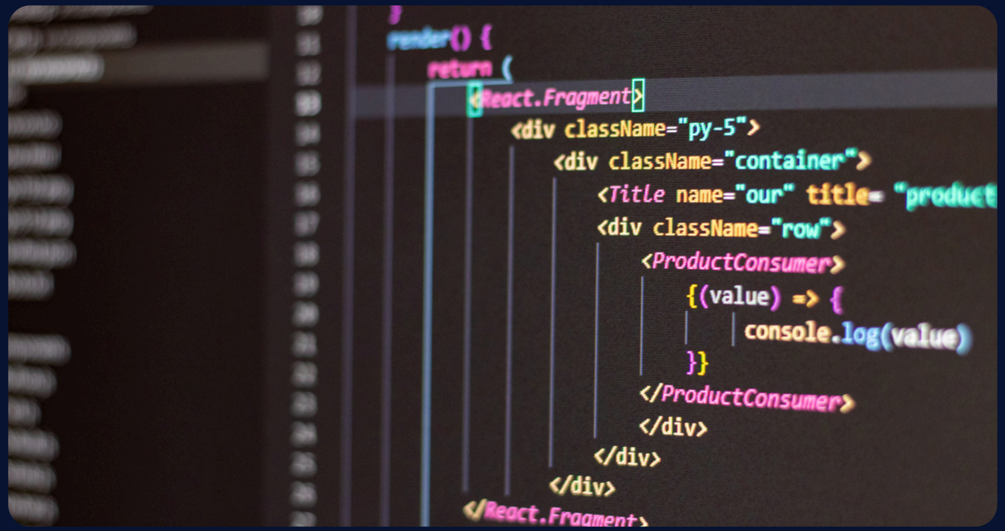


```
render() {
  return (
    <React.Fragment>
      <div className="py-5">
        <div className="container">
          <Title name="our" title= "product
          <div className="row">
            <ProductConsumer>
              {(value) => {
                console.log(value)
              }}
            </ProductConsumer>
          </div>
        </div>
      </div>
    </React.Fragment>
```

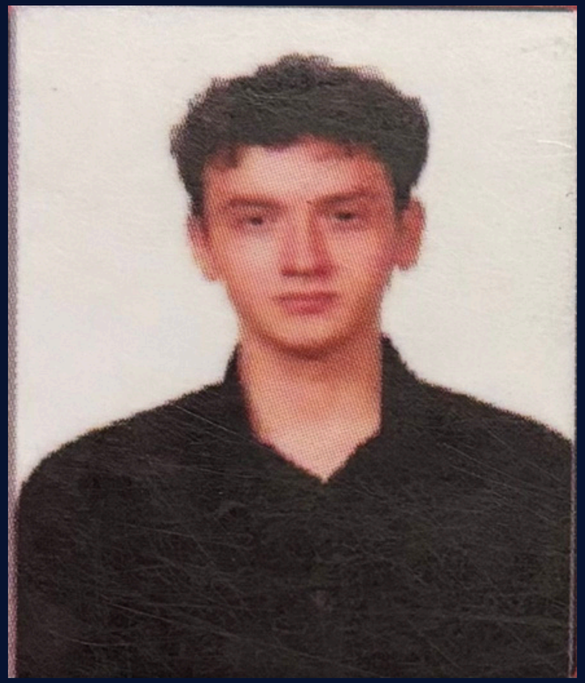
COE203 - P1

Presented By

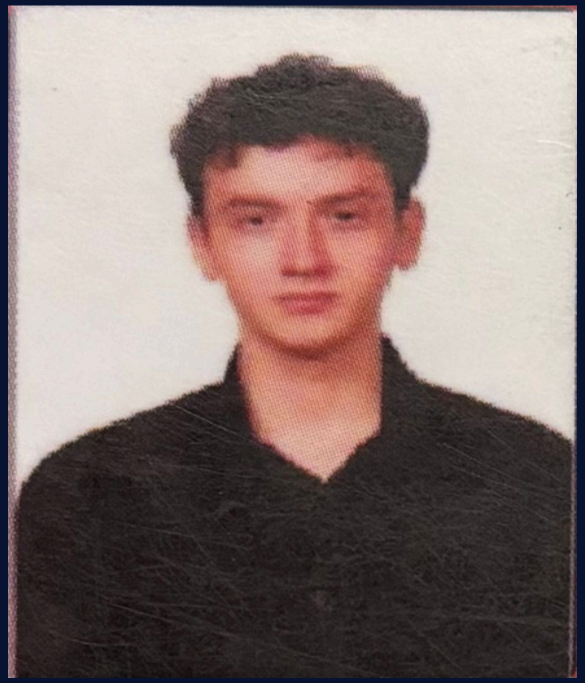
İSMET EFE BALIK

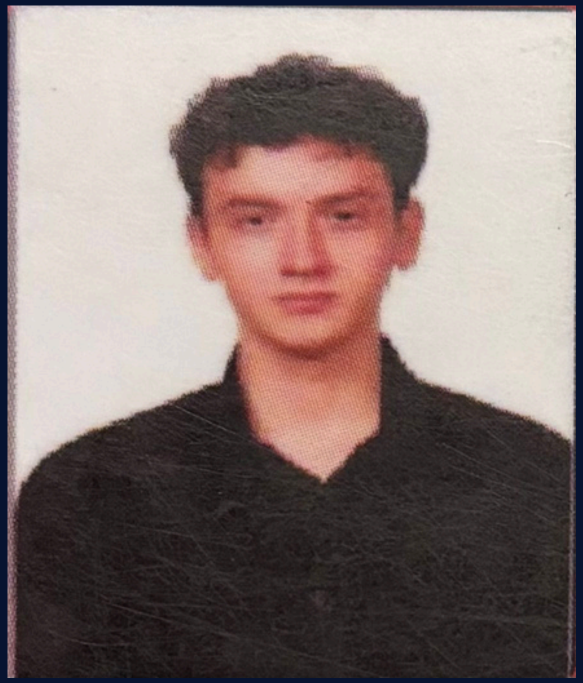
TEAM MEMBERS

TEAM LEAD


BACKEND ENG

FRONT ENG.


