МІНІСТЕРСТВО ОСВІТИ ТА НАУКИ УКРАЇНИ

ЧЕРКАСЬКИЙ ДЕРЖАВНИЙ БІЗНЕС-КОЛЕДЖ

***Кафедра комп’ютерної інженерії та інформаційних технологій***

***Відділення інженерії програмного забезпечення***

**Звіт з теми № 2**

з курсу «Тренінг-курс з основ програмування»

Малишевої Валерії Олексіївни

студентки групи **2П-23**

номер у списку підгрупи - 13

Викладач Марченко С. В.

Черкаси-2025

**1. Арифметичні операції з двійковими числами**Завдання:   
Виконати арифметичні операції: додавання, віднімання, множення, ділення   
для чисел 993 та 678 у двійковій системі.  
Розв'язання у репозиторії у файлі - rozrahynku.txt  
  
**2. (Дробові числа)** Представити дробове число одинарної та подвійної точності в двійковій систем  
Розв'язання у репозиторії у файлі - rozrahynku.txt

**3. (Колір у двійковій системі)** Представити колір у двійковій системі  
Розв'язання у репозиторії у файлі - rozrahynku.txt

**4. (Текст у двійковій системі)** Представити текст, закодований за допомогою таблиці ASCII, у двійковій системі.   
Розв'язання у репозиторії у файлі - rozrahynku.txt

**5. (Зчитування та нормалізація аудіо даних (модуль wave))** Підготуйте та занесіть до звіту коротку інформацію про WAV-формат та ознайомтесь з документацією модуля wave.  
**Лістинг коду до завдання 5**:  
**Імпорт бібліотек:**

import wave

import numpy as np

import matplotlib.pyplot as plt

**Лістинг 1.2. Читання метаданих WAV-файлу**

with wave.open('Nikow.wav', 'rb') as wav\_file:

print("Кількість каналів: ", wav\_file.getnchannels())

print("Частота дискретизації: ", wav\_file.getframerate())

print("Кількість фреймів: ", wav\_file.getnframes())

print("Бітрейт: ", wav\_file.getsampwidth() \* 8)

**Лістинг 1.3. Читання перших 100 фреймів**

with wave.open('Nikow.wav', 'rb') as wav\_file:

frames = wav\_file.readframes(100)

print("Перші 100 фреймів: ", frames)

Лістинг 1.4. Читання всіх фреймів та перетворення в масив

with wave.open('Nikow.wav', 'rb') as wav\_file:

frames = wav\_file.readframes(-1)

signal = np.frombuffer(frames, dtype=np.int16)

print("Амплітуди: ", signal[:100])

Лістинг 1.5. Побудова графіка звукової хвилі

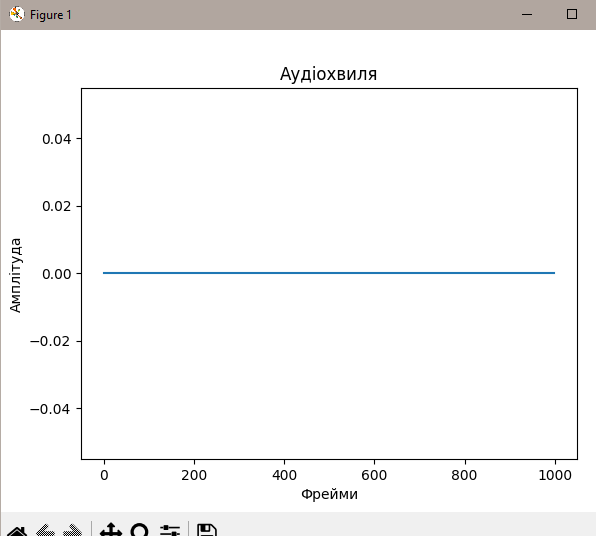
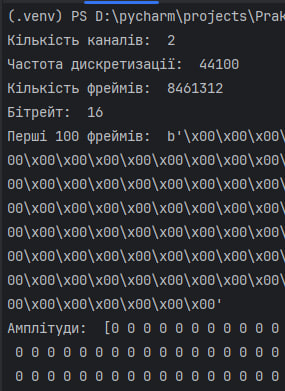
plt.plot(signal[:1000])

plt.title("Аудіохвиля")

plt.xlabel("Фрейми")

plt.ylabel("Амплітуда")

plt.show()



Результат: Виведено метадані WAV-файлу. Візуалізовано аудіохвилю за допомогою matplotlib

6. (Абстрактне синтаксичне дерево (модуль ast)) Напишіть програму, яка аналізує Python-код за допомогою модуля ast, виконує кілька завдань одночасно, і генерує зведений звіт. Програма повинна: 1) Вивести перелік усіх функцій із їхніми аргументами. 2) Підрахувати кількість умов (if), циклів (for, while) та операторів присвоєння. 3) Знайти змінні, які не відповідають стилю snake\_case (тільки маленькі літери, цифри та \_ в ідентифікаторі).

