**Task 49**

def is\_perfect\_number(n):

    if n <= 0:

        return False  # Müsbət tam ədəd olmalıdır

    divisors\_sum = sum(i for i in range(1, n) if n % i == 0)

    return divisors\_sum == n

# Test nümunələri

print(is\_perfect\_number(28))  # True

print(is\_perfect\_number(6))   # True

print(is\_perfect\_number(12))  # False

print(is\_perfect\_number(100)) # False

**Use remove() when you want an error if the element is missing (ensuring it's present before deletion).Use discard() when you want to avoid errors if the element is absent.**

**Task 50**

import string

def word\_frequency\_counter(filename):

    try:

        with open(filename, 'r', encoding='utf-8') as file:

            text = file.read().lower()  # Bütün mətni kiçik hərflərə çeviririk

        text = text.translate(str.maketrans("", "", string.punctuation))  # İmlâ işarələrini silirik

        words = text.split()  # Sözləri ayırırıq

        word\_count = {}

        for word in words:

            word\_count[word] = word\_count.get(word, 0) + 1

        sorted\_words = sorted(word\_count.items(), key=lambda x: x[1], reverse=True)  # Azalan sıraya görə sıralayırıq

        for word, count in sorted\_words:

            print(f"{word}: {count}")

    except FileNotFoundError:

        print("Fayl tapılmadı. Zəhmət olmasa düzgün fayl adı daxil edin.")

# Test üçün fayl adı

word\_frequency\_counter("sample.txt")  # Burada "sample.txt" sənin fayl adın olmalıdır

1. del – Açarı (key) silir, əgər açar yoxdursa, KeyError verir.

2. pop(key) – Açarı silir və onun dəyərini qaytarır. Açar yoxdursa, KeyError verə bilər.

3. popitem() – Son əlavə olunan elementi silir və (key, value) qaytarır.

4. clear() – Lüğətin bütün elementlərini silir.

**Task 51**

def find\_character\_indices(text, char):

    first\_index = text.find(char)

    if first\_index == -1:

        return -1, -1

    last\_index = text.rfind(char)

    return first\_index, last\_index

# Alternative implementation that raises an error

def find\_character\_indices\_strict(text, char):

    first\_index = text.find(char)

    if first\_index == -1:

        raise ValueError(f"Character '{char}' not found in string")

    last\_index = len(text) - 1 - text[::-1].find(char)  # Using slicing as required

    return first\_index, last\_index

**Task 52**

def count\_word\_occurrences(sentence, word):

    """

    Counts how many times a word appears in a sentence (case-insensitive).

    Args:

        sentence: The input string to search in

        word: The word to search for

    Returns:

        int: Number of occurrences of the word

    """

    words\_in\_sentence = sentence.lower().split()

    target\_word = word.lower()

    return words\_in\_sentence.count(target\_word)

**Task 53**

Python's len() function works similarly for both strings and lists, but there are some key differences:

**\***Common Behavior:\*

- Returns the number of elements/characters

- Works with any sequence type in Python

- Time complexity is O(1) for both

\*With Strings:\*

- Counts the number of Unicode characters

- Example: len("hello") returns 5

- Includes whitespace and special characters in count

\*With Lists:\*

- Counts the number of elements in the list

- Example: len([1, 2, 3]) returns 3

- Counts each element regardless of its type or size

def get\_string\_length(input\_string):

    """

    Returns the length of a string or a message for empty strings.

    Args:

        input\_string: The string to measure

    Returns:

        int or str: Length of string or "Empty string" message

    """

    if not input\_string:  # This handles None and empty string

        return "Empty string"

    return len(input\_string)

**Task 54**

These are both sequence types in Python, but with important differences:

\*Mutability:\*

- list() creates a mutable sequence (can be modified after creation)

- tuple() creates an immutable sequence (cannot be modified after creation)

def starts\_with\_ignore\_case(main\_string, prefix):

    """

    Checks if a string starts with a specified prefix (case-insensitive).

