Using AI Tools and Python to Automate Tasks

A two-day, beginner-friendly journey to streamline workflows, boost productivity, and integrate AI into everyday tasks.

Goals

1. **Understand the basics of Python scripting without prior programming experience**

Goals

1. **Understand the basics of Python scripting without prior programming experience**
2. **Explore ways to leverage LLMs in combination with Python**

Goals

1. **Understand the basics of Python scripting without prior programming experience**
2. **Explore ways to leverage LLMs in combination with Python**
3. **Learn how to automate tasks such as file organization and data entry**

Goals

1. **Understand the basics of Python scripting without prior programming experience**
2. **Explore ways to leverage LLMs in combination with Python**
3. **Learn how to automate tasks such as file organization and data entry**
4. **Use Python scripts to integrate with external LLM APIs**

Survey Question: Do you have a subscription for ChatGPT/Claude/Gemini?

Programming?

Making Machines Do What You Want 🤖

Example: Instructing a Robot

Example: Instructing a Robot

- The Task: **"Move forward, pick up the box, turn right, place the box"**

Example: Instructing a Robot

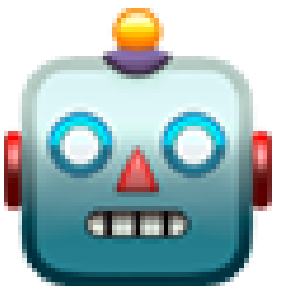
- The Task: **"Move forward, pick up the box, turn right, place the box"**
- The Code:

```
def deliver_package():
  robot.move_forward(steps=10)
  robot.pick_up("box")
  robot.turn_right(90)
  robot.place_item("box")
  robot.say("Package delivered!")
```

Example: Instructing a Robot

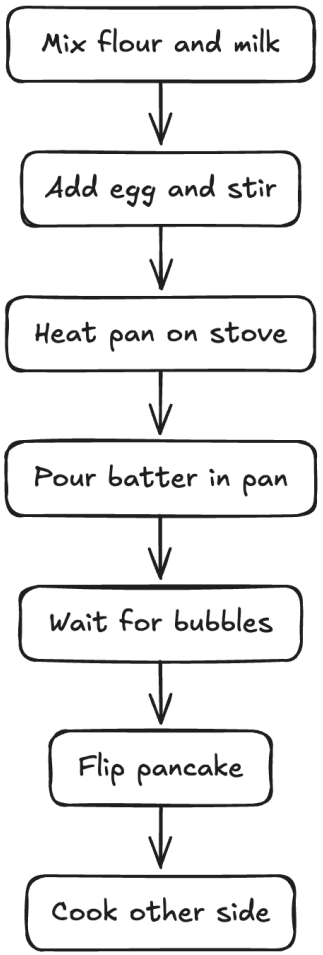
- The Task: **"Move forward, pick up the box, turn right, place the box"**
- The Code:

```
def deliver_package():
  robot.move_forward(steps=10)
  robot.pick_up("box")
  robot.turn_right(90)
  robot.place_item("box")
  robot.say("Package delivered!")
```



The robot follows your instructions exactly!

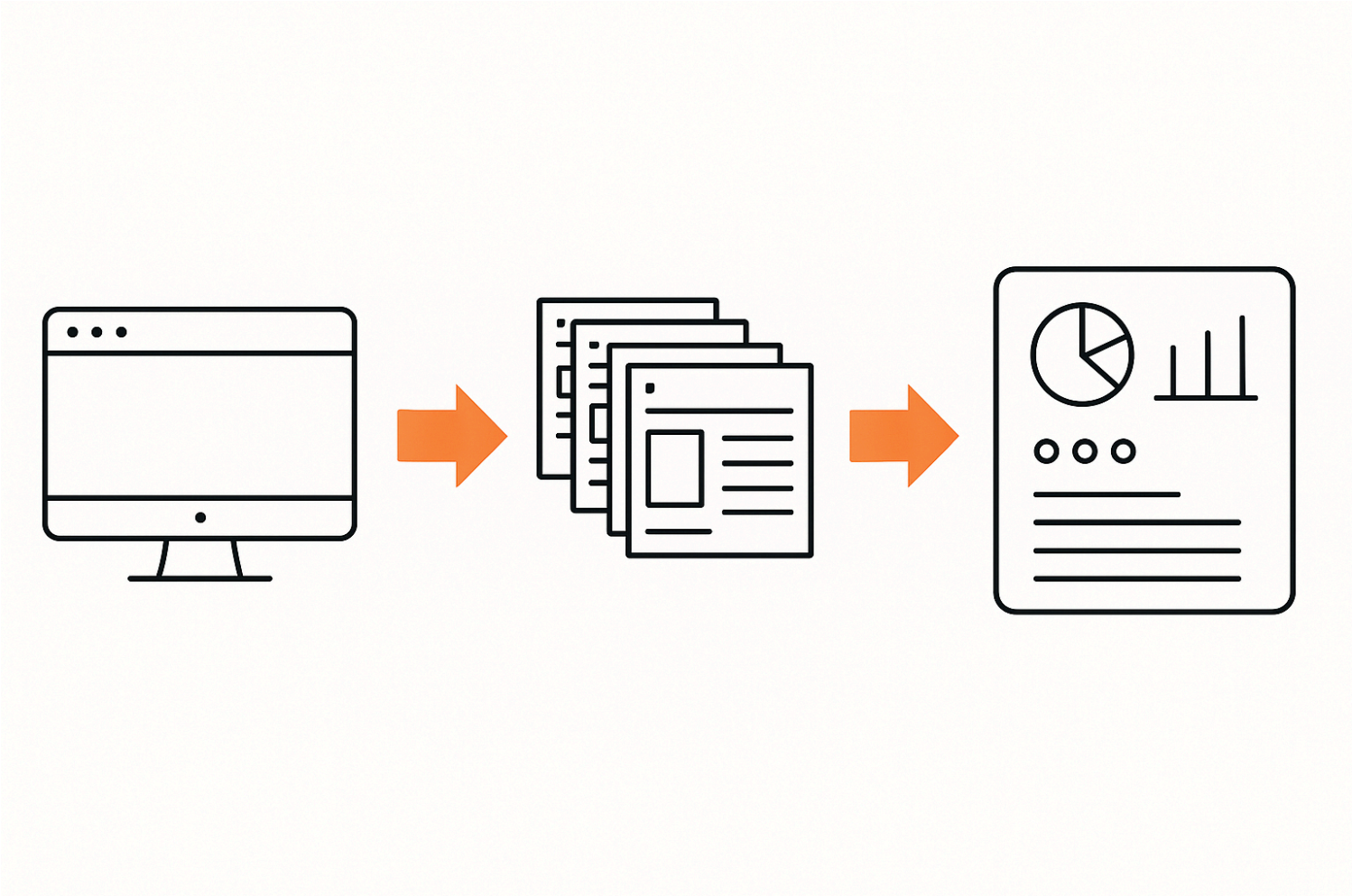
It's Like Writing Recipes




```
# Pancake Making Algorithm

    def make_pancake():
        # Step 1: Prepare batter base
        batter = mix_flour_and_milk()

        # Step 2: Complete the batter
        batter = add_egg_and_stir(batter)

        # Step 3: Prepare cooking surface
        pan = heat_pan_on_stove()

        # Step 4: Start cooking
        pour_batter_in_pan(pan, batter)

        # Step 5: Wait for indicator
        wait_for_bubbles()

        # Step 6: Turn the pancake
        flip_pancake()

        # Step 7: Finish cooking
        cook_other_side()

        return pancake
```

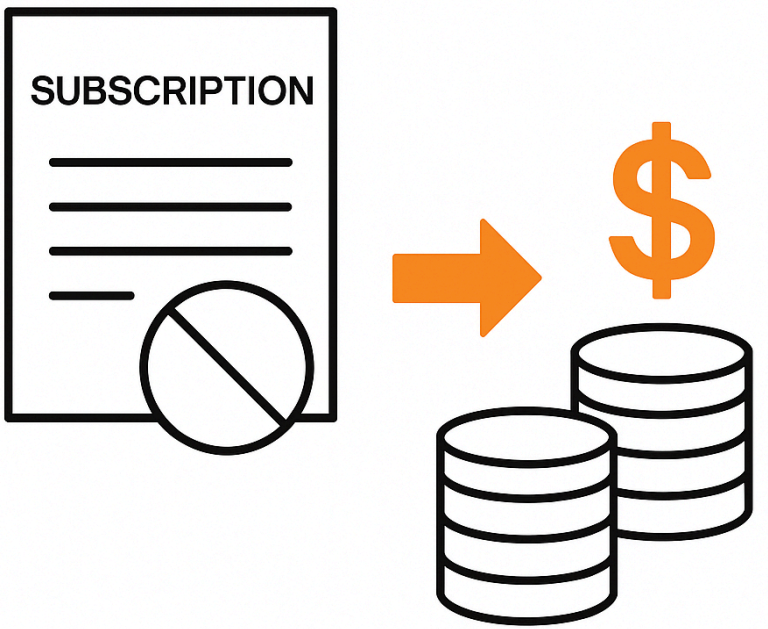
Why Learn to Automate?

