

Programming Foundations in Python

Prepared by Dr. Hamada I. AbdulWakel

Lecture 10





مجموعة جامعة
الجميع الذكية
للتعليم القابضة

MIDOCEAN
UNIVERSITY
BY EVERYONE'S SMART UNIVERSITY

File Handling and Graphical User Interface

Lecture 10

Prepared by
Dr. Hamada I. AbdulWakel

www.midocean.ae

Lecture

10

RECAP: LECTURE 9

- The operator overloading.
- Inheritance and polymorphism.

Lecture

10

LECTURE MAIN POINTS

- The meaning of file handling.
- The file operations.
- The Command-line vs. Graphical-user interfaces.
- The GUI module in python.

FILES

- Since RAM is volatile (which loses its data when the computer is turned off).
- **Files** are named locations on disk to store related information.
- They are used to permanently store data in a non-volatile memory (e.g. hard disk).



Lecture

10

FILE OPENING

- The syntax of opening a file is:
`File_object_name = open(filename, mode)`
- The "filename" is a **string** argument which specify filename along with its path.
- The mode is a **string** argument which used to specify how file will be used.
 - The reading mode represented by "r".
 - The writing mode represented by "w".
 - The appending mode represented by "a".

FILE CLOSING

- When we are done with performing operations on the file, we need to properly close the file.
- Closing a file will free up the resources that were tied with the file.
- The syntax for closing a file is:

```
File_object_name.close()
```

FILE CLOSING

Cont'd

- If an exception occurs when we are performing some operation with the file, the code exits without closing the file. So, we use **try .. finally** statement.

try:

```
File_object_name = open(filename, mode)
```

finally:

```
File_object_name.close()
```

FILE CLOSING

Cont'd

- The best way to close a file is by using the **with** statement.
- This ensures that the file is closed when the block inside the with statement is exited.
- If an exception occurs before the end of the block, it will close the file before the exception is caught.

```
with open(filename, mode) as f:  
    #file operations
```

Lecture

10

FILE READING DATA

- We must open the file in reading **r** mode.
- To read data back from the file you need one of these methods:
 - **read([number])**: return specified number of characters from the file.
 - If the number is omitted, it will read the entire content.
 - **readline()**: returns the next line of the file.
 - **readlines()**: read all the lines as a list of strings.
- **FileNotFoundError** is the error generated if the file you try to read is not exists.

FILE READING DATA

Cont'd

- Code:

```
f = open("myfile.txt", "r")    # open file in current directory
f = open("D:\\path\\to\\myfile.txt", "r")    # specifying full path in Windows
f = open("/path/to/myfile.txt", "r")    # specifying full path in Linux
# return the file name
print( f.name )
# return the file opening mode
print( f.mode )
# check if the file close or not
print( f.closed )
```

FILE READING DATA

Cont'd

- Code:

```
f = open("myfile.txt", "r")  
f = open("D:\\path\\to\\myfile.txt", "r") # Windows  
f = open("/path/to/myfile.txt", "r") # Linux  
print( f.read(10) )  
print( f.read() )  
print( f.readline() )  
lines = f.readlines()  
for i in lines:  
    print(i)  
f.close()
```

Lecture

10

FILE READING DATA

Cont'd

- Code:

The screenshot displays a Python IDE environment with three overlapping windows:

- test.py**: Contains the following Python code:

```
1 f = open("D:\\test\\test.txt", "r")
2 print(f.read())
3 f.close()
```
- File Explorer**: Shows the directory `D:\test` containing a file named `test` (Text Document, 4 bytes).
- test - Notepad**: Displays the content of the file `test`, which is the word `test`.

The IDE's terminal window shows the execution of the script:

```
Windows PowerShell
Copyright (C) 2016 Microsoft Corporation. All rights reserved.

PS D:\python> & C:/Users/pc/AppData/Local/Programs/Python/Python313/python.exe d:/python/test.py
test
PS D:\python>
```

FILE WRITING DATA

Cont'd

- We need to open it in write w, append a or exclusive creation x mode.
- We need to be careful with the w mode, as it will overwrite into the file if it already exists.
- Code:

```
import os
```

```
os.chdir("D:\\path\\to\\the\\folder")
```



```
chdir("/path/to/the/folder/")
```

```
f.write("My Name: Hamada I.")
```

```
f.write("I'm a teacher at \n" )
```



```
with open("myfile.txt", "w") as f:
```

