```
print ('hello world') # Generate "hello world".
print("hello world") # Generate "hello world". The output is the same when single and double quotation marks are carried in input
```

```
hello world
hello world
```

```
#string using double qoutes
print("Hello Huawei")

#multiple line string
print( '''this is the long string
     this is the second
     the thrid line''')

#when you need to write the string in multiple lines
print("hello " \
      "world"
      )
```

```
Hello Huawei
this is the long string
     this is the second
     the thrid line
hello world
```

Comments

```
# This is a comment
height = 442 # Meters
```

```
# All of this code is ignored
'''
for i in range(10):
print("Hello", i)
...
'''
```

```
'\nfor i in range(10):\nprint("Hello", i)\n...\n'
```

Variables

```
"""A variable is a name for some value
Variable names usually follow same rules as C
[A-Za-z_][A-Za-z0-9_]
You do not declare types (int, float, etc.)
"""
height = 442 # An integer
height = 442.0 # Floating point
height = 'Really tall' # A string
```

```
π = 3.14159
㭎 = 2*π
```

Declaring

```
# declaring the var
Number = 100

# display
print("Before declare: ", Number)

# re-declare the var
Number = 120.3

print("After re-declare:", Number)
```

```
Before declare:  100
After re-declare: 120.3
```

```
a = b = c = 10

print(a)
print(b)
print(c)
```

```
10
10
10
```

```
a, b, c = 1, 20.2, "GeeksforGeeks"

print(a)
print(b)
print(c)
```

```
1
20.2
GeeksforGeeks
```

```
a = 10
a = "GeeksforGeeks"

print(a)
```

```
GeeksforGeeks
```

```
a = 10
b = 20
print(a+b)

a = "Geeksfor"
b = "Geeks"
print(a+b)
```

```
30
GeeksforGeeks
```

```
a = 10
b = "Geeks"
print(a+b)
```

```
---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
Cell In[13], line 3
      1 a = 10
      2 b = "Geeks"
----> 3 print(a+b)

TypeError: unsupported operand type(s) for +: 'int' and 'str'
```

Naming Conventions

```
#Use "snake case" for multiple words
first_name = "Guido"
last_name = "van Rossum"
```

```
#Use leading _ for "private" names
_name = "khaled"
```

Input

```
a = input("Enter a number: ")
print("You entered:", a)
```

```
You entered: 5
```

Exercises

Python Exercise: Printing, Comments, and Variables Instructions

Create variables to store your name, age, and favorite color.

Use comments to explain each step of your code.

Use the print() function to display these details in a full sentence.

Add another print statement to display each variable on a separate line.

Run your code and make sure the output is correct.

```
Start coding or generate with AI.
```

Changing Variable Values

Create a variable called temperature and set it to 30.

Print "The temperature is 30°C today."

Change the value to 35.

Print "The temperature is now 35°C."

```
Start coding or generate with AI.
```

Create variables:

apples = 5

bananas = 8

Print "I have 5 apples and 8 bananas."

Print the total number of fruits using a variable total_fruits.

```
Start coding or generate with AI.
```

Ask the user to enter their name and age using input().

Store them in variables.

Print: "Hello, [name]! You are [age] years old."

Add comments explaining each step

```
Start coding or generate with AI.
```

Operators

```
print(True+False)# The output is 1. By default, True indicates 1, and False indicates 0.
print(True or False)# If True is displayed, enter or or perform the OR operation.
print(5//2)# The output is 2, and // is the rounding operator.
int(5)/int(2) #no casting for the operation
print(5/2)# The output is 2.5, and // is the rounding operator.
print(5%2)# The output is 1, and % is the modulo operator.
print(3**2) # The output is 9, and ** indicates the power operation.
print(5+1.6) # The output is 6.6. By default, the sum of numbers of different precisions is the number of the highest precision type.
```

```
1
True
2
2.5
1
9
6.6
```

```
# Addition (+)
print(5 + 3)   # 8

# Subtraction (-)
print(10 - 4)  # 6
```

```
# Multiplication (*)
print(7 * 6)   # 42

# Division (/)
print(8 / 2)   # 4.0  (always returns float in Python 3)

# Modulus (%) → remainder
print(10 % 3)  # 1
```

```
8
6
42
4.0
1
```

```
# Left Shift (<<) → shifts bits to the left, filling with zeros
print(5 << 1)  # 10   (binary: 101 → 1010)

# Right Shift (>>) → shifts bits to the right, dropping bits
print(20 >> 2) # 5    (binary: 10100 → 101)
```

```
10
5
```

```
# AND (&) → bit-by-bit AND
print(5 & 3)   # 1   (binary: 101 & 011 → 001)

# OR (|) → bit-by-bit OR
print(5 | 3)   # 7   (binary: 101 | 011 → 111)

# XOR (^) → bit-by-bit exclusive OR
print(5 ^ 3)   # 6   (binary: 101 ^ 011 → 110)

# Bitwise NOT (~) → flips all bits
print(~5)      # -6  (in two's complement form)
```

```
1
7
6
-6
```

Expressions

```
"""
An expression → something that returns a value.

A statement → something that does an action (may or may not return a value).

"""

x = 5 + 3   # "5 + 3" is an expression → evaluates to 8
print(x)    # print is a statement → action: display the value
```

```
8
```

```
#Relational / Comparison Expressions
#Return True or False:

print(5 > 3)   # True
print(10 == 5) # False
print(7 != 2)  # True
```

```
True
False
True
```

```
"""Logical Expressions
Combine boolean values:
"""

x = True
y = False
print(x and y)  # False
print(x or y)   # True
print(not x)    # False
```

```
False
True
False
```

```
# Bitwise Expressions
print(5 & 3)  # 1  (101 & 011 = 001)
print(5 | 3)  # 7  (101 | 011 = 111)
```

```
1
7
```

Formatted Printing

```
# f-strings
name = "Khaled"
age = 25
print(f"My name is {name} and I am {age} years old.")
```

```
My name is Khaled and I am 25 years old.
```

```
x = 5
y = 3
print(f"{x} + {y} = {x + y}")
```

```
5 + 3 = 8
```

```
pi = 3.14159265
print(f"Pi to 2 decimal places: {pi:.2f}")
```

```
Pi to 2 decimal places: 3.14
```

```
salary = 1500000
print(f"My salary is {salary:,}")
```

```
My salary is 1,500,000
```

```
score = 0.875
print(f"Success rate: {score:.2%}")
```

```
Success rate: 87.50%
```

```
# .format() method
print("My name is {} and I am {} years old.".format(name, age))
```

```
My name is Khaled and I am 25 years old.
```

```
print("{0} is learning {1}".format("Khaled", "Python"))
print("{1} is being learned by {0}".format("Khaled", "Python"))
```

```
Khaled is learning Python
Python is being learned by Khaled
```

```
print("My name is {name} and I am from {country}".format(name="Ali", country="Egypt"))
```

```
My name is Ali and I am from Egypt
```

```
pi = 3.14159265
print("Pi to 2 decimal places: {:.2f}".format(pi))
```

```
Pi to 2 decimal places: 3.14
```

```
salary = 1500000
print("My salary is {:,}".format(salary))
```

```
My salary is 1,500,000
```

Placeholder	Meaning	Example Value
`%s`	String	`"Python"`
`%d`	Integer (decimal)	`42`
`%f`	Floating-point number	`3.14`
`%.nf`	Float with `n` decimals	`%.2f` → `3.14`
`%x`	Hexadecimal (lowercase)	`255` → `ff`
`%o`	Octal	`8` → `10`

```
name = "Khaled"
age = 25
print("My name is %s and I am %d years old." % (name, age))
```

```
My name is Khaled and I am 25 years old.
```

```
pi = 3.14159265
print("Pi to 2 decimal places: %.2f" % pi)
```

```
Pi to 2 decimal places: 3.14
```

```
item = "apple"
price = 2.5
print("The %s costs $%.2f" % (item, price))
```

```
The apple costs $2.50
```

```
number = 42
print("Binary: %b is not supported directly in %% formatting")
print("Hex: %x" % number)  # Hexadecimal
print("Octal: %o" % number) # Octal
```

```
Binary: %b is not supported directly in %% formatting
Hex: 2a
Octal: 52
```

```
name = "Sara"
score = 95.678
print("%s scored %.1f%% in the exam." % (name, score))

"""Explanation:

%s → Placeholder for a string → "Sara"

%.1f → Floating-point number with 1 decimal place → 95.7

%% → Escapes the % symbol → prints % literally."""
```

```
Sara scored 95.7% in the exam.
```

