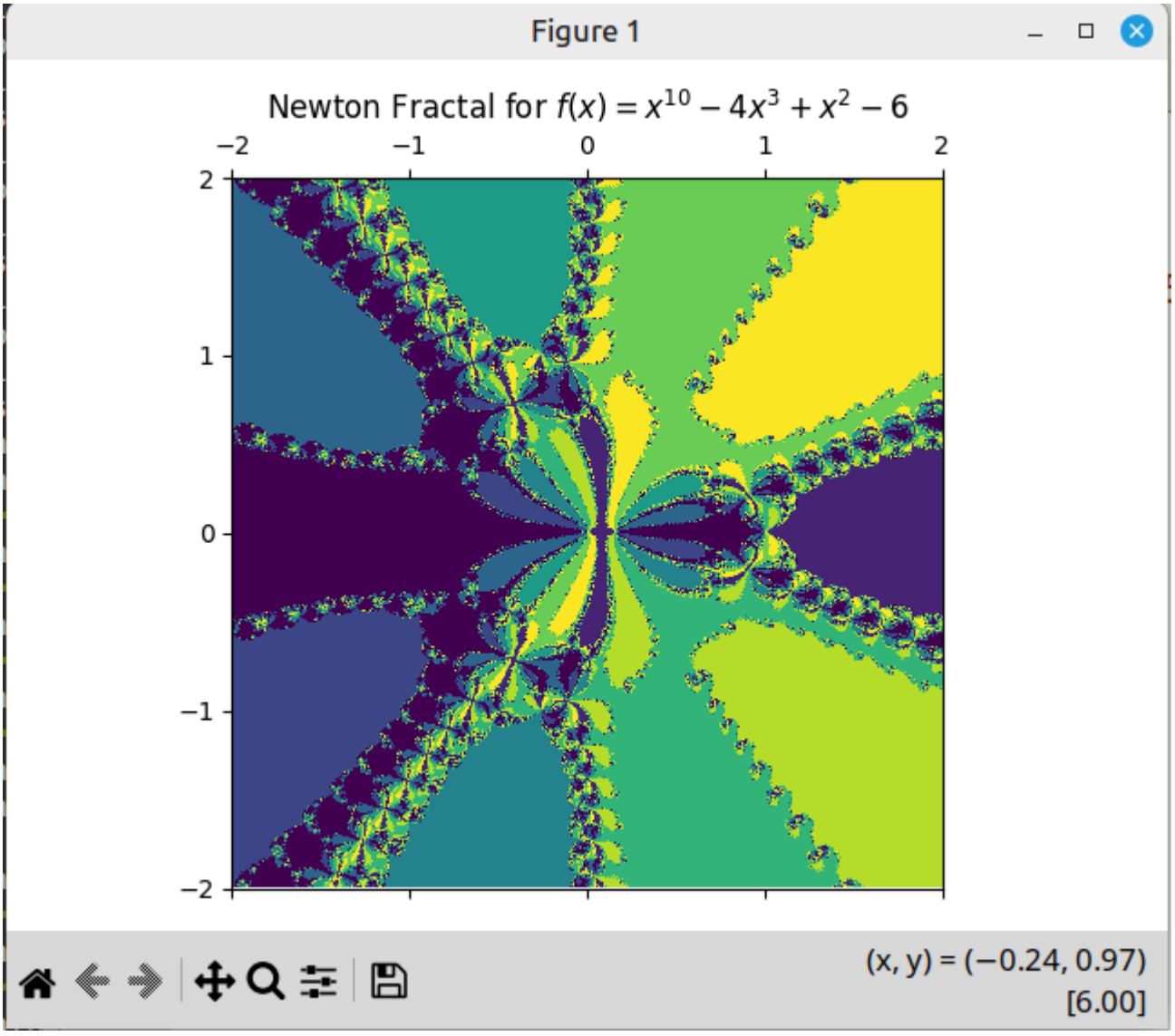
Copy the code

The code is available on my website: https://leofebeytech.com.au/services/workshops/

Then click "Beginner Python Workshop"

```
from fractpy.models import NewtonFractal
model = NewtonFractal("x**10 - 4x**3 + x**2 - 6")
p = model.plot(-2, 2, -2, 2, (1000, 1000))
p.savefig('fractal.jpg', format='jpeg', dpi=150, bbox_inches='tight')
p.show()
```

Click Run to see the preview



Try modifying

- The input - eg `x**8 - 3x**3 + x**2 - 4`
- Use a zoom plot instead `p = model.zoom_plot()`

Demo 3: OpenCV Face Detection

Python can be used to do some very technically impressive things, like any other programming language. OpenCV is a computer vision library which can be used to detect things in images, such as faces, limbs, objects. It uses a variety of algorithms and methods in the background, in many cases very efficiently such that it can run on lower-end devices.

