НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ
«КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ
ІМЕНІ ІГОРЯ СІКОРСЬКОГО»

Кафедра мікроелектроніки

ЗВІТ

до лабораторної роботи № 1

з дисципліни **«ТЕОРІЯ СИГНАЛІВ»**

на тему: «Основи програмування мовою Python»

Студента 3 курсу ОС «Бакалавр»
групи ДП -82
зі спеціальності
153 «Мікро- та наносистемна техніка»
Лищенкл Б.В
(прізвище та ініціали)

Викладач: Порєва Г. С.

Оцінка: _____

Кількість балів: _____

Підпис: _____

Київ - 2020 рік

Лабораторна робота № 1

Основи програмування мовою Python

1. МЕТА РОБОТИ

Ознайомлення з основами програмування мовою Python на прикладі використання стандартних функцій, побудови файлів-сценаріїв та створення функцій користувача (на прикладі розв’язку системи рівнянь моделі «хижак-жертва»).

2. ЗАВДАННЯ

1. Ознайомитися з типами даних та представленням змінних в Python.

- з правилами введення змінних та називання змінних;

- з операціями над числами та матрицями. Вивчити матричні та поелементні операції над матрицями.

3. Ознайомитися:

- з задаванням масиву, елементи якого є арифметичною послідовністю;

- з роботою функцій генерації випадкових чисел із заданими густинами розподілу імовірності. Ознайомитися з функцією побудови гістограм, побудувати гістограми випадкових чисел з різними розподілами густини ймовірності;

4. Ознайомитися з написанням власних файлів-сценаріїв. У власному файлі-сценарії побудувати графік лінійної функції однієї змінної. Позначити вісі та заголовок графіку, нанести координатну сітку.

5. Написати файл-сценарій, в якому:

5.1 побудувати графіки синусоїд частот 1, 10, 50 Гц. Тривалість сигналів – 1 сек., частота дискретизації 256 Гц. Графіки будувати в одному вікні, але в різних осях. Амплітуди кожної синусоїди повинні бути випадковими числами;

5.2 виконати теж саме, але задавати амплітуду кожної синусоїди з клавіатури;

5.3 підписати заголовок кожного графіку текстом, який буде містити значення частоти та амплітуди відповідної синусоїди.

6. Ознайомитися з роботою функцій, що генерують прямокутні імпульси, гаусівські імпульси, трикутні імпульси та послідовності імпульсів заданої форми.

6.1 Побудувати одиночний прямокутний імпульс. Задати проміжок значень часу 10 секунд, частота дискретизації 256 Гц. Побудувати графік одиничного прямокутного імпульсу шириною 300 мс, з центром в момент часу 4 с.

6.2 Написати файл-сценарій для побудови графіку прямокутного імпульсу, тривалість та амплітуда якого буде задаватися з клавіатури. Розташування імпульсу задавати випадковим числом, але передбачити перевірку, чи не виходе імпульс за межі графіка;

6.3 Побудувати послідовність прямокутних імпульсів для двох випадків: а) коли інтервали між імпульсами однакові, б) коли інтервали між імпульсами випадкові і задаються програмно.

7. Зберегти дані розрахунку функції в файл. Прочитати їх із файлу в іншому сценарії, побудувати графік функції

8. Побудувати власний файл-функцію для побудови графіка синусоїдального сигналу із заданою частотою, амплітудою та тривалістю для частоти дискретизації 256 Гц. В якості вихідного параметру функції вивести середнє значення синусоїди.

3. Порядок роботи, частина 2*

1. Ознайомитися з написанням власних функцій користувача в Python. Ознайомитись з роботою та параметрами функції для розв'язання систем диференціальних рівнянь (пропонується використовувати методи 4-5 порядку).

2. Розв'язати систему рівнянь, яка описує співіснування людей, зомбі та тимчасово померлих, використовуючи метод Рунге-Кутти та отримати залежності від часу кількості зомбі, людей та жертв. Вивести графіки в часовій області. Отримати розв'язки та візуально їх проаналізувати для таких варіантів:

- початкова кількість людей набагато більша за зомбі;
- початкова кількість людей набагато менша за зомбі;
- різні комбінації коефіцієнтів здатності людини боротися із зомбі (альфа), та коефіцієнту зомбіфікації (бета).

3. Знайти програмними засобами перший момент часу, в який кількість зомбі дорівнює кількості людей. Побудувати залежність величини цього часу від параметру β. Передбачити можливість, при якій не буде точного співпадіння кількостей, але графіки перетинаються.

4. Промоделювати «оптимістичний» сценарій, при якому існують ліки від зомбіфікації. В моделі треба передбачити можливість для зомбі перетворюватися знову на людей (параметр c). Втім, такі ліки не дають імунітету, отже, зомбі, які стали людьми, можуть знову стати зомбі. Побудувати графіки для різних коефіцієнтів c, виконати візуальний аналіз результатів.

4. Хід роботи

1. Код до пункут 1-3.1:

```
# 1. Ознайомитися з типами даних та представленням змінних в Python.
# - з правилами введення змінних та називання змінних;

#   Числа
a = 2
# змінна a являється классом типу 'int'
print(a, "is of type", type(a))
b = 3.0
# змінна b являється классом типу 'float'
print(b, "is of type", type(a))
c = 1+3j
```

```
# змінна c являється классом типу 'complex'
print(c, "is of type", type(c))

#   Списки (Масиви)
arr = [1, 1.1, 'Python']
# змінна arr являється классом типу 'list'
print(arr, "is of type", type(arr))

#   Кортежі
t = (5, 'program', 1+3j)
# змінна t являється классом типу 'tuple'
print(t, "is of type", type(t))

#   Рядки
s = " 'Hello World!' "
# змінна s являється классом типу 'string'
print(s, "is of type", type(s))

#   Множини
m = {5, 2, 3, 1, 4}
# змінна m являється классом типу 'set'
print(m, "is of type", type(m))

#   Словарі (для мене обєкти)
obj = {1: 'value', 'key': 2}
# змінна obj являється классом типу 'dict'
print(obj, "is of type", type(obj))

# - з операціями над числами та матрицями.
print('\n', 'Операції над числами та матрицями')
print('6+2=', 6 + 2)  # додавання
print('6-2=', 6 - 2)  # віднімання
print('6*2=', 6 * 2)  # множення
print('7/2=', 7 / 2)  # ділення
print('7//2=', 7 // 2)  # цілочисленне ділення
print('7**3=', 7**3)  # зведення в ступінь
print('7%2=', 7 % 2)  # залишок від ділення

#   Арифметичні операції з присвоєнням
print('\n', 'Арифметичні операції з присвоєнням')
number = 10
number += 5
print('10+=5:', number)  # 15

number -= 3
print('15-=3:', number)  # 12

number %= 4
print('12%=4:', number)  # 0

#  Полетіли...Вивчити матричні та поелементні операції над матрицями.

import numpy as np  # імпортуємо бібліотеку NumPy для роботи з матрицями

# Способи створення масивів
A = np.array([[1, 2, 3], [4, 5, 6]])  # перетворюємо список в масив
B = A.copy()  # копіюємо існуючий масив
Z = np.zeros((2, 3), 'complex')  # створення нульового масива заданого розміру з певним
типом данних
Ons = np.ones((3, 2), 'str')  # створення одиничного масива заданого розміру з певним
типом данних
D = np.zeros_like(A, 'int')  # створення нульового масива заданого розміру з існуючного
масиву
```

```
E = np.eye(3)  # одинична діагональна матриця
# побудувати масив чисел від From (включаючи) до To (не включаючи) з кроком Step
From = 2.5
To = 7
Step = 0.5
Ranged = np.arange(From, To, Step)
# Використовуючи метод astype, можна привести масив до іншого типу.
T = A.astype('str')
# Багатовимірний масив можна представити як одновимірний масив максимальної довжини,
# нарізаний на фрагменти по довжині самої останньої осі і покладений шарами по осях,
починаючи з останніх.
ArrayTF = np.arange(24)
Layer1 = ArrayTF.reshape(4, 6)
Layer2 = ArrayTF.reshape(4, 3, 2)
print('Layer1\n', Layer1)
print('\nLayer2\n', Layer2)

# Розмірність масиву (кількість осей) - поле ndim (число)
# Розмір уздовж кожної осі - shape (кортеж) або shape.
# Повна кількість елементів в масиві size:
print(Layer2.ndim, Layer2.shape, len(Layer2.shape), ArrayTF.size)

#   Траспонування
print('\nТранспонування\n')
ArrSample = np.array([[1, 2, 3], [4, 5, 6]])
print('ArrSample\n', ArrSample)
print('\nArrSample data\n', ArrSample.ravel())

B_transposition = ArrSample.T
print('\nB_transposition\n', B_transposition)
print('\nB_transposition data\n', B_transposition.ravel())

#   Об'єднання матриць
A = np.ones((2, 1, 2))
B = np.zeros((2, 3, 2))

C = np.concatenate((A, B), 1)
print('\n Конкатинація матриць:\n', A, '\n', B, '\n\n=', C.shape)
#   Додавання матриць
ArrSum = ArrSample + ArrSample
print('\n Додавання матриць:\n', ArrSample, '\n+\n', ArrSample, '\n=\n', ArrSum)
#   Математичні операції над елементами масиву
A = np.array([[-1., 2., 3.], [4., 5., 6.], [7., 8., 9.]])
B = np.array([[1., -2., -3.], [7., 8., 9.], [4., 5., 6.], ])

C = A + B
D = A - B
E = A * B
F = A / B
G = A ** B
print('\n Математичні операції над елементами масиву\n')
print('+\n', C, '\n')
print('-\n', D, '\n')
print('*\n', E, '\n')
print('/\n', F, '\n')
print('**\n', G, '\n')
# Поелементне множення матриць
ElemArr1 = np.array([1, 2, 3])
ElemArr2 = np.array([4, 5, 6])
ElemDoubl = ElemArr1 @ ElemArr2
print(ElemArr1, '@', ElemArr2, '=', ElemDoubl, '\n')
ElemDoubl2 = np.multiply(ElemArr1, ElemArr2)
print(ElemArr1, '*', ElemArr2, '=', ElemDoubl2, '\n')
```

```
#  Задаванням масиву, елементи якого є арифметичною послідовністю;
print('Задаванням масиву, елементи якого є арифметичною послідовністю\n')
From = int(input('Start at:'))
To = int(input('End at:'))
Step = int(input('Step:'))
sequenceArithmetic = np.arange(From, To, Step)
print(sequenceArithmetic)
```