PM


DEVOPS

libraries

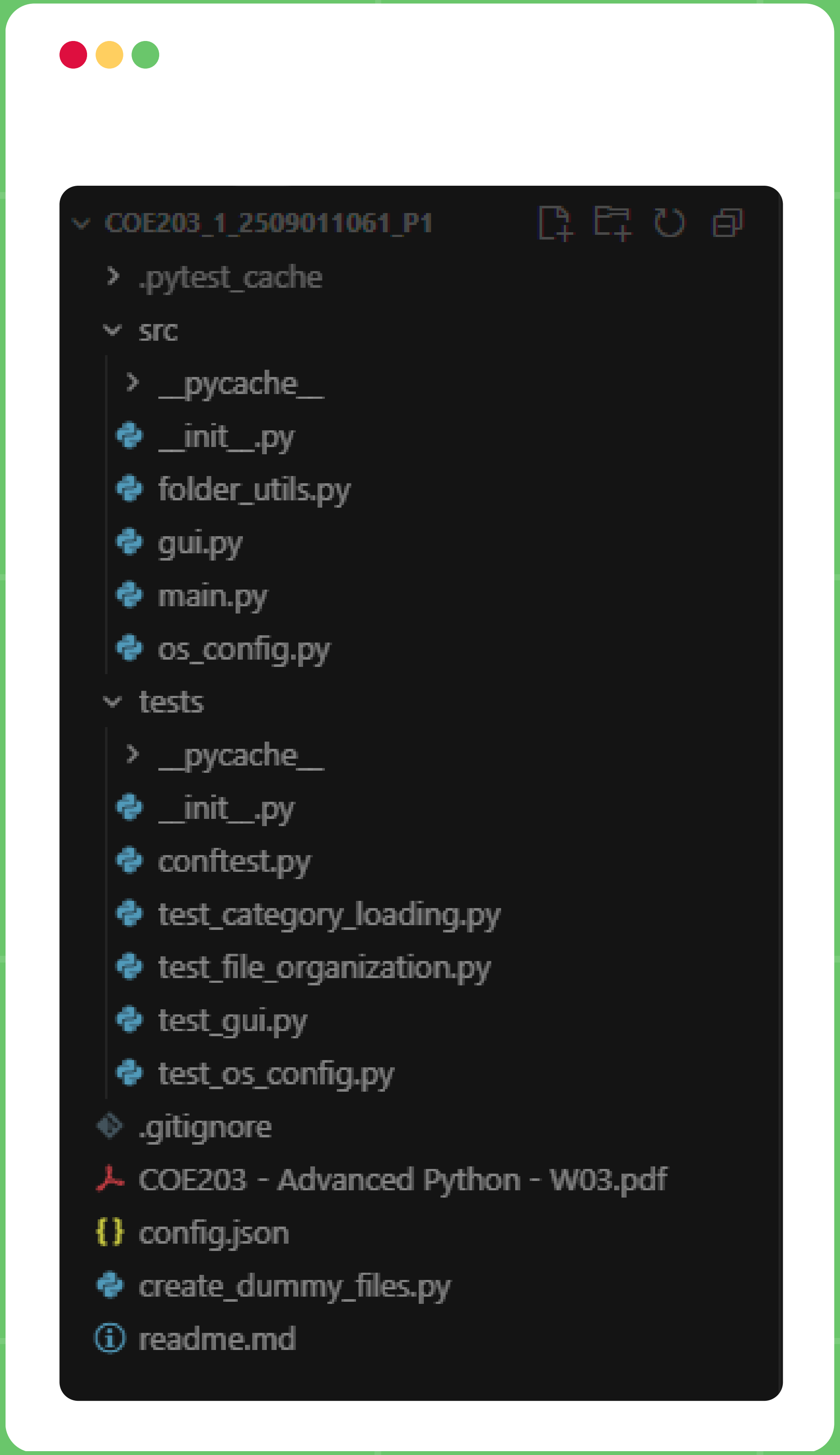
built-in

- sys
- os
- pathlib
- shutil
- tkinter

external

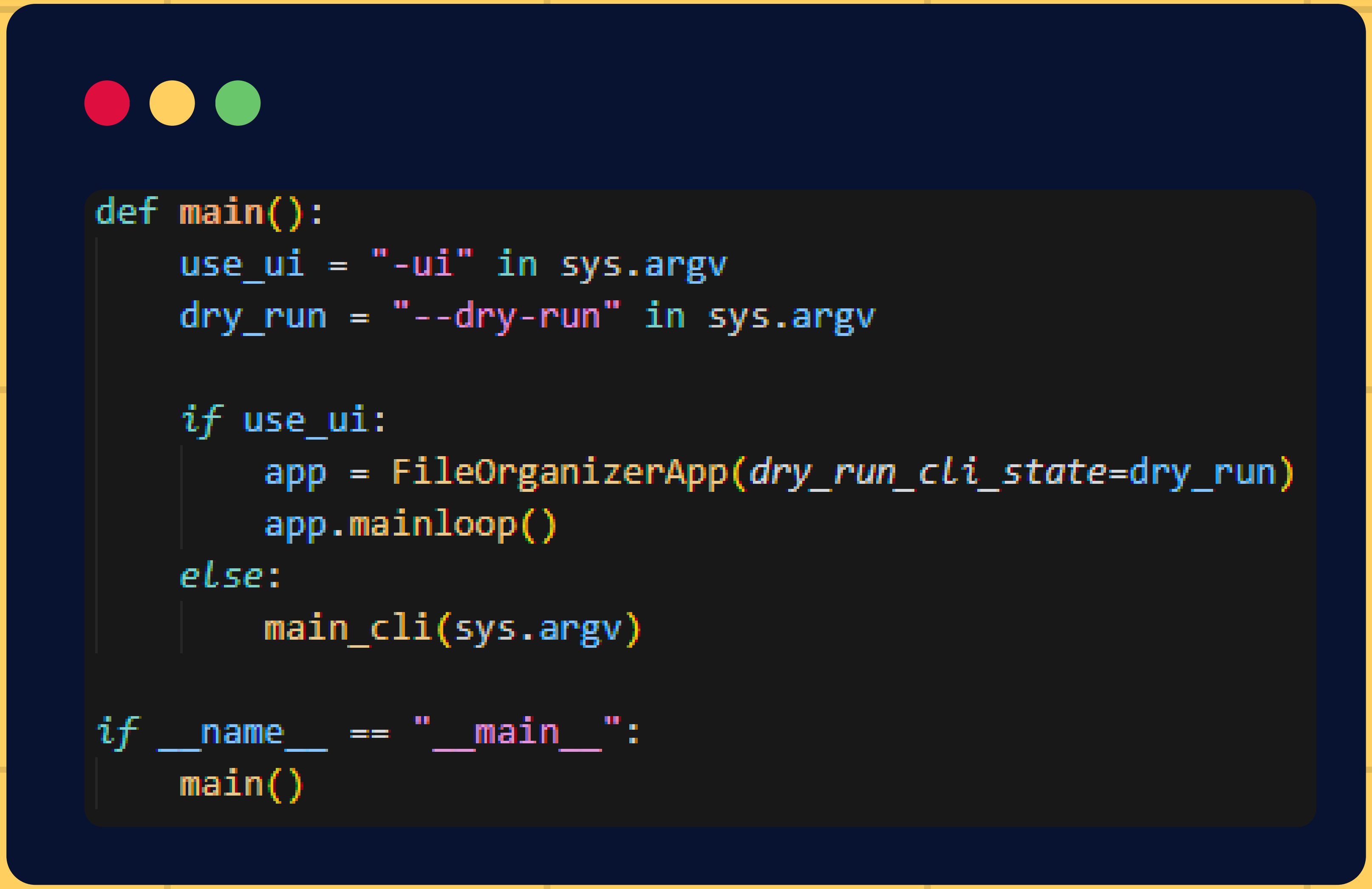
pytest

structure



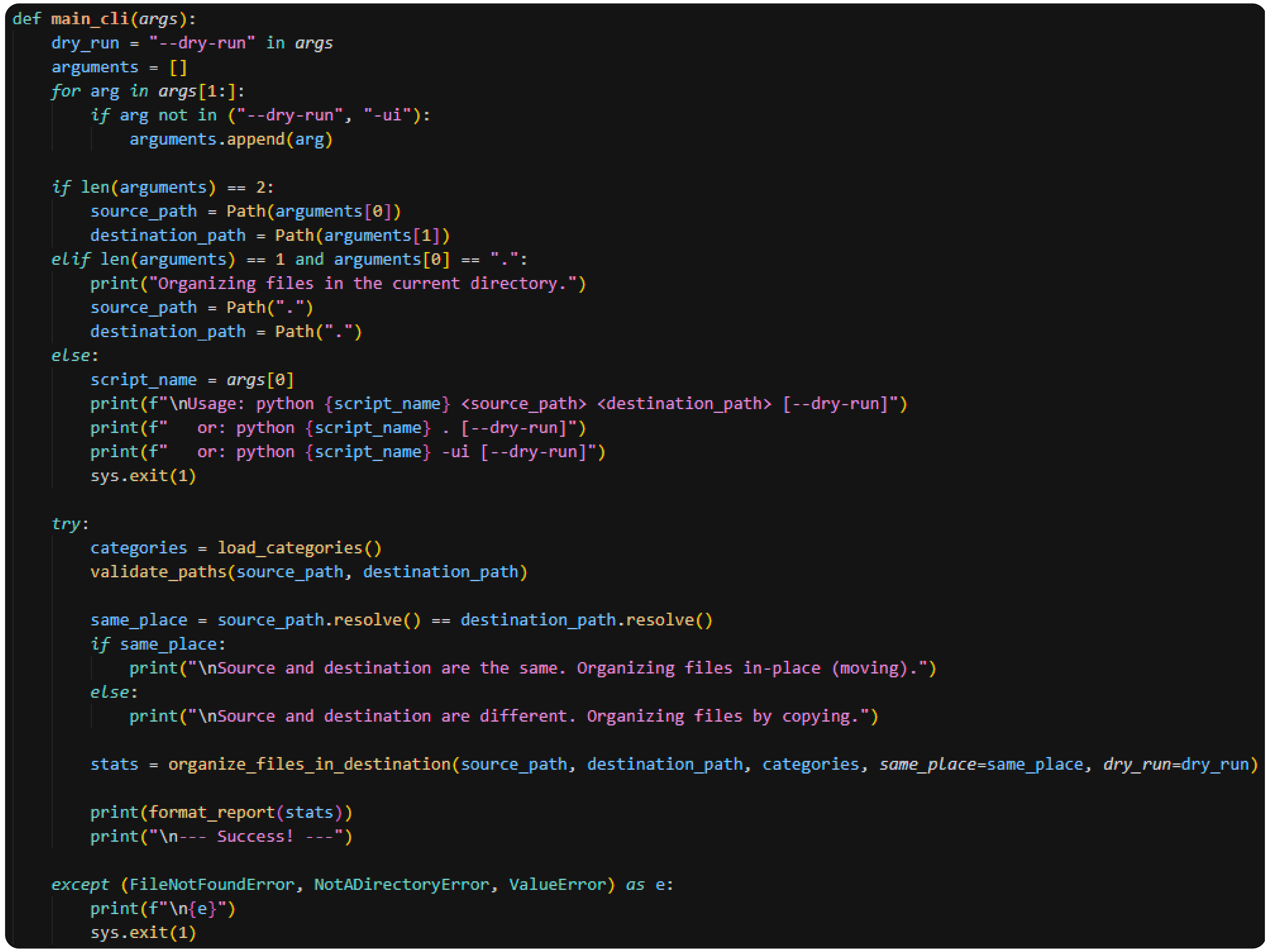
```
COE203_1_2509011061_P1
  > .pytest_cache
  v src
    > __pycache__
      __init__.py
      folder_utils.py
      gui.py
      main.py
      os_config.py
  v tests
    > __pycache__
      __init__.py
      conftest.py
      test_category_loading.py
      test_file_organization.py
      test_gui.py
      test_os_config.py
    .gitignore
    COE203 - Advanced Python - W03.pdf
    config.json
    create_dummy_files.py
    readme.md
```

