Цей блок коду реалізує функцію **format\_ingredients** і форматує список інгредієнтів відповідно до описаного вами формату.

1. Перша умова **if len(ingredients) == 0:** перевіряє, чи список інгредієнтів порожній (не має жодного елемента). Якщо ця умова вірна, функція повертає порожній рядок **""**, оскільки немає інгредієнтів для форматування.
2. Друга умова **elif len(ingredients) == 1:** перевіряє, чи список має тільки один елемент. Якщо ця умова вірна, функція повертає сам єдиний елемент списку **ingredients[0]**, оскільки немає необхідності розділяти єдиний елемент комами та союзом "and".
3. Останній блок **else:** виконується, якщо список має більше одного елемента. Тут ми використовуємо метод **.join()** для об'єднання всіх елементів списку, крім останнього. Вираз **", ".join(ingredients[:-1])** об'єднує елементи списку **ingredients** комами та пробілами, залишаючи останній елемент необробленим. Потім ми додаємо " and " та останній елемент **ingredients[-1]** до сформованого рядка. Отриманий відформатований рядок зберігається в змінній **formatted**, і цей рядок повертається з функції.

Цей умовний блок дозволяє правильно відформатувати список інгредієнтів незалежно від його довжини.

if len(ingredients) == 0:

return ""

elif len(ingredients) == 1:

return ingredients[0]

else:

formatted = ", ".join(ingredients[:-1]) + " and " + ingredients[-1]

return formatted

**Задание 6 из модуля 4**

def split\_list(grade):

if not grade:

return [], []

average = sum(grade) / len(grade)

lower\_part = [x for x in grade if x <= average]

upper\_part = [x for x in grade if x > average]

return lower\_part, upper\_part

**Модуль 4 задача 8**

def game(terra, power):

for sublist in terra:

for energy in sublist:

if energy <= power:

power += energy

else:

break

return power

**Модуль 4 задача 9**

def is\_valid\_pin\_codes(pin\_codes):

if len(pin\_codes) != len(set(pin\_codes)):

return False

if not pin\_codes:

return False

for pin in pin\_codes:

if len(pin) != 4 or not pin.isdigit():

return False

return True

**Модуль 4 задача 10**

Создаём пустую строку с именем **password**, в которой будем накапливать символы пароля.

pythonCopy code

for \_ in range(8):

Начинаем цикл, который будет выполняться 8 раз. Здесь **\_** используется, когда нам не нужно значение переменной из итерации (обычно так делают, когда не используют значение переменной в цикле).

pythonCopy code

random\_num = randint(40, 126)

Генерируем случайное целое число **random\_num** в диапазоне от 40 до 126 включительно. Это будет ASCII-кодом случайного символа.

pythonCopy code

password += chr(random\_num)

С помощью функции **chr()** преобразуем числовое значение **random\_num** в символ с соответствующим ASCII-кодом, а затем добавляем этот символ к строке **password**.

pythonCopy code

return password

После того как цикл завершится, возвращаем полученный пароль в качестве результата выполнения функции.

Таким образом, этот код генерирует случайный пароль из 8 символов, используя случайные ASCII-коды символов из диапазона 40-126, и возвращает этот пароль.

from random import randint

def get\_random\_password():

password = ""

for \_ in range(8):

random\_num = randint(40, 126)

password += chr(random\_num)

return password

**Модуль 4 задача 11**

def is\_valid\_password(password):

if len(password) != 8:

return False

has\_uppercase = False

has\_lowercase = False

has\_digit = False

for char in password:

if char.isupper():

has\_uppercase = True

elif char.islower():

has\_lowercase = True

elif char.isdigit():

has\_digit = True

return has\_uppercase and has\_lowercase and has\_digit

**Модуль 4 задание 12**

from random import randint

def get\_random\_password():

result = ""

count = 0

while count < 8:

random\_symbol = chr(randint(40, 126))

result = result + random\_symbol

count = count + 1

return result

def is\_valid\_password(password):

if len(password) != 8:

return False

has\_upper = False

has\_lower = False

has\_num = False

for ch in password:

if ch.isupper():

has\_upper = True

elif ch.islower():

has\_lower = True

elif ch.isdigit():

has\_num = True

return has\_upper and has\_lower and has\_num

def get\_password():

max\_attempts = 100

for \_ in range(max\_attempts):

password = get\_random\_password()

if is\_valid\_password(password):

return password

return None

**Модуль 5 задача 1**

def real\_len(text):

control\_chars = ['\n', '\f', '\r', '\t', '\v']

count = 0

for char in text:

if char not in control\_chars:

