




Topic	FILE SYSTEM EVENT HANDLERS	
Class Description	The student will be introduced to the watchdog module to manage file system creation events. The student will automate folder/files movement as soon as the file is downloaded, and segregate it into different folders based on the type of the file.	
Class	PRO C103	
Class time	45 mins	
Goal	<ul style="list-style-type: none"> • Introduction to the watchdog module. • Learn about file system events. • Learn about the exceptions in Python. • Automate downloaded files movement to different folders based on the type of file. 	
Resources Required	<ul style="list-style-type: none"> • Teacher Resources: <ul style="list-style-type: none"> ○ Laptop with internet connectivity ○ Earphones with mic ○ Notebook and pen ○ Visual Studio Code • Student Resources: <ul style="list-style-type: none"> ○ Laptop with internet connectivity ○ Earphones with mic ○ Notebook and pen ○ Visual Studio Code 	
Class structure	Warm-Up Teacher-Led Activity 1 Student-Led Activity 1 Wrap-Up	05 mins 15 mins 20 mins 05 mins
Credit	The class uses a module called “watchdog” written by Yesudeep Mangalapilly.	
WARM-UP SESSION - 05 mins		

<div>  </div> <p>Teacher Starts Slideshow</p> <p>Slide 1 to 3</p> <p>Refer to speaker notes and follow the instructions on each slide.</p>	
Teacher Action	Student Action
<p>Hey <student's name>. How are you? It's great to see you! Are you excited to learn something new today?</p> <p>Following are the WARM-UP session deliverables:</p> <ul style="list-style-type: none"> Greet the student. Revision of previous class activities. Quizzes. 	<p>ESR: Hi, thanks! Yes I am excited about it!</p> <p>Click on the slide show tab and present the slides</p>
<p>WARM-UP QUIZ</p> <p>Click on In-Class Quiz</p>	
<div>  </div> <p>Continue WARM-UP Session</p> <p>Slide 4 to 19</p>	
<p>Following are the session deliverables:</p> <ul style="list-style-type: none"> Appreciate the student. Narrate the story by using hand gestures and voice modulation methods to bring in more interest in students. 	
<div>  </div> <p>Teacher Ends Slideshow</p>	
<p>TEACHER-LED ACTIVITY - 10 mins</p>	
<p>Teacher Initiates Screen Share</p>	
<p><u>ACTIVITY</u></p> <ul style="list-style-type: none"> Introducing the watchdog module. 	

- Creating watchdog observer & file system event handler.
- Explain the try & except block.

Teacher Action	Student Action
<p>Well done! You answered both the questions well.</p> <p>In today's class, we will learn how to automate moving the files among folders as soon as they are downloaded.</p> <p>Are you excited to try out?</p> <p>Let's get started.</p> <p><i>The teacher creates a new folder 103. Opens the folder in VSC and create a new file named DownloadAndMove.py</i></p>	<p>ESR: Yes.</p>
<p>Let us start with importing all necessary modules:</p> <ol style="list-style-type: none"> 1. The time module provides the functions for working with time. 2. The os module in python provides functions for interacting with the operating system. It comes under Python's standard utility modules. 3. The shutil module offers a number of high-level operations on files and collections of files. <p><i>Note: These modules have been covered in previous classes.</i></p>	
<pre>import time import os import shutil</pre>	

Now to automate moving the files among folders as soon as they are downloaded, for which we will use **watchdog** module in Python.

What comes to your mind when we say watchdog?

ESR: Varied

In big companies, a watchdog is a person or group whose job is to keep an eye to make sure that large companies respect people's rights.

In Python, a **watchdog** module can be used to monitor our file system for changes like the **creation, modification, movement, or deletion** of a file or of a directory.

When a change occurs, the watchdog reports it to us by raising a specific event that we can handle.

We will be using the **FileSystemEventHandler** class & **Observer** class of the **watchdog** module.

Let's import these modules:

4. The **watchdog.observers.Observer** is the class that will look for any change in the given path, based on the changes, it calls the specific event handler.
5. The **watchdog.events.FileSystemEventHandler** is the event handler class that will manage the file system events(like creation, modification, movement, deletion of files).

