My Python Course Notes

Structured Revision for Every Lesson

Siddharth Patel
HTW Berlin

June 12, 2025

Contents

1 Lesson 1: Print Function – Full Usage Guide

```python
# PRINT FUNCTION - FULL USAGE GUIDE

# Basic Syntax:
# print(*objects, sep=' ', end='\n', file=sys.stdout, flush=False)

# Parameters:
# *objects → One or more objects to be printed (comma-separated).
# sep      → String inserted between objects. Default is ' ' (space).
# end      → String appended after the last object. Default is '\n' (new line).
# file     → A file-like object (stream); default is sys.stdout.
# flush    → If True, forcibly flush the stream. Default is False.

# ------------------------------------------

# 1. Basic print
print("Hello, World!")  # Hello, World!

# 2. Printing multiple objects
print("Hello", "Python", 3)  # Hello Python 3

# 3. Using 'sep' to change separator
print("2025", "05", "27", sep="-")  # 2025-05-27

# 4. Using 'end' to avoid new line
print("Loading", end="...")  # Loading...

# 5. Using custom separator and end together
print("Name", "Age", sep=": ", end=" years\n")  # Name: Age years

# 6. Printing to a file
with open("output.txt", "w") as f:
    print("Saving this line to a file.", file=f)

# 7. Forcing flush (useful in loops/real-time output)
import time
for i in range(3):
    print(i, end=" ", flush=True)
    time.sleep(0.5)  # Output appears immediately

# 8. Printing escape characters
print("Line1\nLine2")          # New line
print("Tabbed\tSpace")         # Tab space
print("He said \"hello\"")     # Quotes inside string

# 9. Printing with formatted strings (f-strings)
name = "Siddhart"
age = 21
print(f"Hello, my name is {name} and I am {age} years old.")

# 10. Using print with unpacking
nums = [1, 2, 3, 4]
print(*nums)                # 1 2 3 4
print(*nums, sep=", ")      # 1, 2, 3, 4

# 11. Printing Unicode/emojis (note: removed for LaTeX safety)
print("Python is fun")
```

Additional Functions Used in This Lesson

Referenced Functions – Syntax and Output Type

Function	Syntax	Return / Output Type
with open()	`with open("file.txt", "w") as f:`	File object
print(..., file=f)	`print("text", file=f)`	Writes to file, returns None
range()	`range(3)` or `range(start, stop, step)`	Range object (iterable)
time.sleep()	`time.sleep(seconds)`	None (pauses execution)

2   Lesson 2: Input Function – Full Usage Guide

```python
# INPUT FUNCTION - FULL USAGE GUIDE

# Basic Syntax:
# input(prompt='')

# Parameters:
# prompt → A string, written to standard output without a trailing newline,
#          to ask the user for input. Default is an empty string ''.
# Returns → A string entered by the user (always str type).
# Notes   → Always returns a string. You need to convert it using int(), float(), etc. if needed.

# -----------------------------------------

# 1. Basic usage with no prompt
user_input = input()
print("You entered:", user_input)

# 2. Input with a prompt
name = input("Enter your name: ")
print("Hello,", name)

# 3. Converting input to integer
age = int(input("Enter your age: "))
print("You will be", age + 1, "next year.")

# 4. Converting input to float
height = float(input("Enter your height in meters: "))
print("Your height in cm is", height * 100)

# 5. Reading multiple values (as strings)
x, y = input("Enter two words separated by space: ").split()
print("Word 1:", x)
print("Word 2:", y)

# 6. Reading and converting multiple values to int
a, b = map(int, input("Enter two integers: ").split())
print("Sum =", a + b)

# 7. Reading many values into a list of ints
numbers = list(map(int, input("Enter multiple numbers: ").split()))
print("You entered:", numbers)
```

```
42
43   # 8. Handling invalid input using try/except
44   try:
45       salary = float(input("Enter your monthly salary: "))
46       print("Yearly salary:", salary * 12)
47   except ValueError:
48       print("Invalid input! Please enter a number.")
49
50
```

Referenced Functions – Syntax and Output Type

Function / Statement	Syntax		Return / Output Type
`.split()`	`string.split()` `string.split("delimiter")`	or	List of strings
`map()`	`map(function, iterable)`		Map object (can be converted to list)
`list()`	`list(iterable)`		List object
`try / except`	`try:` `code` `except ErrorType:` `fallback`		Flow control – no return value; handles runtime errors