Install the package

As before, install the `opencv-python` package. It might take a minute or so to download and install.

Copy and past the code

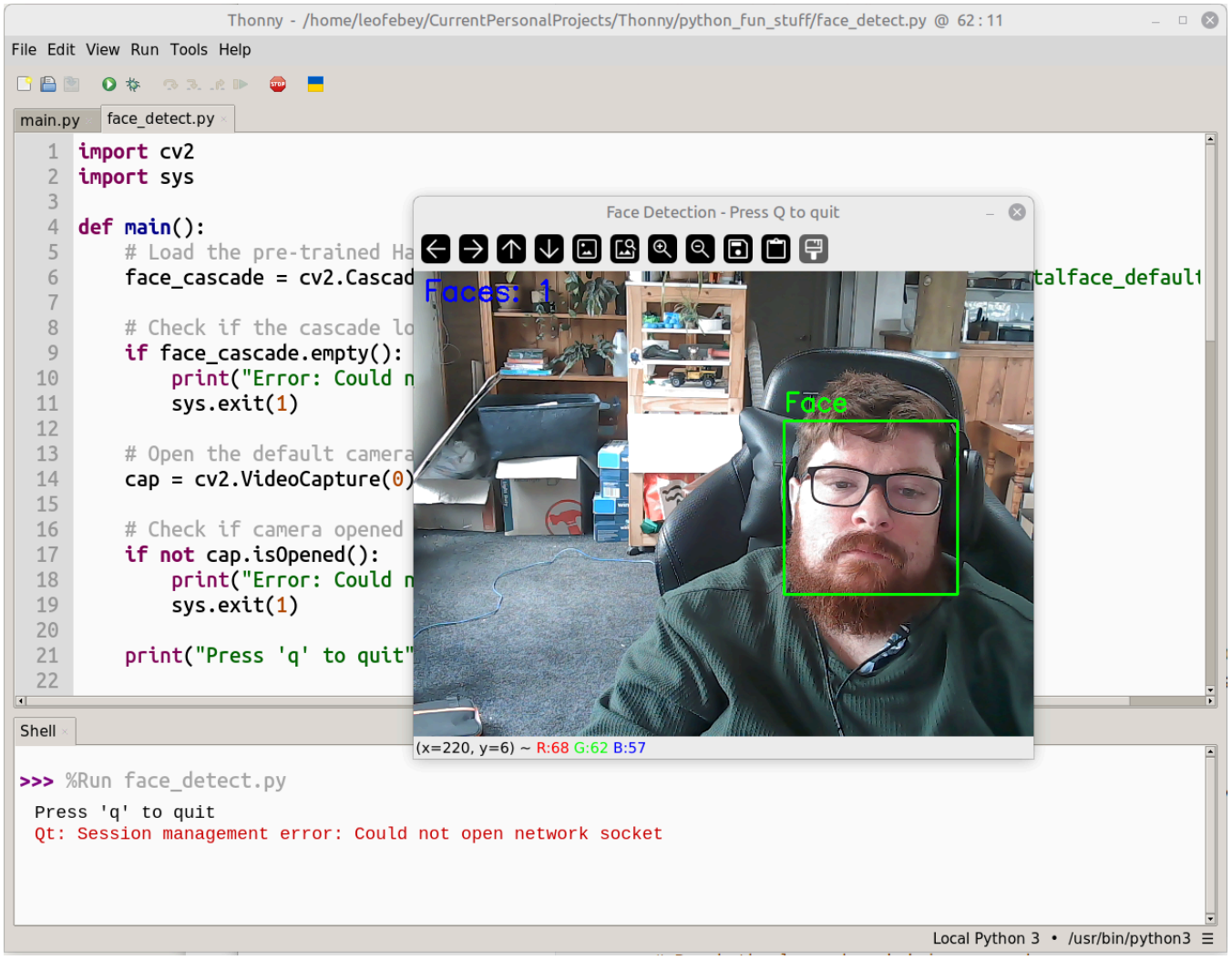
This is a lot more code than before, but you can find it on the website as above, to then copy and past into Thonny.

```
import cv2
import sys

def main():
    # Load the pre-trained Haar Cascade classifier for face detection
    face_cascade = cv2.CascadeClassifier(cv2.data.haarcascades +
'haarcascade_frontalface_default.xml')

    # Check if the cascade loaded successfully
    if face_cascade.empty():
        print("Error: Could not load face cascade classifier")
        sys.exit(1)

    # Open the default camera (0)
    cap = cv2.VideoCapture(0)

    # Check if camera opened successfully
    if not cap.isOpened():
        print("Error: Could not open camera")
        sys.exit(1)

    print("Press 'q' to quit")

    while True:
        # Capture frame-by-frame
        ret, frame = cap.read()

        if not ret:
            print("Error: Could not read frame")
            break

        # Convert to grayscale for face detection
        gray = cv2.cvtColor(frame, cv2.COLOR_BGR2GRAY)

        # Detect faces in the frame
        faces = face_cascade.detectMultiScale(
            gray,
            scaleFactor=1.1,
            minNeighbors=5,
            minSize=(30, 30)
        )

        # Draw rectangles around detected faces
        for (x, y, w, h) in faces:
            cv2.rectangle(frame, (x, y), (x+w, y+h), (0, 255, 0), 2)
            cv2.putText(frame, 'Face', (x, y-10), cv2.FONT_HERSHEY_SIMPLEX, 0.9, (0, 255,
0), 2)
```

```
        # Display the number of faces detected
        cv2.putText(frame, f'Faces: {len(faces)}', (10, 30), cv2.FONT_HERSHEY_SIMPLEX, 1,
(255, 0, 0), 2)

        # Display the resulting frame
        cv2.imshow('Face Detection - Press Q to quit', frame)

        # Break the loop when 'q' is pressed
        if cv2.waitKey(1) & 0xFF == ord('q'):
            break

    # Release the camera and close windows
    cap.release()
    cv2.destroyAllWindows()

if __name__ == "__main__":
    main()
```