Exercises

Create two variables:

x = 10

y = 3

Use Python operators to calculate:

Addition

Subtraction

Multiplication

Division

Floor division

Modulus (remainder)

Exponentiation (x to the power of y)

Print each result in a descriptive way. (Example: print("x + y =", x + y))

```
Start coding or generate with AI.
```

Create two variables:

age1 = 18
age2 = 21

Compare them using:

Greater than

Less than

Equal to

Not equal to

Greater than or equal to

Less than or equal to

Print the results. (Example: print("age1 > age2:", age1 > age2))

```
Start coding or generate with AI.
```

Create two Boolean variables:

is_student = True
has_id_card = False

Use logical operators to check:

and → Is the person a student and has an ID card?

or → Is the person a student or has an ID card?

not → Is the person not a student?

Print the results.

```
Start coding or generate with AI.
```

Create two variables:

a = 6 # binary: 110
b = 3 # binary: 011

Perform and print results of:

a & b (Bitwise AND)

a | b (Bitwise OR)

a ^ b (Bitwise XOR)

```
Start coding or generate with AI.
```

Take a number n = 5 (binary: 101).

Perform:

Left shift by 1 (n << 1)

Right shift by 1 (n >> 1)

Print results and explain what happened in comments.

```
Start coding or generate with AI.
```

Let:

x = 12 # binary: 1100 y = 8 # binary: 1000

Check:

If (x & y) > 0 → print "x and y share common bits"

Else → print "No common bits"

```
Start coding or generate with AI.
```

Double-click (or enter) to edit

Conditionals

1 The if Statement

```
x = 10
if x > 5:
    print("x is greater than 5")
```

2 if...else Statemen

```
x = 3
if x > 5:
    print("x is greater than 5")
else:
    print("x is not greater than 5")
```

3 if...elif...else Statement

```
x = 7
if x > 10:
    print("x is greater than 10")
elif x > 5:
    print("x is greater than 5 but not greater than 10")
else:
    print("x is 5 or less")
```

4 Nested Conditionals

```
x = 12
if x > 5:
    if x % 2 == 0:
        print("x is greater than 5 and even")
```

5 Conditional Expressions (Ternary Operator)

Single-line if...else for quick assignments.

```
x = 8
result = "Even" if x % 2 == 0 else "Odd"
print(result)  # Even
```

6 Truthy and Falsy in Conditionals

```
name = ""
if name:
    print("Name exists")
```

```
else:
    print("No name")  # runs because "" is falsy
```

Conditionals work closely with comparison operators (>, <, ==, !=, >=, <=) and logical operators (and, or, not) to form complex decisions.

```
age = 25
income = 40000
has_degree = True

if (age >= 21 and income > 30000) or not has_degree:
    print("Eligible for the program")
else:
    print("Not eligible")
```

```
a = 10
b = 12
c = 0
if a or b or c:
    print("Atleast one number has boolean value as True")
else:
    print("All the numbers have boolean value as False")
```

```
a = 10
b = -10
c = 0
if a > 0 or b > 0:
    print("Either of the number is greater than 0")
else:
    print("No number is greater than 0")
if b > 0 or c > 0:
    print("Either of the number is greater than 0")
else:
    print("No number is greater than 0")
```

```
if name in namelist:
    # Not implemented yet (or nothing)
    pass
else:
    statements
```

Exercises

Challenge: Movie Ticket Price Calculator

Scenario: A cinema charges ticket prices based on **age** and **membership** status:

- **Age Rules:**
  - Children (under 12) → 5 USD
  - Teens (12–17) → 7 USD
  - Adults (18–59) → 10 USD
  - Seniors (60+) → 6 USD
- **Membership Discount:**
  - Members get **2 USD** off the price.

Your Task

1. Ask the user to enter their **age**.
2. Ask if they are a cinema **member** (`yes` or `no`).
3. Use `if`, `elif`, and `else` to determine the ticket price.
4. If they are a member, subtract 2 from the price.
5. Print the final ticket price.

Example Output

```
Enter your age: 15
Are you a member? yes
Your ticket price is: $5
```

```
Start coding or generate with AI.
```

Number Sign & Even/Odd Checker *(Easy → Medium)*

Task:

1. Ask the user to enter a number.
2. Use `if` conditions to check:
   - If the number is **positive**, **negative**, or **zero**.
   - If it is **even** or **odd**.
3. Print the results.

Example Output:

```
Enter a number: -7
The number is negative and odd.
```

---

2. Student Grade Evaluator *(Medium)*

Rules:

- 90+ → A
- 80–89 → B
- 70–79 → C
- 60–69 → D
- Below 60 → F

Task:

1. Ask the user for their exam score (0–100).
2. Use `if`, `elif`, and `else` to determine the grade.
3. Print the grade.

Example Output:

```
Enter your score: 85
Your grade is: B
```

```
Start coding or generate with AI.
```

ATM Withdrawal Simulation *(Medium → Hard)*

Scenario:

- The ATM only gives cash in multiples of **50**.
- Maximum withdrawal per transaction is **500**.
- The balance is **1000**.

Task:

1. Ask the user for the withdrawal amount.
2. Use `if` statements to check:
   - If the amount is greater than balance → print `"Insufficient balance"`
   - If the amount is not a multiple of 50 → print `"Invalid amount. Must be multiple of 50"`
   - If the amount is greater than 500 → print `"Limit exceeded"`
   - Otherwise → deduct from balance and print `"Withdrawal successful. Remaining balance: ..."`

Example Output:

```
Enter withdrawal amount: 250
Withdrawal successful. Remaining balance: 750
```

Strings

```
# """
# A string is a sequence of Unicode characters.

# Enclosed in:

# Single quotes → 'Hello'

# Double quotes → "Hello"

# Triple quotes → '''Hello''' or """Hello""" (used for multi-line strings or docstrings).
# """
```

```
s1 = 'hello'
s2 = "world"
s3 = """This is
a multi-line string."""
```

```
# 2. Characteristics

# Ordered: Characters have a fixed order.

# Immutable: Cannot be changed after creation.

# Iterable: You can loop through them.

# Indexed: Characters accessed by position.
```

```
s = "Python"
print(s[0])     # P
print(s[-1])    # n
print(s[2:5])   # tho
print(s[:3])    # Pyt
print(s[3:])    # hon
print(s[::-1])  # nohtyP (reverse string)
```

Operation	Example	Output
Concatenation	`"Hello" + "World"`	`"HelloWorld"`
Repetition	`"Hi" * 3`	`"HiHiHi"`
Membership	`"a" in "cat"`	`True`
Comparison	`"abc" == "ABC"`	`False`
Iteration	`[ch for ch in "dog"]`	`['d','o','g']`

```
"Hello" + "World"
```

```
"Hi" * 3
```

```
"a" in "cat"
```

```
"abc" == "ABC"
```

```
[ch for ch in "dog"]
```

```
Start coding or generate with AI.
```

Escape Code	Meaning	Example Output
`\'`	Single Quote	`'It\'s OK'` → It's OK
`\"`	Double Quote	`"She said \"Yes\""`
`\\`	Backslash	`\\` → \

