Contents

CustomTkinter Cheat Sheet

Semester Project & Exam Revision
Visit Documentation

1. Basics

- **Library**: Modern UI layer built on Tkinter
- **Python**: 3.7+
- **Import**: `import customtkinter as ctk`

```
import customtkinter as ctk

ctk.set_appearance_mode("dark")      # "dark" | "light" | "system"
ctk.set_default_color_theme("blue") # "blue" | "dark-blue" | "green"

app = ctk.CTk()                      # Main window
app.geometry("500x300")
app.title("My App")
app.mainloop()
```

2. Core Widgets

Widget	Purpose	Key Options
CTkLabel	Display text	text, font, text_color
CTkButton	Clickable button	text, command, fg_color
CTkEntry	Text input	placeholder_text, show
CTkFrame	Container	fg_color, corner_radius
CTkSwitch	ON / OFF toggle	variable, command
CTkCheckBox	Checkbox	onvalue, offvalue
CTkSlider	Slider (range)	from_, to, command
CTkProgressBar	Progress indicator	set(0–1)

3. Widget Examples

Buttons & Entry

```
btn = ctk.CTkButton(app,
    text="Click Me",
    command=lambda: print("Clicked"))
btn.pack(pady=10)

entry = ctk.CTkEntry(app,
    placeholder_text="Email")
entry.pack(pady=10)
```

Switch & Checkbox

```
mode_var = ctk.StringVar(value="off")

switch = ctk.CTkSwitch(app,
    text="Dark Mode",
    variable=mode_var,
    onvalue="on",
    offvalue="off")
switch.pack()
```

Slider + ProgressBar

```
def update(val):
    progress.set(float(val)/100)

slider = ctk.CTkSlider(app,
    from_=0, to=100,
```

```
    command=update)
slider.pack()

progress = ctk.CTkProgressBar(app)
progress.pack()
```

4. Layout Managers (WITH EXAMPLES)

Layout managers control how widgets are positioned inside their parent container.

Import (required once in the file):

```
import customtkinter as ctk
```

Manager	Main Usage
pack	Simple layouts (menus, vertical lists, sidebars)
grid	Structured layouts (forms, tables)
place	Absolute positioning (avoid in real projects)

pack() Example and Explanation

Use case: Vertical menu or list of buttons

```
btn1 = ctk.CTkButton(app, text="Home")
btn1.pack(pady=10)

btn2 = ctk.CTkButton(app, text="Settings")
btn2.pack(pady=10)
```

Explanation:

- Widgets are stacked automatically
- `pady=10` adds vertical spacing
- Order of code = order on screen

Exam note: Use **pack()** for navigation menus and sections.

grid() Example and Explanation

Use case: Form (Label + Entry)

```
ctk.CTkLabel(app, text="Name").grid(row=0, column=0, padx=10, pady=10)
name_entry = ctk.CTkEntry(app)
name_entry.grid(row=0, column=1)
```

Explanation:

- row = vertical position
- column = horizontal position
- grid aligns widgets perfectly

Important Rule (EXAM):

- Never mix **pack()** and **grid()** in the same container

place() Example (Not Recommended)

```
btn.place(x=100, y=50)
```

Why avoid place():

- Not responsive
- Breaks on window resize
- Penalized in exams

5. Frames & Organization (WITH USAGE)

Frames are containers used to group widgets.

Import:

```
import customtkinter as ctk
```

Example: Sidebar + Main Content

```
# Sidebar frame
sidebar = ctk.CTkFrame(app, width=200)
sidebar.pack(side="left", fill="y")

# Main content frame
main = ctk.CTkFrame(app)
main.pack(side="right", fill="both", expand=True)
```

Explanation:

- Sidebar has fixed width
- fill="y" fills vertical space
- expand=True allows resizing

Why Frames are important:

- Clean code
- Easy CRUD layouts
- Required for large apps

Scrollable Frame (Database Lists)

```
scroll = ctk.CTkScrollableFrame(
    main,
    width=300,
    height=200
)
scroll.pack(pady=10)
```

Typical Usage:

- Display database rows
- Contact lists
- Product inventories

6. Themes & Appearance (WITH EXAMPLES)

CustomTkinter supports modern UI themes.

