□ Python Programming

DAY 1: FUNDAMENTALS

Complete Lecture Notes, Code Examples, Projects & Interview Questions

Topics Covered
Introduction • Variables & Data Types • Operators • Strings • I/O • Conditionals • Loops • Data Structures

SECTION 1: Introduction to Python

1.1 What is Python?

Python is a high-level, interpreted, general-purpose programming language created by Guido van Rossum and first released in 1991. It emphasizes code readability and simplicity, allowing programmers to express concepts in fewer lines of code compared to languages like C++ or Java.

Python is named after the British comedy show 'Monty Python's Flying Circus' — not the snake!

Key Characteristics of Python

- Interpreted: Code runs line-by-line without a compilation step
- Dynamically typed: No need to declare variable types
- High-level: Abstracts away complex memory management
- Multi-paradigm: Supports procedural, object-oriented, and functional styles
- Cross-platform: Runs on Windows, macOS, Linux, and more
- Extensive standard library: 'Batteries included' philosophy

1.2 Why Python?

Use Case	Why Python Excels
`Use Case`	Why Python Excels
`Web Development`	Django, Flask, FastAPI frameworks
`Data Science`	NumPy, Pandas, Matplotlib ecosystem
`Machine Learning`	TensorFlow, PyTorch, Scikit-learn
`Automation`	Simple scripting for file/task automation
`APIs & Backend`	Fast prototyping and production-ready
`DevOps`	Ansible, Fabric, system administration

Python Popularity
Python has been ranked #1 or #2 on the TIOBE Index for several years running. It is used by Google, Instagram, Spotify, Netflix, NASA, and millions of developers worldwide.

1.3 Installing Python and IDE Setup

Installing Python

1. Go to https://python.org/downloads

2. Download the latest Python 3.x installer for your OS
3. Run installer — IMPORTANT: Check 'Add Python to PATH' on Windows
4. Verify installation by opening terminal and typing: python --version

```
# Verify Python is installed
# Open terminal/command prompt and run:
python --version
# or
python3 --version

# Expected output:
# Python 3.11.x  (or similar)
```

IDE Options

IDE	Description
`VS Code`	Free, lightweight, highly extensible. Best for beginners and professionals. Install Python extension by Microsoft.
`PyCharm`	Full-featured Python IDE by JetBrains. Community edition is free. Best for large projects.
`Jupyter Notebook`	Browser-based, cell-by-cell execution. Best for data science and learning.
`IDLE`	Comes bundled with Python. Simple, good for absolute beginners.
`Thonny`	Designed for beginners. Shows variable states step-by-step.

Setting Up VS Code for Python

5. Install VS Code from https://code.visualstudio.com
6. Open VS Code, go to Extensions (Ctrl+Shift+X)
7. Search and install 'Python' by Microsoft
8. Create a new file: hello.py
9. Write code and press F5 to run

1.4 Python Interpreter and Your First Program

The Python interpreter reads your code and executes it line by line. You can use it interactively (REPL - Read Eval Print Loop) or run .py files.

Interactive Mode (REPL)

```
# Type 'python' in terminal to enter REPL
>>> print("Hello, World!")
Hello, World!
>>> 2 + 3
```

```
5
>>> exit()  # to quit
```

Your First Python Program

```
# File: hello.py
# This is your first Python program

print("Hello, World!")
print("Welcome to Python Programming!")
print("Today we start an amazing journey!")
```

```
# Output:
# Hello, World!
# Welcome to Python Programming!
# Today we start an amazing journey!
```

Interview Q: What is Python and what makes it different from compiled languages?
Answer: Python is an interpreted, high-level, dynamically typed language. Unlike C++ or Java which require compilation to machine code before execution, Python code is executed line-by-line by an interpreter at runtime. This makes development faster but generally slower in execution speed. Python also handles memory management automatically via garbage collection.

Interview Q: What does 'dynamically typed' mean in Python?
Answer: It means you don't need to declare variable types before using them. Python infers the type at runtime. For example, x = 5 creates an integer, and x = 'hello' re-assigns x as a string. This is different from statically typed languages like Java where you must write: int x = 5;

SECTION 2: Variables and Data Types

2.1 Variables

A variable is a named container that stores a value in memory. In Python, you create a variable simply by assigning a value to a name using the = operator. There is no need for type declarations.

```
# Creating variables
name = "Alice"          # string variable
age = 25                # integer variable
height = 5.6            # float variable
is_student = True       # boolean variable

# Printing variables
print(name)             # Alice
print(age)              # 25
print(height)           # 5.6
print(is_student)       # True

# Multiple assignment
x = y = z = 0           # x, y, z all equal 0
a, b, c = 1, 2, 3       # a=1, b=2, c=3

# Swap variables (Python trick!)
a, b = b, a
print(a, b)             # 2 1
```

Variable Naming Conventions

Convention	Usage & Example
snake_case	Standard Python convention: my_variable, user_name, total_count
UPPER_CASE	Constants: MAX_SIZE, PI, DATABASE_URL
CamelCase	Class names: MyClass, BankAccount, StudentRecord
_single_under	Weak 'private' indicator: _internal_method
__double_under	Name mangling in classes: __private_attr

✓Valid vs Invalid Variable Names
Valid: name, user_age, _total, myVar2 Invalid: 2name (starts with number), user-age (hyphen), class (reserved keyword), my name (space)

```
# Good variable names
user_name = "John"
total_price = 99.99
is_logged_in = False
```

```
max_retries = 3

# Reserved keywords - CANNOT be used as variable names
# False, None, True, and, as, assert, async, await,
# break, class, continue, def, del, elif, else, except,
# finally, for, from, global, if, import, in, is, lambda,
# nonlocal, not, or, pass, raise, return, try, while, with, yield

# Check all keywords:
import keyword
print(keyword.kwlist)
```

2.2 Data Types

Python has several built-in data types. Every value in Python has a type, and you can check it using the type() function.

Numeric Types

```
# INTEGER (int) - whole numbers, no decimal point
age = 25
negative = -10
big_number = 1_000_000      # underscores for readability
binary = 0b1010             # binary: 10
octal = 0o17                # octal: 15
hexadecimal = 0xFF          # hex: 255

print(type(age))            # <class 'int'>
print(type(big_number))     # <class 'int'>

# FLOAT - numbers with decimal point
pi = 3.14159
temperature = -40.5
scientific = 1.5e3          # scientific notation: 1500.0
small = 2.5e-4              # 0.00025

print(type(pi))             # <class 'float'>

# COMPLEX - real + imaginary parts
c = 3 + 4j
print(type(c))              # <class 'complex'>
print(c.real)               # 3.0
print(c.imag)               # 4.0
```

String Type

```
# STRING (str) - sequence of characters
name = "Alice"
message = 'Hello, World!'
multiline = """This is a
multiline string"""
```

```
raw_string = r"C:\Users\Alice"  # raw string, backslash is literal

print(type(name))            # <class 'str'>
print(len(name))             # 5 (number of characters)

# Strings can use single or double quotes
greeting1 = "He said 'Hello'"   # double quotes wrapping single
greeting2 = 'She said "Hi"'     # single quotes wrapping double
```

Boolean Type

```
# BOOLEAN (bool) - True or False
is_active = True
has_permission = False
is_valid = (5 > 3)           # True (result of comparison)

print(type(is_active))       # <class 'bool'>

# Booleans are subclass of int in Python
print(True + True)           # 2
print(True + 1)              # 2
print(False * 10)            # 0

# Truthy and Falsy values
# Falsy: 0, 0.0, "", [], {}, (), None, False
# Everything else is Truthy
print(bool(0))               # False
print(bool(""))              # False
print(bool([]))              # False
print(bool(None))            # False
print(bool(42))              # True
print(bool("hello"))         # True
```

None Type

```
# NoneType - represents absence of value
result = None
print(type(result))          # <class 'NoneType'>
print(result is None)        # True (use 'is' not '==' for None)

# Common use: default parameter values
def greet(name=None):
    if name is None:
        print("Hello, stranger!")
    else:
        print(f"Hello, {name}!")
```

2.3 Type Conversion (Casting)

You can convert between data types using built-in functions. This is called explicit type conversion or casting.

```python
# Implicit conversion (Python does it automatically)
result = 5 + 2.0             # int + float = float
print(result, type(result)) # 7.0 <class 'float'>

# EXPLICIT CONVERSION (you do it manually)

# int() - convert to integer
print(int("42"))             # 42
print(int(3.9))              # 3 (truncates, does NOT round)
print(int(True))             # 1
print(int(False))            # 0
# print(int("3.14"))         # ERROR - cannot convert "3.14" directly

# float() - convert to float
print(float("3.14"))         # 3.14
print(float(42))             # 42.0
print(float("inf"))          # inf

# str() - convert to string
print(str(42))               # "42"
print(str(3.14))             # "3.14"
print(str(True))             # "True"
print(str(None))             # "None"

# bool() - convert to boolean
print(bool(1))               # True
print(bool(0))               # False
print(bool(""))              # False
print(bool("hello"))         # True

# Type checking
x = 42
print(type(x))               # <class 'int'>
print(isinstance(x, int))    # True
print(isinstance(x, float)) # False
print(isinstance(x, (int, float)))  # True (check multiple types)
```

Interview Q: What is the difference between int() and round() when converting float to int?
Answer: int() truncates (removes the decimal part): int(3.9) = 3, int(-3.9) = -3. round() rounds to nearest integer: round(3.9) = 4, round(3.5) = 4, round(-3.5) = -4. For banking or scientific work, round() is almost always preferred.