Escape Code	Meaning	Example Output
`\n`	New Line	`"Hello\nWorld"`
`\t`	Tab	`"Hello\tWorld"`
`\r`	Carriage return	`"Hello\rWorld"` → `World`
`\b`	Backspace	`"Hello\bWorld"` → `HellWorld`

```
s = 'It\'s OK'
print(s)  # Output: It's OK (escaped single quote)
```

```
"It's OK"
```

```
s = "She said \"Yes\""
print(s)
```

```
'She said "Yes"'
```

```
s = "Hello\nWorld"
print(s)
```

```
'Hello\nWorld'
```

```
s = "Hello\rWorld"
print(s)  # Output: "World" (carriage return moves cursor back to start of line)
```

```
World
```

```
s = "Hello\tWorld"
print(s)
```

```
s = "Hello\rWorld"
print(s)
```

```
s = "Hello\bWorld"
print(s)
```

Methods

```
s = "hello world"
print(s.upper())   # HELLO WORLD
print(s.lower())   # hello world
print(s.title())   # Hello World
print(s.capitalize())  # Hello world
print(s.swapcase())    # HELLO WORLD → hello world, vice versa
```

```
HELLO WORLD
hello world
Hello World
Hello world
HELLO WORLD
```

Searching

```
s = "Python programming"
print(s.find("pro"))     # 7 (first occurrence)
print(s.rfind("o"))      # 9 (last occurrence)
print(s.index("Py"))     # 0
print(s.startswith("Py")) # True
print(s.endswith("ing"))  # True
```

```
7
9
0
True
True
```

Modification

```
s = "   hello world   "
print(s.strip())   # "hello world"
print(s.lstrip())  # "hello world   "
print(s.rstrip())  # "   hello world"
print(s.replace("world","Python"))  # hello Python
```

Splitting & Joining

```
s = "apple,banana,cherry"
print(s.split(","))   # ['apple', 'banana', 'cherry']
print(" ".join(["I","love","Python"]))  # I love Python
```

Validation Methods (Boolean)

```
s = "Python3"
print(s.isalpha())   # False (contains number)
print("123".isdigit()) # True
print("abc123".isalnum()) # True
print("hello".islower())  # True
print("HELLO".isupper())  # True
print("  ".isspace())     # True
print("Python".istitle()) # True
```

```
False
True
True
True
True
True
True
```

Raw Strings

```
path = r"C:\Users\Khaled\Documents"
print(path)  # C:\Users\Khaled\Documents
```

```
C:\Users\Khaled\Documents
```

Useful Tricks

```
# Reverse a string
s = "Python"
print(s[::-1])  # nohtyP

# Count characters
print(len(s))   # 6

# Count frequency of letters
from collections import Counter
print(Counter("banana"))  # {'a':3, 'b':1, 'n':2}

# Palindrome check
word = "madam"
print(word == word[::-1])  # True
```

🔷 Strings, Bytes, Encoding & Decoding

1. Strings in Python (`str`)

- A **string** (`str`) in Python is a sequence of **Unicode characters**.
- Unicode is a universal standard that assigns a unique number (code point) to every character in every language (`'A' = U+0041`, `'ب' = U+0628`, etc.).

Example:

```
s = "Python"
print(type(s))   # <class 'str'>
```

```
<class 'str'>
```

2. Encoding (`str → bytes`)

When you **encode** a string, you convert the **Unicode characters** into a sequence of **bytes** using a specific encoding (like UTF-8).

- **UTF-8** is the most common encoding because it supports all Unicode characters.

```
s = "Python"
encoded = s.encode("utf-8")
print(encoded)       # b'Python'
print(type(encoded)) # <class 'bytes'>
#Notice the prefix `b'...'` → it means the object is of type **bytes**, not string.
```

3. Decoding (`bytes → str`)

When you **decode**, you take those **bytes** and interpret them back into a **string** using the same encoding.

```
decoded = encoded.decode("utf-8")
print(decoded)        # Python
print(type(decoded))  # <class 'str'>
#If you decode with the wrong encoding, you may get errors or strange characters.
```

4. Why is this important?

- **When reading/writing files:**

```
with open("file.txt", "w", encoding="utf-8") as f:
    f.write("مرحبا بالعالم")  # Arabic text
#  If you don’t specify the encoding, Python might default to the system encoding, which can cause problems with non-English text.
```

- **When working with networks/APIs:** Data sent over the internet is in **bytes**, so you must **encode** before sending and **decode** after receiving.

```
Start coding or generate with AI.
```

5. Example with Non-English Text

```
s = "مرحبا"
encoded = s.encode("utf-8")
print(encoded)         # b'\xd9\x85\xd8\xb1\xd8\xad\xd8\xa8\xd8\xa7'
```

```
b'\xd9\x85\xd8\xb1\xd8\xad\xd8\xa8\xd8\xa7'
```

```
decoded = encoded.decode("utf-8")
print(decoded)         # مرحبا
```

```
مرحبا
```

examples of encoding in Python

```
## 🔹 Example 1: Basic UTF-8 Encoding
s = "Python"
encoded = s.encode("utf-8")
print(encoded)
```

```
## 🔹 Example 2: Encoding Non-English Text
## 👉 Each Arabic character is represented with **2 bytes** in UTF-8.
s = "مرحبا"   # Arabic word for "Hello"
encoded = s.encode("utf-8")
print(encoded)
```

```
## 🔷 Example 3: Encoding in UTF-16

s = "Hello"
encoded = s.encode("utf-16")
print(encoded)
```

🔷 Example 4: Encoding with ASCII

```
s = "Python"
encoded = s.encode("ascii")
print(encoded)   # b'Python'
```

But if you try:

```
s = "مرحبا"
encoded = s.encode("ascii")  # ❌ Error
```

You'll get:

```
UnicodeEncodeError: 'ascii' codec can't encode characters in position 0-4
```

👉 Because ASCII only supports **0–127 characters** (English letters, digits, symbols).

🔷 Example 5: Safe Encoding with Errors Handling

```
s = "مرحبا Python"
encoded = s.encode("ascii", errors="ignore")
print(encoded)   # b' Python'

encoded = s.encode("ascii", errors="replace")
print(encoded)   # b'????? Python'
```

👉 `"ignore"` removes unsupported characters, `"replace"` substitutes them with `?`.

✅ **Key Takeaway:**

- Use `.encode("utf-8")` for most real-world applications (web, files, APIs).
- Other encodings (`ascii`, `utf-16`, `latin-1`) exist but are less universal.

```
Start coding or generate with AI.
```

📝 String Exercises in Python

🔷 Exercise 1: Basic String Operations

1. Create a string variable called `name` with your name.
2. Print:
    - The first character.
    - The last character.
    - The length of the string.

👉 Example:

```
name = "Python"
# Output: P, n, 6
```

🔷 Exercise 2: String Slicing

Given:

```
text = "Artificial Intelligence"
```

1. Print the substring `"Artificial"`.
2. Print the substring `"Intelligence"`.
3. Print only `"Intel"`.
4. Print the string in reverse.

🔷 Exercise 3: String Methods

Take the string:

```
quote = "python programming is fun"
```

1. Convert it to **uppercase**.
2. Convert it to **title case**.
3. Count how many times `"n"` appears.
4. Replace `"fun"` with `"awesome"`.

🔷 Exercise 4: String Search

Given:

```
sentence = "I love learning Python programming"
```

1. Check if the word `"Python"` exists in the sentence.
2. Find the index of `"learning"`.
3. Check if the sentence starts with `"I love"`.
4. Check if the sentence ends with `"programming"`.

🔷 Exercise 5: Palindrome Checker (Challenge)

A **palindrome** is a word that reads the same backward and forward (e.g., `"madam"`, `"racecar"`). 👉 Write a program that asks the user to enter a word and checks if it is a palindrome.

