**Variables**

Variables are mainly used for **Storage Representation**.

**Variables** are **used** to store information to be referenced and manipulated in a computer program.



Just like other Software Programming Languages, Variables are used in UiPath for automations. Variables are entities or data items that can store some value within them. This value can be changed, depending on computations implemented in the Program.

Variable simply refers to the name given to a memory location which is processed and retrieved during execution of a Program, as per its user's needs.

Variables are configured through their properties. You can set them in the Variables panel.

**The main properties in UiPath are:**

1. Name
2. Type
3. Default Value
4. Scope

**Name:** It should be as descriptive as possible to make your automation easy to read by other developers and to save time.

**Type:** Defines what kind of data can be stored in the variable. In UiPath, the type is declared when the variable is created, however there are some specific types that are more generic and can accommodate different types of data. More about variables types right below.

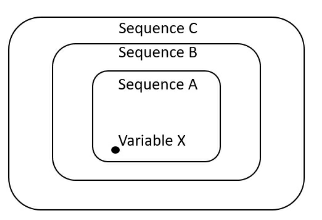
**Default Value:** In general, variables have initial values that change throughout the process. If no initial value is assigned at the creation of the variable, there is generally a default rule that assigns a value.

**Scope:** This simply defines the boundary of variable where it can be accessed by the activities and computed further. Beyond that scope, the variable is invisible for the activities. In UiPath, we define the scope of the variables at the time when we create them.

The part of the workflow in which the variable can be used. Some variables can be global, others local. In real automation scenarios, there are many variables in use. Making multiple variables unnecessarily global can cause efficiency issues as well as possibility for confusion.

UiPath allows variables to be kept within sequence or flowchart, so that they are not accessible to the outside activities of the workflow. The sequence or flowchart, in this case, defines the scope of visibility of that variable. This concept is called **variable scoping**.

**Consider the variable X in this case:**



When created in UiPath, be default, the scope of the variable X will be activities within sequence A. However, in Variable pane, the scope of the Variable X can be changed up to Sequence B or Sequence C, depending on its usage.

**Creating Variables:**

**There are 3 ways to create variables in UiPath:**

**From the Variables panel** – Open the Variables panel, select the ‘Create new Variable’ option, and fill in the fields as needed. When you need it, provide its name in the Designer panel or in the desired Properties field.

**From the Designer panel** – Drag an activity with a variable field visible (i.e. ‘Assign’) and press Ctrl+K. Name it and then check its properties in the Variables panel.

**From the Properties panel** – In the Properties panel of the activity, place the cursor in the field in which the variable is needed (i.e. Output) and press Ctrl+K. Name it and then check its properties in the Variables panel.

**Good Case Practices - Creating Variables**

**Use clear and consistent naming conventions** - one of the most common is Camel case (each word in the middle of the phrase is capitalized)

Snake case:

First1\_Name2, first\_name2

Upper or lower Camel case:

FirstName, lastName

Pascal case:

First1Name2, First1Name

Kebab case:

First-Name, First-Name1

**Make sure you define the scope of each variable correctly** - remember that a variable defined on a limited scope cannot be used globally. At the same time, in real automation scenarios, it is crucial for variables to be defined only in the scope in which they are used. Making multiple variables unnecessarily global can cause efficiency issues as well as possibility for confusion.

**Arguments**

In UiPath, the scope of a variable cannot exceed the workflow in which it was defined. Since business automation projects rarely consist of single workflows, arguments have to be used.

Arguments are very similar to variables – they store data dynamically, they have the same data types and they support the same methods.

**The difference between Variables and Arguments** is that **Arguments** can be used to pass the data between different workflows (UiPath Projects) but the **Variables** can't be used to pass the data between different workflows.

**Arguments** have an additional property – the direction from/to which the data is passed. The direction can be **In**, **Out** and **In/Out**.

* **In:** Arguments with "In" direction are supposed to be used within the current project itself. These are used to receive the value passed on to them from external workflows.
* **Out:** Arguments with "Out" direction are used to pass the information from the current workflow to some other workflows, for further computations
* **In/Out:** Arguments with "In/Out" direction can be used to other workflows as well as receive information from them.