entry point



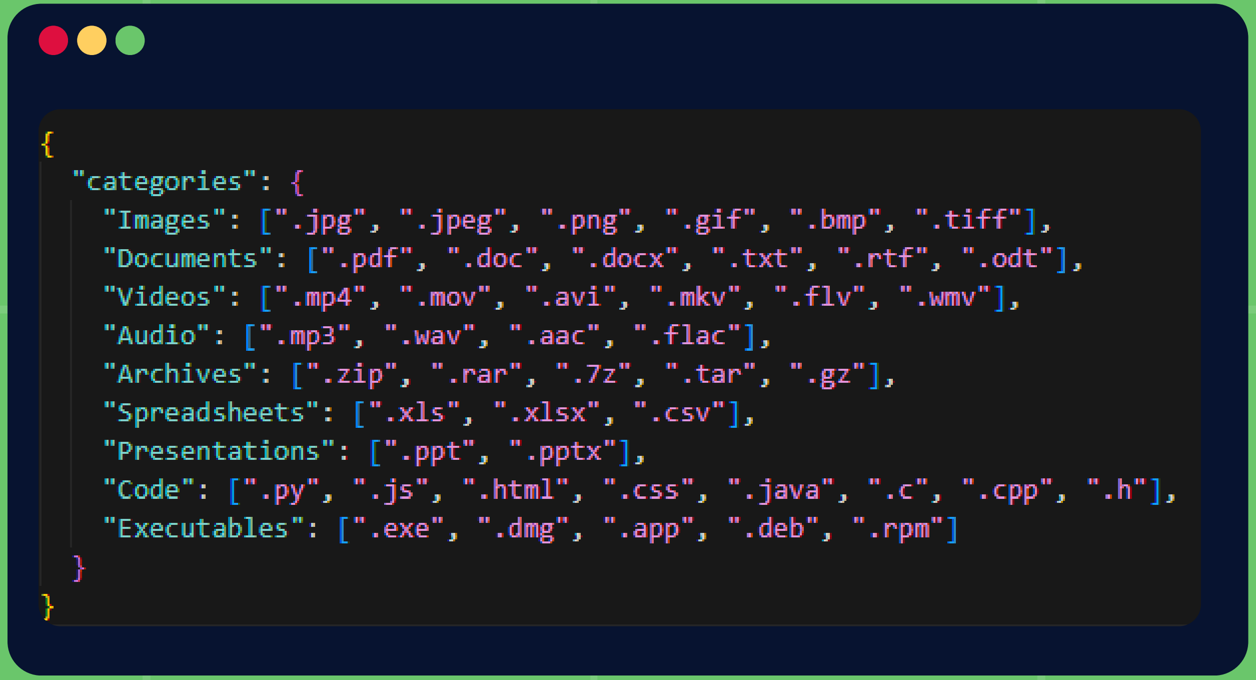
```python
def main():
    use_ui = "-ui" in sys.argv
    dry_run = "--dry-run" in sys.argv

    if use_ui:
        app = FileOrganizerApp(dry_run_cli_state=dry_run)
        app.mainloop()
    else:
        main_cli(sys.argv)

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

cli part

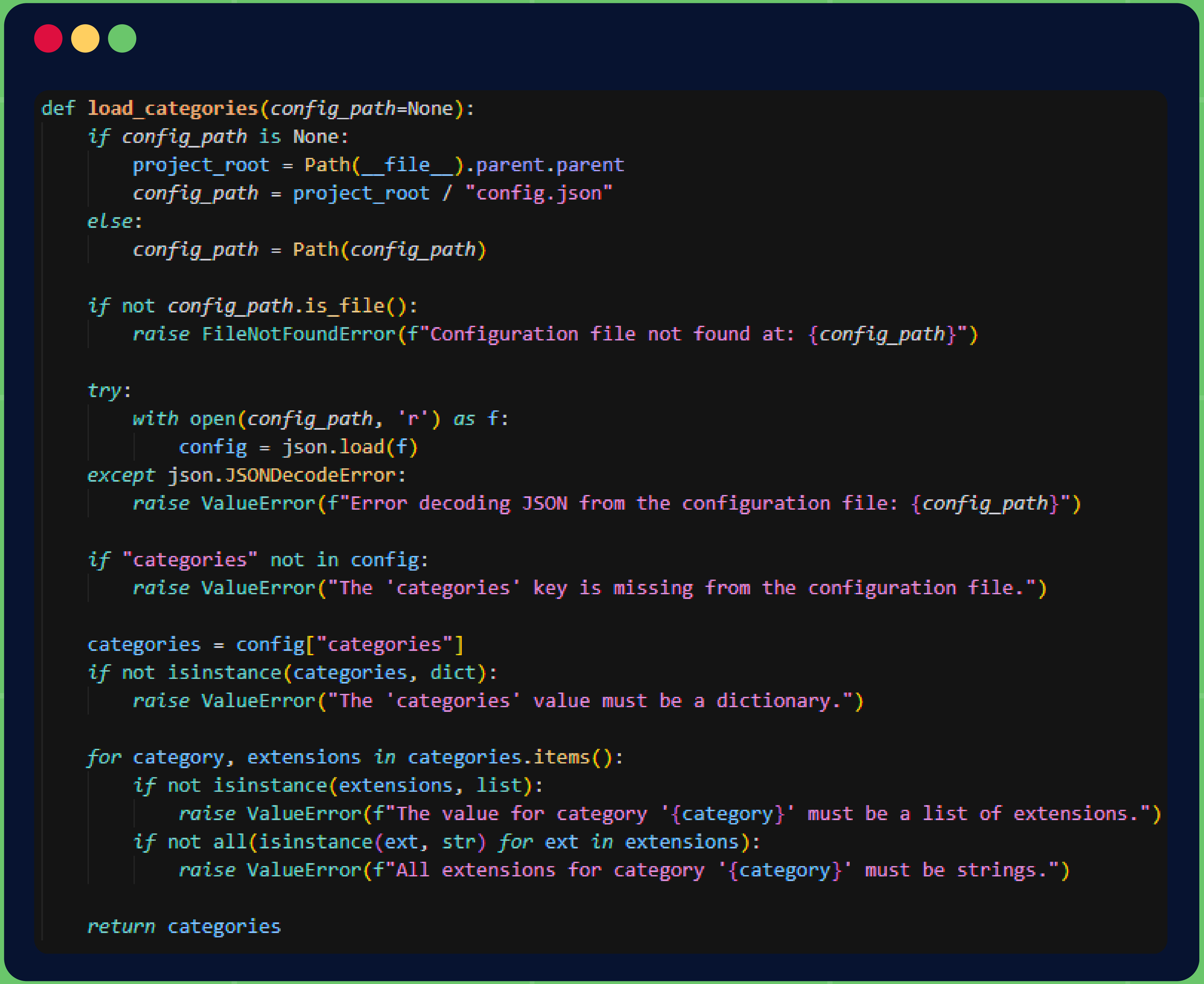


```python
def main_cli(args):
    dry_run = "--dry-run" in args
    arguments = []
    for arg in args[1:]:
        if arg not in ("--dry-run", "-ui"):
            arguments.append(arg)

    if len(arguments) == 2:
        source_path = Path(arguments[0])
        destination_path = Path(arguments[1])
    elif len(arguments) == 1 and arguments[0] == ".":
        print("Organizing files in the current directory.")
        source_path = Path(".")
        destination_path = Path(".")
    else:
        script_name = args[0]
        print(f"\nUsage: python {script_name} <source_path> <destination_path> [--dry-run]")
        print(f"   or: python {script_name} . [--dry-run]")
        print(f"   or: python {script_name} -ui [--dry-run]")
        sys.exit(1)

    try:
        categories = load_categories()
        validate_paths(source_path, destination_path)

        same_place = source_path.resolve() == destination_path.resolve()
        if same_place:
            print("\nSource and destination are the same. Organizing files in-place (moving).")
        else:
            print("\nSource and destination are different. Organizing files by copying.")

        stats = organize_files_in_destination(source_path, destination_path, categories, same_place=same_place, dry_run=dry_run)

        print(format_report(stats))
        print("\n--- Success! ---")