count += 1

return count

**Модуль 5 задача 2**

articles\_dict = [

{"title": "Endless ocean waters.",

"author": "Jhon Stark",

"year": 2019,},

{"title": "Oceans of other planets are full of silver",

"author": "Artur Clark",

"year": 2020,},

{"title": "An ocean that cannot be crossed.",

"author": "Silver Name",

"year": 2021,},

{"title": "The ocean that you love.",

"author": "Golden Gun",

"year": 2021,},

]

def find\_articles(key, letter\_case=False):

result = []

for article in articles\_dict:

author = article['author']

title = article['title']

year = article['year']

if not letter\_case:

author = author.lower()

title = title.lower()

key = key.lower()

if key in author or key in title:

result.append({'author': article['author'], 'title': article['title'], 'year': article['year']})

return result

разберем функцию **find\_articles** пошагово:

1. Определяем функцию **find\_articles** с двумя параметрами: **key** (искомый ключевой термин) и **letter\_case** (параметр, определяющий, нужно ли учитывать регистр букв).
2. Создаем пустой список **result**, в который будем сохранять найденные статьи.
3. С помощью цикла **for** перебираем каждую статью в списке **articles\_dict**.
4. Для каждой статьи из полученного словаря (**article**) получаем значения ключей: **author**, **title** и **year**.
5. Проверяем, учитывать ли регистр букв в поиске. Если **letter\_case** установлено в **False**, то мы преобразуем все буквы в нижний регистр для **author**, **title** и **key**, чтобы обеспечить регистронезависимый поиск.
6. Проверяем, содержит ли искомый ключевой термин **key** автора или название статьи. Если да, добавляем словарь с **author**, **title** и **year** в список **result**.
7. По завершении цикла возвращаем список **result**, который содержит словари с найденными статьями.
8. Пример использования показывает, как вы можете использовать эту функцию. Задан список **articles\_dict** с тремя статьями, а ключевой термин **search\_key** установлен как "Python". Результат поиска выводится на экран.

**Модуль 5 задание 6**

def is\_spam\_words(text, spam\_words, space\_around=False):

text\_lower = text.lower()

for spam in spam\_words:

spam\_lower = spam.lower()

if space\_around:

if f" {spam\_lower} " in f" {text\_lower} ":

return True

elif f"{spam\_lower}." in text\_lower:

return True

else:

if spam\_lower in text\_lower:

return True

return False

1. **text\_lower = text.lower()** - Мы приводим весь текст к нижнему регистру для обеспечения независимости от регистра при проверке.
2. Далее используем цикл **for spam in spam\_words:** для перебора запрещенных слов из списка **spam\_words**.
3. **spam\_lower = spam.lower()** - Аналогично приводим запрещенное слово к нижнему регистру.
4. **if space\_around:** - Проверяем, включен ли параметр **space\_around**.
5. В следующем блоке, если **space\_around** включен, мы проверяем наличие запрещенного слова с обеих сторон, окруженного пробелами, что означает, что оно стоит отдельно. Здесь мы используем **f" {spam\_lower} "** для проверки слова с пробелами вокруг него и **f"{spam\_lower}."** для проверки, стоит ли слово в начале предложения.
6. Если **space\_around** выключен, мы просто проверяем наличие запрещенного слова в тексте.
7. Если в предыдущих шагах было найдено запрещенное слово, функция возвращает **True**, что означает, что текст содержит запрещенное слово.
8. Если ни одно запрещенное слово не было найдено, функция возвращает **False**

**Модуль 5 задача 8**

grades = {"A": 5, "B": 5, "C": 4, "D": 3, "E": 3, "FX": 2, "F": 1}

Спочатку ви визначаєте словник **grades**, де кожний ключ (оцінка) відображається на відповідне числове значення (бал).

def formatted\_grades(students): Далі визначається функція **formatted\_grades**, яка приймає словник **students** з оцінками студентів.

formatted\_list = [] Ви створюєте порожній список **formatted\_list**, в який будуть додаватися відформатовані рядки таблиці.

for idx, (name, grade) in enumerate(students.items(), start=1):

formatted\_row = f"{idx:4d}|{name:<10}|{grade:^5}|{grades[grade]:^5}"

formatted\_list.append(formatted\_row)

return formatted\_list

Використовуючи цикл **for**, ви перебираєте пари **(name, grade)** (ім'я студента та його оцінка) зі словника **students**. Параметр **idx** відстежує номер рядка, починаючи з 1. У цьому циклі ви форматуєте рядок за допомогою f-рядка. Він містить наступні елементи:

