SNOWPARK FOR PYTHON end to end

Snowpark is a developer framework built inside of Snowflake. It allows developers to use their favorite programming languages (Python, Java, Scala) to interact directly with Snowflake's cloud-based data platform.

We will focus on using Python, a favorite among data professionals for its simplicity and powerful libraries.

Snowpark's integration with Python enables DataFrame-style programming right within Snowflake. This allows developers to analyze complex data, develop machine learning models, and build data-driven applications easily within the Snowflake environment, taking full advantage of its scalability and performance.

How Snowpark Works

Snowpark consists of both a client-side library and a server-side sandbox, enabling Python developers to write and execute code in a familiar environment while leveraging Snowflake's robust data processing capabilities.

Client-Side Library (The Python Coding Environment)

The client-side library is similar to a Python developer's toolkit on their own computers. This library contains all the necessary tools and commands for writing Python code in an environment they are already familiar with, just like they would in any standard Python environment.

Server-Side Sandbox (Snowflake's Secure Environment)

The server-side sandbox is a secure, isolated space Snowflake provides on its servers. When Python developers run their code, it's executed in this sandbox. This means that the heavy lifting, especially when dealing with large data sets or complex computations, is done on Snowflake's powerful servers rather than the developer's computer. This setup not only ensures faster processing and more efficient handling of data but also adds an extra layer of security, as the actual data processing happens within Snowflake's secure environment.

Key Features and Capabilities

Easy Data Handling with DataFrames: Snowpark simplifies data work using DataFrames, which represent Snowflake tables. Snowflake designed DataFrames to increase the efficiency of sorting, filtering, and analyzing operations. While users work with DataFrames, Snowflake seamlessly translates their actions into SQL, enhancing performance and simplifying the process.

Take Advantage of Snowflake's Performance Engine: All operations in Snowpark are executed within Snowflake's environment, ensuring that Snowflake's powerful engine handles the heavy lifting of data processing. This approach not only enhances performance but also keeps user's data secure and well-managed.

Custom Code with User-Defined Functions (UDFs): Snowpark lets users create their own special commands, known as UDFs, to perform custom tasks in Python. These custom commands run inside Snowflake's secure environment, offering a secure way to extend Snowflake's capabilities with their custom code.

Simplifying Data Workflows

With Snowpark for Python, we simplify data processing for Snowflake's users. Traditionally, working with data involves juggling different systems and complex steps. Snowpark for Python streamlines this by bringing everything into Snowflake's environment. This means users can focus more on analyzing and working with their data, and less on the hassle of managing multiple tools and systems.

Empowering Data Professionals

This course is perfect for those who are familiar with Python and are ready to explore its applications in Snowflake's development environment. It's tailored for those who are curious about leveraging Python for practical tasks and projects in Snowflake, regardless of their professional background. Join us and explore the exciting world of Snowpark for Python in Snowflake!

❄ Welcome to the Environment Setup!

This part will guide you through setting up your Snowflake environment using Conda. We will cover the benefits of Conda environments, install Miniconda, create a Conda environment for connecting with Snowpark, and install the necessary packages. By the end, you will have a fully functional Conda environment with Python and the required packages, ready-to-use Snowpark for your data processing tasks.

Prerequisites:

1. Basic understanding of Python programming language.
2. Understanding Snowflake's data platform.

Learning Objectives:

By the end of this section, you will be able to:

- ❖ Understand the concept and benefits of a Conda environment.
- ❖ Install Miniconda and set up a new Conda environment.
- ❖ Install necessary packages, including Snowpark for Python and Pandas, in the Conda environment.
- ❖ Verify the successful installation of packages in your Conda environment.

❄ Understanding Conda's Role in Your Projects

Snowpark allows you to bring Python directly into Snowflake, enabling you to leverage Python's powerful libraries and capabilities within Snowflake's robust Data Cloud Platform. Conda plays an essential role by providing isolated environments for each of your Snowpark projects, ensuring that the libraries and versions used in one project do not interfere with another, and making sure everything works smoothly and reliably.

Key Advantages of Using Conda:

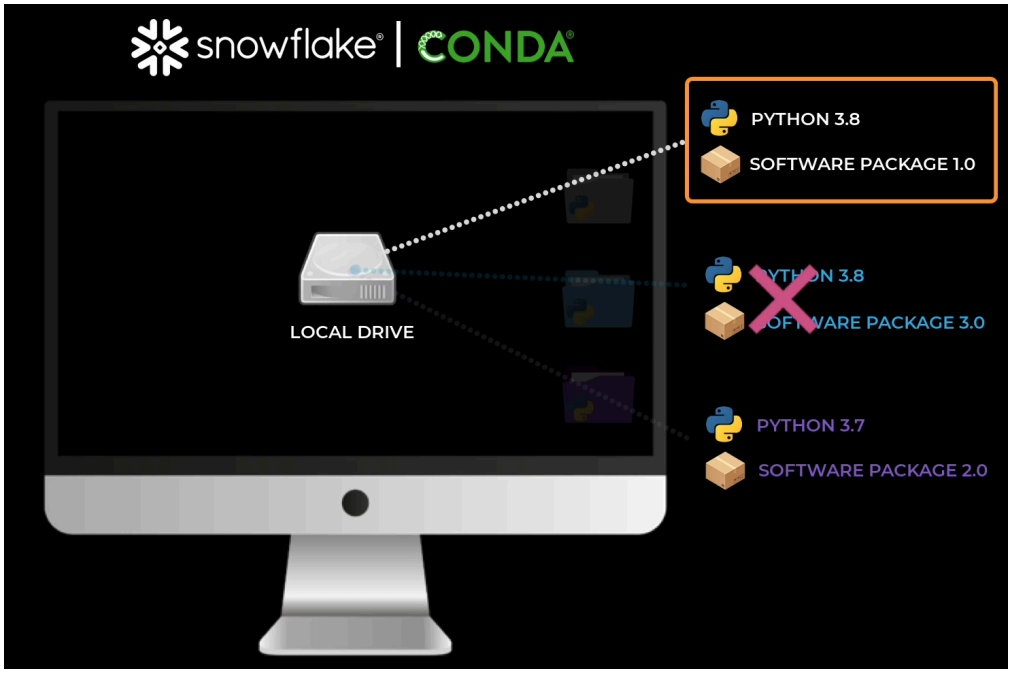
Separate Spaces for Projects: Conda excels at creating separate spaces, called environments, for each of your projects. This prevents your projects from interfering with each other, which helps avoid problems and keeps your work consistent across your development and production environments.

Software Management: Think of Conda as your digital librarian, organizing and managing your project's software needs. It simplifies installing, updating, and removing software packages, ensuring you always have the right tools for your projects.

Stable and Conflict-Free Work: Using Conda helps you maintain a stable and consistent setup for your work, avoiding common problems that arise when software doesn't work well together. This is especially helpful when working with others, ensuring everyone is using the same setup, which makes collaboration smoother and reduces mistakes.

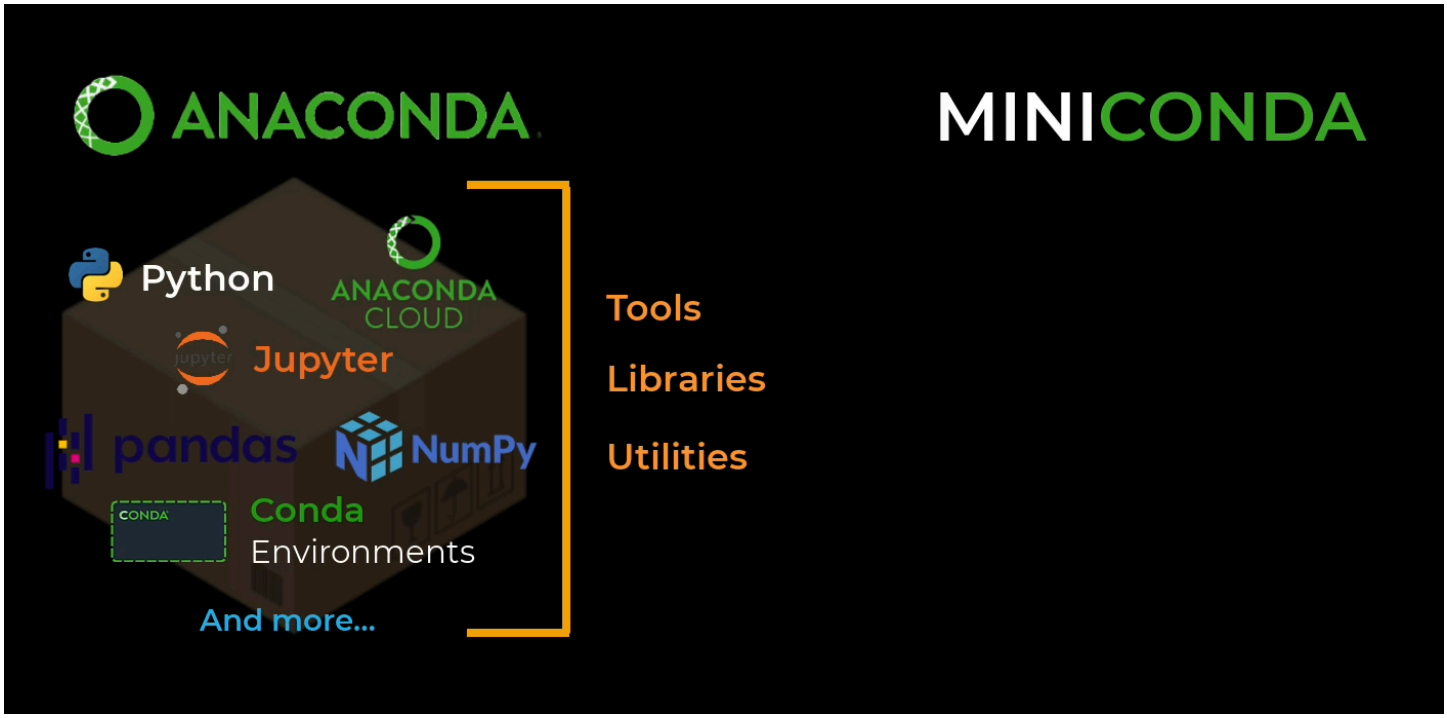
📝 Installation Guide

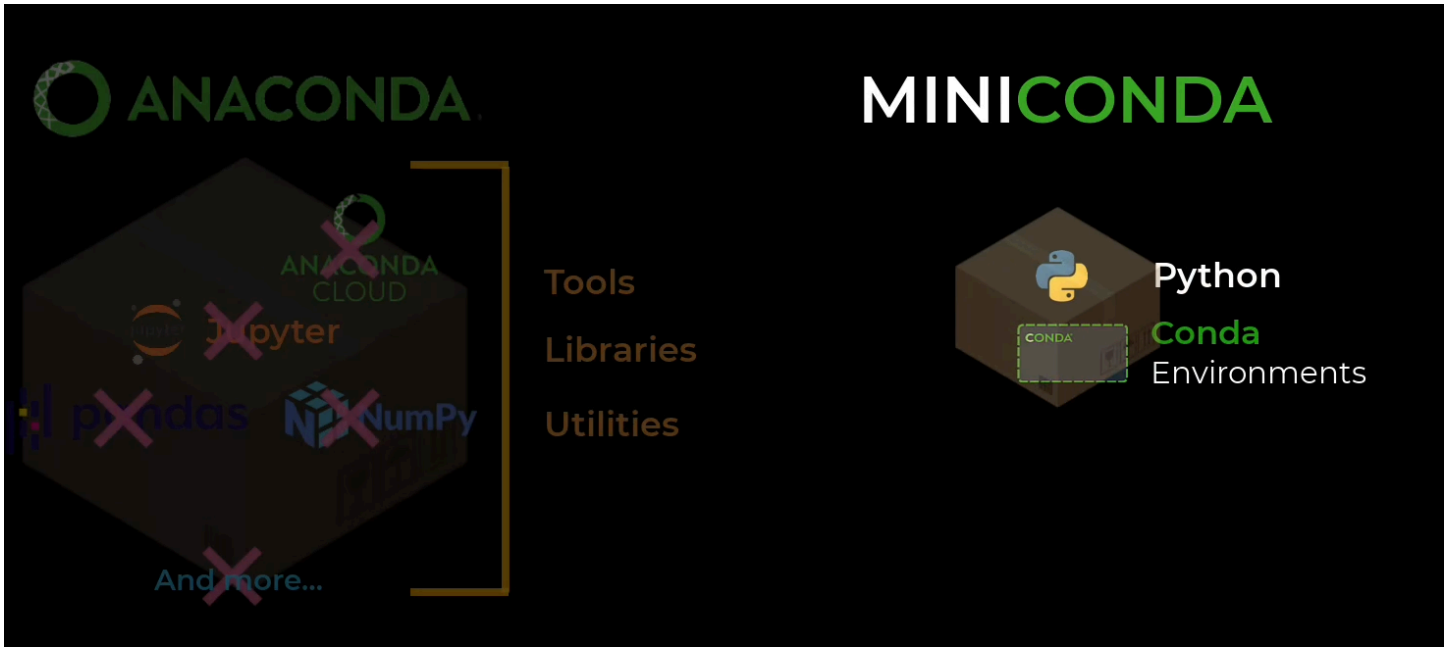
If you use the latest Miniconda installers, they include a Python version that may still need to be compatible with Snowflake. Therefore, you must create an environment with a compatible Python version after installation.




