LECTURE 05. EXCEPTION HANDLING AND DEBUGGING

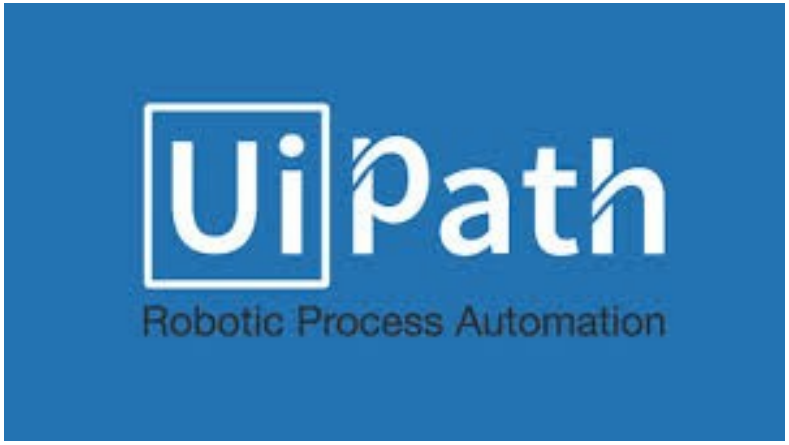
Robotic Process Automation
[31 October 2022]

Elective Course, 2022-2023, Fall Semester

Camelia Chisăliţă-Creţu, Lecturer PhD
Babeş-Bolyai University

Acknowledgements

This course is presented to our Faculty with the support of UiPath Romania.



Contents

Errors. Details

- **Error** is
  - an event that obstruct the regular execution of the program;
- based on their source, there are different types of errors:

Syntax errors

Where the compiler/interpreter cannot parse the written code into meaningful computer instructions

User errors

Where the software determines that the user's input is not acceptable for some reason

Programming errors (bugs)

Where the program contains no syntax errors but does not produce the expected results

Exceptions. Details

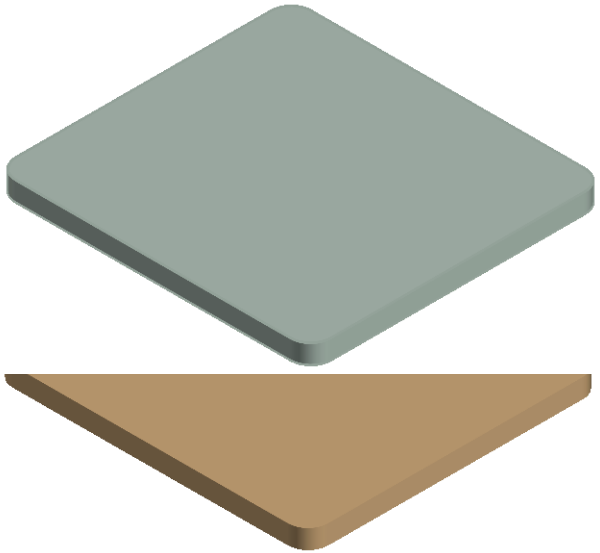
- **Exception** is
  - an event that interrupts the normal flow of instructions while executing a program;
  - a type of error that is recognized (caught) by the program, categorized and handled;
- it refers to the amount of deviation in the output shown from the required business or the agreed process.
- some general exceptions are:
  - time exceptions,
  - I/O exceptions,
  - user exceptions,
  - class exceptions.

Exception. Types

- In UiPath there are two types of exceptions:

System Exception

The disruption caused due to a system failure is called a system exception.



Business Exception

An exception that occurs in the business process is called a business exception.

Business Exception. Details

- **Business rule** is
  - a set of specific instructions on how the input and output of data should be processed in order to achieve a meaningful business result;
- **Business exception** is
  - any deviation from the standard business rule in a process or activity;
  - dependent on the type of business;
- E.g.: if the subject of an email is not standard and keeps on changing, the **Robot** will not be able to pick it; this results in a business exception;
- **solution:**
  - to avoid the business exception the business process needs to be understood very carefully.

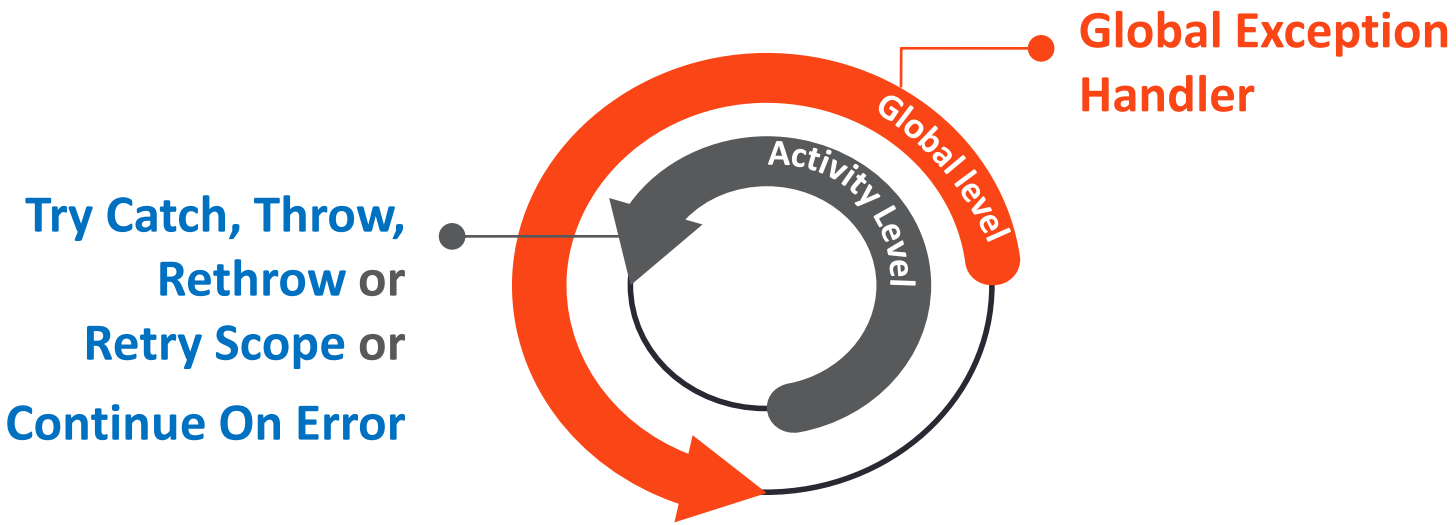
System Exception. Details

- **System exception (error)** occurs
  - when the normal flow of the automation process is stopped due to the failure of system;
- E.g.: if Outlook is not working, the **Robot** will not be able to open any attachment. This is categorized as *system error*;
  - the error caused is due to Microsoft Outlook failure and is not related to the programming of the business rule, hence this is categorized as *system error*;

- **solution:**
  - a *system error* can be minimized by making the code error proof.

