Code:

```
import os

import shutil

from pathlib import Path

from tkinter import Tk, filedialog, messagebox, StringVar

from tkinter import ttk

from collections import defaultdict


FILE_CATEGORIES = {

    "Images": [".jpg", ".jpeg", ".png", ".gif", ".bmp", ".svg", ".ico", ".webp", ".tiff", ".tif", ".heic"],

    "Videos": [".mp4", ".avi", ".mov", ".wmv", ".flv", ".webm", ".mkv", ".m4v", ".3gp", ".mpg", ".mpeg"],

    "Audio": [".mp3", ".wav", ".flac", ".aac", ".ogg", ".wma", ".m4a", ".opus", ".amr"],

    "Documents": [".pdf", ".doc", ".docx", ".xls", ".xlsx", ".ppt", ".pptx", ".txt", ".rtf", ".odt", ".ods", ".odp"],

    "Archives": [".zip", ".rar", ".7z", ".tar", ".gz", ".bz2", ".xz", ".iso", ".dmg"],

    "Code": [".py", ".js", ".html", ".css", ".java", ".cpp", ".c", ".h", ".cs", ".php", ".rb", ".go", ".rs", ".swift", ".kt", ".ts", ".jsx", ".tsx", ".json", ".xml", ".yaml", ".yml"],

    "Executables": [".exe", ".msi", ".deb", ".rpm", ".dmg", ".pkg", ".app"],

    "Spreadsheets": [".csv", ".xls", ".xlsx", ".ods"],

    "Presentations": [".ppt", ".pptx", ".odp", ".key"],

    "Others": []  # Default category for unrecognized files

}


def identify_file_type(file_path: Path) -> str:

    """

    Identify the file type based on its extension.


    Returns the category name (e.g., "Images", "Documents") or "Others" if not recognized.

    """

    extension = file_path.suffix.lower()
```

```
    for category, extensions in FILE_CATEGORIES.items():
        if extension in extensions:
            return category

    return "Others"

def create_category_folders(base_path: Path, categories: set[str]) -> None:
    """Create folders for each category that will be used."""
    for category in categories:
        folder_path = base_path / category
        folder_path.mkdir(exist_ok=True)

def organize_files(directory_path: Path, progress_callback=None) -> dict[str, int]:
    """
    Organize files in the specified directory by moving them into category folders.

    Args:
        directory_path: Path to the directory to organize
        progress_callback: Optional function to call with progress updates

    Returns:
        Dictionary with counts of files moved per category
    """
    if not directory_path.exists() or not directory_path.is_dir():
        raise ValueError(f"Invalid directory: {directory_path}")

    files_to_organize = []
    categories_needed = set()

    for item in directory_path.iterdir():
        if item.is_file():
```

```
        category = identify_file_type(item)
        files_to_organize.append((item, category))
        categories_needed.add(category)

create_category_folders(directory_path, categories_needed)
stats = defaultdict(int)
moved_count = 0
total_files = len(files_to_organize)

for file_path, category in files_to_organize:
    try:
        destination_folder = directory_path / category
        destination_path = destination_folder / file_path.name



        if destination_path.exists():
            base_name = file_path.stem
            extension = file_path.suffix
            counter = 1
            while destination_path.exists():
                new_name = f"{base_name}_{counter}{extension}"
                destination_path = destination_folder / new_name
                counter += 1

        shutil.move(str(file_path), str(destination_path))
        stats[category] += 1
        moved_count += 1

        if progress_callback:
            progress_callback(moved_count, total_files, file_path.name)
```

```
            except Exception as e:
                print(f"Error moving {file_path.name}: {e}")
                stats["Errors"] = stats.get("Errors", 0) + 1

        return dict(stats)

class FileOrganizerGUI:
    """GUI application for file organization."""