🔷 Exercise 6: String Encoding

1. Take the string `"مرحبا"` and encode it into UTF-8.
2. Decode it back to normal text.
3. Try encoding it in ASCII with error handling (`ignore` or `replace`).

```
Start coding or generate with AI.
```

LIST

1. What is a List?

A list in Python is an ordered collection of items that can be of any data type (numbers, strings, booleans, even other lists).
Lists are mutable, meaning you can change them after creation.

```
# Empty list
my_list = []

# List with numbers
numbers = [1, 2, 3, 4, 5]

# List with mixed data types
```

```
mixed = [1, "apple", True, 3.14]

# List of lists (nested list)
nested = [[1, 2], [3, 4]]
```

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

print(fruits[0])   # First item: apple
print(fruits[1])   # Second item: banana
print(fruits[-1])  # Last item: cherry
```

```
fruits = ["apple", "banana", "cherry"]
fruits[1] = "blueberry"
print(fruits)  # ['apple', 'blueberry', 'cherry']
```

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

# Append (add to the end)
fruits.append("cherry")

# Insert at a specific position
fruits.insert(1, "mango")

print(fruits)  # ['apple', 'mango', 'banana', 'cherry']
```

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

# Remove by value
fruits.remove("banana")

# Remove by index
del fruits[0]

# Pop (remove and return item)
last_fruit = fruits.pop()

print(last_fruit)  # cherry
print(fruits)      # []
```

```
numbers = [0, 1, 2, 3, 4, 5]

print(numbers[1:4])   # [1, 2, 3]
print(numbers[:3])    # [0, 1, 2]
print(numbers[3:])    # [3, 4, 5]
print(numbers[::2])   # [0, 2, 4] (every second element)
```

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

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

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

print("apple" in fruits)   # True
print("mango" not in fruits)  # True
```

11. List Comprehensions

A quick way to create lists in one line.

```
squares = [x**2 for x in range(5)]
print(squares)  # [0, 1, 4, 9, 16]
```

```
evens = [x for x in range(10) if x % 2 == 0]
print(evens)  # [0, 2, 4, 6, 8]
```

```
list1 = [1, 2, 3]
list2 = list1.copy()

list2.append(4)

print(list1)  # [1, 2, 3]
print(list2)  # [1, 2, 3, 4]
# (Avoid list2 = list1 because it links both lists to the same memory reference.)
```

```
[1, 2, 3]
[1, 2, 3, 4]
```

```
matrix = [
    [1, 2, 3],
    [4, 5, 6]
]

print(matrix[1][2])  # 6
```

Methods and Functions

```
fruits = ["apple", "banana", "cherry"]
len(fruits)
```

```
max([4, 7, 1])
```

```
min([4, 7, 1])
```

```
sum([4, 7, 1])
```

```
sorted([3,1,2])
```

```
sorted([3,1,2])
```

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

fruits.append("cherry")     # add at end
fruits.insert(1, "orange")  # insert at index 1
fruits.extend(["mango", "grape"])  # add multiple
print(fruits)
```

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

fruits.remove("banana")  # remove by value
last_item = fruits.pop() # remove last (or by index)
fruits.clear()           # remove all
```

```
numbers = [1, 2, 3, 2, 4, 2]

print(numbers.index(3))   # 2 → first occurrence index
print(numbers.count(2))   # 3 → number of times 2 appears
```

```
nums = [4, 1, 7, 3]

nums.sort()          # in-place sort
nums.sort(reverse=True) # descending
nums.reverse()       # reverse order (not sort)

print(nums)
```

```
fruits = ["apple", "banana"]
copy_list = fruits.copy()   # shallow copy
```

List Exercise

📝 Exercises on Python List Functions & Methods

---

1. Basic Exercises

1. Create a list of 5 numbers and print:
   - The length of the list
   - The maximum value
   - The minimum value
   - The sum of all numbers
2. Create a list of fruits.
   - Add `"mango"` at the end.
   - Insert `"orange"` at index `1`.
   - Extend the list with `["kiwi", "grape"]`.
   - Print the updated list.
3. Start with this list:

```
colors = ["red", "blue", "green", "blue", "yellow", "blue"]
```

   - Remove `"green"`.
   - Remove the last item using `.pop()`.
   - Count how many times `"blue"` appears.

---

2. Intermediate Exercises

4. Create a list:

```
numbers = [5, 2, 9, 1, 5, 6]
```

   - Sort it in ascending order.
   - Sort it in descending order.
   - Reverse the list without sorting.
5. Copy a list of animals into another list and prove that modifying the new list does not affect the original one.
6. Create a list of 10 random numbers (you can write them manually).
   - Use list comprehension to create a new list containing only even numbers.
   - Use list comprehension to create a new list of their squares.

---

3. Challenge Exercises

7. Create a 2D list (matrix):

```
matrix = [[1,2,3], [4,5,6], [7,8,9]]
```

   - Print the element in the **second row, third column**.
   - Print all elements of the first column.
8. Given a list of student scores:

```
scores = [85, 42, 79, 95, 66, 58, 90]
```

   - Find the highest and lowest score.
   - Sort the list.
   - Find how many students passed (score ≥ 60).
9. Write a Python program that:
   - Asks the user to enter 5 numbers (store them in a list).
   - Prints the average of those numbers.
   - Prints all numbers greater than the average.

Set

🔷 What is a Set in Python?

A **set** is a **collection of unique, unordered, and mutable elements**.

- ✅ **Unique** → no duplicates allowed
- ✅ **Unordered** → elements have no index or position
- ✅ **Mutable** → you can add or remove elements
- ❌ **Not indexable** → you cannot access by position (`myset[0]` ❌)

🔷 Creating Sets

```
# Empty set
s1 = set()

# Set with values
s2 = {1, 2, 3, 4}

# Set automatically removes duplicates
s3 = {1, 2, 2, 3, 4, 4}
print(s3)  # {1, 2, 3, 4}
```

🔷 Set Functions & Methods

```
s = {1, 2, 3}
s.add(4)        # Add one element
s.update([5,6]) # Add multiple elements
s.remove(2)     # Remove element (error if not found)
s.discard(10)   # Remove safely (no error if not found)
s.pop()         # Removes and returns a random element
s.clear()       # Remove all elements
```

2. Set Operations (like math)

```
A = {1, 2, 3, 4}
B = {3, 4, 5, 6}

print(A | B)   # Union: {1, 2, 3, 4, 5, 6}
print(A & B)   # Intersection: {3, 4}
print(A - B)   # Difference: {1, 2}
print(A ^ B)   # Symmetric Difference: {1, 2, 5, 6}
```

3. Membership & Length

```
s = {10, 20, 30}
print(20 in s)     # True
print(40 not in s) # True
print(len(s))      # 3
```

🔷 Frozen Sets

```
fs = frozenset([1, 2, 3, 4])
print(fs)
# fs.add(5) ❌ (Error: can't modify frozenset)
```

```
fruits = {"apple", "banana", "cherry"}
more_fruits = {"cherry", "orange", "mango"}

# Find common fruits
common = fruits & more_fruits
print("Common:", common)

# Find unique fruits
```

```
unique = fruits ^ more_fruits
print("Unique:", unique)
```

```
Common: {'cherry'}
Unique: {'mango', 'banana', 'orange', 'apple'}
```

🔷 Advanced Set Operations

```
### 1. Subset, Superset, Disjoint

A = {1, 2, 3}
B = {1, 2, 3, 4, 5}
C = {6, 7}

print(A.issubset(B))   # True (A ⊆ B)
print(B.issuperset(A)) # True (B ⊇ A)
print(A.isdisjoint(C)) # True (no common elements)
```

```
True
True
True
```

```
### 2. Copying Sets
s1 = {1, 2, 3}
s2 = s1.copy()
print(s2)  # {1, 2, 3}
```

```
{1, 2, 3}
```

```
### 3. Set Comprehension

#Like list comprehension, but creates a set:


squares = {x*x for x in range(6)}
print(squares)  # {0, 1, 4, 9, 16, 25}
```

```
### 4. Removing Duplicates Using Sets

nums = [1, 2, 2, 3, 4, 4, 5]
unique_nums = list(set(nums))
print(unique_nums)  # [1, 2, 3, 4, 5]
```

```
### 5. Difference Between `remove()` and `discard()`


s = {1, 2, 3}
s.remove(2)   # Works fine
# s.remove(10) ❌ Error

s.discard(10) # No error if element not found
```

🔷 Real-World Use Cases of Sets

1. *Remove Duplicates from Data*

```
emails = ["a@gmail.com", "b@gmail.com", "a@gmail.com"]
unique_emails = set(emails)
print(unique_emails)  # {'a@gmail.com', 'b@gmail.com'}
```

2. *Find Common Students in Courses*

```
courseA = {"Ali", "Sara", "Omar"}
courseB = {"Sara", "Mona", "Ali"}

common = courseA & courseB
print("Students in both courses:", common)
```

3. *Check Unique Characters*

```
word = "programming"
print(len(set(word)))   # Unique letters count
```

🔷 Practice Exercises on Sets

Exercise 1: Remove Duplicates

Write a program that takes a list of numbers with duplicates and prints the unique numbers using a set.