FILE WRITING DATA

Cont'd

- Code:



```
import os  
os.chdir("D:\\path\\to\\the\\folder")  
with open("myfile.txt", "a") as f:  
    f.write("Midocean University")
```



```
os.chdir("/path/to/the/folder/")
```

Lecture

10

FILE RENAMING AND DELETING

- The os module provides methods that help to perform file-processing operations, such as renaming and deleting.
- To rename an existing file, rename() method is used.
- It takes two arguments, current file name and new file name.
- Syntax:

```
os.rename(current_file_name, new_file_name)
```

```
os.remove(filename)
```

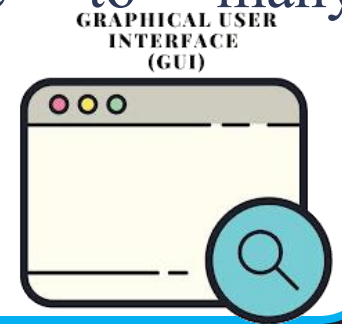
COMMAND LINE INTERFACE

- **The command-line interface (CLI):** takes input from the user via text commands entered in a terminal or console window.
- **CLI features:**
 - Based on commands a user inputs in the terminal.
 - More efficient and faster since it doesn't require much memory or processing power as the entire interaction is performed in a terminal window.
 - Requires memorizing commands, which sometimes have complex syntax and many arguments.
 - Takes input from the keyboard.
 - Users cannot change CLI's appearance.



GRAPHICAL USER INTERFACE

- **The graphical-user interface (GUI) version:** based on graphics and visual elements, allowing users to interact with the system using a keyboard and a mouse.
- **GUI features:**
 - Based on visual elements such as icons, buttons, windows, dialog boxes, etc.
 - Requires more memory and processing power due to many graphical components that need to be loaded.
 - Icons method reduces the required knowledge level.
 - Takes input from the mouse and keyboard.
 - Appearance can be modified to suit the users' needs.

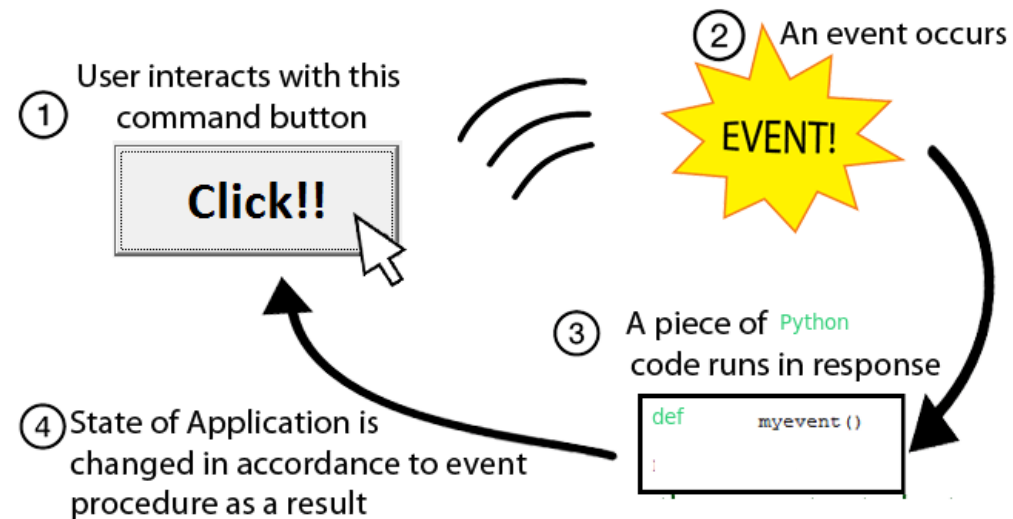


Lecture

10

EVENT-DRIVEN PROGRAMMING

- In GUI, the user-generated events (i.e., mouse clicks, button clicks) trigger operations in the program to respond by pulling in inputs, processing them, and displaying results.



Lecture

10

TKINTER GUI MODULES

- For GUI, there are a number of modules used to create UI:
 - PyGUI.
 - Wax.
 - Pyforms.
 - PySimpleGUI.
 - Tkinter.
 -, and more

Lecture

10

TKINTER MODULE

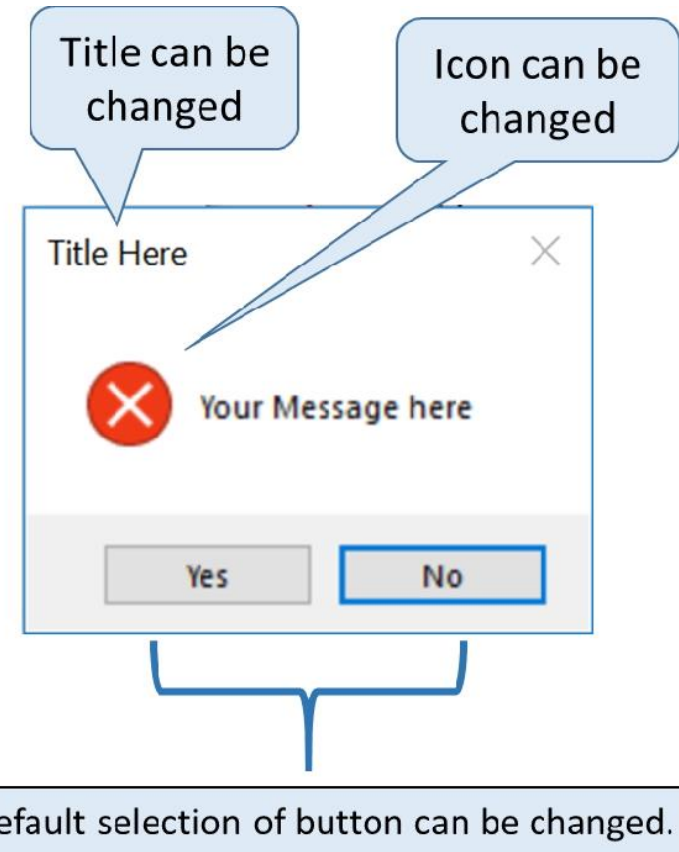
- Tkinter is the standard GUI library for Python.
- Tkinter provides a fast and easy way to create GUI applications.
- Tkinter provides a powerful object-oriented interface to the Tk GUI toolkit.
- Tkinter is included with Python as a library.
- To use it: `from tkinter import *`

Lecture

10

TKINTER MESSAGE BOXES

- The messagebox module is used to display the message boxes in the python applications.
- There are the various functions which are used to display the relevant messages depending upon the application requirements.



TKINTER MESSAGE BOXES

Cont'd

- To create a messagebox:

`messagebox.function_name(title, message [, options])`

- **function_name:** It represents an appropriate message box function.
- **title:** It is a string which is shown as a title of a message box.
- **message:** It is the string to be displayed as a message on the message box.
- **options:** There are various options which can be used to configure the message dialog box.

TKINTER MESSAGE BOXES

Cont'd

- Code:

```
from tkinter import messagebox

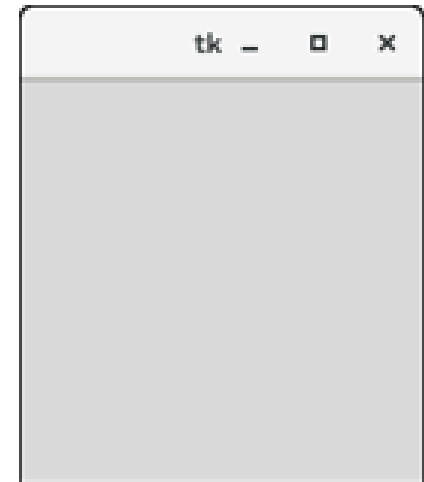
messagebox.showinfo("title", "Information")
messagebox.showwarning("title", "Warning")
messagebox.showerror("title", "Error")
messagebox.askquestion("title", "Do you love GUI?")
response = messagebox.askyesnocancel("title", "Do you love GUI?")
print(response)
```

Lecture

10

TKINTER FRAME

- A frame is a rectangular region on the screen.
- It is used to organize a group of widgets.
- An empty Tkinter top-level window can be created by using the following steps:
 - import the Tkinter module.
 - Create the main application window.
 - Add the widgets like labels, buttons, frames, etc.
 - Call the main event loop so that listening to the events and perform the actions on the frame.



TKINTER FRAME

Cont'd

- Code:

```
from tkinter import *
```

```
mainFrame = Tk()
```

```
mainFrame.title("The main frame title")
```

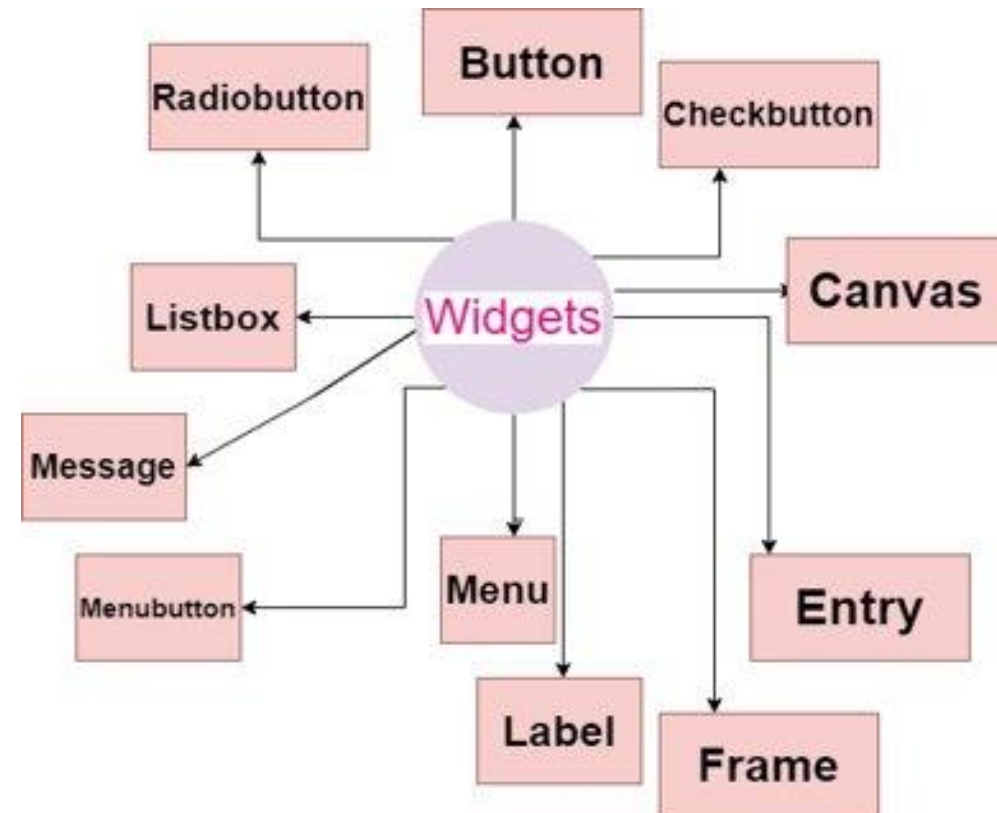
```
mainFrame.geometry("300x300")
```

```
mainFrame.mainloop() #listens for events
```

Lecture

10

TKINTER WIDGETS



TKINTER WIDGETS

Cont'd

- A widget is a graphical component on the screen.
- Any widget can be added using this syntax:

```
widget_var_name = widgetName(parent, option = placeholder)
```

Where **parent** is the argument for the parent frame while **option** is a placeholder that can have various values differ from widget to the other.

TKINTER WIDGETS

Cont'd

- **Label widget:**
 - Display box where you can place text or images.
 - The text displayed by this widget can be updated at any time you want.

```
label = Label(root, text = "Label text" )
```

- **Button widget:**
 - Used to perform an action when clicked.
 - You can attach a function or a method to a button, which is called automatically when you click the button.

```
btnOk = Button(root, text="Ok", fg="red", bd=2, bg="light blue")
```

TKINTER WIDGETS

Cont'd

- **Entry widget:**

- Used to accept single-line text strings from a user.

```
usName = Entry(root, width=20)
```

- **Checkbutton widget:**

- Used to display a number of options to a user as toggle buttons.

```
var1 = IntVar()
```

```
var2 = IntVar()
```

```
CBtn1 = Checkbutton(root, text="Option 1", variable=var1, onvalue=1, offvalue=0)
```

```
CBtn2 = Checkbutton(root, text="Option 2", variable=var2, onvalue=1, offvalue=0)
```

- **Radiobutton widget:**
 - implements a multiple-choice button, which let user choose only one of them.
 - In order to implement this functionality, each group of radiobuttons must be associated to the same variable.