3   Lesson 3: Math Operators – Full Usage Guide

```
1    # MATH OPERATORS – FULL USAGE GUIDE
2
3    # Basic Syntax:
4    # <operand1> <operator> <operand2>
5
6    # Operators:
7    # +   Addition              →  a + b
8    # -   Subtraction           →  a - b
9    # *   Multiplication        →  a * b
10   # /   Division              →  a / b
11   # //  Floor Division        →  a // b
12   # %   Modulus (Remainder)   →  a % b
13   # **  Exponentiation        →  a ** b
14
15   # ----------------------------------------
16
17   # 1. Addition
18   print("1 + 1 =", 1 + 1)
19
20   # 2. Subtraction
21   print("2 - 3 =", 2 - 3)
22
23   # 3. Multiplication
24   print("4 * 5 =", 4 * 5)
25
26   # 4. Division (always returns float)
```

```python
27  print("6 / 3 =", 6 / 3)
28
29  # 5. Floor Division (truncates decimals)
30  print("7 // 2 =", 7 // 2)
31
32  # 6. Rounded division result using round()
33  number1 = 1.85
34  number2 = 1.35
35  number3 = 1.5
36  print(f"{number1} rounded is:", round(number1))  # 2
37  print(f"{number2} rounded is:", round(number2))  # 1
38  print(f"{number3} rounded is:", round(number3))  # 2
39
40  # 7. Exponentiation
41  print("3 ** 3 =", 3 ** 3)  # 27
42
43  # 8. Modulus (Remainder)
44  print("20 / 6 =", 20 / 6)      # Division
45  print("20 % 6 =", 20 % 6)      # Remainder (2)
46
47  # 9. Operator Precedence in Python:
48  # 1. ()
49  # 2. **
50  # 3. * and /
51  # 4. + and -
52  # Evaluated left to right within same level
```

4   Lesson 4: Strings – Full Usage Guide

```python
1   # STRINGS – FULL USAGE GUIDE
2
3   # Basic Explanation:
4   # A string is a sequence of characters enclosed in single (' ') or double (" ") quotes.
5   # Strings are immutable in Python.
6
7   # -------------------------------------------
8   # 1. Creating Strings
9   name = 'math'     # single-quoted string
10  subject = "math"  # double-quoted string
11
12  # 2. String Addition and Printing
13  print("math" + "works")     # mathworks
14  print("math", "works")      # math works
15
16  # 3. String Multiplication
17  string1 = "hello"
18  string2 = "world"
19  number = 5
20
21  print(string1, string2)     # hello world
22  print(string1 + string2)    # helloworld
23  print(string1 * number)     # hellohellohellohellohello
24
25  # 4. Invalid Concatenation Example
26  # print(string1 + number)   # TypeError: can only concatenate str (not "int")
27
28  # STRING METHODS – TOP 10 DEFINITIONS
29
```

```python
text = "hello WORLD"

# 5. capitalize()
# Returns string with first character uppercased, rest lowercased.
print(text.capitalize())  # Hello world

# 6. lower()
# Converts all characters to lowercase.
print(text.lower())       # hello world

# 7. title()
# Capitalizes first letter of each word.
print(text.title())       # Hello World

# 8. casefold()
# Aggressive lowercase, suitable for comparisons.
text2 = "Straße"
print(text2.casefold())   # strasse

# 9. upper()
# Converts all characters to uppercase.
print(text.upper())       # HELLO WORLD

# 10. count()
# Counts how many times a substring appears.
print(text.count("l"))          # 3
print(text.count("l", 3, 6))    # 1

# 11. find()
# Finds index of substring, or -1 if not found.
print(text.find("WORLD"))       # 6
print(text.find("not_here"))    # -1

# 12. replace()
# Replaces substring with another.
print(text.replace("WORLD", "Python"))     # hello Python
print(text.replace("l", "X", 2))           # heXXo WORLD

# 13. swapcase()
# Swaps uppercase to lowercase and vice versa.
print("Hello World".swapcase())  # hELLO wORLD

# 14. join()
# Joins elements of iterable with separator.
words = ["hello", "world"]
print("-".join(words))           # hello-world
```

Referenced Methods – Syntax and Output Type

Method / Function	Syntax	Return / Output Type
`.capitalize()`	`str.capitalize()`	str
`.lower()`	`str.lower()`	str
`.title()`	`str.title()`	str
`.casefold()`	`str.casefold()`	str
`.upper()`	`str.upper()`	str
`.count()`	`str.count(substring, start, end)`	int
`.find()`	`str.find(substring, start, end)`	int
`.replace()`	`str.replace(old, new, count)`	str
`.swapcase()`	`str.swapcase()`	str
`.join()`	`"separator".join(iterable)`	str