Консоль лог:

```
"E:\Теорія Сигналів\PythonLabs\venv\Scripts\python.exe" "E:/Теорія Сигналів/PythonLabs/Laba1/index.py"
2 is of type <class 'int'>
3.0 is of type <class 'int'>
(1+3j) is of type <class 'complex'>
[1, 1.1, 'Python'] is of type <class 'list'>
(5, 'program', (1+3j)) is of type <class 'tuple'>
 'Hello World!'  is of type <class 'str'>
{1, 2, 3, 4, 5} is of type <class 'set'>
{1: 'value', 'key': 2} is of type <class 'dict'>

 Операції над числами та матрицями
6+2= 8
6-2= 4
6*2= 12
7/2= 3.5
7//2= 3
7**3= 343
7%2= 1
  Арифметичні операції з присвоєнням
 10+=5: 15
 15-=3: 12
 12%=4: 0
 Layer1
  [[ 0  1  2  3  4  5]
  [ 6  7  8  9 10 11]
  [12 13 14 15 16 17]
  [18 19 20 21 22 23]]

 Layer2
  [[[ 0  1]
   [ 2  3]
   [ 4  5]]

  [[ 6  7]
   [ 8  9]
   [10 11]]

  [[12 13]
   [14 15]
   [16 17]]

  [[18 19]
   [20 21]
   [22 23]]]
 3 (4, 3, 2) 3 24
```

```
Транспонування

ArrSample
 [[1 2 3]
 [4 5 6]]

ArrSample data
 [1 2 3 4 5 6]

B_transposition
 [[1 4]
 [2 5]
 [3 6]]

B_transposition data
 [1 4 2 5 3 6]

 Конкатинація матриць:
 [[[1. 1.]]

 [[1. 1.]]]
 [[[0. 0.]
  [0. 0.]
  [0. 0.]]

 [[0. 0.]
  [0. 0.]
  [0. 0.]]]

= (2, 4, 2)

 Додавання матриць:
 [[1 2 3]
 [4 5 6]]
+
 [[1 2 3]
 [4 5 6]]
=
 [[ 2  4  6]
 [ 8 10 12]]
```

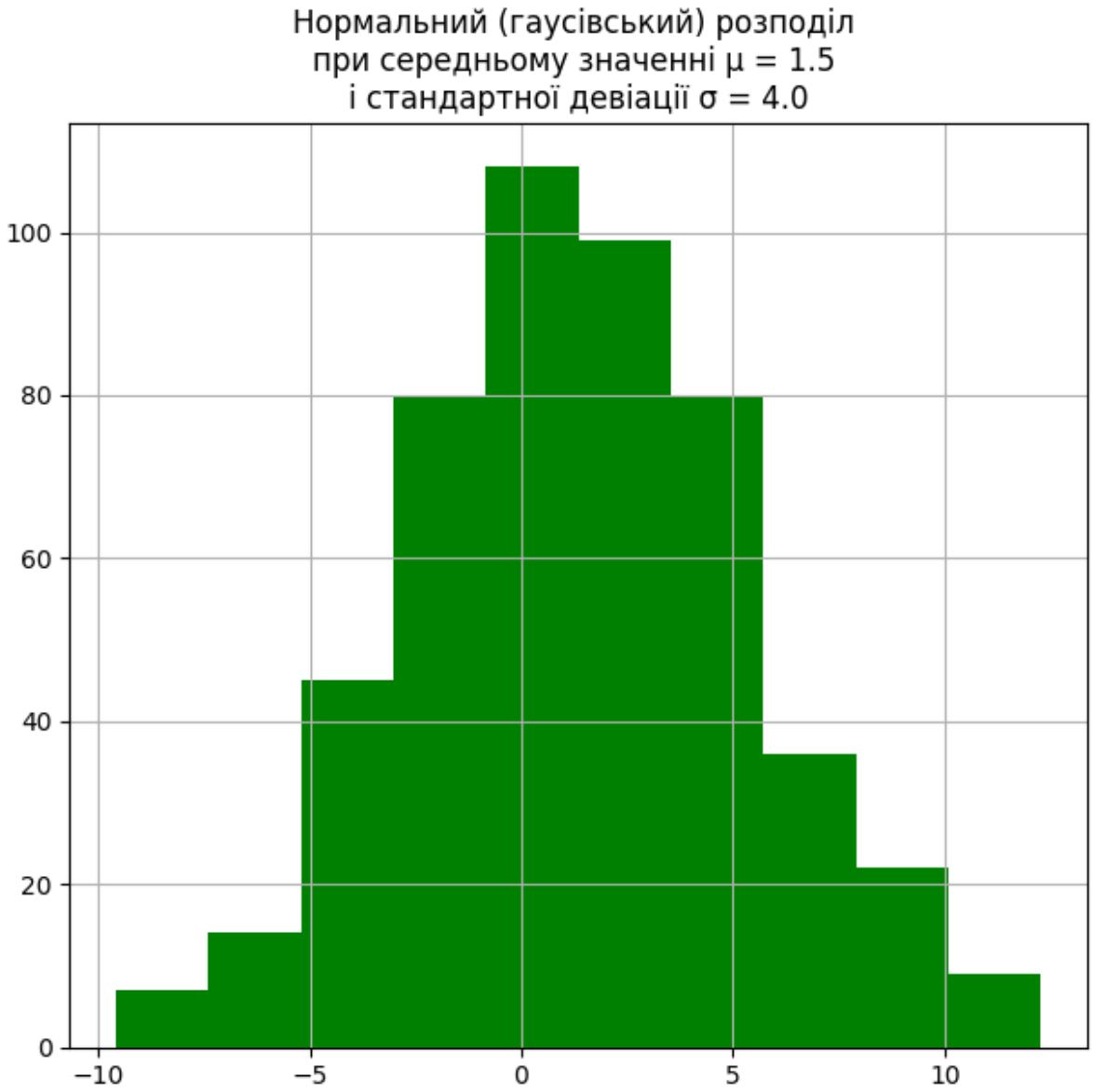
```
 Математичні операції над елементами масиву

+
 [[ 0.  0.  0.]
 [11. 13. 15.]
 [11. 13. 15.]]

-
 [[-2.  4.  6.]
 [-3. -3. -3.]
 [ 3.  3.  3.]]

*
 [[-1. -4. -9.]
 [28. 40. 54.]
 [28. 40. 54.]]

/
 [[-1.         -1.         -1.        ]
 [ 0.57142857  0.625       0.66666667]
 [ 1.75        1.6         1.5       ]]

**
 [[-1.0000000e+00  2.5000000e-01  3.7037037e-02]
 [ 1.6384000e+04  3.9062500e+05  1.0077696e+07]
 [ 2.4010000e+03  3.2768000e+04  5.3144100e+05]]

[1 2 3] @ [4 5 6] = 32

[1 2 3] * [4 5 6] = [ 4 10 18]

Задаванням масиву, елементи якого є арифметичною послідовністю

Start at:1
End at:50
Step:3
[ 1  4  7 10 13 16 19 22 25 28 31 34 37 40 43 46 49]

Process finished with exit code 0
```