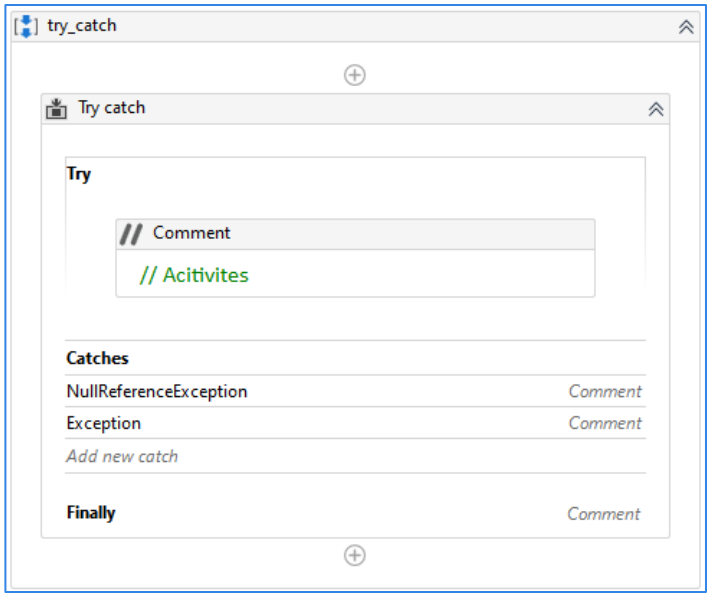
Error Handling Approaches

- it is common for automation projects to encounter events that interrupt or interfere with the projected execution.
- Actions executed when exceptions are caught:
  - **Stopping the execution;**
  - **Executing automatically explicit actions within the workflow;**
  - **Escalating/sending the issue to a human operator.**



Try Catch Activity. Details

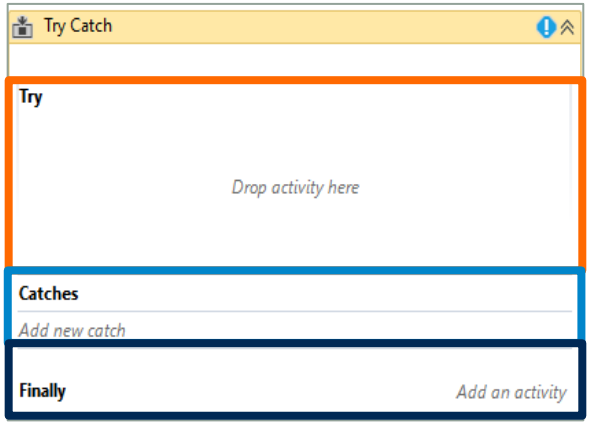
- in every automation process, user want to be able to detect the errors as they happen and perform various actions to rectify the errors when they occur;
- **Try Catch** activity
  - is used to enable the program to recover from specific error instead of crashing and terminating the whole workflow;
- represents a reliable method available in UiPath to identify the reason of failure.



see **Demo1 – TryCatchFinally**

Try Catch Activity. Components

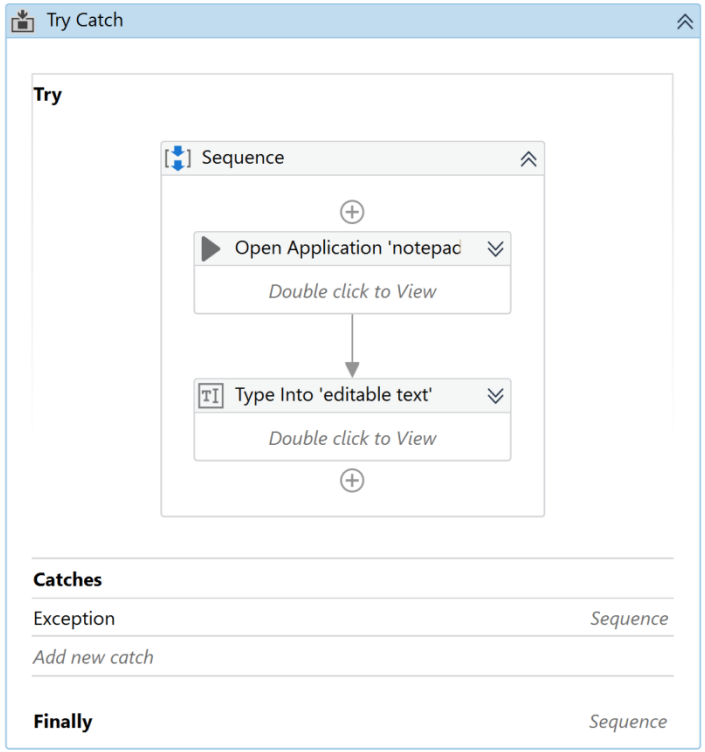
- for handling errors in **Try Catch** block, the workflow is divided into 3 parts:
- **Try Block**
  - all the possible activities that can create or cause error should be placed in this block;
- **Catches Block**
  - the user can add multiple type of catches in this block;
  - the clause name is “Catches” in UiPath;
- **Finally Block**
  - it is used for the actions to be performed after the **Try-Catch** blocks;
  - it is executed in case of an error or no error is caught in the **Catches** block.



see **Demo1 – TryCatchFinally**

Demo1. Try Catch Activity

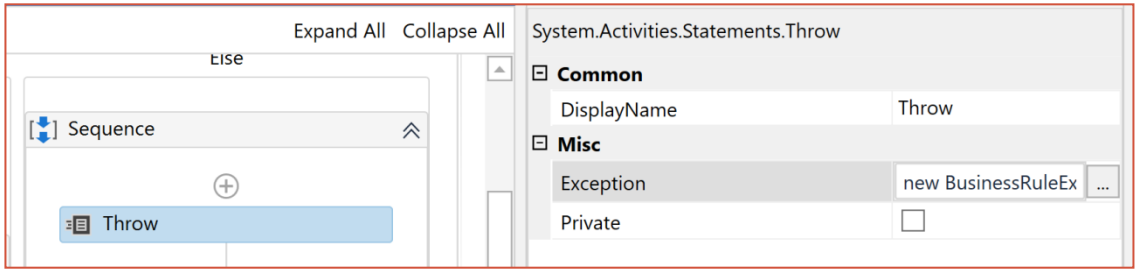
- Design a workflow that attempts to execute the following steps:
  - 1. *opens* the file ***File1.txt***
  - 2. *fills in* the text “Hello dear friend!”
  - 3. *fills in* the text “Have a nice day!” *---> issue*
- **Goals**: emphasize the use of **Try Catch** activity by:
  - Including the desired steps into the **Try** component;
  - Handling any exception occurred by presenting using the **MessageBox** activity the exception message inside the **Catch** component;
  - Closing the workflow with particular action for the **Finally** component.