Automate Repetitive Tasks



Increase Productivity

Replacing Subscriptions (saving $)



Analyse Massive Amounts of Data

Date	Sales
06/05	$12400
04/15	$16100
04/12	$5040
05/25	$23200
...	...
05/10	$1200

The Real Reason....

Writing code = the most general skill you can have!

It is a superpower! :)

Important Clarification

Learning to automate ≠ Everyone should be a developer

Automation is for Everyone

Automation is for Everyone

- Automation skills are for **everyone**

Automation is for Everyone

- Automation skills are for **everyone**
- Professional software engineering is a **specialized career**

Automation is for Everyone

- Automation skills are for **everyone**
- Professional software engineering is a **specialized career**
- You can be productive with code **without becoming a full-time developer**

Automation is for Everyone

- Automation skills are for **everyone**
- Professional software engineering is a **specialized career**
- You can be productive with code **without becoming a full-time developer**
- It's about writing **scripts** not fancy software

Automation is for Everyone

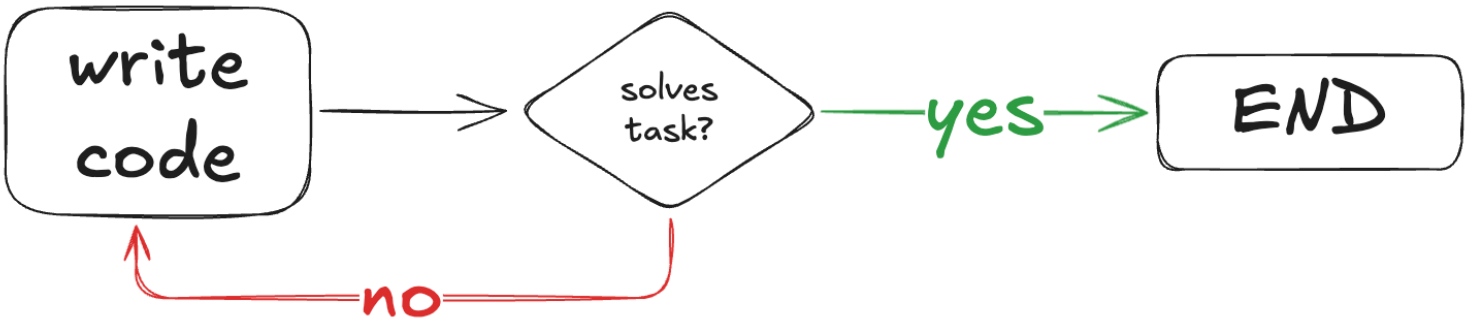
- Automation skills are for **everyone**
- Professional software engineering is a **specialized career**
- You can be productive with code **without becoming a full-time developer**
- It's about writing **scripts** not fancy software
- Script = **simple program** that automates a task

Automation is for Everyone

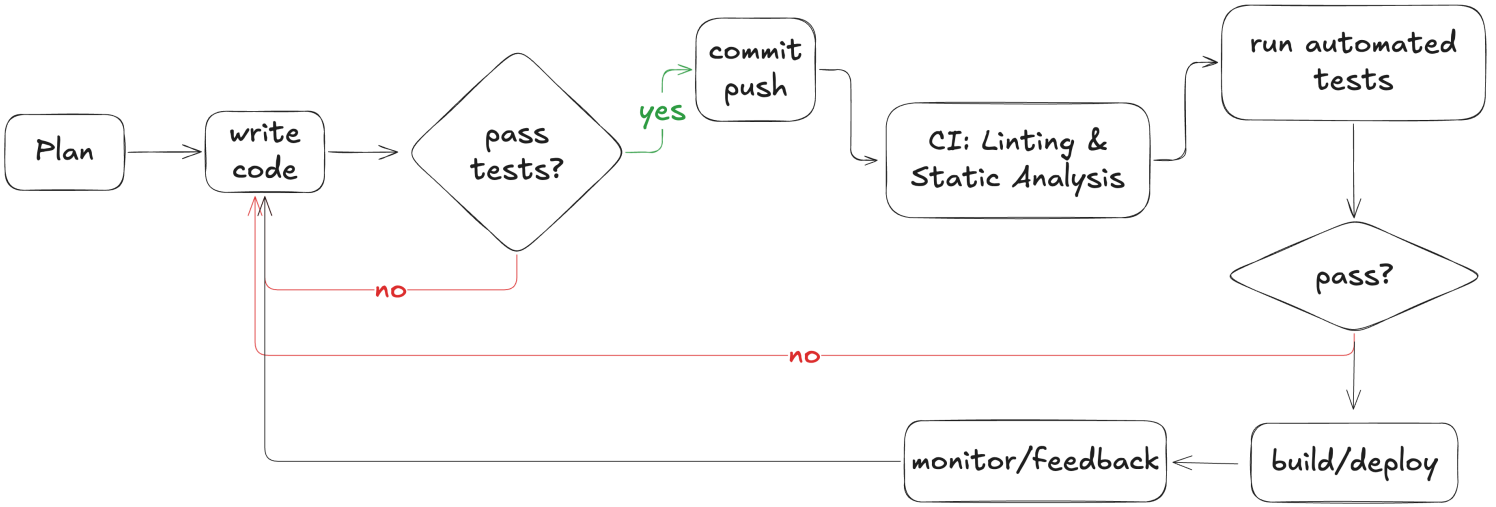
- Automation skills are for **everyone**
- Professional software engineering is a **specialized career**
- You can be productive with code **without becoming a full-time developer**
- It's about writing **scripts** not fancy software
- Script = **simple program** that automates a task
- Professional software = **complex program engineered** to be used by many people

Scripting ≠ 'Software Engineering'

Scripting



Software Engineering



Scripting

- Writing code to full fill a set of pre-defined steps that accomplish a task

Scripting

- Writing code to full fill a set of pre-defined steps that accomplish a task
- Quick, targeted, and task-oriented

Scripting

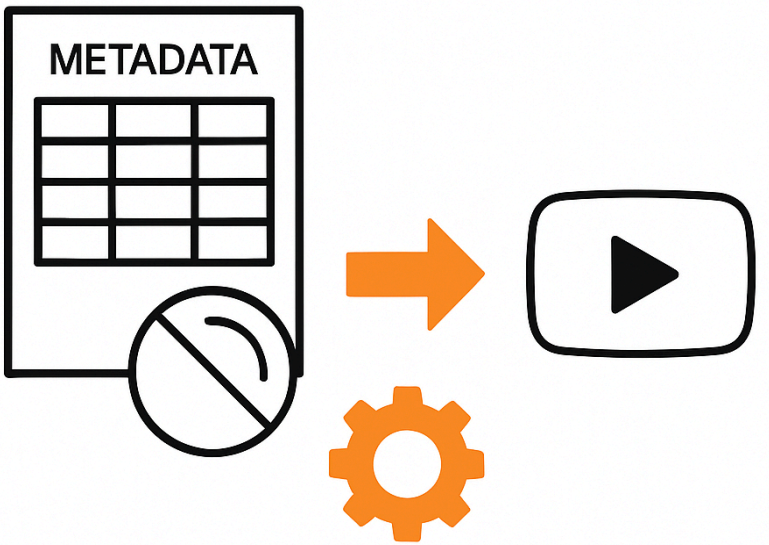
- Writing code to full fill a set of pre-defined steps that accomplish a task
- Quick, targeted, and task-oriented
- You don't think that much about it

Scripting

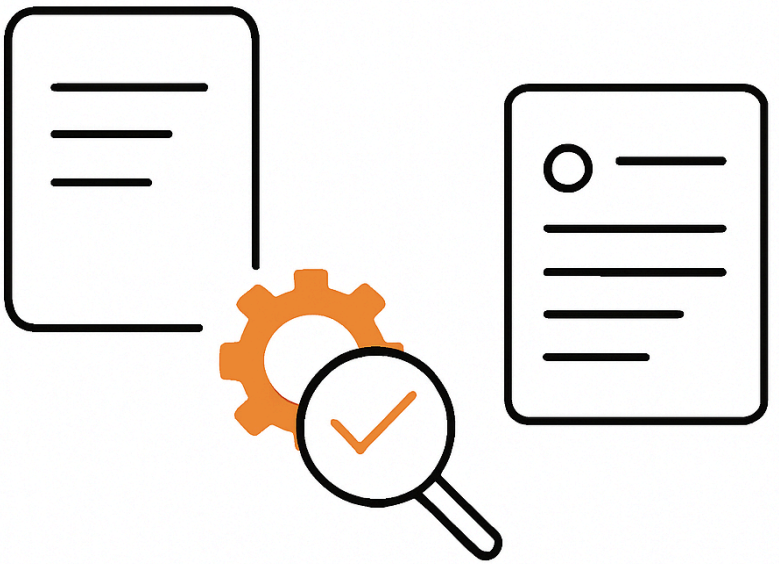
- Writing code to full fill a set of pre-defined steps that accomplish a task
- Quick, targeted, and task-oriented
- You don't think that much about it
- Typically used for automations, data manipulation, or utility tasks

Real Automation Examples (that I use!)