    Args:

        main\_string: The string to check

        prefix: The prefix to look for

    Returns:

        bool: True if the string starts with the prefix (case-insensitive), False otherwise

    """

    return main\_string.lower().startswith(prefix.lower())

**Task 55**

These are both string methods but serve different purposes:

\*index() Method:\*

- Returns the lowest index where the substring is found

- Raises ValueError if substring is not found

- Syntax: str.index(sub[, start[, end]])

- Useful when you need the position of a substring

\*count() Method:\*

- Returns the number of occurrences of a substring

- Returns 0 if substring is not found (no error)

- Syntax: str.count(sub[, start[, end]])

- Useful when you need occurrence statistics

def is\_txt\_file(filename):

    """

    Checks if a filename ends with .txt extension (case-insensitive).

    Args:

        filename: The filename to check

    Returns:

        bool: True if filename ends with .txt extension, False otherwise

    """

    return filename.lower().endswith('.txt')

**Task 56**

def clean\_whitespace(input\_string):

    """

    Removes leading and trailing whitespace from a string and prints the result.

    Handles edge cases like all-space strings.

    Args:

        input\_string: The string to clean

    Returns:

        None: Prints the cleaned string directly

    """

    cleaned = input\_string.strip()

    print(f"Cleaned string: '{cleaned}'")  # Quotes show exact boundaries

hese string methods serve completely different purposes:

\*strip() Method:\*

- Removes leading and trailing characters (whitespace by default)

- Returns a new string

- Syntax: str.strip([chars])

- Modifies string boundaries

- Useful for cleaning user input or file data

\*split() Method:\*

- Splits a string into a list of substrings

- Uses a delimiter (whitespace by default)

- Syntax: str.split(sep=None, maxsplit=-1)

- Divides string content

- Useful for parsing text or CSV data

**Task 58**

def replace\_word(sentence):

    # Replace 'Python' with 'JavaScript' in a case-insensitive manner

    return sentence.replace('Python', 'JavaScript').replace('python', 'JavaScript').replace('PYTHON', 'JavaScript')

# Example usage

input\_sentence = "Python is an awesome programming language. I love python!"

output\_sentence = replace\_word(input\_sentence)

print(output\_sentence)

Python-da bir mətnin **yalnız ilk uyğun gələn sözünü dəyişdirmək üçün** replace() və re.sub() metodlarından istifadə olunur.

✅ **replace() metodu** üçüncü parametr kimi **dəyişdiriləcək maksimum sayı təyin etməyə imkan verir**.  
✅ **re.sub() metodu** isə **regular ifadələrlə işləyir** və count=1 parametri ilə yalnız ilk uyğunluğu əvəzləyə bilər.

**Task 59**

def remove\_prefix(string, prefix):

    """

    Removes the specified prefix from the string if it exists.

    Parameters:

    string (str): The original string.

    prefix (str): The prefix to be removed.

    Returns:

    str: The modified string without the prefix.

    """

    # Check if the string starts with the prefix

    if string.startswith(prefix):

        return string[len(prefix):]  # Remove the prefix

    return string  # Return original string if prefix not found

# Example usage

result = remove\_prefix("55HelloWorld", "55")

print(result)  # Output: HelloWorld

Python-da **startswith('55') funksiyası** bir sətirin "55" ilə başlayıb-başlamadığını yoxlayır.

✅ Əgər başlayırsa, **True**, başlamırsa, **False** qaytarır.  
✅ **Alternativ çap üsulları:**

* if-else istifadə edərək nəticəni ayrıca çap etmək
* Ternary operator (print(True if ... else False)) ilə bir sətirdə yazmaq
* int() ilə True → 1, False → 0 kimi göstərmək.

**Task 60**

def remove\_suffix(filename):

    """

    Removes the .html suffix from a given filename if it exists.