see **Demo1 – TryCatchFinally**

Throw Activity. Details

- **Throw** activity is
  - used to throw a specific type of exception that the user intends to expose;
- the **Catches** block handles that particular type of exception and performs a specific action intended by the user for that special circumstances.



see **Demo2 – ThrowRethrow**

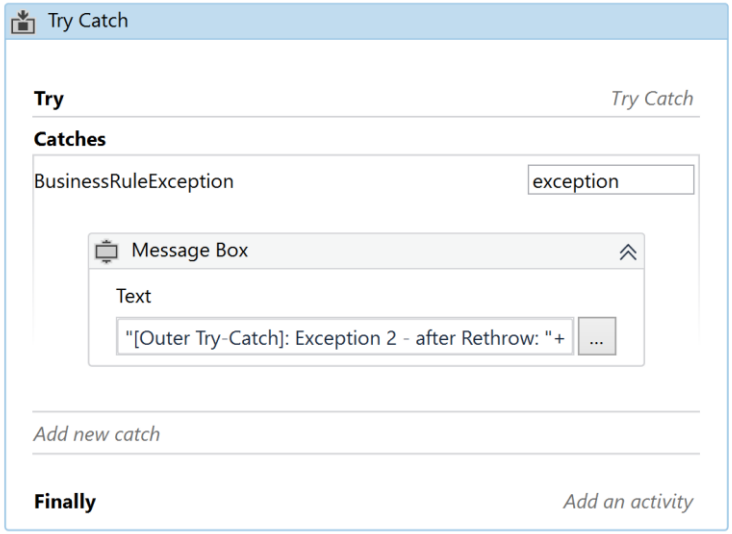
Rethrow Activity. Details

- **Rethrow** activity is
  - used to clear an exception;
- it is similar to the **Throw** activity and can be used in **Catches** block.

see **Demo2 – ThrowRethrow**

Demo2. Throw, Rethrow Activities

- Design a workflow that attempts to execute the following steps:
  - 1. *reads* an integer number (nr1)
  - 2. *reads* an integer number (nr2)
  - 3. *checks* if n1 divides nr2, i.e., **nr1 % nr2 == 0** (inner **Try Catch** activity)
    - if yes ==> a message is shown
    - if no ==> an exception is thrown, i.e., an object of type **BusinessRuleException** is created and thrown;
    - the inner **Catch** handles the exception by **Rethrow**-ing it again.
    - the outer **Try Catch** activity catches the exception within the **Catch** component and handles it.



see **Demo2 – ThrowRethrow**

UiPath

Retry Scope Activity. Details

- **Retry Scope** activity
  - retries the contained activities as long as **the condition is not met**, or **an error is thrown;**

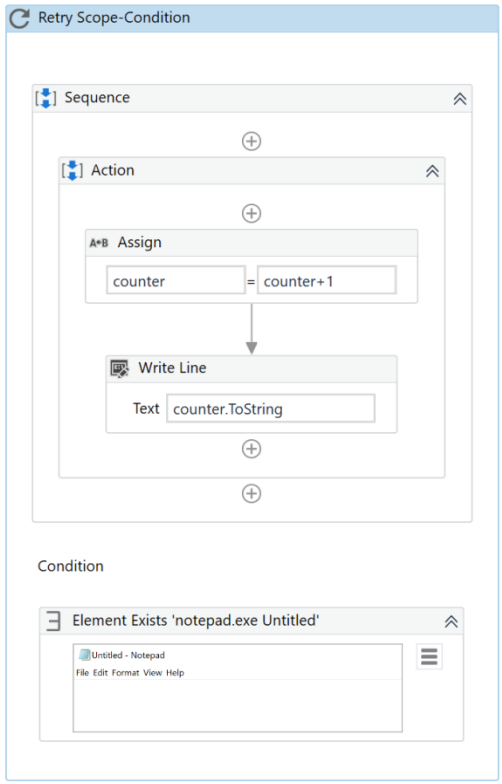
- it is used to retry the execution in situations in which an error is expected. The execution will be retried ***until a certain event happens*** (for a number of times) or ***without any condition*** (for a number of times).

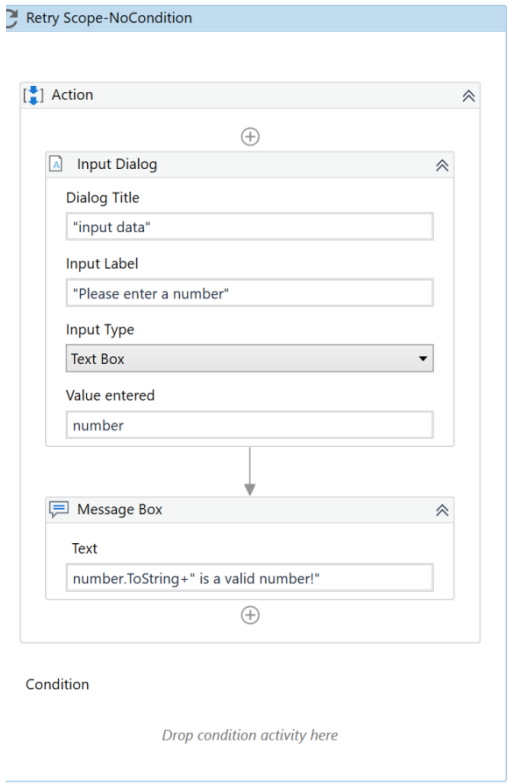
- it is used for catching and handling an error, being similar to **Try Catch** with the difference that **Retry Scope** activity simply retries the execution without providing a more complex handling mechanism.