Bulk YouTube Upload Automation



Notes Processor



Your automation doesn't need to be perfect
—it just needs to solve YOUR problem!

Why Python?

What is Python?

Python is a high-level, general-purpose programming language designed for readability and simplicity.

What is Python?

Python is a high-level, general-purpose programming language designed for readability and simplicity.

Created by Guido van Rossum in 1991, Python emphasizes code that reads like English.

What is Python?

Python is a high-level, general-purpose programming language designed for readability and simplicity.

Created by Guido van Rossum in 1991, Python emphasizes code that reads like English.

English:

"For each item in my shopping list, print the item"

Python:

```
for item in shopping_list:
    print(item)
```

What is Python?

Python is a high-level, general-purpose programming language designed for readability and simplicity.

Created by Guido van Rossum in 1991, Python emphasizes code that reads like English.

English:

"For each item in my shopping list, print the item"

Python:

```
for item in shopping_list:
    print(item)
```

See how similar they are? That's Python's superpower!

Python is Easy and Everywhere

Python is Easy and Everywhere

- **General Purpose**: Can be used for web development, data science, automation, AI, and more

Python is Easy and Everywhere

- **General Purpose**: Can be used for web development, data science, automation, AI, and more
- **Most Popular**: #1 language among developers worldwide (TIOBE Index, Stack Overflow)

Python is Easy and Everywhere

- **General Purpose**: Can be used for web development, data science, automation, AI, and more
- **Most Popular**: #1 language among developers worldwide (TIOBE Index, Stack Overflow)
- **Powers Modern Technology**:
  - 🤖 AI & Machine Learning (TensorFlow, PyTorch)
  - 🚗 Self-driving cars (Tesla, Waymo)
  - 👁️ Computer vision & image recognition
  - 🌐 Web applications (Instagram, Spotify, Netflix)
  - 🔬 Scientific computing & research

Python is Easy and Everywhere 

- **General Purpose**: Can be used for web development, data science, automation, AI, and more
- **Most Popular**: #1 language among developers worldwide (TIOBE Index, Stack Overflow)
- **Powers Modern Technology**:
  - 🤖 AI & Machine Learning (TensorFlow, PyTorch)
  - 🚗 Self-driving cars (Tesla, Waymo)
  - 👁️ Computer vision & image recognition
  - 🌐 Web applications (Instagram, Spotify, Netflix)
  - 🔬 Scientific computing & research
- **Beginner-Friendly**: Reads like English, minimal syntax complexity

Scripting with Python

Scripting with Python

- Much faster learning curve - *if it works don't change it*

Scripting with Python

- Much faster learning curve - *if it works don't change it*
- Immediate practical benefits - *makes your life better*

Scripting with Python

- Much faster learning curve - *if it works don't change it*
- Immediate practical benefits - *makes your life better*
- Ideal for nondevelopers who just want to get things done

Scripting with Python

- Much faster learning curve - *if it works don't change it*
- Immediate practical benefits - *makes your life better*
- Ideal for nondevelopers who just want to get things done
- **It's about learning to solve YOUR specific problems**

Scripting with Python

- Much faster learning curve - *if it works don't change it*
- Immediate practical benefits - *makes your life better*
- Ideal for nondevelopers who just want to get things done
- **It's about learning to solve YOUR specific problems**
- Scripts should be **tailored to your unique needs**, not one-size-fits-all solutions

The Ostrich Approach to Learning Python

Ostrich algorithm

From Wikipedia, the free encyclopedia

In computer science, the **ostrich algorithm** is a strategy of ignoring potential problems on the basis that they may be exceedingly rare. It is named after the ostrich effect which is defined as "to stick one's head in the sand and pretend there is no problem". It is used when it appears the situation may be more cost-effectively managed by allowing the problem to continue to occur rather than to attempt its prevention.

It's about Learning What We Need for Our Tasks!

It's about Learning What We Need for Our Tasks!

And purposefuly ignoring that which does not seem to have any effect on its success.

The Busy Person Guide to Python Basics

Python Things We Will Learn

- **Basic Syntax**: Indentation, variable assignments, printing

Python Things We Will Learn

- **Basic Syntax**: Indentation, variable assignments, printing
- **Common Data Types**: Strings, integers, floats, booleans

Python Things We Will Learn

- **Basic Syntax**: Indentation, variable assignments, printing
- **Common Data Types**: Strings, integers, floats, booleans
- **Control Structures**: `if` statements, loops (`for`, `while`)

Python Things We Will Learn

- **Basic Syntax**: Indentation, variable assignments, printing
- **Common Data Types**: Strings, integers, floats, booleans
- **Control Structures**: `if` statements, loops (`for`, `while`)
- **Useful Libraries**: `os` for file operations, `csv` for handling CSVs, `requests` for making web requests

Python Things We Will Learn

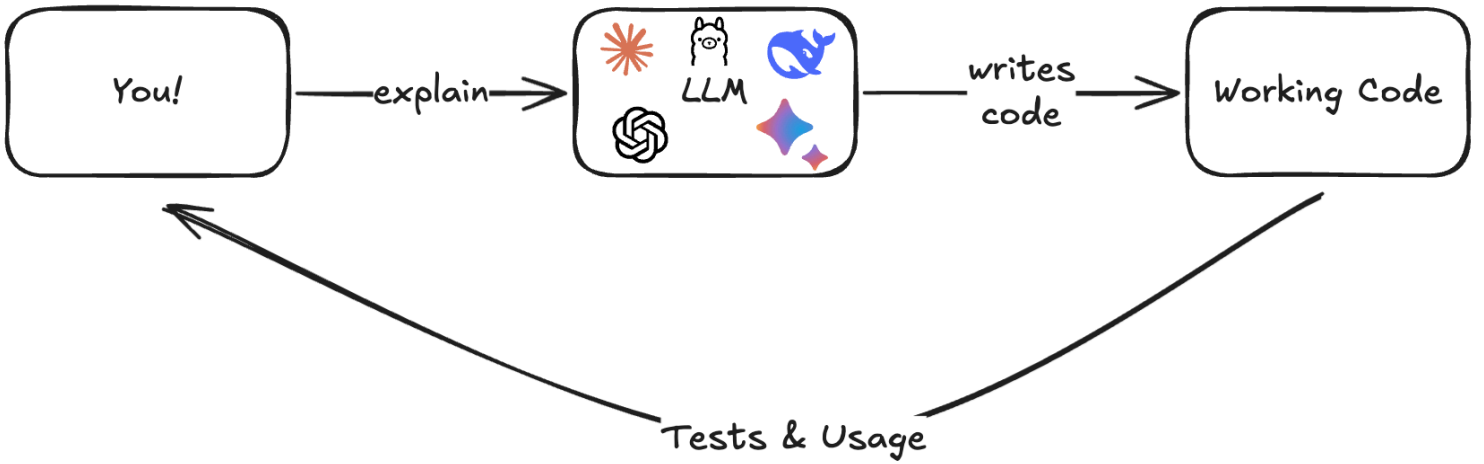
- **Basic Syntax**: Indentation, variable assignments, printing
- **Common Data Types**: Strings, integers, floats, booleans
- **Control Structures**: `if` statements, loops (`for`, `while`)
- **Useful Libraries**: `os` for file operations, `csv` for handling CSVs, `requests` for making web requests
- That is a lot! How can we manage?