    except (FileNotFoundError, NotADirectoryError, ValueError) as e:
        print(f"\n{e}")
        sys.exit(1)
```

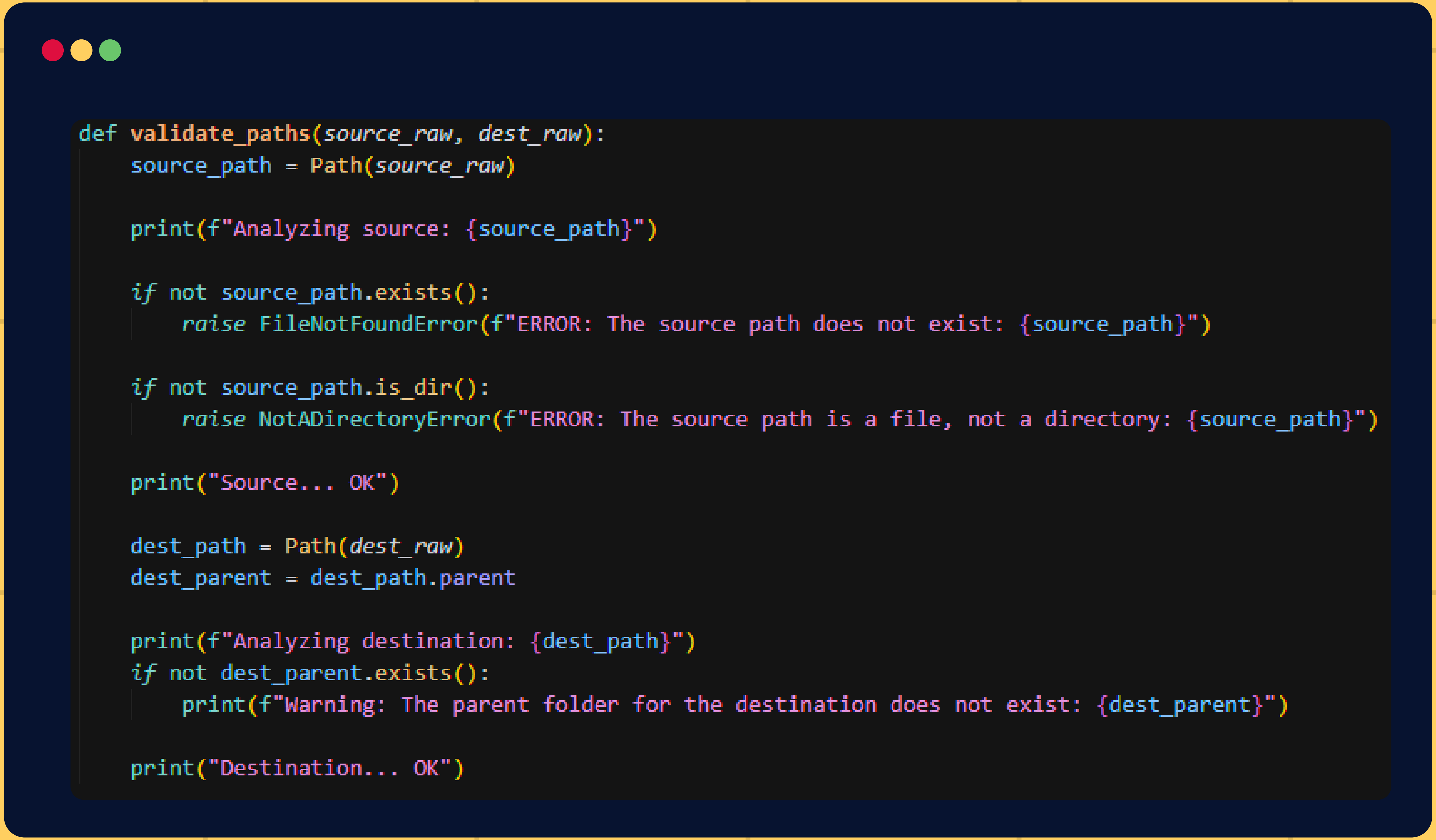
category



```json
{
  "categories": {
    "Images": [".jpg", ".jpeg", ".png", ".gif", ".bmp", ".tiff"],
    "Documents": [".pdf", ".doc", ".docx", ".txt", ".rtf", ".odt"],
    "Videos": [".mp4", ".mov", ".avi", ".mkv", ".flv", ".wmv"],
    "Audio": [".mp3", ".wav", ".aac", ".flac"],
    "Archives": [".zip", ".rar", ".7z", ".tar", ".gz"],
    "Spreadsheets": [".xls", ".xlsx", ".csv"],
    "Presentations": [".ppt", ".pptx"],
    "Code": [".py", ".js", ".html", ".css", ".java", ".c", ".cpp", ".h"],
    "Executables": [".exe", ".dmg", ".app", ".deb", ".rpm"]
  }
}
```



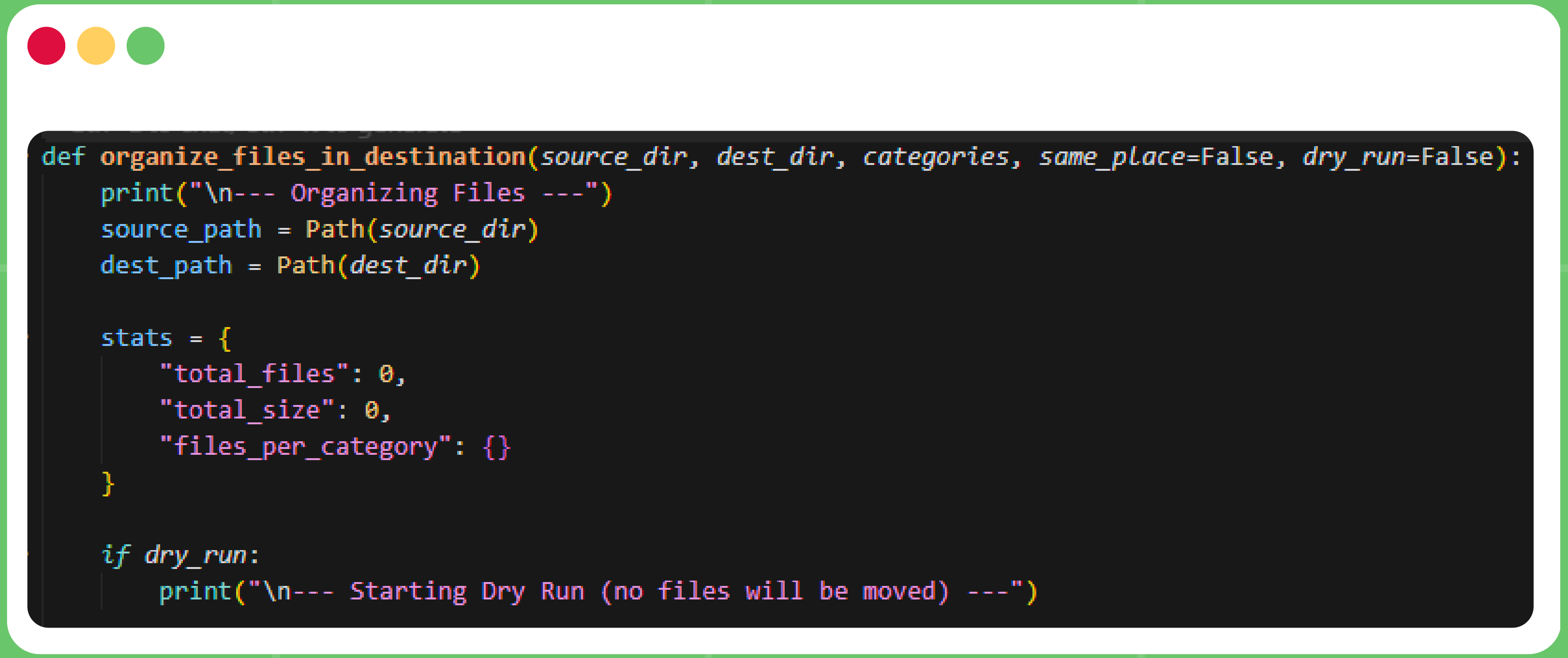
```python
def load_categories(config_path=None):
    if config_path is None:
        project_root = Path(__file__).parent.parent
        config_path = project_root / "config.json"
    else:
        config_path = Path(config_path)

    if not config_path.is_file():
        raise FileNotFoundError(f"Configuration file not found at: {config_path}")

    try:
        with open(config_path, 'r') as f:
            config = json.load(f)
    except json.JSONDecodeError:
        raise ValueError(f"Error decoding JSON from the configuration file: {config_path}")

    if "categories" not in config:
        raise ValueError("The 'categories' key is missing from the configuration file.")