2. Код до пункут 3.1-3.2:

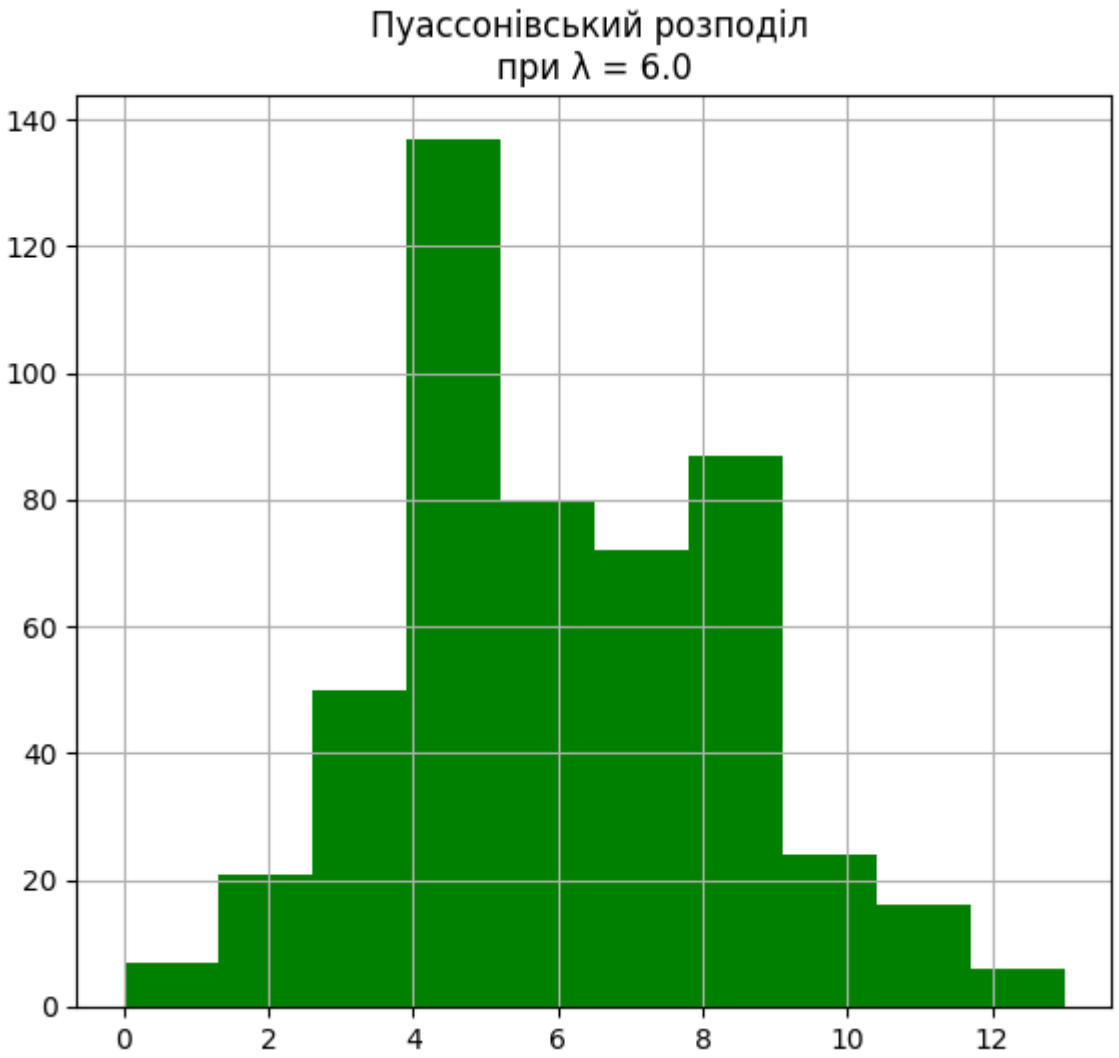
```
# Функція генерації випадкових чисел із заданими густинами розподілу імовірності.
import random  # модуль роботи з випадковими числами

# Імовірнісні розподілу
print('Імовірнісні розподілу\n')
print(
'\n', random.triangular(low=0, high=1, mode=0.5), '- випадкове число з плаваючою
точкою\n'
'\n', random.betavariate(alpha=1, beta=2), '- бета-розподіл\n'
'\n', random.expovariate(lambd=1), '- експоненціальне розподіл\n'
'\n', random.gammavariate(alpha=1, beta=2), '- гамма-розподіл\n',
'\n', random.gauss(mu=1, sigma=2), '- розподіл Гаусса\n',
'\n', random.lognormvariate(mu=1, sigma=2), '- логарифм нормального розподілу\n'
'\n', random.normalvariate(mu=1, sigma=0.1), '- нормальний розподіл\n'
'\n', random.vonmisesvariate(mu=1, kappa=0), '- середній кут, виражений в радіанах від 0
до 2π\n'
'\n', random.paretovariate(alpha=1), '- розподіл Парето\n'
'\n', random.weibullvariate(alpha=1, beta=2), '- розподіл Вейбулла\n'
)
```

```
"E:\Теорія Сигналів\PythonLabs\venv\Scripts\python.exe" "E:/Теорія Сигналів/PythonLabs/Laba1/3.2.py"
Імовірнісні розподілу


 0.6021662353867883 - випадкове число з плаваючою точкою

 0.43414983899125065 - бета-розподіл

 0.8022167107472956 - експоненціальне розподіл

 1.2502871043549462 - гамма-розподіл

 0.29104111771517027 - розподіл Гаусса

 55.15755588822291 - логарифм нормального розподілу

 0.9225340535825837 - нормальний розподіл

 5.168715942185176 - середній кут, виражений в радіанах від 0 до 2π

 1.8374262864328 - розподіл Парето

 1.996443459433359 - розподіл Вейбулла
```

