ZainSpire

PYTHON HANDWRITTEN NOTES

20 pages to learn, practise & revise



Simple explanations | Code + output | Practice prompts

Start small. Keep coding.

ZainSpire | Python Notes

01 / 20 — START WITH PYTHON

Small instructions. Useful results.

WHAT IS PYTHON?

Python is a programming language.
Use it to automate tasks, build apps and explore data.

Write code → Run it → See output

YOUR FIRST SCRIPT

hello.py

```
# A friendly greeting
name = "Zain"
print("Salam,", name)
print(2 + 3)
```

OUTPUT

```
Salam, Zain
5
```

print() displays a result.

RUN IT

- Save the file as hello.py
- Terminal: python hello.py
- In an editor, you can also use Run.

SPACES MATTER

```
if 5 > 2:
    print("Yes!")
```

4 spaces

- A colon starts this block.
- Indent the lines inside it.
- Python is case-sensitive.

COMMENTS

```
# explains your code; Python ignores the comment.
```

TRY IT

Print your name twice. Predict the output, then run it.

02 / 20 — VARIABLES & TYPES

A NAME FOR A VALUE

A variable is a name that refers to a value.

```
name = "Zain"      → str
age = 20           → int
price = 99.5       → float
is_ready = True    → bool
result = None      → NoneType
```

CHECK THE TYPE

```
print(type(age).__name__)
```

Output: int

Quotes make text: "20" is not the number 20.

INPUT + CONVERSION

input() always returns a string.

```
age = int(input("Age: "))
print(age + 1)
```

Example input: 20
Output: 21

```
int("20") → 20
float("2.5") → 2.5
str(20) → "20"
```

! Invalid numeric text can raise ValueError.

GOOD NAMES

- Use letters, digits and underscores.
- Do not start with a digit or use a keyword.

✓ Good: user_name, score2

✗ Wrong: 2score, user-name

- name and Name are different names.

TRY IT

- Store your name in name. Print it with your age.
- Remember: = assigns a value; == compares values.

03 / 20 — OPERATORS & DECISIONS

NUMBER TOOLS

For a = 7 and b = 2:

a + b → 9	a - b → 5
a * b → 14	a / b → 3.5
a // b → 3	a % b → 1
a ** b → 49	

- // is floor division; % gives the remainder.
- Use parentheses to make the order clear.

COMPARE + COMBINE

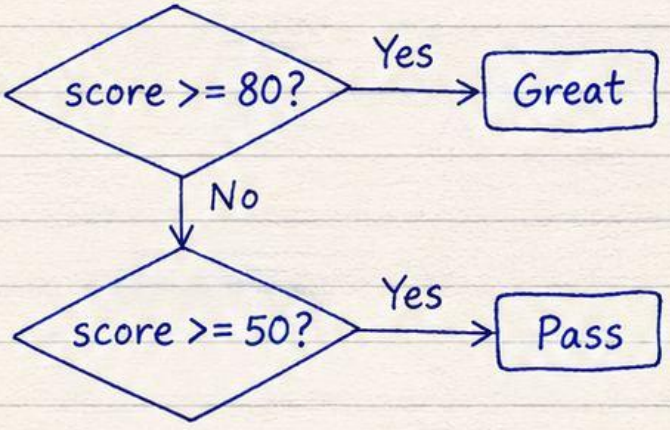
== equal	!= not equal
< less	> greater
<= at most	>= at least

- **and**: both conditions are true
- **or**: at least one is true
- **not**: reverses truth
- score += 1 means score = score + 1.

MAKE A DECISION

```
score = 76
if score >= 80:
    print("Great")
elif score >= 50:
    print("Pass")
else:
    print("Try again")
```

Output: Pass



- Only the first matching branch runs.

COMMON MISTAKE

⚠ Use == to compare, not =.

- Keep the colon and indent each branch.

