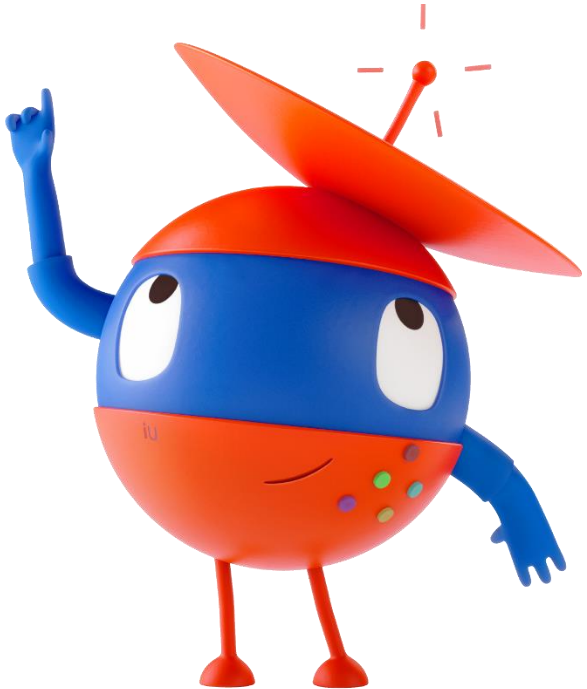
RPA Design and Development

v4.0



UiPath

Lesson 3&4 ¦ Studio Interface, Variables & Arguments

UiPath

Exam Topics:
Studio Interface, Variables & Arguments

Studio Interface:

1. Describe the different variable types, how they are used, managed, and the best practice for using the variable scope
2. Describe the functions and differences between variables and arguments; including how arguments are used, managed, and best practices

Variables and Arguments:

1. Explain what data types are and how they are used
2. Create, manage and use variables
3. Create, manage and use In, Out and In/Out arguments.
4. Create, manage and use global constants and variables
5. Explain the difference between variables, arguments, global constants and global variables.

What is a variable?

A variable is an element that holds data or values of a certain type.
It serves the essential purpose of storing data and passing it between activities.

Variables can be configured through four properties.
They are name, variable type, scope, and default value.

There are four ways of creating variables in Studio. They are:

- From the Data Manager
- From the Body of an Activity
- From the Properties Panel
- From the Variables Panel

Introduction to Variables

Variables are containers that are used to store multiple types of data. A variable:

- makes it easier to label and store data which can later be used throughout the automation process

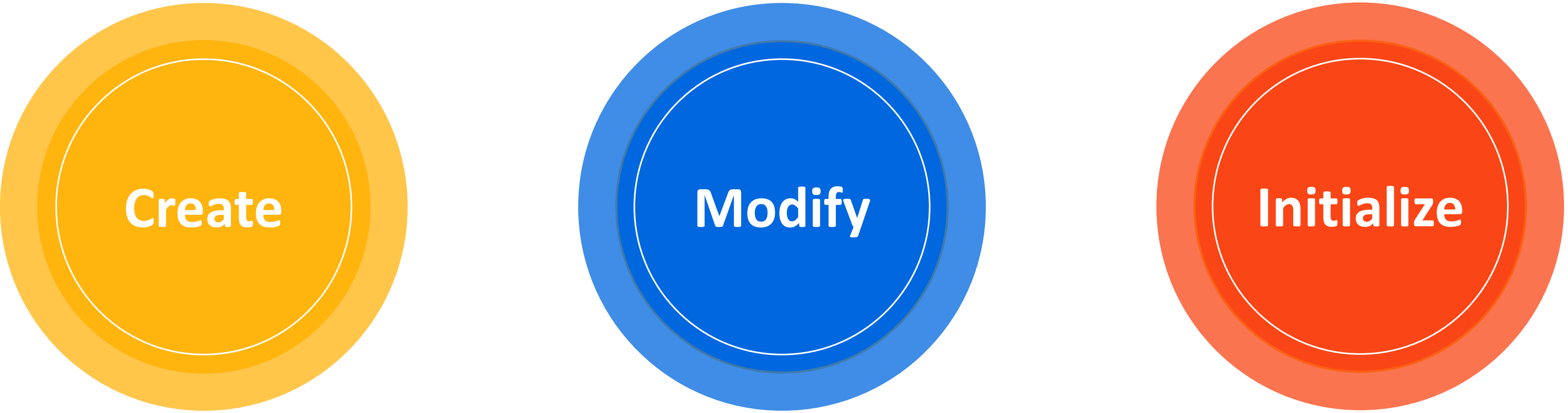
- can be used throughout the automation process later and can also be used in multiple workflows with modifications

- has an initial value that may change during the program through an external input, data manipulation or passing from one activity to another

- is like a box that stores data
  Example: A box (variable) named Counter that tracks the number of times users clicked on an item

Managing Variables

In Studio, a user can create, modify and initialize variables from the Variables Panel.



Create

Modify

Initialize

Creating, Using & Managing Variables

You can imagine variables as containers that hold data (value) of a certain type.

The value of a variable can change during the program's execution due to an external input, data manipulation, or as a result of passing from one activity to another. In other words, variables store data dynamically.

Variables are vital in automation as they are one of the fundamental methods of storing data and passing it between activities.

Configuring a variable

Variables in Studio are configured through four main properties :

- Name
- Variable Type
- Scope
- Default Value

Properties of Variables

Users can configure variables through their properties:

1

Name

Name of the variable

2

Type

Kind of data that the variable is intended to store

3

Value

Data that a variable holds (may change during the process)

4

Scope

Designates parts of a program that can use a variable (local, global)

Configuring a variable : Name Property

It is the unique attribute that is used to identify a variable.
This is a mandatory field.
If you don't add a name to a variable, one is automatically generated.

In order to make automations easy to understand and manage, the names of the variables must be meaningful and as descriptive as possible.

While not the only option, **we recommend using the PascalCase naming convention**.

In this convention, the first letter of each word in a variable is capitalized. For example: ItemValue, LastName, or KeyItem.

Configuring a variable : Variable Type

It defines the kind of data stored in a variable.
This is a mandatory field.

Some of the common data types include:

- Boolean
- Int32
- String
- Object
- System.Data.DataTable
- Array of [T]

Apart from these, UiPath also supports data types from imported dependencies.

Configuring a variable : Scope Property

It defines the context in which a variable can be used in the project.
The scope is a mandatory field.

The scope of variables can be set to the current workflow file or any of the container activity within the workflow file.

Apart from this, **the scope of variables can also be set to global**, meaning that they are accessible to all activities and workflows in an automation project. These are called global variables.

Scope of a Variable

The scope determines the containers in which the variable is available.

A variable declared in any specific activity is available only for the scope of that activity

The scope is chosen from the list of sequences in the **Scope** drop-down field while creating a variable. The variable is available in the selected container