    categories = config["categories"]
    if not isinstance(categories, dict):
        raise ValueError("The 'categories' value must be a dictionary.")

    for category, extensions in categories.items():
        if not isinstance(extensions, list):
            raise ValueError(f"The value for category '{category}' must be a list of extensions.")
        if not all(isinstance(ext, str) for ext in extensions):
            raise ValueError(f"All extensions for category '{category}' must be strings.")

    return categories
```



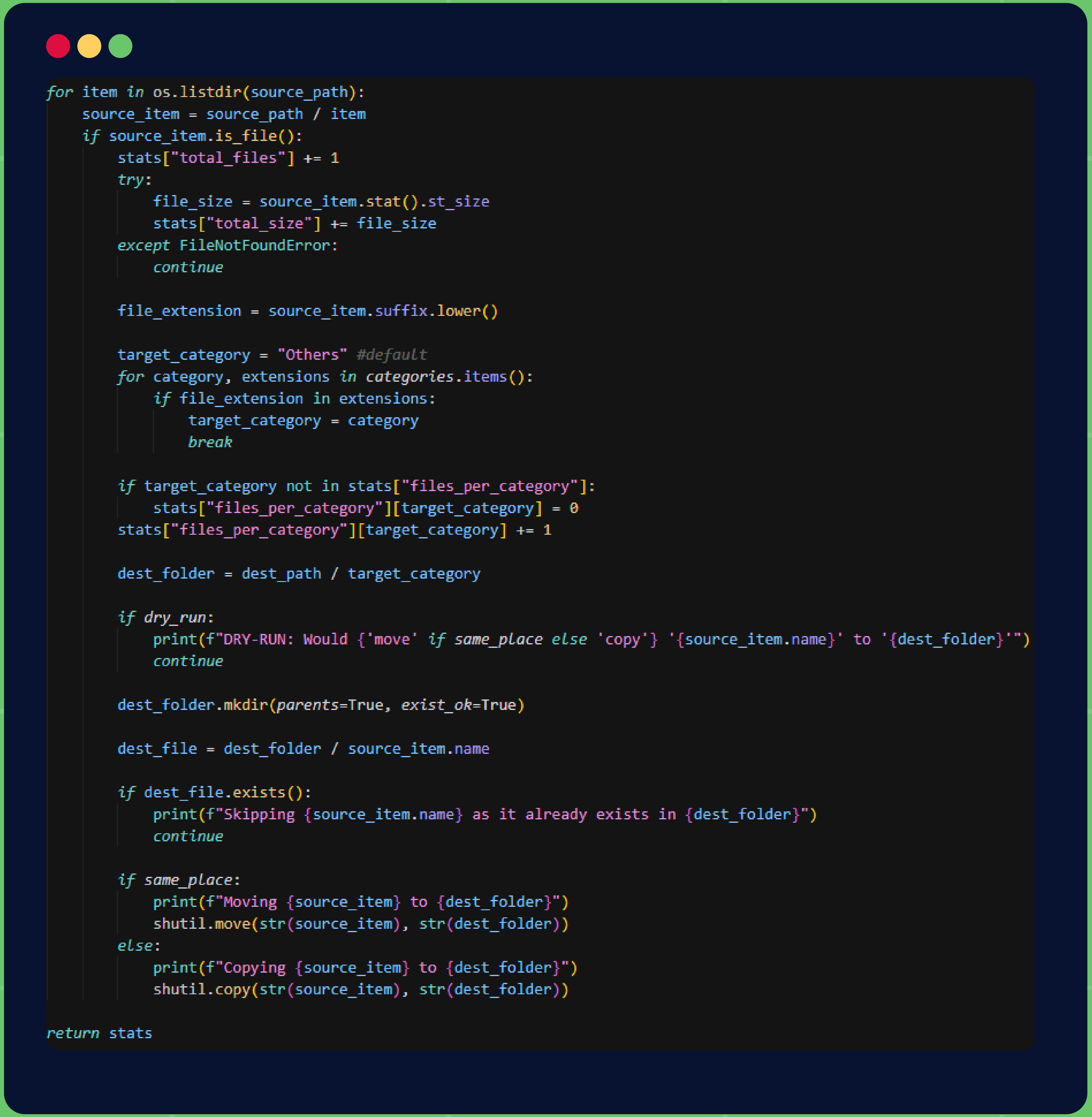
```python
def validate_paths(source_raw, dest_raw):
    source_path = Path(source_raw)

    print(f"Analyzing source: {source_path}")

    if not source_path.exists():
        raise FileNotFoundError(f"ERROR: The source path does not exist: {source_path}")

    if not source_path.is_dir():
        raise NotADirectoryError(f"ERROR: The source path is a file, not a directory: {source_path}")

    print("Source... OK")

    dest_path = Path(dest_raw)
    dest_parent = dest_path.parent

    print(f"Analyzing destination: {dest_path}")
    if not dest_parent.exists():
        print(f"Warning: The parent folder for the destination does not exist: {dest_parent}")

