Day 36: Pathlib Library - Renaming, Copying, Moving, Deleting & Creating Empty Files

Renaming Files

When you want to rename files, you can use `.with_stem()`, `.with_suffix()`, or `.with_name()`. They return the original path but with the filename, the file extension, or both replaced.

- If you want to change a file's extension, then you can use `.with_suffix()` in combination with `.replace()`:

  ```
  >>> from pathlib import Path
  >>> txt_path = Path("/home/rohit/realpython/hello.txt")
  >>> txt_path
  PosixPath("/home/rohit/realpython/hello.txt")

  >>> md_path = txt_path.with_suffix(".md")
  PosixPath('/home/rohit/realpython/hello.md')

  >>> txt_path.replace(md_path)
  # Using .with_suffix() returns a new path. To actually rename the file, you use .replace().
  # This moves txt_path to md_path and renames it when saving.
  ```

- If you want to change the complete filename, including the extension, then you can use `.with_name()`:

  ```
  >>> from pathlib import Path
  >>> txt_path = Path("/home/rohit/realpython/hello.txt")
  >>> txt_path
  PosixPath("/home/rohit/realpython/hello.txt")
  ```

```
>>> md_path = txt_path.with_name("goodbye.md")
PosixPath('/home/rohit/realpython/goodbye.md')

>>> txt_path.replace(md_path)
```

- If you want to rename the filename only, keeping the suffix as it is, then you can use `.with_stem()`.

Copying Files

Surprisingly, `Path` doesn't have a method to copy files. But with the knowledge that you've gained about `pathlib` so far, you can create the same functionality with a few lines of code:

```
>>> from pathlib import Path
>>> source = Path("shopping_list.md")
>>> destination = source.with_stem("shopping_list_02")
>>> destination.write_bytes(source.read_bytes())
```

- `.read_bytes():` read the content
- `.write_bytes():` write the content

Moving and Deleting Files

- Through `pathlib`, you also have access to basic file system–level operations like moving, updating, and even deleting files. For the most part, these methods don't give a warning or wait for confirmation before getting rid of information or files. So, be careful when using these methods.
- To move a file, you can use `.replace()`. Note that if the destination already exists, then `.replace()` will overwrite it. To avoid possibly overwriting the destination path, you can test whether the destination exists before replacing:

```
from pathlib import Path
```

```
source = Path("hello.py")
destination = Path("goodbye.py")

if not destination.exists():
    source.replace(destination)
```

Creating Empty Files

- To create an empty file with `pathlib`, you can use `.touch()`. This method is intended to update a file's modification time, but you can use its side effect to create a new file:

```
>>> from pathlib import Path
>>> filename = Path("hello.txt")
>>> filename.exists()
False

>>> filename.touch()
>>> filename.exists()
True

>>> filename.touch()
```

- If you don't want to modify files accidentally, then you can use the `exist_ok` parameter and set it to `False`:

```
>>> filename.touch(exist_ok=False)
Traceback (most recent call last):
  ...
FileExistsError: [Errno 17] File exists: 'hello.txt'
```

- Creating an empty file with `Path.touch()` can be useful when you want to reserve a filename for later use, but you don't have any content to write to it yet.

Python `pathlib` Examples

Counting Files

- With `pathlib`, you can conveniently use the `.iterdir()` method, which iterates over all the files in the given directory.

```
>>> from pathlib import Path
>>> from collections import Counter
>>> Counter(path.suffix for path in Path.cwd().iterdir())
Counter({'.md': 2, '.txt': 4, '.pdf': 2, '.py': 1})
```

- You can create more flexible file listings with the methods `.glob()` and `.rglob()`. For example, `Path.cwd().glob("*.txt")` returns all the files with a `.txt` suffix in the current directory. In the following, you only count file extensions starting with `p`:

```
>>> Counter(path.suffix for path in Path.cwd().glob("*.p*"))
Counter({'.pdf': 2, '.py': 1})

# If you want to recursively find all the files in both the directory and its su
bdirectories, then you can use .rglob().
```

Displaying a Directory Tree

- To traverse the subdirectories as well, you use the `.rglob()` method:

```
def tree(directory):
    print(f"+ {directory}")
    for path in sorted(directory.rglob("*")):
        depth = len(path.relative_to(directory).parts)
        spacer = "    " * depth
        print(f"{spacer}+ {path.name}")
```

- Note that you need to know how far away from the root directory a file is located. To do this, you first use `.relative_to()` to represent a path relative to the root directory. Then, you use the `.parts` property to count the number of directories in the representation. When run, this function creates a visual tree like the following:

```
>>> from pathlib import Path
>>> from display_dir_tree import tree
>>> tree(Path.cwd())
+ /home/rohit/realpython
    + directory_1
        + file_a.md
    + directory_2
        + file_a.md
        + file_b.pdf
        + file_c.py
    + file_1.txt
    + file_2.txt
```