Original Edition

DaScient Press, Ltd. | 2025

ISBN: 9798271627309

[www.dascient.com/press](http://dascient.com/press)

[www.GitHub.com/dascient/makeitcute](http://www.GitHub.com/dascient/makeitcute)

Table of Contents

1.0 Python, But Make It Understandable 6

Word Bank (read this first) 6

1.1 What Python is — (and why people love it) 8

1.2 What “Syntax” MeaNs(with receipts) 9

Let’s see this in action 10

Mini Rulebook 13

1.3 print() — What it Actually Prints 16

Concept 17

Code 17

What You’ll See 17

Why It Matters 18

Your Turn 19

Reflection 21

1.4 Variables — Name Your Vibes 22

Concept 23

Code 24

What You’ll See 24

Why It Matters 25

Your Turn 26

Mini Tip 28

Reflection 29

1.5 Functions — Reusable Moves 30

Concept 31

Code 32

What You’ll See 32

Why It Matters 33

Your Turn 34

Mini Tip 36

Reflection 37

1.6 Loops — Repeat Without Boredom 38

Concept 39

Code ( for-loop ) 40

What You’ll See 40

Why It Matters 41

Code ( while-loop ) 42

What You’ll See 42

Why It Matters 43

Your Turn 44

Mini Tip 45

Reflection 46

1.7 Decisions — If, Elif, Else 47

Concept 47

Code 48

What You’ll See 49

Why It Matters 49

Your Turn 50

Mini Tip 51

Reflection 52

1.8 Comments & Errors — Your Gentle Safety Net 54

Concept 55

Code 55

What You’ll See 56

Why It Matters 56

Your Turn 57

Mini Tip 58

Reflection 60

1.9 Running Python — Three Easy Modes 61

Concept 62

Mode 1: REPL — “Text Python like a friend” 63

Mode 2: Script — “Save your masterpiece” 65

You’ll see: 66

Mode 3: Notebook — “Pretty notes meet code” 66

Why It Matters 68

Your Turn 69

Reflection 70

1.10 Mini Exercises 5 Minutes Each 71

Exercise A — The Greeter 71

Exercise B — The Friendly Counter 73

Exercise C — The Tiny Decision Bot 74

Exercise D — Function Glow-Up 75

Exercise E — Comment Queen 77

Reflection 78

1.11 A Very Short Cheat Sheet 79

Talking to Python 79

Storing Information 80

Repeating Actions 80

Decisions 81

Functions 81

Lists 82

Randomness 82

Quick Syntax Reminders 83

It’s officiaL — You’re a python baddie 84

Remember: 85

About DaScient 86

you’re officially a python baddie Baddie 87

1.0 Python, But Make It Understandable

# Word Bank (read this first)

• Python: A popular programming language that reads like clear English. Great for beginners and powerful for pros.

• Syntax: The rules for how you must write code so the computer understands it (spelling + grammar for code).

• Interpreter: The Python program that reads your code and runs it line by line.

• Terminal/Command Prompt: The text window where you run commands (including Python).

• Script: A file that contains Python code, usually ending in .py, that you run with the interpreter.

• Statement: A complete instruction (e.g., print("hi")).

• Expression: A piece of code that becomes a value (e.g., 2 + 2 becomes 4).

• Argument/Parameter: The data you pass into a function: "hello" is an argument to print.

• Variable: A named box that stores a value: name = "Nicole".

• String: Text wrapped in quotes: "Python".

• Integer / Float: Whole numbers (3) / decimals (3.14).

• Boolean: True or False.

• List: An ordered collection: ["tea", "matcha", "coffee"].

• Indentation: Spaces at the start of a line. In Python, this defines blocks (what’s “inside” a loop or function).

• Library/Module: Extra code you can import to add features. Example: random, datetime.

• pip: Python’s package installer (pip install something).

• Comment: Notes to humans in your code, ignored by Python. Start with #.

1.1 What Python is — (and why people love it)

Python is a general-purpose language: you can use it to automate tasks, analyze data, build websites, train AI models, or just tidy your files. It’s famous for readability. Many languages can do the same things; Python often does them with less code that’s easier to read and change.

Think of Python as a very patient assistant. You write clear instructions; Python executes them top to bottom.