    Parameters:

    filename (str): The filename from which to remove the suffix.

    Returns:

    str: The filename without the .html suffix.

    """

    # Check if the filename ends with .html

    if filename.endswith('.html'):

        # Remove the suffix

        return filename[:-5]

    return filename

# Example usage

file\_name\_1 = "index.html"

file\_name\_2 = "document.txt"

print(remove\_suffix(file\_name\_1))  # Output: index

print(remove\_suffix(file\_name\_2))  # Output: document.txt

Python-da **tuple dəyişməzdir (immutable)**, yəni birbaşa elementlərini **dəyişmək mümkün deyil**.

**Task61**

def absolute\_value\_message(number):

    """

    Bu funksiya bir tam ədəd qəbul edir və onun mütləq dəyərini

    "This is absolute value" ifadəsi ilə birlikdə qaytarır.

    """

    # Mütləq dəyəri tap

    abs\_value = abs(number)

    # Format funksiyası ilə düzgün qaydada çap et

    message = "This is absolute value: {}".format(abs\_value)

    return message

# Funksiyanı test edək

print(absolute\_value\_message(-10))  # This is absolute value: 10

print(absolute\_value\_message(5))    # This is absolute value: 5

print(absolute\_value\_message(-100)) # This is absolute value: 100

 round() funksiyası **ədədi yuvarlaqlaşdıraraq** 2 rəqəm saxlayır.

 format() və f-string metodları isə **ədədi sətir kimi formatlayır**, lakin dəyəri dəyişmir.

**Task62**

def calculate\_power():

    """

    Bu funksiya istifadəçidən iki ədəd (x və y) alır və x^y-ni hesablayır.

    Həm tam, həm də onluq ədədləri qəbul edir.

    """

    try:

        # İstifadəçidən ədədləri al

        x = float(input("Əsas (x) ədədini daxil edin: "))

        y = float(input("Üst (y) ədədini daxil edin: "))

        # Üstlü qüvvəti hesabla

        result = x \*\* y

        # Nəticəni qaytar və formatla

        print(f"{x} ^ {y} = {result:.2f}")

    except ValueError:

        # Əgər istifadəçi səhv məlumat daxil edərsə

        print("Xəta! Zəhmət olmasa, yalnız ədədlər daxil edin.")

# Funksiyanı işə sal

calculate\_power()

x\*\*y sadedir ve birbasa hesablayir.Sadece quvvet hesablayir.

Pow(x,y) daha funksionaldir

**Task 63**

def round\_number():

    num = float(input("Ondalıklı ədəd daxil edin: "))

    rounded\_value = round(num, 2)

    print(f"Yuvarlaqlaşdırılmış dəyər: {rounded\_value}")

round\_number()

str() funksiyası ilə

Nümunə:

age = 25

message = "Mənim yaşım " + str(age) + " -dir."

print(message)

**Task 64**

def integer\_division(a, b):

    if b == 0:

        return "Error: Division by zero is not allowed"

    quotient = a // b  # Tam bölmə

    remainder = a % b   # Qalıq

    return quotient, remainder

**Python-da / və // operatorları bölmə üçün istifadə olunur, lakin fərqli nəticələr qaytarırlar:**

**/ (Adi bölmə): Həmişə float tipli nəticə qaytarır.**

**// (Tam bölmə): Bölmənin yalnız tam hissəsini qaytarır**

**Task 65**

def format\_number(num):

    formatted\_num = "{:>10.2f}".format(num)

    print(formatted\_num)

Python-da **sətiri bayt formatında çap etmək üçün** encode() metodundan istifadə olunur.

✅ **UTF-8 və ya başqa kodlaşdırma formatı seçilə bilər**.  
✅ **Bayt formatında çap edildikdə** nəticə b'' şəklində görünür.

