Python Programming 1

(CP1S02005E)

Białystok University of Technology
Faculty of Electrical Engineering

Industry Digitization, semester II
Academic year 2024/2025

Lecture no. 08 (23.04.2025)

Jarosław Forenc, PhD

Topics

- Files in Python
  - opening a file - open() function
  - closing a file - close() method
  - reading from a text file
  - writing to a text file
  - CSV format

Python - files (opening a file)

- to open a file, use the built-in open() function

```
open(file, mode='r', buffering=-1, encoding=None)
```

- file - the name of the file along with its path
- mode - the file opening mode:
  - 'r' - opens the file for reading (default)
  - 'w' - opens the file for writing; if the file already exists, its content will be erased; if it doesn't exist, a new file will be created
  - 'a' - opens the file for appending; new content will be added at the end of the existing content
  - 'r+' - opens the file for both reading and writing
  - 'b' - binary mode, used to open the file in binary mode (e.g. 'rb', 'wb', 'ab')

Python - files (opening a file)

- to open a file, use the built-in open() function

```
open(file, mode='r', buffering=-1, encoding=None)
```

- buffering - determines whether data is buffered; the default is -1, which means the default buffering settings are used; you can provide 0 to disable buffering or a value greater than 0 to specify the buffer size
- encoding - the encoding used for a text file; the default is None, which means the default encoding of the environment is used, e.g. 'utf-8', 'utf-16', 'utf-32', 'ascii'

- the open() function returns a file object, which is used to perform file operations; this object is usually assigned to a variable

```
file = open("file_name.txt", "w")
```

Python - files (opening a file, examples)

- opening the file data.txt for reading (default text mode)

```
file = open("data.txt", "r")
```

- opening the file date.txt located on disk D in the results directory for appending (default text mode)

```
file = open("d:/results/data.txt", "a")
```

- opening the file data.txt for reading and writing (default text mode)

```
file = open("data.txt", "r+")
```

- opening the file data.dat for reading and writing in binary mode

```
file = open("results/data.dat", "rb+")
```

Python - files (opening a file)

- trying to open a non-existent file for reading

```
file = open("data.txt", "r")
# file operations
file.close()
```

will result in an error and raise a FileNotFoundError exception

```
Traceback (most recent call last):
  File "d:\myapp.py", line 1, in <module>
    file = open("data.txt", "r")
           ^^^^^^^^^^^^^^^^^^^^^
FileNotFoundError: [Errno 2] No such file or
directory: 'data.txt'
```

Python - files (opening a file)

- to prevent opening a non-existent file, you can use exception handling (try-except section)

```
try:
    file = open("data.txt", "r")
    # file operations
    file.close()
except FileNotFoundError:
    print("File does not exist.")
```

```
File does not exist.
```

Python - files (closing a file)

- to close the file, the close() method is used, which is called on the file object after the operation on it is completed

```
file = open("data.txt", "w")
# file operations
file.close()
```

- closing a file is important because it frees up system resources used by the file and ensures that all buffered data is written to disk
- after calling the close() method, further operations on the file using the same object will not be possible and will raise an exception: ValueError: I/O operation on closed file

Python - files (opening and closing a file)

- instead of the traditional method of opening and closing a file:

```
file = open("data.txt", "w")
# file operations
file.close()
```

- it is recommended to use the with statement, which ensures the file is automatically closed after the with block is exited

```
with open("data.txt", "r") as file:
    # file operations
```

- this helps avoid issues related to unclosed files and results in cleaner, more readable code

Python - files (reading from text file)

- the read() method reads the entire contents of a file as a single string

```
with open("file_name.txt", "r") as file:
    content = file.read()
    print(content)
```

- the readline() method reads the next line of text from the file

```
with open("file_name.txt", "r") as file:
    line = file.readline()
    while line:
        print(line)
        line = file.readline()
```

Python - files (reading from text file)

- the readlines() method reads all lines from the file and returns them as a list of strings

```
with open("file_name.txt", "r") as file:
    lines = file.readlines()
    for line in lines:
        print(lines)
```

- you can also use a for loop to iterate over a file; each line in the file will be treated as the next iterable element

```
with open("file_name.txt", "r") as file:
    for line in file:
        print(file, end="")
```

- setting end="" in the print() function means that it will not automatically add a newline character after each line read from the file, which helps avoid double spacing between lines

Python - files (reading from text file, example)