snowflake | CONDA
CONDA
PYTHON 3.8
SOFTWARE PACKAGE 1.0
CONDA
PYTHON 3.8
SOFTWARE PACKAGE 3.0
CONDA
PYTHON 3.7
SOFTWARE PACKAGE 2.0


snowflake | CONDA
CONDA
PYTHON 3.7
SOFTWARE PACKAGE 2.0


ANACONDA.
MINICONDA
Python
ANACONDA CLOUD
Jupyter
pandas
NumPy
Conda
Environments
And more...
Tools
Libraries
Utilities

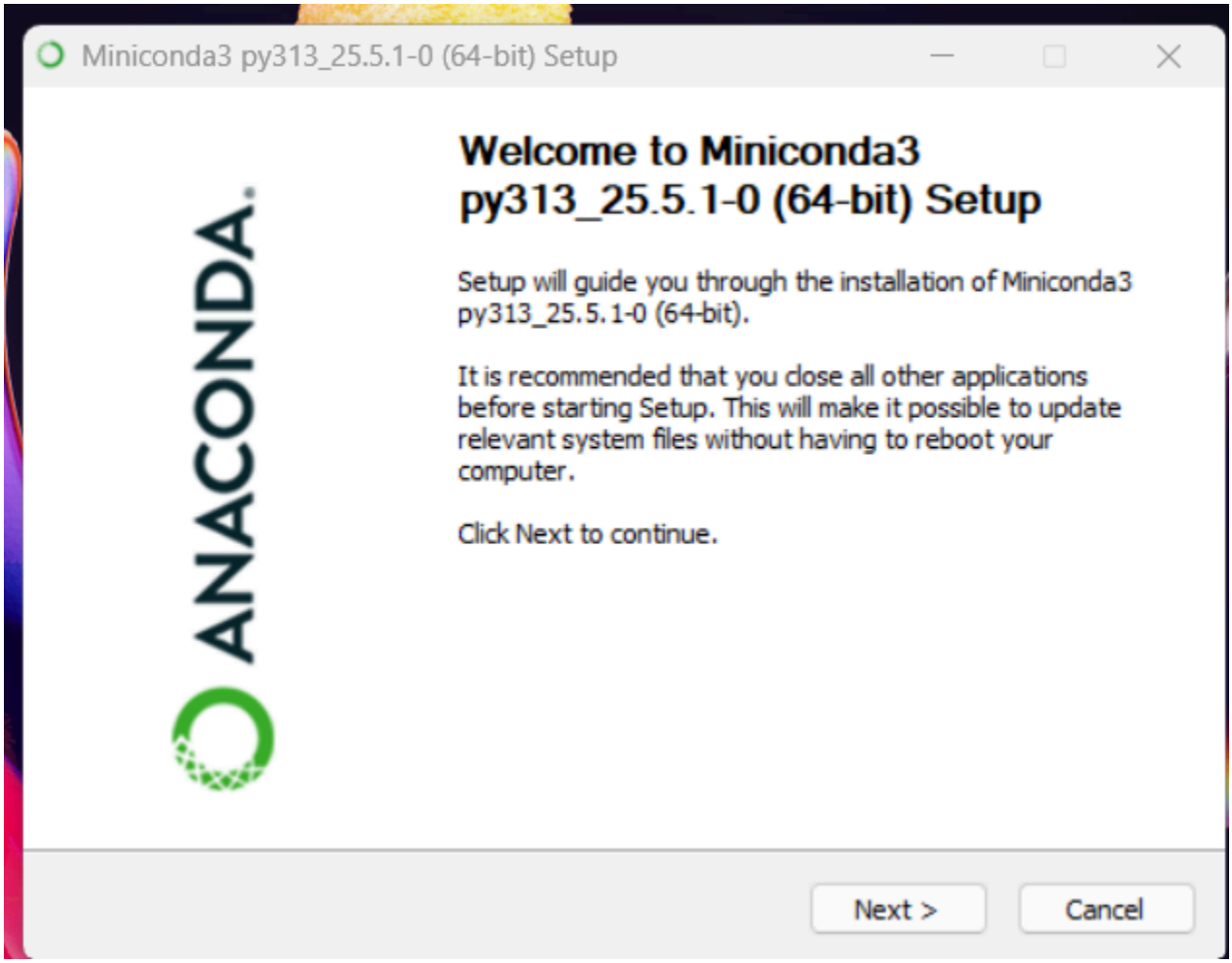

ANACONDA.
MINICONDA
ANACONDA CLOUD
Jupyter
pandas
NumPy
And more...
Tools
Libraries
Utilities
Python
Conda
Environments

1. Install Miniconda

Installing Miniconda - Anaconda : (opens in a new tab)

Please review the system requirements(opens in a new tab) for the Miniconda version you will install.

If you also have Anaconda installed, you may face issues installing Conda. Uninstall Anaconda and reinstall Miniconda if this happens.



2. Create new environment

Create a new environment with a Python version supported by Snowflake (opens in a new tab).
conda create -n <environment name> python=<version>

Below are two examples

```
conda create -n my_python311_env python=3.11
OR
conda create -n sandbox python=3.9
```

3. Activate the new environment

Create a new environment with the following code: conda activate <environment name> Below is an example:

```
conda activate my_python311_env
```

4. Install Snowpark

```
conda install snowflake-snowpark-python
```

```
(base) C:\Users\derek>python --version
Python 3.13.5

(base) C:\Users\derek>conda create -n my_python311_env python=3.11
Channels:
 - defaults
Platform: win-64
Collecting package metadata (repodata.json): done
Solving environment: done

## Package Plan ##

  environment location: C:\Users\derek\miniconda3\envs\my_python311_env

  added / updated specs:
    - python=3.11


The following packages will be downloaded:

    package                    |            build
    ---------------------------|-----------------
    bzip2-1.0.8                |       h2bbff1b_6          90 KB
    ca-certificates-2025.2.25  |       haa95532_0         130 KB
    expat-2.7.1                |       h8ddb27b_0         259 KB
    libffi-3.4.4               |       hd77b12b_1         122 KB
    openssl-3.0.16             |       h3f729d1_0         7.8 MB
    pip-25.1                   |     pyhc872135_2         1.3 MB
    python-3.11.13             |       h981015d_0        18.7 MB
    setuptools-78.1.1          |  py311haa95532_0         2.3 MB
    sqlite-3.45.3              |       h2bbff1b_0         973 KB
    tk-8.6.14                  |       h5e9d12e_1         3.5 MB
```

```
Proceed ([y]/n)? y


Downloading and Extracting Packages:

Preparing transaction: done
Verifying transaction: done
Executing transaction: done
#
# To activate this environment, use
#
#     $ conda activate my_python311_env
#
# To deactivate an active environment, use
#
#     $ conda deactivate


(base) C:\Users\derek>conda install snowflake-snowpark-python
```

A little troubleshoot!!

```
Could not solve for environment specs
The following packages are incompatible
├─ pin on python 3.13.* =* * is installable and it requires
│  └─ python =3.13 *, which can be installed;
└─ snowflake-snowpark-python =* * is not installable because there are no viable options
   ├─ snowflake-snowpark-python [0.10.0|0.11.0|...|1.9.0] would require
   │  └─ python >=3.8,<3.9.0a0 *, which conflicts with any installable versions previously reported;
   ├─ snowflake-snowpark-python [1.10.0|1.11.1|...|1.9.0] would require
   │  └─ python >=3.10,<3.11.0a0 *, which conflicts with any installable versions previously reported;
   ├─ snowflake-snowpark-python [1.10.0|1.11.1|...|1.9.0] would require
   │  └─ python >=3.11,<3.12.0a0 *, which conflicts with any installable versions previously reported;
   ├─ snowflake-snowpark-python [1.10.0|1.11.1|...|1.9.0] would require
   │  └─ python >=3.9,<3.10.0a0 *, which conflicts with any installable versions previously reported;
   └─ snowflake-snowpark-python [1.27.0|1.28.0|...|1.33.0] would require
      └─ python >=3.12,<3.13.0a0 *, which conflicts with any installable versions previously reported.

Pins seem to be involved in the conflict. Currently pinned specs:
 - python=3.13



(base) C:\Users\derek>conda create -n snowpark_py311 python=3.11
Channels:
```

Knowledge Check

1. What is the purpose of creating isolated Conda environments for each project?
   A. To use multiple languages in a single environment
   B. To manage different versions of Python and packages without conflicts
   C. To increase the processing speed of your Snowpark project
   D. To connect to different databases simultaneously

2. What happens if you install a new version of a software package globally in Conda?
   A. It will create a new isolated environment automatically.
   B. It overwrites the previous version, limiting you to the most recent version.
   C. It sends a notification to all users.
   D. It archives the old version for future retrieval.

3. What can you specify when creating a Conda environment according to the discussion?
   A. The version of the Python programming language and the internet browser version.
   B. The version of the Python programming language and the operating system version.
   C. The version of the Python programming language and additional packages.

4. Why would you choose Miniconda over Anaconda for your projects as suggested in the lesson?
   A. Miniconda supports more programming languages than Anaconda.
   B. Miniconda provides a minimal setup without extra packages included in Anaconda.
   C. Miniconda automates environment setup.
   D. Miniconda includes a built-in IDE, unlike Anaconda.

5. According to the discussion, what is a Conda environment?
   A. A virtual machine for Python development.
   B. A cloud storage service for Python code.
   C. A self-contained workspace for Python projects.
   D. A Python code compiler.

6. What issue does the discussion highlight that Conda environments help to solve?
   A. Snowflake connectivity issues.
   B. Snowpark syntax errors.
   C. Conflicts between different versions of software packages.
   D. Python code execution speed.

7. What is the consequence of not using isolated environments for different projects as per the lesson?
   A. Increased security risks.
   B. Inability to install Python packages.
   C. Limited to using only a single version of a software package globally.

8. According to the discussion, what does activating a Conda environment ensure?
   A. The project is shared with other developers.
   B. The Python script is converted into an executable format.
   C. The project is executed within a specific set of dependencies and packages.
   D. The Python interpreter is updated to the latest version.

9. Now finally, what are the two main functionalities of Conda as described in the discussion?
   A. Data visualization and analysis.
   B. Package management and environment management.
   C. Database management and web development.
   D. Code compilation and debugging.

Answer keys

1. A
2. B
3. C
4. B
5. C
6. C
7. C
8. C
9. B

❄ Connect to Snowflake using Snowpark for Python!

In this section, we will guide you through the process of connecting your Python application to the Snowflake database using Snowpark. We will cover the concept of session objects, setting up your Snowpark environment for Python, importing essential libraries, and creating a session object to establish a connection to Snowflake. By the end of this course, you will have a solid understanding of how to use Snowpark for Python to interact with Snowflake.

Prerequisites:

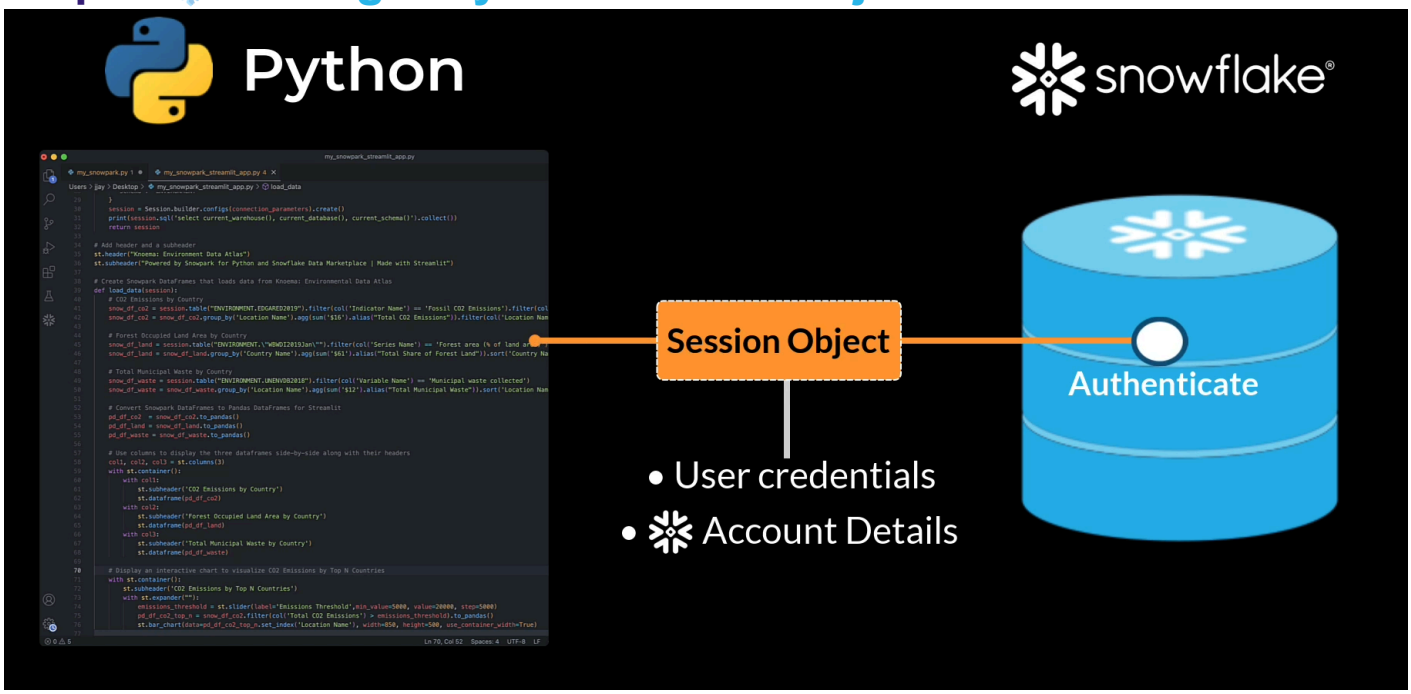
1. Basic understanding of Python programming language.
2. Familiarity with Snowflake's data platform.
3. Knowledge of Integrated Development Environments (IDEs).
4. Familiarity with Python libraries and packages.

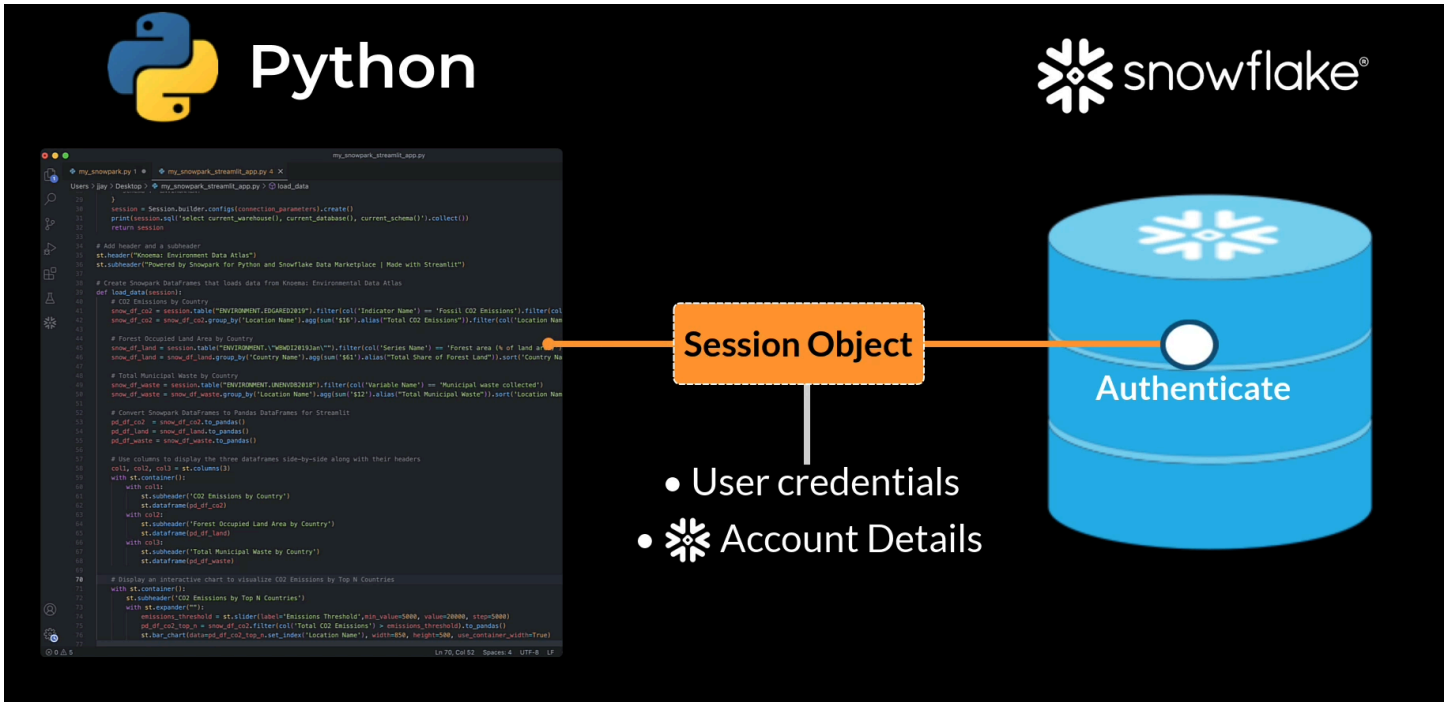
Learning Objectives:

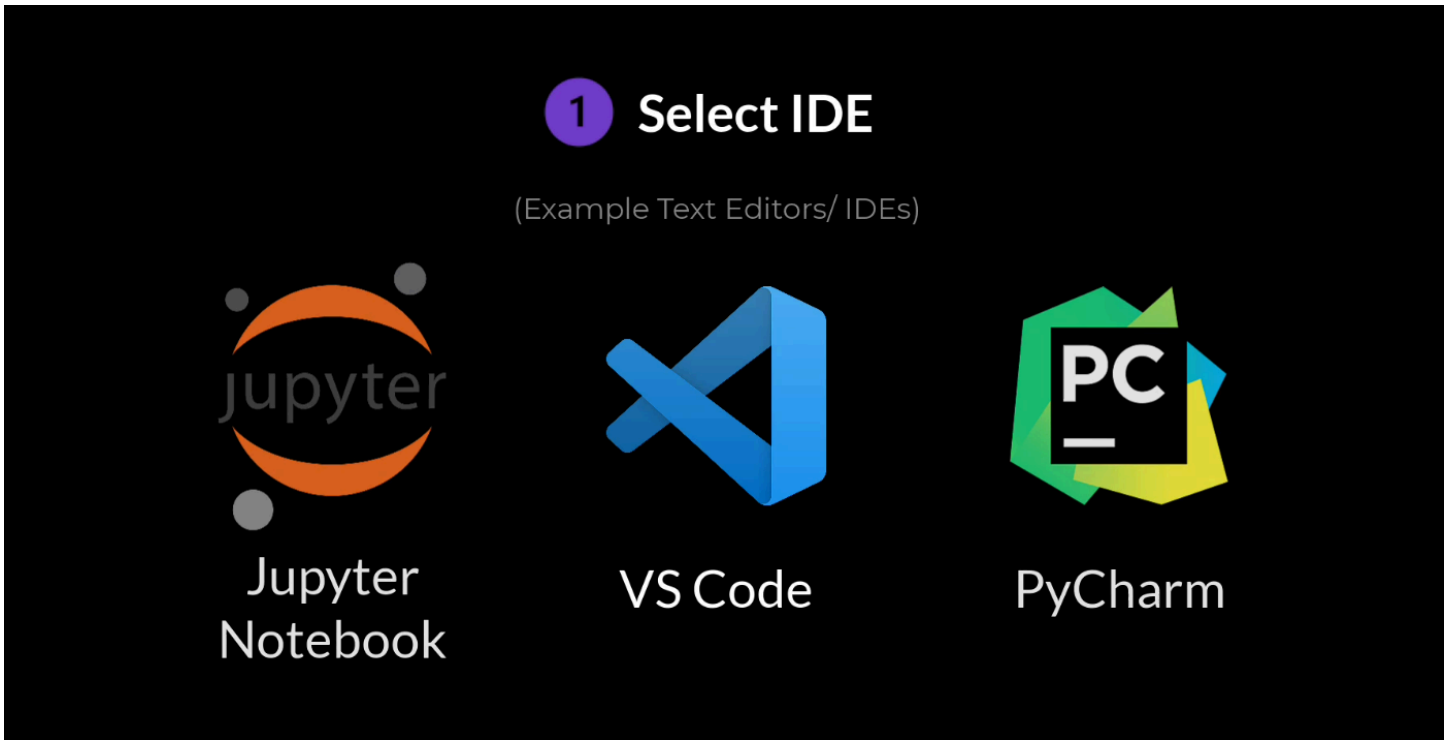
By the end of this section, you will be able to:

A. Understand the role and usage of session objects in connecting to Snowflake.
B. Import the necessary libraries for working with Snowflake.
C. Create a session object and establish a connection to Snowflake.