A variable declared for a parent activity is available in the entire workflow

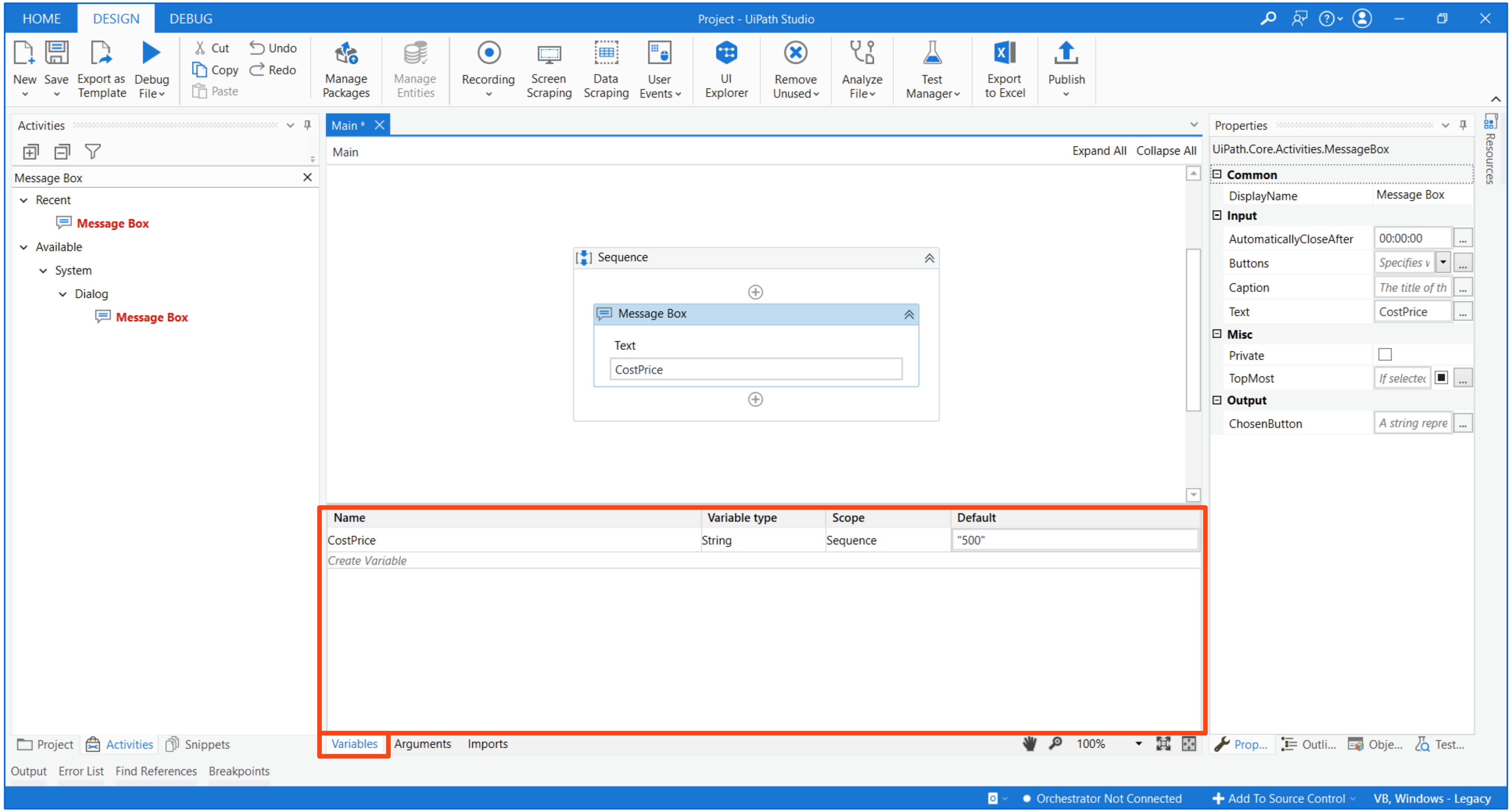
Configuring a variable : Default Value

It is the default value of the variable.
This is an **optional** field.

If a variable is declared with this field empty, then a default value corresponding to the variable's data type is assigned to it. For example, for an Int32, the default value is 0.

The Variables Panel

It enables the users to create variables and modify them.



Message Box

Text

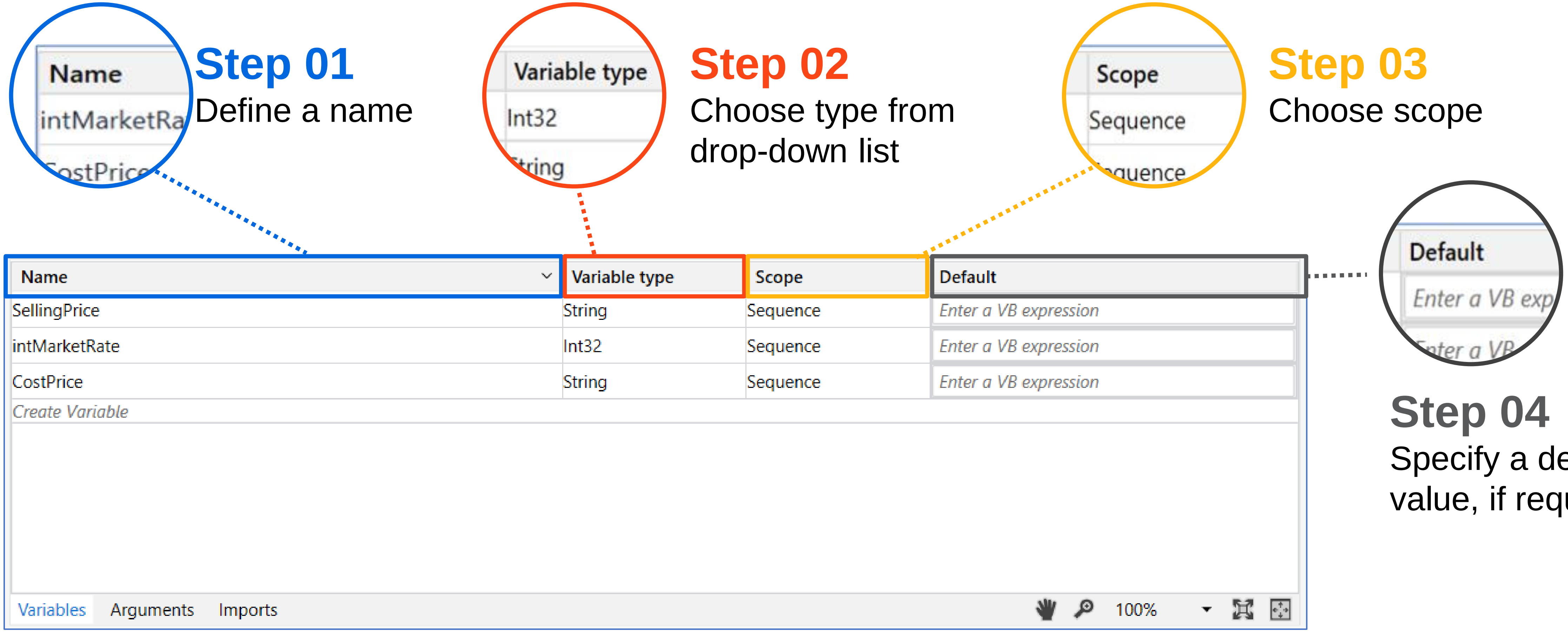
CostPrice

Name	Variable type	Scope	Default
CostPrice	String	Sequence	"500"
Create Variable			

UiPath.Core.Activities.MessageBox

Common	
DisplayName	Message Box
Input	
AutomaticallyCloseAfter	00:00:00
Buttons	*Specifies v*
Caption	*The title of th*
Text	CostPrice
Misc	
Private	☐
TopMost	*If selectec*
Output	
ChosenButton	*A string repre*

Variables Panel



Name	Variable type	Scope	Default
SellingPrice	String	Sequence	Enter a VB expression
intMarketRate	Int32	Sequence	Enter a VB expression
CostPrice	String	Sequence	Enter a VB expression
Create Variable			

Step 01
Define a name

Step 02
Choose type from drop-down list

Step 03
Choose scope

Step 04
Specify a default value, if required

UiPath™ Academy

Chapter 3:

Variables, Constants and Arguments in Studio

Creating, Using and Managing Variables

Best Practices for Naming Variables

A variable’s name should be meaningful and hint towards the information it stores. While naming any variable, the user should:

- Use clear & meaningful names
- Assign names in a consistent manner
- Use Camel case to name variable
- Keep the names descriptive yet short

Best practices when working with variables

Assign Meaningful Names

Meaningful names should be assigned to variables in order to accurately describe their usage in a project.