Press run and preview

As you can see, it should detect a face. It will likely be a bit glitchy, and detect many false positives. However it could still be used in practical applications, such as detecting if a person is in a room, or some kind of automatic video streaming script when a person is present.



Modify it

The code uses the *Haar Cascade* classifier model, which in theory you can modify the training data to detect things other than faces. This is of course very advanced for this workshop.

---

3. Take Home Exercises

Exercise 1: Personal Greeting Program

Create a program that:

- Asks for the user's name
- Asks for their favorite color
- Prints a personalized message using both pieces of information

Example output:

```
What is your name? Sarah
What is your favorite color? blue
Hello Sarah! I like blue too!
```

Exercise 2: Simple Calculator

Create a calculator that:

- Asks the user for two numbers
- Adds them together
- Displays the result

Bonus: Also show subtraction, multiplication, and division.

Exercise 3: Temperature Checker

Create a program that:

- Asks for the current temperature
- If temperature is above 25, print "It's hot!"
- If temperature is between 15 and 25, print "It's pleasant"
- If temperature is below 15, print "It's cold!"

Exercise 4: Countdown Timer

Create a program that:

- Asks the user for a number
- Counts down from that number to 1

- Prints "Blast off!" at the end

Example:

```
Enter a number: 5
5
4
3
2
1
Blast off!
```

Exercise 5: Shopping List

Create a program that:

- Creates an empty list
- Asks the user to enter 3 items to add to their shopping list
- Displays all items in the list

Bonus: Print each item with a number (1. Bread, 2. Milk, etc.)

Exercise 6: Age Calculator Function

Create a function called `calculate_age_in_months` that:

- Takes age in years as a parameter
- Returns age in months (years × 12)
- Test it with different ages

---

4. Useful References

Python Basics

Official Python Documentation for Beginners

- Python Tutorial: https://docs.python.org/3/tutorial/
- Python for Beginners: https://www.python.org/about/gettingstarted/

Thonny Resources

- Thonny Homepage: https://thonny.org/
- Thonny User Guide: https://github.com/thonny/thonny/wiki

Quick Reference Code Snippets

Printing

```
print("text")
print(variable_name)
```

Variables

```
number_variable = 42
text_variable = "Hello"
```

User Input

```
text_input = input("Question? ")
number_input = int(input("Enter a number: "))
```

If Statements

```
if condition:
    # code here
elif other_condition:
    # code here
else:
    # code here
```