    def __init__(self, root: Tk):
        self.root = root
        self.root.title("File Organizer")
        self.root.geometry("600x400")
        self.root.resizable(False, False)

        self.selected_directory = None

        self.setup_ui()

    def setup_ui(self):
        """Set up the user interface components."""
        main_frame = ttk.Frame(self.root, padding="20")
        main_frame.pack(fill="both", expand=True)

        title_label = ttk.Label(
            main_frame,
            text="File Organizer",
            font=("Arial", 18, "bold")
        )
        title_label.pack(pady=(0, 20))
```

```
# Description
desc_label = ttk.Label(
    main_frame,
    text="Select a directory to organize files by type",
    font=("Arial", 10)
)
desc_label.pack(pady=(0, 20))

# Directory selection frame
dir_frame = ttk.Frame(main_frame)
dir_frame.pack(fill="x", pady=(0, 20))

self.dir_label = ttk.Label(
    dir_frame,
    text="No directory selected",
    font=("Arial", 9),
    foreground="gray"
)
self.dir_label.pack(side="left", fill="x", expand=True, padx=(0, 10))

browse_btn = ttk.Button(
    dir_frame,
    text="Browse Directory",
    command=self.browse_directory
)
browse_btn.pack(side="right")

info_frame = ttk.LabelFrame(main_frame, text="File Categories", padding="10")
info_frame.pack(fill="both", expand=True, pady=(0, 20))

categories_text = ", ".join([cat for cat in FILE_CATEGORIES.keys() if cat != "Others"])
```

```
info_label = ttk.Label(
    info_frame,
    text=f"Files will be organized into: {categories_text}",
    font=("Arial", 9),
    wraplength=550
)
info_label.pack(anchor="w")

self.progress_var = StringVar(value="Ready to organize")
self.progress_label = ttk.Label(
    main_frame,
    textvariable=self.progress_var,
    font=("Arial", 9)
)
self.progress_label.pack(pady=(0, 5))

self.progress_bar = ttk.Progressbar(
    main_frame,
    mode="determinate",
    length=560
)
self.progress_bar.pack(pady=(0, 20))



self.organize_btn = ttk.Button(
    main_frame,
    text="Organize Files",
    command=self.organize_files,
    state="disabled"
)
self.organize_btn.pack()
```

```
def browse_directory(self):
    """Open directory selection dialog."""
    directory = filedialog.askdirectory(title="Select Directory to Organize")

    if directory:
        self.selected_directory = Path(directory)
        self.dir_label.config(
            text=f"Selected: {self.selected_directory}",
            foreground="black"
        )
        self.organize_btn.config(state="normal")
        self.progress_var.set("Ready to organize")
        self.progress_bar["value"] = 0

def update_progress(self, current: int, total: int, filename: str):
    """Update progress bar and label."""
    percentage = (current / total) * 100 if total > 0 else 0
    self.progress_bar["value"] = percentage
    self.progress_var.set(f"Organizing: {filename} ({current}/{total})")
    self.root.update_idletasks()

def organize_files(self):
    """Handle the file organization process."""
    if not self.selected_directory:
        messagebox.showwarning("Warning", "Please select a directory first.")
        return


    response = messagebox.askyesno(
        "Confirm",
```

```
    f"Organize files in:\n{self.selected_directory}\n\n"
    "Files will be moved into category folders. Continue?"
)

if not response:
    return

try:
    self.organize_btn.config(state="disabled")
    self.progress_var.set("Starting organization...")
    self.progress_bar["value"] = 0

    def progress_callback(current, total, filename):
        self.update_progress(current, total, filename)

    stats = organize_files(self.selected_directory, progress_callback)



    total_moved = sum(stats.values())
    stats_text = "\n".join([f"{cat}: {count} files" for cat, count in stats.items()])

    messagebox.showinfo(
        "Success",
        f"Organization complete!\n\n"
        f"Total files organized: {total_moved}\n\n"
        f"Breakdown:\n{stats_text}"
    )

    self.progress_var.set(f"Complete! Organized {total_moved} files")
    self.progress_bar["value"] = 100
```

```
        except Exception as e:
            messagebox.showerror("Error", f"An error occurred:\n{str(e)}")
            self.progress_var.set("Error occurred")

        finally:
            self.organize_btn.config(state="normal")

def main():
    """Main entry point for the application."""
    root = Tk()
    app = FileOrganizerGUI(root)
    root.mainloop()

if __name__ == "__main__":
    main()
```

Output: