Module
https://learn.microsoft.com/en-us/training/modules/use-imperative-dev-techniques-powerapps-canvas-app/1-imperative-vs-declarative

Unit 1 Imperative versus declarative development

- In **imperative** development, the focus is on **how to** achieve the goal.
    - Imperative provides more **flexibility** because you **control** every step in the process, but that means more **code** and more **complexity**.
    - You focus on creating the sandwich in your "code". You go to the kitchen, get the ingredients, put the sandwich together, and then send it to the user. You spend a lot of time on the steps, but you have all of the specific functions you want to make it exactly the way you want. No tomatoes? No problem.
    - This is the approach you will see in languages like **C#** or other popular coding languages
- With **declarative**, the focus is on getting the **result**.
    - Declarative is much **simpler** and **straightforward** to use but can **lack** the ability to have the complete control that you might want.
    - The difference is now you are focused on **producing** the sandwich, **not how to** make a sandwich. This is much **less complex**, but you might also run into the issue with tomatoes. If the function you use to get the sandwich **doesn't support** the option of no tomato you are out of luck. Your code may be as straightforward as follows.
        - GetSandwich(Kitchen, Mouth)
    - Low-code tools like **Excel** use this approach to development. The focus is on **pulling data**.
- Power Apps has capabilities for both imperative and declarative logic

Unit 2 The three types of variables in Power Apps

- Variables are a key driver for imperative logic in Power Apps because they allow you to "build the sandwich" piece by piece.
- Power Apps has three different types of variables
    - **Global variables** -- The most traditional type of variable. You use the **Set function** to **create** and **set** its **value**
        - Common use is to store a **user's DisplayName** when the app loads
    - **Context variables** -- A context variable is **only available on the screen** where you create it using the **UpdateContext function**.
        - One example: used for functionality that controls a **pop-up screen**
    - **Collections** -- A collection is a special type of variable for **storing a table of data**
        - Collections are available throughout your app, like global variables, and they are created using the **Collect** or **ClearCollect** function
- **Variables** are **temporary** and only available to the current user in their **current session**

Unit 3 Global variables

- With Power Apps, you can show a welcome message and get the user's name in a declarative manner by using the following **formula in a Label control**.
    - **"Welcome " & User().FullName**
- A better approach would be to store that information in a **global variable** when the app opens, and then reference that variable throughout your app.
    - You could do this by Modifying the **OnStart** property of the app with the following formula.
    - **Set(varUserDisplayName, User().FullName)**
- Now for your **Label control**, you would change the formula to the following.
    - **"Welcome " & varUserDisplayName**
- With a declarative mindset, you would set the Visible property for the warning icon to the following.
    - **CountRows(Filter(InvoiceEntity, CustomerNumber = ThisCustomersNumber And Status = "Outstanding")) > 3**
- A better approach is only to run the complex call once, store the result in a variable, and then use that variable to control the **Visible** property of each control.
    - To do this, configure the **OnVisible** property of the screen to set the variable.
    - **Set(varOustandingExceeded, CountRows(Filter(InvoiceEntity, CustomerNumber = ThisCustomersNumber And Status = "Outstanding")) > 3)**

Unit 4 Contextual variables

- Many apps use pop-up dialog boxes to confirm things like deleting a record.
    - A common way to implement this is to set a Contextual variable to true when the user selects the delete button.
    - You do that by setting the **OnSelect** property of the button to the following.
    - **UpdateContext({varShowPopUp: true})**
    - You then set the **Visible** property of the pop-up controls to **varShowPopUp**
- If you copy the controls (using Ctrl+C) to another screen, then you will have two instances of **varShowPopUp**.
    - These two instances use the same name, but can have different values.
- One unique behavior of the **UpdateContext** function is that you can declare more than one variable at a time
    - **UpdateContext({varCount: 1, varActive: true, varName: User().FullName})**
- To do the same thing with Global variables, you would use the following.
    - **Set(varCount, 1);Set(varActive, true);Set(varName, User().FullName)**

Unit 5 Collections

- The most common reason for using collections is to optimize performance by reducing calls to the same table in a data source
- To store a copy of the **Projects** table from your data source into a collection called **collectProjects**
    - **Collect(collectProjects, Projects)**
- Considerations that you need to understand about using collections

    - The **Collect** function is **not delegable**. This means by default **only the first 500 records** from the data source will be retrieved and stored in the collection
    - Changes to the data in the collection **do not automatically save** to the data source
    - When you **close** the **app**, the collection and all of its **contents are removed**
- You can also **create a collection from** information directly **within your app**
    - This is often done to **provide values for a drop-down** menu or combo box and to store large amounts of data before writing to a data source.
    - **Collect(collectColors, {Name: "Shane", FavoriteColor: "Orange"}, {Name: "Mary", FavoriteColor: "Blue"}, {Name: "Oscar", FavoriteColor: "Yellow"})**
- The one exception where **Collections** vary from tabular data sources is you cannot use them with the **Form control**.