**Task 66**

def factorial(n):

    if n < 0:

        print("Error: Factorial is not defined for negative numbers.")

        return None

    elif n == 0:

        return 1

    else:

        result = 1

        for i in range(1, n + 1):

            result \*= i

        return result

For döngüsü :döngünün neçə dəfə işləyəcəyini əvvəlcədən bilirik.

While döngüsü isə müəyyən bir şərt doğru olduğu müddətdə işləyir.

**Task 67**

String: Dəyişməzdir (immutable). Yəni, bir string yaradıldıqdan sonra onu dəyişdirmək olmaz. Yeni bir dəyər təyin etmək üçün yenidən string yaratmaq lazımdır.String: Yalnız simvollardan (character) ibarətdir.List: Dəyişkəndir (mutable). Siyahının (listin) elementlərini dəyişmək, silmək və ya yenisini əlavə etmək mümkündür.List: Fərqli tipdə (int, float, string, hətta başqa listlər) elementlər saxlayır**.**

# 1-ci üsul: Daxili (built-in) funksiyalarla

def min\_max\_builtin(numbers):

    """

    Bu funksiya daxili Python funksiyaları olan min() və max() ilə

    minimum və maksimum dəyərləri tapır.

    Əgər siyahı boşdursa, None qaytarır.

    """

    if not numbers:

        return None

    return min(numbers), max(numbers)

# 2-ci üsul: Dövr (loop) və müqayisə ilə

def min\_max\_manual(numbers):

    """

    Bu funksiya dövr (loop) istifadə edərək siyahıdakı

    minimum və maksimum dəyərləri əl ilə hesablayır.

    Əgər siyahı boşdursa, None qaytarır.

    """

    if not numbers:

        return None

    min\_val = numbers[0]

    max\_val = numbers[0]

    for num in numbers[1:]:

        if num < min\_val:

            min\_val = num

        if num > max\_val:

            max\_val = num

    return min\_val, max\_val

# Nümunə istifadə:

nums = [12, 5, 87, 3, 45, 99]

print("Built-in nəticə:", min\_max\_builtin(nums))

print("Manual nəticə:", min\_max\_manual(nums))

**Task 68**

def filter\_even\_numbers(lst):

    # Boş siyahı yaradılır ki, cüt ədədlər burada toplanacaq

    even\_numbers = []

    # Siyahıdakı hər bir ədəd üzərində dövr edilir

    for num in lst:

        # Əgər ədədin 2-yə bölümündən qalığı 0-dırsa (yəni ədəd cütdür)

        if num % 2 == 0:

            # O, cüt ədədlər siyahısına əlavə edilir

            even\_numbers.append(num)

    # Funksiya, yalnız cüt ədədlərin yerləşdiyi siyahını qaytarır

    return even\_numbers

# Funksiyanı test edirik

numbers = [1, 2, 3, 4, 5, 6, 7, 8, 9, 10]

print(filter\_even\_numbers(numbers))  # Nəticə: [2, 4, 6, 8, 10]

tuple\_list = [("name", "Alice"), ("age", 25), ("city", "Baku")]

dictionary = dict(tuple\_list)

print(dictionary) # Nəticə: {'name': 'Alice', 'age': 25, 'city': 'Baku'}

**Task 69**

# Fibonacci ardıcıllığını döngü ilə yaradan funksiya

def fibonacci(n):

    """

    Bu funksiya istifadəçidən alınan 'n' sayda Fibonacci ədədlərini qaytarır.

    Fibonacci ardıcıllığı: 0, 1, 1, 2, 3, 5, 8, ...

    Əgər n < 1 olarsa, boş siyahı qaytarır.