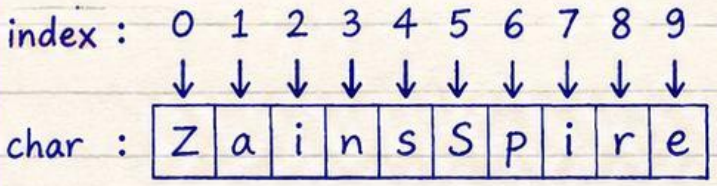
TRY IT

Use n % 2 == 0 to check whether n is even.

04 / 20 — STRINGS: WORK WITH TEXT

INDEX + SLICE

```
text = "ZainsSpire"
```




```
text[0]     → "Z"
text[-1]    → "e"
text[:4]    → "Zain"
text[::2]   → "Zispr"
text[::-1]  → "eripSsniaZ"
```

- Slice: text[start:stop:step]
- The stop index is excluded.

USEFUL METHODS

```
name = "  Zain  "

name.strip()                → "Zain"
"Zain".upper()              → "ZAIN"
"Zain".lower()              → "zain"
"Zain".replace("Z","R")     → "Rain"
"Zain".find("a")            → 1
"Zain".count("a")           → 1
"Zain".endswith("in")       → True
len("Zain")                 → 4
```

BUILD A MESSAGE

```
name = "Zain"

f"Salam, {name}!"           → "Salam, Zain!"
"Hi, {}".format(name)       → "Hi, Zain"
"-".join(["py", "notes"])   → "py-notes"
```

REMEMBER

- Single or double quotes create strings.
- Triple quotes can span multiple lines.
- \n is a newline; \t is a tab.
- Strings are immutable: methods return new strings.

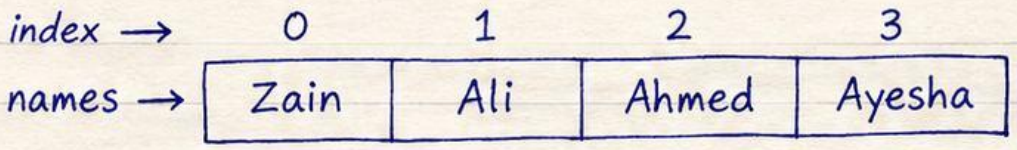
TRY IT

→ Clean " python " and convert it to uppercase.

05 / 20 — LISTS & TUPLES

LIST = CHANGEABLE COLLECTION

names = ["Zain", "Ali", "Ahmed", "Ayesha"]



- names[0] → "Zain"
- names[-1] → "Ayesha"
- names[:2] → ["Zain", "Ali"]
- names[1] = "Umar"
- Lists keep order and allow duplicates.

LIST TOOLBOX

- append(x): add at the end
- insert(i, x): add at an index
- pop(i): remove and return an item
- remove(x): remove the first matching value
- sort(): sort the list in place
- reverse(): reverse the list in place

```
nums = [3, 1, 2]
nums.sort()
print(nums)
```

Output: [1, 2, 3]

! Do not write nums = nums.sort(): sort returns None.

TUPLE = FIXED SEQUENCE

```
point = (3, 5)
x, y = point
print(x + y)
```

Output: 8

- one = (7,) # the comma matters
- (1, 1, 2).count(1) → 2
- (4, 8).index(8) → 1
- Tuple items cannot be reassigned.

TRY IT

- Create scores = [8, 5, 9]. Append 7, then find sum(scores).
- Answer: 29.

06 / 20 — DICTIONARIES & SETS

DICTIONARY = KEY → VALUE

```
student = {"name": "Zain", "score": 90}
student["name"] → "Zain"
student.get("age", 0) → 0
student["score"] = 95
```

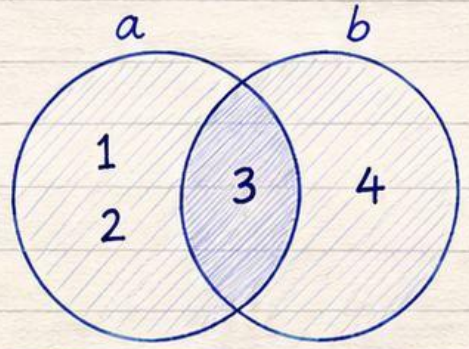
- Dictionaries keep insertion order.
- Keys are unique and must be hashable.
- Setting an existing key replaces its value.