Python Things We Will Learn

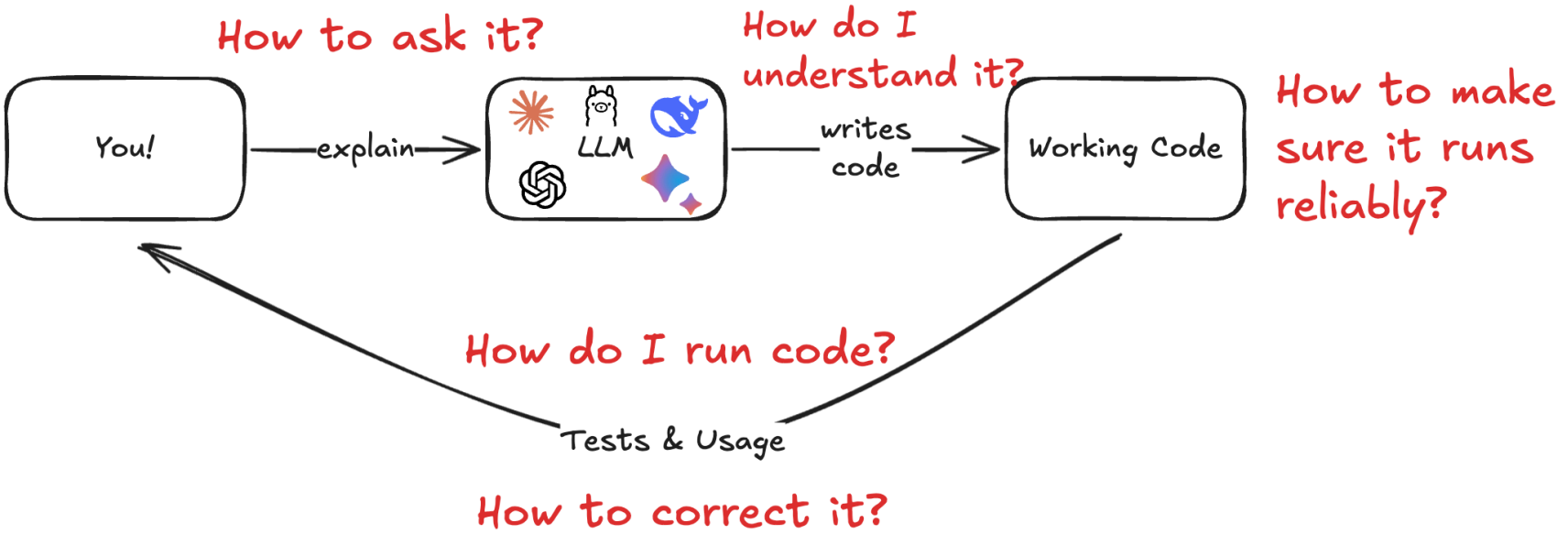
- **Basic Syntax**: Indentation, variable assignments, printing
- **Common Data Types**: Strings, integers, floats, booleans
- **Control Structures**: `if` statements, loops (`for`, `while`)
- **Useful Libraries**: `os` for file operations, `csv` for handling CSVs, `requests` for making web requests
- That is a lot! How can we manage?
- Use AI conversationally! Asking questions and clarifying what you don't know!

The "Just Ask AI" Naive Approach

The "Just Ask AI" Naive Approach



The "Just Ask AI" Naive Approach



Demo - Setting Up Your Environment

(Installing Python, uv, Editor, Jupyter Notebook, AI chatbot)

Data Types, Operations, Variables

- Writing code is about writing text that changes or manipulates data in some way

Data Types, Operations, Variables

- Writing code is about writing text that changes or manipulates data in some way
- For that we need? ..... (drumbroll) 
- **Data!** Specifically a way to describe it, reference it, talk about it etc...

Data Types, Operations, Variables

- Writing code is about writing text that changes or manipulates data in some way
- For that we need? ..... (drumbroll) 
- **Data!** Specifically a way to describe it, reference it, talk about it etc...
- In Python data can be of different **types** (like numbers, text, image etc...)

Data Types, Operations, Variables

- Writing code is about writing text that changes or manipulates data in some way
- For that we need? ..... (drumbroll) 
- **Data!** Specifically a way to describe it, reference it, talk about it etc...
- In Python data can be of different **types** (like numbers, text, image etc...)
- The things we can do to it are called **operations** (like 5+5)

Data Types, Operations, Variables

- Writing code is about writing text that changes or manipulates data in some way
- For that we need? ..... (drumbroll) 
- **Data!** Specifically a way to describe it, reference it, talk about it etc...
- In Python data can be of different **types** (like numbers, text, image etc...)
- The things we can do to it are called **operations** (like 5+5)
- To organize things, we use **variables** to define what each thing is

Data Types, Operations, Variables - Example

- Here is a piece of code that defines a variable of some type and performs a simple operation on the data stored in the variable

```
# This is data of type string!
name = "Lucas"

# This is data of type integer!
actual_age = 33

# This is data of type integer!
mental_age = 12

# our operation
average_age_between_actual_and_mental = (actual_age + mental_age) / 2

# special function that displays what goes inside of it
print(average_age_between_actual_and_mental)

# this would be a float!
# Output: 22.5
```

Data Types; Operations; Variables - Demo

Data Types, Operations, Variables

- **Core Data Types**:

Data Types, Operations, Variables

- **Core Data Types**:
  - **int**: whole nubmers like 1, 2, 3 ...

Data Types, Operations, Variables

- **Core Data Types**:
  - **int**: whole nubmers like 1, 2, 3 ...
  - **float**: decimal numbers like 1.1, 1.0, etc...

Data Types, Operations, Variables

- **Core Data Types**:
  - **int**: whole nubmers like 1, 2, 3 ...
  - **float**: decimal numbers like 1.1, 1.0, etc...
  - **string**: text like `'Hello'`

Data Types, Operations, Variables

- **Core Data Types**:
  - **int**: whole nubmers like 1, 2, 3 ...
  - **float**: decimal numbers like 1.1, 1.0, etc...
  - **string**: text like `'Hello'`
  - **bool**: logical booleans like True or False

Data Types, Operations, Variables

- **Core Data Types**:
  - **int**: whole nubmers like 1, 2, 3 ...
  - **float**: decimal numbers like 1.1, 1.0, etc...
  - **string**: text like `'Hello'`
  - **bool**: logical booleans like True or False
- **Operations**:

Data Types, Operations, Variables

- **Core Data Types**:
  - **int**: whole nubmers like 1, 2, 3 ...
  - **float**: decimal numbers like 1.1, 1.0, etc...
  - **string**: text like `'Hello'`
  - **bool**: logical booleans like True or False
- **Operations**:
  - **Arithmetic** `(+,-,,/,*,//,%)`

Data Types, Operations, Variables

- **Core Data Types**:
  - **int**: whole nubmers like 1, 2, 3 ...
  - **float**: decimal numbers like 1.1, 1.0, etc...
  - **string**: text like `'Hello'`
  - **bool**: logical booleans like True or False
- **Operations**:
  - **Arithmetic** (`+`,`-`,`/`,`*`,`//`,`%`)
  - **String concatenation** (`'Lucas' + ' is' + ' wonderful' = 'Lucas is wonderful'`)

Data Types, Operations, Variables

- **Core Data Types**:
  - **int**: whole nubmers like 1, 2, 3 ...
  - **float**: decimal numbers like 1.1, 1.0, etc...
  - **string**: text like `'Hello'`
  - **bool**: logical booleans like True or False
- **Operations**:
  - **Arithmetic** (`+`,`-`,`/`,`*`,`//`,`%`)
  - **String concatenation** (`'Lucas' + ' is' + ' wonderful' = 'Lucas is wonderful'`)
  - **Logical operations** (`and`, `or`, `not`)

Data Types, Operations, Variables

- **Core Data Types**:
  - **int**: whole nubmers like 1, 2, 3 ...
  - **float**: decimal numbers like 1.1, 1.0, etc...
  - **string**: text like `'Hello'`
  - **bool**: logical booleans like True or False
- **Operations**:
  - **Arithmetic** (`+`,`-`,`/`,`*`,`//`,`%`)
  - **String concatenation** (`'Lucas' + ' is' + ' wonderful' = 'Lucas is wonderful'`)
  - **Logical operations** (`and`, `or`, `not`)
  - **Comparison operations** (`>`, `<`, `>=`, `<=`, `==`, `!=`)

- **Variables**: Storing data for reuse, assigning and reassigning values:

```
a = 10
b = 20
print(a + b)
# Output: 30
```

Demo: Functions, Lists & Loops

Functions

Functions

- Functions are a way to group code that performs a specific task

Functions

- Functions are a way to group code that performs a specific task
- Functions can take parameters and return values

Functions

- Functions are a way to group code that performs a specific task
- Functions can take parameters and return values
- For example, a function that calculates the total cost of an item including tax

Functions

- Functions are a way to group code that performs a specific task
- Functions can take parameters and return values
- For example, a function that calculates the total cost of an item including tax
- We could do this by simply writing a script:

Functions

- Functions are a way to group code that performs a specific task
- Functions can take parameters and return values
- For example, a function that calculates the total cost of an item including tax
- We could do this by simply writing a script:

```
# Arithmetic operation: multiplication of parameters
tax = price * tax_rate
# Arithmetic operation: addition of variables
total = price + tax
```

- But what if I want to re-use this code for different prices and tax rates?