**Naming Conventions - Arguments:**

Argument names should be in Upper CamelCase with a prefix stating the argument direction, Such as

in\_DefaultTimeout, in\_FileName, out\_TextResult, and io\_RetryNumber

**Data Types**

Each data variable has a data type associated with it. This type defines the nature of the value that variable is permissible to hold.

The data types in UiPath are borrowed from VB.Net.

Various types of data supported include Text, Integers, Date, Images, Decimals, Arrays, Lists, Files and many more.

**Below are some of the most common ones used:**

**1. Numeric (Category):**

Used to store numbers. Which ranges from -2147483648 to +2147483648.

There are different sub-types of numerical variables:

**Int32** - System.Int32 (signed integers): 10, 299, -100, 0x69

**Long** - System.Int64 (long integers): 5435435343O, -11332424D

**Double** - System.Double (allows decimals, 15-16 digits precision): 19.1234567891011

**2. Boolean:**

Is used to store logical values. It can only have one of the two possible values - True or False (System.Boolean)

**Note:** The values true and false are not case-sensitive.

**3. Date and Time (Category):**

DateTime data type is used to store values of date and time. By default, the format used by UiPath to store date and time is "mm/dd/yyyy hh:mm:ss"

**DateTime** - System.DateTime: Used to store specific time coordinates (mm/dd/yyyy hh:mm:ss). This kind of variable provides a series of specific processing methods (subtracting days, calculating time remaining vs. today, and so on). For example, to get the current time, assign the expression DateTime.Now to a variable of type DateTime.

**TimeSpan** - System.TimeSpan: Used to store information about a duration (dd:hh:mm:ss). You can use it to measure the duration between two variables of the type DateTime. For example, you can save the time at the start of the process in one variable (of type DateTime), the time at the end in another (of type DateTime) and store the difference in a variable of type TimeSpan.

**4. String:**

System.String

This data type is used to store multiple alphanumeric characters. The value of a string variable is enclosed in double quotes("").

This type of data comes with many specific methods of processing, and will be addressed in depth in another document, namely **Data Operations**.

**Please refer the "Data Operations" document for the operations which we can perform on variables; i.e. Integer and String.**

**Note:**

In activities such as **"Message Box"** and **"Write Line"**, to output the value of variables (**Other than String Type**), the variables are usually converted into String Type **(By using ToString method)**. UiPath also makes use of pre-defined functions available in .Net programming language to do computations such as Trim a string, Convert to Upper Case, Lower Case etc...

**5. Collection (Category):**

This category reunites all the collections of objects, with each object being identified through its index in the collection. Collections are largely used for handling and processing complex data. Some of the most encountered collections are:

**Array**- ArrayOf<T> or System.*DataType*[]:The array variable is a type of variable that enables storing multiple values of the same data type. Think of it as a group of elements with a size that is defined at creation, and each item can be identified by its index.

In UiPath Studio, you can create arrays of numbers, of strings, of Boolean values and so on

The size (number of objects) is defined at creation;

**Note:** Array comprises of two elements; i.e. Index and Value and shown in the below picture.



Array index points the execution flow to the location of data item within the array. Just as other programming languages such as Java, C, or .Net it always starts with a Zero (0). On the other hand array value shows the value of data item at that particular index. Thus as per the above figure, for an array with a name a, the first data item with value 23 would be available at index 0; i.e. A(0)

In UiPath, the index of an array is denoted by array name followed by circular brackets (). The values, however are initialized and/or assigned values by putting the assigning statement under curly braces {}.

**Ex:**

Array A = {1,2,3,4,5,6,7} Where

A(0)=1

A(1)=2

A(2)=3

A(3)=4

A(4)=5

A(5)=6

A(6)=7

**What are some business scenarios in which we can use arrays?**

1. When we want to save the names of the months to a variable
2. When a fixed collection of bank accounts has to be stored and used in the payment process
3. When all the invoices paid in the previous month have to be processed
4. When the names of the employees in a certain unit have to be verified in a database

**List** - System.Collections.Generic.List<T>: used to store multiple values of the same data type, just like Arrays. Unlike Arrays, their size is dynamic;

To add a date item to a List, use **"Invoke Method"** activity with the Target Type as List variable name, method name as "Add" and parameters (within Property Panel) set to Direction In, Type with Data Type of list variable and Values as the data item that needs to be added.