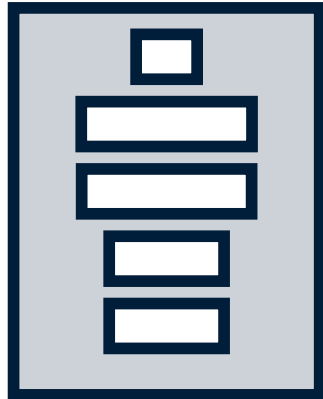
```
var1 = StringVar()
```

```
RBbtn1 = Radiobutton(root, text = "Option1", variable = var1, value = 1)
```

```
RBbtn2 = Radiobutton(root, text = "Option2", variable = var1, value = 2)
```

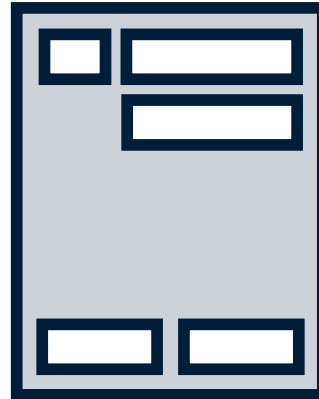
TKINTER GEOMETRY MANAGER

- The geometry measurements allow you to organize the widgets and throughout the parent frames or parent widget area.



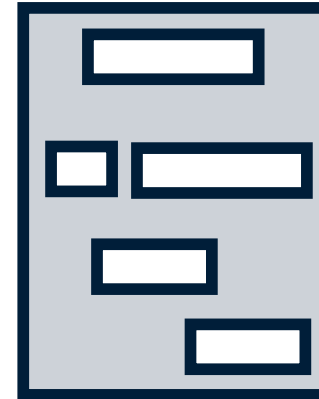
Pack

One after another,
vertically or horizontally



Grid

Assigned to grid cells,
using rows and columns



Place

Put stuff anywhere,
using coordinates

TKINTER GEOMETRY MANAGER

Cont'd

- The geometry allows you to organize the widgets on the parent frames.
 - **pack()**: It organizes the widgets in a block manner, and the complete available width is occupied by it.
`widget.pack(options)`
 - **grid()**: It organizes the widgets in a table-like structure.
`widget.grid(options)`
 - **place()**: Its purpose is to place the widgets at a specific position as instructed by the user in the parent widget.
`widget.place(options)`

TKINTER GEOMETRY MANAGER

Cont'd

- Code:

```
username = Label(root, text="User Name:").pack(side=TOP)
```

```
username = Label(root, text="User Name:").grid(row=0)
```

```
username = Label(root, text="User Name:").place(x = 30, y = 50)
```

Lecture

10

BUTTON EXAMPLE

- Code:

```
import tkinter
from tkinter import messagebox
top = tkinter.Tk()
def helloCallBack():
    messagebox.showinfo( "Hello Python", "Hello World")

B = tkinter.Button(top, text ="Hello", command = helloCallBack)
B.pack()
top.mainloop()
```

Lecture

10

CALCULATOR EXAMPLE

- Code:

```
from tkinter import *  
expression = ""
```

```
def press(num):  
    global expression  
    expression = expression + str(num)  
    equation.set(expression)
```

CALCULATOR EXAMPLE

Cont'd

- Code:

```
def equalpress():  
    global expression  
    total = str(eval(expression))  
    equation.set(total)  
    expression = ""  
  
def clear():  
    global expression  
    expression = ""  
    equation.set("")
```

CALCULATOR EXAMPLE

Cont'd

- Code:

```
gui = Tk()
gui.title("Simple Calculator")
gui.geometry("250x130")
equation = StringVar()
expression_field = Entry(gui, textvariable=equation)
expression_field.grid(columnspan=2)
button1 = Button(gui, text=' 1 ', fg='black', bg='red',
                  command=lambda: press(1), height=1, width=7)
button1.grid(row=2, column=0)
```