- contents of the file data.txt

```
Jarosław Kamiński
Grażyna Wójcik
Piotr Wiśniewski
```

- displaying the contents of the file with line numbering

```
with open("data.txt", "r", encoding = "utf-8") as file:
    line_number = 1
    line = file.readline()
    while line:
        print(f"{line_number}: {line}", end="")
        line = file.readline()
        line_number += 1
```

```
1: Jarosław Kamiński
2: Grażyna Wójcik
3: Piotr Wiśniewski
```

Python - files (reading from text file, example)

- if character encoding is not specified, the text may be displayed incorrectly

```
with open("data.txt", "r") as file:
    line_number = 1
    line = file.readline()
    while line:
        print(f"{line_number}: {line}", end="")
        line = file.readline()
        line_number += 1
```

```
1: JarosŁ,aw KamiŁ„ski
2: GraŁŁyna WĂłjcik
3: Piotr WiŁ›niewski
```

- this is due to differences between the character encoding used when the file was created and the one used for displaying it

```
import os
os.system("chcp")
```

```
Active code page: 852
```

Python - files (reading from text file, example)

- the text file contains integers
- calculating and displaying the sum and arithmetic mean of the numbers

```
with open("numbers.txt", "r") as file:
    sum = 0
    number_of_elements = 0
    line = file.readline()
    while line:
        number = int(line)
        sum += number
        number_of_elements += 1
        line = file.readline()
    average = sum / number_of_elements
    print("Sum of numbers from file:", sum)
    print("Arithmetic mean of numbers from file:", average)
```

```
23
18
-11
53
6
```

```
Sum of numbers from file: 89
Arithmetic mean of numbers from file: 17.8
```

Python - files (reading from text file, example)

- sum and average of numbers from a text file with error handling

```
try:
    with open("numbers.txt", "r") as file:
        sum = 0
        number_of_elements = 0
        line = file.readline()
        while line:
            number = int(line)
            sum += number
            number_of_elements += 1
            line = file.readline()
        average = sum / number_of_elements
        print("Sum of numbers from file:", sum)
        print("Arithmetic mean of numbers from file:", average)
except FileNotFoundError:
    print("File 'numbers.txt' not found.")
except ValueError:
    print("Error converting data to integer.")
except IOError:
    print("An error occurred while reading the file.")
```

Python - files (saving to text file)

- the write() method is used to write single lines or fragments of text to an open text file

```
with open("output.txt", "w") as file:
    file.write("This is the first line.\n")
    file.write("This is the second line.\n")
```

- the writelines() method is used to write multiple lines of text at once; it takes a list of strings as an argument

```
lines = ["The first line.\n", "The second line.\n"]
with open("output.txt", "w") as file:
    file.writelines(lines)
```

Python - files (saving to text file)

- the print() function has an optional file parameter that allows you to write text to a specified file instead of to the standard output (e.g., the screen)

```
with open("output.txt", "w") as file:
    print("The first line.", file=file)
    print("The second line.", file=file)
```

- the data written to the file can be formatted

```
with open("output.txt", "w") as file:
    name = "John"
    age = 30
    file.write(f"Name: {name}, Age: {age}\n")
```

```
Name: John, Age: 30
```

Python - files (saving to text file, example)

- saving 10 pseudo-random real numbers in a text file; each number should be in the range from 0 to 99, rounded to 3 decimal places

```
import random
num = [round(random.uniform(0, 99), 3) for _ in range(10)]
with open("numbers.txt", "w") as file:
    for number in num:
        file.write(f"{number:.3f}\n")
print("The numbers have been saved.")
```

```
56.766
7.212
10.048
43.205
60.943
69.050
90.860
42.654
85.622
6.440
```

- the random.uniform(a, b) function from the random module generates pseudo-random real numbers in the interval [a, b]
- the round(number, ndigits) function rounds a number to the specified number of decimal places, where number is the value to round, and ndigits is the number of digits after the decimal point

Python - CSV format

- CSV (Comma-Separated Values) - a simple format for storing tabular data, where each row represents a single record and the values are separated by a delimiter (most commonly a comma)
- the delimiter can also be a semicolon, space, tab, etc.
- example of a CSV file (oscilloscope data):

```
X,CH1,
Second,Volt,
-3.00000e-04,-3.20e-01,
-2.99000e-04,-6.40e-01,
-2.98000e-04,-7.20e-01,
-2.97000e-04,-1.04e+00,
```