5   Lesson 5: If, Else, and Conditional Operators

```python
# IF / ELSE / ELIF — FULL USAGE GUIDE

# Basic Syntax:
# if condition:
#     block of code
# elif another_condition:
#     another block
# else:
#     fallback block


# Conditional Operators:
# ==   → Equal to                  → (x == y)
# !=   → Not equal to              → (x != y)
# <    → Less than                 → (x < y)
# <=   → Less than or equal to     → (x <= y)
# >    → Greater than              → (x > y)
# >=   → Greater than or equal to  → (x >= y)

# Logical Operators:
# and  → True if both are True       → (x > 5 and x < 10)
# or   → True if at least one is True → (x > 5 or x < 3)
# not  → Inverts the truth value    → not (x > 5)

# -----------------------------------------

# 1. Simple if statement
x = 10
if x > 5:
    print("x is greater than 5")

# 2. if-else statement
if x % 2 == 0:
    print("x is even")
else:
    print("x is odd")

# 3. if-elif-else ladder
grade = 85
if grade >= 90:
    print("Grade: A")
```

```python
41  elif grade >= 80:
42      print("Grade: B")
43  elif grade >= 70:
44      print("Grade: C")
45  else:
46      print("Grade: F")
47
48  # 4. Nested if statements
49  number = 42
50  if number > 0:
51      if number % 2 == 0:
52          print("Positive even number")
53      else:
54          print("Positive odd number")
55  else:
56      print("Negative number or zero")
57
58  # 5. Using logical 'and'
59  age = 25
60  if age > 18 and age < 65:
61      print("Adult and working age")
62
63  # 6. Using logical 'or'
64  language = "Python"
65  if language == "Python" or language == "Java":
66      print("Popular programming language")
67
68  # 7. Using logical 'not'
69  is_logged_in = False
70  if not is_logged_in:
71      print("User not logged in")
72
73  # 8. Short form if-else (Ternary Expression)
74  # → Python provides a one-line shorthand for simple if-else statements.
75  # → Syntax: value_if_true if condition else value_if_false
76  # → Returns: One of two values based on the boolean result of the condition.
77
78  value = 8
79
80  # Traditional if-else version:
81  if value % 2 == 0:
82      result = "Even"
83  else:
84      result = "Odd"
85
86  print("Traditional form:", result)  # Even
87
88  # Shortened using ternary expression:
89  result = "Even" if value % 2 == 0 else "Odd"
90  print("Ternary form:", result)      # Even
91
```

Referenced Operators – Syntax and Output Type

Operator	Syntax	Return / Output Type
== (Equal)	x == y	bool
!= (Not Equal)	x != y	bool
< (Less Than)	x < y	bool
<= (Less Than or Equal)	x <= y	bool
> (Greater Than)	x > y	bool
>= (Greater Than or Equal)	x >= y	bool
and (Logical AND)	x > 5 and x < 10	bool
or (Logical OR)	x < 5 or x > 10	bool
not (Logical NOT)	not (x > 5)	bool
Ternary Expression	value1 if condition else value2	Result of value1 or value2

6 Lesson 6: While Loop – Full Usage Guide

```python
# WHILE LOOP – FULL USAGE GUIDE

# Basic Syntax:
# while condition:
#     block of code

# The code inside the loop runs repeatedly as long as the condition is True.

# -------------------------------------------

# 1. Basic while loop
counter = 0
while counter < 5:
    print("Counter is:", counter)
    counter += 1  # same as: counter = counter + 1

# 2. Using break to exit loop early
i = 0
while True:
    if i == 3:
        print("Breaking at", i)
        break
    print(i)
    i += 1

# 3. Using continue to skip to next iteration
x = 0
while x < 5:
    x += 1
    if x == 3:
        continue  # skips printing 3
    print("x =", x)

# 4. while loop with else block
z = 0
while z < 3:
```

```
37      print("z =", z)
38      z += 1
39  else:
40      print("Loop ended normally (no break)")
41
42  # 5. Infinite loop (be careful!)
43  # while True:
44  #     print("This runs forever")
45
46  # 6. Compound condition
47  n = 0
48  while n < 10 and n != 7:
49      print(n)
50      n += 2
51
52  # -----------------------------------------
53  # Counter update operators
54
55  # → counter += 1   → same as counter = counter + 1
56  # → counter -= 1   → same as counter = counter - 1
57  # → counter *= 2   → same as counter = counter * 2
58  # → counter /= 2   → same as counter = counter / 2
59
60  # Note:
61  # Python does NOT support the ++ or -- operators like other languages.
62  # Using x++ or x-- will cause a SyntaxError.
```

Referenced Keywords and Operators – Syntax and Output Type

Keyword / Operator	Syntax	Return / Effect
while	while condition:	Repeats block while condition is True
break	break	Immediately exits the nearest enclosing loop
continue	continue	Skips current iteration and continues with the next
+=	x += y	Updates: x = x + y
-=	x -= y	Updates: x = x - y
*=	x *= y	Updates: x = x * y
/=	x /= y	Updates: x = x / y
++ / --	Not supported in Python	Causes SyntaxError