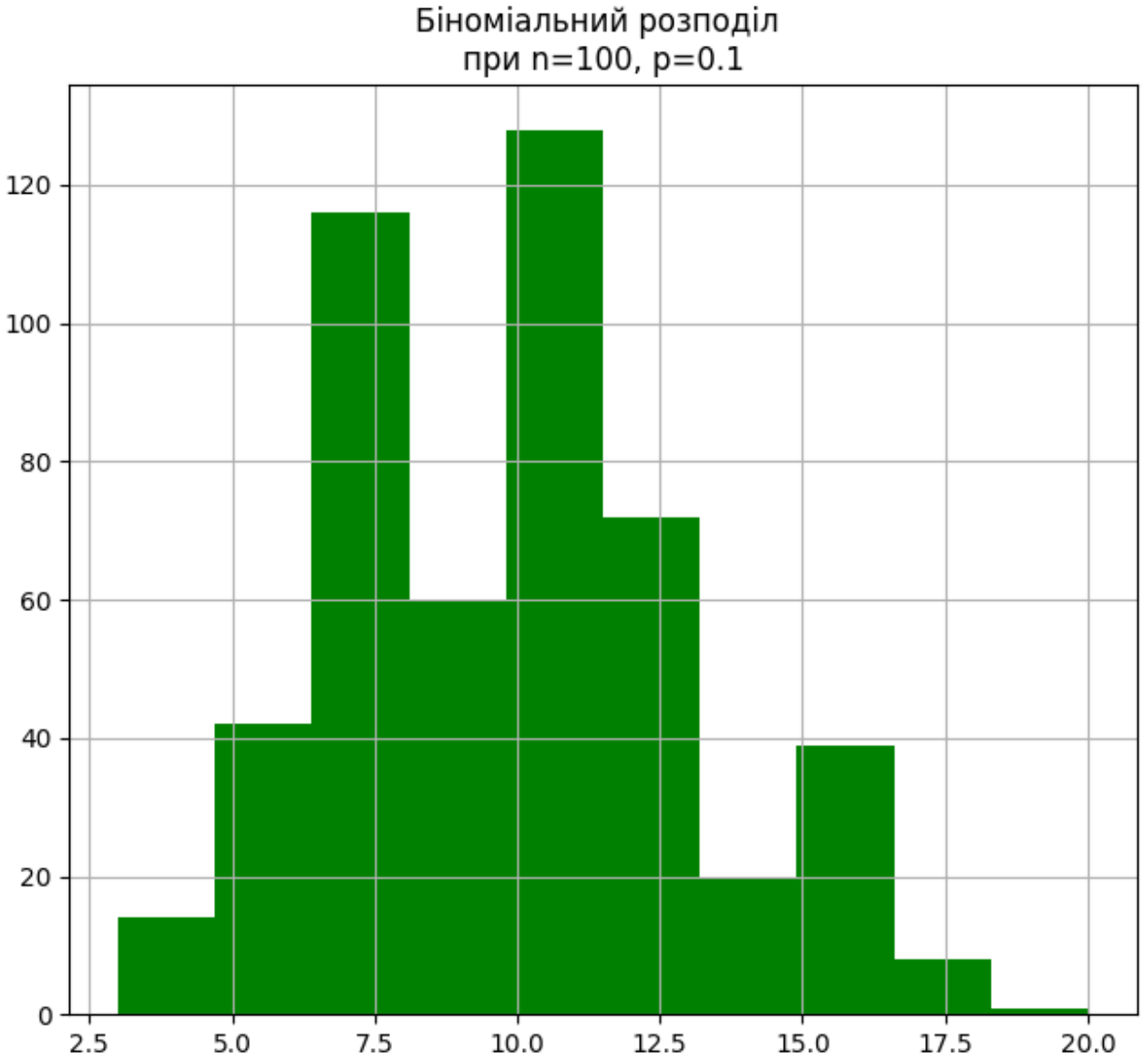
```
import matplotlib.pyplot as plt  # бібліотека для роботи з графічним
представленням
import numpy as np  # модуль NumPy надає зручні методи для генерації випадкових
даних

np.random.seed(0)  # змінюємо значення, яке передається в random для генерації
випадкового числа
x = np.random.normal(1.5, 4.0, 500)  # отримуємо масив значень нормального
(гаусівського) розподілу (500 точок)
fig = plt.figure()  # створюємо Figure контейнер самого високого рівня.
grid1 = plt.grid(True)
plt.title('Нормальний (гаусівський) розподіл \nпри середньому значенні μ = 1.5
\ni стандартної девіації σ = 4.0')
plt.hist(x, color='g')  # функція побудови гістограм.
plt.show()  # функція, яка показує поточний стан нашого графіку на figure.
```



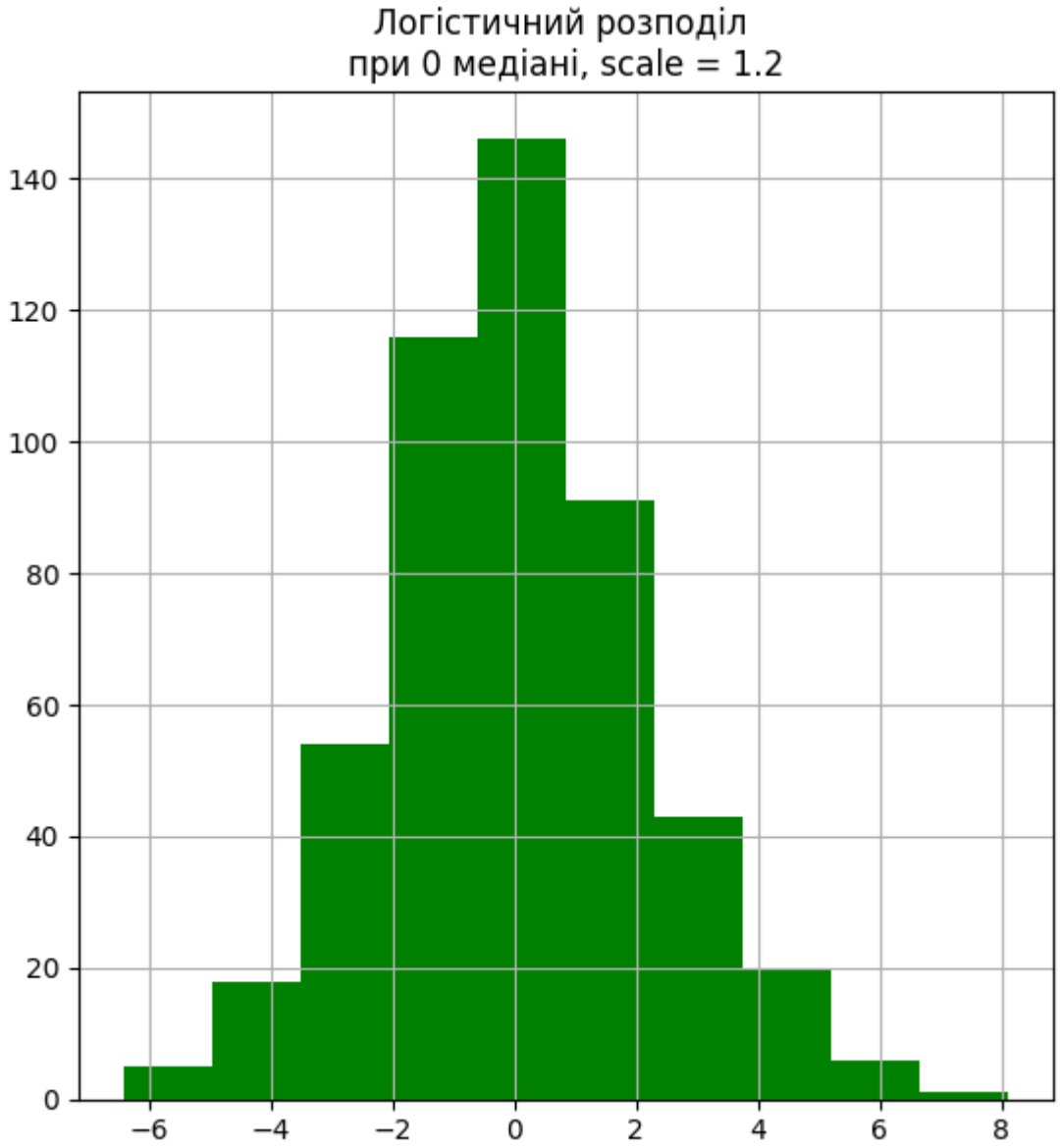
```
np.random.seed(0)  # змінюємо значення, яке передається в random для генерації
випадкового числа
x = np.random.poisson(6.0, 500)  # отримуємо масив значень Пуассонівського розподілу
(500 точок)
fig = plt.figure()  # створюємо Figure контейнер самого високого рівня.
grid1 = plt.grid(True)
plt.title('Пуассонівський розподіл \nпри λ = 6.0')
plt.hist(x, color='g')  # функція побудови гістограм.
plt.show()  # функція, яка показує поточний стан нашого графіку на figure.
```



```
np.random.seed(0)  # змінюємо значення, яке передається в random для генерації
випадкового числа
x = np.random.binomial(100, 0.1, 500)  # отримуємо масив значень Біноміального розподілу
(500 точок)
fig = plt.figure()  # створюємо Figure контейнер самого високого рівня.
grid1 = plt.grid(True)
plt.title('Біноміальний розподіл \nпри n=100, p=0.1')
plt.hist(x, color='g')  # функція побудови гістограм.
plt.show()  # функція, яка показує поточний стан нашого графіку на figure.
```