Interview Q: What are truthy and falsy values in Python?
Answer: Falsy values evaluate to False in a boolean context: 0, 0.0, 0j, '' (empty string), [] (empty list), {} (empty dict), () (empty tuple), set() (empty set), None, and False itself. Everything else is truthy. This allows writing: if my_list: instead of if len(my_list) > 0:

Interview Q: What is the difference between == and is operators?
Answer: == checks value equality (do they have the same value?). is checks identity (do they point to the same object in memory?). For None, always use is None. Example: a = [1,2]; b = [1,2]; a == b is True, but a is b is False because they are different objects.

SECTION 3: Basic Operators

3.1 Arithmetic Operators

```
a = 17
b = 5

print(a + b)    # Addition:         22
print(a - b)    # Subtraction:      12
print(a * b)    # Multiplication:   85
print(a / b)    # Division:         3.4  (always float)
print(a // b)   # Floor Division:   3    (integer quotient)
print(a % b)    # Modulus:          2    (remainder)
print(a ** b)   # Exponentiation:   1419857

# Real-world usage
price = 100
discount = 15
final = price - (price * discount / 100)
print(f"Final price: {final}")    # 85.0

# Check if number is even/odd using modulus
number = 17
if number % 2 == 0:
    print("Even")
else:
    print("Odd")    # Odd

# Integer division use case
minutes = 137
hours = minutes // 60        # 2
remaining_min = minutes % 60 # 17
print(f"{hours}h {remaining_min}m")  # 2h 17m
```

3.2 Comparison Operators

```
x = 10
y = 20

print(x == y)   # Equal to:             False
print(x != y)   # Not equal to:         True
print(x > y)    # Greater than:         False
print(x < y)    # Less than:            True
print(x >= y)   # Greater or equal:     False
print(x <= y)   # Less or equal:        True

# Chained comparisons (Python unique feature!)
age = 25
print(18 <= age <= 65)      # True - valid working age range

# Comparing strings (alphabetical order)
```

```
print("apple" < "banana")   # True
print("Python" == "python") # False (case-sensitive)
print("abc" == "abc")       # True
```

3.3 Logical Operators

```
# AND - True only if BOTH are True
print(True and True)     # True
print(True and False)    # False
print(False and True)    # False

# OR - True if AT LEAST ONE is True
print(True or False)     # True
print(False or False)    # False
print(True or True)      # True

# NOT - reverses boolean
print(not True)          # False
print(not False)         # True

# Practical example: login validation
username = "admin"
password = "secret123"
is_active = True

if username == "admin" and password == "secret123" and is_active:
    print("Login successful!")
else:
    print("Login failed!")

# Short-circuit evaluation
# 'and' stops at first False
# 'or' stops at first True
x = None
result = x or "default"     # "default" (since x is falsy)
print(result)

name = "Alice"
result = name or "Anonymous"  # "Alice" (since name is truthy)
print(result)
```

3.4 Assignment Operators

```
x = 10
x += 5      # x = x + 5    => 15
x -= 3      # x = x - 3    => 12
x *= 2      # x = x * 2    => 24
x /= 4      # x = x / 4    => 6.0
x //= 2     # x = x // 2   => 3.0
x %= 2      # x = x % 2    => 1.0
x **= 3     # x = x ** 3   => 1.0
```

```
# Walrus operator := (Python 3.8+)
# Assigns and returns a value in the same expression
numbers = [1, 2, 3, 4, 5]
if (n := len(numbers)) > 3:
    print(f"List is long: {n} elements")  # List is long: 5 elements
```

3.5 Bitwise Operators

```
a = 12   # binary: 1100
b = 10   # binary: 1010

print(a & b)     # AND:  1000 = 8
print(a | b)     # OR:   1110 = 14
print(a ^ b)     # XOR:  0110 = 6
print(~a)        # NOT:  ...0011 = -13
print(a << 1)    # Left shift:  11000 = 24
print(a >> 1)    # Right shift: 0110 = 6

# Practical: Check if number is even using bitwise
def is_even(n):
    return (n & 1) == 0   # Last bit is 0 for even numbers
```

3.6 Operator Precedence

```
# Order: () > ** > unary > * / // % > + - > comparisons > not > and > or
print(2 + 3 * 4)        # 14 (not 20, * before +)
print((2 + 3) * 4)      # 20 (parentheses override)
print(2 ** 3 ** 2)      # 512 (right-to-left: 3**2=9, 2**9=512)
print(10 - 2 + 3)       # 11 (left-to-right)
print(not True or True) # True (not True = False, False or True = True)
```

Interview Q: What is the difference between / and // in Python?
Answer: / always returns a float even when dividing two integers: 10/3 = 3.3333. // is floor division, it always returns the floor (rounds down): 10//3 = 3, -7//2 = -4 (not -3!). This is important for algorithms that need integer division.

Interview Q: What is short-circuit evaluation? Why is it useful?
Answer: In 'and', Python stops evaluating when it finds the first False. In 'or', it stops at the first True. This is useful for performance and for guarding against errors: if obj is not None and obj.value > 0: — if obj is None, the second condition is never checked, preventing an AttributeError.

SECTION 4: String Manipulation

Strings are one of the most commonly used data types. Python provides rich built-in methods to manipulate strings efficiently.

4.1 String Basics

```
# Creating strings
s1 = "Hello"
s2 = 'World'
s3 = """Multi
line
string"""
s4 = r"C:\path\to\file"   # raw string

# String properties
print(len("Hello"))          # 5
print("Hello"[0])            # H (first character)
print("Hello"[-1])           # o (last character)
print(type("Hello"))         # <class 'str'>

# Strings are IMMUTABLE
word = "Hello"
# word[0] = "h"   # TypeError! Cannot modify string
word = "hello"    # This creates a NEW string
```

4.2 String Indexing and Slicing

```
# String:   H  e  l  l  o  ,     W  o  r  l  d
# Index:    0  1  2  3  4  5  6  7  8  9  10  11
# Neg:    -12 -11 -10 -9 -8 -7 -6 -5 -4 -3  -2  -1

text = "Hello, World"

# Indexing
print(text[0])      # H
print(text[7])      # W
print(text[-1])     # d
print(text[-5])     # W

# Slicing: [start:stop:step]  (stop is exclusive)
print(text[0:5])    # Hello
print(text[7:12])   # World
print(text[:5])     # Hello (start defaults to 0)
print(text[7:])     # World (stop defaults to end)
print(text[:])      # Hello, World (full copy)
print(text[::2])    # Hlo ol (every 2nd character)
print(text[::-1])   # dlroW ,olleH (reverse!)
```

4.3 String Methods

```
text = "  Hello, Python World!  "

# Case methods
print(text.upper())             # "  HELLO, PYTHON WORLD!  "
print(text.lower())             # "  hello, python world!  "
print(text.title())             # "  Hello, Python World!  "
print(text.capitalize())        # "  hello, python world!  "
print(text.swapcase())          # "  hELLO, pYTHON wORLD!  "

# Stripping whitespace
print(text.strip())             # "Hello, Python World!"
print(text.lstrip())            # "Hello, Python World!  "
print(text.rstrip())            # "  Hello, Python World!"

# Searching
sentence = "The quick brown fox"
print(sentence.find("quick"))       # 4 (index), -1 if not found
print(sentence.index("fox"))        # 16 (raises error if not found)
print(sentence.count("o"))          # 2
print(sentence.startswith("The"))   # True
print(sentence.endswith("fox"))     # True
print("quick" in sentence)          # True (in operator)

# Replacing
print(sentence.replace("quick", "slow"))  # "The slow brown fox"
print(sentence.replace("o", "0", 1))      # Replace only 1st occurrence

# Splitting and joining
csv_data = "apple,banana,cherry"
fruits = csv_data.split(",")         # ['apple', 'banana', 'cherry']
print(fruits)

words = ["Python", "is", "awesome"]
joined = " ".join(words)             # "Python is awesome"
print(joined)
print("-".join(words))               # "Python-is-awesome"

# Checking content
print("12345".isdigit())      # True
print("hello".isalpha())      # True
print("hello123".isalnum())   # True
print("   ".isspace())        # True
print("Hello World".istitle()) # True
```