*The teacher can refer to [Teacher Activity 1](#) to know more about the **watchdog** module.*

```
import time
import os
import shutil
from watchdog.observers import Observer
from watchdog.events import FileSystemEventHandler
```

Before we understand how these classes are used, can you tell me what are **events**?

Events represent certain action.

Can you tell me what could be a possible event related to the file system in our computer?

Superb!

To manage events/actions like creating files, moving files, renaming files, deleting files we need **Event Handlers**.

Event Handler manages certain events.

We will be using the **FileSystemEventHandler** class to manage file system events.

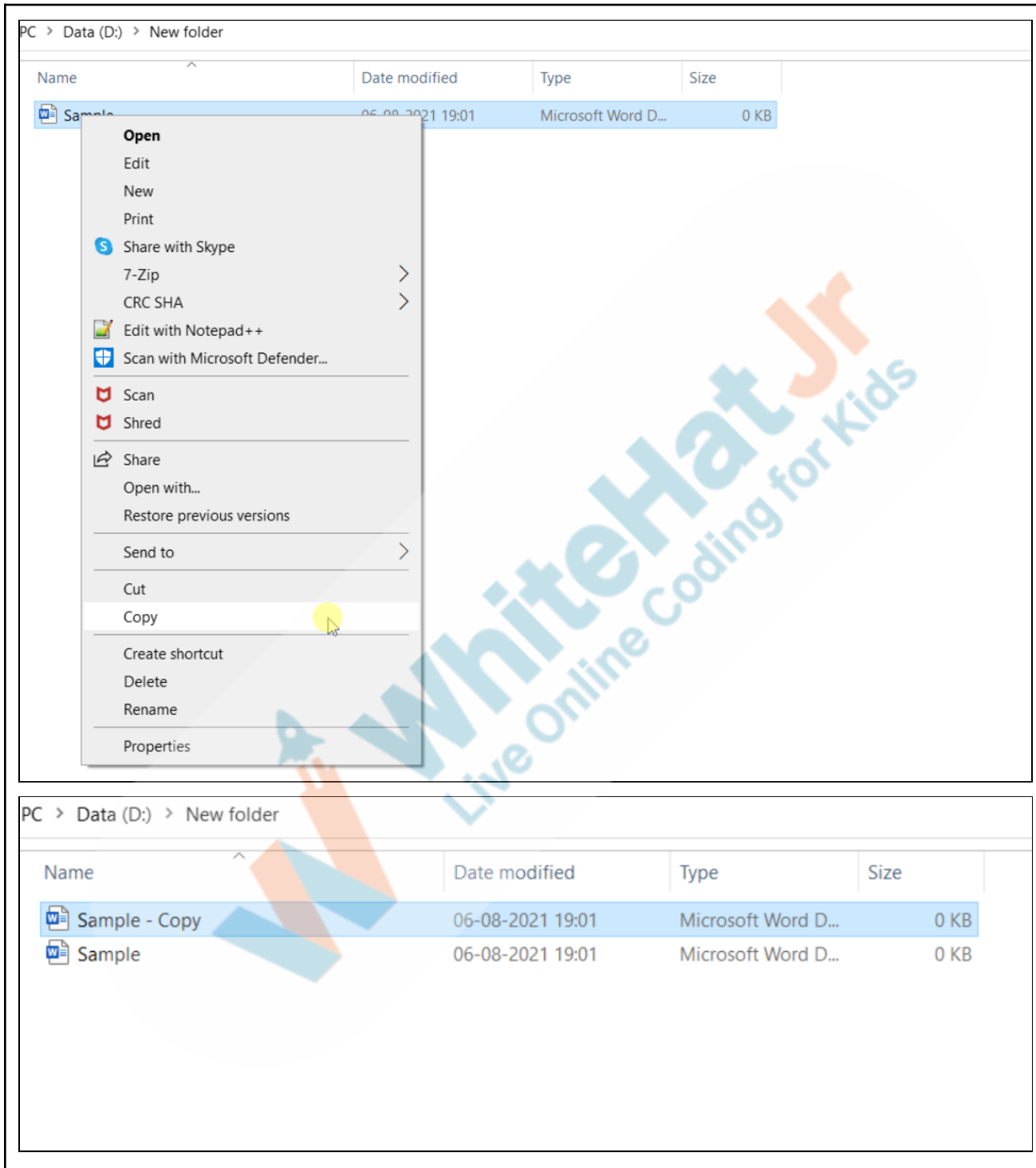
Open [Teacher Activity 2](#) to show some examples of files system events(New file creation, Copying a file & Renaming a file) happening manually on Windows.