1.2 What “Syntax” MeaNs(with receipts)

Syntax is just grammar for computers—the rules that keep your code from sounding like gibberish to Python’s interpreter.

When you speak, a sentence like “Coffee need I now” sounds wrong because it breaks English word order.

When you code, something like print"hello" sounds wrong to Python for the same reason—it breaks the rule for how a function call must look.

# Let’s see this in action

Correct syntax (Python understands):

print("Hello, Nicole!")

Why it works

• print → the function name—an instruction that already exists in Python.

• ( and ) → parentheses tell Python, “I’m about to give you some information.”

• "Hello, Nicole!" → the argument (the data you’re handing to the function).

• ) → closes the thought. You’re done talking; Python can act.

If you run this in a Python shell (>>> print("Hello, Nicole!")), you’ll see:

Hello, Nicole!

Incorrect syntax (Python panics):

print"Hello, Nicole!"

You forgot the parentheses.

When you click Run, Python responds with:

File "main.py", line 1

print"Hello, Nicole!"

^^^^^^^^^^^^^^^^

SyntaxError: Missing parentheses in call to 'print'. Did you mean print(...)?

That’s the interpreter being a grammar teacher—it’s telling you exactly which rule you broke.

Another common mistake:

Print("Hello")

That capital P breaks the rule that Python is case-sensitive.

print and Print are two totally different names.

Computers don’t assume what you meant; they follow what you typed.

# Mini Rulebook

1. Parentheses matter. Every opening ( needs a closing ).

2. Quotes must match. "..." or '...', but not both in the same string.

3. Indentation = structure. Four spaces tell Python what’s “inside” something (like a loop or function).

4. A colon (:) starts a block—what follows must be indented.

5. Case counts. Name ≠ name.

Your Turn — Quick Practice

Try this side by side:

*# Correct*

*print("My favorite color is pink.")*

*# Broken*

*print"My favorite color is pink."*

Run both.

• The first prints normally.

• The second throws a SyntaxError.

Now deliberately break a few rules:

• delete a quote,

• change a lowercase to uppercase,

• remove a colon after an if.

Watch the exact message Python gives you.

Each one is a clue about which grammar rule you bent.

If you can read an error and fix it, congratulations—you now speak enough Python to hold a polite, nerdy conversation.

You’re not just typing code; you’re learning the language one polite correction at a time.

Function: A named, reusable action. You “call” it with parentheses: print("hello").

1.3 print() — What it Actually Prints

When you click Run, Python’s interpreter becomes your digital assistant.

It reads each line you wrote, figures out what that line means, and performs it—like a barista taking an order, making the drink, and sliding it across the counter.

That’s all programming is: you write the order; Python fulfills it.

# Concept

print() is one of Python’s built-in functions—pre-made instructions ready for you to use.

When you “call” a function, you’re saying, “Hey Python, please run this recipe.”

# Code

print("Hello, world!")

# What You’ll See

Hello, world!

That line of text appears instantly in your console.

It’s not magic—it’s Python writing to something called standard output, a fancy term for “the screen.”

# 

# Why It Matters

Step-by-step, here’s what really happens:

1. Read: The interpreter scans your line and checks the grammar (syntax).

2. Evaluate: It sees a function named print.

3. Execute: It runs that function and hands it your argument "Hello, world!".

4. Output: The function sends that text to your console window.

Every line you write in Python goes through this read → evaluate → execute cycle.

You can use print() anytime you want to:

• show information to yourself while debugging,

• display results for users,

• or just make sure something actually ran.

# 

# Your Turn

1. Open a terminal window.

Type % python:

>>> print("Hi, I’m learning Python!”)

>>> print("This feels... oddly empowering.")

2. Run it by pressing enter.

Notice how Python executes top to bottom, one line after another.

3. Now add a third line that prints your name five times:

>>> print("Nicole " \* 5)

That \* 5 tells Python to repeat the string five times—your first glimpse of operators at work.

# Reflection

You just ran your first real command sequence:

• multiple print() calls,

• sequential execution,

• string repetition.

Every future Python program—no matter how advanced—still relies on this same rhythm.

So, when in doubt, remember:

>>> print() is Python’s way of talking back.