4.4 String Formatting

```
name = "Alice"
age = 30
score = 95.75

# Method 1: f-strings (Python 3.6+) - RECOMMENDED
print(f"Name: {name}, Age: {age}")
print(f"Score: {score:.2f}")          # 2 decimal places: 95.75
print(f"Score: {score:.0f}")          # 0 decimal places: 96
print(f"Score: {score:10.2f}")        # Width 10: "     95.75"
print(f"{'Left':<10}|{'Right':>10}") # Alignment: "Left      |     Right"
print(f"{name!r}")                     # Repr: 'Alice' (with quotes)
print(f"{1000000:,}")                 # Thousand separator: 1,000,000
print(f"{0.0125:.2%}")                # Percentage: 1.25%
print(f"{255:08b}")                   # Binary: 11111111
print(f"{255:x}")                     # Hex: ff

# Method 2: .format()
print("Name: {}, Age: {}".format(name, age))
print("Name: {0}, again: {0}".format(name))  # Reuse by index
print("Name: {n}, Age: {a}".format(n=name, a=age))  # by keyword
print("{:.2f}".format(score))

# Method 3: % operator (old style, avoid in new code)
print("Name: %s, Age: %d" % (name, age))
print("Score: %.2f" % score)
```

4.5 String Escape Sequences

```
# Common escape sequences
print("Line 1\nLine 2")      # \n = newline
print("Tab\there")           # \t = tab
print("He said \"Hello\"")   # \" = double quote
print('It\'s Python')         # \' = single quote
print("Backslash: \\")        # \\ = backslash
print("\a")                   # \a = bell
print("\0")                   # \0 = null character

# Raw strings ignore escape sequences
path = r"C:\Users\Alice\Documents"
print(path)   # C:\Users\Alice\Documents (literal backslashes)
```

Interview Q: What is the difference between find() and index() in strings?
Answer: Both find the first occurrence of a substring. find() returns -1 if not found. index() raises a ValueError if not found. Use find() when you want to check if something exists (check if result == -1), and use index() when you're confident it exists and want an exception for debugging if it doesn't.

Interview Q: How do you reverse a string in Python?
Answer: The most Pythonic way is using slicing with a step of -1: reversed_str = my_string[::-1]. You can also use ''.join(reversed(my_string)) or ''.join(list(my_string)[::-1]). The slice method is preferred for its simplicity and performance.

Interview Q: Are strings mutable in Python?
Answer: No, strings are immutable. Once created, you cannot change individual characters. Any string operation (like replace, upper, etc.) returns a NEW string object — it doesn't modify the original. This design enables Python to safely use strings as dictionary keys and makes them hashable.

SECTION 5: Input/Output Operations

5.1 The print() Function

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

# Multiple items separated by space (default)
print("Name:", "Alice", "Age:", 30)  # Name: Alice Age: 30

# Custom separator
print("a", "b", "c", sep="-")         # a-b-c
print("a", "b", "c", sep=", ")        # a, b, c
print("a", "b", "c", sep="")          # abc

# Custom end character (default is \n)
print("Hello", end=" ")
print("World")                          # Hello World (on same line)

print("Hello", end="!\n")              # Hello!

# Print multiple items on same line
for i in range(5):
    print(i, end=" ")                   # 0 1 2 3 4
print()                                 # newline after loop

# Print to stderr (for error messages)
import sys
print("Error occurred!", file=sys.stderr)

# Formatted print
items = ["apple", "banana", "cherry"]
print(*items)                           # apple banana cherry (unpacking)
print(*items, sep="\n")                # each on new line

# Print with repr (useful for debugging)
text = "Hello\nWorld"
print(repr(text))    # 'Hello\nWorld' (shows \n as literal)
```

5.2 The input() Function

```
# Basic input - always returns a STRING
name = input("Enter your name: ")
print(f"Hello, {name}!")

# Input with type conversion
age = int(input("Enter your age: "))
height = float(input("Enter your height in meters: "))
print(f"In 5 years you'll be {age + 5} years old")
```

```
# Safe input with error handling
while True:
    try:
        number = int(input("Enter a number: "))
        break   # exit loop if successful
    except ValueError:
        print("Invalid! Please enter a whole number.")

# Multiple inputs on one line
# User types: 10 20 30
values = input("Enter 3 numbers: ").split()
a, b, c = int(values[0]), int(values[1]), int(values[2])

# Pythonic way
a, b, c = map(int, input("Enter 3 numbers: ").split())
print(f"Sum: {a + b + c}")

# Input for list of numbers
numbers = list(map(int, input("Enter numbers: ").split()))
print(f"Sum: {sum(numbers)}")
print(f"Max: {max(numbers)}")
```

5.3 Formatted Output

```
# Creating a formatted table with print
print("=" * 40)
print(f"{'Student Name':<20} {'Score':>8} {'Grade':>8}")
print("=" * 40)

students = [
    ("Alice Johnson", 95, "A"),
    ("Bob Smith", 82, "B"),
    ("Charlie Brown", 67, "C"),
]

for name, score, grade in students:
    print(f"{name:<20} {score:>8} {grade:>8}")

print("=" * 40)

# Output:
# ========================================
# Student Name            Score    Grade
# ========================================
# Alice Johnson              95        A
# Bob Smith                  82        B
# Charlie Brown              67        C
# ========================================
```

Interview Q: What does input() always return, and why is this important?
Answer: input() always returns a string, regardless of what the user types. If you type 25, Python receives the string '25', not the integer 25. This is important because string '5' + string '3' = '53'

(concatenation), not 8 (addition). Always convert with int(), float(), etc. when you need numeric operations.

SECTION 6: Conditional Statements

Conditional statements allow your program to make decisions — executing different code blocks based on whether conditions are True or False.

6.1 if, elif, else

```python
# Basic if statement
age = 20
if age >= 18:
    print("You are an adult.")

# if-else
score = 75
if score >= 60:
    print("Pass")
else:
    print("Fail")

# if-elif-else chain
grade = 85
if grade >= 90:
    letter = "A"
elif grade >= 80:
    letter = "B"
elif grade >= 70:
    letter = "C"
elif grade >= 60:
    letter = "D"
else:
    letter = "F"
print(f"Grade: {letter}")   # B

# Important: Python uses INDENTATION (4 spaces) to define blocks
# NO curly braces like in C++/Java

# Multiple conditions
username = "admin"
password = "pass123"
is_active = True

if username == "admin" and password == "pass123":
    if is_active:
        print("Welcome, Admin!")
    else:
        print("Account is deactivated.")
else:
    print("Invalid credentials.")
```

6.2 Nested Conditions

```python
# ATM-like program
balance = 1500
requested = 200
pin_correct = True

if pin_correct:
    if requested <= balance:
        if requested > 0:
            balance -= requested
            print(f"Dispensing ${requested}")
            print(f"Remaining balance: ${balance}")
        else:
            print("Amount must be positive.")
    else:
        print(f"Insufficient funds. Balance: ${balance}")
else:
    print("Incorrect PIN. Please try again.")

# Better approach: flatten with and
if pin_correct and 0 < requested <= balance:
    balance -= requested
    print(f"Dispensing ${requested}. Balance: ${balance}")
```

6.3 Ternary Operator (Conditional Expression)

```python
# Syntax: value_if_true if condition else value_if_false
age = 20
status = "adult" if age >= 18 else "minor"
print(status)   # adult

# Equivalent to:
if age >= 18:
    status = "adult"
else:
    status = "minor"

# In print
score = 75
print("Pass" if score >= 60 else "Fail")

# Nested ternary (use sparingly - reduces readability)
n = 5
result = "positive" if n > 0 else "negative" if n < 0 else "zero"
print(result)   # positive

# Common uses
max_val = a if a > b else b  # max without function
abs_val = x if x >= 0 else -x  # absolute value
```

6.4 match Statement (Python 3.10+)

```python
# match is Python's version of switch/case
```

```
command = "quit"

match command:
    case "start":
        print("Starting...")
    case "stop":
        print("Stopping...")
    case "quit" | "exit":      # multiple values with |
        print("Goodbye!")
    case _:                     # default case
        print("Unknown command")

# Match with guards (conditions)
point = (1, 0)
match point:
    case (0, 0):
        print("Origin")
    case (x, 0):
        print(f"On x-axis at {x}")
    case (0, y):
        print(f"On y-axis at {y}")
    case (x, y):
        print(f"Point at ({x}, {y})")
```

Interview Q: What is the difference between 'if x:' and 'if x is not None:'?
Answer: 'if x:' checks if x is truthy — it's False for 0, empty string, empty list, None, etc. 'if x is not None:' ONLY checks if x is not the None value — it's True for 0, '', [], etc. Use 'if x is not None:' when 0 or '' are valid values you want to allow through.

Interview Q: Can you have an if without else?
Answer: Yes. An if without else simply does nothing when the condition is False. This is valid and very common: if debug_mode: print('Debug info'). The else branch is optional and should only be added when there's meaningful code for the False case.

SECTION 7: Loops

Loops allow you to execute a block of code repeatedly. Python has two types of loops: for loops (for iterating over a sequence) and while loops (for repeating while a condition is true).

7.1 for Loops