see **Demo3 – RetryScope**

Demo3. Retry Scope Activity

- Design a workflow that attempts to execute the following steps:
  - 1. *increases* a counter;
    - the increase **is conditioned** by the existence of the **Notepad** app on the screen;
    - there are 3 attempts with 2 seconds retry interval;
    - it is combined with **ContinueOnError** property when the **Notepad** app is not available;
  - 2. *reads* an integer number
    - there are 3 attempts with 4 seconds retry interval;
    - **no condition is provided;**
    - it is combined with the **ContinueOnError** property when no valid number is read after all attempts.





see **Demo3 – Retry Scope**

Global Exception Handler. Details

- **Global Exception Handler** is
  - a predefined workflow used to handle all unhandled exception;
- it has a predefined structure and behaviour that can be adapted:
  - **predefined arguments;**
  - **predefined actions.**

see **Demo4 – GEH**

Global Exception Handler. Structure

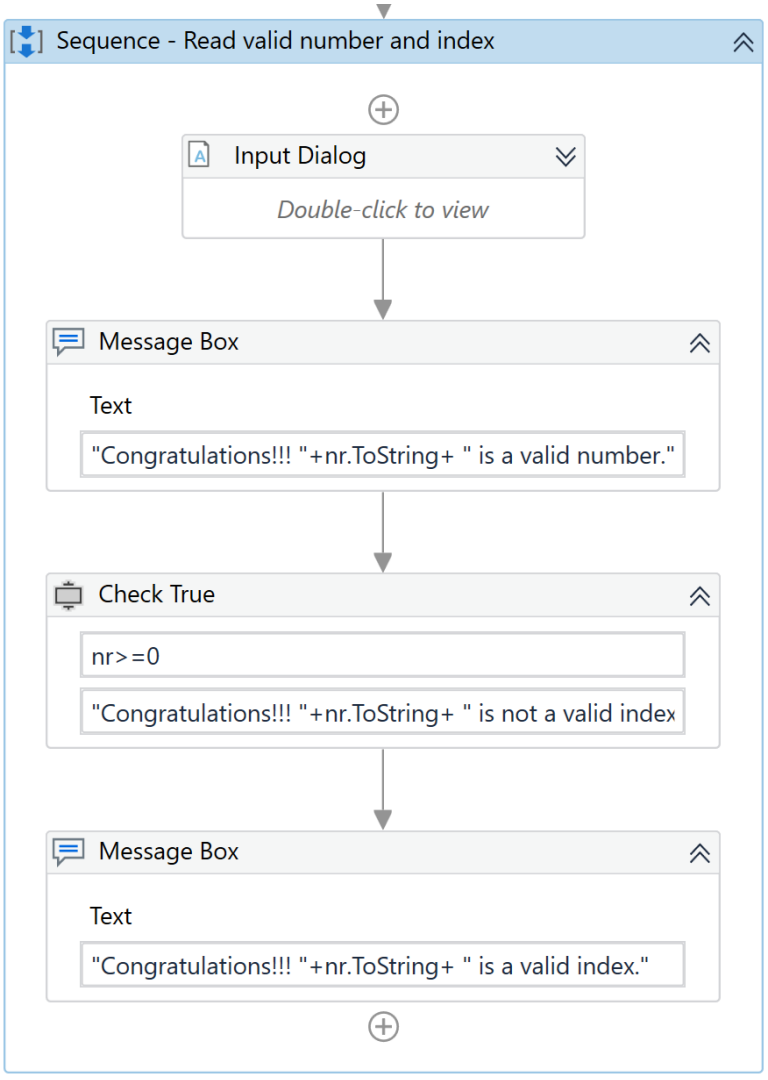
- there are two **predefined arguments** (should be kept):
  - `errorInfo` (***In*** direction) - contains the error that was thrown and the workflow that failed;
  - `result` (***Out*** direction) - will store the next behavior of the process when it encounters the error;

- there are two **predefined actions** (can be removed, others can be added):
  - **Log Error** – used to log the error (Fatal, Error, Warning, Info, etc.);
  - **Choose Next Behavior** – the action chosen to be taken when an error is encountered during execution:
    - *Continue* - the exception is re-thrown;
    - *Ignore* - the exception is ignored, and the execution continues from the next activity;
    - *Retry* - the activity which threw the exception is retried;
    - *Abort* - the execution stops after running the current handler.

see **Demo4 – GEH**

Demo4. GEH

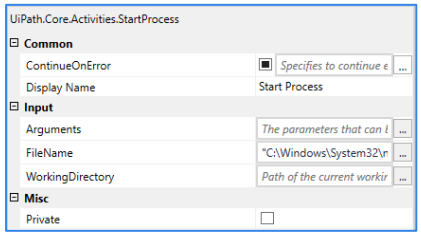
- Design a workflow that attempts to execute the following steps:
  - 1. *fills in* some text in a file that should be opened;
    - there are 4 attempts to achieve the above task, otherwise an exception is thrown and the workflow ends;
    - after each attempt the task is retried.
  - 2. *reads* an integer number (**nr**)
    - there are 4 attempts to read a valid number using the **Input Dialog** activity; after the 4th attempt an exception is thrown and the workflow ends.
  - 3. *checks* if **nr >= 0** using the **Check True** activity;
    - if the condition is **False** an exception is thrown; this check is performed 4 times before the workflow ends and the corresponding message is shown indicating whether **nr** is a valid index (>=0) or not (<0).



see **Demo4 – GEH**

Continue On Error Activity Field (Property). Details

- **Continue On Error** allows
  - to specify if the automation should continue, even if the activity throws an error.
- useful for activities that work with UI interaction;



- the field supports only Boolean values (**True**, **False**-default);
- if this field is blank and an error is thrown, the execution of the project stops;
- if the value is True, the execution of the project continues regardless of any error.

- if it is set to True on an activity that has a scope (such as **Attach Window** or **Attach Browser**), then all the errors that occur in the activities ***inside*** that scope are also ignored.