If you don’t see output, you probably forgot to ask.

1.4 Variables — Name Your Vibes

Imagine your computer as a closet and every variable as a labeled storage box.

You write a label, drop something inside, and later say, “Hey Python, open that box.”

That’s all a variable is — a name attached to a value.

# Concept

A variable stores data so you can reuse or modify it.

Think of it as a memory bookmark: Python remembers what you wrote even after that line finishes running.

Each variable has:

• a name → what you call the box

• a value → what’s inside

• a type → the kind of data stored (text, number, list, etc.)

# Code

creator = "Nicole"

followers = 1200

ratio = followers / 50

print("Creator:", creator)

print("Follower ratio:", ratio)

# What You’ll See

Creator: Nicole

Follower ratio: 24.0

# 

# Why It Matters

Let’s decode what happened:

1. creator = "Nicole" → You created a variable named creator that holds text (a string).

2. followers = 1200 → A variable holding a whole number (integer).

3. ratio = followers / 50 → Python calculated 1200 ÷ 50 and stored the result (float, because it’s a decimal).

4. The print() lines just retrieve and display those stored values.

Now Python’s memory looks roughly like this:

Name Type Value

creator str “Nicole”

followers int 1200

ratio float 24.0

# Your Turn

1. In your terminal, type:

name = "YourName"

age = 23

mood = "focused"

print("Hi,", name)

print("Age:", age)

print("Current mood:", mood)

Run it — Python repeats your words back like a little mirror.

2. Now change one value and run again.

See how Python remembers the latest version?

That’s assignment — the = operator replaces what’s in the box.

3. Try some quick math:

points = 45

bonus = 10

total = points + bonus

print("Total points:", total)

Every time you use +, -, \*, or /, Python computes the result and can store it again.

# Mini Tip

Variable names can’t start with numbers or contain spaces.

Use underscores (\_) if needed:

favorite\_color = "lavender"

Good naming is a love language between you and future-you.

# Reflection

You just taught Python to remember information and perform simple calculations — the same foundation used in every program, from TikTok algorithms to NASA simulations.

When in doubt, name your vibe.

If you can store it, you can change it.

And that’s how automation begins.

1.5 Functions — Reusable Moves

If variables are labeled boxes, functions are your routines—the “how-to” guides you can call whenever you need the same task done again.

Instead of copying the same lines over and over, you write them once, name them, and reuse them forever.

It’s like saving your skincare routine as a preset: cleanse(), tone(), moisturize().

# Concept

A function is a block of code that:

1. takes input (we call those parameters or arguments),

2. does something with them, and

3. (optionally) returns a result you can use.

Python already comes with dozens of built-in functions (print, len, type, round), but you can also create your own.

# Code

def compliment(name):

return f"{name}, your code is cleaner than my skincare shelf."

msg = compliment("Nicole")

print(msg)

# What You’ll See

Nicole, your code is cleaner than my skincare shelf.

# Why It Matters

• def → short for define. It tells Python: “I’m writing a new function.”

• compliment → the function’s name. Use clear names like you label variables.

• (name) → a parameter — the info your function needs to work.

• return → hands back a value to whoever called it.

When you run the script, Python reads the def block first and stores it in memory.

Nothing happens until you call the function (compliment("Nicole")).

Then it executes the code inside and hands the result back to msg, which you print.

Functions let you bundle logic into named actions so your code stays organized and aesthetic.

# Your Turn

1. Write a simple function that adds two numbers:

def add(a, b):

return a + b

print(add(3, 5))

2. Try changing the numbers, or even passing variables instead of literals.

x = 10

y = 7

print(add(x, y))

3. Now make it cuter:

def affirm(name):

print(f"{name}, remember to hydrate and debug gently.")

affirm("Nicolette")

Each time you call affirm("Someone"), it prints a personalized message.

# Mini Tip

If you forget to write return, the function still runs but gives back None.

Think of None as Python’s way of saying, “I did it, but there’s nothing to hand back.”

# Reflection

Functions are your future superpower. Every big app is just a well-organized collection of tiny functions working together.

When you find yourself copy-pasting code, pause and say: “Time to make this a function.”

You’re not just writing lines anymore — you’re building skills you can reuse everywhere.