7   Lesson 7: For Loop – Full Usage Guide

```python
# FOR LOOP – FULL USAGE GUIDE

# Basic Syntax:
# for variable in iterable:
#     block of code

# -------------------------------------------
# 1. Using range()
# range(stop) → from 0 to stop-1
# range(start, stop) → from start to stop-1
# range(start, stop, step)

for i in range(5):
    print("i =", i)

for i in range(2, 6):
    print("From 2 to 5:", i)

for i in range(10, 0, -2):
    print("Countdown by 2:", i)

# -------------------------------------------
# 2. Iterating over a list
fruits = ["apple", "banana", "cherry"]
for fruit in fruits:
    print("Fruit:", fruit)

# 3. Iterating over a string
text = "hello"
for char in text:
    print("Char:", char)

# 4. Iterating over a tuple
# A tuple is an ordered, immutable collection.
# You can iterate through it just like a list.
coordinates = (10, 20, 30)
for value in coordinates:
    print("Value:", value)


# 5. Iterating over a set
# A set is an unordered collection of unique elements.
# Iteration works, but order is not guaranteed.
unique_numbers = {1, 2, 3}
for num in unique_numbers:
    print("Unique:", num)


# 6. Iterating over a dictionary
# A dictionary stores key-value pairs.
# Iterating over it by default gives you the keys.
person = {"name": "Alice", "age": 25}
for key in person:
    print("Key:", key, "| Value:", person[key])

# 7. Iterating with .items()
# .items() returns a list of (key, value) pairs from a dictionary.
# Useful when you need both key and value at once.
for key, value in person.items():
    print(f"{key} => {value}")
```

```
61
62  # 8. Using enumerate()
63  # enumerate() gives both the index and the item during iteration.
64  colors = ["red", "green", "blue"]
65  for index, color in enumerate(colors):
66      print(f"{index}: {color}")
67
68
69  # 9. Using break
70  for n in range(5):
71      if n == 3:
72          break
73      print("Breaking loop at:", n)
74
75  # 10. Using continue
76  for n in range(5):
77      if n == 2:
78          continue
79      print("Continuing:", n)
80
81  # 11. Using else with for
82  for n in range(3):
83      print(n)
84  else:
85      print("Loop completed without break.")
86
87  # -------------------------------------------
88  # range() - Recap:
89  # range(stop)
90  # range(start, stop)
91  # range(start, stop, step)
92  # returns a range object which is an iterable of numbers
```

Referenced Iteration Helpers – Syntax and Output Type

Function / Method	Syntax	Return / Output Type
tuple	(a, b, c)	Iterable (ordered, immutable)
set	{a, b, c}	Iterable (unordered, unique)
dict.items()	dict.items()	Iterable of (key, value) pairs
enumerate()	enumerate(iterable)	Iterable of (index, item) tuples

8   Lesson 8: Functions – Full Usage Guide

```
1  # FUNCTIONS - FULL USAGE GUIDE
2
3  # Basic Syntax:
4  # def function_name(parameters):
```

```python
5    #      block of code
6
7    # --------------------------------------------
8    # 1. Defining a basic function
9    def greet():
10       print("Hello from the function!")
11
12   # Call the function AFTER defining it
13   greet()
14
15   # --------------------------------------------
16   # 2. DOs and DON'Ts
17   # Don't call a function before it's defined:
18   # greet()  # This would raise NameError if called before definition
19
20   # Always define before calling:
21   def welcome():
22       print("Welcome to Python!")
23
24   welcome()
25
26   # --------------------------------------------
27   # 3. Function that performs a task (prints something)
28   def print_sum(a, b):
29       print("The sum is:", a + b)
30
31   print_sum(4, 5)  # Output: The sum is: 9
32
33   # --------------------------------------------
34   # 4. Function that calculates and returns a value
35   def get_sum(a, b):
36       return a + b
37
38   result = get_sum(10, 20)
39   print("Returned sum:", result)
40
41   # --------------------------------------------
42   # 5. Local vs Global Variables
43
44   # Global variable
45   counter = 100
46
47   def increase_counter():
48       # Local variable (does not affect global counter)
49       counter = 0
50       counter += 1
51       print("Local counter:", counter)
52
53   increase_counter()
54   print("Global counter remains:", counter)
55
56   # --------------------------------------------
57   # 6. Using 'global' keyword to modify global variable
58   count = 0
59
60   def modify_global():
61       global count
62       count += 1
63       print("Modified global count:", count)
64
65   modify_global()
66   print("Global count after function:", count)
```

9   Lesson 9: Lists – Introduction

```python
1   # LISTS - INTRODUCTION
2
3   # What is a list?
4   # A list is a built-in data structure in Python that stores an ordered collection of items.
5   # Lists are mutable, meaning you can change their contents after creation.
6   # Defined using square brackets: []
7
8   # ------------------------------------------
9   # 1. Defining a simple list (homogeneous)
10  fruits = ["apple", "banana", "cherry"]
11  print("Fruits:", fruits)
12
13  # 2. List of numbers
14  numbers = [1, 2, 3, 4, 5]
15  print("Numbers:", numbers)
16
17  # 3. List of mixed data types
18  mixed = ["hello", 42, 3.14, True]
19  print("Mixed list:", mixed)
20
21  # 4. Empty list
22  empty = []
23  print("Empty list:", empty)
24
25  # 5. Nested list (list inside a list)
26  matrix = [[1, 2], [3, 4]]
27  print("Nested list:", matrix)
28
29  # 6. List using list() constructor
30  from_string = list("hello")
31  print("List from string:", from_string)  # ['h', 'e', 'l', 'l', 'o']
32
33  # 7. List from range()
34  range_list = list(range(5))
35  print("List from range():", range_list)  # [0, 1, 2, 3, 4]
```

