סיכום שיעור רביעי:

1. input/output – I/O

- The io module contains nearly every class you might ever need to perform input and output (I/O) in Python.
- All these streams represent an input source and an output destination.
- A stream can be defined as a sequence of data. There are two kinds of Streams −
    - InputStream − The InputStream is used to read data from a source.
    - OutputStream − The OutputStream is used for writing data to a destination.
- In simple words, the io module gives us the ability to read data from files (e.g. text files, XML files etc.) and write data into files.

File

- File is a named location on disk to store related information.
- It is used to permanently store data in our hard disk.
- When we want to read from or write to a file we need to open it first.
- When we are done, it needs to be closed, so that resources that are tied with the file are freed.
- Hence, in Python, a file operation takes place in the following order.
- Open a file
- Read or write (perform operation)
- Close the file
- Python has a built-in function open() to open a file.
- This function returns a file object, to read or modify the file accordingly.

```
My_file = open("C:/Python33/README.txt")
```

- When we open our file, we can specify the mode while opening a file.
- In mode, we specify whether we want to read 'r', write 'w' or append 'a' to the file.
- We also specify if we want to open the file in text mode or binary mode.
- The default is reading in text mode, so we get strings when reading from the file.
- Binary mode returns bytes and this mode is used when dealing with non-text files like images.

Mode	Description
'r'	Open a file for reading. (default)
'w'	Open a file for writing. Will create a new file if it does not exist.
'x'	Open a file for exclusive creation. If the file already exists, the operation fails.
'a'	Open for appending the file without truncating it. Creates a new file if it does not exist.
't'	Open in text mode. (default)
'b'	Open in binary mode (has to come with another char).
'+'	Open a file for updating (reading and writing)

```
My_file = open("C:/Python33/README.txt", 'r+')
```

- To read our file content, we can simply call read function:

```
My_file = open("C:/Python33/README.txt", 'r')
content = My_file.read()
```

- To write into our file, we can use write function:

```
My_file = open("C:/Python33/README.txt", 'r+')
content = My_file.write("text to add")
```

- Remember, when using w in your mode the old content will be overridden, and when using a, the new content will be appended to the old content.
- When working with files in text mode, it is highly recommended to specify the encoding type.
- For example, if we will try to read/write Hebrew text, we will receive an encoding error, to solve it, we can simply add the following:

```
My_file = open("C:/Python33/README.txt", 'r' , encoding='utf-8')
```

- When we are done with operations to the file, we need to properly close the file.
- Closing a file will free up the resources that were tied with the file and is done using **close**() method.

```
My_file = open("C:/Python33/README.txt", 'r')
My_file.read()
My_file.close()
```

Decorators .2

- By definition, a decorator is a function that takes another function and extends the behavior of the latter function *without* explicitly modifying it.
- A decorator is marked with @ symbol (this is called “pie” syntax).
- For example we saw it earlier when we added @static decoration to a function.
- We didn’t change the function, only added the decorator.

```
class Shark:
@staticmethod
    def drink():
        print(123)
```

- There is a variety of examples of using decorators, which we will see later.

Garbage collection .3

- Python’s memory allocation and deallocation method is automatic.
- The user does not have to preallocate or deallocate memory similar to using dynamic memory allocation in languages such as C or C++.
- Python uses reference counting, which means, for every object, there is a count of the total number of references to that object, if that count ever falls to 0, then you can immediately deallocate that object because it is no longer live.
- The garbage collector, can be called using gc.collect(), although, there is no reason to call it in standard programs.

Error handling .4

- In Python, there are two kinds of errors:
    - **Syntax errors**
- Syntax errors, also known as parsing errors, are the most basic type of error.
- They arise when the Python parser is unable to understand a line of code.
- When a piece of code has a syntax error, this code will not be able to run.
- An easy way to see syntax error will be writing something unrecognized such as your name in a Python module.

    - **Exceptions**
- An exception is an event, which occurs during the execution of a program that disrupts the normal flow of the program's instructions.
- In general, when a Python script encounters a situation that it cannot cope with, it raises an exception.
- An exception is a Python object that represents an error.
- When a Python script raises an exception, it must either handle the exception immediately otherwise it terminates and quits.

