My Python Course Notes

Python Exercises and Examples

Siddharth Patel
HTW Berlin

July 14, 2025

Contents

1 If Else – Decision Making in Python

Q1: Greeting by First Name

Write a program that asks the user for their first name. If the user's name is `Marie` or `Otto`, the program should print `Hello Marie!` or `Hello Otto!` respectively. Otherwise, nothing should be printed. Leading and trailing spaces should be ignored. The input and output must match the following examples exactly.

Example 1:
`First name: Marie`
`Hello Marie!`

Example 2:
`First name: Karl`
(No output expected)

Q1 – Solution

```
name = input("First name: ").strip()

if name == "Marie":
    print("Hello Marie!")
elif name == "Otto":
    print("Hello Otto!")
```

Q2: Compare Two Floating-Point Numbers

Write a program that asks the user to input two floating-point numbers, `a` and `b`. Use an `if-else` statement to determine and print the larger number.

Example:
`a = 10`
`b = 11.2`
`The larger number is: 11.2`

Q2 – Solution

```
a = float(input("a = "))
b = float(input("b = "))

if a > b:
    print("The larger number is:", a)
else:
    print("The larger number is:", b)
```

Q3: Compare with a Single if Statement

Solve the previous task again, but this time using only a single `if` statement without `else`. The input and output must match the example exactly.

Example:
a = 10
b = 11.2
The larger number is: 11.2

Q3 – Solution

```
a = float(input("a = "))
b = float(input("b = "))

if b > a:
    print("The larger number is:", b)
if a > b:
    print("The larger number is:", a)
```

Q4: Largest of Three Numbers

Write a program that asks the user to input three floating-point numbers: a, b, and c. Then determine and print the largest number.

Example:
a = 10
b = 11.2
c = -12.31
The largest number is: 11.2

Q4 – Solution

```
a = float(input("a = "))
b = float(input("b = "))
c = float(input("c = "))

if a >= b and a >= c:
    print("The largest number is:", a)
elif b >= a and b >= c:
    print("The largest number is:", b)
else:
    print("The largest number is:", c)
```

Q5: Age-Based Classification (if-elif-else)

Write a program that asks the user for their age and prints a message based on the following rules:

- Under 18: You are a minor.
- Between 18 and 65: You are an adult.
- 66 and older: You are of retirement age.

Example 1:
Please enter your age: 23
You are an adult.

Example 2:
```
Please enter your age: 13
You are a minor.
```

Q5 – Solution

```
age = int(input("Please enter your age: "))

if age < 18:
    print("You are a minor.")
elif age <= 65:
    print("You are an adult.")
else:
    print("You are of retirement age.")
```

Q6: Age-Based Classification with Extended Range

Write a program similar to Q5 but with the following additional condition:

- Under 18: `You are a minor.`
- Between 18 and 65: `You are an adult.`
- Between 66 and 99: `You are of retirement age.`
- 100 or older: `You are in the Methuselah age group.`

Example:
```
Please enter your age: 101
You are in the Methuselah age group.
```

Q6 – Solution

```
age = int(input("Please enter your age: "))

if age < 18:
    print("You are a minor.")
elif age <= 65:
    print("You are an adult.")
elif age <= 99:
    print("You are of retirement age.")
else:
    print("You are in the Methuselah age group.")
```

Q7: Min and Max from Input Without Using min/max

Write a program that asks the user to enter a list of integers separated by commas. Then determine and print the smallest and largest number without using the built-in `min()` or `max()` functions.

Example:
```
Enter numbers: 1, 3, 7, -4, 9, 0, 2
Smallest number: -4
Largest number: 9
```

Q7 – Solution

```
raw_input = input("Enter numbers: ")
numbers = [int(x.strip()) for x in raw_input.split(",")]

smallest = numbers[0]
largest = numbers[0]

for num in numbers[1:]:
    if num < smallest:
        smallest = num
    if num > largest:
        largest = num

print("Smallest number:", smallest)
print("Largest number:", largest)
```

2 Match Statement – Pattern Matching in Python

Q1: Weekday Abbreviation to Full Name

Write a program that asks the user to input a weekday abbreviation (2 letters), such as `Mo`, `Di`, `Mi`, etc. The program should return the full weekday name. If the input is invalid, print `Error`. Use a `match` statement to solve this.