DICTIONARY TOOLBOX

- keys(): a view of keys
- values(): a view of values
- items(): a view of (key, value) pairs
- update({…}): add or update entries
- pop(key): remove and return a value
- Missing key with [] → KeyError.
- get() can provide a default.

SET = UNIQUE VALUES

```
a = {1, 2, 3}
b = {3, 4}
a | b → {1, 2, 3, 4}   union
a & b → {3}            intersection
a - b → {1, 2}         difference
a ^ b → {1, 2, 4}      symmetric difference
```



SET TOOLBOX

- add(x), discard(x), remove(x), clear()
- remove(x) raises KeyError if x is missing.
- Sets have no indexing; display order may vary.
- Set elements must be hashable: no lists inside.
- {} is an empty dictionary; set() is an empty set.

TRY IT

```
len(set([2, 2, 5, 5, 8])) → 3
```

07 / 20 — LOOPS: REPEAT SMARTLY

FOR: VISIT EACH ITEM

```
for n in range(1, 4):
    print(n)
```

Output:

```
1
2
3
```

→ range(start, stop, step) excludes stop.

→ range(0, 6, 2) gives 0, 2, 4.

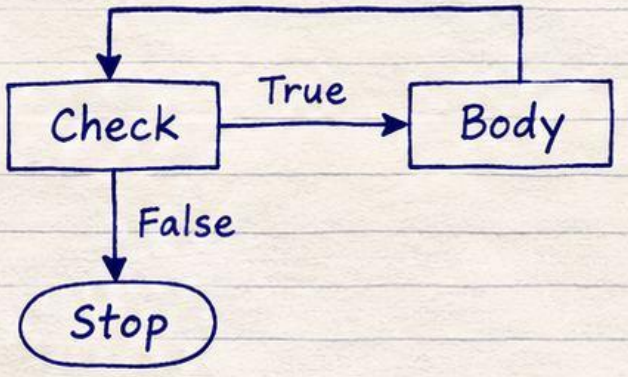
WHILE: REPEAT WHILE TRUE

```
count = 3
while count > 0:
    print(count)
    count -= 1
```

Output:

```
3
2
1
```

→ Update the condition to avoid an endless loop.



CONTROL THE LOOP

- **break**: leave the loop now
- **continue**: skip to the next iteration
- **pass**: do nothing; a placeholder

→ A loop's **else** runs if no break occurs.

INDEX + VALUE

```
names = ["Zain", "Ali", "Ayesha"]
for i, name in enumerate(names, start=1):
    print(i, name)
```

Output:

```
1 Zain
2 Ali
3 Ayesha
```

TRY IT

```
for n in range(5):
    if n == 2:
        continue
    print(n)
```

Output:

```
0
1
3
4
```

08 / 20 — FUNCTIONS & RECURSION

WRITE ONCE, CALL AGAIN

```
def greet(name="Zain"):
    return f"Salam, {name}!"
print(greet())
```

Output: Salam, Zain!

- name is a parameter; a supplied value is an argument.
- A default is used when no argument is supplied.
- return sends a result back; print displays it.
- Without an explicit return, a function returns None.

ARGUMENTS

```
def add(a, b):
    return a + b
```

add(2, 3) → 5

add(b=3, a=2) → 5

- Use positional or keyword arguments.

SCOPE

- Names created inside a function are usually local.
- global lets you reassign a module-level name.
- Prefer passing inputs and returning results.

RECURSION = A FUNCTION CALLS ITSELF

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

For non-negative integers:

factorial(4) → 24

CALL CHAIN:

4 × 3 × 2 × 1 × 1 = 24

```
factorial(4)
  ↳ 4 × factorial(3)
      ↳ 4 × 3 × factorial(2)
          ↳ 4 × 3 × 2 × factorial(1)
              ↳ 4 × 3 × 2 × 1 × factorial(0)
                  ↳ 4 × 3 × 2 × 1 × 1
```

- The base case stops the calls.
- Each step must move toward it.