Try-except

- If you have some suspicious code that may raise an exception, you can defend your program by placing the suspicious code in a **try:** block.
- After the **try:** block, include an except: statement, followed by a block of code which handles the kind of the problem which was defined.

```
try:
    f = open("C:\\Users\dgotl\\testfile.txt", "r")
    f.write("123")
except IOError:
    print("Error: can't find file or read data")
```

- A single try statement can have multiple except statements. This is useful when the try block contains statements that may throw different types of exceptions.
- We can also provide a generic except clause, which handles any exception (bare except).

```
try:
    f = open("C:\\Users\dgotl\\testfile.txt", "r")
except:
    print("All kind of errors will be handled")
```

Finally

- We can use a **finally:** block along with a **try: / try:except:** block.
- The finally block is a place to put any code that must execute, whether the try-block raised an exception or not.

```
try:
    f = open("C:\\Users\dgotl\\testfile.txt", "r")
except IOError:
    print("Fatal error...")
finally:
    print("this will run anyway")
```

- Or:

```
try:
    f = open("C:\\Users\dgotl\\testfile.txt", "r")
finally:
    print("this will run anyway")
```

5. Debugger

- Debugging allows you to run a program interactively while watching the source code and the variables during the execution.
- A breakpoint in the source code specifies where the execution of the program should stop during debugging.
- Once the program is stopped you can investigate variables, change their content, etc.
- To stop the execution, if a field is read or modified, you can specify watchpoints.

6. Conventions

- Python **naming convention** is a rule to follow as you decide what to name your identifiers such as class, package, variable, constant, method etc.
- It is not forced to follow, therefore, it is known as convention not rule.

Name	Convention
class name	Class names should follow the UpperCaseCamelCase convention Dog, Cat..
Function name	Function names should be lowercase, with words separated by underscores and be a verb action_performed(), main(), print()
variable name	Variable names should be lowercase, with words separated by underscores first_name, order_number
package name	should be in lowercase letter lang, sql, util etc.
constants name	Should be in uppercase letter. RED, YELLOW, MAX_PRIORITY etc.

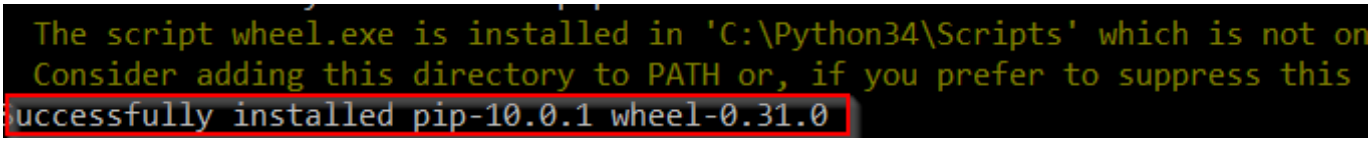
- Some general guidelines:
    - Everything should have a reasonable length.
    - You can't use reserved words as a variable name because Python has got them reserved them for other things.
    - Pick a logical name instead of a short name. For example – get_user_input is better than having just get.

The main goal is obvious! Make code readable by you and by others

Pip .7

- pip stands for either "Pip Installs Packages" or "Pip Installs Python".
- pip is a package management system used to install and manage software packages written in Python.
- Many packages can be found in the default source for packages and their dependencies — Python Package Index (PyPI)
- To install modules we can easily use the following syntax: **Pip install <package>**

- To check whether or not Pip is install, open CMD and type: **Pip,** in case you see many logs, Pip is installed.
- If you are getting "Pip is not recognized.." message, you will need to add manually:
    - Go to https://bootstrap.pypa.io/get-pip.py (or search get-pip.py in Google)
    - Copy-Paste the code into a file named get-pip.py (**make sure its not .txt**)
    - Open CMD/Terminal and navigate to the file directory by typing **cd <path of file>**
    - Type **python get-pip.py**
    - Once installation is done you will see a success message (look at picture below)
    - Add <YOUR_PYTHON_PATH>/scripts folder into PATH system variable.
    - To test it, type **pip**



תזכורת ממה שעשינו בכיתה:

```
# opening a file
My_file = open("C:/Python33/README.txt")

#reading file content
My_file = open("C:/Python33/README.txt", 'r')
content = My_file.read()

# writing to file
My_file = open("C:/Python33/README.txt", 'r+')
content = My_file.write("text to add")

# using file wiyh specific encoding
My_file = open("C:/Python33/README.txt", 'r' , encoding='utf-8')

# closing file at the end
My_file.close()

# error handling
# using try-except to protect a code block
# trying to write to a file that was opened to read only
try:
    f = open("C:\\Users\dgotl\\testfile.txt", "r")
    f.write("123")
except IOError:
    print("Error: can't find file or read data")

# try-except-finally
try:
    f = open("C:\\Users\dgotl\\testfile.txt", "r")
except IOError:
    print("Fatal error…")
finally:
    print("this will run anyway")
```