Example 1:
`Input: Mo`
`Monday`

Example 2:
`Input: Mon`
`Error`

Q1 – Solution

```python
day = input("Input: ").strip()

match day:
    case "Mo":
        print("Monday")
    case "Di":
        print("Tuesday")
    case "Mi":
        print("Wednesday")
    case "Do":
        print("Thursday")
    case "Fr":
        print("Friday")
    case "Sa":
        print("Saturday")
    case "So":
        print("Sunday")
    case _:
        print("Error")
```

Q2: Month Abbreviation to Season

Write a program that asks the user to enter a three-letter month abbreviation (e.g., `Jan`, `Feb`). The input should be case-insensitive. The program should return the corresponding season:

- Spring: Mar–May
- Summer: Jun–Aug
- Autumn: Sep–Nov
- Winter: Dec–Feb

Invalid input should return `Error`. Use a `match` statement with as few `case` blocks as possible (5 total).

Example:
`Input: Feb`
`Winter`

Q2 – Solution

```
month = input("Input: ").strip().lower()

match month:
    case "mar" | "apr" | "may":
        print("Spring")
    case "jun" | "jul" | "aug":
        print("Summer")
    case "sep" | "oct" | "nov":
        print("Autumn")
    case "dec" | "jan" | "feb":
        print("Winter")
    case _:
        print("Error")
```

Q3: Rewrite if-elif-else Using match

Given the following code:

```
        val = int(input('Value: '))
        if val >= 0 and val < 4:
        print('Small number')
        elif val < 6:
        print('Medium number')
        elif val == 7 or val == 9:
        print('Special number')
        else:
        print('Some other number')
```

Rewrite the program using a `match` statement instead of `if-elif-else` to achieve the same logic and output.

Q3 – Solution

```
val = int(input("Value: "))

match val:
    case 0 | 1 | 2 | 3:
        print("Small number")
    case 4 | 5:
        print("Medium number")
    case 7 | 9:
        print("Special number")
    case _:
        print("Some other number")
```

3 The for Loop – Counting, Iteration, and Containers

Q1: Count from 1 to a User-Defined Value

Write a program that asks the user to input a number. Then use a `for` loop to print all numbers from 1 to the entered value.

Example:

```
Last count value: 4
1
2
3
4
```

Q1 – Solution

```python
n = int(input("Last count value: "))
for i in range(1, n + 1):
    print(i)
```

Q2: Count Down to 1

Repeat the previous task, but count down from the input value to 1, using a step of -1.

Example:

```
Last count value: 4
4
3
2
1
```

Q2 – Solution

```python
n = int(input("Last count value: "))
for i in range(n, 0, -1):
    print(i)
```

Q3: Factorial with a for Loop

Write a program that calculates the factorial of a user-given number `n`. Use a `for` loop starting at 1.

Example:

```
Please enter n: 4
n! = 24
```

Q3 – Solution

```
n = int(input("Please enter n: "))
factorial = 1
for i in range(1, n + 1):
    factorial *= i
print("n! =", factorial)
```

Q4: Count Occurrences of a Character in a String

Write a program that asks the user for a string and counts how many times the letter e appears using a `for` loop.

Example:
`Input: Das Leben ist schoen!`
`Number of e: 3`

Q4 – Solution

```
text = input("Input: ")
count = 0
for char in text:
    if char == 'e':
        count += 1
print("Number of e:", count)
```

Q5: Sum All Elements in a List

Given the list:

`intlist = [1, 4, 3, 7, -5, 3, 5]`

Write a program that uses a `for` loop to sum all elements and print the result.

Output:
`Sum: 18`

Q5 – Solution

```
intlist = [1, 4, 3, 7, -5, 3, 5]
total = 0
for number in intlist:
    total += number
print("Sum:", total)
```

4 Functions – Parameters, Return Values, and Reusability

Q1: Average of Three Numbers

Write a function that accepts three floating-point numbers as parameters and returns their average. Add a main program that asks the user to enter three values and then calls the function and displays the result.

Example:

```
a = 12
b = 18
c = 6
Average = 12
```

Q1 – Solution

```python
def average(a, b, c):
    return (a + b + c) / 3

a = float(input("a = "))
b = float(input("b = "))
c = float(input("c = "))

result = average(a, b, c)
print("Average =", int(result))
```

Q2: Factorial Function

Write a function that receives an integer `n` as a parameter and returns `n!`. Add a main program that gets input from the user and prints the result.

Example:

```
n = 4
n! = 24
```

Q2 – Solution

```python
def factorial(n):
    result = 1
    for i in range(1, n + 1):
        result *= i
    return result

n = int(input("n = "))
print("n! =", factorial(n))
```

Q3: Area and Perimeter of Rectangle

Write a function `rect(a, b)` that receives two sides of a rectangle and returns the area and perimeter. Add a main program that takes inputs and prints both results.