List Types and Creation Methods

List Type	Syntax / Example	Notes
Homogeneous List	[1, 2, 3, 4]	All elements of the same type
String List	["apple", "banana"]	List of strings
Mixed-Type List	["hi", 42, 3.14, True]	Supports multiple data types
Empty List	[]	No elements yet
Nested List	[[1, 2], [3, 4]]	List inside a list
From String	list("abc")	Converts string into list of characters
From Range	list(range(5))	Converts range object into list

10   Lesson 10: List Methods – General Usage

```python
# LIST METHODS - GENERAL PURPOSE

# Lists are mutable and support many built-in methods for adding, removing, searching, and modifying.

# 1. append()
# → Adds a single element to the end
items = ["pen", "book"]
items.append("pencil")
print("append():", items)

# 2. insert()
# → Inserts an element at a specific index
items.insert(1, "eraser")
print("insert():", items)

# 3. remove()
# → Removes the first matching value
items.remove("book")
print("remove():", items)

# 4. pop()
# → Removes and returns element at index (default = last)
removed = items.pop()
print("pop():", removed)
print("After pop:", items)

# 5. clear()
# → Empties the list
temp = ["a", "b"]
temp.clear()
print("clear():", temp)

# 6. copy()
# → Returns a shallow copy
original = ["x", "y", "z"]
cloned = original.copy()
print("copy():", cloned)

# 7. extend()
# → Adds multiple elements from another iterable
tools = ["pen", "pencil"]
tools.extend(["marker", "sharpener"])
print("extend():", tools)

# 8. index()  [Use with care]
# → Finds index of first match; raises error if not found
names = ["Alice", "Bob", "Charlie"]
try:
    idx = names.index("Bob")
    print("index():", idx)
except ValueError:
    print("Name not found")

# Alternative using 'in'
print("Eve" in names)  # False

# 9. count()
# → Counts number of times an item appears
letters = ["a", "b", "a", "c", "a"]
print("count():", letters.count("a"))  # 3
```

```
61
62  # 10. reverse()
63  # → Reverses the list in-place
64  words = ["start", "middle", "end"]
65  words.reverse()
66  print("reverse():", words)
67
68  # 11. sort() [Only works if items are comparable]
69  # → Sorts the list (only if all items can be compared)
70  languages = ["python", "c", "java"]
71  languages.sort()
72  print("sort():", languages)
```

Common List Methods – Overview Table

Method	Description	Return Value
append(x)	Add element x to the end	None
insert(i,x)	Insert x at index i	None
remove(x)	Remove first occurrence of x	None
pop(i)	Remove and return item at index i (last by default)	Element
clear()	Remove all items from the list	None
copy()	Return a shallow copy of the list	List copy
extend(iter)	Append elements from iterable	None
index(x)	Return first index of x (error if not found)	Integer
count(x)	Count occurrences of x	Integer
reverse()	Reverse items in-place	None
sort()	Sort the list in-place	None

11   Lesson 11: List Methods – Numeric Lists Only

```
1   # LIST METHODS – NUMERIC LISTS ONLY
2
3   # These methods are especially useful and commonly used with lists that contain only numbers.
4
5   # Numeric list for demonstration
6   numbers = [5, 2, 8, 3, 5, 1, 8]
7
8   # 1. sort()
9   # → Sorts the list in ascending order (in-place)
10  numbers.sort()
11  print("sort():", numbers)  # [1, 2, 3, 5, 5, 8, 8]
12
13  # 2. reverse()
14  # → Reverses the list order in-place
15  numbers.reverse()
16  print("reverse():", numbers)  # [8, 8, 5, 5, 3, 2, 1]
17
18  # 3. count()
19  # → Counts occurrences of a specific number
20  print("count(5):", numbers.count(5))  # 2
```

```
21
22  # 4. max()
23  # → Returns the largest number in the list
24  print("max():", max(numbers))  # 8
25
26  # 5. min()
27  # → Returns the smallest number in the list
28  print("min():", min(numbers))  # 1
29
30  # 6. sum()
31  # → Returns the sum of all elements
32  print("sum():", sum(numbers))  # 32
33
34  # 7. average (manual)
35  # → Average = sum / count
36  average = sum(numbers) / len(numbers)
37  print("Average:", average)  # 4.571...
38
39  # 8. sorted()
40  # → Returns a new sorted list without modifying the original
41  original = [10, 3, 7]
42  sorted_list = sorted(original)
43  print("original:", original)      # [10, 3, 7]
44  print("sorted():", sorted_list)   # [3, 7, 10]
```

List Methods for Numbers – Reference Table

Method	Description	Return Value
sort()	Sort list in ascending order (in-place)	None
reverse()	Reverse the list (in-place)	None
count(x)	Count how many times x appears	Integer
max(lst)	Return maximum element	Element
min(lst)	Return minimum element	Element
sum(lst)	Sum of all list items	Numeric
sorted(lst)	Return a new sorted list	New list
avg = sum()/len()	Compute average (manual)	Float