Loops

```
# While loop
while condition:
    # code here

# For loop
for item in range(10):
    # code here

# Loop through list
for item in my_list:
    # code here
```

Lists

```
my_list = [item1, item2, item3]
my_list.append(new_item)
my_list[0]  # First item
```

Functions

```
def function_name(parameter):
    # code here
    return value
```

Common Operators

```
+   # Addition
-   # Subtraction
*   # Multiplication
/   # Division
==  # Equal to
!=  # Not equal to
>   # Greater than
<   # Less than
>=  # Greater than or equal to
<=  # Less than or equal to
```

Learning Resources

Learning Resources

Interactive Practice

- Python Tutor (visualize code): https://pythontutor.com/
- Codecademy Python Course: https://www.codecademy.com/learn/learn-python-3
- W3Schools Python Tutorial: https://www.w3schools.com/python/

Video Tutorials

- Corey Schafer's Python Tutorials: https://www.youtube.com/c/Coreyms
- Programming with Mosh (Python for Beginners): https://www.youtube.com/watch?v=_uQrJ0TkZlc

Books and Guides

- "Automate the Boring Stuff with Python" (free online): https://automatetheboringstuff.com/
- "Python Crash Course" by Eric Matthes
- Think Python (free online): https://greenteapress.com/wp/think-python-2e/

Getting Help

When you get stuck:

1. Read the error message carefully - Python tells you what's wrong!
2. Check your indentation (spaces at the start of lines)
3. Make sure you spelled everything correctly
4. Use `print()` to check what your variables contain
5. Search for your error message online

6. Ask on Stack Overflow: https://stackoverflow.com/

Python Community

- r/learnpython (Reddit): https://www.reddit.com/r/learnpython/
- Python Discord: https://discord.gg/python
- Python Forum: https://python-forum.io/

Python Project Ideas and Libraries

Once you're comfortable with the basics, explore these exciting areas of Python:

Game Development

Pygame - Create 2D games

- Official Site: https://www.pygame.org/
- Documentation: https://www.pygame.org/docs/
- Tutorial: https://realpython.com/pygame-a-primer/
- Project Ideas: Pong, Snake, Platformers, Puzzle games
- Install: `pip install pygame` (or use Thonny's package manager)

Pygame Zero - Even simpler game development for beginners

- Official Site: https://pygame-zero.readthedocs.io/
- Great for learning game concepts without complex code
- Install: `pip install pgzero`

Arcade - Modern alternative to Pygame

- Official Site: https://api.arcade.academy/
- Clean, modern Python game library
- Install: `pip install arcade`

Mobile & Desktop Apps

Kivy - Build mobile apps and touch interfaces

- Official Site: https://kivy.org/
- Documentation: https://kivy.org/doc/stable/
- Create apps for Android, iOS, Windows, Mac, and Linux
- Great for: Touch interfaces, multi-touch apps, mobile games
- Install: `pip install kivy`

Tkinter - Built-in GUI library (comes with Python)

- Documentation: https://docs.python.org/3/library/tkinter.html
- Tutorial: https://realpython.com/python-gui-tkinter/

- Good for: Desktop applications, simple interfaces
- No installation needed!

PyQt or **PySide** - Professional desktop applications

- PyQt: https://www.riverbankcomputing.com/software/pyqt/
- PySide: https://wiki.qt.io/Qt_for_Python
- Used for professional-grade applications

Data Science & Visualization

Pandas - Data analysis and manipulation

- Official Site: https://pandas.pydata.org/
- Perfect for: Working with spreadsheets, CSV files, data analysis
- Install: `pip install pandas`

Matplotlib - Create graphs and charts

- Official Site: https://matplotlib.org/
- Make: Line graphs, bar charts, scatter plots, histograms
- Install: `pip install matplotlib`

Plotly - Interactive graphs and dashboards

- Official Site: https://plotly.com/python/
- Beautiful, interactive visualizations
- Install: `pip install plotly`

NumPy - Mathematical computing

- Official Site: https://numpy.org/
- Fast mathematical operations and arrays
- Install: `pip install numpy`

Machine Learning & AI

Scikit-learn - Beginner-friendly machine learning

- Official Site: https://scikit-learn.org/
- Documentation: https://scikit-learn.org/stable/user_guide.html
- Great for: Classification, regression, clustering
- Install: `pip install scikit-learn`