```python
# Basic for loop over a list
fruits = ["apple", "banana", "cherry"]
for fruit in fruits:
    print(fruit)
# apple
# banana
# cherry

# for loop with range()
for i in range(5):           # 0, 1, 2, 3, 4
    print(i, end=" ")
print()

for i in range(1, 6):        # 1, 2, 3, 4, 5
    print(i, end=" ")
print()

for i in range(0, 10, 2):    # 0, 2, 4, 6, 8 (step=2)
    print(i, end=" ")
print()

for i in range(10, 0, -1):   # 10, 9, 8, ...1 (countdown)
    print(i, end=" ")
print()

# Iterating over string
for char in "Python":
    print(char, end="-")     # P-y-t-h-o-n-

# Iterating with enumerate (index + value)
fruits = ["apple", "banana", "cherry"]
for index, fruit in enumerate(fruits):
    print(f"{index}: {fruit}")
# 0: apple, 1: banana, 2: cherry

for index, fruit in enumerate(fruits, start=1):  # start from 1
    print(f"{index}. {fruit}")

# Iterating over dictionary
student = {"name": "Alice", "age": 20, "grade": "A"}
for key in student:                     # iterate keys
    print(key)
for value in student.values():         # iterate values
    print(value)
for key, value in student.items():     # iterate both
```

```
    print(f"{key}: {value}")

# zip - iterate multiple lists together
names = ["Alice", "Bob", "Charlie"]
scores = [95, 82, 78]
for name, score in zip(names, scores):
    print(f"{name}: {score}")
```

7.2 while Loops

```
# Basic while loop
count = 0
while count < 5:
    print(count, end=" ")   # 0 1 2 3 4
    count += 1              # CRITICAL: always update to avoid infinite loop

# Input validation with while
while True:
    age = input("Enter your age (0-120): ")
    if age.isdigit() and 0 <= int(age) <= 120:
        age = int(age)
        break
    print("Invalid age. Try again.")
print(f"Your age is: {age}")

# Countdown timer
import time
seconds = 5
while seconds > 0:
    print(f"T-{seconds}...")
    # time.sleep(1)
    seconds -= 1
print("Launch!")

# Finding first occurrence
numbers = [3, 7, 1, 8, 4, 9, 2, 6]
target = 8
index = 0
found = False
while index < len(numbers):
    if numbers[index] == target:
        found = True
        break
    index += 1

if found:
    print(f"Found {target} at index {index}")
else:
    print(f"{target} not found")
```

7.3 Loop Control: break, continue, pass

```
# BREAK - exit the loop immediately
for i in range(10):
    if i == 5:
        break            # exits when i == 5
    print(i, end=" ")    # 0 1 2 3 4

# CONTINUE - skip current iteration, go to next
for i in range(10):
    if i % 2 == 0:
        continue         # skip even numbers
    print(i, end=" ")    # 1 3 5 7 9

# PASS - placeholder, does nothing
for i in range(5):
    if i == 3:
        pass             # placeholder (code to be added later)
    print(i, end=" ")    # 0 1 2 3 4 (pass does nothing)

# else clause with loops (runs if loop completes normally, not via break)
for i in range(2, 10):
    for j in range(2, i):
        if i % j == 0:
            break
    else:
        print(f"{i} is prime")   # 2 3 5 7 are prime

# Search with for-else pattern
names = ["Alice", "Bob", "Charlie"]
target = "Dave"
for name in names:
    if name == target:
        print(f"Found: {target}")
        break
else:
    print(f"{target} not found")   # Dave not found
```

7.4 Nested Loops

```
# Multiplication table
for i in range(1, 4):
    for j in range(1, 4):
        print(f"{i}x{j}={i*j}", end="  ")
    print()   # newline after each row

# Triangle pattern
rows = 5
for i in range(1, rows + 1):
    print("*" * i)
# *
# **
# ***
# ****
# *****

# Matrix operations
```

```
matrix = [
    [1, 2, 3],
    [4, 5, 6],
    [7, 8, 9]
]
for row in matrix:
    for element in row:
        print(f"{element:3}", end="")
    print()

# Nested loops with break (only breaks inner loop)
for i in range(3):
    for j in range(3):
        if j == 1:
            break          # only breaks inner loop
        print(f"({i},{j})", end=" ")
    print()
# (0,0) (1,0) (2,0)
```

7.5 List Comprehensions (Bonus - Pythonic Loops)

```
# Traditional loop
squares = []
for i in range(1, 6):
    squares.append(i ** 2)

# List comprehension (more Pythonic!)
squares = [i ** 2 for i in range(1, 6)]
print(squares)    # [1, 4, 9, 16, 25]

# With condition
evens = [i for i in range(20) if i % 2 == 0]
print(evens)      # [0, 2, 4, 6, 8, 10, 12, 14, 16, 18]

# String manipulation
words = ["hello", "world", "python"]
upper = [word.upper() for word in words]
print(upper)      # ['HELLO', 'WORLD', 'PYTHON']

# Nested comprehension - flatten matrix
matrix = [[1,2,3],[4,5,6],[7,8,9]]
flat = [n for row in matrix for n in row]
print(flat)       # [1, 2, 3, 4, 5, 6, 7, 8, 9]
```

Interview Q: What is the difference between break and continue?
Answer: break immediately terminates the entire loop and execution jumps to the first statement after the loop. continue skips the rest of the current iteration and jumps back to the loop's condition check (for while) or the next iteration (for for). Think of continue as 'skip this one' and break as 'stop the whole loop'.

Interview Q: What is a list comprehension and when should you use it?

Answer: A list comprehension is a concise way to create lists: [expression for item in iterable if condition]. Use it for simple transformations and filters. Avoid it for complex logic — if it becomes unreadable, use a regular for loop. Comprehensions are generally faster than equivalent for loops with append().

SECTION 8: Basic Data Structures

8.1 Lists

A list is an ordered, mutable (changeable) collection that allows duplicate elements. Lists are one of the most versatile data structures in Python.

```
# Creating lists
empty = []
empty2 = list()
numbers = [1, 2, 3, 4, 5]
mixed = [1, "hello", 3.14, True, None]   # mixed types allowed!
nested = [[1, 2], [3, 4], [5, 6]]        # nested lists

# Creating with list()
from_range = list(range(1, 6))           # [1, 2, 3, 4, 5]
from_string = list("Python")             # ['P', 'y', 't', 'h', 'o', 'n']

# Accessing elements
fruits = ["apple", "banana", "cherry", "date", "elderberry"]
print(fruits[0])        # apple (first)
print(fruits[-1])       # elderberry (last)
print(fruits[1:3])      # ['banana', 'cherry'] (slicing)
print(fruits[::-1])     # reversed list
print(len(fruits))      # 5

# Modifying elements (lists are MUTABLE)
fruits[0] = "avocado"    # change first element
fruits[1:3] = ["blueberry", "coconut"]  # replace slice

print(fruits)
```

8.2 List Methods

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

# ADDING elements
fruits.append("date")           # add to end: ['apple', 'banana', 'cherry',
'date']
fruits.insert(1, "avocado")     # insert at index 1
fruits.extend(["elderberry", "fig"])  # add multiple items

# REMOVING elements
fruits.remove("banana")         # remove by value (first occurrence)
popped = fruits.pop()           # remove and return last item
popped2 = fruits.pop(0)         # remove and return item at index 0
del fruits[0]                   # delete by index
# fruits.clear()                # remove all items

# FINDING elements
```

```
fruits = ["apple", "banana", "cherry", "banana"]
print(fruits.index("banana"))   # 1 (first occurrence)
print(fruits.count("banana"))   # 2 (how many times)
print("apple" in fruits)        # True

# SORTING
numbers = [3, 1, 4, 1, 5, 9, 2, 6]
numbers.sort()                  # sort in place: [1, 1, 2, 3, 4, 5, 6, 9]
numbers.sort(reverse=True)      # descending: [9, 6, 5, 4, 3, 2, 1, 1]

words = ["banana", "Apple", "cherry"]
words.sort()                    # alphabetical: ['Apple', 'banana', 'cherry']
words.sort(key=str.lower)       # case-insensitive sort

# sorted() returns NEW list (doesn't modify original)
nums = [3, 1, 4, 1, 5]
sorted_nums = sorted(nums)
print(nums)         # [3, 1, 4, 1, 5] (unchanged)
print(sorted_nums)  # [1, 1, 3, 4, 5]

numbers.reverse()               # reverse in place

# OTHER methods
nums = [1, 2, 3, 4, 5]
print(min(nums))    # 1
print(max(nums))    # 5
print(sum(nums))    # 15
copied = nums.copy()  # shallow copy
```