12   Lesson 12: 2D Lists and Nested Loops

```
1   # 2D LISTS AND NESTED LOOPS — FULL GUIDE
2
3   # A 2D list is a list of lists (matrix/grid style)
4
5   # Defining a 3x3 grid
6   num_grid = [
7       [1, 2, 3],
8       [4, 5, 6],
9       [7, 8, 9]
10  ]
11
12  # ----------------------------------------
13  # 1. Accessing a specific element (row 0, column 0)
```

```
14  print("Top-left element:", num_grid[0][0])  # Output: 1
15
16  # 2. Accessing other positions
17  print("Middle element:", num_grid[1][1])     # Output: 5
18  print("Bottom-right:", num_grid[2][2])        # Output: 9
19
20  # -------------------------------------------
21  # 3. Iterating through 2D list using nested for-loops
22
23  # Outer loop goes over each row
24  for row in num_grid:
25      # Inner loop goes over each column (element in that row)
26      for column in row:
27          print(column, end=" ")
28      print()  # For new line after each row
```

13 Lesson 13: Tuples – Full Usage Guide

```
1   # TUPLES – FULL USAGE GUIDE
2
3   # 1. What is a tuple?
4   # → A tuple is an immutable, ordered collection of items.
5   # → Defined using parentheses: ()
6
7   # 2. Tuple vs List – Key Differences:
8   # → Tuples use ()       → Lists use []
9   # → Tuples are immutable → Lists are mutable (can be changed)
10  # → Tuples have fewer built-in methods
11  # → Tuples are faster and used for fixed data
12
13  # -------------------------------------------
14  # 3. Defining tuples
15
16  empty_tuple = ()
17  single_item = ("apple",)   # Note the comma!
18  fruits = ("apple", "banana", "cherry")
19
20  print("Fruits tuple:", fruits)
21  print("First item:", fruits[0])
22
23  # -------------------------------------------
24  # 4. What can we do with tuples?
25
26  # Tuple concatenation
27  more_fruits = ("mango", "orange")
28  combined = fruits + more_fruits
29  print("Combined tuple:", combined)
30
31  # Tuple unpacking
32  x, y, z = fruits
33  print("Unpacked:", x, y, z)
34
35  # Cannot change a tuple element
36  # fruits[0] = "kiwi"  # TypeError
37
38  # Cannot append or remove items
39  # fruits.append("kiwi")  # AttributeError
40  # fruits.remove("banana")  # AttributeError
```

```
41
42  # You can delete the whole tuple
43  temp = (1, 2, 3)
44  del temp
45  # print(temp)  # NameError if uncommented
46
47  # -------------------------------------------
48  # 5. Built-in functions usable with tuples
49
50  values = (10, 20, 5, 30)
51
52  # 1. len()
53  print("Length:", len(values))  # 4
54
55  # 2. max()
56  print("Maximum:", max(values))   # 30
57
58  # 3. min()
59  print("Minimum:", min(values))   # 5
60
61  # 4. tuple() constructor
62  sample_list = ["x", "y", "z"]
63  converted = tuple(sample_list)
64  print("Converted to tuple:", converted)
```

Tuple Functions – Syntax and Return Type

Function	Syntax	Return Type
len()	len(tuple)	int (length of tuple)
max()	max(tuple)	Largest item from tuple
min()	min(tuple)	Smallest item from tuple
tuple()	tuple(iterable)	A new tuple object

14 Lesson 14: Sets – General Usage Guide

```
1  # SETS — GENERAL USAGE GUIDE
2
3  # 1. What is a set?
4  # → A set is an unordered, unindexed collection with no duplicate elements.
5  # → Defined using curly braces {} or the set() constructor.
6
7  # -------------------------------------------
8  # 2. Creating a set
9
10  # From a list with duplicates
11  values = [2, 3, 3, 4, 5]
12  unique_values = set(values)
13  print("Set from list:", unique_values)  # {2, 3, 4, 5}
14
```

```
15  # Direct set definition
16  colors = {"red", "green", "blue"}
17  print("Set of colors:", colors)
18
19  # Empty set must be created using set()
20  empty_set = set()
21  print("Empty set:", empty_set)
22
23  # ----------------------------------------------
24  # 3. General set methods
25
26  # 1. add()
27  colors.add("yellow")
28  print("After add():", colors)
29
30  # 2. remove()
31  colors.remove("green")   # Raises error if not found
32  print("After remove():", colors)
33
34  # 3. discard()
35  colors.discard("blue")   # Safe: no error if not found
36  print("After discard():", colors)
37
38  # 4. pop()
39  item = colors.pop()   # Removes random item (sets are unordered)
40  print("Popped:", item)
41
42  # 5. clear()
43  numbers = {1, 2, 3}
44  numbers.clear()
45  print("After clear():", numbers)
46
47  # 6. copy()
48  original = {"a", "b"}
49  cloned = original.copy()
50  print("Copy of set:", cloned)
51
52  # 7. update()
53  colors.update({"white", "black"})
54  print("After update():", colors)
55
56  # ----------------------------------------------
57  # Sets do NOT support indexing or duplicate elements.
```