TensorFlow - Deep learning framework

- Official Site: https://www.tensorflow.org/
- Tutorial: https://www.tensorflow.org/tutorials
- Advanced: Neural networks, image recognition, NLP
- Install: `pip install tensorflow`

PyTorch - Another popular deep learning framework

- Official Site: https://pytorch.org/
- Tutorials: https://pytorch.org/tutorials/
- Popular in research and industry
- Install: `pip install torch`

OpenCV - Computer vision and image processing

- Official Site: https://opencv.org/
- Tutorial: https://docs.opencv.org/4.x/d6/d00/tutorial_py_root.html
- Projects: Face detection, object tracking, image filters
- Install: `pip install opencv-python`

Web Development

Flask - Lightweight web framework

- Official Site: https://flask.palletsprojects.com/
- Tutorial: https://flask.palletsprojects.com/en/stable/tutorial/
- Great for: Beginners, APIs, small to medium websites
- Install: `pip install flask`

Django - Full-featured web framework

- Official Site: https://www.djangoproject.com/
- Tutorial: https://docs.djangoproject.com/en/stable/intro/tutorial01/
- Used for: Large websites, content management systems
- Install: `pip install django`

FastAPI - Modern, fast web framework for APIs

- Official Site: https://fastapi.tiangolo.com/
- Great for: Building APIs quickly
- Install: `pip install fastapi`

Web Scraping & Automation

Beautiful Soup - Parse HTML and extract data from websites

- Documentation: https://www.crummy.com/software/BeautifulSoup/bs4/doc/
- Use for: Extracting data from web pages
- Install: `pip install beautifulsoup4`

Selenium - Automate web browsers

- Official Site: https://selenium-python.readthedocs.io/
- Use for: Web testing, automating repetitive browser tasks
- Install: `pip install selenium`

Requests - Simple HTTP requests

- Documentation: https://requests.readthedocs.io/
- Use for: Downloading web pages, working with APIs
- Install: `pip install requests`

File & Document Processing

Pillow (PIL) - Image processing

- Documentation: https://pillow.readthedocs.io/
- Use for: Resize, crop, filter images, create thumbnails
- Install: `pip install pillow`

PyPDF2 - Work with PDF files

- Documentation: https://pypdf2.readthedocs.io/
- Use for: Merge, split, extract text from PDFs
- Install: `pip install pypdf2`

openpyxl - Read and write Excel files

- Documentation: https://openpyxl.readthedocs.io/
- Use for: Automating Excel spreadsheet tasks
- Install: `pip install openpyxl`

Hardware & Electronics

Raspberry Pi Projects - Physical computing

- Official Site: https://www.raspberrypi.com/documentation/computers/getting-started.html
- Projects: Home automation, robots, sensors
- GPIO Library: https://gpiozero.readthedocs.io/

Arduino with Python - Control Arduino boards

- PySerial: https://pyserial.readthedocs.io/
- Firmata: https://github.com/tino/pyFirmata
- Projects: Robots, sensors, LED control

Fun & Creative Projects

Turtle Graphics - Built-in drawing library

- Documentation: https://docs.python.org/3/library/turtle.html
- Great for: Learning programming through art
- No installation needed!

PyGame Music/Audio - Sound and music

- Create music players, sound effects, audio tools

Discord Bots - Create your own Discord bot

- discord.py: https://discordpy.readthedocs.io/
- Install: `pip install discord.py`

Telegram Bots - Automate Telegram

- python-telegram-bot: https://python-telegram-bot.org/
- Install: `pip install python-telegram-bot`

Project Path Suggestions

If you're interested in...

Games: Start with Pygame Zero → Pygame → Arcade

Mobile Apps: Start with Kivy basics → Build a simple app → Explore advanced features

Data & Analytics: Start with Pandas → Matplotlib → Scikit-learn

AI/ML: Start with NumPy → Pandas → Scikit-learn → TensorFlow/PyTorch

Web Development: Start with Flask → Build a simple site → Django (for larger projects)

Automation: Start with Requests → Beautiful Soup → Selenium

Images/Media: Start with Pillow → OpenCV (for advanced projects)

Tips for Success

1. **Practice regularly** - Even 15 minutes a day helps!
2. **Type the code yourself** - Don't just copy and paste
3. **Experiment** - Try changing things to see what happens
4. **Make mistakes** - Errors are how you learn
5. **Build small projects** - Apply what you learn to real problems
6. **Be patient** - Programming takes time to learn

Happy Coding! 🐍