For Mac, this may vary. Only for Mac users, guide the student to explore some file system events in Mac.

ESR: Varied

ESR: Creating files, moving files, renaming files, deleting files etc.

File System Copying Event Example (Windows)



The **FileSystemEventHandler** class:

This is a **predefined** class that will handle **file system events** like **creation, modification**, etc.

We will write a **user-defined** class, that is our own custom class to handle **FileSystemEventHandler** class events:

1. Create a class named **FileMovementHandler**.
2. Create an object variable, called **event_handler** of the **FileSystemEventHandler** class.

```
# Event Hanlder Class

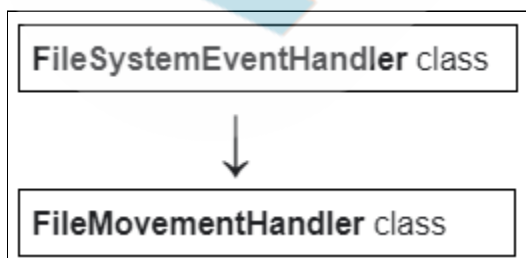
class FileMovementHandler():

    # CODE TO HANDLE NEW FILE CREATION EVENT IN A DIRECTORY

# Initialize Event Handler Class
event_handler = FileMovementHandler()
```

3. Pass **FileSystemEventHandler** as the parameter of the class.

This will help us use all attributes and methods from **FileSystemEventHandler** class inside **FileMovementHandler** class.



This concept, of using functionalities of one class into another, is known as **inheritance**.

***Note:** We will not be going into details of [inheritance in Python](#). Only use the simple explanation discussed above to help the student understand. Although teachers/students can explore the Python document to understand [inheritance](#).*

```
# Event Hanlder Class

class FileMovementHandler(FileSystemEventHandler):

    # CODE TO HANDLE NEW FILE CREATION EVENT IN A DIRECTORY

# Initialize Event Handler Class
event_handler = FileMovementHandler()
```

In the **FileSystemEventHandler** class, we have different methods to monitor different events:

- **on_any_event()**: Called for all event handlers.
- **on_created()**: Called when a file or a directory is created.
- **on_modified()**: Called when a file or directory is modified.
- **on_moved()**: Called when a file or a directory is moved or renamed.
- **on_deleted()**: Called when a file or directory is deleted.

We want to move the files into separate directories based on the type of the file whenever any file is downloaded from your browser.

Can you tell which event would be triggered when this is saved in the folder after downloading from the browser?

Great!

We will only need the **on_created()** method.

Let's define the **on_created()** method of **FileSystemEventHandler** inside our **FileMoveHandler** class.

4. Define the **on_create()** event method and print the **event** inside the method.

Now if we run the program and download some files from the browser, we do not see the output printed on the screen.

FileSystemEventHandler defines events, but we need the **Observer** class to tell the computer to start watching this event handler.

ESR: The new files are created, so **on_created()** event will be triggered.

```
class FileMovementHandler(FileSystemEventHandler):
    def on_created(self, event):
        print(event)
```

The **Observer** class:

This class will be used to monitor file changes using a **schedule()** method.

To implement these methods:

1. Create an object of the **Observer** class, called **observer**.
2. Schedule a path monitoring the **Observer** instance:

```
observer.schedule(event_handler, path, recursive=True)
```

- a. **event_handler**: An **event handler** object.
- b. **path**: A **directory path** to monitor.
Take a variable, **from_dir** that contains the path of the directory which we need to watch.

The **schedule()** method monitors the specified path **from_dir**, which triggers any event to call **event_handler**.