General Set Methods – Syntax and Return Type

Method	Description	Return Type
add(x)	Adds element x to the set	None
remove(x)	Removes x; error if not found	None
discard(x)	Removes x; no error if not found	None
pop()	Removes and returns an arbitrary element	Element
clear()	Removes all items from the set	None
copy()	Returns a shallow copy of the set	Set copy
update(iter)	Adds elements from another iterable	None

15   Lesson 15: Sets – Mathematical Operations

```python
# SETS - MATHEMATICAL OPERATIONS

# We use sets to perform classic mathematical operations like:
# union, intersection, difference, symmetric difference, etc.

a = {1, 2, 3, 4}
b = {3, 4, 5, 6}

# --------------------------------------------
# 1. union() - combines both sets
print("Union:", a.union(b))        # {1, 2, 3, 4, 5, 6}
print("Using | :", a | b)

# 2. update() - adds elements from b to a
a1 = a.copy()
a1.update(b)
print("Update:", a1)

# --------------------------------------------
# 3. intersection() - common elements
print("Intersection:", a.intersection(b))  # {3, 4}
print("Using & :", a & b)

# 4. intersection_update() - keeps only common elements in a
a2 = a.copy()
a2.intersection_update(b)
print("Intersection Update:", a2)

# --------------------------------------------
# 5. difference() - items in a but not in b
print("Difference (a - b):", a.difference(b))  # {1, 2}
print("Using - :", a - b)

# 6. difference_update() - removes items in b from a
a3 = a.copy()
a3.difference_update(b)
print("Difference Update:", a3)

# --------------------------------------------
# 7. symmetric_difference() - elements in a or b but not both
print("Symmetric Difference:", a.symmetric_difference(b))  # {1, 2, 5, 6}
print("Using ^ :", a ^ b)

# 8. symmetric_difference_update() - modifies a to symmetric diff
a4 = a.copy()
a4.symmetric_difference_update(b)
print("Symmetric Difference Update:", a4)

# --------------------------------------------
# 9. issubset()
print("Is a subset of b:", a.issubset(b))  # False

# 10. issuperset()
print("Is a superset of b:", a.issuperset(b))  # False

# 11. isdisjoint()
x = {10, 20}
print("Is disjoint:", a.isdisjoint(x))  # True (no common elements)
```
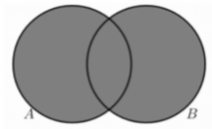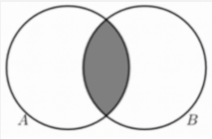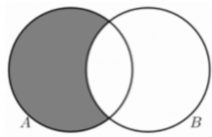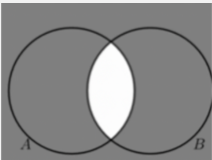
Set Operations – Syntax and Result

Method / Operator	Description	Return Type
`a.union(b)` / `a	b`	All unique elements from both sets	New set
`a.update(b)`	Adds all elements from b to a	None (in-place)
`a.intersection(b)` / `a & b`	Elements common to both sets	New set
`a.intersection_update(`	Keeps only common elements in a	None (in-place)
`a.difference(b)` / `a - b`	Elements in a but not in b	New set
`a.difference_update(b)`	Removes items in b from a	None (in-place)
`a.symmetric_difference` / `a ^ b`	Elements in a or b, not both	New set
`a.symmetric_difference`	Updates a to symmetric diff	None (in-place)

Set Comparison – Membership and Relationship

Method	Description	Returns
`a.issubset(b)`	True if all elements of a are in b	Boolean
`a.issuperset(b)`	True if a contains all elements of b	Boolean
`a.isdisjoint(b)`	True if a and b have no elements in common	Boolean

Set Operations – Visual Explanation

Set Operation	Diagram	Explanation	In Python
Union		$A \cup B$ is the set of all elements that are a member of A, or B, or both.	union()
Intersection		$A \cap B$ is the set of all elements that are a member of both A and B.	intersection()
Difference		$A \setminus B$ is the set of all elements of A that are not in B.	difference()
Symmetric Difference		$A \Delta B$ is the set of elements in either A or B, but not both.	symmetric _difference()

16 Lesson 16: Dictionary

```python
# DICTIONARIES - BASIC USAGE GUIDE

# 1. What is a dictionary?
# → A dictionary is a collection of key-value pairs.
# → Each key must be unique, and values can be of any type.

# Unlike lists or tuples (which store values), dictionaries store data like a "map":
#   Key   →   Value

# ------------------------------------------
# 2. Creating dictionaries

# Method 1: Using curly braces
student = {
    "name": "Alice",
    "age": 21,
    "major": "Computer Science"
}

# Method 2: Using dict() constructor
profile = dict(name="Bob", age=25)
print("Created with dict():", profile)
```

```
23
24  # -------------------------------------------
25  # 3. Accessing values
26
27  print("Name:", student["name"])  # Alice
28  print("Age:", student["age"])    # 21
29
30  # If key doesn't exist, it raises a KeyError
31  # print(student["grade"])  # KeyError
32
33  # -------------------------------------------
34  # 4. Adding new key-value pairs
35
36  student["grade"] = "A"
37  print("Updated student:", student)
38
39  # -------------------------------------------
40  # 5. Iterating over a dictionary (keys by default)
41
42  for key in student:
43      print("Key:", key, "| Value:", student[key])
```