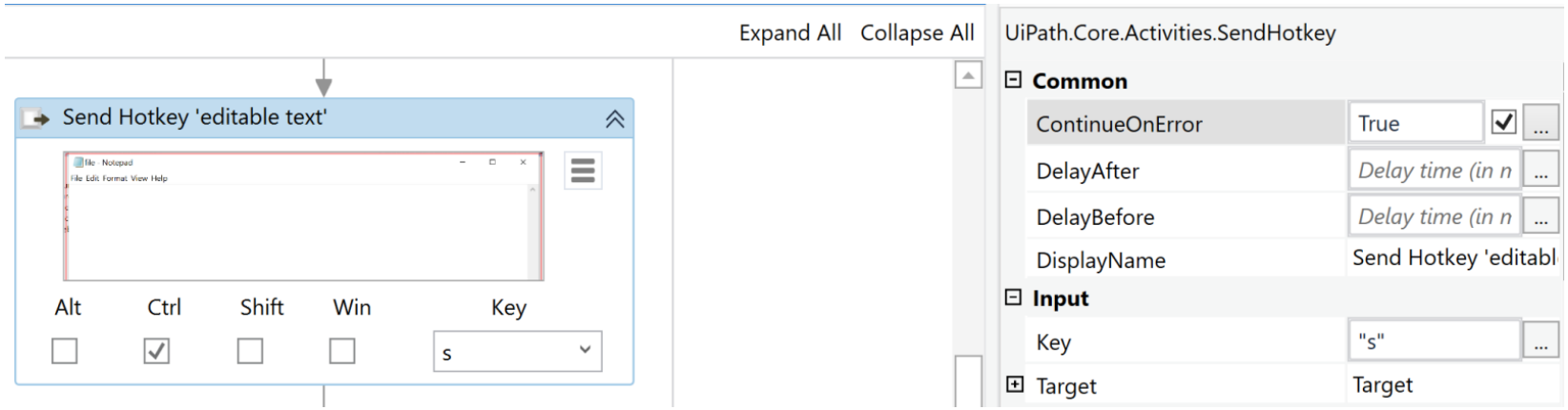
- recommended :
  - for data scraping – to avoid throwing an exception on the last page (when the selector of the 'Next' button is not found);
  - when the user is not interested in capturing the error, just to execute the activity.

see **Demo5 – ContinueOnError**

UiPath

Demo5. Continue on Error Property

- Design a workflow that attempts to execute the following steps:
  - 1. *fills in* some text in a file that should be opened;
    - if the property **ContinueOnError** is set to **False**, when an exception is thrown the execution of the workflow ends;
    - if the property **ContinueOnError** is set to **True**, if an exception is thrown this is ignored and the execution of the workflow continues.
  - 2. *saves* the files;
  - 3. *closes* the file.



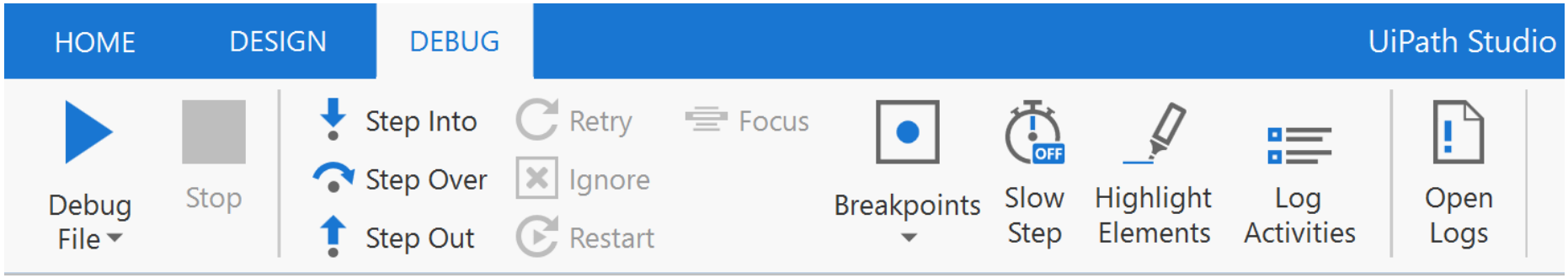
see **Demo5 – ContinueOnError**

Debugging. Details

- **Debugging**
  - is the process of identifying error or bug in the running software or Robot;
  - ensures that the error is identified and resolved so that it does not impact the operation and running of the Robot;
  - is a multi-step process that ensures that the software works according to its requirements;
  - helps in maintaining the quality and continuity of the code by resolving the errors.

Debugging. Steps to Apply

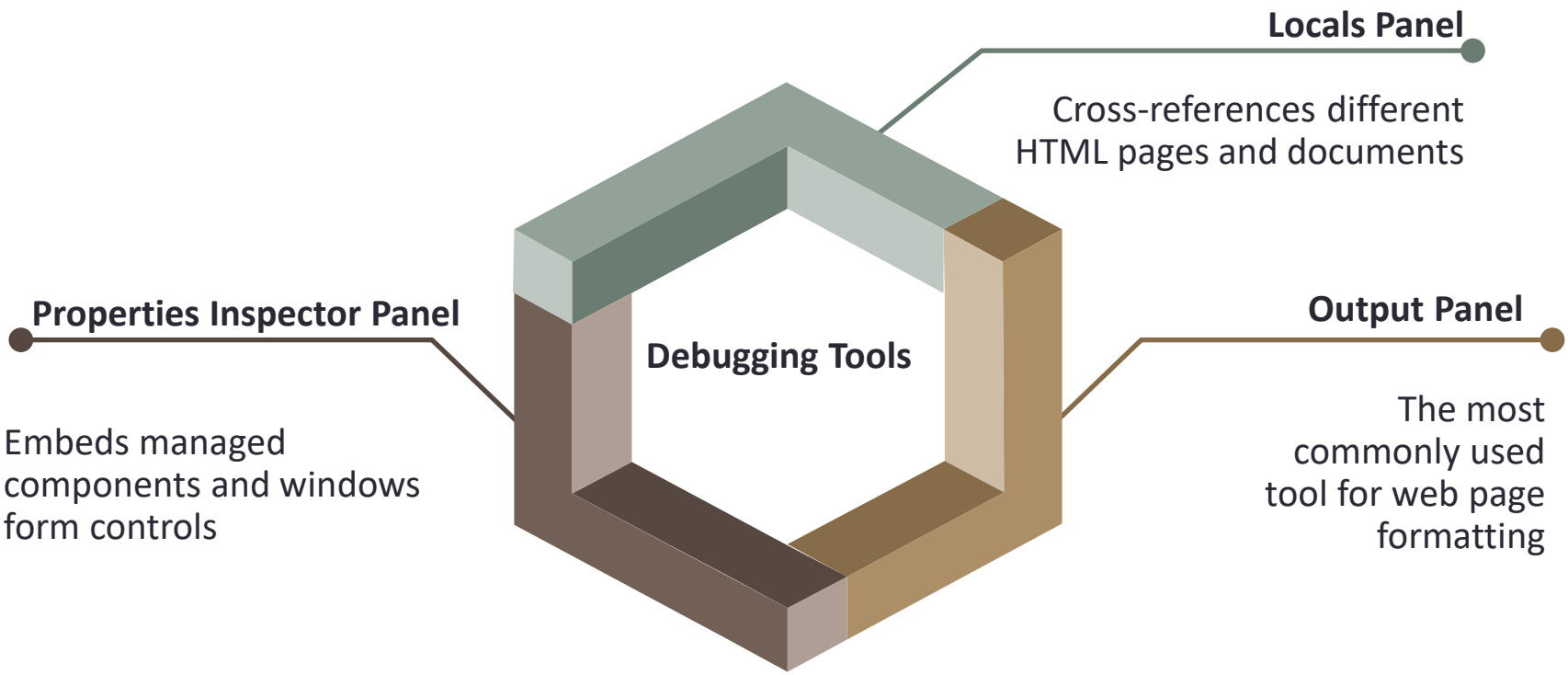
- In RPA, there are several steps which are helpful in **error handling**:
  - **Step Into** and **Step Over;**
  - **Breakpoints;**
  - **Slow Steps;**
  - **Log Activities.**