Example:

```
a = 4
b = 3
Area = 12
Perimeter = 14
```

Q3 – Solution

```python
def rect(a, b):
    area = a * b
    perimeter = 2 * (a + b)
    return area, perimeter

a = int(input("a = "))
b = int(input("b = "))

area, perimeter = rect(a, b)
print("Area =", area)
print("Perimeter =", perimeter)
```

Q4: Count Vowels Using Function

Write a function that accepts a string `s` and a letter `letter`, and returns how often that letter appears (case-insensitive). In the main program, prompt for a string and print the number of occurrences of `a, e, i, o, u`.

Q4 – Solution

```python
def count_letter(s, letter):
    s = s.lower()
    letter = letter.lower()
    count = 0
    for char in s:
        if char == letter:
            count += 1
    return count

s = input("Input: ")
for vowel in "aeiou":
    print(f"{vowel}: {count_letter(s, vowel)}")
```

Example:

```
Input: Die Summe der Kathetenquadrate ist Gleich dem Hypotenusenquadrat
a: 5
e: 10
i: 3
o: 1
u: 4
```

Q5: Custom `join()` Implementation

Write a function `myJoin(lst, sep)` that replicates the behavior of `join()` without actually using it. It should combine list elements into a single string, separated by `sep`.

Test Code:

```
    mylist = ['Wir', 'müssen', 'uns', 'Sisyphos', 'als', 'einen', 'glücklichen', 'Menschen',
    ↪  'vorstellen']
    mystring = myJoin(mylist, '-')
    print(mystring)
```

Expected Output:
`Wir-müssen-uns-Sisyphos-als-einen-glücklichen-Menschen-vorstellen`

Q5 – Solution

```
def myJoin(lst, sep):
    result = ""
    for i in range(len(lst)):
        result += lst[i]
        if i != len(lst) - 1:
            result += sep
    return result

mylist = ['Wir', 'müssen', 'uns', 'Sisyphos', 'als', 'einen', 'glücklichen', 'Menschen',
↪  'vorstellen']
mystring = myJoin(mylist, '-')
print(mystring)
```

Q6: Recursive Sum Function

Write a recursive function `total(n)` that returns the sum from 1 to `n` without using loops. Include a main program to prompt the user and show the result.

Example:
`n = 4`
`Sum = 10`

Q6 – Solution

```
def total(n):
    if n <= 1:
        return n
    return n + total(n - 1)

n = int(input("n = "))
print("Sum =", total(n))
```

5 The `while` Loop – Manual Iteration and Termination Conditions

Q1: Rewrite a `for`-loop as a `while`-loop (0 to 4)

Given the following `for`-loop:

```
    for i in range(5):
    print(i)
```

Rewrite it using a `while`-loop to produce the same output:

Expected Output:

```
0
1
2
3
4
```

Q1 – Solution

```
i = 0
while i < 5:
    print(i)
    i += 1
```

Q2: Rewrite a `for`-loop with start, stop, step

Given the following `for`-loop:

```
    for i in range(5, 13, 2):
    print(i)
```

Rewrite it using a `while`-loop to produce the same output:

Expected Output:

```
5
7
9
11
```

Q2 – Solution

```
i = 5
while i < 13:
    print(i)
    i += 2
```

Q3: Capitalize User Input Until "Ende"

Write a program that reads user input, converts it to uppercase, and prints it. The loop should end when the user enters ende (case-insensitive). Use a `while`-loop.

Example:

```
Input: hallo welt
Output: HALLO WELT
Input: Noch ein Test
Output: NOCH EIN TEST
Input: Ende
```

Q3 – Solution

```
while True:
    text = input("Input: ").strip()
    if text.lower() == "ende":
        break
    print("Output:", text.upper())
```

Q4: Sum Until Empty Input

Write a program that asks the user to input integers, one at a time, inside a `while`-loop. When the user enters an empty line, stop and display the sum. Ignore extra spaces.

Example:

```
Input: 1
Input:  55
Input: 7
Input:
Sum: 63
```

Q4 – Solution

```
total = 0

while True:
    line = input("Input: ").strip()
    if line == "":
        break
    total += int(line)

print("Sum:", total)
```

Q5: Sum from Multi-Value Inputs with Nested Loops

Write a program that allows the user to input multiple integers in a single line (space-separated). Continue until an empty line is entered, then print the total sum. Use only `while`-loops.

Example:

```
Input: 1 3 5
Input:  7 8
Input: 4
Input:
Sum: 28
```

Q5 – Solution

```
total = 0

while True:
    line = input("Input: ").strip()
    if line == "":
        break
    for num in line.split():
        total += int(num)

print("Sum:", total)
```

6 Importing Modules and Libraries – Using Built-in Functionality

Q1: Simulating Dice Rolls with `random.randint()`

Write a program that asks the user how many times a die should be rolled. Then generate that many random integers between 1 and 6 using `random.randint()`. Make sure to call `random.seed(0)` at the beginning so that the sequence is repeatable.

