My Python Course Notes

Structured Revision for Every Lesson

Siddharth Patel
HTW Berlin

May 28, 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)
```

```python

# 8. Handling invalid input using try/except
try:
    salary = float(input("Enter your monthly salary: "))
    print("Yearly salary:", salary * 12)
except ValueError:
    print("Invalid input! Please enter a number.")


```

Referenced Functions – Syntax and Output Type

Function / Statement	Syntax	Return / Output Type
.split()	string.split() or string.split("delimiter")	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

```python
# MATH OPERATORS - FULL USAGE GUIDE

# Basic Syntax:
# <operand1> <operator> <operand2>

# Operators:
# +   Addition            →  a + b
# -   Subtraction         →  a - b
# *   Multiplication      →  a * b
# /   Division            →  a / b
# //  Floor Division      →  a // b
# %   Modulus (Remainder) →  a % b
# **  Exponentiation      →  a ** b

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

# 1. Addition
print("1 + 1 =", 1 + 1)

# 2. Subtraction
print("2 - 3 =", 2 - 3)

# 3. Multiplication
print("4 * 5 =", 4 * 5)

# 4. Division (always returns float)
```

```python
print("6 / 3 =", 6 / 3)

# 5. Floor Division (truncates decimals)
print("7 // 2 =", 7 // 2)

# 6. Rounded division result using round()
number1 = 1.85
number2 = 1.35
number3 = 1.5
print(f"{number1} rounded is:", round(number1))  # 2
print(f"{number2} rounded is:", round(number2))  # 1
print(f"{number3} rounded is:", round(number3))  # 2

# 7. Exponentiation
print("3 ** 3 =", 3 ** 3)  # 27

# 8. Modulus (Remainder)
print("20 / 6 =", 20 / 6)      # Division
print("20 % 6 =", 20 % 6)      # Remainder (2)

# 9. Operator Precedence in Python:
# 1. ()
# 2. **
# 3. * and /
# 4. + and -
# Evaluated left to right within same level
```

4 Lesson 4: Strings – Full Usage Guide

```python
# STRINGS - FULL USAGE GUIDE

# Basic Explanation:
# A string is a sequence of characters enclosed in single (' ') or double (" ") quotes.
# Strings are immutable in Python.

# --------------------------------------------
# 1. Creating Strings
name = 'math'     # single-quoted string
subject = "math"  # double-quoted string

# 2. String Addition and Printing
print("math" + "works")     # mathworks
print("math", "works")      # math works

# 3. String Multiplication
string1 = "hello"
string2 = "world"
number = 5

print(string1, string2)     # hello world
print(string1 + string2)    # helloworld
print(string1 * number)     # hellohellohellohellohello

# 4. Invalid Concatenation Example
# print(string1 + number)   # TypeError: can only concatenate str (not "int")




# STRING METHODS – TOP 10 DEFINITIONS

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
elif grade >= 80:
    print("Grade: B")
elif grade >= 70:
    print("Grade: C")
else:
    print("Grade: F")

# 4. Nested if statements
number = 42
if number > 0:
    if number % 2 == 0:
        print("Positive even number")
    else:
        print("Positive odd number")
else:
    print("Negative number or zero")

# 5. Using logical 'and'
age = 25
if age > 18 and age < 65:
    print("Adult and working age")

# 6. Using logical 'or'
language = "Python"
if language == "Python" or language == "Java":
    print("Popular programming language")

# 7. Using logical 'not'
is_logged_in = False
if not is_logged_in:
    print("User not logged in")

# 8. Short form if-else (Ternary Expression)
# → Python provides a one-line shorthand for simple if-else statements.
# → Syntax: value_if_true if condition else value_if_false
# → Returns: One of two values based on the boolean result of the condition.

value = 8

# Traditional if-else version:
if value % 2 == 0:
    result = "Even"
else:
    result = "Odd"

print("Traditional form:", result)  # Even

# Shortened using ternary expression:
result = "Even" if value % 2 == 0 else "Odd"
print("Ternary form:", result)      # Even

```

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`