**"Invoke Method"** activity also supports pre-defined functions such as remove, reverse, sort, clear etc. For more information, check out the List<T> methods for Microsoft.NET Framework.

**List Properties:**

* Count - Works as Length does on an array, it returns the number of items in the collection.

**List methods:**

* Add(item) - This method takes an item as a parameter and adds to the end of the list.
* AddRange(collec­tion) - Adds multiple items to the list, e.g. from an array.
* Clear() - Removes all items in the List.
* Contains(item) - Returns true/false depending on whether the List contains the given item.
* CopyTo(array) - Copies all items in the list to a given array. We can specify the starting index and the number of items.
* IndexOf(item) - Returns the index of the first occurrence of a given item in the list (like with an array). Returns -1 in case of failure.
* Insert(index, item) - Inserts an item at the given index, position, in the List.
* InsertRange(index, collection) - Inserts multiple items from the collection at the given index in the List.
* LastIndexOf(item) - Returns the index of the last occurrence of a given item in the list. Returns -1 in case of failure.
* Remove(item) - Removes the first occurrence of an item.
* RemoveAt(index) - Removes the item at the specified index.
* RemoveRange(index, count) - Removes a specified number of items starting at a specific index.
* Reverse() - Works the same as it would in an array. It reverses the List so the first item is the last and vice versa. The method doesn't return anything, the changes are made directly to the List.
* Sort() - This method sorts the items in the List. The method doesn't return anything either.
* ToArray() - Copies the items from the List to a given array and returns it.

**Other List Methods:**

List also provides all of the methods that are present in arrays:

* Average() - Returns the average of the items in the List as a double.
* Distinct() - Returns all of the items of which there is only one instance in the List.
* First() - Returns the first item.
* Last() - Returns the last item.
* Intersect(collec­tion) - Returns the intersection of the List with the specified collection.
* Union() - Returns the union of the List with the specified collection.
* Min() - Returns the smallest element.
* Max() - Returns the largest element.
* Sum() - Returns the sum of the elements.
* Take(number) - Returns a specified number of elements starting at the beginning.

Now we have seen that the List collection can do much more than arrays. The biggest advantage to it is being able to add and remove items. Drawbacks in the performance are insignificant to us now.

**Dictionary** - System.Collections.Generic.Dictionary<TKey, TValue>: used to store objects in the form of (key, value) pairs, where each of the two can be of a separate data type.

In general, ***Dictionary*** is collection of words and their meanings/definitions.

Dictionary is having two elements; i.e. **Key and Value**

**Whereas Key - Word**

**Value - Meanings/Definitions**

Similar to Arrays, Dictionaries are like two synchronized collection of names and associated values. But instead indexes, these values are referenced by the Names. This data type is mostly used to extract the data from Configuration File and extract the data from Queues within UiPath Orchestrator.

To assign a value to a dictionary variable, simply use "Assign" activity in the format:

Variable\_name("Key") = "Value"

**Points to remember while working with Dictionary:**

1. By default Dictionary activities are not available in UiPath Studio. For this we have to install the "Microsoft.Activities.Extenstions" package.

2. Dictionary is a Case-Sensitive.

3. Dictionary cannot include duplicate or null values, whereas values can be duplicated or set as null.

4. A Dictionary stores Key-Value pairs where Key must be unique.

**Difference between Array and List:**

Data Type List is same as Array. The only difference is only that *an Array has a Fixed Length* while *length of a List can be increased or decreased* by adding or removing elements from it.

**6. GenreicValue:**

UiPath.Core.GenericValue

GenericValue Data Type variables are specific to UiPath.

This is a UiPath proprietary variable type that can store any kind of data, including text, numbers, dates, and arrays. This type is mainly used in activities in which we are not sure what type of data we will receive, yet in general the use of this is temporary.

**What are some business scenarios in which we can use GenericValue variables?**

1. Data is extracted from a UI field and forwarded to another workflow without processing
2. Two versions of the same Excel file are being compared column by column. The columns are different in terms of data type, the only relevant thing is which entries have changes

**Sample Exercise:**

Initialize two *generic variables* with values "Hello" and "World" and concatenate them using '+' operator. Display the output in a Message Box. Now change the values of these two variables into some integers and continue the same operation. Do you the difference?