8.3 Tuples

A tuple is an ordered, immutable collection. Once created, its contents cannot be changed. Tuples are faster than lists and are used for data that should not be modified.

```
# Creating tuples
empty = ()
single = (42,)          # IMPORTANT: trailing comma for single-element tuple
single2 = 42,           # also valid
coordinates = (10, 20)
rgb = (255, 128, 0)
mixed = (1, "hello", 3.14)

# Packing and unpacking
point = 10, 20          # packing (parentheses optional)
x, y = point            # unpacking
print(x, y)             # 10 20

# Extended unpacking
first, *rest = [1, 2, 3, 4, 5]
print(first)    # 1
print(rest)     # [2, 3, 4, 5]

*start, last = [1, 2, 3, 4, 5]
print(start)    # [1, 2, 3, 4]
```

```python
print(last)     # 5

# Tuples are IMMUTABLE
coords = (10, 20, 30)
print(coords[0])     # 10
# coords[0] = 99    # TypeError! tuples cannot be modified

# Tuple methods
numbers = (1, 2, 3, 2, 4, 2)
print(numbers.count(2))    # 3
print(numbers.index(3))    # 2

# Converting between list and tuple
my_list = [1, 2, 3]
my_tuple = tuple(my_list)    # list to tuple
my_list2 = list(my_tuple)    # tuple to list

# Named tuples (more readable tuples)
from collections import namedtuple
Point = namedtuple('Point', ['x', 'y'])
p = Point(x=10, y=20)
print(p.x, p.y)      # 10 20
print(p[0], p[1])    # 10 20 (also works by index)

# Tuple vs List comparison
import sys
my_list = [1, 2, 3, 4, 5]
my_tuple = (1, 2, 3, 4, 5)
print(sys.getsizeof(my_list))    # larger (typically ~104 bytes)
print(sys.getsizeof(my_tuple))   # smaller (typically ~80 bytes)
```

When to Use Tuple vs List
Use TUPLE when: data should not change (coordinates, RGB values, database records), using as dictionary keys, returning multiple values from functions, or slight performance matters. Use LIST when: you need to add/remove/modify elements, order matters and content changes, or you need list methods like sort/append.

8.4 Introduction to Dictionaries and Sets

```python
# DICTIONARY - key-value pairs (covered more in Day 2)
person = {
    "name": "Alice",
    "age": 30,
    "city": "New York"
}
print(person["name"])        # Alice
print(person.get("age"))     # 30
person["email"] = "alice@email.com"  # add new key

# SET - unordered, unique elements (covered more in Day 2)
unique_nums = {1, 2, 3, 2, 1, 4}
print(unique_nums)           # {1, 2, 3, 4} (duplicates removed)
```

Interview Q: What is the difference between a list and a tuple in Python?
Answer: Lists are mutable (can be changed) while tuples are immutable (cannot be changed after creation). Lists use [] and tuples use (). Tuples are faster to iterate and use less memory. Tuples can be used as dictionary keys (they're hashable) while lists cannot. Use tuples for data that shouldn't change (coordinates, settings) and lists for data that will be modified.

Interview Q: What is the difference between remove(), pop(), and del for lists?
Answer: remove(value) removes the first occurrence of a specific value and raises ValueError if not found. pop(index) removes and RETURNS the item at the given index (default -1 = last item). del list[index] removes by index but returns nothing. pop() is useful when you need the removed value.

Interview Q: What is the difference between list.sort() and sorted()?
Answer: list.sort() modifies the list in-place and returns None. sorted(iterable) returns a NEW sorted list and doesn't modify the original. sorted() works on any iterable (strings, tuples, etc.), while sort() only works on lists. Use sorted() when you need to keep the original order or when working with non-list iterables.

SECTION 9: Mini Projects

□□ projects □□□□ □□□□ □□□ □□□□□□□ □□□□□□□□□□ Section 1 □□□□ Section 8 □□□□□□□□ □□ □□□□□□□ □□ □□□□□□□ □□□□ function □□□ — □□□□ variables, data types, operators, strings, input/output, conditionals, loops, lists, □□□ tuples □□□□□□□□ □□□ □□□□□□□□ □□□□□□□□ project □□ □□□ theory explanation □□□ □□□□ □□□□ □□□□□□ □□□□ □□□ concept □□□□□□□ □□□□ □□□□□□□

□ **Important Note for Beginners**
□□ projects □ functions (def keyword) □□□□□□□□ □□□ □□□□□□□ □□□□□□□□ □□□□ □□□ □□□□ □□□□ □□□□□□□ execute □□□□ □□□□ code □□ copy □□□ □□□□□□□ run □□□□ □□□□□□□ functions □□□□□□ Day 2 □□□

Project 1: Number Guessing Game

Theory — □□ Project □ □□ □□ □□□□□□□□□ □□□□□□□

□□ project □ □□□□ □□□□□□ □□□□□□□ import statement □□□□□□ random module □□□□ random number generate □□□□ □□□, while True loop □□□□□□ indefinitely □□□□□□□ □□□, break □□□□□□ loop □□□□ □□□□ □□□, □□□ if/elif/else □□□□□□ guess compare □□□□ □□□□

Concept	□□□ □□□□□□□□□ □□□□□□□
`import random`	Python □□ built-in random module load □□□□ random.randint(1,100) □□□□□□ 1 □□□□ 100 □□ □□□□□□ □□□□□□□ integer □□□□□□□ □□□□□□
`while True:`	Infinite loop — break □□ □□□□□□□ □□□□□□□□ □□□□ □□□□□□ Input validation □□ □□□□ □□□ □□□□□□□
`attempts list`	□□□□□□□□ guess list □ append □□□ □□□□ □□□ — □□□ □□ guess □□□□□□□ □□□□□□□□ □□□□□□
`break`	□□□□□ guess □□ maximum □□□□□□□ □□□ □□□ loop □□□□ □□□ □□□□□□□ □□□□ □□□□□□□□□□
`Chained comparison`	1 <= guess <= 100 — □□□ Python □□ □□□□□□ feature, □□ □□□□□□ range check□

```
# ==============================================
# PROJECT 1: NUMBER GUESSING GAME
# Concepts used: import, variables, while loop,
#   if/elif/else, input, int(), list, append,
#   len(), break, continue, string formatting
# ==============================================

import random

print("=" * 45)
```

```python
print("      □ NUMBER GUESSING GAME")
print("=" * 45)
print("I am thinking of a number between 1 and 100.")
print("You have 7 chances to guess it!")
print("=" * 45)

# Generate a secret random number between 1 and 100
secret_number = random.randint(1, 100)

# Variables to track the game
max_attempts = 7
attempts_used = 0
guesses_made = []    # list to store all guesses
game_won = False

# Main game loop
while attempts_used < max_attempts:
    remaining = max_attempts - attempts_used
    print(f"\nAttempts remaining: {remaining}")

    # Get user input
    user_input = input("Your guess: ")

    # Check if input is a valid number
    if not user_input.isdigit():
        print("✕ Please enter a valid number!")
        continue    # skip rest, go back to top of loop

    guess = int(user_input)

    # Check range
    if guess < 1 or guess > 100:
        print("✕ Number must be between 1 and 100!")
        continue

    # Store the guess in our list
    guesses_made.append(guess)
    attempts_used += 1

    # Compare guess with secret number
    if guess == secret_number:
        game_won = True
        break       # exit the loop — game is over!
    elif guess < secret_number:
        difference = secret_number - guess
        if difference > 30:
            print("□ Way too LOW! Very far away.")
        elif difference > 10:
            print("□ Too LOW! Getting warmer...")
        else:
            print("□ Too LOW! Very close!")
    else:
        difference = guess - secret_number
        if difference > 30:
            print("□ Way too HIGH! Very far away.")
        elif difference > 10:
            print("□ Too HIGH! Getting warmer...")
        else:
            print("□ Too HIGH! Very close!")

# ---- Game Result ----
print("\n" + "=" * 45)
if game_won:
    print(f"□ CONGRATULATIONS! You won!")
    print(f"The number was: {secret_number}")
    print(f"You guessed it in {attempts_used} attempt(s)!")

    # Give a rating based on attempts
    if attempts_used <= 3:
        rating = "*** GENIUS!"
    elif attempts_used <= 5:
        rating = "** GREAT!"
    else:
        rating = "* GOOD JOB!"
    print(f"Rating: {rating}")
else:
    print(f"□ GAME OVER! You ran out of attempts.")
    print(f"The secret number was: {secret_number}")

# Show history of guesses
print(f"\nYour guesses were: {guesses_made}")
print(f"Total guesses made: {len(guesses_made)}")
print("=" * 45)
```