NOTE: Give the **from_dir** path as per your system.

- c. **recursive**: Set the recursive parameter value as **True** to observe the changes in all the sub directories of the given path.

By default, **recursive=False** which will not allow the **schedule()** method to monitor sub-directories.

Each call to **schedule()** method to monitor a path is called a **watch**.

3. Start the **Observer** using **start()** method.

```
from_dir = "C:/Users/preet/Downloads"

# Event Hanlder Class

class FileMovementHandler(FileSystemEventHandler):

    def on_created(self, event):

        print(event)

# Initialize Event Handler Class
event_handler = FileMovementHandler()

# Initialize Observer
observer = Observer()

# Schedule the Observer
observer.schedule(event_handler, from_dir, recursive=True)

# Start the Observer
observer.start()
```

We can also print “**running...**” to show the program is running.

We can use a **while** loop with “**True**” as a condition to keep printing until the program stops running.

Add a delay of 2 secs to print “running...” using **time.sleep()** method.

***Note:** Add **time.sleep()** method at other places in the code to add the delay in the execution, as sometimes one or the other process may take time.
For example, if the file might take some time to download completely, then we can tell the system to wait before it checks the existence of a file.*

```
while True:
    time.sleep(2)
    print("running...")
```

Run the code and test the output:

1. Run the .py file
2. Go to the browser.
3. Search for any image and download it in the directory used for watching.

We get the **event_type** and **src_path** value of the downloaded file from the **FileSystemEventHandler on_created()** event. We can see the type of **event** is **created**.

```
running...
<FileCreatedEvent: event_type=created, src_path='C:/Users/preet/Downloads\\0371ee62-e720-429d-a24f-a334de408d02.tmp', is_directory=False>
running...
running...
<FileCreatedEvent: event_type=created, src_path='C:/Users/preet/Downloads\\14d0b4df-3f99-4c6c-88fd-d6ef8e9c7d99.tmp', is_directory=False>
running...
running...
<FileCreatedEvent: event_type=created, src_path='C:/Users/preet/Downloads\\4ae479dd-fb09-40d8-a059-a03f1ee634d7.tmp', is_directory=False>
running...
running...
<FileCreatedEvent: event_type=created, src_path='C:/Users/preet/Downloads\\images.jfif', is_directory=False>
running...
running...
```

We learned that the **Observer** class can watch the file system events.

Now we need to tell the computer when to stop the **Observer** class scheduler.

Before we do that, let's understand the **Exceptions** in Python.

Exceptions:

An exception is an event that is triggered when an error occurs in a program.

The exception event is triggered by the programming language software to tell which error has occurred.

There are:

1. **Built-in Exception** events: These events are defined by some commonly known error that can occur while the program is running.
2. **User-defined Exception** events: These events are defined by the users to track errors differently from built-in.

Can you tell me some common errors you have seen in your program?

One of the most common we get is the “**ValueError**” exception. This is triggered when the argument value is incorrect while calling the function.

ESR: Varied.

An error can occur in any part of the program when the program is running. This disturbs the flow of the program and the program stops running all of a sudden.

Let's understand this with an example.

Open the [Teacher Activity 3](#) & explain the code to the student.

We have a list of numbers, we are dividing 5 by every element in the list using a **for** loop.

```
num_list = [5, 0, 2]

for elem in num_list:
    result = 5/elem
    print("The result of 5/",elem, "is", result)
```

Note: *Make a Copy of the Colab Notebook & Run the code in the 2nd cell.*

We can see that the program stopped running and gave an error "**ZeroDivisionError**". This is also one of the common errors.

Can you tell me why we get this error?

Amazing!

ESR: Because we have one element in the list which is 0, and division by any number is not defined.

```
num_list = [5, 0, 2]

for elem in num_list:
    result = 5/elem
    print("The result of 5/",elem, "is", result)
```

The result of 5/ 5 is 1.0

```
-----
ZeroDivisionError                                Traceback (most recent call last)
<ipython-input-9-a0141bd9e4ee> in <module>()
      2
      3 for elem in num_list:
----> 4     result = 5/elem
      5     print("The result of 5/",elem, "is", result)
      6
```

```
ZeroDivisionError: division by zero
```

Exception Handling:

To avoid crashing the program abruptly, the errors can be handled. This will help the program to ignore the errors and continue running.

