



The Islamic University of Gaza
Faculty of Engineering
Computer Engineering Department



Introduction to Computers

LAB #8

FILES AND EXCEPTION HANDLING

Eng. Hashem M. Hejazy

Eng. Mohammed O. AlHabbash

Fall 2026

Introduction

Why use files?

Data stored in variables is lost when a program ends. Files provide permanent storage on disk, allowing data to be saved and later reused by the same or other programs.

Why exception handling?

Runtime errors (e.g., trying to open a non-existent file) can crash a program. Exception handling enables a program to detect such errors and take corrective action without abrupt termination.

File paths

Absolute path: Full path from the root (e.g., D:\Universty\python.txt) on Windows.

Relative path: Path relative to the current working directory (e.g., "python.txt").

How to Get your current working directory:

```
import os
print(os.getcwd())
```

Change current working directory:

```
import os
os.chdir(r"D:\Universty\python")
```

File Handling

File handling refers to operations performed on files stored in secondary storage. These operations include:

- Creating files
- Writing data
- Reading data
- Appending data
- Closing files

Opening and Closing a File

To work with a file, you must first open it using the `open()` function. This returns a file object that provides methods for reading/writing.

```
file_object = open(filename, mode)
# Perform operations...
file_object.close()
```

Common file modes

Mode	Description
"r"	Read (file must exist)
"w"	Write (creates new file; overwrites if exists)
"a"	Append (writes at end; creates if doesn't exist)
"rb"	Read binary

Writing Data to a Text File

The `write()` method writes a string to the file. It does not automatically add a newline; you must include `\n` explicitly.

```
def main():
    outfile = open("Presidents.txt", "w")
    outfile.write("Bill Clinton\n")
    outfile.write("George Bush\n")
    outfile.write("Barack Obama")
    outfile.close()
main()
```

Checking File Existence Before Writing

To avoid accidentally overwriting an existing file, test its existence with `os.path.isfile()`.

```
import os.path

if os.path.isfile("Presidents.txt"):
    print("File already exists. Choose a different name.")
else:
    outfile = open("Presidents.txt", "w")
    # write data...
    outfile.close()
```

Reading Data from a Text File

Once a file is opened for reading ("r"), you can use several methods:

Method	Description
<code>read()</code>	Reads the entire file as a single string.
<code>read(n)</code>	Reads exactly <code>n</code> characters (or fewer if EOF).
<code>readline()</code>	Reads one line (including the trailing <code>\n</code>).
<code>readlines()</code>	Reads all lines and returns a list of strings.

Ex:

```
def main():
    outfile = open("Presidents.txt", "w")
    outfile.write("Bill Clinton\n")
    outfile.write("George Bush\n")
    outfile.write("Barack Obama")
    outfile.close()
main()

infile = open("Presidents.txt", "r")
print("Using read():")
print(infile.read())
infile.close()

infile = open("Presidents.txt", "r")
print("Using read(4):", infile.read(4))
print("Using read(10):", repr(infile.read(10)))    # repr
shows escape chars
infile.close()

infile = open("Presidents.txt", "r")
print("Using readline():")
print(repr(infile.readline()))
print(repr(infile.readline()))
infile.close()

infile = open("Presidents.txt", "r")
print("Using readlines():", infile.readlines())
infile.close()
```

Appending Data to an Existing File

Open the file in append mode ("a"). The file pointer is placed at the end; all writes go to the end without erasing existing content.

```
def main():
    outfile = open("Info.txt", "a")
    outfile.write("\nPython is interpreted\n")
    outfile.close()

main()
```

If `Info.txt` originally contained `"Programming is fun."`, after running it will contain:

```
Programming is fun.  
Python is interpreted
```

Full Example:

```
import os.path  
  
if os.path.isfile("test.txt"):  
    print("File already exists. Choose a different  
name.")  
else:  
    myFile = open("test.txt", "w")  
    for i in range(0,50):  
        myFile.write(f"This is Student number {i}  
\n")  
    myFile.close()  
  
myFile = open("test.txt", "r")  
print(myFile.read(100))  
print(myFile.readline(10))  
print(myFile.readlines(50))
```

Exception Handling Basics

When an error occurs, Python raises an exception and prints a traceback. To handle exceptions gracefully, use `try/except`.

Handling Multiple Exceptions and Using `finally`

You can have multiple `except` clauses for different exception types. The `finally` block (if present) is always executed, regardless of whether an exception occurred – ideal for cleanup.

```
try:
    num = int(input("Enter an integer: "))
    result = 10 / num
    print("Result:", result)
except ValueError:
    print("That's not a valid integer.")
except ZeroDivisionError:
    print("Cannot divide by zero.")
finally:
    print("Execution finished.") # always printed
```