    print("Destination... OK")
```

path validation

organizing

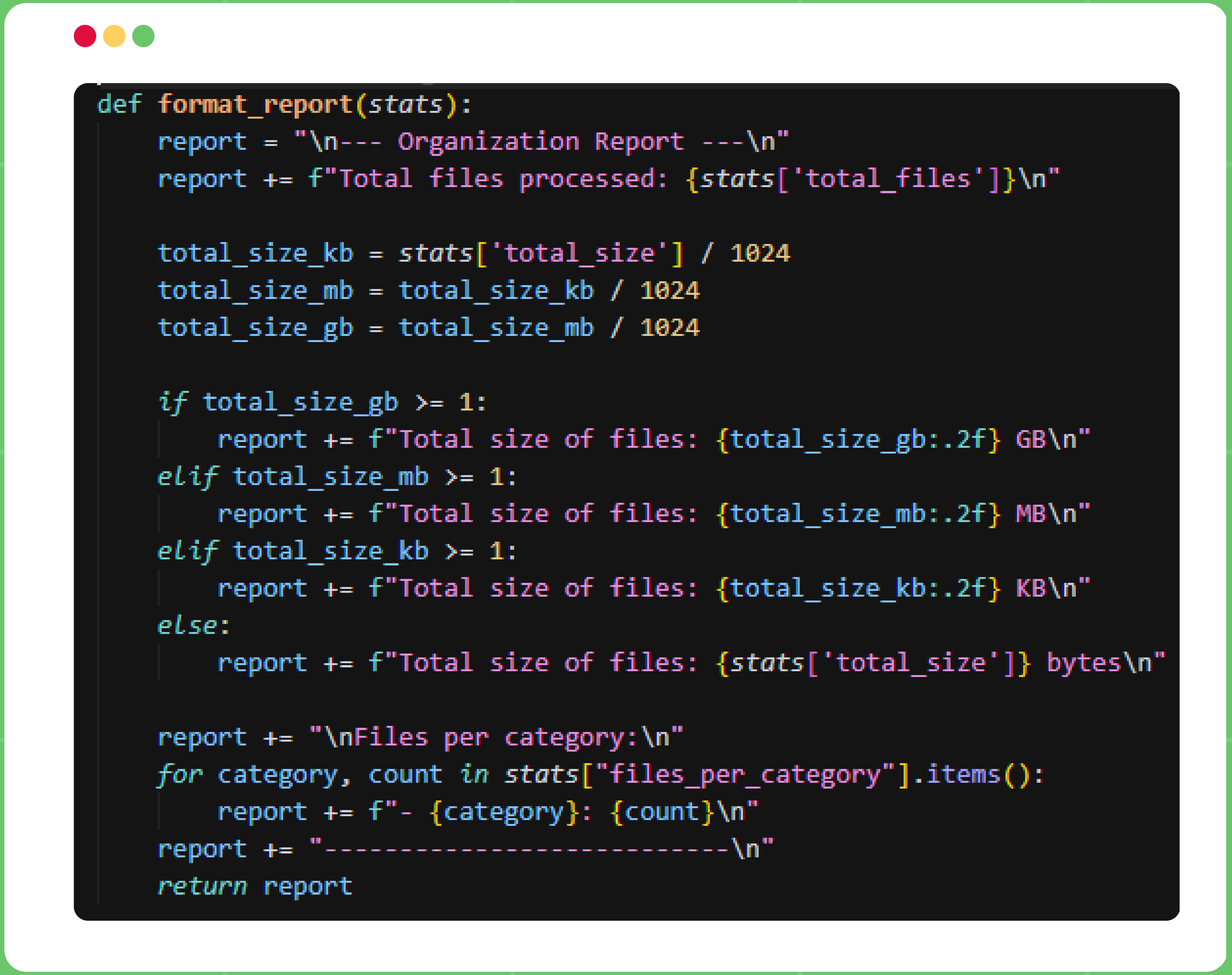


```python
def organize_files_in_destination(source_dir, dest_dir, categories, same_place=False, dry_run=False):
    print("\n--- Organizing Files ---")
    source_path = Path(source_dir)
    dest_path = Path(dest_dir)

    stats = {
        "total_files": 0,
        "total_size": 0,
        "files_per_category": {}
    }

    if dry_run:
        print("\n--- Starting Dry Run (no files will be moved) ---")
```



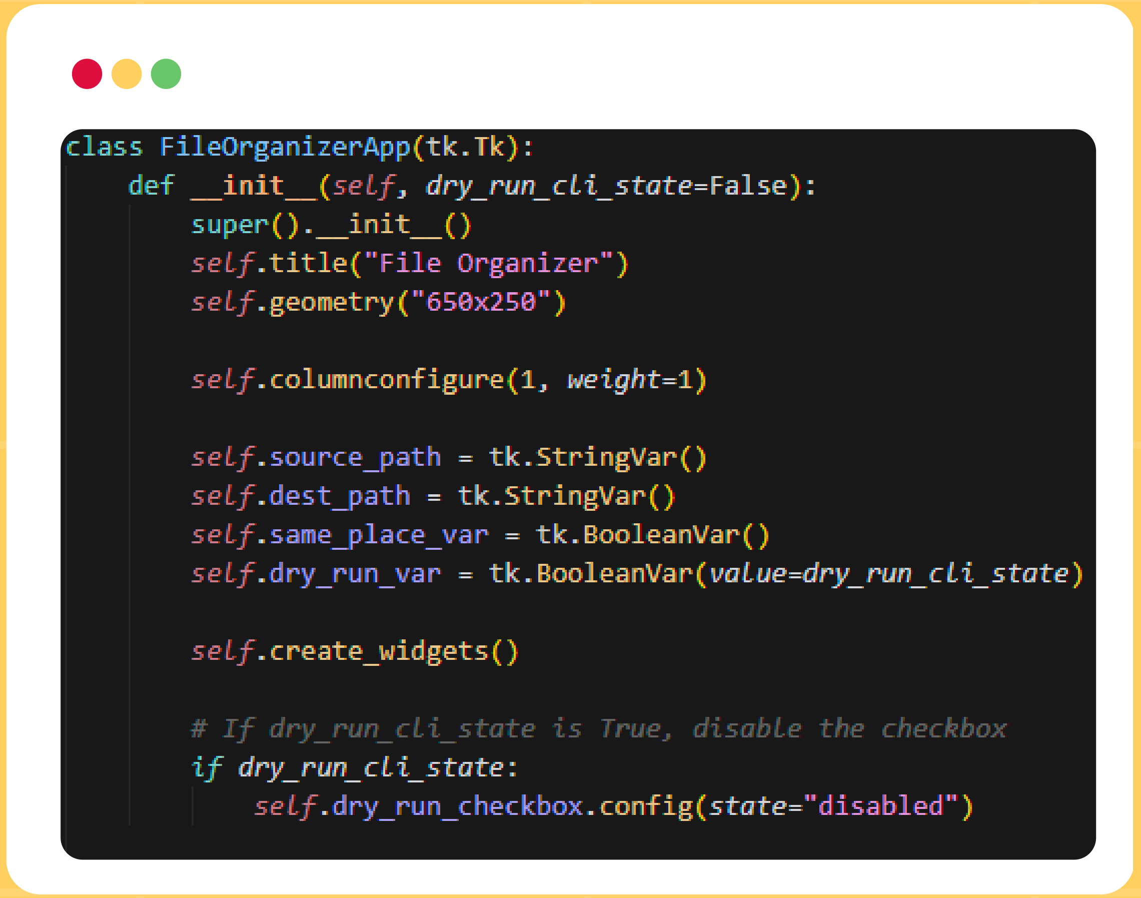
```python
for item in os.listdir(source_path):
    source_item = source_path / item
    if source_item.is_file():
        stats["total_files"] += 1
        try:
            file_size = source_item.stat().st_size
            stats["total_size"] += file_size
        except FileNotFoundError:
            continue

        file_extension = source_item.suffix.lower()