We use **try-except** to find and handle the error in some code block by triggering exception events.

We can see that the **result = 5/elem** is giving us an error. We can avoid stopping the program by putting this block of code inside **try** and **except**

Python executes code inside the **try** statement as a "normal" part of the program.

The code inside the **except** statement is the program's response to any exceptions in the preceding try clause.

Run the code in the 4th cell.

We can see that the program keeps running smoothly. The program prints the error message and continues to execute.

```
num_list = [5, 0, 2]

for elem in num_list:
    try:
        result = 5/elem
        print("The result of 5/", elem, "is", result)
    except ZeroDivisionError:
        print("Oops! An error!")
```

```
The result of 5/ 5 is 1.0
Oops! An error!
The result of 5/ 2 is 2.5
```

Now that we understand what are exceptions and how we can find them to handle in the program,

You will write when to stop the observer.

Next, we will now read the extension and will move the files with the help of the **shutil** module in their respective folders.

So are you ready to code?

Share your screen and I will guide you.

ESR: Yes


Teacher Stops Screen Share

Teacher Starts Slideshow

Slide 20 to 23

Refer to speaker notes and follow the instructions on each slide.



<p>We have one more class challenge for you. Can you solve it?</p> <p>Let's try. I will guide you through it.</p>	
<p style="text-align: center;">Teacher Ends Slideshow</p> 	
STUDENT-LED ACTIVITY - 20 mins	
<ul style="list-style-type: none"> • Ask the student to press the ESC key to come back to the panel. • Guide the student to start Screen Share. • The teacher gets into Fullscreen. 	
<p style="text-align: center;">ACTIVITY</p> <ul style="list-style-type: none"> • Use the OS module to read the file names, extensions and create a path. • Use the Shutil module to move files from the Downloads folder to the new folder when it is downloaded based on their extension. 	
Teacher Action	Student Action
<p><i>Guide the student to download the code from Student Activity 1.</i></p> <p>First, I want you to change the path in “from_dir” and “to_dir”</p> <p><i>The teacher guides the student to update the section to add paths in the variables: from_dir and to_dir.</i></p> <p>Today, we will not limit our code to work on only image files, but we will segregate all types of files into their respective folders upon downloading.</p> <p>As you notice, we have been given a dictionary dir_tree as a dictionary with keys as the Directory/Folder names we will use to move the file based on its extension and values</p>	<p><i>The student downloads the folder from Student Activity 1, Unzip and saves it as C103. Open in VSC.</i></p>

as a **list** of extensions possible of one type of file.

The teacher can allow the student to go over the dictionary and add if he/she wants to add any more extensions.

```
from watchdog.observers import Observer
from watchdog.events import FileSystemEventHandler

from_dir = "ENTER THE PATH OF DOWNLOAD FOLDER (USE "/" " in VSC"
to_dir = "ENTER THE PATH OF DESTINATION FOLDER(USE "/" " in VSC"

dir_tree = {
    "Image_Files": ['.jpg', '.jpeg', '.png', '.gif', '.jif'],
    "Video_Files": ['.mpg', '.mp2', '.mpeg', '.mpe', '.mpv', '.mp4', '.m4p', '.m4v', '.avi', '.mov'],
    "Document_Files": ['.ppt', '.xls', '.csv', '.pdf', '.txt'],
    "Setup_Files": ['.exe', '.bin', '.cmd', '.msi', '.dmg']
}

class FileMovementHandler(FileSystemEventHandler):
```

We will segregate files based on the type of file into their respective folders upon downloading.

For this, we need to **define the segregations and movement steps**:

1. Extract the **name**, **extension** of the downloaded file using **os.path.splitext()** method.

Remember the **path** of the downloaded file is returned from the **event** object of the **on_created()** method?

We can get the path of the directory where the file is created using **event.src_path**.