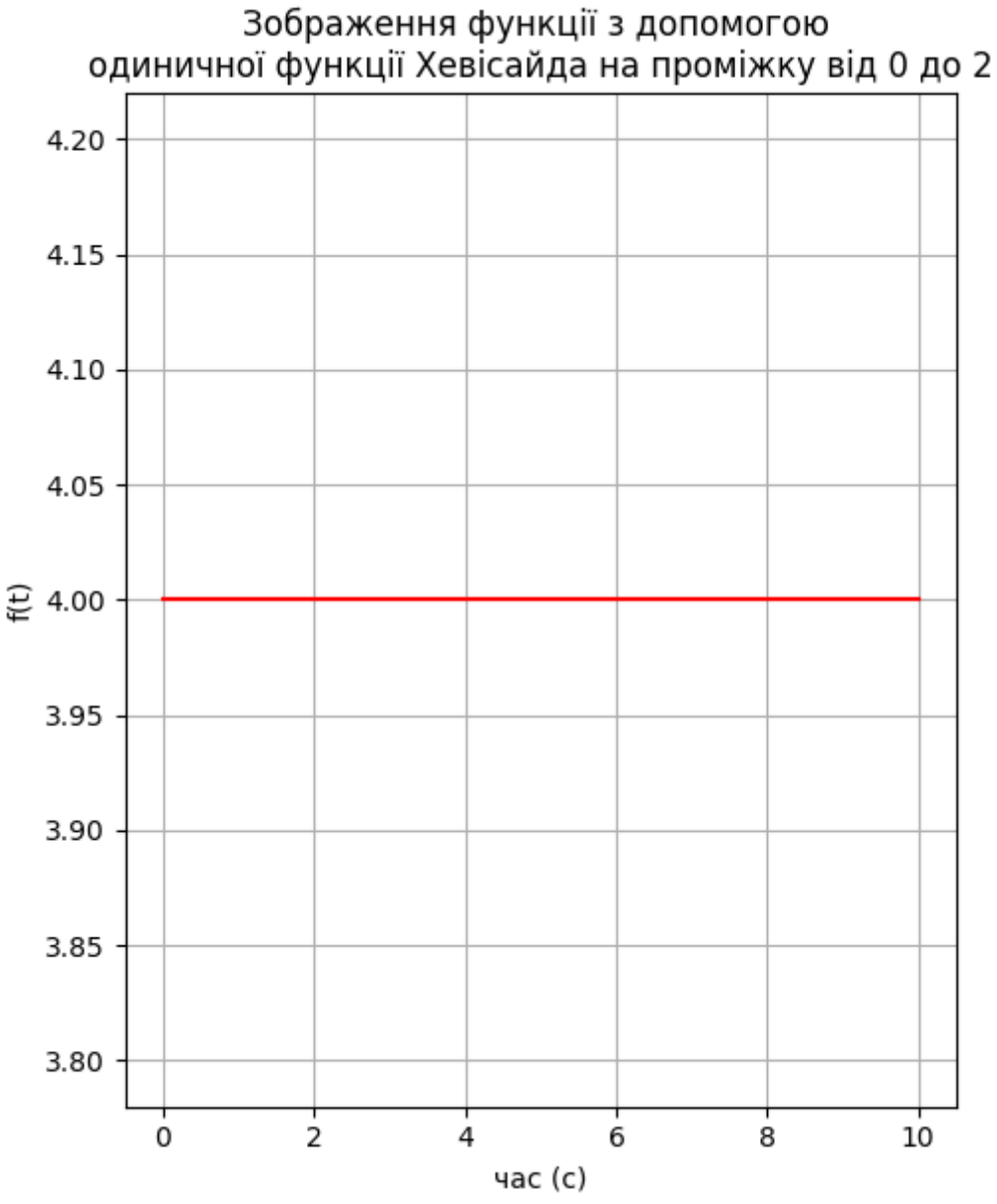


```
np.random.seed(0)  # змінюємо значення, яке передається в random для генерації випадкового числа
x = np.random.logistic(0, 1.2, 500)   # отримуємо масив значень Логістичного розподілу (500 точок)
fig = plt.figure()  # створюємо Figure контейнер самого високого рівня.
grid1 = plt.grid(True)
plt.title('Логістичний розподіл \nпри 0 медіані, scale = 1.2')
plt.hist(x, color='g')  # функція побудови гістограм.
plt.show()  # функція, яка показує поточний стан нашого графіку на figure.
```



