

**Introduction**

For data engineers, notebooks are a highly important tool, particularly in settings like Microsoft Fabric. They enable interactive code writing and execution, but what if you require your notebook operate dynamically with shifting inputs? Parameterization can help with that! A concept similar to programming functions receiving parameters. The fundamentals of parameters, their importance, and an efficient implementation of parameterization in Fabric will all be covered in this article.

**What Are Parameters?**

Similar to placeholders, parameters allow you to enter data into a notebook or your function. Consider them as variables whose values have the ability to vary based on the requirements and work at hand. For instance, in order to make a function reusable, we frequently define parameters while writing it in programming:

**def** greet(name):

**return** f"Hello, {name}!"

greet("Ahmed")

**# Output**: Hello, Ahmed!

def calculate\_area(length, width):

return length \* width

length = 5

width = 10

calculate\_area(length, width)

**# Output: 50**

In the same way, notebooks can operate dynamically by accepting parameters. To make the notebook reusable, for example, you send in a file location or date as inputs rather of hardcoding them.

**What Is Parameterization?**

Parameterization is the process of designing your function or a notebook so it can accept these parameters as inputs. Instead of editing the notebook every time your requirements change, you simply pass the relevant parameter values, making your notebooks flexible and reusable.

Parameterization in notebooks works just like it does in functions. By defining placeholders for inputs, you can provide values dynamically when executing the notebook.

**Importance of Parameterization**

Why is parameterization such a game-changer?

**1. Reusability**

With parameterization, you may easily utilize the same notebook or function for several tasks by just changing the inputs. This reduces redundancy and streamlines operations by doing away with the need to duplicate code or notebooks.

**2. Flexibility in Functions**

With parameters, a single function can dynamically handle various inputs, enabling adaptive and versatile solutions.  
**Example:**

def greet(name):

return f"Hello, {name}!"

print(greet("Ahmed")) # Output: Hello, Ahmed

print(greet("Abdullah")) # Output: Hello, Abdullah

**3. Modularity**

Numerous stages are frequently involved in complex workflows. You can divide these procedures into more manageable chunks by parameterizing notebooks. By exchanging information and context through parameters, these modular notebooks can improve readability, maintainability, and teamwork.

**4. Scalability**

For automation and managing massive amounts of data, parameterization is especially important. Parameters allow you to easily adjust, saving time and resources when processing various datasets or executing comparable logic across several settings.

**5. Dynamic Configurations**

Configurations like file locations, database connections, and processing settings can be changed at runtime by using parameterization. Due to this, your notebooks may be deployed in many environments without affecting their basic logic, making them environment-independent.

**6. Enhanced Automation**

Automating repetitious processes is made easier by parameterization. Dynamic input handling allows you to efficiently analyze huge quantities of data or carry out iterative logic by looping over numerous parameters.

**Types of Notebook Parameterization in Fabric**

When working with parameterized notebooks in Fabric, the process varies slightly depending on the location of the notebooks—whether they are in the root directory or in subfolders. Let’s explore both types.

**1. Root Notebook Parameterization**

In this scenario, the notebook you want to call is located in the root directory of your workspace. You can pass parameters to it directly by specifying the notebook's name and the required parameters. Here's how you can do it:

* Use the notebook.run() function to call the root-level notebook.
* Pass the parameters as a dictionary or a list, depending on your implementation.