UiPath

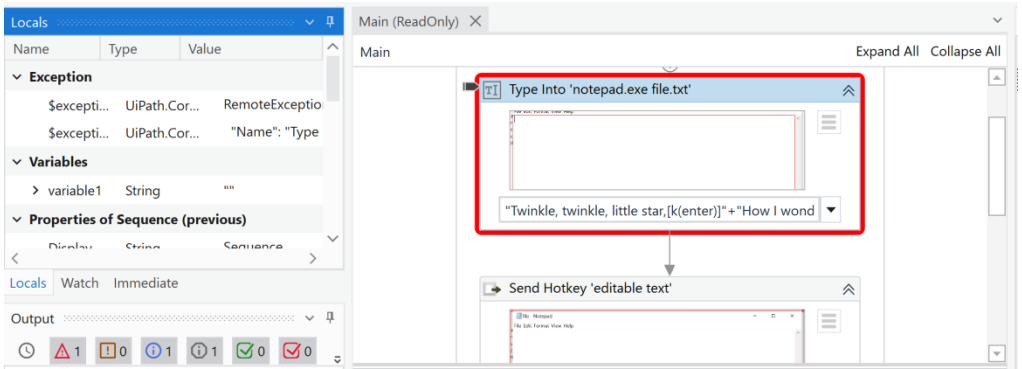
Debugging. Tools

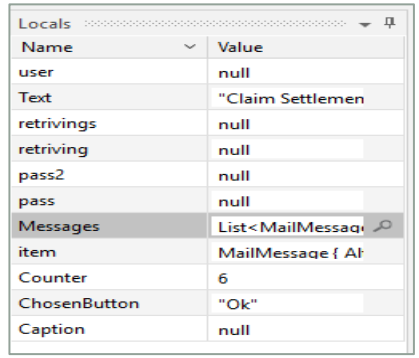
- on clicking **Debug File**, three panels appear on the workstation screen:

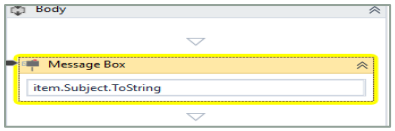


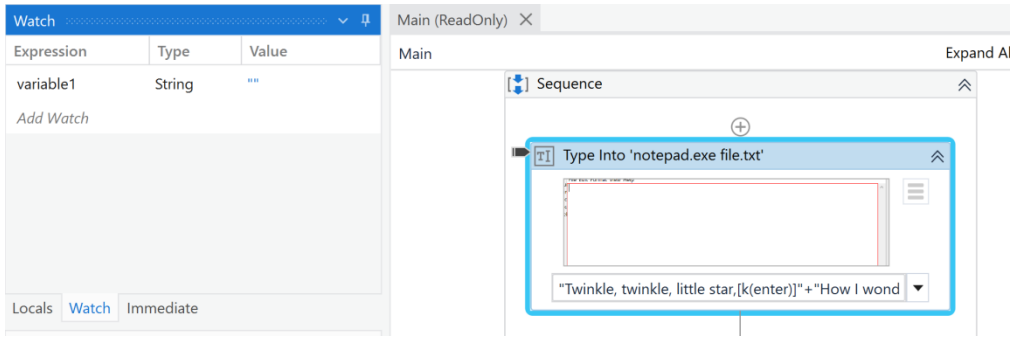
Locals Panel. Details

- **Locals Panel** shows
  - the **values of all the variables** in the current scope and
  - highlights the currently executing activity in colour (red, yellow, blue).



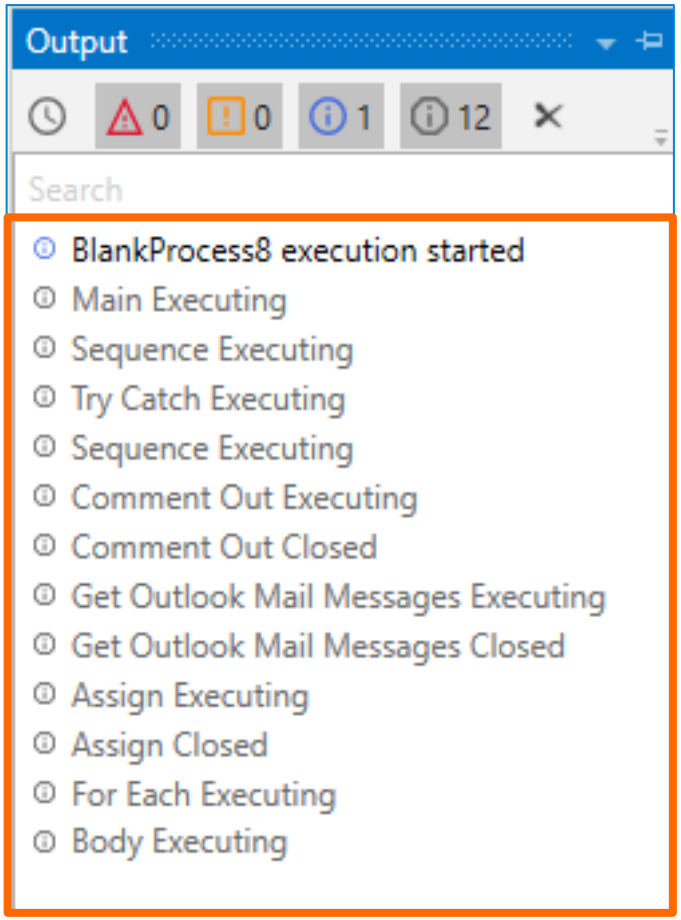






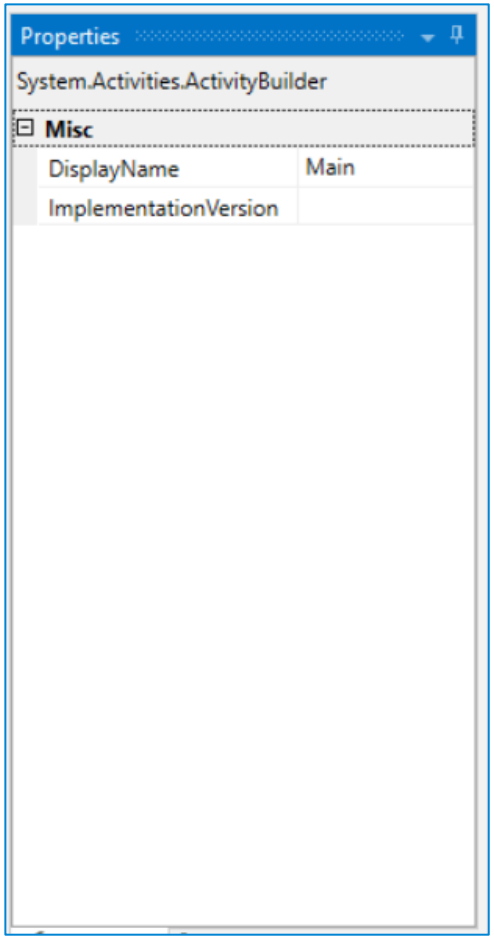
Output Panel. Details

- **Output Panel** shows
  - the **detailed log** of the current stage of workflow and the status of all activities executed by the Robot.



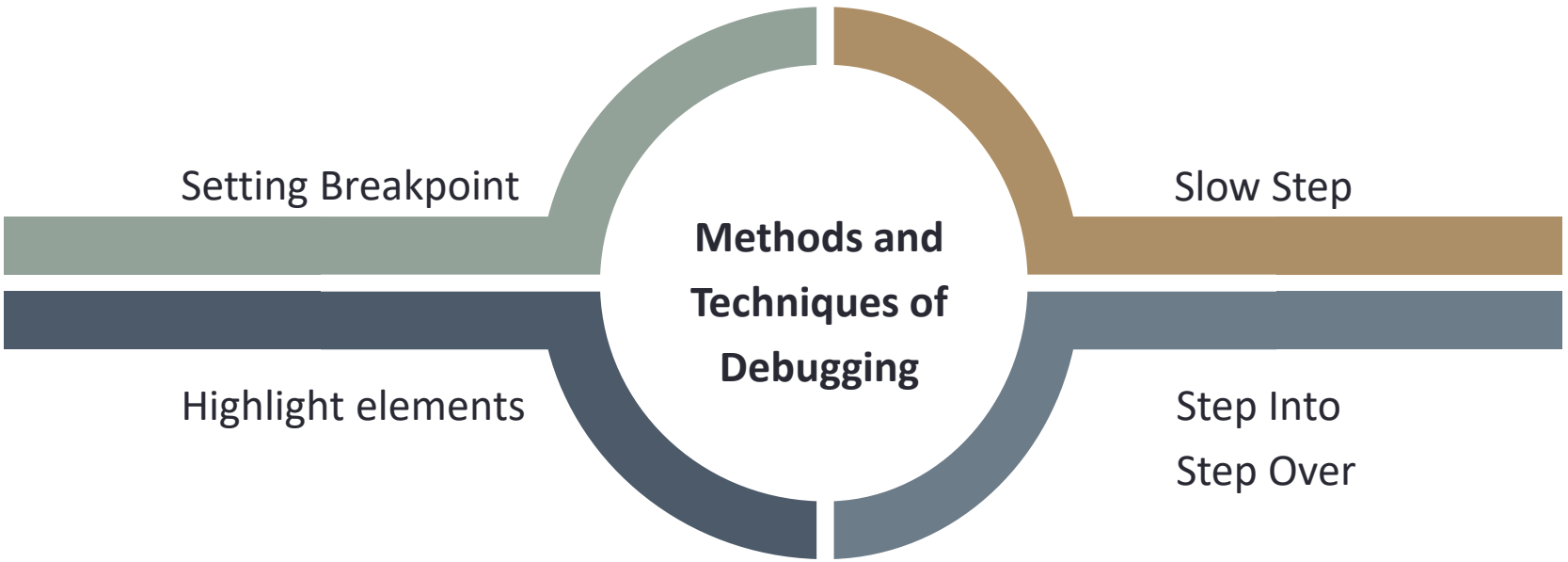
Properties Inspector Panel. Details

- **Properties Inspector Panel** shows
  - the active action properties, variable values declaration and debugging in the given data scope.



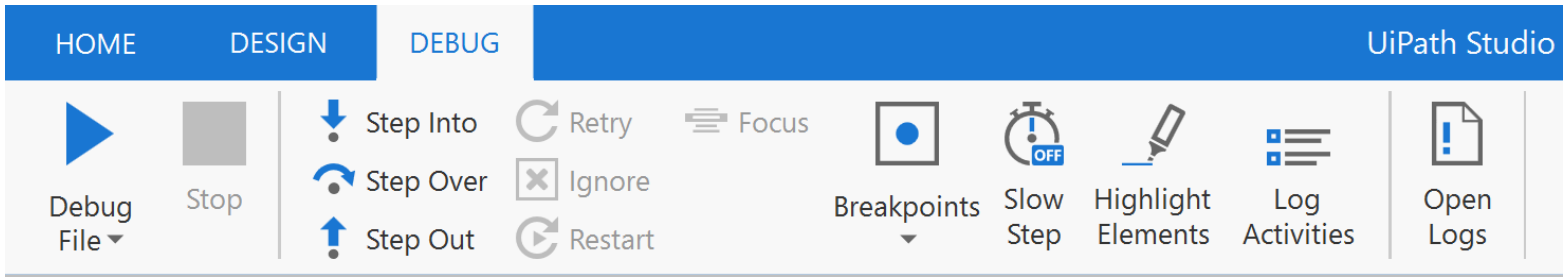
Debugging Techniques. Details

- **Debugging** is a feature embedded in UiPath Studio which can be accessed through the **Execute** Ribbon.
- there are various techniques for debugging.