□ Key Learning Points from Project 1
1. import random — module □□□□ code borrow □□□ 2. while + break pattern — condition-based loop exit 3. continue — invalid input skip □□□ 4. List as a log — guesses_made list □ □□ data □□□□ 5. Nested if/elif inside loop — multiple condition check

Project 2: Student Report Card

Theory — □□ Project □ □□ □□ □□□□□□□□ □□□□□□□

□□ project □ □□□□ □□□□□ □□□□□□□ □□□□ real-world problem □□ Python □□□□□ solve □□□ □□□□□ □□□□ student □□ multiple subject □□ marks □□□□□□□ □□□, □□□□□□□ list □ store □□□ □□□, □□□□□ average, highest, lowest □□□ □□□ □□□ □□□ grade □□□□□□□ □□□□

Concept	□□□ □□□□□□□□ □□□□□□□
`for loop + range()`	□□□□□□□□□□ □□□□□□□ □□□ loop □□□□□□□ — □□□□□ subject □□□□□□□ □□□□□□□□□□□
`list.append()`	□□□□□□□□ subject □□ □□□ □ marks dynamically list □ add □□□□
`sum() / len()`	Average □□□ □□□□: sum(marks) / len(marks)□ Built-in functions□
`max() / min()`	□□□□□□□□□ □ □□□□□□□□□□ marks □□□ □□□□
`zip()`	□□□□□ list □□□□□□□ iterate □□□ — subjects □ marks list□
`f-string formatting`	Report card □ aligned columns □□□□ □□□□ :<20 □□□ :>8 □□□□□□□□□

```python
# ================================================
# PROJECT 2: STUDENT REPORT CARD
# Concepts used: variables, for loop, range(),
#   while loop, list, append, sum, max, min,
#   zip, if/elif/else, f-strings, input/output
# ================================================

print("=" * 50)
print("       □ STUDENT REPORT CARD SYSTEM")
print("=" * 50)

# Get student information
student_name = input("Enter student name: ").strip()
student_class = input("Enter class/grade: ").strip()

# Get number of subjects
num_subjects = 0
while num_subjects <= 0:
    user_input = input("How many subjects? ")
    if user_input.isdigit():
        num_subjects = int(user_input)
        if num_subjects <= 0:
            print("Must have at least 1 subject!")
    else:
        print("Please enter a valid number!")

# ---- Collect Subject Names and Marks ----
subject_names = []    # list to store subject names
subject_marks = []    # list to store marks

print(f"\nEnter marks for {num_subjects} subjects (out of 100):")
print("-" * 40)

for i in range(num_subjects):
    # Get subject name
    subject = input(f"Subject {i+1} name: ").strip()
    subject_names.append(subject)

    # Get marks with validation
    marks = -1
    while marks < 0 or marks > 100:
        marks_input = input(f"Marks for {subject} (0-100): ")
        if marks_input.replace(".", "").isdigit():
            marks = float(marks_input)
            if marks < 0 or marks > 100:
                print("Marks must be between 0 and 100!")
        else:
            print("Please enter a valid number!")

    subject_marks.append(marks)

# ---- Calculate Results ----
total_marks = sum(subject_marks)
average = total_marks / num_subjects
highest_marks = max(subject_marks)
lowest_marks = min(subject_marks)
```

```python
# Determine overall grade
if average >= 90:
    overall_grade = "A+"
    overall_comment = "Outstanding! Excellent work!"
elif average >= 80:
    overall_grade = "A"
    overall_comment = "Very Good! Keep it up!"
elif average >= 70:
    overall_grade = "B"
    overall_comment = "Good! Room for improvement."
elif average >= 60:
    overall_grade = "C"
    overall_comment = "Average. Work harder!"
elif average >= 50:
    overall_grade = "D"
    overall_comment = "Below average. Needs improvement."
else:
    overall_grade = "F"
    overall_comment = "Failed. Please study harder."

# Determine pass/fail
if average >= 50:
    status = "PASS ✅"
else:
    status = "FAIL ❌"

# ---- Print Report Card ----
print("\n")
print("*" * 50)
print(f"*{'REPORT CARD':^48}*")
print("*" * 50)
print(f"  Student Name : {student_name}")
print(f"  Class        : {student_class}")
print(f"  Status       : {status}")
print("-" * 50)
print(f"  {'SUBJECT':<22} {'MARKS':>8}  {'GRADE':>6}")
print("-" * 50)

# Print each subject using zip()
for subject, marks in zip(subject_names, subject_marks):
    # Grade for individual subject
    if marks >= 90:
        sub_grade = "A+"
    elif marks >= 80:
        sub_grade = "A"
    elif marks >= 70:
        sub_grade = "B"
    elif marks >= 60:
        sub_grade = "C"
    elif marks >= 50:
        sub_grade = "D"
    else:
        sub_grade = "F"

    print(f"  {subject:<22} {marks:>8.1f}  {sub_grade:>6}")

print("-" * 50)
print(f"  {'Total Marks':<22} {total_marks:>8.1f}")
print(f"  {'Average':<22} {average:>8.1f}  {overall_grade:>6}")
print(f"  {'Highest Subject':<22} {highest_marks:>8.1f}")
print(f"  {'Lowest Subject':<22} {lowest_marks:>8.1f}")
print("=" * 50)
print(f"  Comment: {overall_comment}")
print("=" * 50)
```

□ Key Learning Points from Project 2
1. □□□□□□ parallel list (subject_names, subject_marks) — related data □□□□□□□ □□□□ 2. zip(list1, list2) — □□□□□□ list □□□□□□□ loop □□□ 3. sum(), max(), min() — list □□ □□□ built-in operations 4. Nested while inside for — input validation □□ real use case 5. f-string alignment — :<22 □□□□ left-align 22 □□□□□□, :>8 □□□□ right-align 8 □□□□□□

Project 3: Simple Contact Book

Theory — Dictionary □□□ □□□□□□?

□□□□ projects □ □□□□ list □□□□□□□□ □□□□□□□ □□□□□□□ □□□ □□□□ □□□□ □□□□□□□: □□□□ □□□ contact □□ □□□ □□□□□□ □□□ phone number □□□□□□ □□□ □□□□□□ List □ □□□ □□□ □□□□ □□□□ index □□□□□□ □□□□□□□ □□□□ □□ □□□□□□ □□□□ dictionary □□□ □□□□□□□□ □□□□ data structure□

Concept	□□□ □□□□□□□□ □□□□□□□
`Dictionary (dict)`	Key-value pair□ contact_book["Alice"] = "01711111111" — name □□□□□ directly number □□□□□□□ □□□□□
`dict.keys()`	Dictionary □ □□ keys (□□□□□ □□ □□□) □□□□ view □□□□□□□ □□□□□□□□
`dict.values()`	Dictionary □ □□ values (□□□□□ □□ numbers) □□□□□□□□
`dict.items()`	Key-value pair □□□□□□□ iterate □□□ — for name, phone in contacts.items()
`"key" in dict`	O(1) lookup — dict □ □□□□ key □□□ □□□□ check □□□□
`del dict[key]`	Dictionary □□□□ □□□□ entry delete □□□□

```
# ================================================
# PROJECT 3: SIMPLE CONTACT BOOK
# Concepts used: dictionary, while loop, if/elif,
#   for loop, input/output, string methods,
#   in operator, list (for sorted display)
# ================================================

print("=" * 45)
print("      □ SIMPLE CONTACT BOOK")
print("=" * 45)

# Dictionary to store contacts: name -> phone number
```

```python
# We start with some sample contacts
contact_book = {
    "Alice": "01711-111111",
    "Bob": "01811-222222",
    "Charlie": "01911-333333"
}

# Main program loop
while True:
    # Show menu
    print("\n--- MENU ---")
    print("[1] View All Contacts")
    print("[2] Search Contact")
    print("[3] Add New Contact")
    print("[4] Update Contact")
    print("[5] Delete Contact")
    print("[6] Exit")
    print("-" * 20)

    choice = input("Enter your choice (1-6): ").strip()

    # ---- Option 1: View All Contacts ----
    if choice == "1":
        if len(contact_book) == 0:
            print("\n□ Contact book is empty!")
        else:
            print(f"\n□ Total contacts: {len(contact_book)}")
            print("-" * 35)
            print(f"  {'NAME':<20} {'PHONE':<15}")
            print("-" * 35)

            # sorted() to show contacts alphabetically
            for name in sorted(contact_book.keys()):
                phone = contact_book[name]
                print(f"  {name:<20} {phone:<15}")

            print("-" * 35)

    # ---- Option 2: Search Contact ----
    elif choice == "2":
        search_name = input("Enter name to search: ").strip()

        # Check if contact exists (case-insensitive search)
        found = False
        for name in contact_book:
            if name.lower() == search_name.lower():
                print(f"\n✅ Contact Found!")
                print(f"   Name  : {name}")
                print(f"   Phone : {contact_book[name]}")
                found = True
                break

        if not found:
            print(f"\n❌ '{search_name}' not found in contact book.")

    # ---- Option 3: Add New Contact ----
    elif choice == "3":
        new_name = input("Enter contact name: ").strip()
        if new_name == "":
            print("❌ Name cannot be empty!")
        elif new_name in contact_book:
            print(f"❌ '{new_name}' already exists!")
            print(f"   Current number: {contact_book[new_name]}")
        else:
            new_phone = input("Enter phone number: ").strip()
            if new_phone == "":
                print("❌ Phone number cannot be empty!")
            else:
                contact_book[new_name] = new_phone
                print(f"\n✅ Contact added!")
                print(f"   {new_name} -> {new_phone}")
                print(f"   Total contacts now: {len(contact_book)}")

    # ---- Option 4: Update Contact ----
    elif choice == "4":
        update_name = input("Enter name to update: ").strip()

        if update_name in contact_book:
            old_phone = contact_book[update_name]
            print(f"   Current number: {old_phone}")
            new_phone = input("   Enter new number: ").strip()

            if new_phone != "":
                contact_book[update_name] = new_phone
                print(f"\n✅ Updated {update_name}: {old_phone} -> {new_phone}")
            else:
                print("❌ Phone number cannot be empty. Update cancelled.")
        else:
            print(f"\n❌ '{update_name}' not found!")

    # ---- Option 5: Delete Contact ----
    elif choice == "5":
        del_name = input("Enter name to delete: ").strip()

        if del_name in contact_book:
            deleted_phone = contact_book[del_name]
            confirm = input(f"Delete '{del_name}' ({deleted_phone})? (yes/no): ")

            if confirm.lower() == "yes" or confirm.lower() == "y":
                del contact_book[del_name]
                print(f"\n✅ '{del_name}' deleted from contact book.")
                print(f"   Total contacts remaining: {len(contact_book)}")
            else:
                print("❌ Delete cancelled.")
        else:
            print(f"\n❌ '{del_name}' not found in contact book.")

    # ---- Option 6: Exit ----
    elif choice == "6":
        print("\nGoodbye! Your contacts are saved. □")
        break

    else:
        print("\n❌ Invalid choice! Please enter 1-6.")
```