Step1: ❄ Configure your Session Object

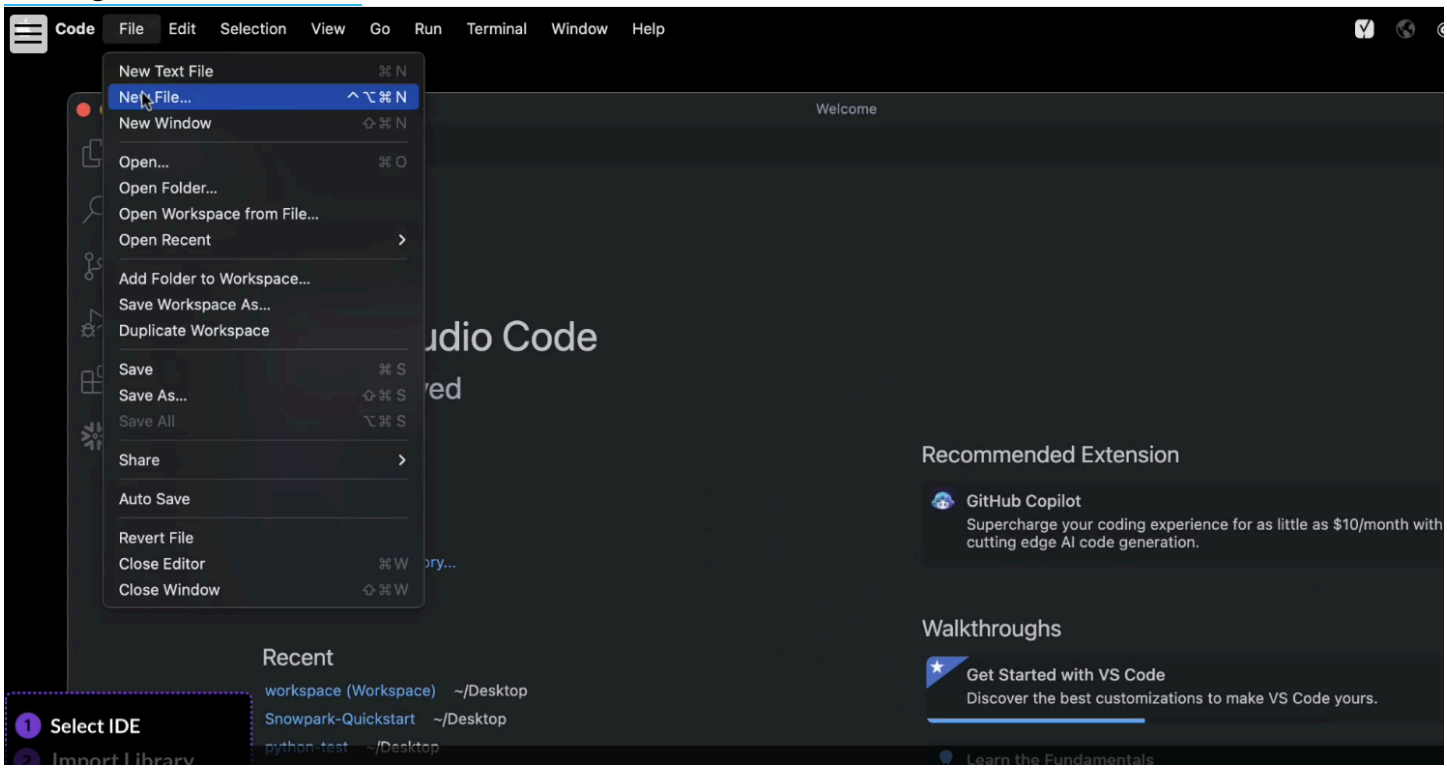




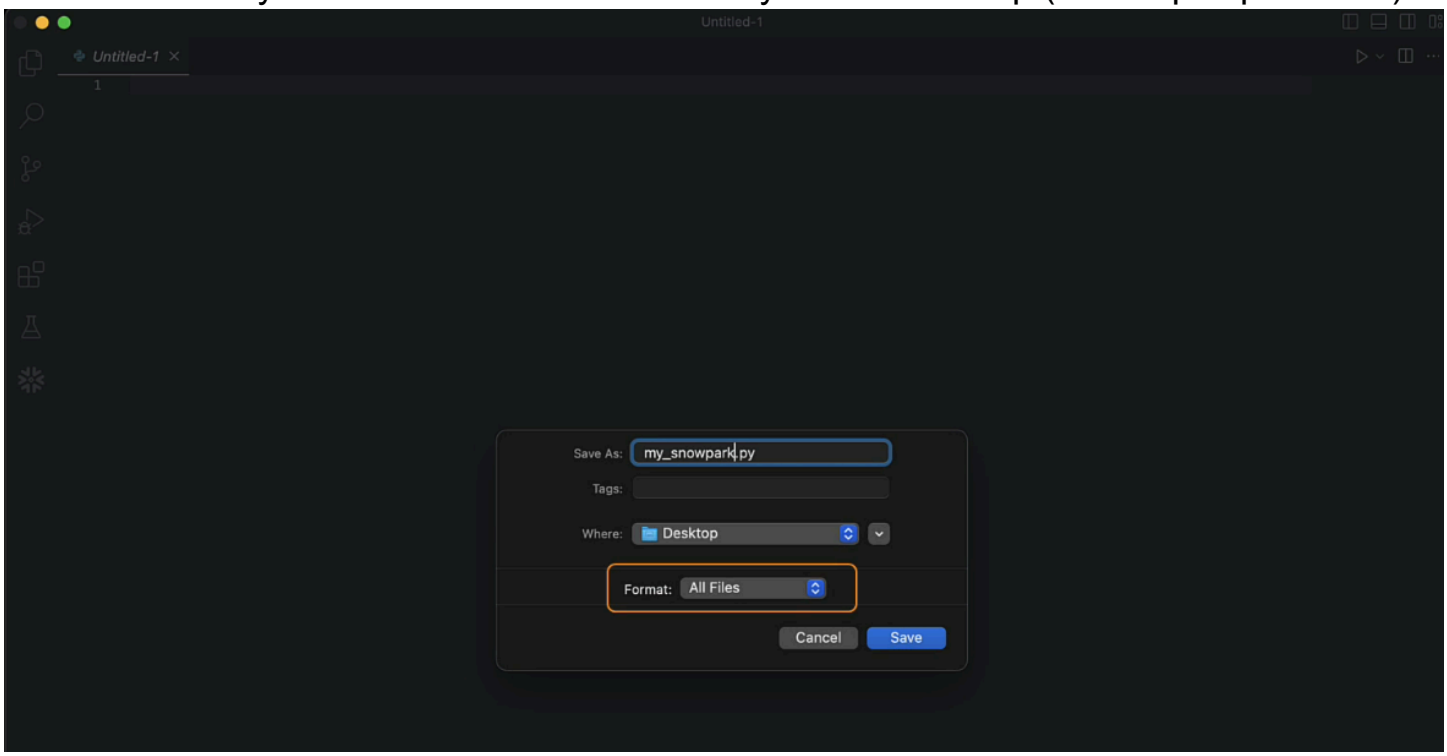


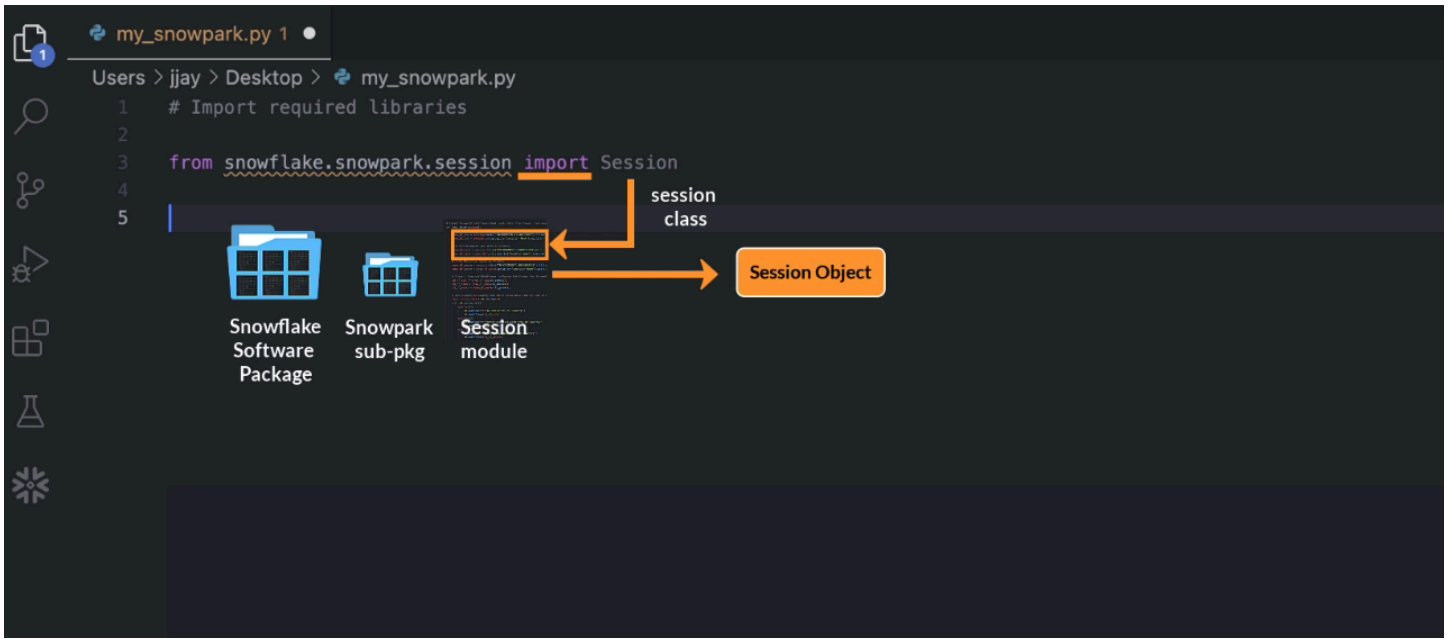
We will use VS code

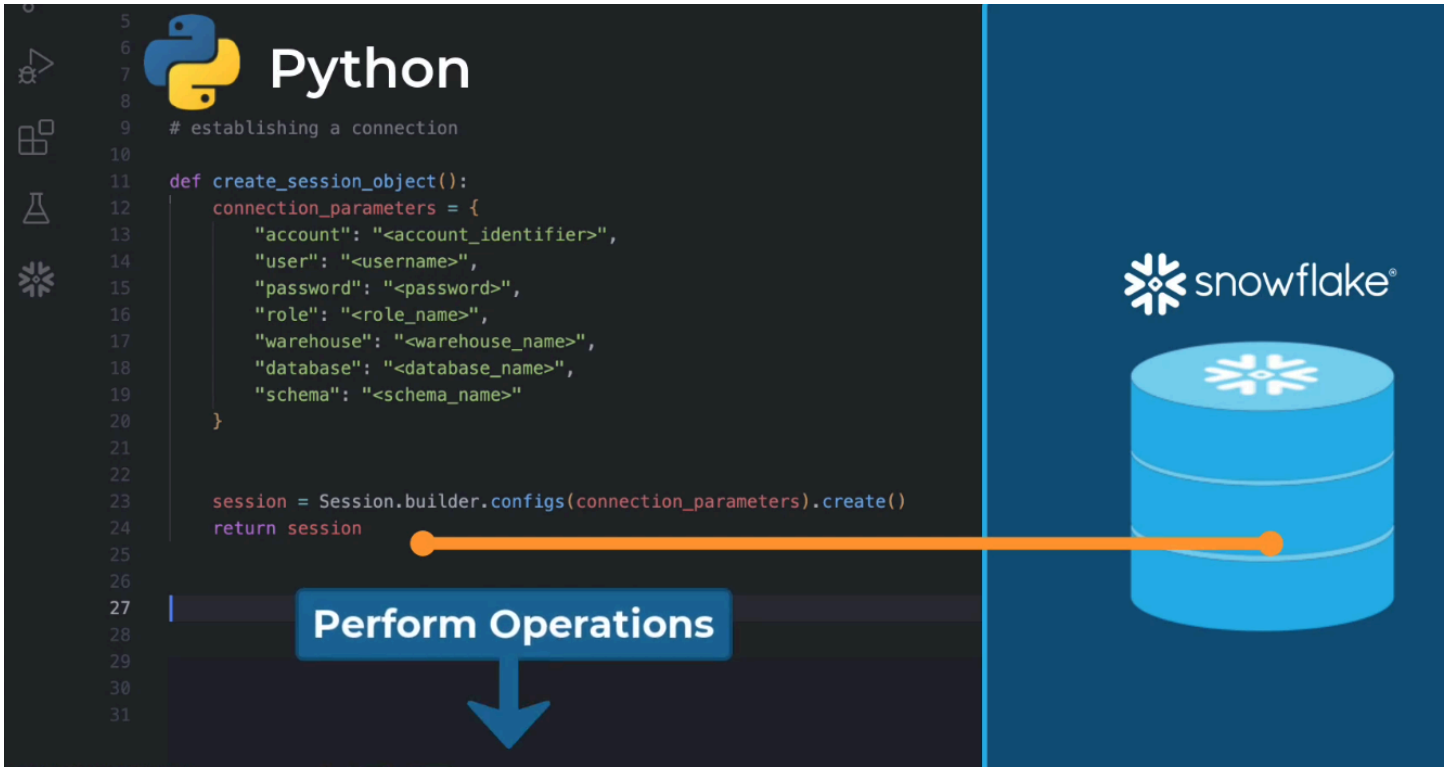
Let’s go to the VS Code



Create a new Python file inside a new directory inside Desktop (Desktop is preferred)







The code is below:

```
# Import the Session class from the snowflake.snowpark.session module
from snowflake.snowpark.session import Session

# Create Session object
def create_session_object():
    # Define the connection parameters
    connection_parameters = {
      "account": "__INSERT_ACCOUNT_HERE__",
      "user": "__INSERT_USERNAME_HERE__",
      "password": "__INSERT_PASSWORD_HERE__",
      "role": "__INSERT_ROLE_HERE__",
      "warehouse": "__INSERT_WAREHOUSE_HERE__",
      "database": "__INSERT_DATABASE_HERE__",
      "schema": "__INSERT_SCHEMA_HERE__"
    }


    # Create the session object with the provided connection parameters
    session = Session.builder.configs(connection_parameters).create()

    # Print the contents of your session object
    # Print(session)

    return session


# Function call
create_session_object()
```

Knowledge Check

1. What is the purpose of a session object in Snowflake?
   A. To execute queries.
   B. To import libraries.
   C. To authenticate the connection.
   D. To create a new file in the IDE.

2. What does the session object contain?
   A. Only your user credentials.
   B. Only the details of the Snowflake database.