Setting Breakpoints. Details

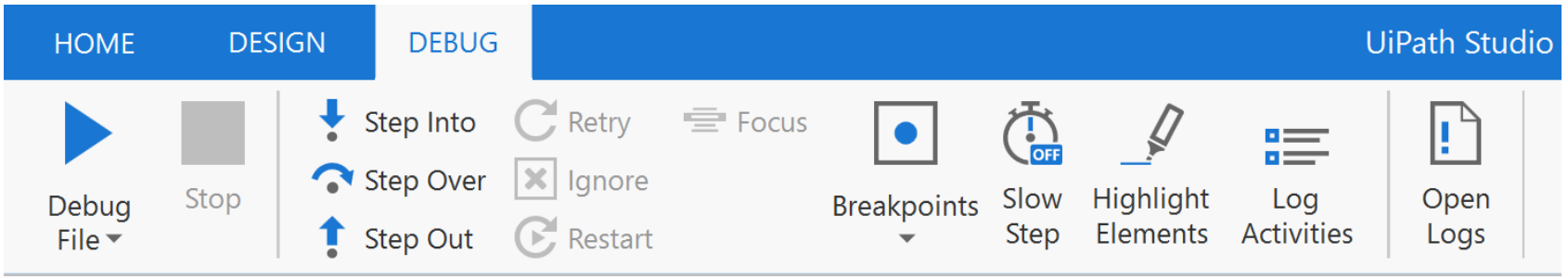
- **Setting Breakpoints** is
  - used when the user wants **to pause the program at a specific location**;
- once the breakpoint is set to an activity, the program will run till that activity but not execute it; the execution is paused and the user can see the current value of the variables, the current state of workflow, identify and correct the error causing element failure.
- the user can choose to **Continue**, **Step Into**, **Step Over**, **Step Out** or **Stop** the debugging process;
- Steps to enable breakpoints:
  1. Select the activity until the point which to execute in the project.
  2. Right-click and choose toggle breakpoint.
- the workflow can be continued by clicking on the **Resume** button which can start the breakpoint activity on the last activity basis.



Step Into & Step Over. Details

- **Step Into** is
  - used to start execution from the part where the user has indicated the breakpoint;
- **Step Over** is
  - used to jump to the next activity from the part where the user has indicated the breakpoint;

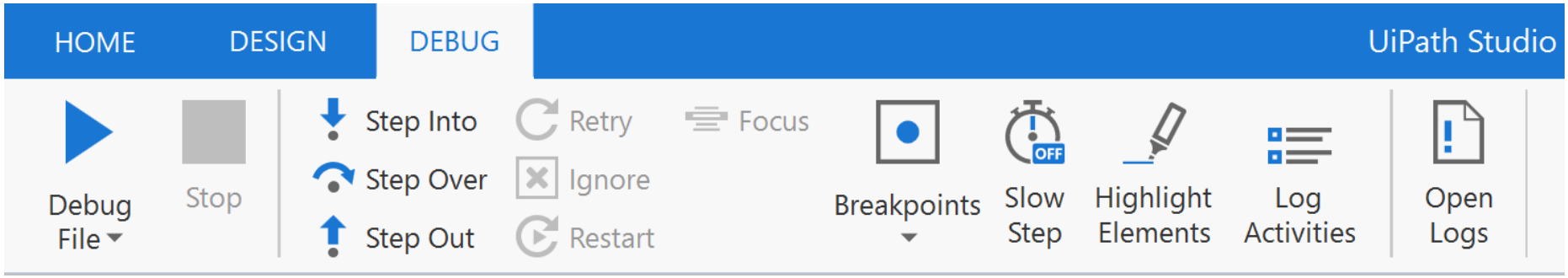
- **Both activities cannot work simultaneously.**



Slow Step & Log Activities. Details

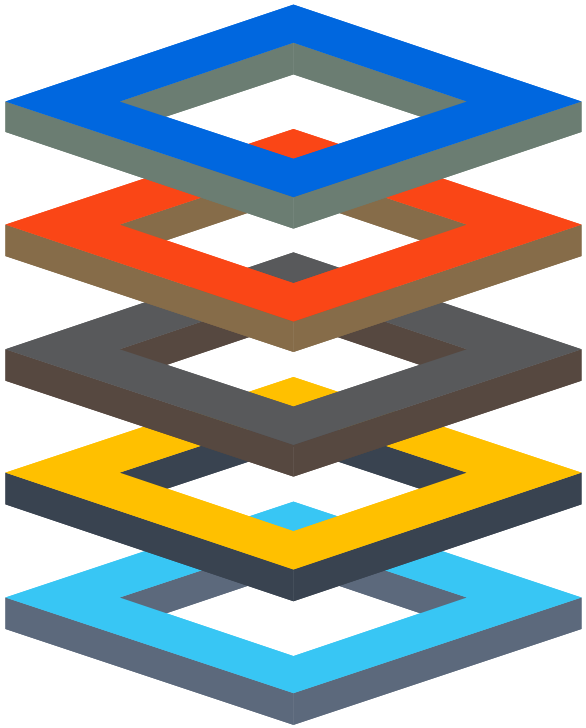
- **Slow Step** is
  - used **to reduce the execution speed** of the process;
- it helps the user to identify each activity and to see the changes of variable values after executing every activity;
- it can be activated and deactivated while the workflow is running.

- **Log Activities** allows
  - to list the activities that are executed while the debugging mode is on.



Best Guidelines for Error Handling

- Analyzing the process thoroughly, identifying the requirements and planning
- Breaking the process into smaller workflows
- Grouping the workflows of the project into different folders based on the target application
- Keeping consistent naming convention across the project
- Using the right type of argument (In/Out/InOut)



- Handling sensitive data responsibly
- Using Try/Catch blocks
- Using Global Exception Handler
- Using Libraries
- Adding annotations to the workflows

Next lecture

- **Lecture 06**
  - UI Interactions;
  - Custom activities.

References

- UiPath Docs - https://docs.uipath.com/studio/lang-en/v2019
- UiPath Forum - https://forum.uipath.com/
- UiPath Help Centre - https://www.uipath.com/developers/guides-and-resources
- UiPath Academy - https://academy.uipath.com/