Example:

```
Rolls: 3
4
4
1
```

Q1 – Solution

```python
import random

random.seed(0)
rolls = int(input("Rolls: "))
for _ in range(rolls):
    print(random.randint(1, 6))
```

Q2: Random Word Selection Using `from ... import`

Write a program that imports only `seed()` and `choice()` from the `random` module. Use `seed(0)`, then ask the user to enter a sentence. Split the sentence into words and use `choice()` to randomly pick one word to display.

Example:

```
Input: red yellow blue
blue
```

Q2 – Solution

```python
from random import seed, choice

seed(0)
sentence = input("Input: ")
words = sentence.split()
print(choice(words))
```

Q3: Full Access to All `random` Functions

Given the following main program:

```
    seed(0)
    print(randint(1, 10))
    print(randrange(1, 10))
```

```
    print(random())
    print(choice(['red', 'yellow', 'green']))
```

Complete the code so that all `random` functions are accessible directly, without using the `random.` prefix.

Q3 – Solution

```
from random import *

seed(0)
print(randint(1, 10))
print(randrange(1, 10))
print(random())
print(choice(['red', 'yellow', 'green']))
```

7 Lists and Tuples – Access, Operations, and Manual Algorithms

Q1: Accessing List Elements

Given the list:

```
    lst = [1, 3, 9, -11, 2, 0, 21]
```

Access and print the second and last element of the list. Then create a new list containing the 3rd, 4th, and 5th elements and print it.

Expected Output:

```
3
21
[9, -11, 2]
```

Q1 – Solution

```
lst = [1, 3, 9, -11, 2, 0, 21]

print(lst[1])         # second element
print(lst[-1])        # last element
print(lst[2:5])       # 3rd, 4th, 5th element as sublist
```

Q2: Element-wise Addition of Equal-Length Lists

Given:

```
    lst1 = [1, 3, 9, -11, 2, 0, 21]
    lst2 = [2, 1, -2, 0, 3, -2, -11]
```

Write a program that adds the lists element by element and stores the result in a new list.

Expected Output:

```
Result: [3, 4, 7, -11, 5, -2, 10]
```

Q2 – Solution

```
lst1 = [1, 3, 9, -11, 2, 0, 21]
lst2 = [2, 1, -2, 0, 3, -2, -11]

result = []
for i in range(len(lst1)):
    result.append(lst1[i] + lst2[i])

print("Result:", result)
```

Q3: Element-wise Addition with Unequal Lengths

Now allow `lst1` and `lst2` to have different lengths. When that's the case, append the extra elements from the longer list to the result.

```
    lst1 = [1, 3, 9, -11, 2, 0, 21]
    lst2 = [2, 1, -2, 0, 3, -2, -11, 4, 8]
```

Expected Output:

```
Result: [3, 4, 7, -11, 5, -2, 10, 4, 8]
```

Q3 – Solution

```
lst1 = [1, 3, 9, -11, 2, 0, 21]
lst2 = [2, 1, -2, 0, 3, -2, -11, 4, 8]

result = []
min_len = min(len(lst1), len(lst2))

for i in range(min_len):
    result.append(lst1[i] + lst2[i])

if len(lst1) > len(lst2):
    result += lst1[min_len:]
else:
    result += lst2[min_len:]

print("Result:", result)
```

Q4: Convert User Input to List of Integers

Prompt the user to enter numbers in a single line. Convert them into integers and store them in a list.

Example:

```
Input: 1 2 3 4 5
Result: [1, 2, 3, 4, 5]
```

Q4 – Solution

```
user_input = input("Input: ")
numbers = [int(x) for x in user_input.split()]
print("Result:", numbers)
```

Q5: Sort List Without Using `sort()` or `sorted()`

Given the list:

```
    lst = [1, 3, 9, -11, 2, 0, 21]
```

Write a program that sorts the list in ascending order using your own sorting algorithm (e.g., selection or bubble sort).

Expected Output:
`Result: [-11, 0, 1, 2, 3, 9, 21]`

Q5 – Solution

```
lst = [1, 3, 9, -11, 2, 0, 21]

for i in range(len(lst)):
    min_index = i
    for j in range(i + 1, len(lst)):
        if lst[j] < lst[min_index]:
            min_index = j
    lst[i], lst[min_index] = lst[min_index], lst[i]

print("Result:", lst)
```

8 Dictionaries – Key-Value Data Storage and Access

Q1: Loop Through Dictionary Keys and Values

Given the dictionary:

```
tab = {'Mo': 1, 'Di': 2, 'Mi': 3, 'Do': 4, 'Fr': 5, 'Sa': 6, 'So': 7}
```

Write a program that prints each key and its corresponding value on a separate line.