□ Key Learning Points from Project 3
1. Dictionary □□ real-world use — name-to-phone mapping 2. 'key in dict' — O(1) □ existence check 3. del dict[key] — entry remove □□□ 4. dict.keys() □□□□□□ iterate □□□ sorted() — alphabetical order 5. CRUD operations — Create, Read, Update, Delete (programming □□ □□□ □□□□□□□) 6. while True + break = menu-driven program □□ standard pattern

Project 4: Basic Calculator with History

Theory — List as a History Log

□□ project □ □□□□ □□□□ calculator □□□□ □□□□ □□□□ □□□□□□□□ calculation □□ record □□□□□□□ □□□ □□□□□□□ □□□□□□□ list □□ □□□□ 'memory' □□ 'log' □□□□□□□ □□□□□□□□ □□□ □□□□ — real apps □ (browser history, undo/redo, audit logs) □□ same pattern □□□□□□□□ □□□□

Concept	□□□ □□□□□□□□ □□□□□□□
`history list`	□□□□□□□□ calculation string □□□□□□□ list □ □□□□ — □□□□ history.append('5 + 3 = 8')
`Tuple unpacking`	Operations □□ tuple □ □□□□: ("+", "Addition") □□□□□□ unpack □□□□
`while loop`	User 'quit' □□ □□□□□□□ □□□□□□□□ calculator □□□□ □□□□□□□
`ZeroDivisionError`	Division □ 0 □□□□ error □□□ — if divisor == 0 check □□□ □□□□ prevent □□□□
`enumerate(history)`	History □□□□□□□□ □□□□ number □□□□□□ □□□□□□□ enumerate □□□□□□□□□

```
# ================================================
# PROJECT 4: CALCULATOR WITH HISTORY
# Concepts used: variables, while loop, if/elif,
#   list, append, enumerate, tuple, f-strings,
#   input, float(), string methods
# ================================================

print("=" * 45)
print("    □ CALCULATOR WITH HISTORY")
print("=" * 45)
print("Type 'history' to see past calculations.")
print("Type 'clear' to clear history.")
print("Type 'quit' to exit.")
print("=" * 45)

# List to store all calculation history
calculation_history = []

# Main calculator loop
while True:
    print("\nOperations: + | - | * | / | % | **")
    user_input = input("\nEnter expression (e.g. 10 + 5) or command:
").strip().lower()

    # Check for commands
    if user_input == "quit":
        print("\nFinal Summary:")
        print(f"Total calculations done: {len(calculation_history)}")
        print("Goodbye! □")
        break

    elif user_input == "history":
        if len(calculation_history) == 0:
            print("\n□ No calculations yet!")
        else:
            print(f"\n□ Calculation History ({len(calculation_history)}
records):")
            print("-" * 40)
            for index, record in enumerate(calculation_history, start=1):
                print(f"  {index}. {record}")
            print("-" * 40)
        continue

    elif user_input == "clear":
        calculation_history = []    # reset list to empty
        print("\n□  History cleared!")
        continue

    # Parse the expression: split by spaces
    parts = user_input.split()

    # Validate: must have exactly 3 parts: number operator number
    if len(parts) != 3:
        print("\n❌ Invalid format! Use: number operator number")
        print("   Example: 10 + 5   or   20 / 4")
        continue

    num1_str = parts[0]
    operator = parts[1]
    num2_str = parts[2]

    # Validate numbers
    valid_num1 = True
    valid_num2 = True

    try:
        num1 = float(num1_str)
    except ValueError:
        valid_num1 = False

    try:
        num2 = float(num2_str)
    except ValueError:
        valid_num2 = False

    if not valid_num1 or not valid_num2:
        print("\n❌ Please enter valid numbers!")
        continue
    # Validate operator
    valid_operators = ["+", "-", "*", "/", "%", "**"]
    if operator not in valid_operators:
        print(f"\n❌ Invalid operator '{operator}'")
        print(f"   Valid operators: {valid_operators}")
        continue

    # Perform calculation
    result = 0
    error_msg = ""
    calculation_ok = True

    if operator == "+":
        result = num1 + num2
        op_name = "Addition"
    elif operator == "-":
        result = num1 - num2
        op_name = "Subtraction"
    elif operator == "*":
        result = num1 * num2
        op_name = "Multiplication"
    elif operator == "/":
        if num2 == 0:
            error_msg = "❌ Cannot divide by zero!"
            calculation_ok = False
        else:
            result = num1 / num2
            op_name = "Division"
    elif operator == "%":
        if num2 == 0:
            error_msg = "❌ Cannot use modulo with zero!"
            calculation_ok = False
        else:
            result = num1 % num2
            op_name = "Modulo"
    elif operator == "**":
        result = num1 ** num2
        op_name = "Power"

    if not calculation_ok:
        print(f"\n{error_msg}")
        continue

    # Display result
    # Show as integer if result is whole number
    if result == int(result):
        result_display = int(result)
    else:
        result_display = round(result, 4)

    print(f"\n{'=' * 30}")
    print(f"  {op_name}")
    print(f"  {num1} {operator} {num2} = {result_display}")
    print(f"{'=' * 30}")

    # Save to history
    history_entry = f"{num1} {operator} {num2} = {result_display}"
    calculation_history.append(history_entry)
    print(f"  (Saved to history #{len(calculation_history)})")
```

□ Key Learning Points from Project 4

1. List as log — calculation_history list □ records store □□□ 2. .split() □□□□□ string parse □□□ — user input □□□□ parts □□□ □□□ 3. try/except □□□□□ invalid input handle — ValueError catch □□□ 4. enumerate(list, start=1) — 1 □□□□ numbered list □□□□□□ 5. list = [] □□□□□ reset — clear history feature 6. continue — invalid input □ □□□□ code skip □□□ loop □□ □□□□□□ □□□□ □□□□□□

SECTION 10: Top Interview Questions & Answers

These are the most commonly asked Python interview questions for freshers and junior developers, covering all Day 1 topics.

Variables, Data Types & Operators

Interview Q: What are Python's built-in data types?
Answer: Numeric: int, float, complex. Text: str. Boolean: bool. Sequence: list, tuple, range. Mapping: dict. Set: set, frozenset. Binary: bytes, bytearray. None: NoneType.

Interview Q: What is the difference between mutable and immutable types?
Answer: Mutable objects CAN be changed after creation: list, dict, set, bytearray. Immutable objects CANNOT be changed: int, float, str, tuple, frozenset, bytes. When you 'modify' an immutable type, Python creates a new object. This distinction matters for performance and for safe use as dictionary keys (only hashable/immutable types can be keys).