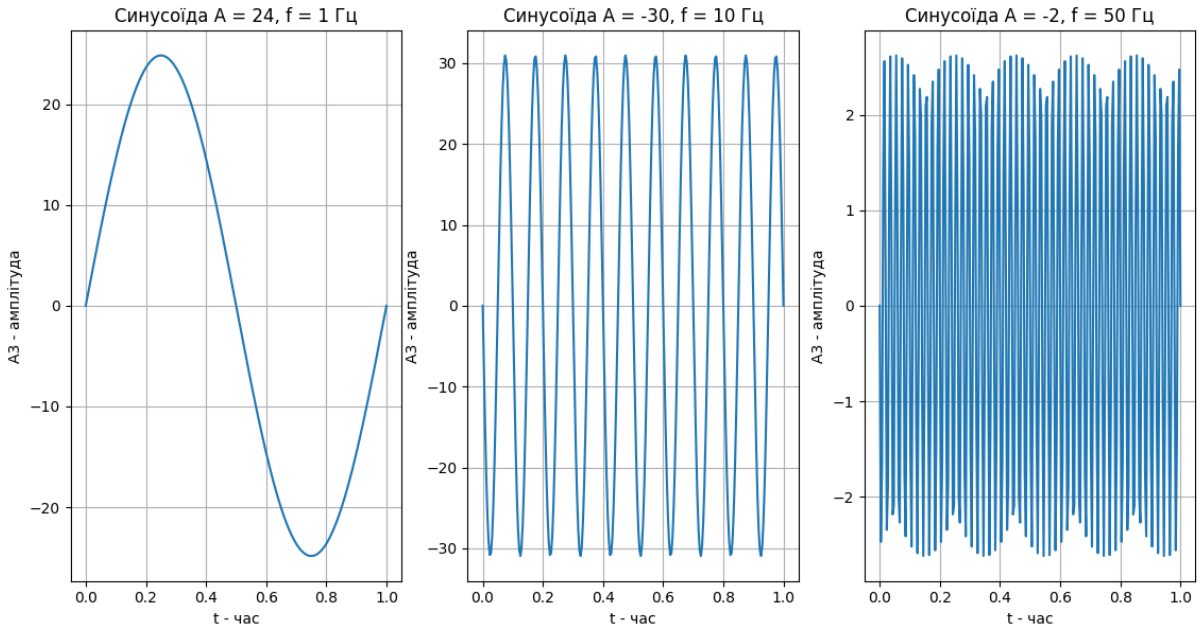
3. Код до пункут 4:

```
# -*- coding: utf-8 -*-
# Программа для побудови графіку функції Хевісайда в певний момент часу

import numpy as np
import matplotlib.pyplot as plt

t = np.linspace(0, 10, 100)  # Час йде лінійно від 0 до 10
f = 2.0*((t - 3.0) - (t-5.0))  # Функція Хевісайда для проміжку від 0 до 2

fig = plt.figure()
plt.grid(True)
plt.xlabel('час (с)')
plt.ylabel('f(t)')
plt.title('Зображення функції з допомогою \подиничної функції Хевісайда на проміжку від
0 до 2')
plt.plot(t, f, color='red')
plt.show()
```



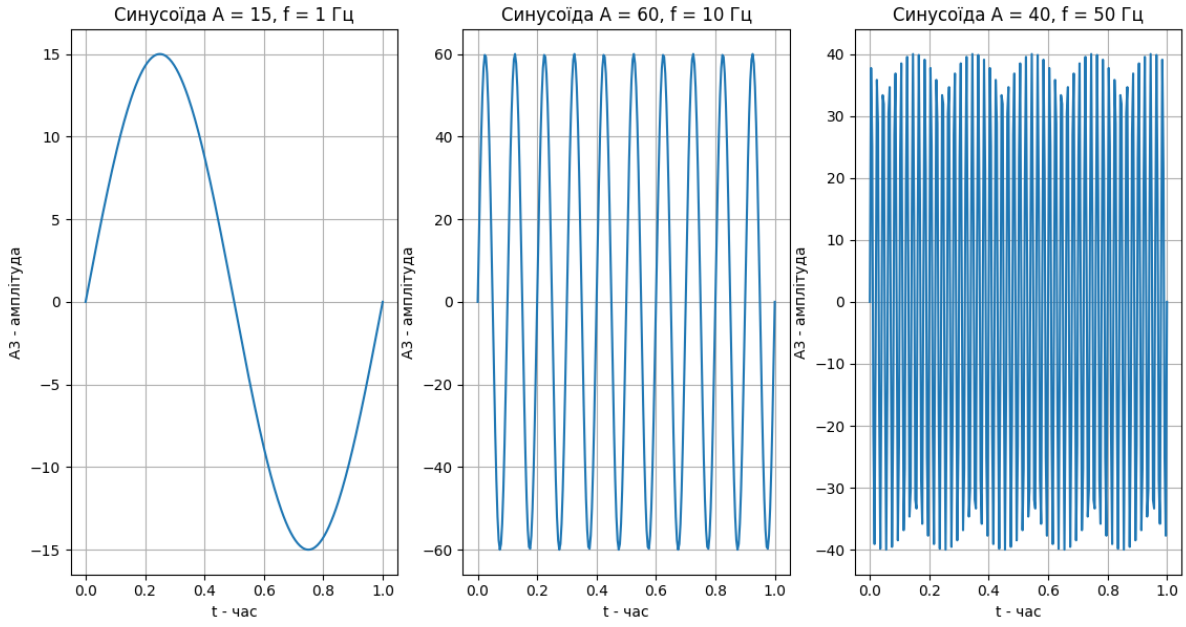
4. Код до пункут 5:

```
import numpy as np
import matplotlib.pyplot as plt
import math

f1 = 1
f2 = 10
f3 = 50

t = 1
Fd = 256
ft = int(t*Fd)

A1 = np.random.normal(1, 100)
A2 = np.random.normal(1, 50)
A3 = np.random.normal(1, 10)

t = np.linspace(0, 1, ft)

S1 = A1 * np.sin(2 * math.pi * f1 * t)
S2 = A2 * np.sin(2 * math.pi * f2 * t)
S3 = A3 * np.sin(2 * math.pi * f3 * t)

fig = plt.figure(1)
fig.add_subplot(1, 3, 1)
plt.plot(t, S1)
plt.title('Синусоїда A = %d, f = %d Гц' % (A1, f1))
plt.ylabel('A3 - амплітуда')
plt.xlabel('t - час')
plt.grid()

fig.add_subplot(1, 3, 2)
plt.plot(t, S2)
plt.title('Синусоїда A = %d, f = %d Гц' % (A2, f2))
plt.ylabel('A3 - амплітуда')
plt.xlabel('t - час')
plt.grid()

fig.add_subplot(1, 3, 3)
plt.plot(t, S3)
plt.title('Синусоїда A = %d, f = %d Гц' % (A3, f3))
plt.ylabel('A3 - амплітуда')
plt.xlabel('t - час')
plt.grid()

plt.show()
```



Замінивши наступні рядки коду, отримаємо можливість вводу амплітуди:

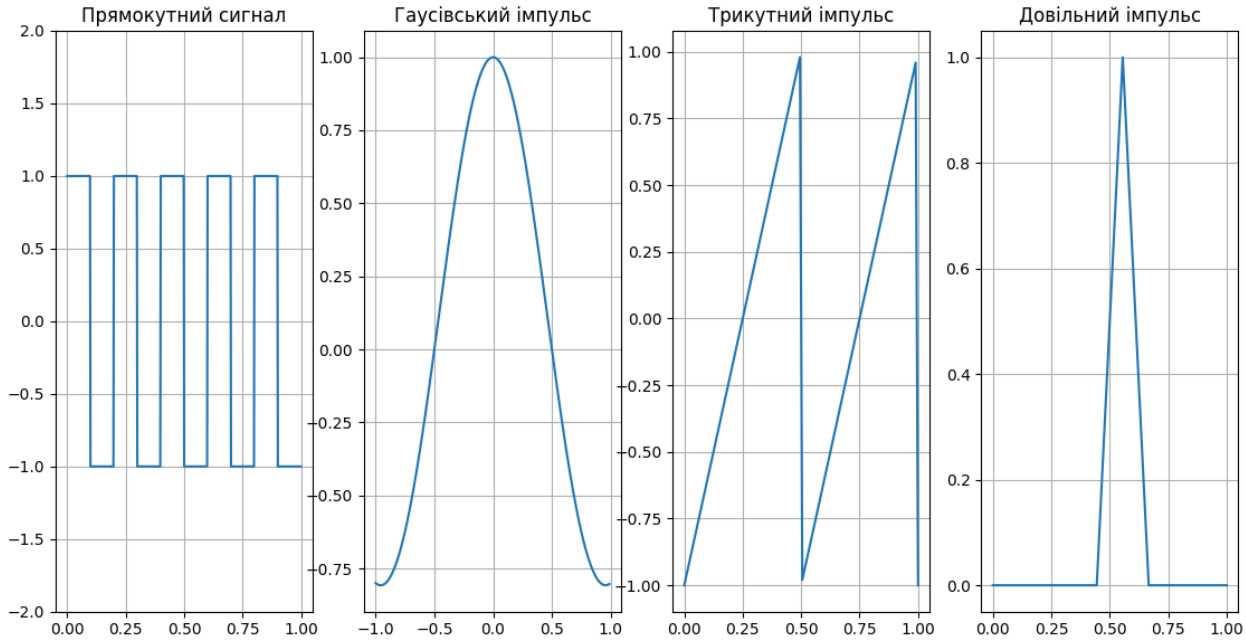
```
A1 = int(input('Амлітуда для першого сигналу: '))
A2 = int(input('Амлітуда для другого сигналу: '))
A3 = int(input('Амлітуда для третього сигналу: '))
```