ESR: Varied.

```
class FileMovementHandler(FileSystemEventHandler):

    def on_created(self, event):

        name, extension = os.path.splitext(event.src_path)
```

2. Loop through all the items in the dictionary. For this, we can use the `<dic_name>.item()` method that returns the **key** and **value** of the dictionary.

```
class FileMovementHandler(FileSystemEventHandler):

    def on_created(self, event):

        name, extension = os.path.splitext(event.src_path)

        for key, value in dir_tree.items():
```

3. Check if the **extension** is in **value**:

Till now the **name** value is the path value where the file was downloaded, but we need the actual name of the downloaded file(excluding the directory path) to form the correct path value strings for the source and destination path to be used in **shutil.move()** method.

Note: If you notice the result of the **.splitext()** method we received the name, extension, we got the entire path in the name:

name >> C:/Users/ADMIN/Downloads/image
extension >> .jfif

For this, we will use **os.path.basename()** on the downloaded file. This method in Python is used to get the basename in the specified path.

This method internally uses **os.path.split()** method to split the specified path into a pair (head, tail).

os.path.basename() method returns the tail part after splitting the specified path into (head, tail) pair.

***Note:** Using **.basename()** method we are extracting the actual file name as **filename.ext**(eg. image.jfif)*

```
class FileMovementHandler(FileSystemEventHandler):  
    def on_created(self, event):  
        name, extension = os.path.splitext(event.src_path)  
        for key, value in dir_tree.items():  
            if extension in value:  
                file_name = os.path.basename(event.src_path)
```

4. Create path variables:

Similar to what we did in the last class, create 3 variables for the name of the directory paths:

- Create **path1** as the name of the source path. Use string concatenation to merge **from_dir+'/' + file_name**

Example:

path1: Downloads/ImageName1.jpg

- Create **path2** as we want to create a new folder with that extension name and move the files to that folder.

<p>Use string concatenation to merge to_dir + '/' + key</p> <p>Example: path2: D:/My Files/Image_Files</p> <ul style="list-style-type: none"> • Create path3 to assign the destination path with the same file name as the source. Use string concatenation to merge to_dir + '/' + key + '/' + file_name <p>Example: path3: D:/MyFiles/Image_Files/ImageName1.jpg</p> <p>The key is taken from dir_tree dictionary.</p> <p>We should also sleep using the time.sleep(secs) method module before we move the files, as it might sometimes take a few changes to perform all the checks on the file.</p>	
<pre>path1 = from_dir + '/' + file_name path2 = to_dir + '/' + key path3 = to_dir + '/' + key + '/' + file_name time.sleep(3)</pre>	
<p>5. Move the files using shutil:</p> <p><i>Note: The code is the same as we wrote in the previous class.</i></p> <p><i>The teacher can make the student copy the code from the previous class file to paste here, OR the teacher can make the student write it.</i></p> <p>We are creating an if condition to check if the</p>	

folder/directory path exists before moving, else make a new folder/directory then move:

- Creating a condition to check if the destination path exists at **path2**.
- If true, **print()** a message moving with the file name which is being moved.
- Use **shutil.move(sourcepath, destinationpath)**
- In this case, **path1** is the source path and **path3** is the destination path.
- If the path is not created.
- Use **os.makedirs()** to create **path2**.
- **print()** a message moving with the file name.
- Use **shutil.move(sourcepath, destinationpath)**

```

path1 = from_dir + '/' + file_name
path2 = to_dir + '/' + key
path3 = to_dir + '/' + key + '/' + file_name

if os.path.exists(to_dir+'/'+key):

    if os.path.exists(path2):
        print("Moving " + file_name + ".....")

        # Move from path1 ---> path3
        shutil.move(path1, path3)

    else:
        os.makedirs(path2)
        print("Moving " + file_name + ".....")
        shutil.move(path1, path3)

# Initialize event handler
event_handler = FileMovementHandler()

```

Now let's try to run the file and test the output.

```

running...
running...
Downloaded elephant-3832516__340.jpg
running...
running...
Directory Exists...
Moving elephant-3832516__340.jpg....
running...

```

We are able to segregate the files as soon as it comes to the Download folder.