Please remember that **v**ariables are a temporary solution at most. When it becomes clear what the data type should be, we strongly recommend that you change it to the specific type.

**Note:**

Keep in mind that the list of types presented above is not a complete list, but the list of the most common types used. Other types may be used in specific situations. When browsing or searching, you will find most of them under the System and **System.Collections** categories.

In some cases, variables are generated automatically by activities, and their types may vary – **for example**, an activity that locates and stores a graphic element will automatically generate a variable of **UiElement** type.

**Control Flow**

**Control Flow Overview:**

It is the order in which **individual statements**, **instructions**or **function calls**are executed or evaluated in a software project.

The execution of a workflow needs to be throughly regulated and managed to make sure that the bot performs precisely as per business needs.

Control Flow refers to the concept of causing the flow of program to advance and/or branch to the next state based on the validity of current state. It defines the rules and validations performed by the workflow.

**This goal is accomplished by using;**

1. The Type of Automation Project

2. The Control Flow Statements

**1. The type of automation project:**

There are 4 predefined types of workflows –

1. Sequence
2. Flowchart
3. State Machine
4. Global Exception Handler
5. **Sequences:**

The process steps flow in a clear succession. Decision trees are rarely used. Activities in sequences are easier to read and maintain, thus, they are highly recommended for simple, linear workflows.

Suitable to linear processes, enabling you to smoothly go from one activity to another, without cluttering your workflow.

1. **Flowcharts**:

The individual activities are a bit more difficult to read and edit, but the flows between them are much clearer. Use flowcharts when decision points and branching are needed in order to accommodate complex scenarios, workarounds and decision mechanisms.

Suitable to a more complex business logic, enabling you to integrate decisions and connect activities in a more diverse manner, through multiple branching logic operators.

1. **State Machine**:

A State Machine uses a finite number of sets in its execution. It can go into a state when it is triggered by an activity; it exits that state when another activity is triggered.

Another important aspect of State Machines is transactions. They enable you to add conditions based on which transactions jump from one state to another. These are represented by arrows or branches between states.

Suitable for very large workflows; they use a finite number of states in their execution which are triggered by a condition (transition) or activity.

1. [**Global Exception Handler**](https://studio.uipath.com/docs/global-exception-handler)**:**

Suitable for determining the workflow behavior when encountering an execution error, and for debugging processes.

**2. The Control Flow Statements:**

The activities and methods used to define the decisions to be made during the execution of a workflow. The most common control flow statements are the I**f/Else** decision (Sequence) and the **Flow Decision** (Flowcharts),the **Loops**and the **Switch**. Let's focus on them one by one.

**If Statement:**

**In UiPath, the If statement is exactly how you’d expect it to be:**

* The condition that is verified (with 2 potential outcomes – true or false)
* The set of actions to be executed when the condition is true (the Then branch)
* The set of actions to be executed when the condition is false (the Else branch)

**What is different is that, based on the chosen type of automation project, there are 2 corresponding activities that fulfill the If statement role:**

* The If Statement in sequences
* The Flow Decision in flowcharts

Moreover, the If decision can be used as an operator inside activities.

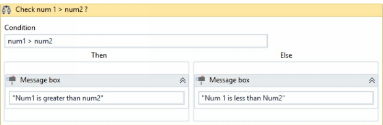
**The general form of If activity is something like this:**

If(Condition = TRUE)

then Sequence a

Else

Sequence b



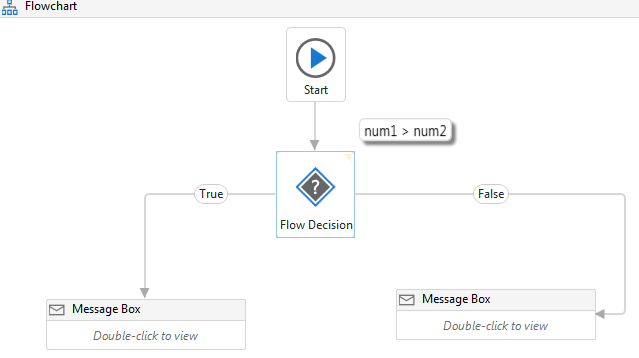
The diagram above shows a simple if statement that validates whether a number is greater than the other or not. The result of this condition will define the direction of the workflow and correspondingly the relevant output message.