Expected Output:

```
Mo: 1
Di: 2
Mi: 3
Do: 4
Fr: 5
Sa: 6
So: 7
```

Q1 – Solution

```
tab = {'Mo': 1, 'Di': 2, 'Mi': 3, 'Do': 4, 'Fr': 5, 'Sa': 6, 'So': 7}

for key in tab:
    print(f"{key}: {tab[key]}")
```

Q2: Check If a Key Exists in Dictionary

Using the same dictionary `tab`, prompt the user for input. If the input is a key in the dictionary, print its value. Otherwise, print `"Unknown word"`.

Example 1:

```
Input: Mi
3
```

Example 2:

```
Input: Mittwoch
Unknown word
```

Q2 – Solution

```
tab = {'Mo': 1, 'Di': 2, 'Mi': 3, 'Do': 4, 'Fr': 5, 'Sa': 6, 'So': 7}

word = input("Input: ")
if word in tab:
    print(tab[word])
else:
    print("Unknown word")
```

Q3: Count Character Frequency Using Dictionary

Prompt the user to enter a sentence. Use a dictionary to count the number of times each character appears. Output the dictionary.

Example:

```
Input: Eins und Eins ist Zwei
Result: {'E': 2, 'i': 4, 'n': 3, 's': 3, ' ': 4, 'u': 1, 'd': 1, 't': 1, 'Z': 1,
'w': 1, 'e': 1}
```

Q3 – Solution

```
text = input("Input: ")
count_dict = {}

for char in text:
    if char in count_dict:
        count_dict[char] += 1
    else:
        count_dict[char] = 1

print("Result:", count_dict)
```

Q4: Interactive Dictionary with Update and Lookup

Write a program that starts with an empty dictionary `tab`. In a loop, prompt the user for input:

- If the input contains a single word and the key exists in `tab`, print the value.
- If the key doesn't exist, print `"Unknown word"`.
- If the input contains two words, insert the first as key and second as value. Print `"New entry"`.
- If the input is `Ende`, stop and print the dictionary.

Example:

```
Input: Hallo
Unknown word
Input: Hallo Welt
New entry
Input: Hallo
Welt
Input: Ende
{'Hallo': 'Welt'}
```

Q4 – Solution

```
tab = {}

while True:
    user_input = input("Input: ").strip()

```

```
    if user_input == "Ende":
        print(tab)
        break

    parts = user_input.split()

    if len(parts) == 1:
        key = parts[0]
        if key in tab:
            print(tab[key])
        else:
            print("Unknown word")

    elif len(parts) == 2:
        key, value = parts
        tab[key] = value
        print("New entry")
```

9 Sets – Unique Collections and Set Operations

Q1: Collect Unique Words Until "Ende"

Write a program that asks the user to enter words in a loop. Store each word in a set. Stop when the user enters `Ende`. The word `Ende` should not be included in the set. After termination, print the resulting set.

Example:

```
Input: Hallo
Input: Berlin
Input: Hallo
Input: Welt
Input: Ende
{'Hallo', 'Welt', 'Berlin'}
```

Q1 – Solution

```python
words = set()

while True:
    entry = input("Input: ")
    if entry == "Ende":
        break
    words.add(entry)

print(words)
```

Q2: Remove Words from a Set

Given the set:

```python
    s = {'der', 'die', 'das', 'und', 'er', 'sie', 'es', 'du'}
```

Prompt the user for words. If the word is in the set, remove it and print `Removed`. If it's not in the set, print `Not found`. Stop on `Ende` and print the set.

Example:

```
Input: Hallo
Not found
Input: und
Removed
Input: und
Not found
Input: Ende
{'der', 'die', 'das', 'er', 'sie', 'es', 'du'}
```

Q2 – Solution

```
s = {'der', 'die', 'das', 'und', 'er', 'sie', 'es', 'du'}

while True:
    word = input("Input: ")
    if word == "Ende":
        break
    if word in s:
        s.remove(word)
        print("Removed")
    else:
        print("Not found")

print(s)
```

Q3: Intersection of Word Sets from Two Sentences

Prompt the user to enter two sentences. Split the sentences into words and store them in sets. Then compute and print the intersection.

Example:

```
Sentence 1: Der Löwe ist der Adler unter den Katzen
Sentence 2: Der Adler ist der Löwen unter den Piematzen
{'der', 'unter', 'ist', 'Der', 'den', 'Adler'}
```

Q3 – Solution

```
sentence1 = input("Sentence 1: ")
sentence2 = input("Sentence 2: ")

set1 = set(sentence1.split())
set2 = set(sentence2.split())

print(set1 & set2)
```

Q4: Union of Word Sets from Two Sentences

Same as Q3, but compute the union of the sets instead.