Exercise 2: Common Elements

Given two sets of fruits:

```
A = {"apple", "banana", "mango"}
B = {"mango", "orange", "apple"}
```

Find:

- Fruits that are common in both sets
- Fruits that are only in A
- Fruits that are only in B

Exercise 3: Subset Check

Check if:

```
A = {1, 2, 3}
B = {1, 2, 3, 4, 5}
```

Is `A` a subset of `B`?

Exercise 4: Unique Characters

Write a function that takes a string and returns the number of **unique characters** in it using sets. Example: `"hello"` → 4 unique characters (`h`, `e`, `l`, `o`).

Exercise 5: Symmetric Difference

Two classes attended a workshop:

```
class1 = {"Ahmed", "Khaled", "Sara"}
class2 = {"Sara", "Omar", "Hana"}
```

Find students who attended **only one class but not both**.

Double-click (or enter) to edit

Dictionary

- **Keys must be immutable** → e.g., `str`, `int`, `tuple` (but not `list` or `dict`).
- **Values can be any type** → even lists, dicts, or objects.
- **Insertion order is preserved (Python 3.7+)**.

- Lookup (`d[key]`) is **O(1)** on average (very fast).

🧩 2. Advanced Dictionary Creation

```
# Empty dictionary
d1 = {}

# Using dict()
d2 = dict()

# With key-value pairs
d3 = {"name": "Alice", "age": 25, "city": "Cairo"}

# Keys must be unique → last one wins
d4 = {"a": 1, "a": 2}
print(d4)  # {"a": 2}
```

```
student = {"name": "Ali", "grade": "A", "age": 20}

print(student.keys())    # dict_keys(['name', 'grade', 'age'])
print(student.values())  # dict_values(['Ali', 'A', 20])
print(student.items())   # dict_items([('name', 'Ali'), ('grade', 'A'), ('age', 20)])
```

```
dict_keys(['name', 'grade', 'age'])
dict_values(['Ali', 'A', 20])
dict_items([('name', 'Ali'), ('grade', 'A'), ('age', 20)])
```

```
# From list of tuples
pairs = [("a", 1), ("b", 2)]
d1 = dict(pairs)  # {'a': 1, 'b': 2}

# From keyword arguments
d2 = dict(x=10, y=20)  # {'x': 10, 'y': 20}

# Using dictionary comprehension
d3 = {x: x**2 for x in range(5)}
# {0: 0, 1: 1, 2: 4, 3: 9, 4: 16}
```

🧩 3. Accessing with `.get()` and `setdefault()`

```
person = {"name": "Ali"}

# .get() → safe access
print(person.get("age", "Not available"))  # Not available

# .setdefault() → return value if key exists, else create it
print(person.setdefault("age", 20))   # 20 (creates age)
print(person)  # {'name': 'Ali', 'age': 20}
```

```
Not available
20
{'name': 'Ali', 'age': 20}
```

🧩 4. Updating & Merging Dictionaries

```
d1 = {"a": 1, "b": 2}
d2 = {"b": 3, "c": 4}

# update() overwrites
d1.update(d2)
print(d1)  # {'a': 1, 'b': 3, 'c': 4}

# New in Python 3.9+: merge operators
d3 = d1 | {"d": 5}   # Union → {'a': 1, 'b': 3, 'c': 4, 'd': 5}
d4 = d1 | d2         # Right dict wins → {'a': 1, 'b': 3, 'c': 4}
```

```
{'a': 1, 'b': 3, 'c': 4}
```

🧩 5. Deleting Keys Safely

```
d = {"x": 1, "y": 2}

# pop()
value = d.pop("x", "Not found")
print(value)  # 1
print(d)      # {'y': 2}

# popitem() → last inserted key-value
key, val = d.popitem()
print(key, val)  # y 2
```

🧩 6. Looping Tricks

```
grades = {"Ali": 90, "Sara": 85, "Omar": 78}

# Enumerate keys
for i, key in enumerate(grades):
    print(i, key)

# Loop sorted by keys
for k in sorted(grades):
    print(k, grades[k])

# Loop sorted by values
for k, v in sorted(grades.items(), key=lambda item: item[1], reverse=True):
    print(k, v)
```

```
0 Ali
1 Sara
2 Omar
Ali 90
Omar 78
Sara 85
Ali 90
Sara 85
Omar 78
```

🧩 7. Dictionary Views

Dictionary methods like `.keys()`, `.values()`, and `.items()` return **views**, not lists.

They are dynamic → reflect changes in the dictionary.

```
d = {"a": 1, "b": 2}
keys_view = d.keys()
print(keys_view)   # dict_keys(['a', 'b'])

d["c"] = 3
print(keys_view)   # dict_keys(['a', 'b', 'c']) (updated automatically)
```

🧩 8. Nested Dictionaries (Complex Data)

```
company = {
    "employee1": {"name": "Ali", "age": 25, "skills": ["Python", "ML"]},
    "employee2": {"name": "Sara", "age": 28, "skills": ["Java", "SQL"]}
}

print(company["employee1"]["skills"][0])  # Python
```

🧩 9. Dictionary Comprehension Examples

```
# Invert a dictionary
d = {"a": 1, "b": 2, "c": 3}
inverted = {v: k for k, v in d.items()}
print(inverted)  # {1: 'a', 2: 'b', 3: 'c'}

# Count characters
text = "banana"
count = {ch: text.count(ch) for ch in set(text)}
print(count)  # {'b': 1, 'n': 2, 'a': 3}
```

🧩 10. Special Dictionary Types (from `collections`)

Python's `collections` module has **powerful dictionary variations**:

```
from collections import defaultdict, OrderedDict, Counter

# defaultdict → auto-create keys
dd = defaultdict(int)
dd["x"] += 1
print(dd)  # defaultdict(<class 'int'>, {'x': 1})

# OrderedDict → remembers order (Python 3.7+ normal dict also does this)
od = OrderedDict(a=1, b=2)

# Counter → frequency counter
cnt = Counter("banana")
print(cnt)  # Counter({'a': 3, 'n': 2, 'b': 1})
```

```
defaultdict(<class 'int'>, {'x': 1})
Counter({'a': 3, 'n': 2, 'b': 1})
```

🧩 11. Dictionary Performance Notes

- Lookup time: **O(1)** average → fast.
- Insertion time: **O(1)** average.
- Worst case: **O(n)** if too many hash collisions (very rare).

🧩 12. Real-Life Example

```
phone_book = {
    "Ali": "01012345678",
    "Sara": "01198765432",
    "Omar": "01234567890"
}

# Search
name = "Sara"
print(f"{name}'s number is {phone_book.get(name, 'Not found')}")

# Add new contact
phone_book["Mona"] = "0151112233"

# Show all
for person, number in phone_book.items():
    print(person, "->", number)
```

🔷 Exercise 1: Countries & Capitals

Create a dictionary of 3 countries and their capitals.

- Print the capital of one country.
- Add a new country-capital pair.
- Update one capital.
- Delete one country.