**Лістинг 2.1 Імпорт модуля ast**

import ast

**Лістинг 2.2 Клас Analiz\_Cody для аналізу коду**

class Analiz\_Cody(ast.NodeVisitor):

def \_\_init\_\_(self):

self.funkcii = []

self.rahynok\_ifok = 0

self.rahynok\_forok = 0

self.rahynok\_while = 0

self.rahynok\_operatoriv\_prusvoennya = 0

**Лістинг 2.3 Метод visit\_FunctionDef для аналізу функцій**

def visit\_FunctionDef(self, node):

func\_name = node.name

args = [arg.arg for arg in node.args.args]

self.funkcii.append((func\_name, args))

self.generic\_visit(node)

**Лістинг 2.4 Метод visit\_If для аналізу умовних операторів**

def visit\_If(self, node):

self.rahynok\_ifok += 1

self.generic\_visit(node)

**Лістинг 2.5 Метод visit\_For для аналізу циклів for**

def visit\_For(self, node):

self.rahynok\_forok += 1

self.generic\_visit(node)

**Лістинг 2.6 Метод visit\_While для аналізу циклів while**

**def visit\_While(self, node):**

**self.rahynok\_while += 1**

**self.generic\_visit(node)**

**Лістинг 2.7 Метод visit\_Assign для аналізу операторів присвоєння**

**def visit\_Assign(self, node):**

**self.rahynok\_operatoriv\_prusvoennya += 1**

**self.generic\_visit(node)**

**Лістинг 2.8 Функція analyze\_code для аналізу коду**

**def analyze\_code(code):**

**tree = ast.parse(code)**

**analiz = Analiz\_Cody()**

**analiz.visit(tree)**

**print("Функції та їх аргументи:")**

**for funciya\_name, args in analiz.funkcii:**

**print(f"Функція: {funciya\_name}, Аргументи: {args}")**

**print("\nКількість умовних операторів (if):", analiz.rahynok\_ifok)**

**print("Кількість циклів for:", analiz.rahynok\_forok)**

**print("Кількість циклів while:", analiz.rahynok\_while)**

**print("Кількість операторів `=` :", analiz.rahynok\_operatoriv\_prusvoennya)**

**Лістинг 2.9 Приклад Python-коду для аналізу**

**cod = """**

**def funciya(a, b):**

**x = 10**

**if a > b:**

**print("a більше b")**

**else:**

**print("b більше а")**

**for i in range(10):**

**print(i)**

**while x > 0:**

**x -= 1**

**def insha\_funciya(c):**

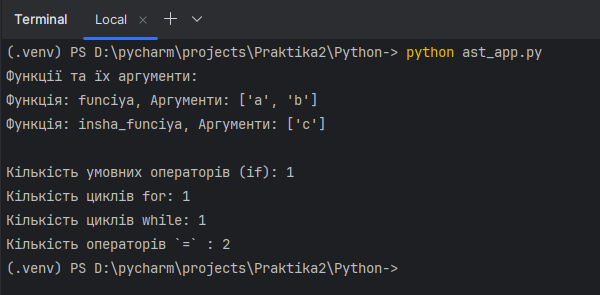
**y = c \* 2**

**return y**

**"""**

**Лістинг 2.10 Виклик функції analyze\_code**

**analyze\_code(cod)**

****

Результат:Виведено інформацію про функції та їх аргументи.Підраховано кількість умовних операторів, циклів та операторів присвоєння.

7. (Байткод Python) Розгляньте та спробуйте відтворити методики обфускації байткоду Python, наведені в статті «Python Opcode Obfuscation: A Powerful Anti-Analysis Technique». Занесіть до звіту власні пояснення поняття обфускації, потреби в ній та запропоновані способи реалізації з демонстрацією, аналогічною до дій зі статті.  
import dis

import types

import marshal

def add\_numbers(a, b):

return a + b

print("Оригінальний байткод функції add\_numbers:")

dis.dis(add\_numbers)

original\_code = add\_numbers.\_\_code\_\_

new\_code = original\_code.co\_code.replace(b'\x17', b'\x37')

new\_code\_obj = types.CodeType(

original\_code.co\_argcount,

original\_code.co\_posonlyargcount,

original\_code.co\_kwonlyargcount,

original\_code.co\_nlocals,

original\_code.co\_stacksize,

original\_code.co\_flags,

new\_code,

original\_code.co\_consts,

original\_code.co\_names,

original\_code.co\_varnames,

original\_code.co\_filename,

original\_code.co\_name,

original\_code.co\_firstlineno,

original\_code.co\_lnotab,

original\_code.co\_freevars,

original\_code.co\_cellvars

)

add\_numbers.\_\_code\_\_ = new\_code\_obj

print("\nЗмінений байткод функції add\_numbers:")

dis.dis(add\_numbers)

result = add\_numbers(10, 20)

print("\nРезультат виклику функції після обфускації:", result)

serialized\_code = marshal.dumps(new\_code\_obj)

with open("obfuscated\_code.bin", "wb") as f:

f.write(serialized\_code)

with open("obfuscated\_code.bin", "rb") as f:

loaded\_code = marshal.loads(f.read())

def loaded\_add\_numbers(a, b):

return a + b

loaded\_add\_numbers.\_\_code\_\_ = loaded\_code

loaded\_result = loaded\_add\_numbers(30, 40)

print("\nРезультат виклику функції з завантаженого байткоду:", loaded\_result)

result = add\_numbers(10, 20)

print("\nРезультат виклику функції після обфускації:", result)

serialized\_code = marshal.dumps(new\_code\_obj)

with open("obfuscated\_code.bin", "wb") as f:

f.write(serialized\_code)

with open("obfuscated\_code.bin", "rb") as f:

loaded\_code = marshal.loads(f.read())

def loaded\_add\_numbers(a, b):

return a + b

loaded\_add\_numbers.\_\_code\_\_ = loaded\_code

loaded\_result = loaded\_add\_numbers(30, 40)

print("\nРезультат виклику функції з завантаженого байткоду:", loaded\_result)

Результат: Виведено оригінальний та змінений байткод. Результат виклику функції після обфускації: 30. Результат виклику функції з завантаженого байткоду: 70.

Висновок:  
У ході виконання практичної роботи було вивчено роботу з двійковими числами, аудіо даними, байткодом Python та іншими аспектами програмування.   
Автор програми: Малишева Валерія Олексіївна  
Час компіляції: 01:53 (12/02/2025)  
  
(Якщо треба більш поширене пояснення кодів,то скажіть і я додам сюди пояснення кожної строчки)