TRY IT

- Write double(n) that returns n * 2.
- double(6) should return 12.

09 / 20 — MODULES, PIP & VENV

REUSE CODE WITH IMPORT

- A module is a file of Python code.
- The standard library is included with Python.

```
import math
print(math.sqrt(81))
```

Output: 9.0

```
from random import randint
```

- randint(1, 6) gives an integer from 1 to 6.
- You can import your own .py files too.

KEEP PROJECTS SEPARATE

- Create an environment:

```
python -m venv .venv
```

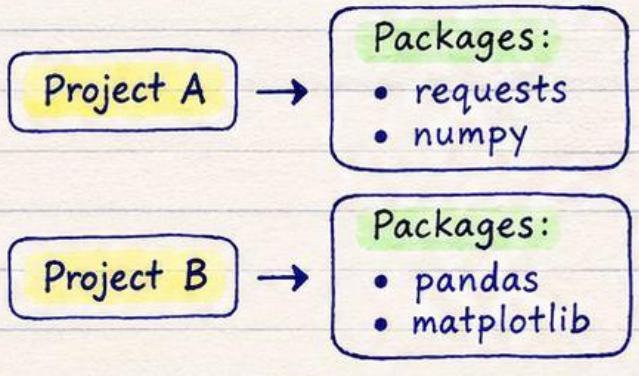
- Activate in Windows PowerShell:

```
.venv\Scripts\Activate.ps1
```

- Activate on macOS / Linux:

```
source .venv/bin/activate
```

- Leave it with: `deactivate`



PACKAGES: TERMINAL COMMANDS

```
python -m pip install requests
python -m pip freeze > requirements.txt
python -m pip install -r requirements.txt
```

- pip installs third-party packages.
- freeze records installed versions.

RUN ONLY WHEN STARTED DIRECTLY

```
def main():
    print("Zain")

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

- The guard prevents this call when imported.

FIRST-TIME SETUP

- Install Python 3 from python.org.
- Check in a terminal:

```
python --version
```

! On some systems use `python3` instead of `python`.

TRY IT

- Create a project folder and its own .venv.

10 / 20 — FILES: SAVE YOUR WORK

WRITE A TEXT FILE

```
with open("notes.txt", "w", encoding="utf-8") as f:
    f.write("Zain\nPython")
```

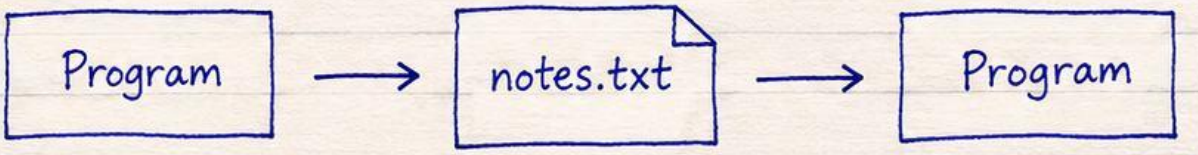
→ with closes the file when the block ends.

READ IT BACK

```
with open("notes.txt", "r", encoding="utf-8") as f:
    text = f.read()
print(text)
```

Output:

```
Zain
Python
```



CHOOSE A MODE

- r: read; the file must exist
- w: write; replaces existing contents
- a: append at the end
- x: create; fail if the file exists
- b: binary mode, e.g. rb for an image
- +: read and write, e.g. r+

WARNING: w clears an existing file.

READING TOOLS

- read(): read remaining content
- readline(): read the next line
- readlines(): read remaining lines into a list
- for line in f: process one line at a time

→ Use text strings in text mode; bytes in binary mode.

TWO FILES, ONE BLOCK