🔷 Exercise 2: Word Counter

Write a program to count how many times each word appears in the sentence:

```
sentence = "python is fun and python is powerful"
```

👉 Expected Output:

```
{'python': 2, 'is': 2, 'fun': 1, 'and': 1, 'powerful': 1}
```

🔷 Exercise 3: Student Grades

Make a dictionary of students and their grades:

```
grades = {"Ali": 85, "Sara": 90, "Omar": 78}
```

- Print each student with their grade.
- Find the student with the highest grade.
- Calculate the average grade.

🔷 **Exercise 4: Shopping Cart**

Create a dictionary that stores items and their prices:

```
cart = {"apple": 3, "banana": 2, "milk": 5}
```

- Add a new item.
- Update the price of an item.
- Calculate the total cost of all items.

🔷 **Exercise 5: Nested Dictionary – School System**

Create a nested dictionary to represent students:

```
school = {
    "student1": {"name": "Ali", "age": 20, "grade": "A"},
    "student2": {"name": "Sara", "age": 22, "grade": "B"}
}
```

- Print the name of student1.
- Change student2's grade to "A+".
- Add a new student.

```
Start coding or generate with AI.
```

```
Start coding or generate with AI.
```

Tuples

✅ 1. Definition Recap

- A **tuple** is an ordered, immutable collection.
- Defined with `()` but can also be created without parentheses by separating values with commas.

```
t = 1, 2, 3     # Tuple without parentheses
print(t)        # (1, 2, 3)
```

```
(1, 2, 3)
```

✅ 2. Immutability (Important Detail)

- You cannot **add, remove, or change** tuple elements directly.
- But if the tuple contains **mutable objects** (like lists), their contents can be modified.
- ⚠️ The tuple reference is immutable, but the object inside can still change.

```
t = (1, [2, 3], 4)
t[1].append(99)
print(t)   # (1, [2, 3, 99], 4) ✅
```

```
(1, [2, 3, 99], 4)
```

✅ 3. Tuple vs List (Key Differences)

Feature	Tuple `( )`	List `[ ]`
Mutability	Immutable	Mutable
Performance	Faster	Slower
Memory usage	Lower	Higher
Methods	Only `count`, `index`	Many (append, pop, extend, etc.)
Hashable	Yes (if all elements are immutable)	No
Use Cases	Fixed data, dict keys	Dynamic data, frequent changes

✅ 4. Nested Tuples

Tuples can contain other tuples (multi-dimensional structure):

```
matrix = ((1, 2, 3),
          (4, 5, 6),
          (7, 8, 9))
print(matrix[1][2])  # 6
```

✅ 5. Tuple Unpacking (Advanced)

You can unpack tuples in flexible ways:

```
# Normal unpacking
name, age, country = ("Ali", 25, "Egypt")

# Using *
numbers = (1, 2, 3, 4, 5)
a, b, *rest = numbers
print(a, b, rest)  # 1 2 [3, 4, 5]

# Ignoring values
x, _, y = (10, 20, 30)
print(x, y)  # 10 30
```

✅ 6. Returning Multiple Values

A function can return multiple values naturally as a tuple:

```
def calc(a, b):
    return a+b, a-b, a*b

result = calc(5, 2)
print(result)       # (7, 3, 10)
sum_, diff, prod = result
print(sum_, prod)   # 7 10
```

✅ 7. Tuple as Dictionary Keys

- Because tuples are immutable (and hashable), they can be used as **keys in dictionaries**
- Lists cannot be used as dictionary keys since they are mutable.

```
coordinates = {
    (30.0444, 31.2357): "Cairo",
    (51.5074, -0.1278): "London"
}
print(coordinates[(30.0444, 31.2357)])  # Cairo
```

✅ 8. Built-in Functions with Tuples

Tuples support many Python built-ins:

```
t = (10, 20, 30, 40)

print(len(t))     # 4
print(max(t))     # 40
```

```
print(min(t))     # 10
print(sum(t))     # 100
print(sorted(t))  # [10, 20, 30, 40] -> returns list
```

✅ 9. Singleton Tuple (Tricky Case)

```
a = (5)     # ❌ Not a tuple, just an integer
b = (5,)    # ✅ Tuple
print(type(a))  # int
print(type(b))  # tuple
```

```
<class 'int'>
<class 'tuple'>
```

✅ 10. Memory & Performance

- Tuples are **faster** and **use less memory** than lists because they don't need to support item modification.

```
import sys

lst = [1, 2, 3, 4]
tpl = (1, 2, 3, 4)

print(sys.getsizeof(lst))  # 88 (example)
print(sys.getsizeof(tpl))  # 72 (smaller)
```

🔷 Advanced Exercises on Tuples

Exercise 1: Nested Tuples

Create a 3×3 tuple (matrix) and print the middle element.

Exercise 2: Tuple Unpacking

Given:

```
data = ("Ahmed", "Python", 2025, "Egypt")
```

Unpack only the first and last values into variables.

Exercise 3: Function Return

Write a function that takes a list of numbers and returns a tuple:

- Smallest number
- Largest number
- Total sum
- Average

Exercise 4: Tuple as Keys

Make a dictionary with tuples as keys `(x, y)` representing points, and values as distances from the origin. Example: `(3, 4)` → 5.0

Exercise 5: Memory Check

Compare memory usage of a list `(1,2,3,...,1000)` vs a tuple `(1,2,3,...,1000)`.

LOOPS

For Loop

```
for variable in sequence:
    # code to repeat
```

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

```
# Using range() in a loop

# range(start, stop, step) generates a sequence of numbers.

for i in range(1, 6):
    print(i)
```

```
for letter in "Python":
    print(letter)
```

While

```
while condition:
    # code to repeat
```

```
count = 1
while count <= 5:
    print(count)
    count += 1  # increase count to avoid infinite loop
```

```
password = ""
while password != "python123":
    password = input("Enter password: ")
print("Access granted!")
```

Break and Continue

```
for i in range(1, 10):
    if i == 5:
        break
    print(i)
```

```
for i in range(1, 6):
    if i == 3:
        continue
    print(i)
```

More

```
for i in range(3):
    print(i)
else:
    print("Loop finished!")
```

```
for i in range(1, 4):
    for j in range(1, 3):
        print(f"i = {i}, j = {j}")
```

```
total = 0
for i in range(1, 11):
    total += i
print("Sum:", total)
```

```
num = 5
for i in range(1, 11):
    print(f"{num} x {i} = {num * i}")
```

```
n = 5
while n > 0:
    print(n)
    n -= 1
print("Blast off!")
```

🔷 1. `enumerate()`

The **`enumerate()` function** adds a counter (index) to an iterable (like a list, tuple, or string).

📌 **Syntax:**

```
enumerate(iterable, start=0)
```

- `iterable` → list, tuple, string, etc.
- `start` → the starting index (default = 0).

📌 **Example:**

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

for index, fruit in enumerate(fruits, start=1):
    print(index, fruit)
```

✅ Use case: When you want both the **index and value** while looping.

```
Start coding or generate with AI.
```

```
Start coding or generate with AI.
```

🔷 2. `zip()`

The **`zip()` function** combines two (or more) iterables element by element.

📌 **Syntax:**

```
zip(iterable1, iterable2, ...)
```

📌 **Example:** ✅ Use case: Combine related data (like student names & grades).