5. Код до пункут 6:

```
import matplotlib.pyplot as plt
import numpy as np
from scipy import signal  # Бібліотека для побудови сигналів

t = np.linspace(0, 1, 500, endpoint=False)
fig = plt.figure()

fig.add_subplot(1, 4, 1)
```

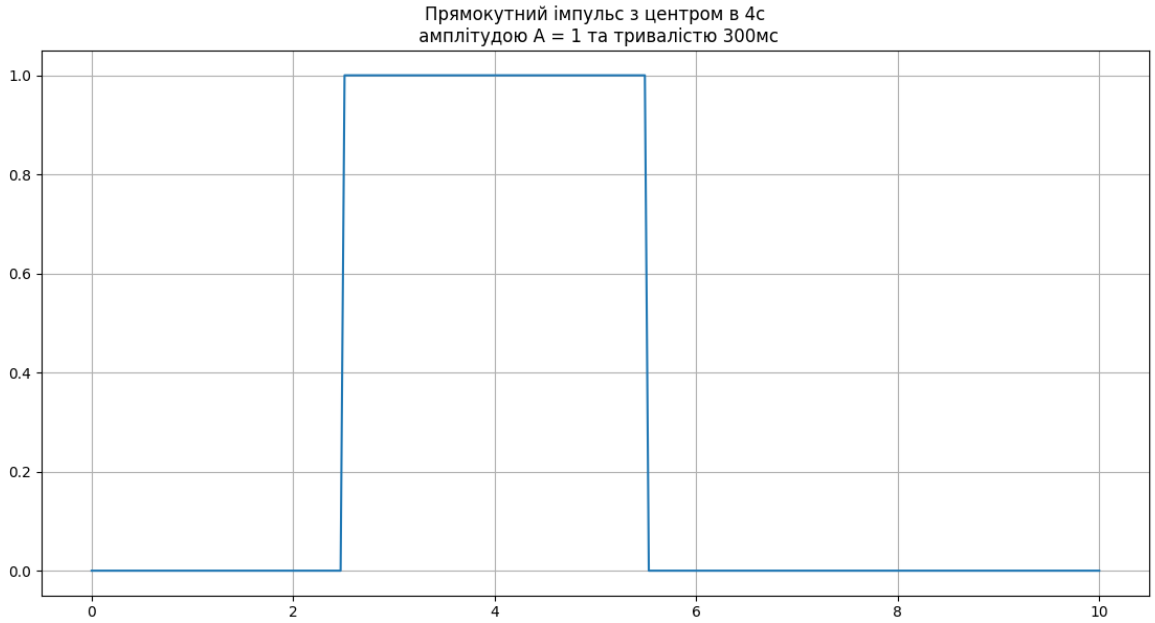
```
plt.plot(t, signal.square(2 * np.pi * 5 * t))
plt.ylim(-2, 2)
plt.title('Прямокутний сигнал')
plt.grid()

fig.add_subplot(1, 4, 2)
t = np.linspace(-1, 1, 2 * 100, endpoint=False)
gaussPulse = signal.gausspulse(t, fc=0.5)
plt.plot(t, gaussPulse)
plt.title('Гаусівський імпульс')
plt.grid()

fig.add_subplot(1, 4, 3)
t = np.linspace(0, 1, 100)
plt.plot(t, signal.sawtooth(2 * np.pi * 2 * t))
plt.title('Трикутний імпульс')
plt.grid()

fig.add_subplot(1, 4, 4)
t = np.linspace(0, 1, 10)
anyShape = signal.unit_impulse(10, 'mid')
plt.plot(t, anyShape)
plt.title('Довільний імпульс')
plt.grid()

plt.show()
```



6. Код до пункут 6.1:

```
import numpy as np
import matplotlib.pyplot as plt

Td = 256
t = 10
ft = Td*t

t = np.linspace(0, t, Td, endpoint=True)
y = np.zeros(len(t))
y[64:141] = 1
```

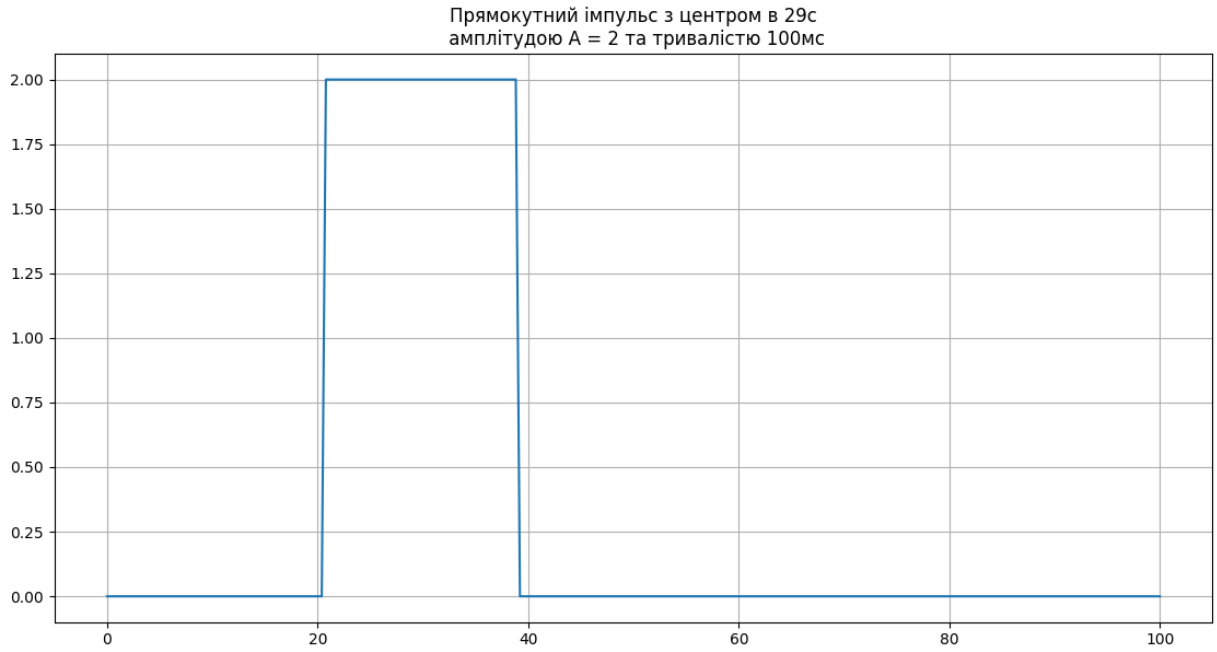
```
plt.plot(t, y)
plt.title('Прямокутний імпульс з центром в 4с\n амплітудою A = 1 та тривалістю 300мс')
plt.grid()
plt.show()
```



6. Код до пункут 6.2:

```
import numpy as np
import matplotlib.pyplot as plt
import math

ti = float(input('Тривалість імпульсу:'))
T = float(input('Амплітуда імпульса:'))

Td = 256
t = ti

randomVal = np.random.randint(0, ti)
randomVal1 = np.random.randint(0, ti)

point = 0
startPoint = 0

t = np.linspace(0, t, Td, endpoint=True)
y = np.zeros(len(t))
if randomVal1 <= randomVal:
    startPoint = randomVal1
    for num in range(len(t)):
        if randomVal1 <= num <= randomVal:
            y[num] = T
            point += 1
else:
    startPoint = randomVal
    for num in range(len(t)):
        if randomVal <= num <= randomVal1:
            y[num] = T
            point += 1


centerOfRange = math.ceil((startPoint - 1 + point/2))
```