- each data row is separated from the next by a newline character
- the first row in a CSV file often contains column headers that describe the content of each column

Python - CSV format (example no. 1)

- loading a CSV file with oscilloscope data and creating a plot using NumPy and Matplotlib libraries
- the libraries need to be installed - in the Terminal window, enter:
  pip install matplotlib numpy

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

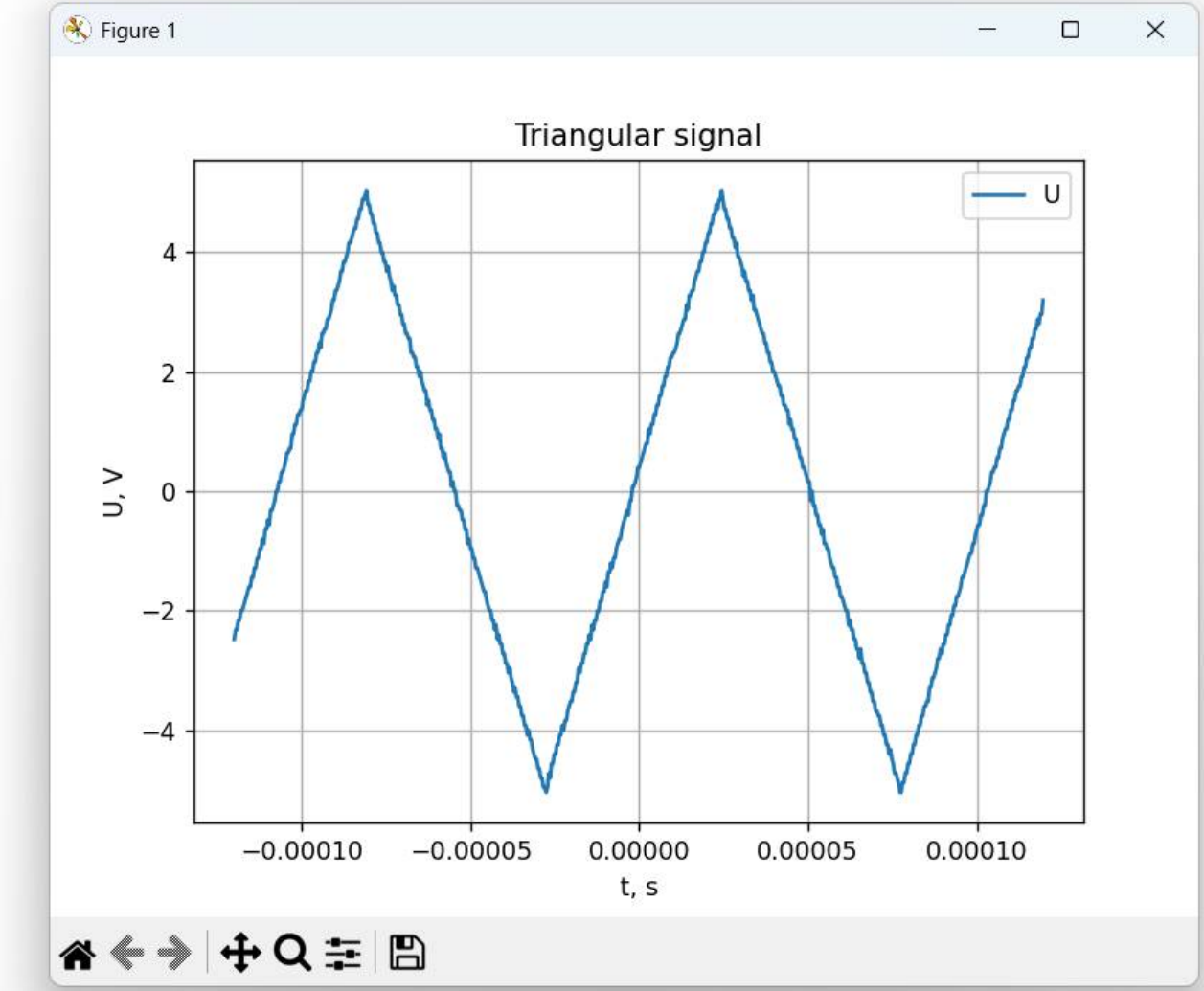
Python - CSV format (example no. 1)

```
# Loading data from a CSV file
x_data = []
y_data = []

with open('d:/results.csv', 'r') as file:
    reader = csv.reader(file)
    next(reader)  # Skip the header row
    next(reader)  # Skip the header row
    for row in reader:
        # Store data in appropriate lists
        x_data.append(float(row[0]))
        y_data.append(float(row[1]))

# Convert lists to numpy arrays for easier processing
x_data = np.array(x_data)
y_data = np.array(y_data)
```