```
names = ["Ali", "Sara", "Omar"]
grades = [85, 90, 78]

for name, grade in zip(names, grades):
    print(name, grade)
```

🔷 3. Similar Functions & Concepts

✅ `items()` (Dictionary Alternative to `enumerate`)

When looping through a dictionary, `items()` gives **key-value pairs**:

```
student = {"Ali": 85, "Sara": 90}

for name, grade in student.items():
    print(name, grade)
```

✅ `range(len(...))` (Old way instead of `enumerate`)

Without `enumerate`, people used `range()`:

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

for i in range(len(fruits)):
    print(i, fruits[i])
```

👉 But **`enumerate` is cleaner**.

```
Start coding or generate with AI.
```

✅ `itertools.zip_longest()` (Safer `zip`)

If iterables have different lengths, `zip` stops at the shortest. Use `zip_longest` to fill missing values:

```
from itertools import zip_longest

names = ["Ali", "Sara"]
grades = [85, 90, 78]

for name, grade in zip_longest(names, grades, fillvalue="N/A"):
    print(name, grade)
```

🔷 Comparison Table

Function	Purpose	Example Output
`enumerate()`	Adds index to iterable	`(0, "apple")`
`zip()`	Combines multiple iterables element by element	`("Ali", 85)`
`dict.items()`	Loops over dictionary key-value pairs	`("Ali", 85)`
`zip_longest()`	Like `zip`, but handles unequal length lists	`("Sara", 90), (N/A, 78)`

```
students = ["Ali", "Sara", "Omar"]
grades = [85, 90, 78]

for index, (name, grade) in enumerate(zip(students, grades), start=1):
    print(f"{index}. {name} got {grade}")
```

```
Start coding or generate with AI.
```

🔷 Exercises on Loops in Python

Exercise 1: Print Numbers

Write a loop to print numbers from **1 to 10**. 👉 Hint: use `range()`.

Exercise 2: Sum of Numbers

Write a loop that calculates the **sum of numbers from 1 to 100**. 👉 Expected Output: `5050`

Exercise 3: Multiplication Table

Ask the user for a number (e.g., `5`) and print its multiplication table up to `10`. 👉 Example for 5:

```
5 x 1 = 5
5 x 2 = 10
...
5 x 10 = 50
```

Exercise 4: Even Numbers

Write a loop that prints only the **even numbers** between `1` and `50`.

Exercise 5: Reverse String

Take a string `"Python"` and print it **in reverse order** using a loop. 👉 Output: `nohtyP`

Exercise 6: Factorial

Write a loop that finds the factorial of a number (e.g., `5! = 120`).

Exercise 7: Guessing Game

Use a `while` loop to create a simple guessing game:

- The program stores a secret number (say `7`).
- The user keeps entering guesses until they guess correctly.

Exercise 8: Nested Loop – Star Pattern

Print the following pattern with a nested loop:

```
*
**
***
****
*****
```

Exercise 9: List Iteration

Given a list:

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

Write a loop to print each fruit with its position (use `enumerate`).

Exercise 10: Break & Continue

Loop through numbers `1 to 10`:

- Skip (continue) if the number is `5`.
- Stop (break) if the number is `8`.

👉 Output should be: `1 2 3 4 6 7`

Double-click (or enter) to edit

```
Start coding or generate with AI.
```

```
Start coding or generate with AI.
```

🔷 Challenge 1: Student Management System

You are building a simple **student management system**.

1. Store students in a **dictionary**, where keys are student names and values are **tuples** containing `(age, [grades])`. Example:

   ```
   {
       "Ali": (20, [80, 90, 85]),
       "Sara": (22, [70, 60, 75])
   }
   ```

2. Write a **menu loop** where the user can:
   - Add a new student
   - Update grades
   - Calculate average grade of each student
   - Find students who passed (avg ≥ 60) and failed (avg < 60)
   - Show all unique ages using a **set**

🔷 Challenge 2: Word Frequency Analyzer

Build a **Word Frequency Analyzer** for a text paragraph.

1. Ask the user for a paragraph of text.
2. Use **string methods** to:
   - Convert all words to lowercase
   - Remove punctuation
   - Split into words (list).
3. Build a **dictionary** with word counts.
4. Find the **most frequent word** and print it.
5. Encode the text into UTF-8, then decode it back.
6. Use a **loop + zip()** to pair each word with its frequency and print nicely.

🔷 Challenge 3: Online Shopping Simulation

Simulate an **online store checkout system**.

1. You have two lists:

   ```
   items = ["laptop", "phone", "headphones", "mouse"]
   prices = [800, 500, 100, 50]
   ```

   Use **zip()** to create a dictionary of products and their prices.
2. Create a **shopping cart** as a dictionary where keys are items and values are quantities.
3. Use a **loop** to:
   - Allow the user to add items to the cart
   - If the item doesn't exist, show an error
   - Show the current cart
4. At checkout:
   - Calculate the total price
   - Apply a discount (10%) if total > 1000
   - Print the final bill
5. Bonus: Store the final bill in a **tuple** `(customer_name, total_amount, items_purchased)`

Double-click (or enter) to edit

✅ Challenge 1: Student Management System

```
# Student Management System

students = {
    "Ali": (20, [80, 90, 85]),
    "Sara": (22, [70, 60, 75])
}

while True:
    print("\n---- Student Management ----")
    print("1. Add Student")
    print("2. Update Grades")
    print("3. Show Averages")
    print("4. Pass/Fail List")
    print("5. Show Unique Ages")
    print("6. Exit")

    choice = input("Enter choice: ")
```

```
    if choice == "1":
        name = input("Enter student name: ")
        age = int(input("Enter age: "))
        students[name] = (age, [])
        print(f"{name} added!")

    elif choice == "2":
        name = input("Enter student name: ")
        if name in students:
            grade = int(input("Enter grade: "))
            students[name][1].append(grade)
            print(f"Grade {grade} added for {name}")
        else:
            print("Student not found!")

    elif choice == "3":
        for name, (age, grades) in students.items():
            if grades:
                avg = sum(grades) / len(grades)
                print(f"{name} (age {age}): Average = {avg:.2f}")
            else:
                print(f"{name}: No grades yet.")

    elif choice == "4":
        passed = []
        failed = []
        for name, (age, grades) in students.items():
            if grades:
                avg = sum(grades) / len(grades)
                if avg >= 60:
                    passed.append(name)
                else:
                    failed.append(name)
        print("✅ Passed:", passed)
        print("❌ Failed:", failed)

    elif choice == "5":
        ages = {age for age, grades in students.values()}
        print("Unique ages:", ages)

    elif choice == "6":
        print("Goodbye!")
        break
    else:
        print("Invalid choice!")
```

✅ Challenge 2: Word Frequency Analyzer

```
import string

# Step 1: Input text
text = input("Enter a paragraph: ")

# Step 2: Clean text
text = text.lower()
for p in string.punctuation:
    text = text.replace(p, "")

words = text.split()

# Step 3: Count frequencies
word_count = {}
```

```
for word in words:
    word_count[word] = word_count.get(word, 0) + 1

# Step 4: Find most frequent word
most_word = max(word_count, key=word_count.get)
print(f"Most frequent word: '{most_word}' ({word_count[most_word]} times)")

# Step 5: Encoding and Decoding
encoded = text.encode("utf-8")
decoded = encoded.decode("utf-8")
print("Decoded back:", decoded)

# Step 6: Print nicely with zip
for word, count in zip(word_count.keys(), word_count.values()):
    print(f"{word}: {count}")
```

✅ Challenge 3: Online Shopping Simulation

```
# Step 1: Items and Prices
items = ["laptop", "phone", "headphones", "mouse"]
prices = [800, 500, 100, 50]

store = dict(zip(items, prices))  # Create store dictionary
cart = {}

# Step 2: Shopping Loop
while True:
    print("\n--- Store ---")
    for item, price in store.items():
        print(f"{item} - ${price}")

    choice = input("Enter item to add (or 'checkout' to finish): ")

    if choice == "checkout":
        break
    elif choice in store:
        qty = int(input("Enter quantity: "))
        cart[choice] = cart.get(choice, 0) + qty
        print(f"Added {qty} {choice}(s) to cart.")
    else:
        print("Item not found!")

# Step 3: Checkout
total = sum(store[item] * qty for item, qty in cart.items())
print("\n🛒 Your Cart:")
for item, qty in cart.items():
    print(f"{item} x {qty} = ${store[item] * qty}")

# Step 4: Discount
if total > 1000:
    total *= 0.9
    print("🎉 10% discount applied!")

print(f"Final Total: ${total:.2f}")

# Step 5: Save final bill in tuple
customer = input("Enter your name: ")
bill = (customer, total, list(cart.keys()))
print("\nBill Summary:", bill)
```

