PRACTICE SHEET

What you will learn:

Virtual Environments in Python, Data Types, Geometry with Programming, Complex Numbers, Range and Mathematical Induction, Strings and Their Concepts, Expression Validation and Evaluation

[EASY] Q1) What is the purpose of a virtual environment in Python?
A. To run Python faster
B. To create isolated environments for different projects
C. To update Python version
D. To improve code readability

[EASY] Q2) Which command creates a virtual environment in Python?
A. virtualenv start
B. python -m venv myenv
C. create env myenv
D. venv –create

[EASY] Q3) What will be the output of type({})?
A. <class 'dict'>
B. <class 'set'>
C. <class 'list'>
D. <class 'tuple'>

[EASY] Q4) Fill in the blank so that the output is float:

```
x = 10

y = 4

result = x / y

print(type(____))
```

A. x
B. result
C. x / y
D. int(x / y)

[EASY] Q5) What will be the output?

x = 10

y = 20

print(x is not y and x == y or x < y)

A. True
B. False
C. SyntaxError
D. None

[MEDIUM] Q6) Fill in the blank: What is printed?

a = 256

b = 256

print(a is b)

Now try:

c = 257

d = 257

print(c is d)

Why output is different ?

[MEDIUM] Q7) Given three sides a, b, c of a triangle, write a program to obtain and print the values of three angles rounded to the next integer. Use the formula :

$a^2 = b^2 + c^2 - 2bc \cos A$, $b^2 = a^2 + c^2 - 2ac \cos B$, $c^2 = a^2 + b^2 - 2ab \cos C$

[MEDIUM] Q8) Write a getChessSquareColor() function that has parameters column and row. The function either returns 'black' or 'white' depending on the color at the specified column and row. Chess boards are 8 x 8 spaces in size, and the columns and rows in this program begin at 0 and end at 7 like in Figure. If the arguments for column or row are outside the 0 to 7 range, the function returns a blank string.

Note that chess boards always have a white square in the top left corner.

[MEDIUM] Q9) How will you perform the following operations:

(a) Print imaginary part out of 2 + 3j.
(b) Obtain conjugate of 4 + 2j.
(c) Print decimal equivalent of binary '1100001110'.
(d) Convert a float value 4.33 into a numeric string.
(e) Obtain integer quotient and remainder while dividing 29 with 5.
(f) Obtain hexadecimal equivalent of decimal 34567.
(g) Round-off 45.6782 to second decimal place.
(h) Obtain 4 from 3.556.
(i) Obtain 17 from 16.7844.
(j) Obtain remainder on dividing 3.45 with 1.22.

[MEDIUM] Q10) In English, ordinal numerals have suffixes such as the “th” in 30th or “nd” in 2nd. Write an ordinalSuffix() function with an integer parameter named number and returns a string of the number with its ordinal suffix. For example, ordinalSuffix(42) should return the string '42nd'. You may use Python’s str() function to convert the integer argument to a string.

[HARD] Q11) You are given a range of integers {L, L+1 ,…, R}. An integer X is said to be *reachable* if it can be represented as a sum of two **not necessarily distinct** integers in this range. Find the number of distinct reachable integers.

Input : The first and only line contains two space-separated integers L and R.

Output : Print a single line containing one integer — the number of reachable integers.

Sample 1:

Input : 2 3

Output : 3

Explanation: 4, 5 and 6 are reachable, since 2+2=4, 2+3=5 and 3+3=6.

Try for Input : 3 4 5

Hint1 : Think of Mathematical Induction studied in class 11^{th}.

Hint2 : Important observation about the given question - it has a continuous set of integers from [L to R] - i.e.
L, L+1, L+2, L+3,…, R - all integers in between are a part of this set.

What is the minimal number that you can reach? The answer is $2*L$, since L+L is definitely less or equal than A+B, for all $A \geqq L$ and $B \geqq L$.

Hint 3 : What is the maximal number you can reach? The answer is $2*R$ (the explanation is very similar to the first hint).

Hint 4 : Can you reach all the numbers from 2L to 2R? The answer is yes! Let's prove it by induction. We know that we can reach 2L by L+L. Now let's look at the number X, we know that we can reach X−1 as a sum of some A+B.
First case: B<R, then X=A+(B+1) and we can reach number X.
Second case: B=R and A<R, then we cannot take number B+1 since it's bigger than R, but we can take A+1, then we can reach X as (A+1)+B.
Third case: A=B=R, A+B=2R, so we are finished.

What is the number of integers between 2L and 2R inclusive? Simple: 2R−2L+1

[MEDIUM] Q12) Find-and-replace is a standard feature in text editors, IDEs, and word processor software. Even the Python language comes with a replace() string method since programs often use it. Write a findAndReplace program that has three input : text is the string with text to be replaced, oldText is the text to be replaced, and newText is the replacement text. Keep in mind that this function must be case-sensitive: if you are replacing 'dog' with 'fox', then the 'DOG' in 'MY DOG JONESY' won't be replaced.