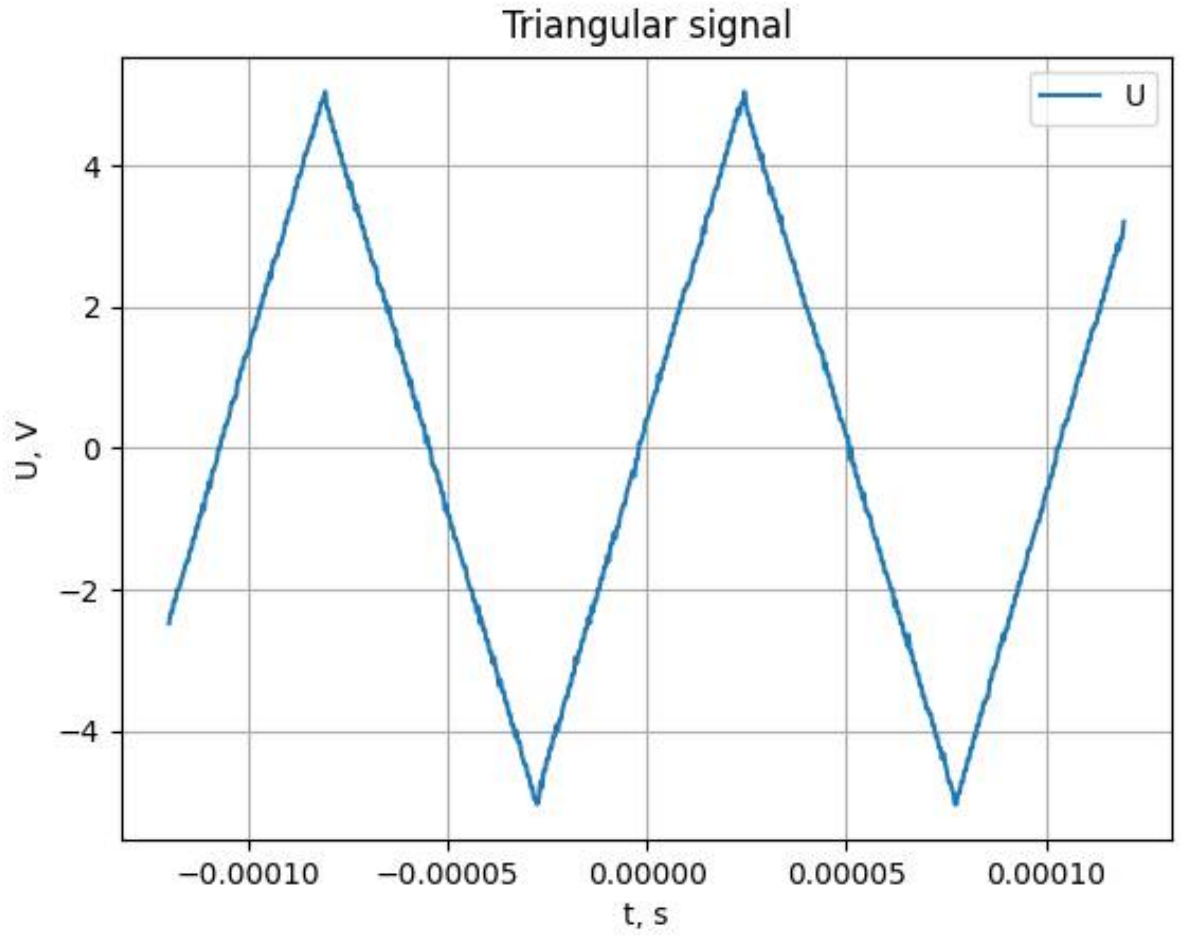
Python - CSV format (example no. 1)

```
# Plotting the graph
plt.plot(x_data, y_data, label='U')

# Setting axis labels
plt.xlabel('t, s')
plt.ylabel('U, V')

# Chart title
plt.title('Triangular signal')

# Adding a legend
plt.legend()

# Enabling the grid
plt.grid(True)

# Displaying the plot
plt.show()
```