17   Lesson 17: Dictionary – Most Important Methods

```
1  # DICTIONARY METHODS — MOST IMPORTANT & COMMON
2
3  person = {
4      "name": "Alice",
5      "age": 22,
6      "country": "Germany"
7  }
8
9  # -------------------------------------------
10 # 1. get(key[, default])
11 # → Returns the value for the key if it exists; otherwise returns default (or None)
12 print("get('name'):", person.get("name"))        # Alice
13 print("get('gender'):", person.get("gender"))    # None
14 print("get('gender', 'Not specified'):", person.get("gender", "Not specified"))
15
16 # -------------------------------------------
17 # 2. keys()
18 # → Returns a view of all keys
19 print("Keys:", person.keys())  # dict_keys(['name', 'age', 'country'])
20
21 # -------------------------------------------
22 # 3. values()
23 # → Returns a view of all values
24 print("Values:", person.values())  # dict_values(['Alice', 22, 'Germany'])
25
26 # -------------------------------------------
27 # 4. items()
28 # → Returns a view of all key-value pairs as tuples
29 print("Items:", person.items())  # dict_items([('name', 'Alice'), ('age', 22), ...])
30
31 # -------------------------------------------
32 # 5. pop(key)
33 # → Removes a key and returns its value
34 age = person.pop("age")
```

```python
35  print("Popped 'age':", age)
36  print("After pop():", person)
37
38  # -------------------------------------------
39  # 6. popitem()
40  # → Removes and returns the last inserted (key, value) pair
41  last_item = person.popitem()
42  print("Popped last item:", last_item)
43  print("After popitem():", person)
44
45  # -------------------------------------------
46  # 7. update(other_dict)
47  # → Merges another dictionary into current one
48  person.update({"name": "Bob", "gender": "Male"})
49  print("After update():", person)
50
51  # -------------------------------------------
52  # 8. clear()
53  # → Removes all key-value pairs from dictionary
54  temp = {"x": 1, "y": 2}
55  temp.clear()
56  print("After clear():", temp)
57
58  # -------------------------------------------
59  # 9. copy()
60  # → Returns a shallow copy of the dictionary
61  original = {"a": 1, "b": 2}
62  duplicate = original.copy()
63  print("Copy:", duplicate)
```

```python
1   # FAMOUS DICTIONARY EXAMPLES USING: get(), keys(), values(), items()
2
3   student = {
4       "name": "Alice",
5       "age": 21,
6       "major": "Computer Science",
7       "grade": "A"
8   }
9
10  # -------------------------------------------
11  # 1. .get() — Safe value access
12  # → Famous for avoiding KeyError if the key doesn't exist
13
14  # GOOD: get() returns default instead of crashing
15  print("Country:", student.get("country", "Not specified"))  # Not specified
16
17  # BAD: This would raise KeyError
18  # print(student["country"])
19
20  # -------------------------------------------
21  # 2. .keys() — Useful for looping or checking presence
22  print("Keys in student dict:")
23  for key in student.keys():
24      print("-", key)
25
26  # Check if "age" is a key
27  if "age" in student.keys():
28      print("Yes, 'age' is a key.")
29
30  # -------------------------------------------
```

```python
31  # 3. .values() - Check or search values
32  print("Values in student dict:")
33  for value in student.values():
34      print("-", value)
35
36  # Check if a specific value exists
37  if "Computer Science" in student.values():
38      print("Found the major!")
39
40  # --------------------------------------------
41  # 4. .items() - Iterate over both key and value (most common in loops)
42  print("Student Info:")
43  for key, value in student.items():
44      print(f"{key} → {value}")
45
46  # --------------------------------------------
47  # BONUS: Use in condition
48  if "grade" in student:
49      if student["grade"] == "A":
50          print("Excellent student!")
51
52  # --------------------------------------------
53  # 5. Finding Key from Value (reverse lookup)
54
55  # Let's say we want the key for value "A"
56  target_value = "A"
57
58  # Using a loop to search for matching value
59  for key, value in student.items():
60      if value == target_value:
61          print(f"Key for value '{target_value}' is: {key}")
62
63  # --------------------------------------------
64  # 6. Finding Value from Key (already known way)
65  # Just standard access
66  print("Grade is:", student["grade"])  # A
```

Common Dictionary Methods – Reference Table

Method	Description	Return Type
get(key, default)	Returns value if key exists, else default or None	Value or None
keys()	Returns all keys in the dictionary	dict_keys view
values()	Returns all values in the dictionary	dict_values view
items()	Returns key-value pairs as tuples	dict_items view
pop(key)	Removes and returns value for given key	Value
popitem()	Removes and returns the last key-value pair	(key, value) tuple
update(dict)	Updates dict with another dict or key-value pairs	None
clear()	Removes all items from the dictionary	None
copy()	Returns a shallow copy of the dictionary	New dict