* **{idx:4d}**: Номер рядка з шириною в 4 символи, вирівнювання по правому краю.
* **{name:<10}**: Ім'я студента з шириною в 10 символів, вирівнювання по лівому краю.
* **{grade:^5}**: Оцінка студента з шириною в 5 символів, вирівнювання по центру.
* **{grades[grade]:^5}**: Числове значення оцінки студента зі словника **grades** з шириною в 5 символів, вирівнювання по центру.

students = {"Nick": "A", "Olga": "B", "Mike": "FX", "Anna": "C"}

for el in formatted\_grades(students):

print(el)

**Модуль 5 задание 9**

def formatted\_numbers():

formatted\_list = []

header = "| decimal | hex | binary |"

formatted\_list.append(header)

for i in range(16):

decimal = i

hex\_value = format(i, 'x')

binary = format(i, 'b')

formatted\_row = f"|{decimal:<10}|{hex\_value:^10}|{binary:>10}|"

formatted\_list.append(formatted\_row)

return formatted\_list

for el in formatted\_numbers():

print(el)

**Модуль 5 задание 10**

import re

def find\_word(text, word):

result = re.search(word, text)

if result:

first\_index = result.start()

last\_index = result.end()

search\_string = result.group()

else:

first\_index = None

last\_index = None

search\_string = word

result\_dict = {

'result': result is not None,

'first\_index': first\_index,

'last\_index': last\_index,

'search\_string': search\_string,

'string': text

}

return result\_dict

**Модуль 5 задание 11**

import re

def find\_all\_words(text, word):

pattern = re.compile(rf'(?i){re.escape(word)}', re.IGNORECASE)

results = pattern.finditer(text)

found\_results = [result.group() for result in results]

return found\_results

1. **pattern = re.compile(rf'(?i){re.escape(word)}', re.IGNORECASE)**:
   * Создаём компилированный регулярный выражение (**pattern**), которое ищет соответствующее слово, игнорируя регистр (за счёт **(?i)**).
   * **re.escape(word)** экранирует любые специальные символы в слове, если они есть.
   * Добавляем модификатор **re.IGNORECASE** для игнорирования регистра при поиске.
2. **results = pattern.finditer(text)**:
   * Используем созданное компилированное регулярное выражение для поиска всех вхождений соответствующего слова в тексте.
   * **pattern.finditer** возвращает итератор, который содержит объекты **Match** для каждого совпадения.
3. **found\_results = [result.group() for result in results]**:
   * Проходимся по итератору результатов поиска.
   * Извлекаем текст совпадения из каждого объекта **Match** с помощью **result.group()**.
   * Сохраняем все найденные текстовые совпадения в списке **found\_results**.
4. **return found\_results**:
   * Возвращаем список с найденными текстовыми совпадениями слова, игнорируя регистр.

Модуль 5 задача 12

import re

def replace\_spam\_words(text, spam\_words):

pattern = re.compile(rf'(?i)({"|".join(map(re.escape, spam\_words))})')

def replace\_match(match):

return '\*' \* len(match.group())

new\_text = pattern.sub(replace\_match, text)

return new\_text

1. Создать регулярное выражение для поиска всех форм запрещенных слов в тексте:
   * Собрать паттерн с использованием строкового формата и передать список **spam\_words** в виде альтернатив (**|**).
   * Добавить флаг **re.IGNORECASE**, чтобы игнорировать регистр при поиске.
2. Создать функцию для замены совпадений на шаблон звездочек:
   * Функция будет принимать объект **Match** и возвращать строку, состоящую из звездочек (**\***) той же длины, что и найденное слово.
3. Использовать **re.sub** для замены запрещенных слов в тексте:
   * Передать **pattern** из шага 3, функцию замены из шага 4 и текст для проверки и замены.
4. Вернуть новый текст с замененными запрещенными словами.

**Это детальное разъяснение: pattern = re.compile(...)**: Эта строка создает скомпилированный шаблон регулярного выражения с помощью функции **re.compile()**. Скомпилированный шаблон более эффективен для множественного использования одного и того же шаблона.

1. **rf'...'**: Это сырая форматированная строка (raw f-string). **r** перед строкой указывает на сырую строку (без управляющих последовательностей), а **f** перед строкой указывает на форматированную строку. F-строки позволяют вставлять выражения внутрь строковых литералов.
2. **(?i)**: Это флаг в регулярном выражении, который задает режим. В данном случае **(?i)** устанавливает режим регистронезависимости, так что шаблон будет соответствовать независимо от того, являются ли буквы заглавными или строчными.
3. **({"|".join(map(re.escape, spam\_words))})**: Эта часть интереснее всего. Разберем ее детально:
   * **spam\_words**: По всей видимости, это список (или итерируемый объект) с плохими словами или фразами, которые вы хотите найти в тексте.
   * **map(re.escape, spam\_words)**: Функция **map()** применяет функцию **re.escape()** к каждому слову в **spam\_words**. **re.escape()** используется для экранирования любых специальных символов в словах, чтобы они рассматривались как буквальные символы в регулярном выражении.
   * **("|".join(...))**: Эта часть объединяет экранированные плохие слова с помощью символа **|** (вертикальная черта) в качестве разделителя. Так создается шаблон, который будет соответствовать любому из плохих слов.
   * **({...})**: Весь выражение внутри фигурных скобок формирует захватывающую группу, что позволяет сгруппировать весь шаблон.