CALCULATOR EXAMPLE

Cont'd

- Code:

```
button2 = Button(gui, text=' 2 ', fg='black', bg='red',  
                 command=lambda: press(2), height=1, width=7)  
button2.grid(row=2, column=1)  
plus = Button(gui, text=' + ', fg='black', bg='red',  
              command=lambda: press("+"), height=1, width=7)  
plus.grid(row=3, column=0)  
equal = Button(gui, text=' = ', fg='black', bg='red',  
               command=equalpress, height=1, width=7)  
equal.grid(row=3, column=1)
```

CALCULATOR EXAMPLE

Cont'd

- Code:

```
clear = Button(gui, text='Clear', fg='black', bg='red',  
               command=clear, height=1, width=7)  
clear.grid(row=4, column=0)
```

```
# start the GUI  
gui.mainloop()
```

Lecture

10

QUESTIONS FOR ONLINE DISCUSSION

- Suppose we have this hours.txt data. Compute each worker's total hours and hours/day.

123 Suzy 9.5 8.1 7.6 3.1 3.2

456 Brad 7.0 9.6 6.5 4.9 8.8

789 Jenn 8.0 8.0 8.0 8.0 7.5

- Complete the calculator example with other arithmetic operators.

Lecture

10

PYTHON NEW UPDATES

- Performance is King:
 - The core team is deeply committed to making CPython faster without breaking existing code (the "Faster CPython" project).
- Developer Experience:
 - The focus on clearer error messages is a multi-year project that dramatically helps beginners and experts alike.
- Governance:
 - The language has a clear, community-driven direction with annual release cycles.

PATTERN MATCHING

- Pattern Matching (or **match-case**) is a feature that allows you to check a value against a pattern and execute code based on that pattern.
- It is more powerful than traditional **switch-case** statements in other languages because it can destructure data types and match on patterns, not just values.

CALCULATOR EXAMPLE

- Basic Syntax:

match value:

case pattern_1:

code executed if pattern 1 matches

case pattern_2:

code executed if pattern 2 matches

case _:

code executed if no pattern matches (like else)

Lecture

10

CALCULATOR EXAMPLE

- Example 1 (value matching):

```
def describe_day(day):
```

```
    match day:
```

```
        case "Saturday" | "Sunday":
```

```
            return "Weekend"
```

```
        case "Friday":
```

```
            return "Friday - blessed day"
```

```
        case _:
```

```
            return "Unknown day"
```

```
print(describe_day("Friday")) # Friday - blessed day
```

Lecture

10

CALCULATOR EXAMPLE

- Example 2 (extracting values from list):

```
def process_list(lst):
```

```
    match lst:
```

```
        case []:
```

```
            print("Empty list")
```

```
        case [single]:
```

```
            print(f"Single element list: {single}")
```

```
        case [first, second, *rest]:
```

```
            print(f"First: {first}")
```

```
            print(f"Second: {second}")
```

```
            print(f"Remaining: {rest}")
```

```
process_list([1, 2, 3, 4, 5])
```

TYPE HINTS

- Type hints (or type annotations) are a way to add static type information to Python code.
- Benefits:
 - **Better IDE Support:** Autocomplete, jump to definition, refactoring
 - **Early Bug Detection:** Catch type errors before runtime
 - **Improved Documentation:** Types serve as documentation
 - **Better Architecture:** Forces you to think about data flow
 - **Easier Refactoring:** Safe to change code with type checking

Lecture

10

CALCULATOR EXAMPLE

- Annotations:

Variable annotations

x: int = 10

y: float = 3.14

name: str = "John"

is_valid: bool = True

data: bytes = b"hello"

Lecture

10

CALCULATOR EXAMPLE

- Annotations:

Collections annotations

```
numbers: list[int] = [1, 2, 3]
```

```
mixed_list: list[int | str] = [1, "two", 3]
```

```
point: tuple[int, int] = (10, 20) # Tuples (fixed size)
```

```
items: tuple[int, ...] = (1, 2, 3, 4) # Variable-length tuple
```

Dictionaries

```
person: dict[str, str | int] = {"name": "Alice", "age": 30}
```



مجموعة جامعة
الجميع الذكية
للتعليم القابضة

MIDOCEAN
UNIVERSITY
BY EVERYONE'S SMART UNIVERSITY



Thank You