Follow a naming convention

To improve readability, variable names should also align to a naming convention. We recommend using the **PascalCase** naming convention. In this convention, the first letter of each word in a variable is capitalized.
Eg: First1Name2, First1Name

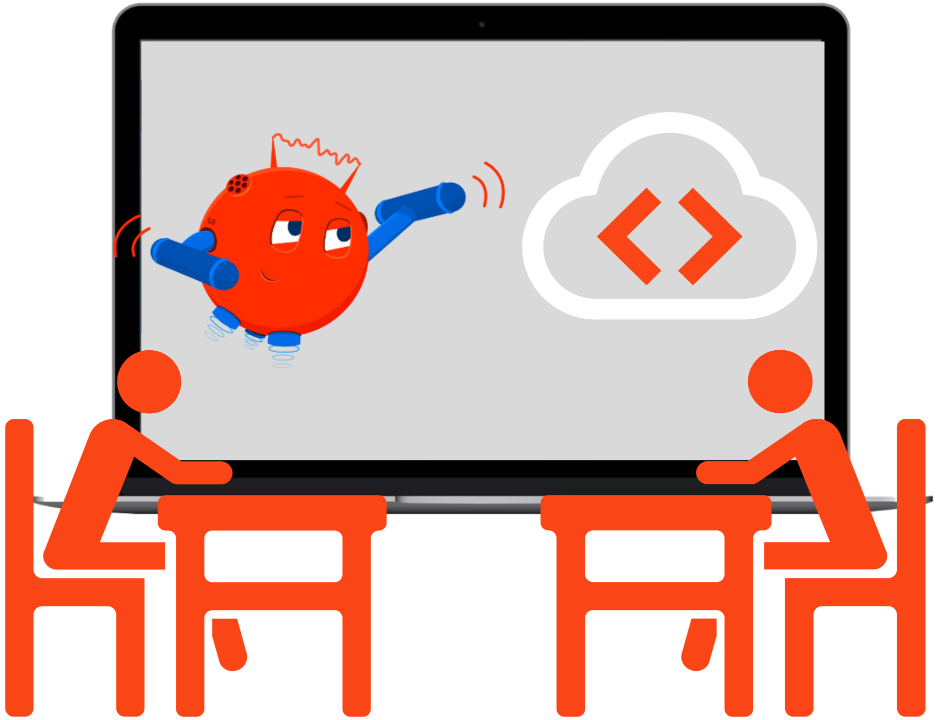
Keep variables in the innermost scope

Variables should be kept in the innermost scope to reduce the clutter in the Variables panel and to show only what is relevant at a particular point in the workflow.

Resources

Managing Variables - UiPath Studio Guide	https://docs.uipath.com/studio/standalone/2022.10/user-guide/managing-variables
Types of Variables - UiPath Studio Guide	https://docs.uipath.com/studio/standalone/2022.10/user-guide/types-of-variables

Practice Exercise - Variables



Build a workflow that swaps two numbers using a third variable

- Ask the user to input two numeric values and store them in two variables
- Swap values of both the variables using a third variable
- Display initial and swapped values of both the variables in the Output panel

Classroom Exercise

Create a project that

1. Asks for the user’s name, stores it and
2. Display only the first letter of the full name
3. Ask for the year of birth as well and calculate the age
4. Display the name and age in a window
5. At the same time, write the result of the execution to the Output Panel

Exploring Data Types

What are data types?

As you can guess by the name, data types describe the kind of data a variable can hold.
For example, if the data type is Int32, then the variable must hold an integer.
Likewise, if the data type is a String, then the variable must hold a text.

Why are data types important?

Variables and arguments in Studio need to have a specific data type defined.
Data types ensure that the right data format is applied at each stage of an automated process.

Types of Variables

Different types of variables in Studio are:

1 String

2 Boolean

3 Number

4 Date and Time

5 DataTable

6 QueueItem

7 Array

1. String Variables

Definition

String variables help store any sequence of text

Usage

To store text and reuse it in the code for specific actions

Example

- Company Name in Invoice
  - Variable name can be CompName with value “XYZ Corp”.

2. Boolean Variables

Definition

Boolean variables hold only two values: “True” or “False”

Usage

Used with control statements to help determine the flow of a program

Example

- Is the item available in the invoice?
  - TRUE when the item is available
  - FALSE when the item is unavailable

3. Number Variables

Definition

Number variables store numeric values

Usage

To execute equations or perform comparisons, pass important data, etc.

Example

- Item Quantity
  - Variable name can be ItemQuant with value 50

4. Array Variables

Definition

An Array variable is a collection that stores multiple elements of the same data type

Usage

Used to organize data so that a related set of values can be easily sorted or searched

Example

- An array of names of two items
  - Variable name can be ArrCompName with value {“ABC”, “XYZ”}

Array vs. String

Array	String
It is a sequential collection of elements of similar data types	It is a sequence of single characters represented as a single data type
Its elements are stored contiguously in increasing memory locations	It can be stored in any manner in memory locations
It is a special variable that can hold more than one value at a time	It can hold only character data
Its length is predefined	Its size is not predefined

Exploring Data Types

Collection

Collections are largely used for handling and processing complex data. Some of the most encountered collections are:

- **Array - ArrayOf<T>** or System.DataType[] stores multiple values of the same data type. The size (number of objects) is defined at creation.
- **List - System.Collections.Generic.List<T>** stores multiple values of the same data type, just like Arrays. But unlike Arrays, the size is dynamic.
- **Dictionary - System.Collections.Generic.Dictionary<TKey, TValue>** stores objects in the form of (key, value) pairs, where each of the two can be a separate data type.

5. DataTable Variables

UiPath

Definition

DataTable variables store tabular data in rows & columns and may hold large pieces of data & act as a database

Usage

Used to migrate data from a database to another, extract information from a website and store it locally in a spreadsheet

Example

- List of all items in the invoice
  - Variable name can be dt_InvItem with values Item1, Item2, Item3,….

Exploring Data Types

DataTable

DataTable represents variables that can store big pieces of information and act as a database or a simple spreadsheet with rows and columns.

.

6. Date and Time Variables

Definition

Date and time variables store information about any date and time

Usage

Used to calculate the number of days between two dates, store current date details, etc.

Example

- Invoice Date
  - Variable name can be InvDate with value 01/01/2020

Exploring Data Types

Date and Time (Category)

DateTime - System.DateTime stores specific time coordinates (mm/dd/yyyy hh:mm:ss). This kind of variable provides a series of specific processing methods like 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 stores 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.

7. QueueItem Variables

Definition

A variable particular to UiPath, the QueueItem variable stores an item extracted from a queue (container of items)

Usage

Used to input extracted items in other processes

Example

- A particular invoice in the queue of invoices

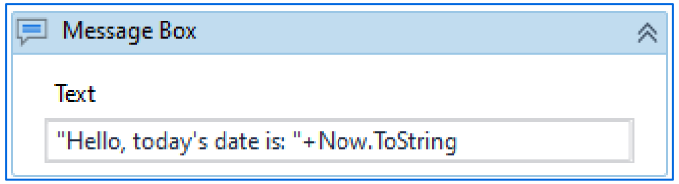
Data Conversion

Data Conversion is the process of converting one type of data to another. The methods for data conversion include:

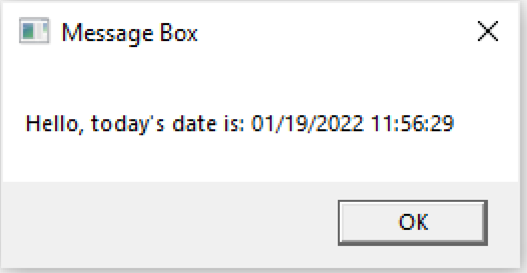
Assign
Simply assigning the data value to the desired data type

.ToString Method
Convert any datatype to string

Raw DateTime

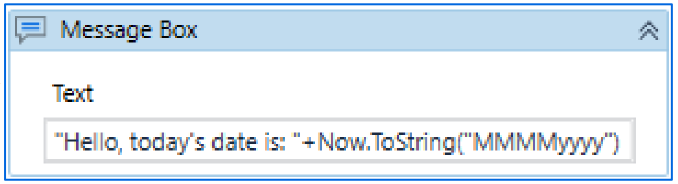


Text
"Hello, today's date is: "+Now.ToString

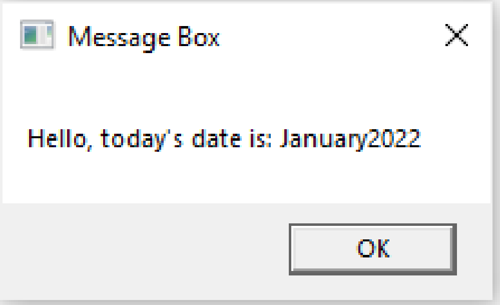


Hello, today's date is: 01/19/2022 11:56:29

Formatted DateTime



Text
"Hello, today's date is: "+Now.ToString("MMMMyyyy")



Hello, today's date is: January2022

Conversion methods of Data Types

There will be scenarios where we'll need to change the data type of a variable to another form. For example, the conversion of an integer to a string or vice versa. In such cases, we can use the conversion methods available in Studio.