Example:

```
Sentence 1: Der Löwe ist der Adler unter den Katzen
Sentence 2: Der Adler ist der Löwen unter den Piematzen
{'Der', 'Katzen', 'der', 'Piematzen', 'Adler', 'den', 'Löwe', 'unter', 'Löwen',
'ist'}
```

Q4 – Solution

```
sentence1 = input("Sentence 1: ")
sentence2 = input("Sentence 2: ")
```

```

set1 = set(sentence1.split())
set2 = set(sentence2.split())

print(set1 | set2)
```

Q5: Difference – Words in First Sentence Only

Now compute the difference of the two sets: words that are in the first sentence but not in the second.

Example:

```
Sentence 1: Der Löwe ist der Adler unter den Katzen
Sentence 2: Der Adler ist der Löwen unter den Piematzen
{'Katzen', 'Löwe'}
```

Q5 – Solution

```
sentence1 = input("Sentence 1: ")
sentence2 = input("Sentence 2: ")

set1 = set(sentence1.split())
set2 = set(sentence2.split())

print(set1 - set2)
```

10 Classes and Objects – OOP with Methods, Attributes, and Inheritance

Q1: Rectangle Class – Area and Perimeter

Write a class `Rectangle` with two attributes `a` and `b` (side lengths). The constructor should initialize both attributes. Add methods `getArea()` and `getPerimeter()` that return the area and perimeter.

Add a main program that creates two rectangles with dimensions 2×3 and 4×7. Output should be:

Expected Output:

```
Rectangle 1 Area: 6.0
Rectangle 1 Perimeter: 10.0
Rectangle 2 Area: 28.0
Rectangle 2 Perimeter: 22.0
```

Q1 – Solution

```python
class Rectangle:
    def __init__(self, a, b):
        self.a = a
        self.b = b

    def getArea(self):
        return self.a * self.b

    def getPerimeter(self):
        return 2 * (self.a + self.b)


r1 = Rectangle(2, 3)
r2 = Rectangle(4, 7)

print("Rectangle 1 Area:", r1.getArea())
print("Rectangle 1 Perimeter:", r1.getPerimeter())
print("Rectangle 2 Area:", r2.getArea())
print("Rectangle 2 Perimeter:", r2.getPerimeter())
```

Q2: Circle Class – Radius and Area (with private attribute)

Write a class `Circle` with a private attribute `_r`. Include:

- `getR()` – returns radius
- `setR(r)` – sets a new radius
- `getArea()` – returns area using $\pi = 3.14$

In the main program, create two circles with radii 3 and 4. Print radius and area. Then update circle1's radius to 10 and print again.

Expected Output:

```
Circle 1: r = 3, A = 28.26
```

```
Circle 2: r = 4, A = 50.24
Circle 1: r = 10, A = 314.0
```

Q2 – Solution

```python
class Circle:
    def __init__(self, r):
        self.__r = r

    def getR(self):
        return self.__r

    def setR(self, r):
        self.__r = r

    def getArea(self):
        return 3.14 * self.__r * self.__r


circle1 = Circle(3)
circle2 = Circle(4)

print(f"Circle 1: r = {circle1.getR()}, A = {circle1.getArea()}")
print(f"Circle 2: r = {circle2.getR()}, A = {circle2.getArea()}")

circle1.setR(10)
print(f"Circle 1: r = {circle1.getR()}, A = {circle1.getArea()}")
```

Q3: Triangle Class with Magic Method

Create a class `Triangle` with private attributes _c (base) and _h (height). Include:

- Constructor to initialize both
- `getArea()` method for triangle area
- `__str__()` magic method to support `print(triangle)` with formatted output

Main program:

```python
    tri = Triangle(4, 5)
    print(tri)
```

Expected Output:

```
Base: 4, Height: 5, Area: 10
```

Q3 – Solution

```python
class Triangle:
    def __init__(self, c, h):
        self.__c = c
        self.__h = h

    def getArea(self):
```

```
        return 0.5 * self.__c * self.__h

    def __str__(self):
        return f"Base: {self.__c}, Height: {self.__h}, Area: {self.getArea()}"


tri = Triangle(4, 5)
print(tri)
```

Q4: Inheritance – Tank, Cylinder, and Cuboid

Write a base class `Tank` with private attributes:

- `_A` – area of the base
- `_h` – height

Include:

- `getVolume()` – returns `A × h`
- `getArea()` – returns `A`

Now define two subclasses: - `Cilinder(r, h)`: calculates and passes πr^2 as base area; implements `getR()` using $\sqrt{A/\pi}$ - `Quarder(a, h)`: calculates and passes a^2 as base area; implements `getA()` using $\sqrt{A}$