**Note:**

If statement is for the sequence workflows and is corollary to the Flow Decision activity, which is handled in Flowchart mode.

**Flow Decision:**

Flow Decision is another conditional activity used within Flowcharts to make decisions based on the accuracy of the condition holds. It includes a true node and false node as its outputs. Both of these are connected to the further actions needed to be undertaken by the workflow. This activity is shaped like a diamond.



The diagram above shows a simple Flow Decision statement that validates whether a number is greater than the other or not. The result of this condition will define the direction of the workflow and correspondingly the relevant output message.

**What are some business scenarios in which I will use the If statement?**

Whenever there are two courses of action that are not arbitrary, an If statement will most likely be used:

* Checking the status of a payment (done/not done) and performing a set of operations in each case
* Making sure that the outcome of the previous operation in the sequence is successful
* Checking the balance of an account to ensure that there is enough money to pay an invoice
* Checking if something has happened in a system, like if an element or an image exists and performing an action based on that.

**Loops:**

A Loop refers to the repeated execution of same set of activities until conditional expression it holds is not valid anymore.

In UiPath, the most important Loops are:

1. While
2. Do While
3. For Each
4. **While:**

A While loop is the most fundamental looping in activity within UiPath. It repeats the execution of a sequence of activities while its controlling expression is true.

It executes a specific sequence while a condition is met. The condition is evaluated before each execution of the statements.

The general expression of While Loop is :

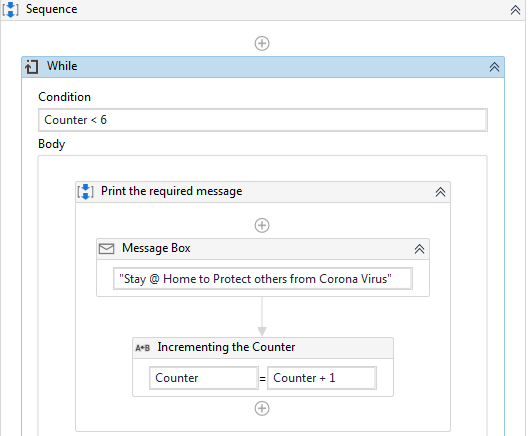
While (Condition = TRUE)

Sequence of Activities

The Condition results in Boolean output, on which further activities are executed. The sequence of activates will keep executing until the condition is valid. In this sequence of activities (called body), there's mostly an iteration or modification of the operand used in conditional expression operand used in conditional expression. This ensures that check holds true (or false) while repeating the execution and the loop doesn't get stuck with infinite executions. As soon as the conditional expression is invalid, the execution comes out of the activities enclosed within the body and continues with rest of the workflow.

The use of while loop can be demonstrated with the following use-case:

Print message "Stay @ Home to Protect others from Corona Virus" - 5 times



**Points to remember:**

1. Counter is an Integer (int32) variable with default value is 1

2. Conditional expression is checking if Counter value is less than 6 (since we want to display message 5 times)

3. Incrementing the Counter variable - 'Assign' activity used for incrementing the value Counter variable by 1

For example, if a Robot would play Blackjack, it should calculate the hand before deciding

whether to draw another card.

1. **Do While:**

The Do While activity is very similar to the While activity described above.

It executes a specific sequence while a condition is met. The condition is evaluated after each execution of the statements.

However, note that if the controlling statement in a **While loop** is **false** since beginning of execution itself, the body of the loop will never be executed. The workflow will be move out of the loop and will continue the execution of subsequent activities. This might not be the desirable case at all times. In other words, there are times when we need to execute the sequence of activities listed in the body at least once. This is where **Do While loop** comes into picture. It makes sure that the body of the loop is executed at least one time. This is because the conditional expression in this loop is evaluated after the body has been executed initially and repetitive execution depends on Condition's validity.

The general form of Do While loop is:

Do Sequence of Activities

While (Condition = TRUE)

**For example**, a robot could perform a refresh command over a website and then check if a relevant element was loaded. It will continue the refresh - check cycle until the element is loaded.

1. **For Each:**

This is one of the most commonly used looping techniques in UiPath. It performs an activity or a series of activities on each element of a collection.