1.6 Loops — Repeat Without Boredom

You know that one TikTok you’ve re-watched twenty times?

That’s a loop—something running again and again until you stop it.

In Python, loops help you automate repetition so you don’t copy-paste lines forever.

# Concept

A loop tells Python: “Keep doing this until I say otherwise.”

Two main kinds exist:

1. for-loops — repeat a fixed number of times or through a collection.

2. while-loops — repeat while a condition remains true.

# Code ( **for-loop** )

for drink in ["tea", "matcha", "coffee"]:

print("Sipping:", drink)

# What You’ll See

Sipping: tea

Sipping: matcha

Sipping: coffee

# Why It Matters

• for → starts the loop.

• drink → a temporary variable that takes one item from the list each round.

• in → tells Python where to look for those items.

• The indented line runs once per item.

Python automatically stops when the list ends.

You just built your first micro-automation—no copy-pasting needed.

# Code ( **while-loop** )

count = 3

while count > 0:

print("Posting in", count)

count = count - 1

print("We’re live!")

# What You’ll See

Posting in 3

Posting in 2

Posting in 1

We’re live!

# Why It Matters

A while-loop keeps going while its condition is true.

Here, count > 0 starts true, so the loop runs.

Each time it subtracts 1; eventually count becomes 0, the condition is false, and Python moves on.

If you forget to change the condition (for example, remove count = count - 1), the loop never ends—an infinite loop.

Your console will look frozen.

To stop it, press Ctrl + C

# Your Turn

1. Make your own affirmation loop:

for i in range(3):

print("You got this, queen!")

2. Create a countdown using while:

energy = 5

while energy > 0:

print("Working…", energy)

energy -= 1

print("Break time!")

3. Bonus: change the number or message until it feels you-coded-it cute.

# Mini Tip

range(n) creates a sequence of numbers from 0 to n-1.

So range(3) gives 0, 1, 2 — three rounds total.

You can even start and stop it yourself: range(2, 5) → 2, 3, 4.

# Reflection

Loops are where Python starts to feel magical—you write one line, and it handles the repetition.

Every scheduler, reminder bot, and auto-poster you’ll build later depends on this concept.

So when life gives you boring repetition, remember:

Don’t repeat manually—loop beautifully.

• REPL: “Read–Eval–Print Loop.” An interactive prompt where you type code and see results immediately.

1.7 Decisions — If, Elif, Else

Sometimes your code needs to make a choice—like deciding whether it’s time to work, rest, or reward yourself with a bubble tea.

That’s where conditional statements come in: they let your program branch off depending on the situation.

# Concept

if, elif, and else are Python’s way of saying:

• if something is true → do this.

• elif (short for else if) → try another condition.

• else → if none of the above apply, do this instead.

They read top to bottom, checking each condition once.

# Code

mood = "tired"

if mood == "tired":

print("Break time. Hydrate.")

elif mood == "thriving":

print("Batch record content.")

else:

print("Check analytics, adjust gently.")

# What You’ll See

Break time. Hydrate.

# Why It Matters

Here’s how Python thinks:

1. It looks at the first line: if mood == "tired".

2. It checks whether the condition is True. (== means “is equal to.”)

3. If it’s True, it runs the indented code right below.

4. If not, it skips to the next option (elif or else).

Because mood equals "tired", the first condition passes and Python runs that block.

The colons (:) and indentation show where each decision’s code begins and ends.

# Your Turn

1. Copy the example into your REPL.

Try changing the variable at the top:

mood = "thriving"

Run it again — what happens?

2. Add a new branch for another vibe:

elif mood == "creative":

print("Light a candle and start scripting.")

3. Keep testing different moods until you’ve got a full routine encoded in logic.

# Mini Tip

• == checks equality.

• = assigns a value (don’t mix them up!).

• You can chain comparisons:

if 8 <= hour < 12:

print("Morning grind.")

• Combine conditions with and / or:

if tired and not hydrated:

print("Water first, then work.")

# Reflection

Decisions are the “if–then” logic behind every app, from filters that hide negative comments to bots that send reminders.

They make your programs adaptive—able to respond instead of just repeat.

When life gives you conditions, remember:

Code it like you mean it. Logic is just intuition written down.