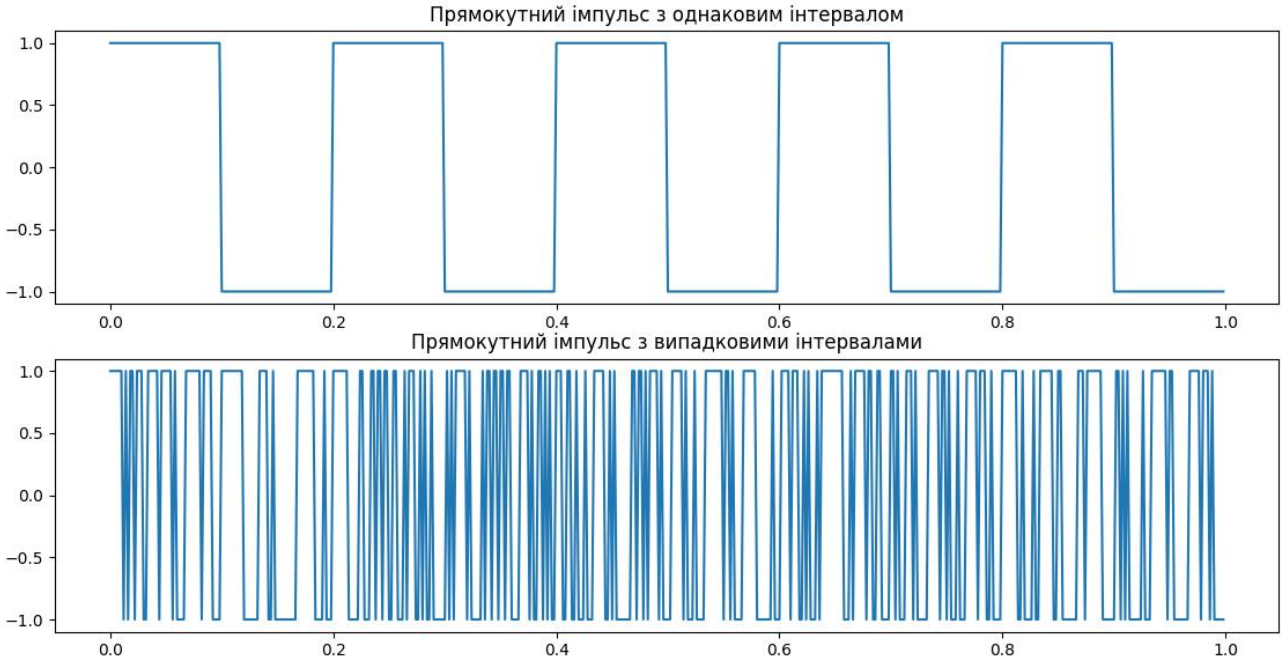
```
plt.plot(t, y)
plt.title('Прямокутний імпульс з центром в %.dc\n амплітудою A = %.d та тривалістю
%.dмс' % (t[centerOfRange], T, ti))
plt.grid()
plt.show()
```




```
import numpy as np
from scipy import signal
import random

t = np.linspace(0, 1, 500, endpoint=False)
si = signal.square(2 * np.pi * 5 * t)
sig = np.array([random.random() for i in range(len(t))])
pwm = signal.square(2 * np.pi * 30 * t, duty=sig)

np.save('data_y.npy', t)
np.save('data_x.npy', si)
np.save('data_rand_x.npy', pwm)
```

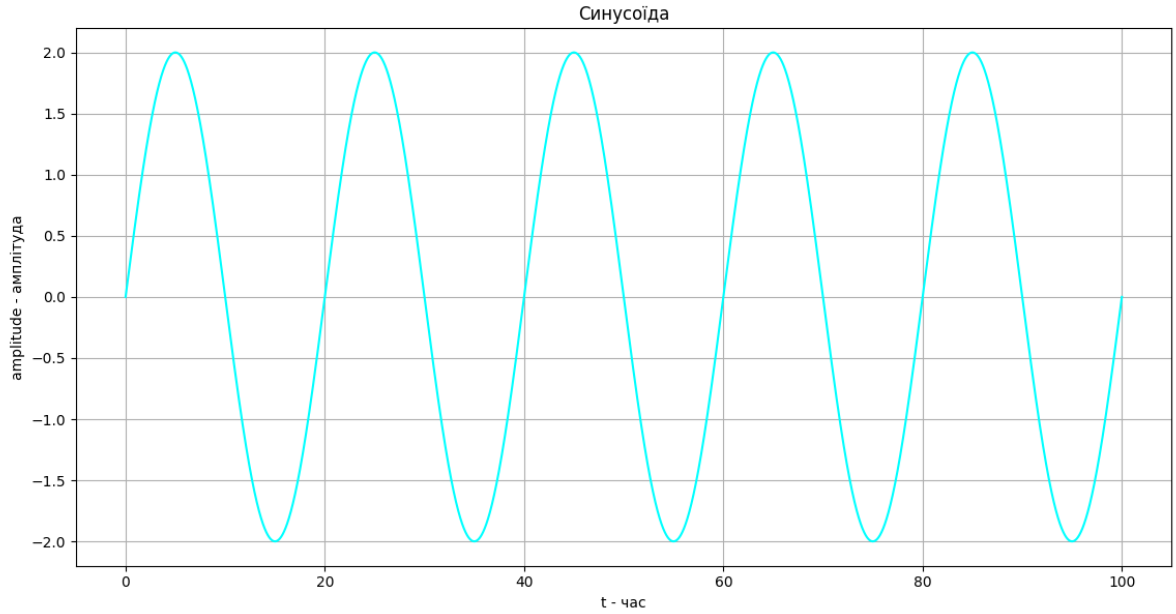
7. Код до пункут 7:

```
import numpy as np
import matplotlib.pyplot as plt


x = np.load('data_x.npy')
x_rand = np.load('data_rand_x.npy')
y = np.load('data_y.npy')

plt.figure()
plt.subplot(2, 1, 1)
plt.plot(y, x)
plt.title('Прямокутний імпульс з однаковим інтервалом')

plt.subplot(2, 1, 2)
plt.plot(y, x_rand)
plt.title('Прямокутний імпульс з випадковими інтервалами')
plt.show()
```



8. Код до пункут 8:

```
import numpy as np
import matplotlib.pyplot as plt


def sinusoid(f, amplitude, t):
    fs = 256
    fd = t*fs

    x = np.linspace(0, t, fd)
    y = amplitude * np.sin(2 * np.pi * f * x)

    plt.plot(x, y, color='cyan')
    plt.ylabel('amplitude - амплітуда')
    plt.xlabel('t - час')
    plt.title('Синусоїда')
    plt.grid()
    plt.show()

    return sum(y/len(x))


print(sinusoid(0.05, 2, 100))
```



Висновок:

Прописавши таку кількість коду та вивівши таку кількість графіків та сигналів, можу виділити той факт, що пітон і бібліотека `matplotlib` дуже гнучка та функціонально зручна, в деяких місцях навіть краще MatLab. Також можна виділити окрему стезю по роботі з випадковими числами. Це одна з самих надійних систем генерування випадкових чисел, яку я зустрічав до цього часу. Дуже гнучка, цікава, непередбачувана. Із основних моментів, які я виділяю особисто для себе, це високий рівень читабельності коду, порівняно з іншими мовами, зрозумілість роботи базових функцій та логічних операцій. Висока передбачуваність написаного коду, та простий однопотоковий режим данних. Було цікаво, не менш складно. Дякую.