Таким образом, окончательный шаблон фактически становится похожим на **(?i)(плохое\_слово\_1|плохое\_слово\_2|плохое\_слово\_3|...)**, где **плохое\_слово\_1**, **плохое\_слово\_2** и так далее — это экранированные и объединенные плохие слова.

В итоге данная строка кода создает скомпилированный шаблон регулярного выражения, который будет соответствовать любому из плохих слов, указанных в списке, игнорируя регистр символов. Это мощный способ эффективно искать несколько плохих слов в заданном тексте.

**Модуль 5 задача 13**

import re

def find\_all\_emails(text):

result = re.findall(r'[A-Za-z][A-Za-z0-9.\_%+-]{1,}@[A-Za-z]+\.[A-Za-z]{2,}\b', text)

return result

**Модуль 6 задача 1**

1. **total = 0.0**
   * Инициализация переменной **total** значением 0.0, которое будет использоваться для подсчета общей суммы заработной платы.
2. **file = open(path, "r")**
   * Открытие файла по указанному пути в режиме чтения ("r") и сохранение объекта файла в переменную **file**.
3. **line = file.readline()**
   * Считывание первой строки из файла с помощью метода **readline()** и сохранение её в переменную **line**.
4. **while line:**
   * Начало цикла **while**, который будет выполняться, пока переменная **line** содержит какую-либо строку (файл не достиг конца).
5. **name, salary = line.strip().split(",")**
   * Разделение строки **line** на две части с использованием **,** в качестве разделителя. Метод **strip()** удаляет лишние пробелы в начале и конце строки. Результаты разделения (имя и зарплата) присваиваются переменным **name** и **salary**.
6. **total += float(salary)**
   * Добавление зарплаты, преобразованной в тип **float**, к переменной **total**, которая накапливает общую сумму.
7. **line = file.readline()**
   * Считывание следующей строки из файла.
8. **file.close()**
   * Закрытие файла, чтобы освободить ресурсы системы.
9. **return total**
   * Возврат вычисленной общей суммы заработной платы.

def total\_salary(path):

total = 0.0

file = open(path, "r")

line = file.readline()

while line:

name, salary = line.strip().split(",")

total += float(salary)

line = file.readline()

file.close()

return total

**Модуль 6 задача 5**

def get\_cats\_info(path):

with open(path, 'r') as fh:

lines = fh.readlines()

employee\_list = []

for line in lines:

parts = line.strip().split(',')

employee\_dict = {

"id": parts[0],

"name": parts[1],

"age": parts[2]

}

employee\_list.append(employee\_dict)

return employee\_list

1. Відкриваємо файл за допомогою менеджера контексту **with** для читання.
2. Зчитуємо всі рядки файлу за допомогою **fh.readlines()**.
3. Використовуємо цикл **for** для обробки кожного рядка.
4. Використовуємо **strip()** для видалення зайвих пробілів та символів нового рядка з початку та кінця рядка.
5. Використовуємо **split(',')** для розділення рядка на складові по комі.
6. Створюємо словник **employee\_dict** з даними про котов.
7. Додаємо цей словник до списку **employee\_list**.
8. Повертаємо список котов.
9. Вказуємо шлях до файлу у змінній **file\_path**.
10. Викликаємо функцію **read\_employees\_from\_file** для читання файлу та збереження списку котов у змінну **employees**.
11. Виводимо список котов на екран.