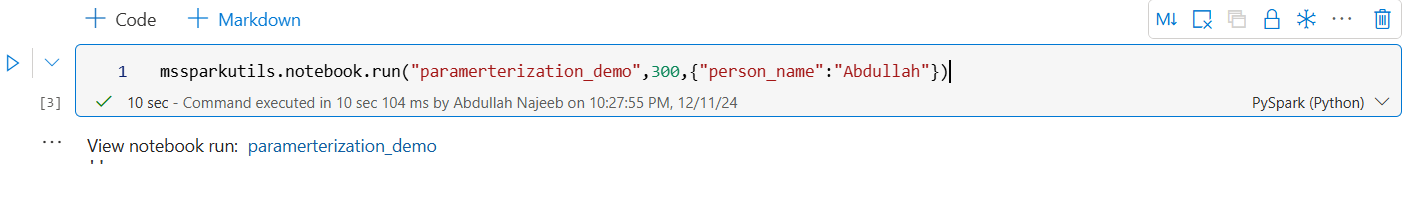
The notebook where you want to pass the parameters should be triggered from another notebook. When triggering it, the parameters will be passed along.

For example, let’s assume there’s a function named greet inside a notebook called parameterization\_demo. This function requires a name to be passed as a parameter. Here's how you would set it up:

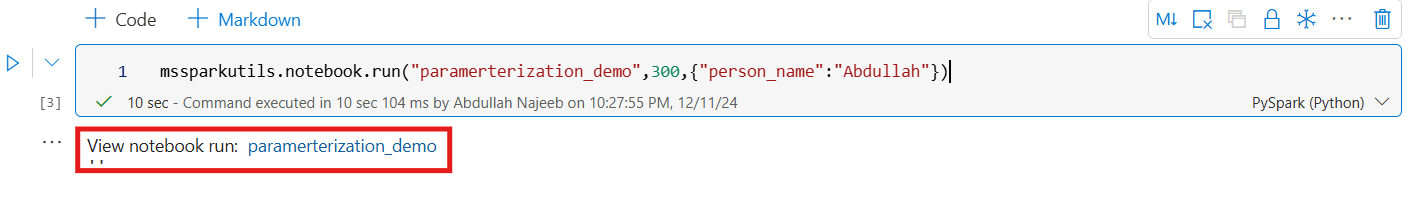
**Passing Single Parameter:**

Step 1: Create a query to trigger the notebook where the parameter is meant to be passed, such as parameterization\_demo.  
Step 2: Passing the parameter while triggering the notebook. For example, person\_name is a parameter with the value 'Abdullah'.

Step 3: Run the cell.