• Traceback: The error message when something goes wrong. It’s a clue, not a catastrophe.

1.8 Comments & Errors — Your Gentle Safety Net

Every coder, even the pros at NASA and TikTok, breaks their code daily.

The difference between frustration and flow is learning how to read the red text calmly and leave yourself little notes for next time.

That’s where comments and tracebacks come in — your gentle safety net.

# Concept

• Comments are notes you write to yourself (and future you).

Python ignores them completely.

• Errors (called exceptions) are Python’s polite way of saying,

“Hey, something here doesn’t make sense yet.”

# Code

# This script reminds me to stay hydrated

print("Drink water!")

# Oops, let's cause an error on purpose

print(Drink water!)

# What You’ll See

Drink water!

File "main.py", line 4

print(Drink water!)

^^^^^

SyntaxError: invalid syntax

# Why It Matters

• Lines that start with # are comments. Python skips them when it runs your code.

• The second print() line breaks the rulebook: missing quotes make "Drink water!" look like a variable instead of text.

• Python points you straight to the problem area and names the issue (SyntaxError).

That red message is a traceback — a breadcrumb trail showing:

1. The file name and line number where it stopped.

2. A caret (^) under the exact spot it got confused.

3. The type of error and its short description.

It’s not an insult; it’s a clue.

# Your Turn

1. Add your own helpful comment:

# This program motivates me before work

print("Let's do this!")

2. Now make a small typo on purpose — delete one quote or misspell print.

Run it and read the traceback.

Can you tell what kind of error it is and which line caused it?

3. Fix it, run again, and feel that tiny dopamine hit.

# Mini Tip

Common early errors:

Error Translation

NameError You used a variable before defining it.

TypeError You mixed incompatible types (like adding a number to a string).

IndentationError Your spaces don’t line up.

SyntaxError Python couldn’t even start reading—it’s like a broken sentence.

To stop a long-running program:

• Press Ctrl + C (in a terminal)

# Reflection

Mistakes are part of the rhythm.

Each red error is just Python asking for clarity, not scolding you.

Comments are your love notes to the future — they’ll thank you later when your codebase grows.

When you see that bright red text, smile and say:

“Okay, Python, I see your point.”

Then fix it and keep sipping your latte.

1.9 Running Python — Three Easy Modes

You’ve learned how to write code — now let’s talk about how to run it.

Think of Python as a playlist: the songs (your code) don’t play until you hit “run.”

There are a few ways to press play, and each one fits a different vibe.

# Concept

Python can run in three main modes:

1. REPL (Read–Eval–Print Loop) — instant feedback, like texting Python directly.

2. Script Mode — save your code to a .py file and run it anytime.

3. Notebook Mode — a blend of text + code cells, perfect for organized creators or tutorials.

# Mode 1: REPL — “Text Python like a friend”

How to open it:

• On your computer, open Terminal (macOS/Linux) or Command Prompt (Windows).

• Type python or python3 and press Enter.

• You’ll see >>> — that’s the REPL prompt saying, “Hey girl, I’m listening.”

Try it:

>>> print("Hello from the REPL!")

Hello from the REPL!

Each time you press Enter, Python reads your message, runs it, and replies instantly.

Exit anytime:

• Ctrl + D (Mac/Linux)

• Ctrl + Z then Enter (Windows)

Use the REPL when you’re experimenting — it’s your quick sandbox for testing ideas.

# Mode 2: Script — “Save your masterpiece”

When you have more than a few lines, you’ll want to save your work.

Steps:

1. Open any text editor.

2. Create a file named hello.py.

3. Inside, write:

print("This is my first saved script.")

4. Run it:

• In Terminal, type python hello.py and press Enter.

# You’ll see:

This is my first saved script.

That’s your program — permanent and portable.

# Mode 3: Notebook — “Pretty notes meet code”

If you like structure, visuals, and journaling your experiments, you’ll adore Jupyter Notebooks.

They let you mix text cells (for explanations) and code cells (for execution).

Perfect for project logs, tutorials, or documenting your bot’s progress.

How to try it:

1. In Terminal, install Jupyter:

pip install notebook

2. Then run:

jupyter notebook

3. A browser tab opens where you can click New → Python 3 Notebook and start coding interactively.

Platforms like Google Colab also offer free notebook environments — no setup required.