Python - CSV format (example no. 1)



Python - CSV format (example no. 1)



Python - CSV format (example no. 2)

- list of male names in the PESEL register as of 19/01/2024 - first name

  (https://dane.gov.pl/pl/dataset/1667,lista-imion-wystepujacych-w-rejestrze-pesel-osoby-zyjace)

```
FIRST_NAME,GENDER,NUMBER_OF_OCCURRENCES
PIOTR,MALE,689313
KRZYSZTOF,MALE,641406
TOMASZ,MALE,536774
ANDRZEJ,MALE,533141
PAWEŁ,MALE,506006
MICHAŁ,MALE,492093
JAN,MALE,477845
MARCIN,MALE,449607
JAKUB,MALE,426709
ADAM,MALE,401674
ŁUKASZ,MALE,383582
MAREK,MALE,378001
...
```

Python - CSV format (example no. 2)

```
import csv

# Open the CSV file for reading
with open('name.csv', newline='', encoding="utf-8") as file:
    # Create a CSV reader
    reader = csv.DictReader(plik)

    # Display column headers
    print("Column headers:", reader.fieldnames)

    # Iterate through each row of data
    for row in reader:
        # Display the data in each row
        print(row)
```

- csv.DictReader() creates a CSV reader that automatically interprets the first row as column headers
- The fieldnames parameter contains the column headers as a list
- each row of data is represented as a dictionary, where the keys are the column headers and the values are the corresponding data

Python - CSV format (example no. 2)

- result of running the program

```
Column headers: ['FIRST_NAME', 'GENDER', 'NUMBER_OF_OCCURRENCES']
{'FIRST_NAME': 'PIOTR', 'GENDER': 'MALE', 'NUMBER_OF_OCCURRENCES': '689313'}
{'FIRST_NAME': 'KRZYSZTOF', 'GENDER': 'MALE', 'NUMBER_OF_OCCURRENCES': '641406'}
{'FIRST_NAME': 'TOMASZ', 'GENDER': 'MALE', 'NUMBER_OF_OCCURRENCES': '536774'}
{'FIRST_NAME': 'ANDRZEJ', 'GENDER': 'MALE', 'NUMBER_OF_OCCURRENCES': '533141'}
{'FIRST_NAME': 'PAWEŁ', 'GENDER': 'MALE', 'NUMBER_OF_OCCURRENCES': '506006'}
{'FIRST_NAME': 'MICHAŁ', 'GENDER': 'MALE', 'NUMBER_OF_OCCURRENCES': '492093'}
{'FIRST_NAME': 'JAN', 'GENDER': 'MALE', 'NUMBER_OF_OCCURRENCES': '477845'}
{'FIRST_NAME': 'MARCIN', 'GENDER': 'MALE', 'NUMBER_OF_OCCURRENCES': '449607'}
{'FIRST_NAME': 'JAKUB', 'GENDER': 'MALE', 'NUMBER_OF_OCCURRENCES': '426709'}
{'FIRST_NAME': 'ADAM', 'GENDER': 'MALE', 'NUMBER_OF_OCCURRENCES': '401674'}
{'FIRST_NAME': 'ŁUKASZ', 'GENDER': 'MALE', 'NUMBER_OF_OCCURRENCES': '383582'}
{'FIRST_NAME': 'MAREK', 'GENDER': 'MALE', 'NUMBER_OF_OCCURRENCES': '378001'}
{'FIRST_NAME': 'MATEUSZ', 'GENDER': 'MALE', 'NUMBER_OF_OCCURRENCES': '377011'}
{'FIRST_NAME': 'GRZEGORZ', 'GENDER': 'MALE', 'NUMBER_OF_OCCURRENCES': '374820'}
{'FIRST_NAME': 'STANISŁAW', 'GENDER': 'MALE', 'NUMBER_OF_OCCURRENCES': '366180'}
{'FIRST_NAME': 'WOJCIECH', 'GENDER': 'MALE', 'NUMBER_OF_OCCURRENCES': '335311'}
{'FIRST_NAME': 'MARIUSZ', 'GENDER': 'MALE', 'NUMBER_OF_OCCURRENCES': '286869`}
```

End of lecture no. 8

Thank you for your attention!