Import:

```
import customtkinter as ctk
```

Appearance Mode

```
ctk.set_appearance_mode("dark")   # dark | light | system
```

Explanation:

- Changes entire application style
- Can be controlled by a switch

Color Theme

```
ctk.set_default_color_theme("green")
```

Affects:

- Buttons
- Sliders
- Switches

Widget Color Customization

```
btn = ctk.CTkButton(
    app,
    text="Save",
    fg_color="#3498db",
    hover_color="#2980b9"
)
btn.pack()
```

Exam Tip:

- Custom colors = UX improvement
- Shows CustomTkinter mastery

7. Windows & Dialogs (WITH USAGE)

Used for secondary windows (edit, settings).

Import:

```
import customtkinter as ctk
```

Toplevel Window Example

```
top = ctk.CTkToplevel(app)
top.title("Edit Record")
top.geometry("400x300")
```

Explanation:

- Separate window
- Shares same theme

Modal Window

```
top.grab_set()
```

Why use grab_set():

- Blocks main window
- Forces user to finish action
- Used in Update/Delete dialogs

Typical Exam Examples:

- Edit user form
- Confirmation window
- Settings dialog

10. Complete Layout Example (EXPLAINED)

Goal: Create a modern application layout with:

- A left sidebar (navigation menu)
- A main content area
- Scrollable content (lists, contacts, settings)

Example: Sidebar + Main Content

```
import customtkinter as ctk

# Create main application window
app = ctk.CTk()
app.title("Layout Avance")
app.geometry("700x500")

# ----------------------
# LEFT SIDEBAR (MENU)
# ----------------------

# Create a frame for the sidebar
sidebar = ctk.CTkFrame(
    app,
    width=200,          # Fixed width
    corner_radius=0     # Sharp edges (modern look)
)

# Position sidebar on the left, fill vertically
sidebar.pack(side="left", fill="y")

# Sidebar title
ctk.CTkLabel(
    sidebar,
    text="Menu",
    font=("Arial", 20, "bold")
).pack(pady=20)

# Navigation buttons
ctk.CTkButton(sidebar, text="Accueil", width=180).pack(pady=10)
ctk.CTkButton(sidebar, text="Parametres", width=180).pack(pady=10)
ctk.CTkButton(sidebar, text="a propos", width=180).pack(pady=10)

# ----------------------
# MAIN CONTENT AREA
# ----------------------

# Main frame takes remaining space
main_frame = ctk.CTkFrame(app)

main_frame.pack(
    side="right",
    fill="both",        # Fill width + height
    expand=True,        # Take all remaining space
    padx=20,
    pady=20
)

# Main title
ctk.CTkLabel(
    main_frame,
    text="Contenu Principal",
    font=("Arial", 24, "bold")
).pack(pady=20)

# ----------------------
# SCROLLABLE CONTENT
# ----------------------

# Scrollable container (for lists, contacts, etc.)
scrollable = ctk.CTkScrollableFrame(
    main_frame,
    width=400,
    height=300
)
scrollable.pack(pady=10)
```

```
# Dynamic content generation
for i in range(20):
    ctk.CTkLabel(
        scrollable,
        text=f"elément {i+1}"
    ).pack(pady=5)

# Start application loop
app.mainloop()
```

Why This Layout Is Important (EXAM POINTS)

- **Frames** separate UI section*s (clean structure)
- **Sidebar** is used for navigation (professional apps)
- **ScrollableFrame** replaces Listbox for modern UI
- **pack()** is ideal for vertical menus and section*s
- **expand=True** allows responsive resizing

Typical Uses:

- Contact manager (list of contacts)
- Settings panel
- Dashboard / admin interface
- Multi-page applications

8. Best Practices (EXAM GOLD)

- Always use **CTk()** instead of Tk()
- Use **Frames** to structure layouts
- Keep UI logic separate from data logic
- Prefer **grid** for forms, **pack** for section*s
- Use **CTkScrollableFrame** for dynamic lists
- Validate input before saving (email, empty fields)
- Store data in **JSON** for projects
- Dark/Light switch = instant bonus points

9. Migration Reminder

Tkinter	CustomTkinter	
Tk	CTk	
Label	CTkLabel	
Button	CTkButton	
Entry	CTkEntry	
Frame	CTkFrame	
Checkbutton	CTkCheckBox	
Scale	CTkSlider	
Progressbar	CTkProgressBar	

11. CRUD Project Essentials (END-SEMESTER CORE)

Goal: Build a CRUD application using:

- CustomTkinter for UI
- MySQL for data storage
- Forms + Buttons + Database

Typical CRUD Projects:

- Contact Management
- Student Records
- Product Inventory

CRUD = Create, Read, Update, Delete

12. Application Structure (WITHOUT POO)

Structure Used:

- One main window
- Frames for layout
- Functions for actions (add, update, delete)

```
import customtkinter as ctk
import mysql.connector

ctk.set_appearance_mode("dark")
ctk.set_default_color_theme("blue")

app = ctk.CTk()
app.title("CRUD Application")
app.geometry("900x600")
```

Why this is OK:

- Simple
- Easy to debug
- Accepted in exams

13. Database Connection (MySQL)

```
def connect_db():
    return mysql.connector.connect(
        host="localhost",
        user="root",
        password="",
        database="crud_db"
    )
```

Used for:

- Insert
- Fetch
- Update
- Delete

14. Form Layout (Label + Entry + Grid)

Used for data input (CREATE / UPDATE).

```
form = ctk.CTkFrame(app)
form.pack(pady=20)

ctk.CTkLabel(form, text="Name").grid(row=0, column=0, padx=10, pady=10)
ctk.CTkLabel(form, text="Email").grid(row=1, column=0, padx=10, pady=10)

name_entry = ctk.CTkEntry(form, width=250)
email_entry = ctk.CTkEntry(form, width=250)

name_entry.grid(row=0, column=1)
email_entry.grid(row=1, column=1)
```

Why grid()?

- Perfect alignment
- Clean professional forms
- Recommended in exams

15. Validation + Error Messages

```
def validate_inputs():
    if name_entry.get() == "" or email_entry.get() == "":
        error_label.configure(text="All fields are required")
        return False

    if "@" not in email_entry.get():
        error_label.configure(text="Invalid email address")
        return False

    error_label.configure(text="")
    return True
```

```
error_label = ctk.CTkLabel(app, text="", text_color="red")
error_label.pack()
```

Why validation matters:

- Prevents invalid data
- Teacher expects it

16. CREATE (Insert Data)

```
def add_record():
    if not validate_inputs():
        return

    db = connect_db()
    cursor = db.cursor()

    query = "INSERT INTO users (name, email) VALUES (%s, %s)"
    cursor.execute(query, (
        name_entry.get(),
        email_entry.get()
    ))

    db.commit()
    db.close()
    fetch_data()
```

Flow: Entry Python MySQL UI refresh

17. READ (Fetch + Display Data)

```
list_frame = ctk.CTkScrollableFrame(app, width=400, height=300)
list_frame.pack(pady=20)
```

```
def fetch_data():
    for widget in list_frame.winfo_children():
        widget.destroy()

    db = connect_db()
    cursor = db.cursor()
    cursor.execute("SELECT * FROM users")
    rows = cursor.fetchall()
    db.close()

    for row in rows:
        btn = ctk.CTkButton(
            list_frame,
            text=f"{row[1]} - {row[2]}",
            command=lambda r=row: select_record(r)
        )
        btn.pack(fill="x", pady=5)
```

Key idea:

- Each row = clickable item
- Used for Update / Delete

18. UPDATE & DELETE (With Selected Item)

Step 1: Select Record

```
selected_id = None

def select_record(row):
    global selected_id
    selected_id = row[0]

    name_entry.delete(0, "end")
    email_entry.delete(0, "end")

    name_entry.insert(0, row[1])
    email_entry.insert(0, row[2])
```

Step 2: Update

```
def update_record():
    if selected_id is None:
        error_label.configure(text="Select a record first")
        return

    db = connect_db()
    cursor = db.cursor()

    query = "UPDATE users SET name=%s, email=%s WHERE id=%s"
    cursor.execute(query, (
        name_entry.get(),
        email_entry.get(),
        selected_id
    ))

    db.commit()
    db.close()
    fetch_data()
```

Step 3: Delete

```
def delete_record():
    if selected_id is None:
        error_label.configure(text="Select a record first")
        return

    db = connect_db()
    cursor = db.cursor()

    cursor.execute(
```

```
        "DELETE FROM users WHERE id=%s",
        (selected_id,)
    )

    db.commit()
    db.close()
    fetch_data()
```

Exam Explanation:

- Click item load data into form
- Update modifies selected row
- Delete removes selected row

11. CRUD Project Essentials (END-SEMESTER CORE)

Typical End-Semester Project:

- CRUD Application (Create, Read, Update, Delete)
- Database-backed (MySQL)
- Modern UI using CustomTkinter

Examples:

- Contact Manager
- Student Management System
- Product Inventory

12. POO (Class-Based) Application Structure

Why POO?

- Clean code
- Easy CRUD operations
- Teacher expects it in projects

```
import customtkinter as ctk

class CrudApp:
    def __init__(self):
        self.app = ctk.CTk()
        self.app.title("CRUD Application")
        self.app.geometry("900x600")

        self.create_sidebar()
        self.create_form()
        self.app.mainloop()
```

Key idea:

- `__init__()` initializes UI
- Each UI section* = separate method

13. Form Layout (Label + Entry + Grid)

Used for:

- Add record
- Update record
- Data validation

```
def create_form(self):
    self.form = ctk.CTkFrame(self.app)
    self.form.pack(padx=20, pady=20)

    # Labels
    ctk.CTkLabel(self.form, text="Name").grid(row=0, column=0, padx=10, pady=10)
    ctk.CTkLabel(self.form, text="Email").grid(row=1, column=0, padx=10, pady=10)

    # Entries
    self.name_entry = ctk.CTkEntry(self.form, width=250)
    self.email_entry = ctk.CTkEntry(self.form, width=250)

    self.name_entry.grid(row=0, column=1)
    self.email_entry.grid(row=1, column=1)

    # Submit button
    ctk.CTkButton(
        self.form,
        text="Add",
        command=self.add_record
    ).grid(row=2, column=0, columnspan=2, pady=15)
```

Why grid()?

- Perfect alignment
- Clean forms
- Exam-preferred for input layouts

14. Dark / Light Mode Switch (Sidebar)

```
def create_sidebar(self):
    self.sidebar = ctk.CTkFrame(self.app, width=200)
    self.sidebar.pack(side="left", fill="y")

    self.mode_switch = ctk.CTkSwitch(
        self.sidebar,
        text="Dark Mode",
        command=self.toggle_mode
    )
    self.mode_switch.pack(pady=30)

def toggle_mode(self):
    if self.mode_switch.get():
        ctk.set_appearance_mode("dark")
    else:
        ctk.set_appearance_mode("light")
```

Exam Tip:

- Switch controlling appearance = bonus UX points

15. Database Connection (MySQL)

Library Used: `mysql-connector-python`

```
import mysql.connector

def connect_db(self):
    return mysql.connector.connect(
        host="localhost",
        user="root",
        password="",
        database="crud_db"
    )
```

Why MySQL?

- Persistent storage
- Required for CRUD projects

16. Insert Data (CREATE)

```
def add_record(self):
    name = self.name_entry.get()
    email = self.email_entry.get()

    db = self.connect_db()
    cursor = db.cursor()

    query = "INSERT INTO users (name, email) VALUES (%s, %s)"
    cursor.execute(query, (name, email))

    db.commit()
    db.close()
```

Flow:

- Entry Python MySQL

17. Fetch and Display Data (READ)

```
def fetch_data(self):
    db = self.connect_db()
    cursor = db.cursor()
    cursor.execute("SELECT * FROM users")

    results = cursor.fetchall()
    db.close()

    for row in results:
        ctk.CTkLabel(
            self.scrollable,
            text=f"{row[1]} - {row[2]}"
        ).pack(pady=5)
```

Key Concept:

- Database UI
- ScrollableFrame = list view

19. Tkinter, VS Code, IDLE & Database Connection (EXAM CLARIFICATION)

Important Concept (Very Common Confusion)

IDLE and VS Code are ONLY code editors. They do not run separately, and they do not connect to each other.

- Tkinter does **NOT** belong to IDLE
- Database code does **NOT** belong to VS Code
- **Python** runs everything

Key Rule:

$$\textit{One Python program} = \textit{One execution} = \textit{One connection}$$

Can Tkinter Be Used in VS Code?

YES. Tkinter and CustomTkinter work perfectly in VS Code.

- IDLE = simple editor
- VS Code = advanced editor
- Execution is identical

Conclusion:

- You should use **only one editor**
- Mixing IDLE and VS Code is wrong and unnecessary

How Tkinter Connects to MySQL (XAMPP)

Connection Flow:

$$\text{Python} \rightarrow \text{Tkinter (UI)} \rightarrow \text{mysql-connector} \rightarrow \text{MySQL (XAMPP)}$$

- Tkinter handles the interface
- mysql-connector handles database access
- Python connects them together

Correct Project Structure

Recommended structure (Professional & Exam-Safe):

```
crud_project/
|
|-- main.py      (Tkinter / CustomTkinter UI)
|-- db.py        (Database connection + queries)
`-- requirements.txt
```

Important:

- All files run in the same Python environment
- Files communicate using **import**

Database File (db.py)

```
import mysql.connector

def connect_db():
    return mysql.connector.connect(
        host="localhost",
        user="root",
        password="",
        database="crud_db"
    )
```

Role:

- Handles MySQL connection
- Executes SQL queries

UI File (main.py)

```
import customtkinter as ctk
from db import connect_db   # Link UI to database

def add_user():
    db = connect_db()
    cursor = db.cursor()
    cursor.execute(
        "INSERT INTO users (name, email) VALUES (%s, %s)",
        (name_entry.get(), email_entry.get())
    )
    db.commit()
    db.close()

app = ctk.CTk()
name_entry = ctk.CTkEntry(app)
email_entry = ctk.CTkEntry(app)
ctk.CTkButton(app, text="Add", command=add_user).pack()
app.mainloop()
```

Explanation:

- Button click triggers a Python function
- Function sends data to MySQL
- UI and database are connected through Python

Exam Question → Answer (VERY IMPORTANT)

Q: Can Tkinter be used in VS Code?
A: Yes. Tkinter works in any Python editor including VS Code.

Q: Should UI code and database code be in different editors?
A: No. Editors do not affect execution. All code runs together in Python.

Q: How does Tkinter communicate with the database?
A: Through Python functions using mysql-connector.

Q: What role does XAMPP play?
A: XAMPP provides the MySQL server that Python connects to.

Final Golden Rule (MEMORIZE)

Editors do not matter. Python runs everything.

18. Exam Question Answer (VERY IMPORTANT)

Q: Why use CustomTkinter instead of Tkinter?
A: Modern UI, dark mode support, better UX, same logic as Tkinter.

Q: Why use Frames?
A: To organize UI into logical section*s (sidebar, content, forms).

Q: Why grid() for forms?
A: Ensures proper alignment of labels and input fields.

Q: How does CRUD work in your project?
A:

- Create: Insert data into MySQL
- Read: Fetch and display data
- Update: Modify selected record
- Delete: Remove record from database

Q: How do you connect UI with database?
A: Using mysql-connector and Python functions triggered by buttons.

Designed for fast revision, not long reading.