Main program: Create a `Cilinder` with `r = 1, h = 2` and a `Quarder` with `a = 3, h = 4`. Output:

Expected Output:

```
Cilinder: r = 1.0, Volume = 6.28
Quarder: a = 3.0, Volume = 36.0
```

Q4 – Solution

```
import math

class Tank:
    def __init__(self, A, h):
        self.__A = A
        self.__h = h

    def getVolume(self):
        return self.__A * self.__h

    def getArea(self):
        return self.__A


class Cilinder(Tank):
    def __init__(self, r, h):
        area = math.pi * r * r
        super().__init__(area, h)

    def getR(self):
        return math.sqrt(self.getArea() / math.pi)


```

```
class Quarder(Tank):
    def __init__(self, a, h):
        area = a * a
        super().__init__(area, h)

    def getA(self):
        return math.sqrt(self.getArea())


c = Cilinder(1, 2)
q = Quarder(3, 4)

print(f"Cilinder: r = {c.getR()}, Volume = {c.getVolume()}")
print(f"Quarder: a = {q.getA()}, Volume = {q.getVolume()}")
```

11 Comprehensions – Lists, Dictionaries, and Sets in One Line

Q1: List of Squares up to N

Prompt the user to enter a number `N`. Use a list comprehension to generate a list of square numbers from 0 to `N`.

Example:

```
N = 5
[0, 1, 4, 9, 16, 25]
```

Q1 – Solution

```python
# Q1: Squares up to N using list comprehension
N = int(input("N = "))
squares = [i**2 for i in range(N+1)]
print(squares)
```

Q2: Range with Step Using Comprehension

Prompt the user for `start` and `end`. Create a list from `start` to `end` (inclusive) with step 2 using a comprehension.

Example:

```
start = 4
end = 10
[4, 6, 8, 10]
```

Q2 – Solution

```python
# Q2: Range with step 2 using list comprehension
start = int(input("start = "))
end = int(input("end = "))
result = [i for i in range(start, end+1, 2)]
print(result)
```

Q3: Numbers Not Divisible by 3

Prompt the user for `N`. Create a list from 1 to `N` that excludes numbers divisible by 3.

Example:

```
N = 10
[1, 2, 4, 5, 7, 8, 10]
```

Q3 – Solution

```python
# Q3: Numbers from 1 to N not divisible by 3
N = int(input("N = "))
filtered = [i for i in range(1, N+1) if i % 3 != 0]
print(filtered)
```

Q4: Capitalized Words in a Sentence

Ask the user to input a sentence. Use a list comprehension to extract only words that start with an uppercase letter.

Example:

Input: In einem rechtwinkligen Dreieck ist die Summe der Kathetenquadrate gleich dem Quadrat der Hypotenuse
['In', 'Dreieck', 'Summe', 'Kathetenquadrate', 'Quadrat', 'Hypotenuse']

Q4 – Solution

```python
# Q4: Capitalized words in a sentence
sentence = input("Eingabe: ")
words = sentence.split()
capitalized = [word for word in words if word[0].isupper()]
print(capitalized)
```

Q5: Invert Dictionary with Comprehension

Given the dictionary:

```python
    tab = {'Montag': 'Monday', 'Dienstag': 'Tuesday', 'Mittwoch': 'Wednesday',
        'Donnerstag': 'Thursday', 'Freitag': 'Friday', 'Samstag': 'Saturday',
        'Sonntag': 'Sunday'}
```

Use a comprehension to create a new dictionary where keys and values are swapped.

Expected Output:

{'Monday': 'Montag', 'Tuesday': 'Dienstag', ...}

Q5 – Solution

```python
# Q5: Invert the tab dictionary using a dictionary comprehension
tab = {
    'Montag': 'Monday',
    'Dienstag': 'Tuesday',
    'Mittwoch': 'Wednesday',
    'Donnerstag': 'Thursday',
    'Freitag': 'Friday',
    'Samstag': 'Saturday',
    'Sonntag': 'Sunday'
}
```

```python

# Invert key and value
tab = {value: key for key, value in tab.items()}
print(tab)
```

Q6: Filter Even Month Numbers

Given the dictionary:

```python
    tab = {'Jan': '1', 'Feb': '2', 'Mär': '3', 'Apr': '4', 'Mai': '5',
        'Jun': '6', 'Jul': '7', 'Aug': '8', 'Sep': '9', 'Okt': '10',
        'Nov': '11', 'Dez': '12'}
```

Create a new dictionary using comprehension that includes only months with even numbers (converted to `int`).