Below is a list of some of the most commonly used conversion methods

- Convert.ToString Method
- Convert.Int32 Method
- Double.ToString Method
- Double.Parse Method
- Boolean.ToString Method
- Convert.ToBoolean Method
- Convert Date and Time to String

Conversion methods of Data Types

Convert.ToString Method

This method converts the specified value to its equivalent string representation.
For example, from an integer to String.

Eg: StrVar = Convert.Tostring(IntVar)

Conversion methods of Data Types

Convert.Int32 Method

This method converts a specified value to an integer.
For example, from a String to integer or from a floating-point number to integer.

Eg:

IntVar = Convert.ToInt32(StrVar)

IntVar = Convert.ToInt32(DblVar)

There's another method to convert a string to an integer - CInt(String)

Eg: IntVar = CInt(StrVar)

Conversion methods of Data Types

Double.ToString Method

This method converts the numeric value of a floating-point number to its equivalent string representation.

Eg: StrVar = DblVar.ToString

Conversion methods of Data Types

Double.Parse Method

This method converts the string representation of a number to its floating-point number equivalent.

Eg: DblVar = Double.Parse(StrVar)

Conversion methods of Data Types

Boolean.ToString Method

This method converts the value of a Boolean to its equivalent string representation (either "True" or "False").

Eg: ToString()

Conversion methods of Data Types

Convert.ToBoolean Method

This method converts a specified value to an equivalent Boolean value.
For example, coveting an integer to an equivalent Boolean value.

Eg: Convert.ToBoolean(IntVar)

Conversion methods of Data Types

Convert Date and Time to String

This method converts the value of the specified DateTime object to its equivalent string representation.

Eg: DateTimeVar.ToString("dd-MM-yyyy")

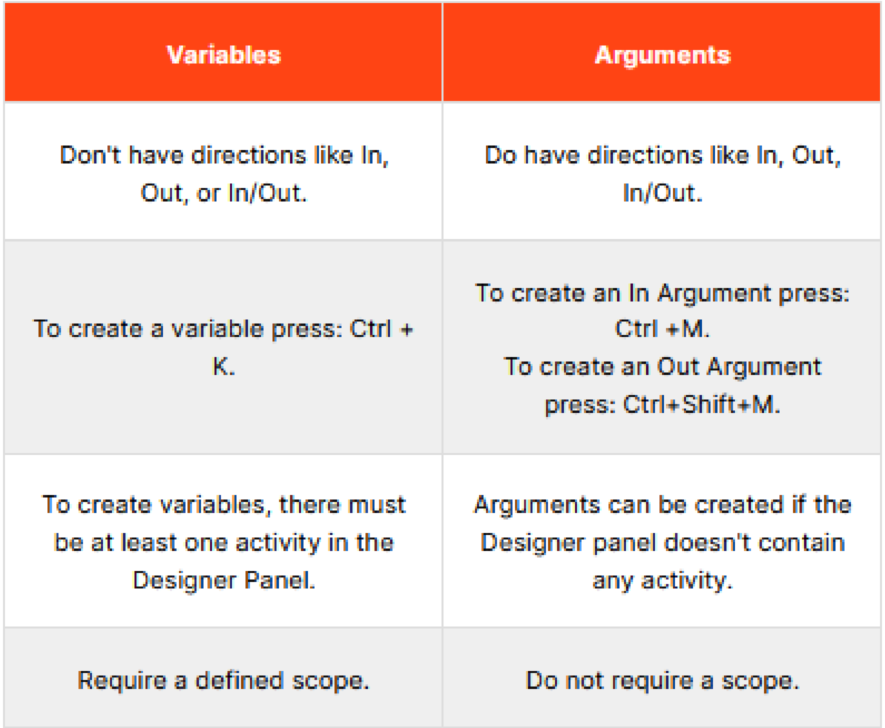
Resources

Data Types Conversion Methods - Microsoft Docs	https://learn.microsoft.com/en-us/dotnet/api/system.convert.toint32?view=net-6.0
Data Types - Microsoft Docs	https://learn.microsoft.com/en-us/dotnet/visual-basic/language-reference/data-types/

Invoke Workflow File activity and Arguments

Workflows are smaller blocks of automation, typically executing a specific part of an automation process, which can be used across different projects.

Arguments, like variables, are elements that store data. While variables pass data between activities within a workflow, arguments pass data between workflows.



Variables	Arguments
Don't have directions like In, Out, or In/Out.	Do have directions like In, Out, In/Out.
To create a variable press: Ctrl + K.	To create an In Argument press: Ctrl +M. To create an Out Argument press: Ctrl+Shift+M.
To create variables, there must be at least one activity in the Designer Panel.	Arguments can be created if the Designer panel doesn't contain any activity.
Require a defined scope.	Do not require a scope.

Arguments vs. Variables

Arguments	Variables
Argument stores data and passes it between workflows	Variable stores data and passes it between activities
It can be used across multiple workflows (direction to be defined)	It is limited to the workflow in which it is defined
It is created & modified through the Arguments Panel	It is created & modified through the Variables Panel
It is defined by properties: Name, Direction, Type, Default Value	It is defined by properties: Name, Type, Scope, Default Value

Workflows

A workflow represents a relatively small piece of an automation project, typically executing a specific part of the process. Once built, it can be reused across different projects.

A workflow is made of Studio activities, interconnected through variables to form a routine. The routine typically has an input and an output. Basically, it defines the flow of automation. Hence the name, workflow.

UiPath Studio provides you with **predefined workflow layouts** to suit all the needs of a fast and reliable automation process.

The workflow layouts are:

1. Sequences
2. Flowcharts
3. State Machines

The fastest, most reliable, and useful way of automating a process is to break it down into smaller bits. This allows for independent testing of components, enables team collaboration, and component reuse. Hence, most of the automation projects require the use of multiple workflows that come together to provide a solid business automation solution.)

Introduction to Arguments

Arguments are used to pass data from one workflow to another.

Enable users
to reuse workflows

Store data
dynamically

Useful in automation projects
with multiple workflows

Introduction to Argument Directions

Specify the direction from/to which the data is passed.

01

In

- Argument can only be used within a given project

02

Out

- Argument can be used to pass data outside a given project

03

In/Out

- Argument can be used both within and outside a project

Arguments

While **variables pass data between activities**, **arguments pass data between workflows**.

An additional property associated with arguments is the direction.

Arguments have specific directions: **In, Out, and In/Out**.

This tells the Robot where the information stored in them is supposed to go.

Arguments are key components when it comes to building more complex automation where you need to store and use data between multiple workflows.

Properties of Arguments

Users can configure arguments through their properties:

01 **Name**
Name of the argument

02 **Direction**
Direction from/to which the data is passed

03 **Type**
Kind of data that the argument is intended to store

04 **Value**
Data that an argument holds

UiPath Academy

Chapter 3:

Variables, Constants and Arguments in Studio

Invoke Workflow File and Arguments

Creating arguments – Where?

We can create arguments from Studio in four ways

1. From the Body of an activity
2. From the Properties Panel
3. From the Arguments Panel
4. From the Data Manager Panel

Creating arguments

1. From the Body of an activity

Directly from the activity panel in an activity input field:

- Right-click a field and select Create In Argument or Create Out Argument from the context menu. Alternatively, press Ctrl+M or Ctrl+Shift+M.
- The Set Arg field is displayed.
- Fill in the name and press Enter. An argument is created.
- Check its direction and type in the Arguments panel.

Alternatively, from the Expression Editor:

- Select a part of the expression and press Ctrl+M (for In arguments) or Ctrl+Shift+M (for Out arguments).
- Then Set Arg field is displayed.
- Fill in the name and press Enter. The argument is created.
- Check its direction and type in the Arguments panel.
- Arguments created in these ways automatically receive the type according to the activity.

Creating arguments

2. From the Properties Panel

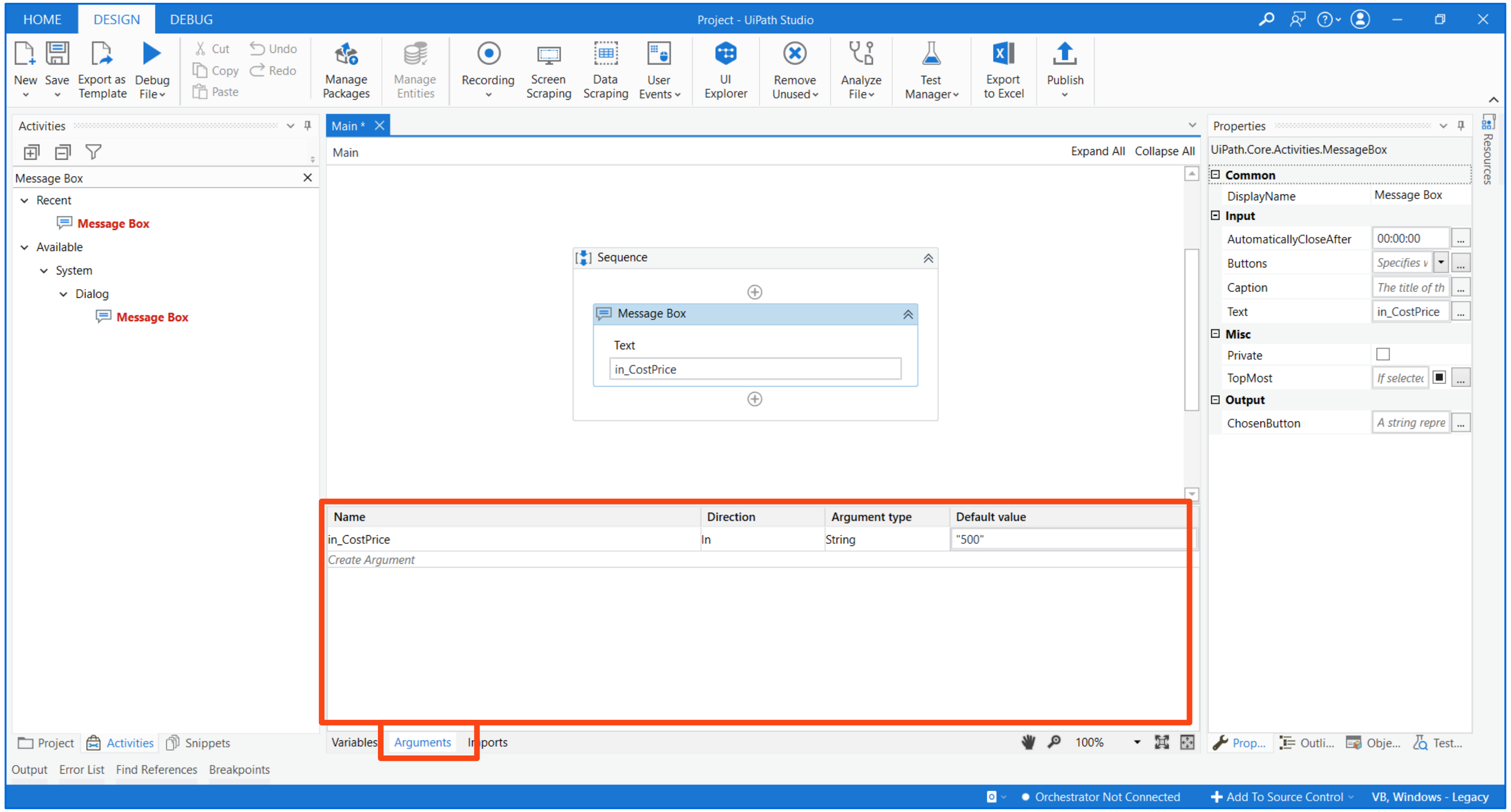
In the Properties panel of any activity:

- Right-click a field that can be edited.
- Select Create In Argument or Create Out Argument from the context menu. Alternatively, press Ctrl+M (In) or Ctrl+Shift+M (Out).
- The Set Arg field is displayed.
- Fill in the name and press Enter. The argument is created and visible in the field. Check its direction and type in the Arguments panel.

The argument type is automatically generated depending on the selected property.

Creating arguments

3. From the Arguments Panel



Text: in_CostPrice

Name	Direction	Argument type	Default value
in_CostPrice	In	String	"500"
Create Argument			

UiPath.Core.Activities.MessageBox

Common	
DisplayName	Message Box
Input	
AutomaticallyCloseAfter	00:00:00
Buttons	*Specifies v*
Caption	*The title of th*
Text	in_CostPrice
Misc	
Private	□
TopMost	*If selectec*
Output	
ChosenButton	*A string repre*

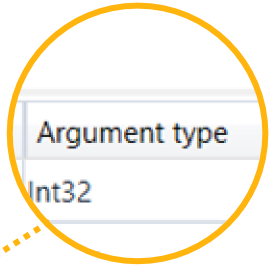
- Navigate to the Arguments panel.
- Select the Create Argument line.
- Fill in the name, direction, and type. A new argument is created.

Managing Arguments : Arguments Panel

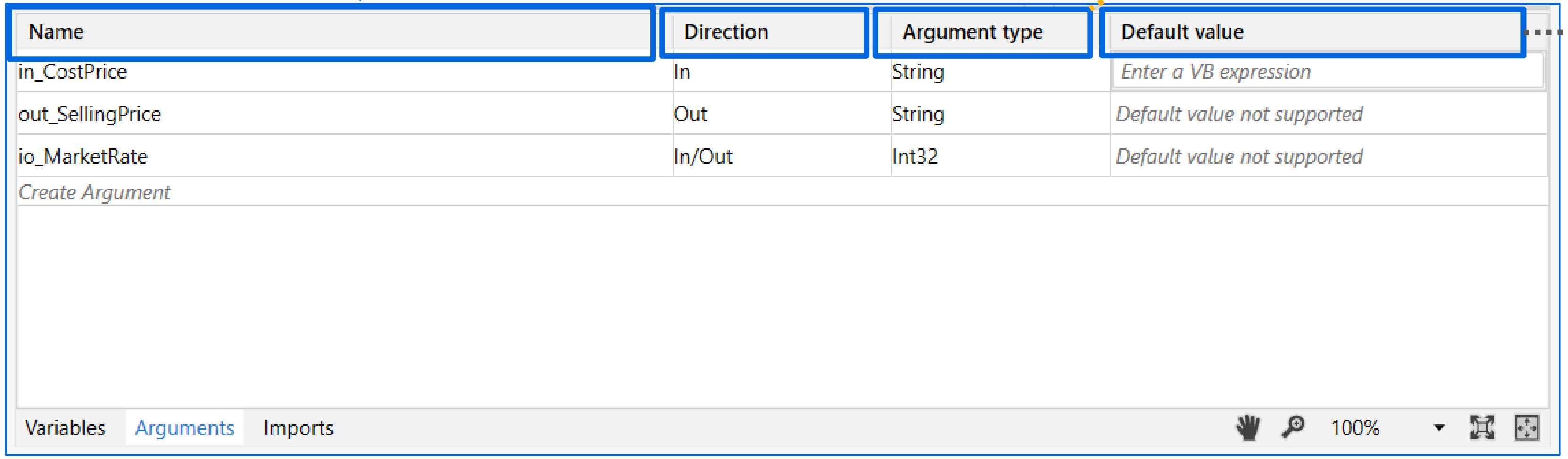
In Studio, a user can create arguments from the Arguments Panel through the ‘Create Argument’ option.

Step 01
Define a name

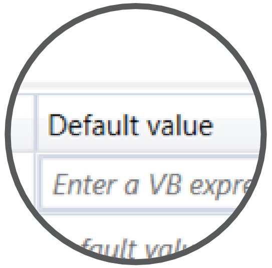
Step 02
Set the direction



Step 03
Specify the data type



Name	Direction	Argument type	Default value
in_CostPrice	In	String	*Enter a VB expression*
out_SellingPrice	Out	String	*Default value not supported*
io_MarketRate	In/Out	Int32	*Default value not supported*
Create Argument			



Step 04
Specify a default value

Creating arguments

4. From the Data Manager Panel

- Navigate to the Data Manager panel.
- Expand the Arguments options and click on New Argument.
- Fill in the name, direction, and data type. A new argument is created.

Best practices when working with arguments and workflow files

Assign meaningful names	Meaningful names should be assigned to workflow files and arguments in order to accurately describe their usage throughout the project.
Follow a naming convention	To improve readability, arguments names should also align to a naming convention. We recommend using the PascalCase naming convention. In this convention, the first letter of each word in a variable is capitalized.
Use prefixes in argument names	Argument names should have a prefix stating the argument type, such as in_DefaultTimeout, in_FileName, out_TextResult, io_RetryNumber.
Use action verbs in workflow names	Except for Main, all workflow names should contain the verb describing what the workflow does, such as GetTransactionData, ProcessTransaction, TakeScreenshot.

Resources

Invoke Workflow File	https://docs.uipath.com/activities/other/latest/user-guide/invoke-workflow-file
About Automation Projects	https://docs.uipath.com/studio/standalone/2022.10/user-guide/about-automation-projects
Managing Arguments	https://docs.uipath.com/studio/standalone/2022.10/user-guide/managing-arguments
Using Arguments	https://docs.uipath.com/studio/standalone/2022.10/user-guide/using-arguments

Classroom Exercise : Invoke Workflow Activity

Build a simple interest calculator using different workflows and arguments. Steps can be broken down as follows:

1. In the Main workflow, ask the user to enter the initial deposit amount and select the period using multiple-choice option. The period can be either 1, 3, or 5 years. Store the input in two variables. Create a third variable that will store the value of the final earnings.

2. Create a new workflow to calculate the simple interest using the user inputs. The formula to calculate the interest can be Deposit Amount * Rate per year * Period chosen / 100. Set the Rate per year to 1.75 for all deposit periods.

3. Pass the result back to Main.xaml and display on-screen the accumulated interest at the end of the period and the final deposit balance.

4. Output: display a message box with the accumulated interest at the end of the period and the final deposit balance.

Global constants and global variables

What are global constants and variables?

Just like variables, global constants and variables are containers that store data.
But unlike variables, these store data in a central location which can be accessed by all parts of an automation project.

How are they useful?

It is sometimes necessary to share data between workflows in the same project.
With global constants and variables, this is possible without having to pass data as an argument.

Global constants and variables allow for greater flexibility as they can be easily accessed and modified as needed.
This can help save time and simplify your automations.

Global constants and global variables

Global constants and variables allow us to store data in a central location which can be accessed by all parts of an automation project.

This allows for easier data management.

Global constants and variables **can be created only from the Data Manger panel**.

The difference between a global constant and a global variable is that none of the properties of a global constant can be altered whereas the value of a global variable can be modified during the program execution.

UiPath™ Academy

Chapter 3:

Variables, Constants and Arguments in Studio

Using Global Constants and Global Variables

Resources

Global Variables and Constants- UiPath Studio Guide	https://docs.uipath.com/studio/standalone/2022.10/user-guide/release-notes-2022-10-3
Managing Variables - UiPath Studio Guide	https://docs.uipath.com/studio/standalone/2022.10/user-guide/managing-variables
UiForum: Use cases of global constants and global variables	https://forum.uipath.com/t/understanding-and-use-case-of-global-variables-and-constants/482631
UiForum: Global Variables and Constants	https://forum.uipath.com/t/global-variable-constants-latest-feature-in-2022-10-version/490513

Collection and Its Types

A collection is used to represent a set of similar data type items in a single unit which is used for grouping and managing the objects. Different types of collections are:

A

Arrays

L

Lists

D

Dictionaries

Collection Manipulation

Studio offers the following methods for manipulating collections:

Add to Collection

- Adds an item to a specified collection
- Example: It can be used to add a new name to a list of company names

Remove From Collection

- Removes an item from a specified collection and can output a Boolean variable that confirms the success of the removal operation
- Example: It can be used to remove an invoice number from a list of invoices to be processed

Exists In Collection

- Indicates whether a given item is present in a given collection by outputting a Boolean as the result
- Example: It can be used to check whether a list of clients contains a specific name

Clear Collection

- Clears a specified collection of all items
- Example: It can be used to empty a collection before starting a new phase of a process that will populate it again

Arrays

Arrays are of **fixed length**

Definition

An Array variable is a collection that stores multiple elements of the same data type

Usage

Used to organize data so that a related set of values can be easily sorted or searched

Example

- An array of names of two items
  - Variable name can be ArrCompName with value {“ABC”, “XYZ”}

Array - ArrayOf<T> or System.DataType[] stores multiple values of the same data type. The size (number of objects) is defined at creation

Array – Business Scenarios

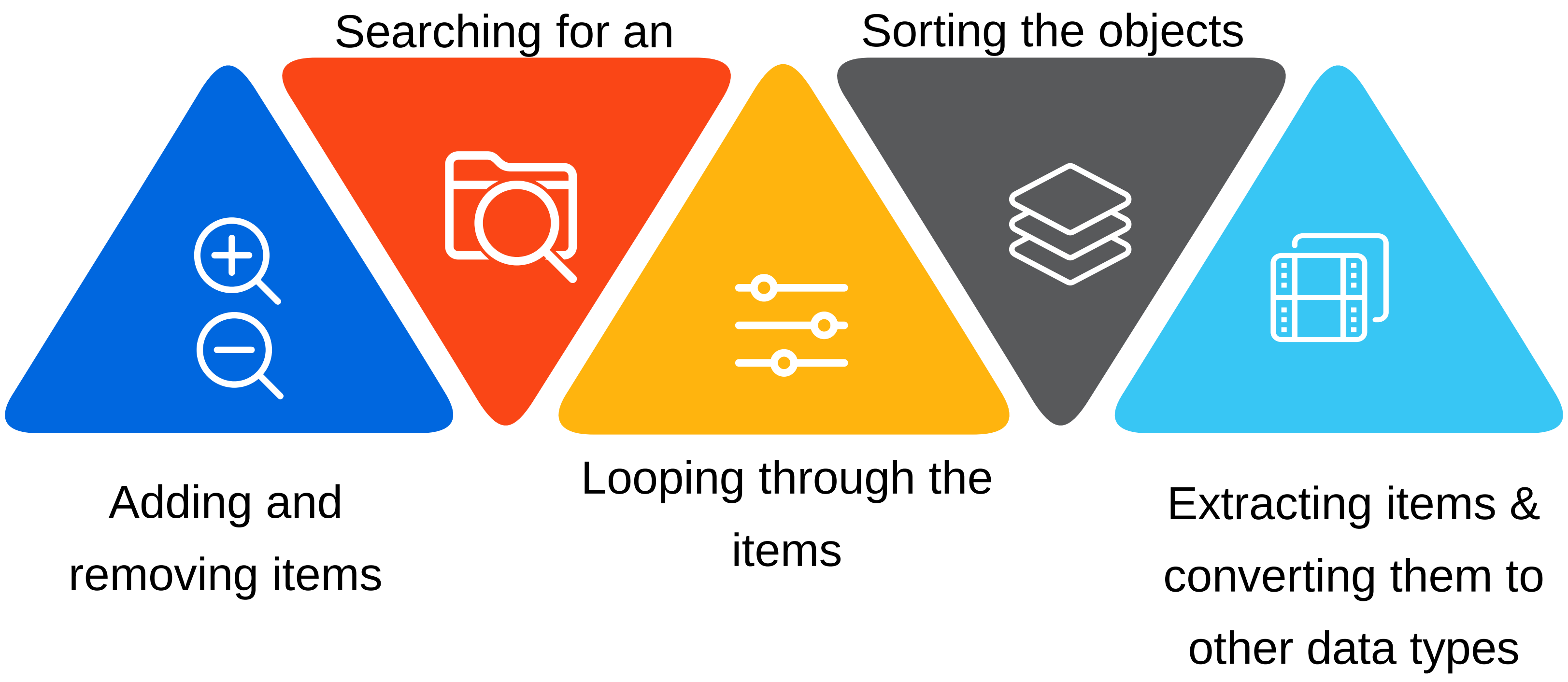
What are some business scenarios where arrays are useful?