[HARD] Q13) Write a Python program that converts totalSeconds into a formatted string with days (d), hours (h), minutes (m), and seconds (s). Omit any unit if its value is zero, except when the input is 0, which should return "0s".

Additional Challenge:

- If the total exceeds **24 hours**, convert extra hours into **days** (e.g., 90042 seconds → 1d 1h 42s).

Input & Expected Output Examples:

Input (Seconds)	**Expected Output**
30	"30s"
60	"1m"
90	"1m 30s"
3600	"1h"
3601	"1h 1s"
3661	"1h 1m 1s"
90042	"1d 1h 42s"
0	"0s"

[HARD] Q14) Write a Python program that converts a resistor circuit into a mathematical expression string using specific enclosure rules:

- **Series connections**: Enclose resistors in curly braces {R1 R2}

- **Parallel connections**: Enclose resistors in square brackets [R1 R2]
  The input is a description of connections, and the output should be a properly nested expression string.

1. **Simple Series Circuit**
    - **Input**: "R1 and R2 in series"
    - **Output**: "{R1 R2}"
2. **Simple Parallel Circuit**
    - **Input**: "R1 and R2 in parallel"
    - **Output**: "[R1 R2]"
3. **Combination Circuit**
    - **Input**: "R1 in series with (R2 parallel R3)"
    - **Output**: "{R1 [R2 R3]}"
4. **Nested Circuit**
    - **Input**: "(R1 parallel R2) in series with (R3 parallel R4)"
    - **Output**: "{[R1 R2] [R3 R4]}"

Represent this circuit : "A 10Ω resistor in series with two parallel resistors (20Ω and 30Ω)"

Expected Output String : "{10 [20 30]}"

Follow Up Question : Can you design a python code for any simple circuit having only resistors(no bridges, etc)? If yes, Can you evaluate them and find the total resistance?

Is it possible? Explore!

[HARD] Q15) Write a Python program that takes a **mathematical expression** as a string input, checks if it is valid, and evaluates it following **BODMAS** (Brackets, Orders/Exponents, Division & Multiplication, Addition & Subtraction) rules. If the expression is invalid, report an error.

Requirements:

1. Input: A string containing a mathematical expression (e.g., "10 + 2 * 3").
2. Output:

- If the expression is valid, compute and print the result.
- If the expression is invalid (e.g., "5 + * 3"), print "Error: Invalid Expression".

3. BODMAS Rule: Follow the correct order of operations.
4. Error Handling: Detect invalid syntax (e.g., mismatched brackets, operators without operands).

Examples:

Input	Output
"3 + 5 * 2"	13
"(10 + 2) * 3"	36
"10 / (2 + 3)"	2.0
"5 + * 3"	Error: Invalid Expression
"2 + (3 * 4"	Error: Mismatched Brackets

Hints & Theory

1. BODMAS Rule (Order of Operations)

- Brackets () → Highest priority
- Orders (Exponents) ** → Next (not covered here, but can be extended)
- Division / & Multiplication * → Left to right
- Addition + & Subtraction - → Left to right

2. Regular Expressions (Regex) for Validation

- Check for valid operators (+, -, *, /).
- Check for balanced brackets using stack approach.
- Ensure no two operators appear consecutively (e.g., 5 + * 3 is invalid).

3. Evaluating the Expression

- Use Python's eval() (with caution) or parse manually.
- Safety Note: eval() can execute arbitrary code, so restrict input if used in real applications.

Code Snippets :

1. **Regular Expressions (**import re**)**

```
import re
# Basic pattern matching
match = re.search(pattern, string)
all_matches = re.findall(pattern, string)
# Full string match (entire string must match)
full_match = re.fullmatch(pattern, string)
# Common patterns:
# - r'\d+'      (one or more digits)
# - r'[a-zA-Z]'  (any letter)
# - r'^...$'     (start to end match)
```

2. **Try-Except Block (Error Handling)**

```
try:
    # Code that might raise exceptions
    result = risky_operation()
except SpecificError as e:
    # Handle specific error
    print(f"Error occurred: {e}")
except AnotherError:
    # Handle different error
```

```
    print("Another error occurred")
except:
    # Catch all other exceptions
    print("Unknown error")
else:
    # Runs if no exceptions occurred
    print("Operation successful")
finally:
    # Always executes (for cleanup)
    print("This always runs")
```

3. **Stack Usage (List Implementation)**

```
stack = []  # Initialize empty stack
# Push operation (add to end)
stack.append(item)
# Pop operation (remove from end)
if stack:  # Always check if stack is not empty
    item = stack.pop()
else:
    print("Stack is empty")
# Peek at top item without removing
if stack:
    top_item = stack[-1]
```