```
with open("a.txt") as a, open("b.txt") as b:
    print(a.read(), b.read())
```

→ Both files must exist for this example.

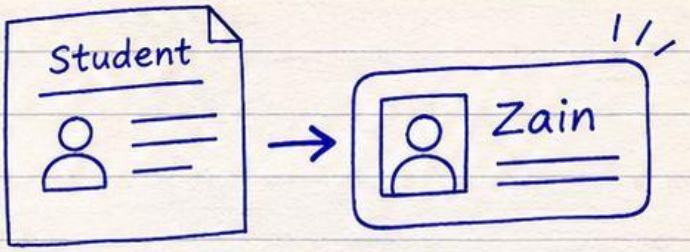
TRY IT

- Append "\nPractice daily" to notes.txt using mode a.

11 / 20 — CLASSES & OBJECTS

CLASS → OBJECT

- A class describes a kind of thing.
- An object is one instance of that class.



BUILD A STUDENT

```
class Student:
    school = "ZainSpire"
    def __init__(self, name):
        self.name = name
    def greet(self):
        return f"Hi, {self.name}"

s = Student("Zain")
print(s.greet())
```

Output: Hi, Zain

WHAT EACH PART DOES

- __init__ sets up a new instance.
- self refers to the instance receiving the call.
- self.name is an instance attribute.
- school is a class attribute.
- s.greet() calls the method for s.

CLASS VS INSTANCE

- Student.school = "Python Club"
  → Changes the class attribute.
- s.school = "Local Club"
  → Creates an attribute on s that shadows the class value.
- Other instances still use the class value unless overridden.

STATIC METHOD

- Inside a class:

```
@staticmethod
def add(a, b):
    return a + b
```

- It receives neither self nor cls automatically.

TRY IT

- Add a score attribute to Student.
- Create Zain with score 90 and print the score.

12 / 20 — INHERITANCE & OOP TOOLS

REUSE + EXTEND

```
class Person:
    def __init__(self, name):
        self.name = name

class Student(Person):
    def __init__(self, name, score):
        super().__init__(name)
        self.score = score

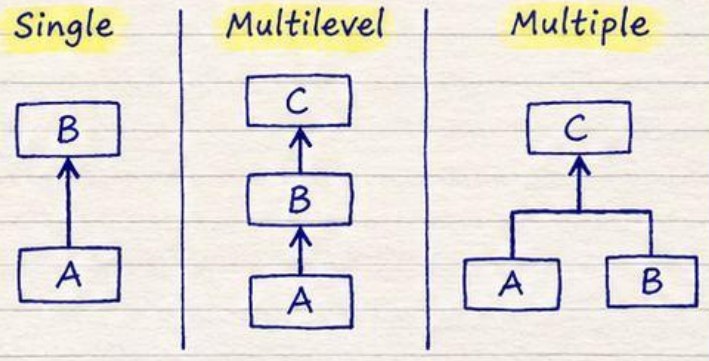
s = Student("Zain", 90)
print(s.name)
```

Output: Zain

→ Override a method to change inherited behaviour.

THREE SHAPES

- Single: A → B
- Multilevel: A → B → C
- Multiple: A + B → C



→ super() follows Python's method resolution order.

CLASS METHOD

- Inside a class:

```
@classmethod
def label(cls):
    return cls.__name__
```

- classmethod receives the class as cls.
- staticmethod receives neither cls nor self.

PROPERTY + SETTER

- Inside a class:

```
@property
def score(self):
    return self._score

@score.setter
def score(self, value):
    if value < 0:
        raise ValueError("Negative score")
    self._score = value
```

- Use obj.score like an attribute.
- Assign obj.score = 90 to call the setter.

SPECIAL METHODS

- __str__: text for str(obj)
- __len__: result of len(obj)
- __add__: custom behaviour for obj + other

→ These are called dunder methods.

TRY IT

- Explain the difference between self and cls.

self refers to an instance (object) and is used to access instance data/methods. cls refers to the class itself and is used in class methods to access class-level data/methods (e.g., cls.__name__).

13 / 20 — ERRORS & EXCEPTIONS

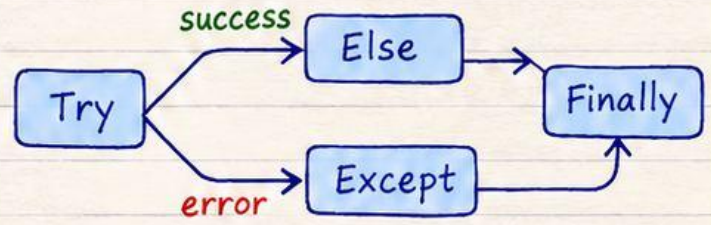
HANDLE EXPECTED FAILURES