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

Disclaimer: As a good case practice, **arrays** are used for **defined sets of data** (for example, the months of the year or a predefined list of files in a folder). Whenever the collection might require **size changes**, a **List** is probably the better option.

Lists

Lists are data structures consisting of objects of the same data type. Each object has a fixed position in the list. Lists provide specific methods of manipulation, such as:



Searching for an

Sorting the objects

Adding and removing items

Looping through the items

Extracting items & converting them to other data types

List - System.Collections.Generic.List<T> stores multiple values of the same data type, just like Arrays. But unlike Arrays, the size is dynamic

Dictionaries

Dictionaries are collections of (key, value) pairs in which the keys are unique. The data types for both keys and values are chosen when the variable is initialized. Operations on dictionaries include:

Retrieving the value associated

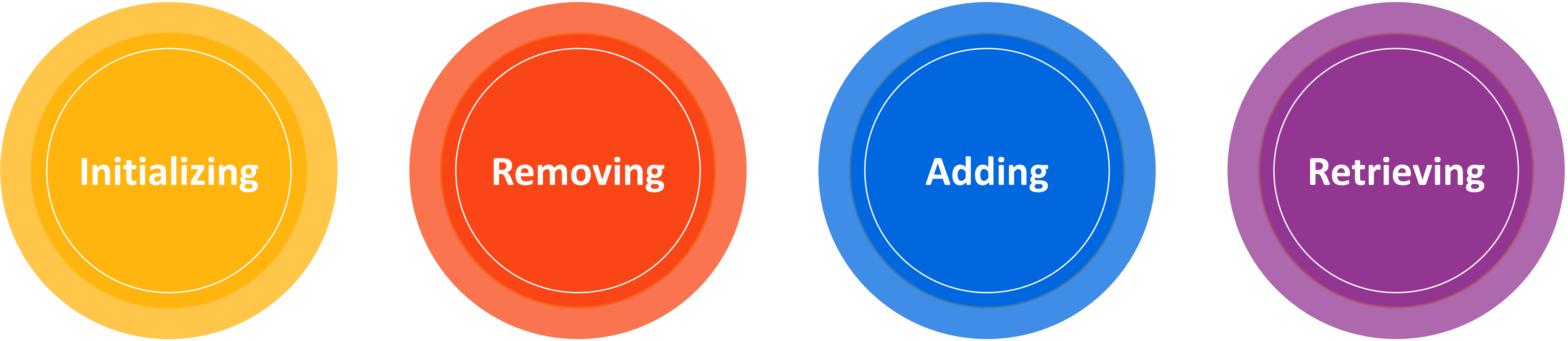
Adding and deleting (key, value) pairs

Re-assigning new values to existing keys

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

Working with Dictionaries

Dictionaries in Studio can be used in the following ways:



Initializing

Removing

Adding

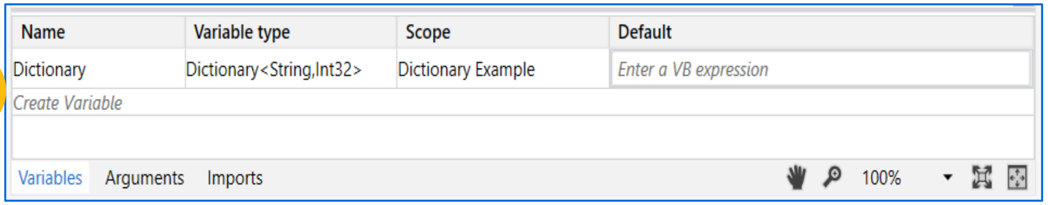
Retrieving

Methods for Working with Dictionaries

Initializing

1. Create a variable of type Dictionary in the Variables panel
2. Drag and drop an Assign activity in the Designer panel
   - Enter Dictionary in the *To* textbox.
   - Initialize the dictionary variable in the *Value* textbox using **new Dictionary (Of String, Int32) From {{key, value}, {key, value}}**, if the key is String and the value is Integer
3. Insert a Message Box activity to display the added key and value pair
4. Execute the workflow. The Value of the first Key displays in the message box

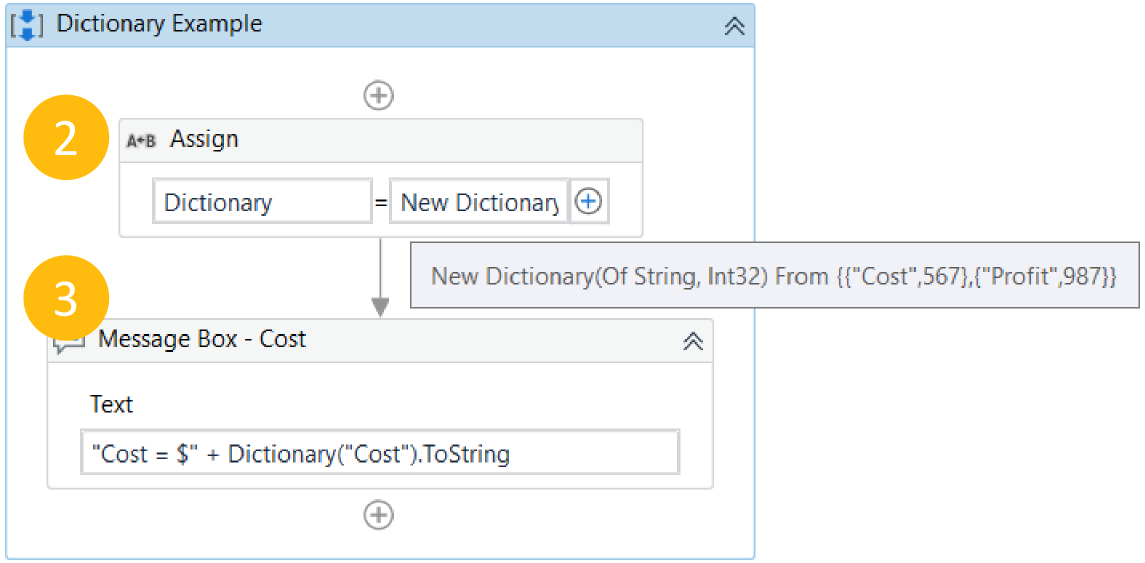
1



Name	Variable type	Scope	Default
Dictionary	Dictionary<String,Int32>	Dictionary Example	*Enter a VB expression*
Create Variable			

2

3



Dictionary Example

Assign

Dictionary = New Dictionary

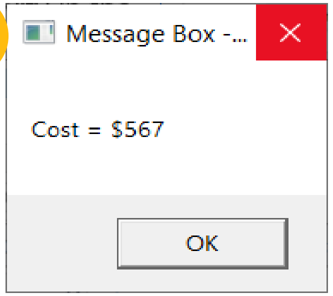
New Dictionary(Of String, Int32) From {{"Cost",567},{"Profit",987}}

Message Box - Cost

Text

"Cost = $" + Dictionary("Cost").ToString

4



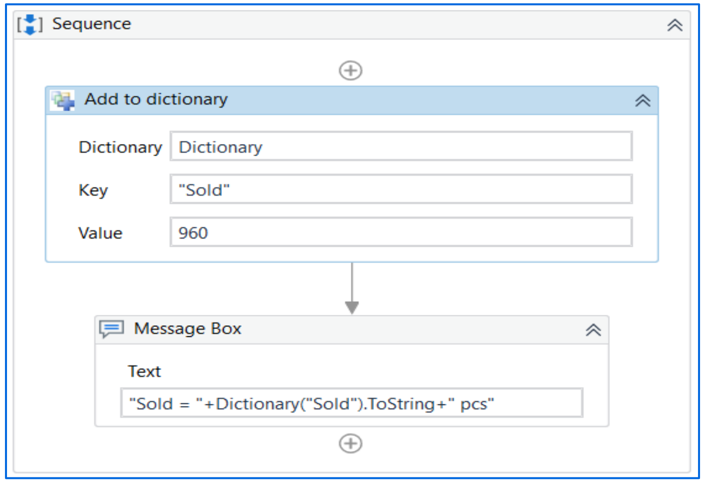
Message Box -...