# Why It Matters

Each mode fits a different goal:

Mood Tool Use Case

Curious & testing ideas REPL Instant results for one-liners

Focused & building something Script Reusable files & automation

Organized & teaching Notebook Blending code + notes + results

Once you master all three, you’ll never wonder where to code again.

# Your Turn

Try all three within the next day:

1. Open a REPL and make it say hello.

2. Save a .py file and run it from Terminal.

3. If you’re feeling ambitious, open a Google Colab notebook and write your name in print().

Screenshot your first success — it’s your first real Python flex.

# Reflection

Learning to run Python is like learning to perform your own spells:

Sometimes you whisper them in real-time (REPL), sometimes you record them for later (script), and sometimes you annotate your whole grimoire (notebook).

No matter the method, the magic’s the same — your words, executed by logic.

“The power isn’t in the platform; it’s in the precision.”

1.10 Mini Exercises 5 Minutes Each

Think of this section as your warm-up playlist.

Nothing heavy, just a few quick moves to lock in what you’ve learned so far.

Each challenge takes about five minutes and gives you an instant result.

# Exercise A — The Greeter

Goal: make Python say hello — properly.

name = "Nicole"

print("Hi", name + "!")

print("Baddie" \* 5)

Try:

• Change the name to yours.

• Add another print line that says how you’re feeling today.

Concept check: variables, strings, print(), string multiplication ("Baddie" \* 5)

# Exercise B — The Friendly Counter

Goal: use a for-loop and range() to create a cheerful to-do list.

for i in range(1, 4):

print(f"Post #{i}: Done.")

Try:

• Change the range to range(1, 6) to post five times.

• Replace “Post” with anything — “Stretch,” “Hydrate,” “Check analytics.”

Concept check: loops, range, f-strings

# Exercise C — The Tiny Decision Bot

Goal: make Python give advice based on the time of day.

hour = 22

if hour >= 22:

print("Log off gently.")

else:

print("Do one focused task.")

Try:

• Adjust the number and re-run (hour = 15, hour = 9).

• Add an elif branch for your creative hours.

Concept check: if/elif/else, comparison operators

# Exercise D — Function Glow-Up

Goal: combine math + functions + print power.

def vibe\_score(likes, comments):

return likes \* 0.7 + comments \* 1.8

print(vibe\_score(120, 18))

Try:

• Swap in your own numbers.

• Wrap the output in a message like:

score = vibe\_score(200, 30)

print(f"Your vibe score is {score:.2f} Baddie")

Concept check: functions, arguments, math, formatted strings

# Exercise E — Comment Queen

Goal: document what you just did.

# My first set of exercises!

# Feeling confident and slightly caffeinated ☕️

Add those comments at the top of your code.

They won’t change how it runs — they just leave a breadcrumb for future you.

Concept check: comments, clarity, self-documentation

# Reflection

If you finished all five, you just touched every building block of Python:

• variables

• loops

• decisions

• functions

• comments

These are the only essentials you need to start automating the fun stuff.

Each exercise you finish is a micro win — proof you can make a computer listen and respond.

So take a sip, stretch your wrists, and say it with pride:

“I can code. And I made it cute.”

1.11 A Very Short Cheat Sheet

This is your pocket-sized reminder of everything you’ve learned so far.

Python’s not about memorizing — it’s about knowing what to look up and why it works.

Keep this page handy whenever you feel lost; it’s your little logic mirror.

# Talking to Python

|  |  |  |
| --- | --- | --- |
| Action | Code | Meaning |
| Say something | print(“Hello”) | Sends text to the screen. |
| Leave a note | # comment | Python ignores everything after #. |
| Combine text | “Hi “ + “there” | Joins two strings together. |
| Repeat text | “Cutie” \* 3 | Repeats the string three times. |

# Storing Information

|  |  |  |
| --- | --- | --- |
| Action | Code | Meaning |
| Create a variable | name = “Nicole” | Stores text or numbers for later use. |
| Add numbers | total = 5 + 3 | Basic math. |
| Use parentheses | (2 + 3) \* 4 | Groups operations like regular math. |
| Check type | type(name) | Tells you if it’s text, number, etc. |