    """

    if n <= 0:

        return []  # Mənfi və ya 0 üçün boş siyahı qaytarılır

    elif n == 1:

        return [0]  # Yalnız ilk element

    elif n == 2:

        return [0, 1]  # İlk iki element

    fib = [0, 1]  # Fibonacci başlanğıcı

    for i in range(2, n):

        next\_num = fib[-1] + fib[-2]  # Son iki ədədi toplayırıq

        fib.append(next\_num)  # Yeni ədəd siyahıya əlavə olunur

    return fib  # Tam Fibonacci siyahısı qaytarılır

# Nümunə istifadə:

print("Fibonacci ardıcıllığı:", fibonacci(10))

Pass - biz çoxlu funksiyalar yazacayıqsa amma içini helə yazmayacayıqsa error verilməsin deye pass yazırıq yəni sonradan yaza bilək deye

**Task 70**

def count\_vowels(s):

    vowels = "aeiou"

    return sum(1 for char in s.lower() if char in vowels)

# Example usage

text = "Hello, World!"

print(count\_vowels(text))  # Output: 3

In Python, you can convert an integer to a string using the str() function.

Example:

num = 123

str\_num = str(num)

print(str\_num) # Output: '123'

print(type(str\_num)) # Output: <class 'str'>

**Task 71**

def average(numbers):

    if not numbers:  # Check if the list is empty

        return 0

    return sum(numbers) / len(numbers)  # Calculate the average

# Example usage

print(average([1, 2, 3, 4, 5]))  # Output: 3.0

print(average([]))  # Output: 0

**1.**Using the union() method:

The union() method returns a new set with all the elements from both sets, removing duplicates.

2.Using the | (pipe) operator:

The | operator is a shorthand for union(). It combines both sets in the same way.

3. Using the update() method:

The update() method modifies the first set by adding elements from the second set.

Key Notes:

union() and | create a new set without modifying the original sets.

update() modifies the first set in place.

**Task 72**

def is\_palindrome(s):

    cleaned\_s = ''.join(s.lower().split())  # Convert to lowercase and remove spaces

    return cleaned\_s == cleaned\_s[::-1]  # Compare with its reverse

# Example usage

print(is\_palindrome("Racecar"))        # Output: True

print(is\_palindrome("A man a plan a canal Panama"))  # Output: True

print(is\_palindrome("Hello"))          # Output: False

The split() function in Python is used to divide a string into a list of substrings based on a specified delimiter (default is whitespace).

Syntax:

string.split(separator, maxsplit)

separator (optional): The delimiter where the string will be split. Default is any whitespace.

maxsplit (optional): The maximum number of splits to perform. Default is -1 (no limit).

**Task 73**

def multiplication\_table(n):

    for i in range(1, 11):

        print(f"{n} x {i} = {n \* i}")

# Example usage

multiplication\_table(5)

/ Division: Performs floating-point (decimal) division, always returns a float.

// Floor Division: Performs integer division, rounding down to the nearest whole number.

**Task 74**

def merge\_and\_sort(list1, list2):

    merged\_list = list1 + list2  # Merge the lists

    return sorted(merged\_list)   # Sort the merged list

# Example usage

list1 = [3, 1, 4]

list2 = [5, 2, 6]

print(merge\_and\_sort(list1, list2))  # Output: [1, 2, 3, 4, 5, 6]

**Feature sort() sorted()**

**Type Method of lists Built-in function**

**Modifies Original List? ✅ Yes (modifies in-place) ❌ No (returns a new sorted list)**

**Returns None (modifies the list directly) A new sorted list**

**Usable on Other Iterables? ❌ No, only for lists ✅ Yes, works with lists, tuples, sets, dictionaries, etc.**

**Task 75**

def count\_unique\_elements(lst):

    return len(set(lst))  # Convert list to a set to remove duplicates, then get the length

# Example usage

print(count\_unique\_elements([1, 2, 3, 2, 4, 5, 1]))  # Output: 5

print(count\_unique\_elements([1, 1, 1]))  # Output: 1

1. index()

Returns the first occurrence index of a specified value in a sequence (string, list, tuple, etc.).Raises a ValueError if the value is not found.2. count()Returns the total number of occurrences of a specified value in a sequence.Does not raise an error if the value is not found; it simply returns 0