Now, can you press stop the program by **<Ctrl+C>** on Windows or **<Cmd +C>** on Mac.

ESR: Yes

```
running...
running...
Downloaded elephant-3832516__340.jpg
running...
running...
Directory Exists...
Moving elephant-3832516__340.jpg....
running...
running...
running...
Traceback (most recent call last):
  File "D:\PRO C103\DownloadAndMove.py", line 70, in <module>
    time.sleep(2)
KeyboardInterrupt
PS D:\PRO C103>
```

Can you tell me what is the exception that came when you stopped the program?

KeyboardInterrupt Is also one of the built-in exceptions in Python. Exception **KeyboardInterrupt** occurs when we press the interrupt key (normally Control-C or Delete).

Can you tell me how we can handle this exception?

Perfect!

We can use **try-except** in the code where the exception occurs.

We can write the **while** loop inside the **try** block.

We can specify the **name of the exception** after the **except** keyword and stop the **Observer** class from using the **.stop()** method.

Make sure to follow indentation properly to avoid getting indentation errors!

ESR: KeyboardInterrupt

ESR: We can try-except.


```
try:
    while True:
        time.sleep(2)
        print("running...")
except KeyboardInterrupt:
    print("stopped!")
    observer.stop()
```

Now we can test the output again. We will not get the **KeyboardInterrupt** exception.

Output Reference

```
running...
Downloaded Night-Sky-Planets-Jupiter-Mercury-i70790763.jpg
Directory Exists...
Moving Night-Sky-Planets-Jupiter-Mercury-i70790763.jpg...
running...
running...
running...
running...
running...
running...
running...
running...
stopped!
```



You can keep this code running in the background, whenever you are downloading any files, and keep the folder organized.

Isn't automation with Python very useful?

We will stop here for today. You can run the code on various folders which you want to organise.

*The teacher can stop the class, however, if the student is fast and time permits continue with **ADDITIONAL ACTIVITY**.*

ESR: Yes.

<i>It includes code for checking if the file name already exists, then rename the file before moving it.</i>	
Teacher Guides Student to Stop Screen Share	
WRAP UP SESSION - 5 mins	
<div> <div>Teacher Starts Slideshow</div> <div>Slide 24 to 29</div>  </div>	
Activity details Following are the WRAP-UP session deliverables: <ul style="list-style-type: none"> • Appreciate the student. • Revise the current class activities. • Discuss the quizzes. 	
WRAP-UP QUIZ Click on In-Class Quiz	
<div> <div>Continue WRAP-UP Session</div> <div>Slide 30 to 35</div>  </div>	
Activity Details: Following are the session deliverables: <ul style="list-style-type: none"> • Explain the facts and trivia • Next class challenge • Project for the day • Additional Activity (Optional) 	
<p style="text-align: center;"><u>FEEDBACK</u></p> <ul style="list-style-type: none"> • Appreciate the student for his/her efforts in the class. • Ask the student to make notes for the reflection journal along with the code they wrote in today's class. 	

Teacher Action	Student Action
<p>You get Hats off for your excellent work!</p> <p>In the next class</p>	<p><i>Make sure you have given at least 2 Hats Off during the class for:</i></p> <div> <div>Creatively Solved Activities +10</div> <div>Great Question +10</div> <div>Strong Concentration +10</div> </div>
<p>PROJECT OVERVIEW DISCUSSION</p> <p>Refer the document below in Activity Links Sections</p>	
Teacher Clicks	<div>✕ End Class</div>
<p>ADDITIONAL ACTIVITIES</p>	
<p>Additional Activity:</p> <p>When we download files, many times we do not pay attention to the file names. We may download many files with the same name. Moving those files can replace the earlier files.</p> <p>We can resolve them by checking the filenames and renaming them with random numbers.</p> <p>Let's modify the code to add the condition to check the file name of the existing files in the respective directories and rename them:</p> <ol style="list-style-type: none"> 1. Write a condition to check if path2 (to_dir+'/' +key) 	

exists.