# Repeating Actions

|  |  |  |
| --- | --- | --- |
| Action | Code | Meaning |
| Loop through a list | for x in [“tea”, “matcha”, “coffee”]: | Runs once per item. |
| Loop a set number of times | for i in range(3): | Runs 3 times (0, 1, 2). |
| While something is true | while energy > 0: | Repeats until condition is false. |

# Decisions

|  |  |  |
| --- | --- | --- |
| Action | Code | Meaning |
| Basic check | if mood == “happy”: | Runs code only if true. |
| Add another option | elif mood == “tired”: | Checks next condition. |
| Default action | else: | Runs if nothing else matched. |
| Compare numbers | >, <, >=, <=, ==, != | Standard comparison operators. |

# Functions

|  |  |  |
| --- | --- | --- |
| Action | Code | Meaning |
| Define one | def greet(): | Starts a reusable block. |
| Call it | greet() | Runs the function. |
| Return a result | return “hi” | Sends data back to where it was called. |
| Use parameters | def add(a, b): return a + b | Inputs → Output. |

# Lists

|  |  |  |
| --- | --- | --- |
| Action | Code | Meaning |
| Make a list | drinks = [“tea”, “matcha”, “coffee”] | Group of values. |
| Access by position | drinks[0] | Gets first item (“tea”). |
| Add an item | drinks.append(“boba”) | Adds to end of list. |
| Length | len(drinks)” | Counts items. |

# Randomness

|  |  |  |
| --- | --- | --- |
| Action | Code | Meaning |
| Import a library | import random | Unlocks extra tools. |
| Pick random item | random.choice(drinks) | Selects one element at random. |

# Quick Syntax Reminders

• Python cares about indentation (4 spaces = inside a block).

• Case matters: print ≠ Print.

• Strings use quotes " " or ' '.

• Every open (, [, or { needs a matching closer.

• Errors are feedback, not failure.

It’s officiaL — You’re a python baddie

Look at you. You just wrote real code. You spoke to a machine and it listened — politely, even.

That’s not luck, babe. That’s logic in heels.

This tiny book was never just about syntax; it was about power — the kind that clicks quietly while everyone else still thinks code looks scary.

Now you know the secret: it’s just words, patterns, and confidence. And you’ve got all three.

So go show another girlie that tech isn’t locked behind jargon or hoodies.

Send her this book, start a study date, or drop a cute TikTok showing your first script.

Share the knowledge like lip gloss — everyone looks better when they shine.

# Remember:

You don’t need permission to be intelligent.

You just need Wi-Fi, curiosity, and maybe a good playlist.

And if this felt good — if you felt that little rush of “wait, I can do this” —

just wait until Book 2.

We’re about to automate the boring stuff, guard our peace with bots, and make the internet a softer place for smart girls everywhere.

# About DaScient

DaScient was founded with one mission:

to make intelligence accessible, aesthetic, and achievable for girlies worldwide.

We believe learning should feel like self-care — stylish, empowering, and a little bit sparkly.

Because brilliance shouldn’t need a dress code.

Keep coding cute. Keep learning loud.

And never forget:

You are the algorithm.

# you’re officially a python baddie Baddie

“you did that.”

look at you. you just wrote real code.

you spoke to a machine and it listened — politely, even.

that’s not luck, babe.

that’s logic in heels.

this tiny workbook was never just about syntax;

it was about power — the quiet, elegant kind

that hums between your fingers when you realize

you can automate your own world.

now you know the secret:

it’s just words, patterns, and confidence.

and you’ve got all three.

“code like you text your future self — concise, intentional, and slightly flirty.”

share the glow.

go show another girlie that tech isn’t locked behind jargon or hoodies.

send her this book. start a study date.

make a cute tiktok showing your first script.

share the knowledge like lip gloss —

everyone looks better when they shine.

and if this felt good —

if you caught that little spark of “wait, I can do this” —

just wait for book two.

we’re about to automate the boring stuff,

guard our peace with bots,

and make the internet a softer, smarter place

for baddies everywhere.

import selflove as default

stay curious. stay soft. stay running your code.

DaScient Press — Accessible Intelligence for Every Girlie™