Functions

- Here is what it would look like if we define a function to do this:

```
def calculate_total(price, tax_rate):
    tax = price * tax_rate
    total = price + tax
    return total
```

Functions

- Here is what it would look like if we define a function to do this:

```
def calculate_total(price, tax_rate):
    tax = price * tax_rate
    total = price + tax
    return total
```

- Now we can re-use the function for different prices and tax rates:

```
shirt_price = 10
shirt_tax_rate = 0.05
pants_price = 20
pants_tax_rate = 0.1

print(calculate_total(shirt_price, shirt_tax_rate))
# Output: 10.5
print(calculate_total(pants_price, pants_tax_rate))
# Output: 22.0
```

Functions

- **Why Functions?**: Reuse code and break tasks into smaller chunks

Functions

- **Why Functions?**: Reuse code and break tasks into smaller chunks
- **Defining a Function**: `def function_name(parameters):`

Functions

- **Why Functions?**: Reuse code and break tasks into smaller chunks
- **Defining a Function**: `def function_name(parameters):`
- **Return Values**: Make your functions flexible and reusable

Functions

- **Why Functions?**: Reuse code and break tasks into smaller chunks
- **Defining a Function**: `def function_name(parameters):`
- **Return Values**: Make your functions flexible and reusable
- **Best Practices**: Keep them short, descriptive, and single-purpose

Lists & Loops

- **Lists**: Python's go-to data structure for ordered collections

Lists & Loops

- **Lists**: Python's go-to data structure for ordered collections

```
# Define a list of tasks
tasks = ["Buy groceries", "Finish project", "Call the bank"]
```

- **Access Elements**: Indexing and slicing

Lists & Loops

- **Lists**: Python's go-to data structure for ordered collections

```
# Define a list of tasks
tasks = ["Buy groceries", "Finish project", "Call the bank"]
```

- **Access Elements**: Indexing and slicing

```
print(tasks[0])
# Output: Buy groceries

print(tasks[1])
# Output: Finish project
```

Lists & Loops

- **Lists**: Python's go-to data structure for ordered collections

```
# Define a list of tasks
tasks = ["Buy groceries", "Finish project", "Call the bank"]
```

- **Access Elements**: Indexing and slicing

```
print(tasks[0])
# Output: Buy groceries

print(tasks[1])
# Output: Finish project
```

- **Slice**: Get a range of elements

Lists & Loops

- **Lists**: Python's go-to data structure for ordered collections

```
# Define a list of tasks
tasks = ["Buy groceries", "Finish project", "Call the bank"]
```

- **Access Elements**: Indexing and slicing

```
print(tasks[0])
# Output: Buy groceries

print(tasks[1])
# Output: Finish project
```

- **Slice**: Get a range of elements

```
print(tasks[0:2])
# Output: ['Buy groceries', 'Finish project']
```

Lists & Loops

- **Loops**: `for` loops to iterate over items

Lists & Loops

- **Loops**: `for` loops to iterate over items

```
tasks = ["Buy groceries", "Finish project", "Call the bank"]

for task in tasks:
    print(task)
```

Lists & Loops

- **Loops**: `for` loops to iterate over items

```
tasks = ["Buy groceries", "Finish project", "Call the bank"]

for task in tasks:
    print(task)
```

- **Processing List Elements**: With `for` loops we can perform the same operation on each element of the list

Lists & Loops

- **Loops**: `for` loops to iterate over items

```
tasks = ["Buy groceries", "Finish project", "Call the bank"]

for task in tasks:
    print(task)
```

- **Processing List Elements**: With `for` loops we can perform the same operation on each element of the list

```
# Below we use an imaginary function that asks a robot to do a task
for task in tasks:
    ask_robot_to_do(task)
```

Lists & Loops

- **Loops**: `for` loops to iterate over items

```
tasks = ["Buy groceries", "Finish project", "Call the bank"]

for task in tasks:
    print(task)
```

- **Processing List Elements**: With `for` loops we can perform the same operation on each element of the list

```
# Below we use an imaginary function that asks a robot to do a task
for task in tasks:
    ask_robot_to_do(task)
```

Common Uses: Batch renaming files in a directory, processing data from tables, etc...

Demo - Dictionaries, Tabular Data, Conditionals

Dictionaries & Tabular Data

- **Dictionaries**: Key-value pairs for storing related data

Dictionaries & Tabular Data

- **Dictionaries**: Key-value pairs for storing related data

```
# Dictionary of product prices
prices = {
    "apple": 0.50,
    "banana": 0.75,
    "orange": 0.60
}
```

Dictionaries & Tabular Data

- **Dictionaries**: Key-value pairs for storing related data

```
# Dictionary of product prices
prices = {
    "apple": 0.50,
    "banana": 0.75,
    "orange": 0.60
}
```

- **Accessing Values**: Use keys to lookup values quickly

Dictionaries & Tabular Data

- **Dictionaries**: Key-value pairs for storing related data

```
# Dictionary of product prices
prices = {
    "apple": 0.50,
    "banana": 0.75,
    "orange": 0.60
}
```

- **Accessing Values**: Use keys to lookup values quickly

```
print(prices["apple"])
# Output: 0.50

print(prices["banana"])
# Output: 0.75
```

Dictionaries & Tabular Data

- **Dictionaries**: Key-value pairs for storing related data

```
# Dictionary of product prices
prices = {
    "apple": 0.50,
    "banana": 0.75,
    "orange": 0.60
}
```

- **Accessing Values**: Use keys to lookup values quickly

```
print(prices["apple"])
# Output: 0.50

print(prices["banana"])
# Output: 0.75
```

- **Common Uses**: Storing configurations, mapping relationships, caching data

Comparators & Conditionals

- **Boolean Comparisons**: ==, !=, >, <, >=, <=

Comparators & Conditionals

- **Boolean Comparisons**: ==, !=, >, <, >=, <=

```
# Example of a boolean comparison
a = 10
b = 20
print(a == b)  # Output: False
print(a != b)  # Output: True
print(a > b)   # Output: False
print(a < b)   # Output: True
print(a >= b) # Output: False
print(a <= b) # Output: True
```

Comparators & Conditionals

- **Boolean Comparisons**: `==`, `!=`, `>`, `<`, `>=`, `<=`

```
# Example of a boolean comparison
a = 10
b = 20
print(a == b)  # Output: False
print(a != b)  # Output: True
print(a > b)   # Output: False
print(a < b)   # Output: True
print(a >= b) # Output: False
print(a <= b) # Output: True
```

- **Logical Operators**: `and`, `or`, `not`

Comparators & Conditionals

- **Boolean Comparisons**: `==`, `!=`, `>`, `<`, `>=`, `<=`

```
# Example of a boolean comparison
a = 10
b = 20
print(a == b)  # Output: False
print(a != b)  # Output: True
print(a > b)   # Output: False
print(a < b)   # Output: True
print(a >= b) # Output: False
print(a <= b) # Output: True
```

- **Logical Operators**: `and`, `or`, `not`

```
# Example of a logical operator
print(True and False)  # Output: False
print(True or False)   # Output: True
print(not True)         # Output: False
```

Comparators & Conditionals

- **Conditionals**: Using conditionals `if/elif/else` to perform different actions based on data

Comparators & Conditionals

- **Conditionals**: Using conditionals `if/elif/else` to perform different actions based on data

```
# Example of an if statement
if a > b:
    print("a is greater than b")
```

Comparators & Conditionals

- **Conditionals**: Using conditionals `if/elif/else` to perform different actions based on data

```
# Example of an if statement
if a > b:
    print("a is greater than b")
```

```
# Example of an if/else statement
if a > b:
    print("a is greater than b")
else:
    print("a is less than or equal to b")
```

Comparators & Conditionals

- **Conditionals**: Using conditionals `if/elif/else` to perform different actions based on data

```
# Example of an if statement
if a > b:
    print("a is greater than b")
```

```
# Example of an if/else statement
if a > b:
    print("a is greater than b")
else:
    print("a is less than or equal to b")
```

```
# Example of an if/elif/else statement
if a > b:
    print("a is greater than b")
elif a == b:
    print("a is equal to b")
else:
    print("a is less than b")
```

Quick Knowledge Check! 🧠

Quiz Question 1

What will this code output?

```
age = 25
if age >= 18:
    print("Adult")
else:
    print("Minor")
```

Quiz Question 1

What will this code output?

```
age = 25
if age >= 18:
    print("Adult")
else:
    print("Minor")
```

Answer: `Adult`

Quiz Question 1

What will this code output?

```
age = 25
if age >= 18:
    print("Adult")
else:
    print("Minor")
```

Answer: `Adult`

Explanation: Since 25 is greater than or equal to 18, the condition is True, so "Adult" is printed.