Unit 6 Additional variable concepts

- Sometimes you need to make a **variable that points to itself**
    - This is often done when you want to either do a **counter type variable** where it **increments** a value or you are **appending a string**
    - Place the following formula on the **OnSelect** property of a button to set up a counter.
        - **Set(varCounter, varCounter + 1)**
    - Next to the button put a **Label**, and in the **Text** property, put **varCounter**
- The default value of a variable will vary based on the variable type if you do not set the default property.
    - Text variables are ""
    - Number variables are 0
    - Boolean variables are false
- You can also **store a record in the variable**
    - **Set(varUser, User())**
    - You can **retrieve** the values of the individual columns using the **dot (.) notation**
- Variables don't automatically update
    - For example, they can use a **variable to store the number of customer invoices** using **OnStart** for the app.
    - Then in the app, the **user creates** a new invoice. The **variable doesn't distinguish** the number of **invoices** in the system **that have changed**

Unit 7 Exercise - Using the variables and collections

- Set the **OnStart** property to this formula that we're using to create a collection of customer numbers and the associated invoice numbers
    - **ClearCollect(colCustomer,**
    - **{CustomerNumber: 7470, InvoiceNumber: "INV70817"},**
    - **{CustomerNumber: 4259, InvoiceNumber: "INV29595"},**
    - **{CustomerNumber: 8251, InvoiceNumber: "INV74302"},**
    - **{CustomerNumber: 2338, InvoiceNumber: "INV35115"},**
    - **{CustomerNumber: 1524, InvoiceNumber: "INV82337"},**

    - **{CustomerNumber: 1530, InvoiceNumber: "INV82338"}**
    - **);**
    - **Set(varUserDisplayName, User().FullName)**
- To view the collection, select the **Insert** tab (or + Insert from the command bar) and add a **Vertical gallery**, set its Items property to **colCustomer**
- Select the Trash icon and the below formula to its **OnSelect** property:
    - **Remove(colCustomer,ThisItem)**
- Press and hold **Alt Key**, and select the first Trash can icon.
    - **This deletes the selected row from the collection**. While this process works, you probably want to give a warning message that the row will be deleted.
- By employing a Contextual variable, let's **create a popup warning** to inform the user about the pending deletion and give them an option to cancel.
    - Change the **OnSelect** of the Trash can icon to the below:
        - **UpdateContext({varPopup:true})**
- Set the lbl_popup **Text** to the below formula:
    - **varUserDisplayName & " Please click Delete to confirm deletion of Invoice#" & " " & Gallery1.Selected.InvoiceNumber**
- Set the Delete Button **OnSelect** property to the below:
    - **Remove(colCustomer,LookUp(colCustomer,CustomerNumber=Gallery1.Selected.CustomerNumber));**
    - **UpdateContext({varPopup:false})**
- Set the Cancel Button **OnSelect** property to:
    - **UpdateContext({varPopup:false})**
- Now let's select all of these controls together so we can **group** them
    - Set the **Visible** property of Group1 to: varPopup

Unit 8 Module assessment

- 1. You are designing a Power App with a feature that requires toggling a label's visibility based on user interactions on the same screen. Which type of variable should you use?
    - **Context variable**
    - Global variable
    - Collection
- 2. A Power App user reports that a pop-up dialog box appears unexpectedly when navigating between screens. What could be causing this issue?
    - A collection is being used to store pop-up state
    - A context variable is incorrectly scoped.
    - **A global variable is controlling the pop-up visibility.**
- 3. An application needs to **dynamically change its behavior based on user actions** in Power Apps. Which development methodology should be prioritized for this requirement?
    - Declarative development methodology
    - Test-driven development methodology
    - **Imperative development methodology**

- 4. You are tasked with reducing network traffic in a Power App by minimizing repetitive data source queries. Which approach would you take?
    - Rely on context variables for all data operations.
    - **Store frequently accessed data in collections.**
    - Use global variables for storing all dynamic data.
- 5. A developer needs to store a list of items temporarily within a screen in Power Apps. What type of variable should be used?
    - Global variable
    - **Context variable**
    - Collection variable
- 6. A developer chooses to write detailed instructions for app functionality, while another developer uses a simple formula to achieve the same functionality. What does this choice illustrate about development methodologies in Power Apps?
    - Both developers are using declarative approaches.
    - Both developers are using imperative approaches.
    - **The first developer is using an imperative approach, while the second is using a declarative approach.**
- 7. Which type of variable should be used to store and manage a table of data that is frequently updated within the app?
    - **Collection variable**
    - Context variable
    - Global variable
- 8. Which characteristic is unique to declarative development methodologies in Power Apps?
    - **Focus on the end result rather than the steps to achieve it**
    - Ability to control each step of the process
    - Use of complex programming languages like C#
- 9. A developer needs to create a pop-up confirmation dialog that warns users before deleting a record. Which type of variable should be used to manage this pop-up visibility?
    - Global variable
    - Collection variable
    - **Context variable**