Cost = $567

Methods for Working with Dictionaries (Contd.)

Adding

- Install the package Microsoft.Activities.Extension.
- Drag and drop the Add to Dictionary activity
- Enter the dictionary variable, key, and value in their respective fields



Dictionary: Dictionary

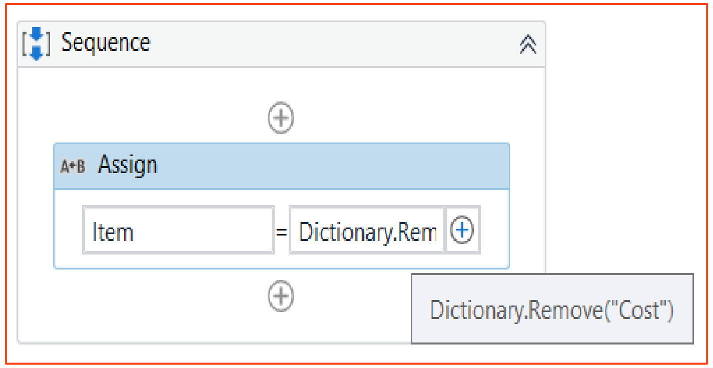
Key: "Sold"

Value: 960

Text: "Sold = "+Dictionary("Sold").ToString+" pcs"

Removing

- VarName.Remove(Key) – removes an item from the Dictionary. It can be used in an ‘Assign’ activity

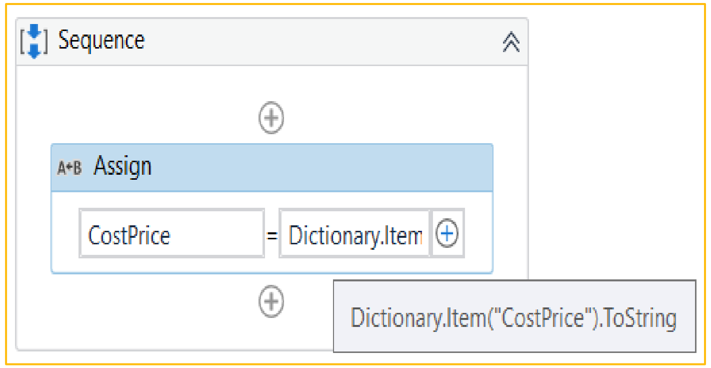


Item = Dictionary.Rem

Dictionary.Remove("Cost")

Retrieving

- VarName.Item(Key) – returns the Dictionary item by its key



CostPrice = Dictionary.Item

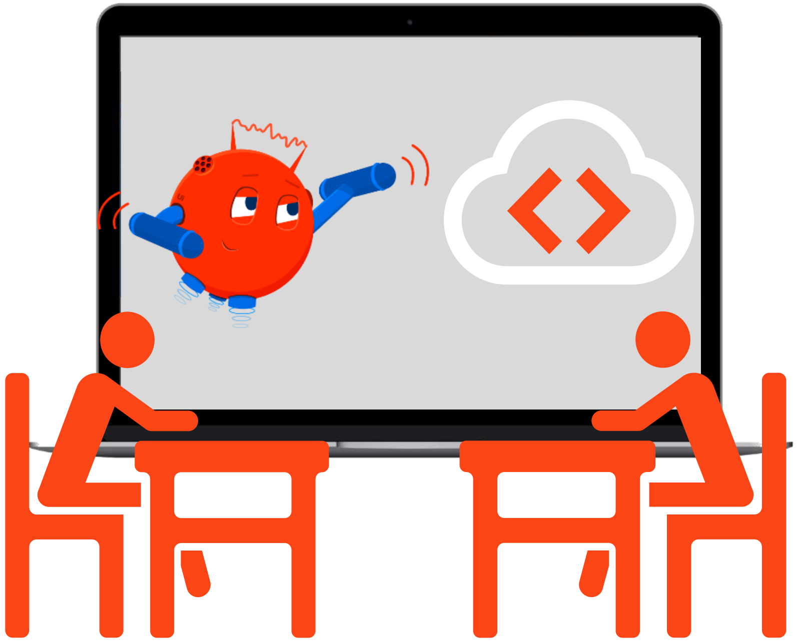
Dictionary.Item("CostPrice").ToString

Classroom Exercise - Lists

Sort a list in reverse order and print the first five items from the list in a message box.

1. Create a list of ten names of people
2. Sort the names in reverse alphabetical order from Z to A
3. Extract the first five names from the list
4. Display the extracted names in a message box

Practice Exercise - Lists



Build a workflow using the **Concat** and **Join** method that merges two lists containing the city names of the UK and Spain, sorts it, capitalizes the first letter of each item, and displays it in a message box.

1. Create a list containing three cities of the UK in Uppercase
2. Create another list containing three cities of Spain in Lowercase
3. Merge both the lists together
4. Sort the final list in alphabetical order from A to Z
5. Capitalize only the first letter of all the items in the final list
6. Display the final list in a message box in string format

UiPath™ Academy

Chapter 3:

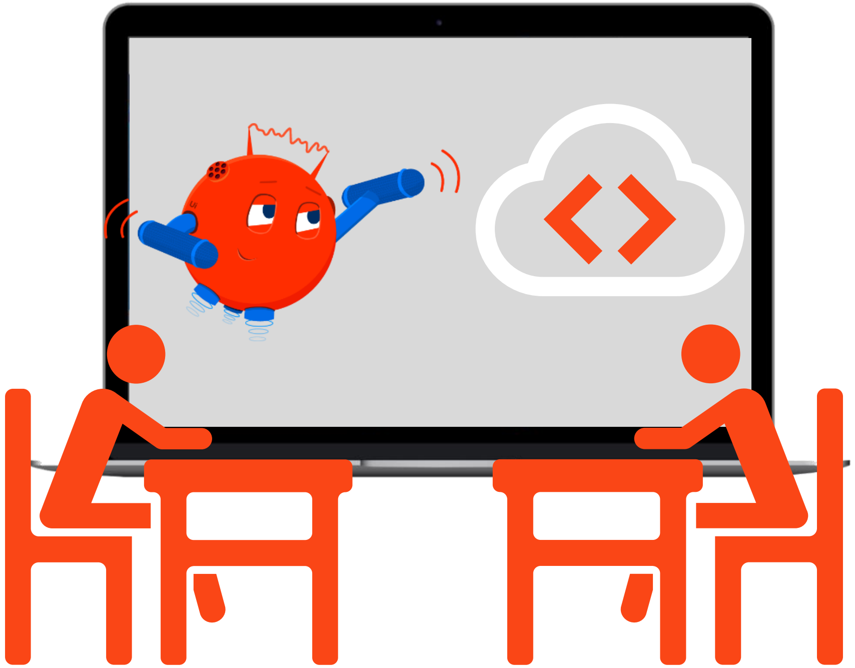
Variables, Constants and Arguments in Studio

Array Type Variables

Resources

Array Variables - UiPath Studio Guide	https://docs.uipath.com/studio/standalone/2022.4/user-guide/array-variables

Classroom Exercise - Arrays



Build a simple process that simulates a flight reservation for users

1. You must request input from the user regarding the destination, departure date, and return date.
2. Then add the user input to an array variable.
3. Lastly, write a text file where you confirm the reservation details using the array variable.
4. Output: Write a text file using the data from the array variable to confirm the user's reservation.