2. If true, check if the **path3**
(`to_dir+'/' + key + '/' + file_name`) i.e. the file name in the destination folder exists.
 - a. If true, print the message ("**File Already Exists in "+ key + "...."**")
 - b. `print("Renaming File " + file_name + "....")`
 - c. Create a **new_file_name** using:
 - i. Use filename without extension, use `os.path.split(file_name)[0]` to get the original name. Use `[0]` to extract just the name without extension.
 - ii. Concatenate with a random integer from **(0,999)**
 - iii. Concatenate with extension using `os.path.split(file_name)[1]`
 - d. Create **path4**
(`to_dir+'/' + key + '/' + new_file_name`)
 - e. `print("Moving " + new_file_name + "....")`
 - f. Use **shutil.move(path1, path4)**
 - g. `time.sleep(1)` for the delay in the next movement.
3. Else if **path3** does not exists:
 - a. `print("Moving " + file_name + "....")`
 - b. Use **shutil.move(path1, path3)**

c. `time.sleep(1)`

Note: Add `time.sleep()` method at other places in the code to add the delay in the execution, as sometimes the one or the other process may take time.

For example, if the file might take some time to download completely then we can tell the system to wait before it checks the existence of a file.

```
path1 = from_dir + '/' + file_name
path2 = to_dir + '/' + key
path3 = to_dir + '/' + key + '/' + file_name

time.sleep(1)
if os.path.exists(path2):

    print("Directory Exists...")
    time.sleep(1)

    if os.path.exists(path3):

        print("File Already Exists in " + key + "....")
        print("Renaming File " + file_name + "....")

        new_file_name = os.path.splitext(file_name)[0] + str(random.randint(0, 999)) + os.path.splitext(file_name)[1]

        path4 = to_dir + '/' + key + '/' + new_file_name

        print("Moving " + new_file_name + "....")
        shutil.move(path1, path4)
        time.sleep(1)

    else:

        print("Moving " + file_name + "....")
        shutil.move(path1, path3)
        time.sleep(1)

else:

    print("Making Directory...")
    os.makedirs(path2)
    print("Moving " + file_name + "....")
    shutil.move(path1, path3)
    time.sleep(1)
```

Run the code and download files with the same name and check the name change in the destination folder.

Output:

```

running...
Downloaded images.jfif
running...
Directory Exists...
File Already Exists in Image_Files....
Renaming File images.jfif....
Moving images93.jfif....
running...
running...
running...

```

ACTIVITY LINKS

Activity Name	Description	Link
Teacher Activity 1	The watchdog Module	https://pythonhosted.org/watchdog/
Teacher Activity 2	File System Events Example(Windows)	https://s3-whjr-curriculum-uploads.whjr.online/3a2a20de-d5c5-4211-8c76-434f15df2c4e.gif
Teacher Activity 3	Exception Handling Example	https://colab.research.google.com/drive/1oMp7g9adhcpPw_Gb30Mdy4kYD_LEGiDc?usp=sharing
Teacher Activity 4	Reference Code	https://github.com/procodingclass/PRO-C103-Reference-Code
Teacher Activity 5	Output Reference	https://s3-whjr-curriculum-uploads.whjr.online/d89cebb3-9a4d-4312-9765-5307b8235d8b.gif
Teacher Activity 6	Reference Code With Additional Activity	https://github.com/procodingclass/PRO-C103-Reference-Code_WithAA
Student Activity 1	Boilerplate Code	https://github.com/procodingclass/PRO-C103-Student-Boilerplate
Teacher Reference 1	Extend Your Knowledge	https://docs.python.org/3/tutorial/

		classes.html#inheritance
Teacher Reference 2	Project Document	https://s3-whjr-curriculum-upload.s.whjr.online/a58b10d8-9348-4215-aebe-bb9587805352.pdf
Teacher Reference 3	Project Solution	https://github.com/procodingclass/PRO-C103-Project-Solution
Teacher Reference 4	Visual-Aid	https://s3-whjr-curriculum-upload.s.whjr.online/2c20629e-3e4e-41db-9ff1-cbf4e9b90901.html
Teacher Reference 5	In-Class Quiz	https://s3-whjr-curriculum-upload.s.whjr.online/68dde349-c52e-45b8-b82f-67ac931ebe35.pdf