        target_category = "Others" #default
        for category, extensions in categories.items():
            if file_extension in extensions:
                target_category = category
                break

        if target_category not in stats["files_per_category"]:
            stats["files_per_category"][target_category] = 0
        stats["files_per_category"][target_category] += 1

        dest_folder = dest_path / target_category

        if dry_run:
            print(f"DRY-RUN: Would {'move' if same_place else 'copy'} '{source_item.name}' to '{dest_folder}'")
            continue

        dest_folder.mkdir(parents=True, exist_ok=True)

        dest_file = dest_folder / source_item.name

        if dest_file.exists():
            print(f"Skipping {source_item.name} as it already exists in {dest_folder}")
            continue

        if same_place:
            print(f"Moving {source_item} to {dest_folder}")
            shutil.move(str(source_item), str(dest_folder))
        else:
            print(f"Copying {source_item} to {dest_folder}")
            shutil.copy(str(source_item), str(dest_folder))

return stats
```

format report

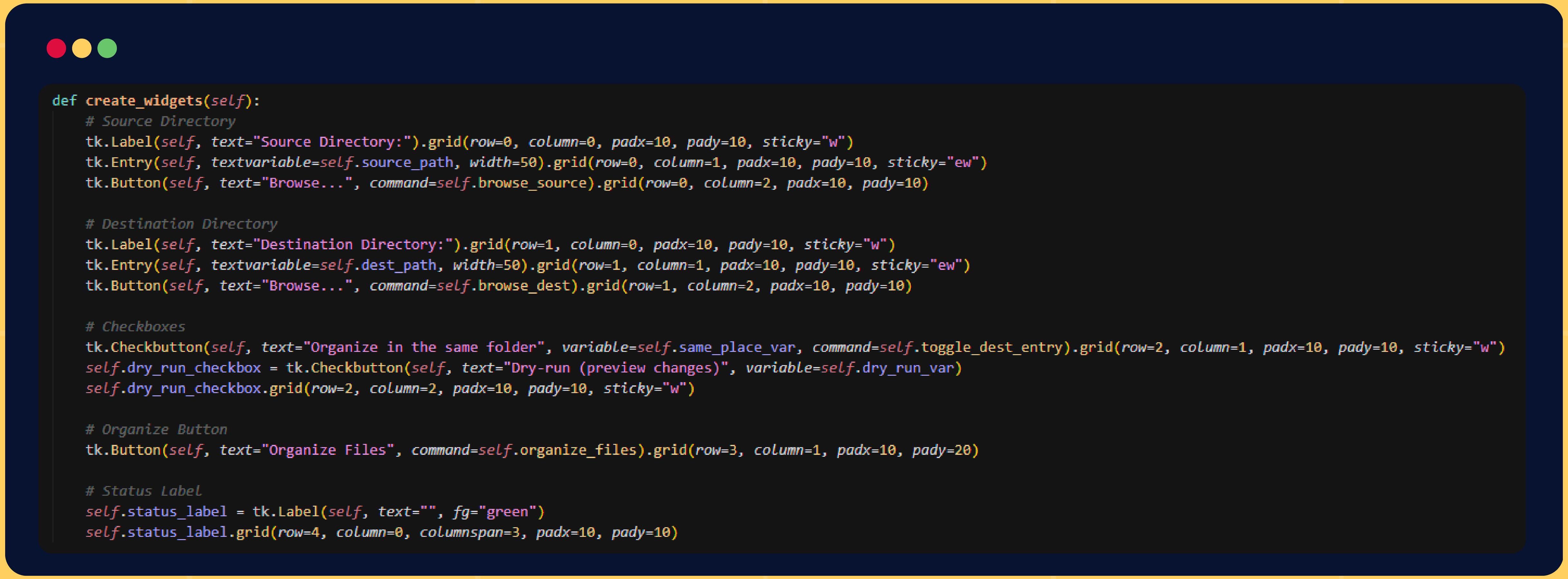


```python
def format_report(stats):
    report = "\n--- Organization Report ---\n"
    report += f"Total files processed: {stats['total_files']}\n"

    total_size_kb = stats['total_size'] / 1024
    total_size_mb = total_size_kb / 1024
    total_size_gb = total_size_mb / 1024

    if total_size_gb >= 1:
        report += f"Total size of files: {total_size_gb:.2f} GB\n"
    elif total_size_mb >= 1:
        report += f"Total size of files: {total_size_mb:.2f} MB\n"
    elif total_size_kb >= 1:
        report += f"Total size of files: {total_size_kb:.2f} KB\n"
    else:
        report += f"Total size of files: {stats['total_size']} bytes\n"

    report += "\nFiles per category:\n"
    for category, count in stats["files_per_category"].items():
        report += f"- {category}: {count}\n"
    report += "----------------------------\n"
    return report
```

ui part



```python
class FileOrganizerApp(tk.Tk):
    def __init__(self, dry_run_cli_state=False):
        super().__init__()
        self.title("File Organizer")
        self.geometry("650x250")

        self.columnconfigure(1, weight=1)

        self.source_path = tk.StringVar()
        self.dest_path = tk.StringVar()
        self.same_place_var = tk.BooleanVar()
        self.dry_run_var = tk.BooleanVar(value=dry_run_cli_state)

        self.create_widgets()

        # If dry_run_cli_state is True, disable the checkbox
        if dry_run_cli_state:
            self.dry_run_checkbox.config(state="disabled")
```



```python
def create_widgets(self):
    # Source Directory
    tk.Label(self, text="Source Directory:").grid(row=0, column=0, padx=10, pady=10, sticky="w")
    tk.Entry(self, textvariable=self.source_path, width=50).grid(row=0, column=1, padx=10, pady=10, sticky="ew")
    tk.Button(self, text="Browse...", command=self.browse_source).grid(row=0, column=2, padx=10, pady=10)

    # Destination Directory
    tk.Label(self, text="Destination Directory:").grid(row=1, column=0, padx=10, pady=10, sticky="w")
    tk.Entry(self, textvariable=self.dest_path, width=50).grid(row=1, column=1, padx=10, pady=10, sticky="ew")
    tk.Button(self, text="Browse...", command=self.browse_dest).grid(row=1, column=2, padx=10, pady=10)

    # Checkboxes
    tk.Checkbutton(self, text="Organize in the same folder", variable=self.same_place_var, command=self.toggle_dest_entry).grid(row=2, column=1, padx=10, pady=10, sticky="w")
    self.dry_run_checkbox = tk.Checkbutton(self, text="Dry-run (preview changes)", variable=self.dry_run_var)
    self.dry_run_checkbox.grid(row=2, column=2, padx=10, pady=10, sticky="w")

    # Organize Button
    tk.Button(self, text="Organize Files", command=self.organize_files).grid(row=3, column=1, padx=10, pady=20)

    # Status Label
    self.status_label = tk.Label(self, text="", fg="green")
    self.status_label.grid(row=4, column=0, columnspan=3, padx=10, pady=10)
```

thanks