```
try:
    age = int(input("Age: "))
except ValueError:
    print("Enter a whole number")
else:
    print(age + 1)
finally:
    print("Done")
```

- Input 20 → prints 21, then Done.
- Input abc → prints Enter a whole number, then Done.

FOLLOW THE FLOW

- try: attempt the operation
- except: handle a matching error
- else: run if try finishes without an exception
- finally: cleanup on exit from the statement



COMMON ERRORS

- ValueError: wrong value, e.g. int("abc")
- TypeError: wrong kind of value, e.g. "2" + 3
- ZeroDivisionError: division by zero
- KeyError: missing dictionary key
- IndexError: sequence index is out of range
- FileNotFoundError: a requested file is missing

RAISE YOUR OWN ERROR

```
score = -1
if score < 0:
    raise ValueError("Negative score")
```

- raise reports an invalid state.
- Catch specific exceptions you can handle.

COMMON MISTAKE

⚠ Do not hide every error with except: pass.

- Read the final line of the traceback first.

TRY IT

- Handle division by zero and print a helpful message.

14 / 20 — MODERN PYTHON TOOLKIT

TYPE HINTS

```
def greet(name: str) -> str:
    return f"Hi, {name}"

scores: list[int] = [80, 90]
student: tuple[str, int] = ("Zain", 20)
marks: dict[str, int] = {"Zain": 90}
user_id: int | str = "Z01"
```

→ Hints help readers and tools.

→ They do not enforce types at runtime.

MATCH / CASE

```
command = "start"
match command:
    case "start":
        print("Go")
    case "stop":
        print("Stop")
    case _:
        print("Unknown")
```

Output: Go

→ _ is the fallback pattern.

ASSIGN INSIDE AN EXPRESSION

```
items = [2, 4, 6]
if (n := len(items)) > 2:
    print(n)
```

Output: 3

→ := assigns a value within an expression.

MERGE DICTIONARIES

```
a = {"x": 1}
b = {"x": 2, "y": 3}
```

a | b → {"x": 2, "y": 3}

→ For shared keys, the right-hand value wins.

→ a |= b updates a in place.

VERSION NOTE

Use Python 3.10+ for all syntax on this page.

→ match/case and int | str require 3.10+.

TRY IT

Add a "help" case that prints "Choose start or stop".

15 / 20 — COMPACT PYTHON PATTERNS

LIST COMPREHENSION

```
nums = [1, 2, 3, 4]
squares = [n * n for n in nums]
print(squares)
```

Output: [1, 4, 9, 16]

```
evens = [n for n in nums if n % 2 == 0]
```

Output of print(evens): [2, 4]

each item → optional filter → expression → new list

LAMBDA: ONE EXPRESSION

```
double = lambda n: n * 2
```

double(5) → 10

- Use def for longer or clearer functions.

MAP: TRANSFORM ITEMS

```
list(map(lambda n: n * 2, nums))
```

Result: [2, 4, 6, 8]

- map returns an iterator; list collects its results.

FILTER: KEEP MATCHES

```
list(filter(lambda n: n > 2, nums))
```

Result: [3, 4]

- filter also returns an iterator.

REDUCE: COMBINE ITEMS

```
from functools import reduce
total = reduce(lambda a, b: a + b, nums)
print(total)
```

Output: 10

1 + 2 → 3 → 3 + 3 → 6 → 6 + 4 → 10

- For addition, sum(nums) is simpler.

TRY IT

Build [3, 6, 9] with a comprehension.

Answer: [n * 3 for n in range(1, 4)]

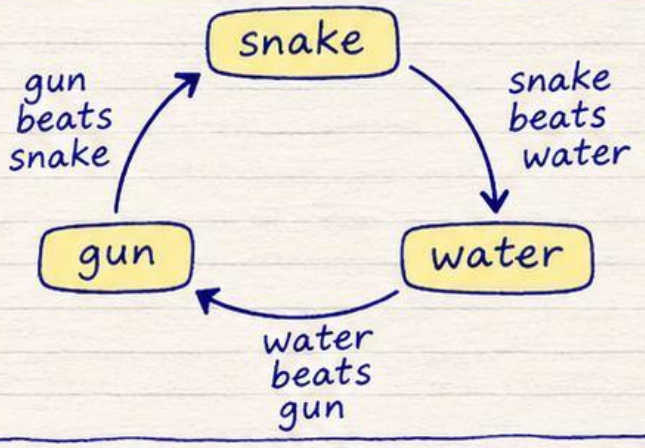
- Prefer readable code over clever code.

16 / 20 — PROJECT: THREE-CHOICE GAME

THE RULES

- Snake beats water.
- Water beats gun.
- Gun beats snake.
- Same choice means a draw.

Skills: input, dictionary, conditions and random.



ONE COMPLETE ROUND

```
import random
wins = {"snake": "water",
        "water": "gun",
        "gun": "snake"}
user = input("snake / water / gun: ").lower()
if user not in wins:
    print("Invalid choice")
else:
    cpu = random.choice(list(wins))
    print("Computer:", cpu)
    if user == cpu:
        print("Draw")
    elif wins[user] == cpu:
        print("You win")
    else:
        print("Computer wins")
```

SAMPLE ROUND

User enters: snake

Computer: water

You win

- The computer choice changes randomly.

HOW IT WORKS

- wins[user] is the choice your move defeats.
- Check input before using it as a dictionary key.
- random.choice selects one item from the choices.

LEVEL UP

- Add a loop for three rounds.
- Track wins, losses and draws.
- Let the player type quit to exit.

CHECK YOUR LOGIC

- snake vs gun → Computer wins
- water vs water → Draw

17 / 20 — PROJECT: GUESS THE NUMBER

THE CHALLENGE

- Guess a secret integer from 1 to 20.
- Receive Higher or Lower hints.
- Count valid guesses until the answer matches.

PLAYABLE VERSION

```
import random
secret = random.randint(1, 20)
attempts = 0
while True:
    try:
        guess = int(input("Guess 1-20: "))
    except ValueError:
        print("Enter a whole number")
        continue
    if not 1 <= guess <= 20:
        print("Stay within 1-20")
        continue
    attempts += 1
    if guess == secret:
        print("Correct! Attempts:", attempts)
        break
    print("Higher" if guess < secret else "Lower")
```

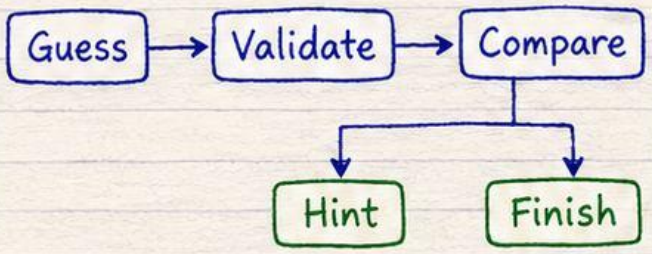
SAMPLE ROUND

Suppose secret = 12.

- Guess 8 → Higher
- Guess 16 → Lower
- Guess 12 → Correct! Attempts: 3

WHY THESE LINES MATTER

- `continue` skips invalid input.
- `break` stops the loop after a correct guess.
- The counter increases only for valid guesses.



LEVEL UP

- Limit the game to five valid guesses.
- Save the best score to a file.
- Offer another round after the game ends.

18 / 20 — BUILD A VOICE ASSISTANT

PROJECT ROADMAP

- Build a small ZainSpire assistant in stages.
- Start with typed commands, then add voice.
- Goal: hear a command, choose an action, reply.

TEST BEFORE EXPANDING

- Try silence, unknown commands and noisy audio.
- Handle timeouts and microphone failures.
- Keep a typed fallback and an exit command.

THE PIPELINE

Microphone → Speech to text → Command router → Action → Spoken reply

START WITH A TYPED PROTOTYPE

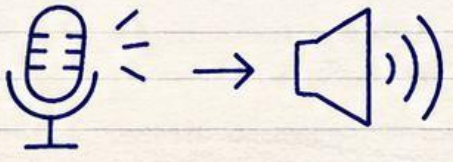
```
from datetime import datetime

command = input("Command: ").lower().strip()
if command == "time":
    print(datetime.now().strftime("%H:%M"))
elif command == "hello":
    print("Salam, Zain!")
else:
    print("Try time or hello")
```

For input **hello**:
Output: Salam, Zain!

ADD ONE PART AT A TIME

1. Capture microphone input.
2. Convert speech to text.
3. Match a few clear commands.
4. Run the selected action.
5. Convert the reply to speech.



TOOLS TO EXPLORE

- datetime: date and time
- webbrowser: open a website
- SpeechRecognition: speech recognition wrapper
- pyttsx3: local text-to-speech
- Audio setup and service needs vary by platform.

NEXT STEP

- Add a music command, then a news lookup.
- This page is a build roadmap, not a full voice app.

19 / 20 — BUILD A REPLY ASSISTANT

PROJECT ROADMAP

- Turn an incoming message into a useful draft.
- Start with simple rules; add an AI model later.
- Keep a review step before sending.

THE PIPELINE



BUILD THE FIRST DRAFT FUNCTION

```
def draft_reply(message):
    text = message.lower()
    if "python" in text:
        return "Which Python topic, Zain?"
    return "Please share a little more detail."

print(draft_reply("I need Python help"))
```

Output: Which Python topic, Zain?

GROW IT IN STAGES

1. Accept a typed message.
2. Add a few topic-based rules.
3. Keep a short conversation history.
4. Connect a model API if needed.
5. Show the draft in a simple interface.

WHAT EACH PART DOES

- Input layer: receives the message.
- Context layer: keeps relevant recent messages.
- Draft layer: produces a possible response.
- Review layer: lets the user edit or approve it.
- Send layer: sends the approved text.

PRACTICAL CHECKS

- Store API keys outside the code.
- Handle connection errors and empty replies.
- Avoid duplicate sends.
- Use only messages you are allowed to process.
- Some services have usage costs.

TRY IT

- Add a branch for messages containing "thanks".

⚠ This page is a build roadmap, not a full chat app.

20 / 20 — REVISION & PRACTICE

PICK THE RIGHT COLLECTION

Type	Meaning	Example
list	ordered, changeable	[1, 2]
tuple	ordered, fixed	(1, 2)
dict	key-value mapping	{"x": 1}
set	unique values	{1, 2}

- Dictionary order follows insertion.
- Sets have no positional indexing.

REMEMBER THESE

- = assigns; == compares.
- input() returns a string.
- Indices start at 0; slice stop is excluded.
- Indent blocks consistently.
- return gives back a value; print displays it.
- Use with for files.
- Catch errors you can handle.
- Test small pieces before combining them.

PREDICT THE RESULT

1. `7 // 2`
2. `"Zain"[-1]`
3. `{2, 2, 5}`
4. `[n * 2 for n in range(3)]`
5. `bool("")`
6. `{"x": 1} | {"x": 9}`

→

CHECK YOUR ANSWERS

1. 3
2. "n"
3. {2, 5}
4. [0, 2, 4]
5. False
6. {"x": 9}

BUILD WITHOUT LOOKING

- An even/odd checker.
- A score tracker with a dictionary.
- A file-based notes saver.
- A guessing game with input validation.

YOUR NEXT STEP

- Pick one project. Build it. Break it. Fix it.
- Read more: docs.python.org/3/tutorial/
- 20-page revision guide complete! ✓