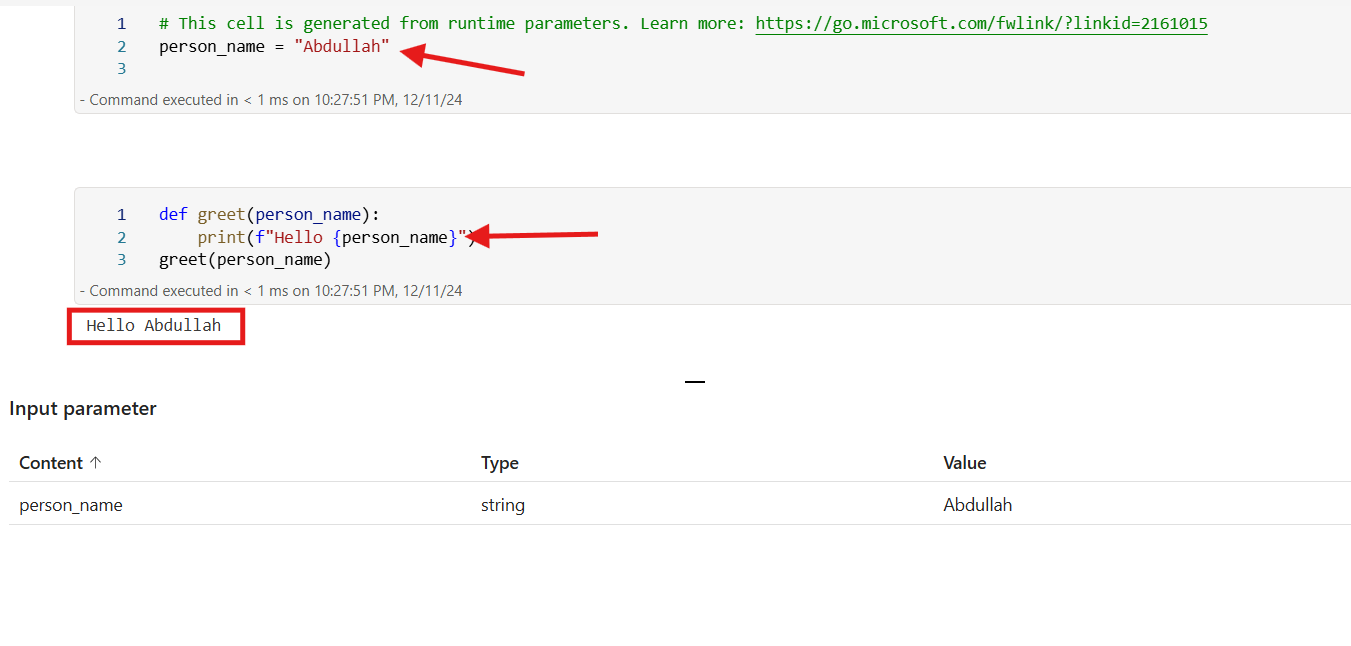


Now, the notebook has ran and we can check the output by clicking at the notebooks name below the cell:



This will show the complete sequence of the notebook run.

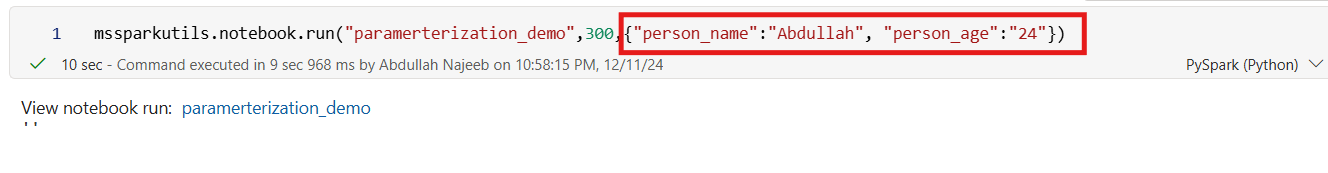
* A new cell is added at the top to assign a parameter value to a variable, which can be used wherever it's needed.

\

Passing Multiple Parameters:

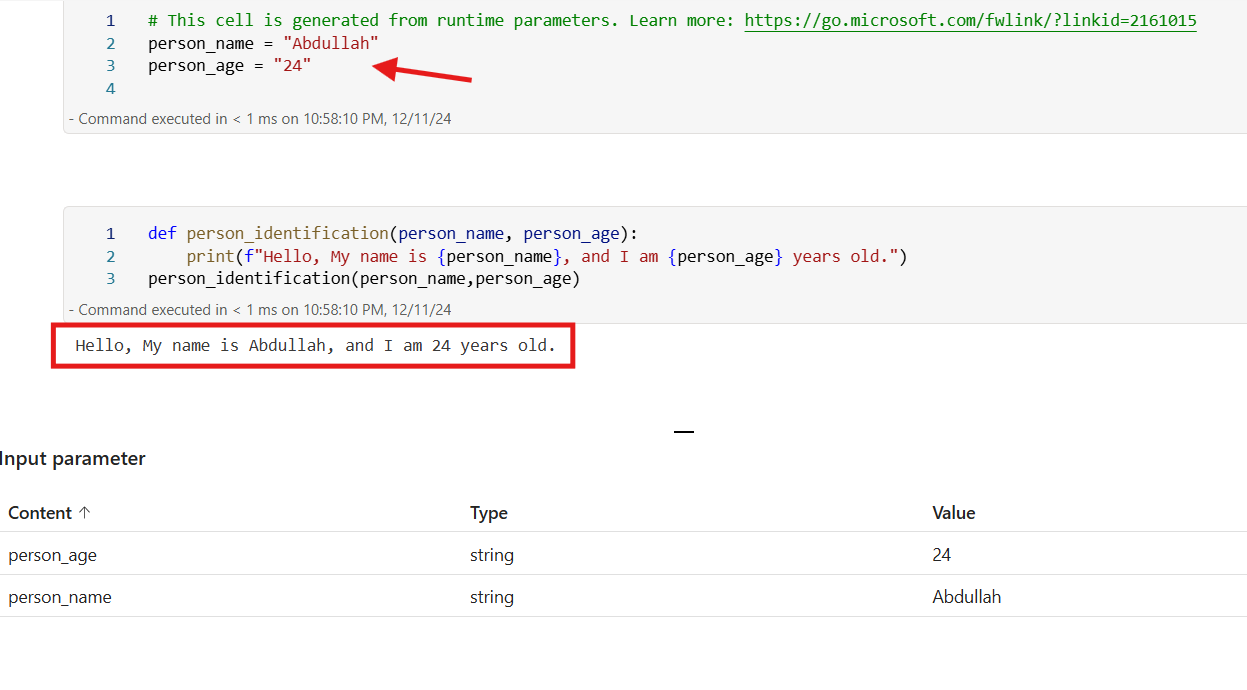
Step 1:

You can pass multiple parameters at the same time. Follow the same steps as in the "Passing Single Parameter" section. Instead of passing one parameter, pass multiple like this:



Output:

After clicking the notebook’s name, it will show a new cell which is added at the top to assign a parameter value to a variable, which can be used wherever it's needed.



**Parameterizing Subfolder Notebooks in Microsoft Fabric: Why useRootDefaultLakehouse is Key**

When working with parameterized notebooks in Microsoft Fabric, you may observe a key difference in how notebooks are triggered based on their storage location.

* **Root-Level Notebooks:** Notebooks stored in the root folder can be triggered directly, without requiring any extra configuration.
* **Subfolder Notebooks:** Notebooks in subfolders, however, often throw errors unless you include the command useRootDefaultLakehouse.

**Why Doesn’t Relative or Absolute Path Work for Subfolder Notebooks?**

Fabric automatically assumes the default lakehouse context for root-level notebooks, making them easy to locate and execute. For subfolder notebooks, however, this assumption doesn’t hold. Even when you provide a relative or absolute path, the system may struggle to locate the notebook.

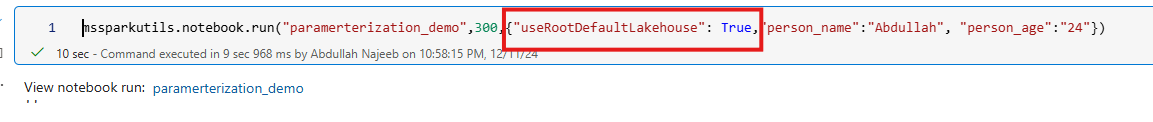
This is because Fabric’s execution environment needs a clear starting point—the lakehouse context—to resolve file paths properly. Subfolder notebooks require explicit instructions to set the root lakehouse as the starting context, which is achieved using the useRootDefaultLakehouse command.

**What Does useRootDefaultLakehouse Do?**

The useRootDefaultLakehouse command aligns the workspace to the root lakehouse context. This alignment ensures that Fabric can correctly interpret paths, enabling it to locate and execute notebooks stored in subfolders. Without this command, the system may fail to traverse the file structure effectively, leading to errors.

**How to Parameterize Subfolder Notebooks**

The process for parameterizing notebooks in subfolders is nearly identical to root-level notebooks. The only additional step is to include the useRootDefaultLakehouse command before triggering the notebook. This ensures that the correct lakehouse context is set for locating the subfolder notebook.

 This ensures that the notebook in the subfolder is correctly located and executed. All other aspects, such as defining parameters and passing dynamic values, remain the same. By incorporating this one extra step, you can effortlessly work with parameterized notebooks, regardless of whether they are in the root folder or subfolders within Fabric.