Q6 – Solution

```python
# Q6: Filter months with even values and convert to int
tab = {
    'Jan': '1', 'Feb': '2', 'Mär': '3', 'Apr': '4',
    'Mai': '5', 'Jun': '6', 'Jul': '7', 'Aug': '8',
    'Sep': '9', 'Okt': '10', 'Nov': '11', 'Dez': '12'
}

# Keep only even-numbered months, convert value to int
filtered = {k: int(v) for k, v in tab.items() if int(v) % 2 == 0}
print(filtered)
```

Expected Output:

```
{'Feb': 2, 'Apr': 4, 'Jun': 6, 'Aug': 8, 'Okt': 10, 'Dez': 12}
```

Q7: Set Comprehension from User Input

Use a set comprehension to prompt the user five times for input. Store all unique values in a set. The full program should be no more than 2 lines.

Example:

```
Input: hallo
Input: welt
Input: hallo
Input: berlin
Input: hallo
{'hallo', 'welt', 'berlin'}
```

Q7 – Solution

```python
# Q7: Set comprehension from 5 user inputs
values = {input("Eingabe: ") for _ in range(5)}
print(values)
```

12 File Exercises

Q1: File Analysis – Count Lines, Words, and Characters

Write a program that reads a file named `automobil.txt`. The program should count the number of lines, the number of words, and the number of characters, and display them.

Example Output:

```
Number of lines: 114
Number of words: 1735
Number of characters: 11807
```

Q1 – Solution

```python
# Q1: Count lines, words, and characters in automobil.txt
with open("automobil.txt", "r", encoding="utf-8") as file:
    content = file.readlines()

lines = len(content)
words = sum(len(line.split()) for line in content)
chars = sum(len(line) for line in content)

print("Number of lines:", lines)
print("Number of words:", words)
print("Number of characters:", chars)
```

Q2: File Uppercase Conversion

Write a program that reads the file `automobil.txt`, converts its content to uppercase, and writes the result into a new file named `result.txt`.

Q2 – Solution

```python
# Q2: Convert contents of automobil.txt to uppercase and write to result.txt
with open("automobil.txt", "r", encoding="utf-8") as infile:
    content = infile.read()

uppercase_content = content.upper()

with open("result.txt", "w", encoding="utf-8") as outfile:
    outfile.write(uppercase_content)
```

Q3: Binary File Comparison

Read two binary files `test1.bin` and `test2.bin`, compare them byte by byte, and count how many bytes differ. Output the result.

Example Output:

```
Differences: 1216
```

Q3 – Solution

```python
# Q3: Compare two binary files and count differing bytes
with open("test1.bin", "rb") as f1, open("test2.bin", "rb") as f2:
    data1 = f1.read()
    data2 = f2.read()

min_len = min(len(data1), len(data2))
diff = sum(b1 != b2 for b1, b2 in zip(data1[:min_len], data2[:min_len]))
diff += abs(len(data1) - len(data2))  # Count extra bytes in longer file

print("Differences:", diff)
```

13 Exceptions

Q1: Reciprocal Calculation with ZeroDivisionError

Prompt the user to enter an integer `x`. Compute the reciprocal `1/x` and display the result. If the user enters 0, catch the `ZeroDivisionError` and print an appropriate message.

Example 1:

```
x = 5
1/x = 0.2
```

Example 2:

```
x = 0
Division by 0 not allowed
```

Q1 – Solution

```
# Q1: Reciprocal with ZeroDivisionError handling
try:
    x = int(input("x = "))
    print("1/x =", 1/x)
except ZeroDivisionError:
    print("Division by 0 not allowed")
```

Q2: Error Handling for Division and Type

Extend the previous program. If the user inputs a word (not an integer), catch a `ValueError` and print an appropriate message.

Example 1:

```
x = 5
1/x = 0.2
```

Example 2:

```
x = 0
Division by 0 not allowed
```

Example 3:

```
x = hello
Please enter a number
```

Q2 – Solution

```
# Q2: Handle ZeroDivisionError and ValueError
try:
    x = int(input("x = "))
    print("1/x =", 1/x)
except ZeroDivisionError:
    print("Division by 0 not allowed")
except ValueError:
    print("Please enter a number")
```

Q3: Raising a ValueError for Negative Input

Raise a `ValueError` when the user enters a negative number. All exceptions (`ZeroDivisionError`, `ValueError`) should be handled with a single `except` block.

Example:

```
x = -1
Negative numbers not allowed
```

Q3 – Solution

```python
# Q3: Raise and catch all exceptions, including custom ValueError
try:
    x = int(input("x = "))
    if x < 0:
        raise ValueError("Negative numbers not allowed")
    print("1/x =", 1/x)
except Exception as e:
    print(e)
```