This is very useful in data processing. Consider an Array of integers. For Each would enable the robot to check whether each numeric item fulfills a certain condition.

**For Each** activity used to traverse through Lists, Arrays, and Collection Data Types, to process each item within that data structure individually. The iteration of the data item within the data structure executes automatically i.e. unlike While and Do While loops, we don't need to use incrementing activity in For Each loops.

The General form of For Each loop is:

For Each data item in data structure ( or VB expression)

Sequence of Activities

**Sample Use-Case:**

Display all the Country Names from CountryNames array with 4 different Countries.

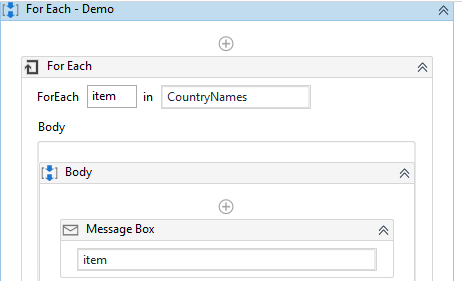
Step 1: Create an array variable of string type with variable name as CountryNames.

Step 2: Initialize the variable with values as India, Singapore, Sri Lanka, USA, and UK.

Step 3. Initiate a For Each loop to traverse through array items individually and change the 'TypeArgument" from Object to String.

Step 4: Display each item of the array.

**The workflow for this use-case would like something like this:**





**Points to Remember:**

- Initialization of an array within default field of variable pane ( or using Assign activity) is always enclosed in curly braces {}

- Array Data Type is always succeeded with square braces '[]' such as String[], Int32[] etc.

- The loop variable is type-casting to string data type to make it displayable within workflow.

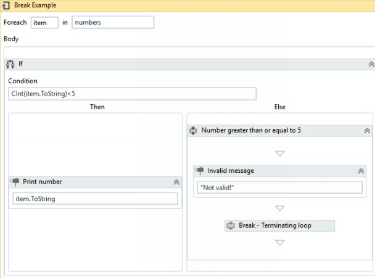
**Break Activity:**

The Break activity is always used within a For Each loop. Once encountered, it forcibly terminates the scope of the loop, sidestepping the conditional statement and any code within the body section of that loop. When this activity is executed inside a loop, any remaining activities within that loop are not executed and the flow of the program jumps to the next activity, available outside of the loop.

**Sample Use-Case:**

Consider that you are traversing through a numeric array and displaying all numbers that hold value less than 5. How, if you encounter a value greater than or equal to 5, you will simply come out of the loop by printing "Not Valid".

The Workflow for this Use-Case is:



Point to Remember:

- The loop variable 'item' has been made readable by making it a string and has been further casted to type integer, to perform the conditional statement.

- By default, item variable is of Object data type.

- Break condition would immediately terminate the execution flow from the loop and will continue with any activities added in the workflow, outside the loop.

1. **Switch:**

Switch activity is a multi-way branching activity. It provides a simple way to carry out the execution of workflow to different paths based on the value of an expression. In simple words, it provides an optimal alternative to a large series of if-else activities.

The general form of this activity looks like:

Switch(expression)

Default: Default sequence

Case Value1:

Sequence 1

Case Value2:

Sequence 2

Case Value 3:

Sequence 3

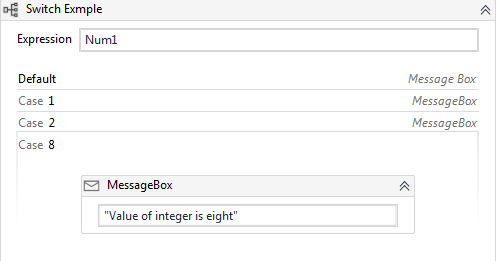
...

....

And so on

**Sample Program:**

Display the value of an integer variable if it lies in the range of 1-10 else display the message "Not Valid Range".





**Points to Remember:**

- Num1 is the integer variable who's value needs to be checked

- Default case displays the "Not Valid Range" message

- All cases define the message with the value of Num1 variable they provide in case value textbox

**What are some business scenarios in which I will use Switch?**

* An invoice that has 3 potential statuses (not started, pending, approved) and 3 sets of actions for each one
* A process of automatically ordering raw materials to 4 suppliers based on certain conditions