Quiz Question 2

What's wrong with this code?

```
tasks = ["Email", "Meeting", "Code"]
print(tasks[3])
```

Quiz Question 2

What's wrong with this code?

```
tasks = ["Email", "Meeting", "Code"]
print(tasks[3])
```

Answer: Index out of range error!

Quiz Question 2

What's wrong with this code?

```
tasks = ["Email", "Meeting", "Code"]
print(tasks[3])
```

Answer: Index out of range error!

Explanation: Lists are zero-indexed. This list has indices 0, 1, 2. There is no index 3!

- `tasks[0]` → "Email"
- `tasks[1]` → "Meeting"
- `tasks[2]` → "Code"

Quiz Question 3

Fill in the blank to print each fruit:

```
fruits = ["apple", "banana", "cherry"]
___ fruit ___ fruits:
    print(fruit)
```

Quiz Question 3

Fill in the blank to print each fruit:

```
fruits = ["apple", "banana", "cherry"]
___ fruit ___ fruits:
    print(fruit)
```

Answer:

```
for fruit in fruits:
    print(fruit)
```

Quiz Question 3

Fill in the blank to print each fruit:

```
fruits = ["apple", "banana", "cherry"]
___ fruit ___ fruits:
    print(fruit)
```

Answer:

```
for fruit in fruits:
    print(fruit)
```

Explanation: The `for` loop iterates through each item in the list, and `in` checks membership.

Demo - Working with Data, APIs & Packages

Reading/Writing

- **Tabular Data**: Data that is organized in a table format

Reading/Writing

- **Tabular Data**: Data that is organized in a table format
- **Common Formats**: `.csv`, `.json`, `.xlsx`, `.xls`

Reading/Writing

- **Tabular Data**: Data that is organized in a table format
- **Common Formats**: `.csv`, `.json`, `.xlsx`, `.xls`
- CSV Files store data in rows and columns via comma-separated values:

```
Name,Age,City
John Smith,32,New York
Jane Doe,28,San Francisco
```

Name	Age	City
John Smith	32	New York
Jane Doe	28	San Francisco

Understanding APIs & Packages

What is an API?

API = Application Programming Interface

What is an API?

API = Application Programming Interface

A way for different programs to communicate with each other.

What is an API?

API = Application Programming Interface

A way for different programs to communicate with each other.

Think of it like a restaurant:

- You (your code) look at the menu (API documentation)
- You tell the waiter (API) what you want
- The kitchen (external service) prepares it
- The waiter brings back your order (response)

What is an API?

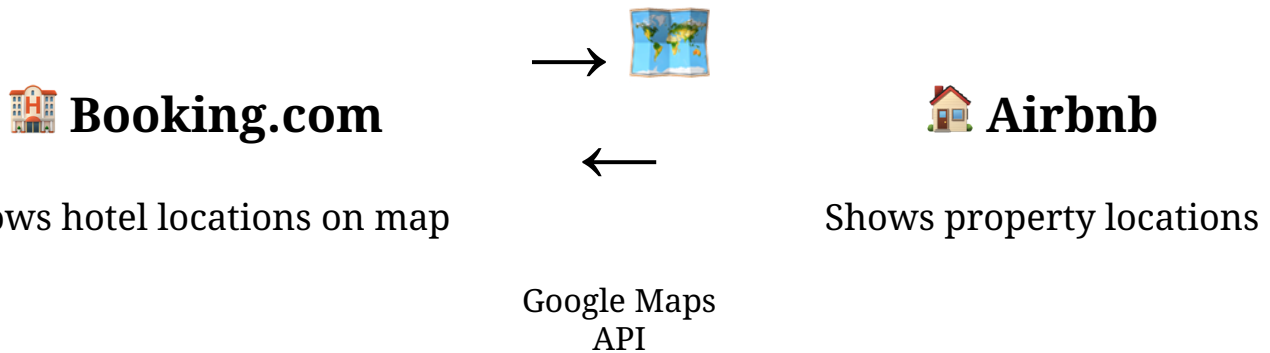
API = Application Programming Interface

A way for different programs to communicate with each other.

Think of it like a restaurant:

- You (your code) look at the menu (API documentation)
- You tell the waiter (API) what you want
- The kitchen (external service) prepares it
- The waiter brings back your order (response)

Real-World Example: Google Maps API



What is an API?

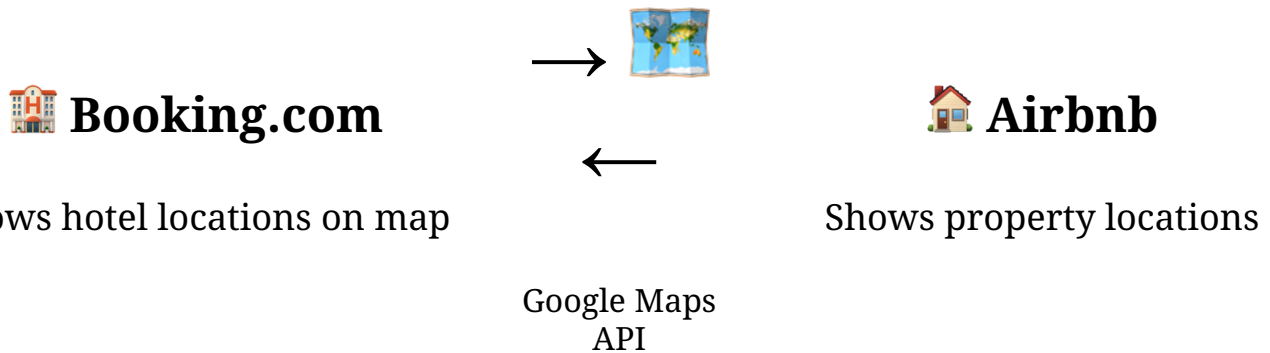
API = Application Programming Interface

A way for different programs to communicate with each other.

Think of it like a restaurant:

- You (your code) look at the menu (API documentation)
- You tell the waiter (API) what you want
- The kitchen (external service) prepares it
- The waiter brings back your order (response)

Real-World Example: Google Maps API



What are Packages and Imports?

Package = A collection of pre-written Python code (functions, classes) that you can reuse

What are Packages and Imports?

Package = A collection of pre-written Python code (functions, classes) that you can reuse

Import = The command to bring a package into your script so you can use it

What are Packages and Imports?

Package = A collection of pre-written Python code (functions, classes) that you can reuse

Import = The command to bring a package into your script so you can use it

Example:

```
# Import the 'requests' package to make HTTP calls
import requests

# Now we can use functions from the requests package
response = requests.get("https://api.example.com/data")
print(response.json())
```

What are Packages and Imports?

Package = A collection of pre-written Python code (functions, classes) that you can reuse

Import = The command to bring a package into your script so you can use it

Example:

```
# Import the 'requests' package to make HTTP calls
import requests

# Now we can use functions from the requests package
response = requests.get("https://api.example.com/data")
print(response.json())
```

Think of it like:

- Package = A toolbox full of tools
- Import = Taking the toolbox out so you can use the tools
- Using the package = Actually using the tools to build something

What are Packages and Imports?