Interview Q: What is the output of: print(type(1/1))?
Answer: The output is <class 'float'>. In Python 3, the division operator / always returns a float, even when dividing two integers. 1/1 = 1.0. To get integer division, use // operator: 1//1 = 1 (int).

Interview Q: How does Python handle integer overflow?
Answer: Python 3 integers have arbitrary precision — they never overflow! Python automatically allocates more memory as needed. You can compute 2**1000 and get the full result. This differs from C/Java where integers have fixed sizes (32 or 64 bits) and can overflow.

Interview Q: What is the output of: print(0.1 + 0.2 == 0.3)?
Answer: False. This is a classic floating-point precision problem. 0.1 + 0.2 equals 0.30000000000000004 in binary floating point. To compare floats, use: abs(0.1 + 0.2 - 0.3) < 1e-9 or use the math.isclose() function or the decimal module for exact decimal arithmetic.

Strings

Interview Q: How do you check if a string is a palindrome?
Answer: s == s[::-1] is the most Pythonic way. For case-insensitive: s.lower() == s.lower()[::-1]. Complete solution: def is_palindrome(s): return s.lower() == s.lower()[::-1]. Example: is_palindrome('racecar') = True, is_palindrome('hello') = False.

Interview Q: What is the difference between 'in' operator for strings and lists?
Answer: For strings, 'in' checks for substring: 'ell' in 'hello' = True. For lists, 'in' checks for element equality: 2 in [1,2,3] = True. For large lists, 'in' is O(n) — must check each element. For sets and dict keys, 'in' is O(1) — uses hash table. When checking membership frequently, use a set.

Interview Q: How do you count occurrences of each character in a string?
Answer: Method 1 (basic): use a dict and loop. Method 2 (Pythonic): from collections import Counter; counts = Counter('hello world'). Method 3: counts = {char: s.count(char) for char in set(s)}. Counter is fastest and most readable.

Conditionals & Loops

Interview Q: What is the difference between a for loop and a while loop?
Answer: Use a for loop when you know the number of iterations in advance or are iterating over a sequence (list, string, range). Use a while loop when you continue until a condition becomes False and don't know how many iterations are needed. for loops are generally safer (no infinite loop risk). while loops are used for event-driven programming, input validation, and game loops.

Interview Q: How do you iterate over a list with both index and value?
Answer: Use enumerate(): for i, val in enumerate(my_list): print(i, val). You can set the start index: for i, val in enumerate(my_list, start=1). Avoid using range(len(my_list)) — it's less Pythonic and less readable.

Interview Q: What does the else clause do in a for/while loop?
Answer: The else block runs ONLY if the loop completed normally (wasn't terminated by break). This is useful for the 'search and not found' pattern: loop through items, break if found, else (not found) run the else block. It replaces the need for a 'found' flag variable.

Lists & Tuples

Interview Q: What is the difference between shallow copy and deep copy?
Answer: Shallow copy (list.copy() or list[:]) creates a new list but nested objects are still references to the same objects. Deep copy (import copy; copy.deepcopy(lst)) creates completely independent copies of all nested objects. For flat lists with primitives, shallow copy is sufficient. For nested lists/dicts, use deep copy to avoid unintended mutations.

Interview Q: How do you remove duplicates from a list while preserving order?
Answer: seen = set(); result = [x for x in lst if not (x in seen or seen.add(x))]. Or in Python 3.7+: result = list(dict.fromkeys(lst)) — dicts preserve insertion order. Simply using list(set(lst)) removes duplicates but loses order.

Interview Q: What is the time complexity of common list operations?
Answer: Append: O(1) amortized. Insert at index 0: O(n). Access by index: O(1). Search (in operator): O(n). Remove by value: O(n). Sort: O(n log n). len(): O(1). pop() from end: O(1). pop(0) from front: O(n) — use collections.deque for O(1) front operations.

Quick Reference: What to memorize
1. Python types: int, float, str, bool, list, tuple, dict, set, None 2. Truthiness: 0, '', [], {}, (), None, False are falsy — everything else truthy 3. List methods: append, insert, extend, remove, pop, sort, reverse, count, index 4. String methods: split, join, strip, replace, find, upper, lower, format 5. range(start, stop, step) — stop is EXCLUSIVE 6. enumerate(iterable, start=0) — gives (index, value) pairs 7. zip(list1, list2) — iterates two lists in parallel 8. Mutable: list, dict, set | Immutable: int, float, str, tuple

SECTION 11: Practice Exercises

Practice these exercises to solidify your Day 1 understanding. Solutions are provided below each problem.

Beginner Level

Exercise 1: Temperature Converter

```python
# Write a program that converts temperature between Celsius and Fahrenheit.
# Formulas: F = (C × 9/5) + 32 | C = (F - 32) × 5/9

# SOLUTION:
def celsius_to_fahrenheit(c):
    return (c * 9/5) + 32

def fahrenheit_to_celsius(f):
    return (f - 32) * 5/9

temp = float(input("Enter temperature: "))
unit = input("Is it C or F? ").upper()

if unit == "C":
    print(f"{temp}°C = {celsius_to_fahrenheit(temp):.2f}°F")
elif unit == "F":
    print(f"{temp}°F = {fahrenheit_to_celsius(temp):.2f}°C")
else:
    print("Invalid unit!")
```

Exercise 2: FizzBuzz

```python
# Classic interview problem: Print 1 to 100.
# For multiples of 3, print "Fizz"
# For multiples of 5, print "Buzz"
# For multiples of both, print "FizzBuzz"

# SOLUTION:
for i in range(1, 101):
    if i % 15 == 0:       # check 15 FIRST (or check both)
        print("FizzBuzz")
    elif i % 3 == 0:
        print("Fizz")
    elif i % 5 == 0:
        print("Buzz")
    else:
        print(i)

# One-liner version:
```

```
print(['FizzBuzz' if i%15==0 else 'Fizz' if i%3==0 else 'Buzz' if i%5==0 else i
for i in range(1,101)])
```

Exercise 3: Palindrome Checker

```
# Check if a word or phrase is a palindrome (ignoring spaces and case)

# SOLUTION:
def is_palindrome(text):
    # Remove spaces and convert to lowercase
    cleaned = text.replace(" ", "").lower()
    return cleaned == cleaned[::-1]

# Test cases
test_cases = ["racecar", "hello", "A man a plan a canal Panama", "Was it a car or
a cat I saw"]
for test in test_cases:
    result = "✅Palindrome" if is_palindrome(test) else "❌Not palindrome"
    print(f'"{test}" -> {result}')
```

Intermediate Level

Exercise 4: Prime Number Checker

```
# Find all prime numbers up to N using the Sieve of Eratosthenes

# SOLUTION:
def find_primes(n):
    if n < 2:
        return []
    sieve = [True] * (n + 1)
    sieve[0] = sieve[1] = False
    for i in range(2, int(n**0.5) + 1):
        if sieve[i]:
            for j in range(i*i, n+1, i):
                sieve[j] = False
    return [i for i in range(n+1) if sieve[i]]

n = int(input("Find primes up to: "))
primes = find_primes(n)
print(f"Found {len(primes)} primes: {primes}")
```

Exercise 5: Word Frequency Counter

```
# Count the frequency of each word in a sentence

# SOLUTION:
sentence = input("Enter a sentence: ").lower()
words = sentence.split()
```

```python
# Count frequencies using a dictionary
freq = {}
for word in words:
    word = word.strip(".,!?;:")  # remove punctuation
    freq[word] = freq.get(word, 0) + 1

# Sort by frequency (highest first)
sorted_freq = sorted(freq.items(), key=lambda x: x[1], reverse=True)

print("\nWord Frequencies:")
for word, count in sorted_freq:
    print(f"  {word}: {count} {'time' if count == 1 else 'times'}")
```

Advanced Challenge

Exercise 6: Caesar Cipher

```python
# Implement Caesar cipher encryption/decryption
# Caesar cipher shifts each letter by N positions

# SOLUTION:
def caesar_cipher(text, shift, mode="encrypt"):
    if mode == "decrypt":
        shift = -shift

    result = []
    for char in text:
        if char.isalpha():
            # Determine the base (uppercase or lowercase)
            base = ord('A') if char.isupper() else ord('a')
            # Shift the character
            shifted = (ord(char) - base + shift) % 26 + base
            result.append(chr(shifted))
        else:
            result.append(char)  # non-alpha chars unchanged

    return ''.join(result)

# Test
message = "Hello, World! This is Python."
shift = 13   # ROT13 is a well-known Caesar cipher with shift=13

encrypted = caesar_cipher(message, shift)
decrypted = caesar_cipher(encrypted, shift, "decrypt")

print(f"Original:  {message}")
print(f"Encrypted: {encrypted}")
print(f"Decrypted: {decrypted}")
print(f"Match: {message == decrypted}")
```