**Модуль 6 задание 7**

1. **def sanitize\_file(source, output):**
   * Мы определяем функцию **sanitize\_file**, которая принимает два аргумента: **source** (путь к исходному файлу) и **output** (путь к выходному файлу).
2. **with open(source, 'r') as source\_file:**
   * Мы открываем исходный файл **source** для чтения и используем контекстный менеджер **with**, чтобы автоматически закрыть файл после завершения работы с ним. Результат открытия файла сохраняется в переменной **source\_file**.
3. **text = source\_file.read():**
   * Мы читаем всё содержимое исходного файла и сохраняем его в переменной **text**.
4. **cleaned\_text = ''.join(char for char in text if not char.isdigit()):**
   * Мы создаем переменную **cleaned\_text**, в которой будем хранить текст, очищенный от цифр. Мы проходим по каждому символу **char** в переменной **text** с помощью генератора списка и проверяем, не является ли символ цифрой с помощью **char.isdigit()**. Если символ не является цифрой, мы добавляем его к **cleaned\_text**.
5. **with open(output, 'w') as output\_file:**
   * Мы открываем выходной файл **output** для записи, также используя контекстный менеджер.
6. **output\_file.write(cleaned\_text):**
   * Мы записываем очищенный текст **cleaned\_text** в выходной файл **output**.
7. Заключительный код показывает, как использовать функцию **sanitize\_file**. Вам нужно заменить значения **source\_file\_path** и **output\_file\_path** на пути к вашим собственным файлам.

def sanitize\_file(source, output):

with open(source, 'r') as source\_file:

text = source\_file.read()

cleaned\_text = ''.join(char for char in text if not char.isdigit())

with open(output, 'w') as output\_file:

output\_file.write(cleaned\_text)

**Модуль 6 задача 10**

def save\_credentials\_users(path, users\_info):

try:

with open(path, 'wb') as file:

for username, password in users\_info.items():

# Формуємо рядок у форматі "username:password" та записуємо його в файл

line = f"{username}:{password}\n"

file.write(line.encode('utf-8'))

print("Інформація про користувачів успішно збережена у файлі:", path)

except Exception as e:

print("Помилка під час запису в файл:", str(e))

1. **try:** - Починаємо конструкцію "спроба-виняток", оскільки може виникнути помилка при роботі з файлами.
2. **with open(path, 'wb') as file:** - Відкриваємо файл за шляхом, який отримали у параметрі **path**, у бінарному режимі для запису. Ця конструкція **with** гарантує, що файл автоматично закриється після завершення роботи з ним.
3. **for username, password in users\_info.items():** - Перебираємо словник **users\_info**, розділяючи пари користувач-пароль на змінні **username** і **password**.
4. **line = f"{username}:{password}\n"** - Формуємо рядок **line**, який містить ім'я користувача і пароль, розділені двокрапкою і з символом нового рядка (**\n**) на кінці.
5. **file.write(line.encode('utf-8'))** - Записуємо рядок **line** в файл. Оскільки ми відкрили файл у бінарному режимі (**'wb'**), ми повинні сконвертувати рядок у байти, використовуючи метод **.encode('utf-8')**, перед записом в файл.
6. **print("Інформація про користувачів успішно збережена у файлі:", path)** - Виводимо повідомлення про успішне завершення операції разом зі шляхом до файлу.
7. **except Exception as e:** - Починаємо обробку помилок. Якщо під час виконання блоку **try** виникає помилка, вона буде оброблена у цьому блоку.
8. **print("Помилка під час запису в файл:", str(e))** - Виводимо повідомлення про помилку та текст помилки, який міститься у змінній **e**.

Начало формы

**Модуль 6 задание 11**

def get\_credentials\_users(path):

try:

with open(path, 'rb') as file:

lines = [line.decode('utf-8').strip() for line in file.readlines()]

return lines

except Exception as e:

print("Помилка під час читання файлу:", str(e))

return []

1. **def get\_credentials\_users(path):** - Это объявление функции **get\_credentials\_users**, которая принимает один аргумент **path**, представляющий путь к бинарному файлу.
2. **try:** - Мы начинаем конструкцию "попробовать-исключение", так как в работе с файлами могут возникнуть ошибки.
3. **with open(path, 'rb') as file:** - Мы открываем файл по указанному пути в бинарном режиме для чтения (**'rb'**). С использованием конструкции **with** файл будет автоматически закрыт после выхода из контекста.
4. **lines = [line.decode('utf-8').strip() for line in file.readlines()]** - Мы считываем все строки из файла, декодируем их из байтового формата в строки с кодировкой utf-8 и удаляем лишние пробелы и символы перевода строки. Полученные строки сохраняются в переменной **lines** в виде списка.
5. **return lines** - Мы возвращаем список строк **lines**, который содержит информацию из файла.
6. **except Exception as e:** - Мы начинаем обработку исключений. Если во время выполнения блока **try** возникает ошибка, она будет обработана в этом блоке.
7. **print("Помилка під час читання файлу:", str(e))** - Мы выводим сообщение об ошибке и текст ошибки, который находится в переменной **e**.
8. **return []** - Мы возвращаем пустой список в случае ошибки, так как не удалось прочитать файл.