Double-click (or enter) to edit

📘 Python File Handling

1. 🔧 **Opening a File** In Python, you use the `open()` function:

```
#open(filename, mode)
file = open("example.txt", "r")
```

Common Modes:

- `"r"` → **Read** (default). File must exist.
- `"w"` → **Write**. Creates a new file or overwrites an existing one.
- `"a"` → **Append**. Adds data to the end of the file without deleting existing content.
- `"x"` → **Create**. Creates a new file, gives an error if the file exists.
- `"b"` → Binary mode (used with images, audio, etc.).
- `"t"` → Text mode (default).

```
f = open("data.txt", "rt")  # text read
f = open("image.png", "rb") # binary read
```

2. 📖 **Reading Files**

```
with open("example.txt", "r") as f:
    content = f.read()
    print(content)
```

```
### Read first N characters:
with open("example.txt", "r") as f:
    print(f.read(10))  # first 10 characters
```

```
### Read first N characters:

with open("example.txt", "r") as f:
    print(f.read(10))  # first 10 characters
```

```
### Read line by line:
with open("example.txt", "r") as f:
    for line in f:
        print(line.strip())  # remove \n
```

```
### Read all lines into a list:


with open("example.txt", "r") as f:
    lines = f.readlines()
    print(lines)
```

3. ✍️ **Writing to Files**

```
### Overwrite (w mode):

with open("example.txt", "w") as f:
    f.write("Hello, world!\n")
    f.write("This will overwrite the old content.")
```

```
### Append (a mode):


with open("example.txt", "a") as f:
    f.write("\nNew line added at the end.")
```

4. 📄 File Position (Pointer)

The file object maintains a "cursor" (pointer).

```
#**Check position:**


with open("example.txt", "r") as f:
    print(f.tell())  # shows current position
```

```
---------------------------------------------------------------------------
FileNotFoundError                         Traceback (most recent call last)
Cell In[27], line 4
      1 #**Check position:**
----> 4 with open("example.txt", "r") as f:
      5     print(f.tell())  # shows current position

File i:\programs installed\miniConda\Lib\site-packages\IPython\core\interactiveshell.py:327, in _modified_open(file, *args, **kwargs)
    320 if file in {0, 1, 2}:
    321     raise ValueError(
    322         f"IPython won't let you open fd={file} by default "
    323         "as it is likely to crash IPython. If you know what you are doing, "
    324         "you can use builtins' open."
    325     )
--> 327 return io_open(file, *args, **kwargs)

FileNotFoundError: [Errno 2] No such file or directory: 'example.txt'
```

```
# **Move pointer:**

with open("example.txt", "r") as f:
    f.seek(5)  # move cursor to 5th character
    print(f.read())
```

5. 🧹 Closing Files

- 👉 Using `with` is better because it **automatically closes** the file.
- If you don't use `with`, you must close the file manually:

```
f = open("example.txt", "r")
print(f.read())
f.close()
```

6. ⚠️ Handling Errors

Use `try-except` to handle file errors:

```
try:
    with open("notfound.txt", "r") as f:
        data = f.read()
except FileNotFoundError:
    print("File does not exist!")
```

7. ✅ Best Practices

1. Always use `with` for safe file handling.
2. Use `"utf-8"` encoding for text files:

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

3. Catch exceptions (`try-except`).
4. For large files, read in **chunks** instead of loading everything at once:

```

```

```
with open("bigfile.txt", "r") as f:
    while chunk := f.read(1024):  # 1024 characters per chunk
        print(chunk)
```

Exercise

🔷 Exercise 1: Writing and Reading a File

1. Create a text file called `notes.txt`.
2. Write three lines of text into it.
3. Read the file and print its contents.

🔷 Exercise 2: Counting Words in a File

1. Write a paragraph (5–6 sentences) into a file called `story.txt`.
2. Write a program that:
   - Reads the file.
   - Counts how many words are in the file.
   - Prints the total word count.

🔷 Exercise 3: Copy File Content

1. Create a file `source.txt` with some text.
2. Write a program that reads the file and copies its contents into `destination.txt`.

🔷 Exercise 4: Search in a File

1. Create a file `data.txt` with at least 10 lines of text.
2. Write a program that asks the user to enter a word.
3. Check if that word exists in the file and print the line number(s) where it appears.

🔷 Exercise 5: Append Data to File

1. Create a file `log.txt`.
2. Write a program that:
   - Appends the current date and time each time the program runs.
   - Prints all the log entries.

🔷 Exercise 6 (Challenge): Student Grades System

1. Create a file `students.txt` with the following format:

```
Ali,85
Sara,90
Omar,78
```

2. Write a program that:
   - Reads the file into a dictionary (student → grade).
   - Prints all students and their grades.
   - Finds the student with the highest grade.
   - Calculates the average grade.

```
Start coding or generate with AI.
```

✅ Exercise 1: Writing and Reading a File

```
# Write to a file
with open("notes.txt", "w", encoding="utf-8") as f:
    f.write("This is the first line.\n")
    f.write("This is the second line.\n")
    f.write("This is the third line.\n")
```

```
# Read the file
with open("notes.txt", "r", encoding="utf-8") as f:
    content = f.read()
    print("File content:\n", content)
```

✅ Exercise 2: Counting Words in a File

```
# Write a paragraph into story.txt
with open("story.txt", "w", encoding="utf-8") as f:
    f.write("Once upon a time there was a student learning Python.\n")
    f.write("They practiced every day and solved many exercises.\n")
    f.write("Sometimes it was hard, but they never gave up.\n")
    f.write("With time, they became very good at programming.\n")
    f.write("Practice and patience are the keys to success.\n")

# Read and count words
with open("story.txt", "r", encoding="utf-8") as f:
    text = f.read()
    words = text.split()
    print("Total words:", len(words))
```

✅ Exercise 3: Copy File Content

```
# Create source file
with open("source.txt", "w", encoding="utf-8") as f:
    f.write("This is the content of the source file.\nIt will be copied.")

# Copy content
with open("source.txt", "r", encoding="utf-8") as source, open("destination.txt", "w", encoding="utf-8") as dest:
    for line in source:
        dest.write(line)

print("Content copied to destination.txt")
```

✅ Exercise 4: Search in a File

```
# Create data.txt with multiple lines
with open("data.txt", "w", encoding="utf-8") as f:
    f.write("Python is fun.\n")
    f.write("I love programming.\n")
    f.write("Python is powerful.\n")
    f.write("Learning new things is exciting.\n")
    f.write("Python makes automation easy.\n")

#### Search word
word = input("Enter a word to search: ")
found = False

with open("data.txt", "r", encoding="utf-8") as f:
    for i, line in enumerate(f, start=1):
        if word.lower() in line.lower():
            print(f"Found '{word}' in line {i}: {line.strip()}")
            found = True

if not found:
    print(f"'{word}' not found in the file.")
```

✅ Exercise 5: Append Data to File

```
import datetime
```

```
# Append current date and time to log.txt
with open("log.txt", "a", encoding="utf-8") as f:
    now = datetime.datetime.now()
    f.write(f"Program run at: {now}\n")

# Print all log entries
with open("log.txt", "r", encoding="utf-8") as f:
    print("Log entries:\n", f.read())
```

✅ Exercise 6 (Challenge): Student Grades System

```
# Create students.txt
with open("students.txt", "w", encoding="utf-8") as f:
    f.write("Ali,85\n")
    f.write("Sara,90\n")
    f.write("Omar,78\n")

# Read file into dictionary
students = {}
with open("students.txt", "r", encoding="utf-8") as f:
    for line in f:
        name, grade = line.strip().split(",")
        students[name] = int(grade)

# Print all students and grades
print("Student Grades:")
for name, grade in students.items():
    print(f"{name}: {grade}")

# Find highest grade
```