Package = A collection of pre-written Python code (functions, classes) that you can reuse

Import = The command to bring a package into your script so you can use it

Example:

```
# Import the 'requests' package to make HTTP calls
import requests

# Now we can use functions from the requests package
response = requests.get("https://api.example.com/data")
print(response.json())
```

Think of it like:

- Package = A toolbox full of tools
- Import = Taking the toolbox out so you can use the tools
- Using the package = Actually using the tools to build something

Popular packages: `requests` (HTTP calls), `pandas` (data analysis), `openai` (AI services)

The Package Management Problem 

When you write a Python automation script, it needs:

The Package Management Problem 

When you write a Python automation script, it needs:

1. **A specific Python version** (e.g., Python 3.11)

The Package Management Problem 

When you write a Python automation script, it needs:

1. **A specific Python version** (e.g., Python 3.11)
2. **Specific package versions** (e.g., `requests==2.31.0`, `pandas==2.0.1`)

The Package Management Problem 

When you write a Python automation script, it needs:

1. **A specific Python version** (e.g., Python 3.11)
2. **Specific package versions** (e.g., `requests==2.31.0`, `pandas==2.0.1`)
3. **Isolated environment** (so different projects don't conflict)

The Package Management Problem 

When you write a Python automation script, it needs:

1. **A specific Python version** (e.g., Python 3.11)
2. **Specific package versions** (e.g., `requests==2.31.0`, `pandas==2.0.1`)
3. **Isolated environment** (so different projects don't conflict)

Traditional Approach: Multiple Files + Virtual Environments

automation.py

```
import requests
      import pandas


# Your code here...
```

requirements.txt

```
requests==2.31.0
    pandas==2.0.1
    numpy==1.24.0
```

.python-version

```
3.11
```

The Package Management Problem 

When you write a Python automation script, it needs:

1. **A specific Python version** (e.g., Python 3.11)
2. **Specific package versions** (e.g., `requests==2.31.0`, `pandas==2.0.1`)
3. **Isolated environment** (so different projects don't conflict)

Traditional Approach: Multiple Files + Virtual Environments

automation.py

```
import requests
      import pandas


# Your code here...
```

requirements.txt

```
requests==2.31.0
    pandas==2.0.1
    numpy==1.24.0
```

.python-version

```
3.11
```

Tools: pipenv, conda, venv, poetry... **Confusing for beginners!**

Demo - The Package Management Problem

Enter UV: The Simple Solution ✨

UV = Python script runner + package manager in one

Enter UV: The Simple Solution ✨

UV = Python script runner + package manager in one

Everything in a single file!

```
# /// script
# requires-python = ">=3.11"
# dependencies = [
#     "requests==2.31.0",
#     "pandas==2.0.1",
# ]
# ///

import requests
import pandas as pd

# Your automation code here...
response = requests.get("https://api.example.com/data")
df = pd.DataFrame(response.json())
print(df.head())
```

UV: Single File vs Traditional Approach

Traditional 😓

- Create virtual environment
- Activate environment
- Install packages
- Manage requirements.txt
- Remember which env to use

📁 Multiple files

📚 Complex workflow

VS

UV 🚀

- Write your script
- Add dependencies at the top
- Run: `uv run script.py`

📄 One file

⚡ Simple workflow

UV: Single File vs Traditional Approach

Traditional 😓

- Create virtual environment
- Activate environment
- Install packages
- Manage requirements.txt
- Remember which env to use

📁 Multiple files

📚 Complex workflow

VS

UV 🚀

- Write your script
- Add dependencies at the top
- Run: `uv run script.py`

📄 One file

⚡ Simple workflow

Perfect for vibe scripting! → Learn more:
https://docs.astral.sh/uv/guides/scripts/

Demo - Vibe Scripting with UV + AI

The Vibe Scripting Workflow 🔄

🤖 **AI** → 💻 **CODE** → 📄 **file.py** → ⚡ `uv run file.py` → 👁️ **Review**

Does it solve your problem?

❌ No	✅ Yes
→ Back to AI 🤖 (refine & iterate)	→ Done! 🎉 (problem solved)

How to Use AI to Learn/Use Python

A quick detour to set up an AI toolkit to speed up our Automation skills

Learning Python with AI

Your Personal Programming Tutor

The Conversational Learning Approach

- **Start with "I don't know Python, but I want to..."** and describe your goal

The Conversational Learning Approach

- **Start with "I don't know Python, but I want to..."** and describe your goal
- **Ask for explanations**: "What does this line do?" or "Why do we use a for loop here?"

The Conversational Learning Approach

- **Start with "I don't know Python, but I want to..."** and describe your goal
- **Ask for explanations**: "What does this line do?" or "Why do we use a for loop here?"
- **Request variations**: "Show me 3 different ways to do this" or "Make this simpler"

The Conversational Learning Approach

- **Start with "I don't know Python, but I want to..."** and describe your goal
- **Ask for explanations**: "What does this line do?" or "Why do we use a for loop here?"
- **Request variations**: "Show me 3 different ways to do this" or "Make this simpler"
- **Build incrementally**: Start with tiny scripts, then ask "How do I add X to this?"

Demo - Conversational Learning with AI for Python

ARUFS Framework

ARUFS Framework

Ask, **R**un, **U**nderstand, **F**ix, Be **S**afe

ARUFS Framework

Ask, **R**un, **U**nderstand, **F**ix, Be **S**afe

- **A**sk → how to ask effectively, give the right context

ARUFS Framework

Ask, **R**un, **U**nderstand, **F**ix, Be **S**afe

- **A**sk → how to ask effectively, give the right context
- **R**un → how to run the code AI gives you

ARUFS Framework

Ask, Run, Understand, Fix, Be Safe

- **A**sk → how to ask effectively, give the right context
- **R**un → how to run the code AI gives you
- **U**nderstand → how to understand it/learn from it

ARUFS Framework

Ask, Run, Understand, Fix, Be Safe

- **A**sk → how to ask effectively, give the right context
- **R**un → how to run the code AI gives you
- **U**nderstand → how to understand it/learn from it
- **F**ix → how to fix it

ARUFS Framework

Ask, **R**un, **U**nderstand, **F**ix, Be **S**afe

- **A**sk → how to ask effectively, give the right context
- **R**un → how to run the code AI gives you
- **U**nderstand → how to understand it/learn from it
- **F**ix → how to fix it
- Be **S**afe → pittfalls & security risks

Ask

- Be **detailed**

Ask

- Be **detailed**
- Use **context** (docs, articles, pdfs, ...)

Ask

- Be **detailed**
- Use **context** (docs, articles, pdfs, ...)
- **Tools** to help with feeding data/repo into LLMs
  - https://github.com/yamadashy/repomix
  - gitingest
  - r.jina.ai
  - arxiv-txt.org
  - llms-txt
  - files-to-prompt
- Use **meta-prompts**:

```
I have this problem:
{describe the problem}
Help me write a good prompt that encapsulates this into a single Python script.
```

Ask

- For tough problems use **reasoning models**
    - Claude + Extended Thinking
    - O-series models from OpenAI
    - Gemini 2.5 Pro
    - DeepSeek R1

Run/Understand/Fix

- The pattern is:

Run/Understand/Fix

- The pattern is:
    - LLM generates the script
    - You run the script
    - You inspect the output
    - You ask the LLM to fix the script (if needed)

Run/Understand/Fix

- The pattern is:
    - LLM generates the script
    - You run the script
    - You inspect the output
    - You ask the LLM to fix the script (if needed)
- Ask for **comments on the code**

Run/Understand/Fix

- The pattern is:
  - LLM generates the script
  - You run the script
  - You inspect the output
  - You ask the LLM to fix the script (if needed)
- Ask for **comments on the code**
- **Feed the output of the terminal** to the AI and ask to fix it

Be Safe

- **Never hardcode credentials** - use environment variables or config files

Be Safe

- **Never hardcode credentials** - use environment variables or config files
- **Review AI-generated code** before running - understand what each line does

Be Safe

- **Never hardcode credentials** - use environment variables or config files
- **Review AI-generated code** before running - understand what each line does
- **Test in isolated environments** first (separate folders, test data)

Be Safe

- **Never hardcode credentials** - use environment variables or config files
- **Review AI-generated code** before running - understand what each line does
- **Test in isolated environments** first (separate folders, test data)
- **Don't share sensitive data** with AI models (personal info, API keys, proprietary data)

Be Safe

- **Never hardcode credentials** - use environment variables or config files
- **Review AI-generated code** before running - understand what each line does
- **Test in isolated environments** first (separate folders, test data)
- **Don't share sensitive data** with AI models (personal info, API keys, proprietary data)
- **Backup important files** before running automation scripts

Be Safe

- **Never hardcode credentials** - use environment variables or config files
- **Review AI-generated code** before running - understand what each line does
- **Test in isolated environments** first (separate folders, test data)
- **Don't share sensitive data** with AI models (personal info, API keys, proprietary data)
- **Backup important files** before running automation scripts
- **Check file paths** - ensure scripts won't delete important files

Be Safe

- **Never hardcode credentials** - use environment variables or config files
- **Review AI-generated code** before running - understand what each line does
- **Test in isolated environments** first (separate folders, test data)
- **Don't share sensitive data** with AI models (personal info, API keys, proprietary data)
- **Backup important files** before running automation scripts
- **Check file paths** - ensure scripts won't delete important files
- **Add safeguards** for loops (break conditions, iteration limits)

Be Safe

- **Never hardcode credentials** - use environment variables or config files
- **Review AI-generated code** before running - understand what each line does
- **Test in isolated environments** first (separate folders, test data)
- **Don't share sensitive data** with AI models (personal info, API keys, proprietary data)
- **Backup important files** before running automation scripts
- **Check file paths** - ensure scripts won't delete important files
- **Add safeguards** for loops (break conditions, iteration limits)
- **Respect rate limits** and website terms of service when scraping

Be Safe

- **Never hardcode credentials** - use environment variables or config files
- **Review AI-generated code** before running - understand what each line does
- **Test in isolated environments** first (separate folders, test data)
- **Don't share sensitive data** with AI models (personal info, API keys, proprietary data)
- **Backup important files** before running automation scripts
- **Check file paths** - ensure scripts won't delete important files
- **Add safeguards** for loops (break conditions, iteration limits)
- **Respect rate limits** and website terms of service when scraping
- **Quick Safety Checklist**: Read code → Test with dummy data → Have backups → Use version control

Demo - Implementing the A.R.U.F.S Framework

Demo - Automating Data Extraction

Automating Data Extraction

- **Target Websites or Documents**: Identify patterns or structures (tables, IDs, HTML tags)

Automating Data Extraction

- **Target Websites or Documents**: Identify patterns or structures (tables, IDs, HTML tags)
- **Techniques**: Using Beautiful Soup, Pandas, or request-response cycles

Automating Data Extraction

- **Target Websites or Documents**: Identify patterns or structures (tables, IDs, HTML tags)
- **Techniques**: Using Beautiful Soup, Pandas, or request-response cycles
- **Practical Examples**: Extracting data from a CSV, scraping a simple webpage for product listings

Automating Basic Data Analysis

- **Data Loading**: Reading CSV files, Excel sheets, or database tables with Pandas

Automating Basic Data Analysis

- **Data Loading**: Reading CSV files, Excel sheets, or database tables with Pandas
- **Basic Analysis**: Calculating averages, sums, or finding patterns in data

Automating Basic Data Analysis

- **Data Loading**: Reading CSV files, Excel sheets, or database tables with Pandas
- **Basic Analysis**: Calculating averages, sums, or finding patterns in data
- **Visualization**: Creating simple charts or graphs to represent findings

Automating Slides

- Effective data wrangling for high quality slides

Automating Slides

- Effective data wrangling for high quality slides
- A Hybrid Approach: AI + Python Scripts

Automating Slides

- Effective data wrangling for high quality slides
- A Hybrid Approach: AI + Python Scripts
- Bulk Processing to save time

Automating the Browser

- **Tools**: `selenium`, `playwright`

Automating the Browser

- **Tools**: `selenium`, `playwright`
- **Common Tasks**: Logging in to websites, navigating pages, clicking buttons, scraping dynamic elements

Automating the Browser

- **Tools**: `selenium`, `playwright`
- **Common Tasks**: Logging in to websites, navigating pages, clicking buttons, scraping dynamic elements
- **Why Automate Browser Tasks?**: Speed up online research, data entry, or repetitive website interactions

Automating Filling Out Forms

- **Form Fields**: Identifying input boxes, radio buttons, checkboxes in HTML

Automating Filling Out Forms

- **Form Fields**: Identifying input boxes, radio buttons, checkboxes in HTML
- **Scripts**: Using `selenium` to locate elements by ID/class/xpath and input data

Automating Filling Out Forms

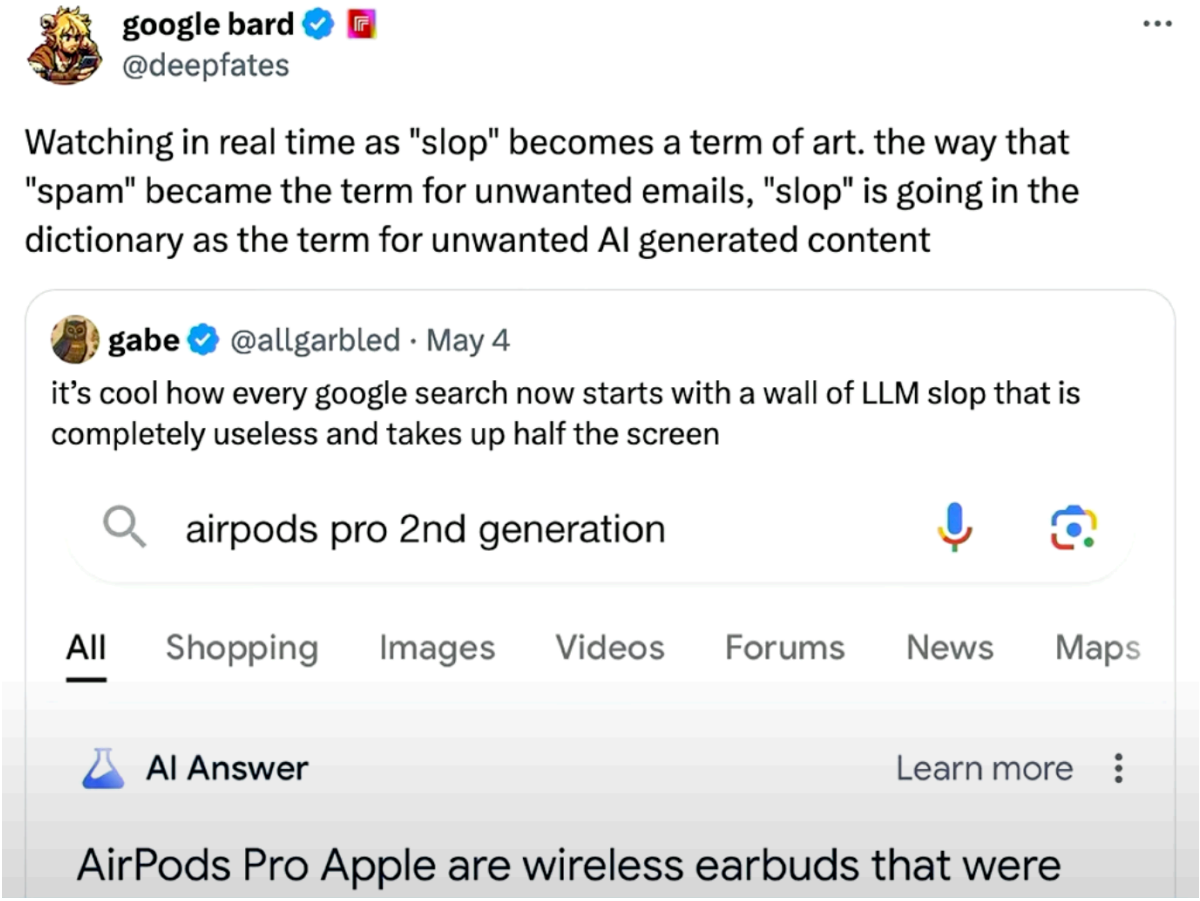
- **Form Fields**: Identifying input boxes, radio buttons, checkboxes in HTML
- **Scripts**: Using `selenium` to locate elements by ID/class/xpath and input data
- **Real-World Use**: Automating repetitive website sign-up processes, survey completion, or internal data-entry forms

Demo - Prompts, Tips & Tricks

Using AI != Slop

Using AI != Slop

Slop is using unreviewed output (like code) from AI models



Using AI != Slop

1. Working with AI involves having a system for effectively reviewing high quality outputs

Using AI != Slop

1. Working with AI involves having a system for effectively reviewing high quality outputs
2. Means having good systems for generating good outputs on the first place

Using AI != Slop

1. Working with AI involves having a system for effectively reviewing high quality outputs
2. Means having good systems for generating good outputs on the first place
3. Then learning to put in place a procedure for effective review and feedback on top of the outputs you get

Using AI != Slop

1. Working with AI involves having a system for effectively reviewing high quality outputs
2. Means having good systems for generating good outputs on the first place
3. Then learning to put in place a procedure for effective review and feedback on top of the outputs you get
4. **If it fixes your problem, it's working!**
    - The goal is to solve YOUR specific problem
    - Perfection is the enemy of done
    - A working automation that saves you time is better than a "perfect" solution that never ships

Connect With Me

 Course materials

 LinkedIn

 Twitter/X - @LucasEnkrateia

 YouTube - @automatalearninglab

 Email: lucasenkrateia@gmail.com