   C. Both your user credentials and the details of the Snowflake database.
   D. Neither your user credentials nor the details of the Snowflake database.

3. Which Python API is used to facilitate interactions with Snowflake?
   A. Conda
   B. Snowpark
   C. Pandas
   D. NumPy

4. What is the role of a library in Python
   A. It serves as a toolbox of pre-written code for specific functionalities.
   B. It is used to establish a connection with a database.
   C. It stores the user credentials and database details.
   D. It is used to close a session.

5. How do we import a class from a specific module or package in Python?
   A. Using the 'import' keyword
   B. Using the 'package' keyword
   C. Using the 'session' keyword
   D. Using the 'class' keyword

6. Which of the following is included in the connection parameters dictionary for the session object?
   A. Username, password, account, role.
   B. Warehouse, database, schema.
   C. The first two options are both correct.
   D. None of the above.

Answer Keys

1. B
2. C
3. B
4. A
5. A
6. C

❄ Understanding and Working with DataFrames in Snowpark

In this section, you will be introduced to Snowpark DataFrames and learn how to use them efficiently. You'll explore the fundamental aspects of DataFrames, exploring their structure and components. Additionally, you will gain practical skills to use the Snowpark DataFrame API, including creating a DataFrame from a Snowflake Table and performing basic operations on it.

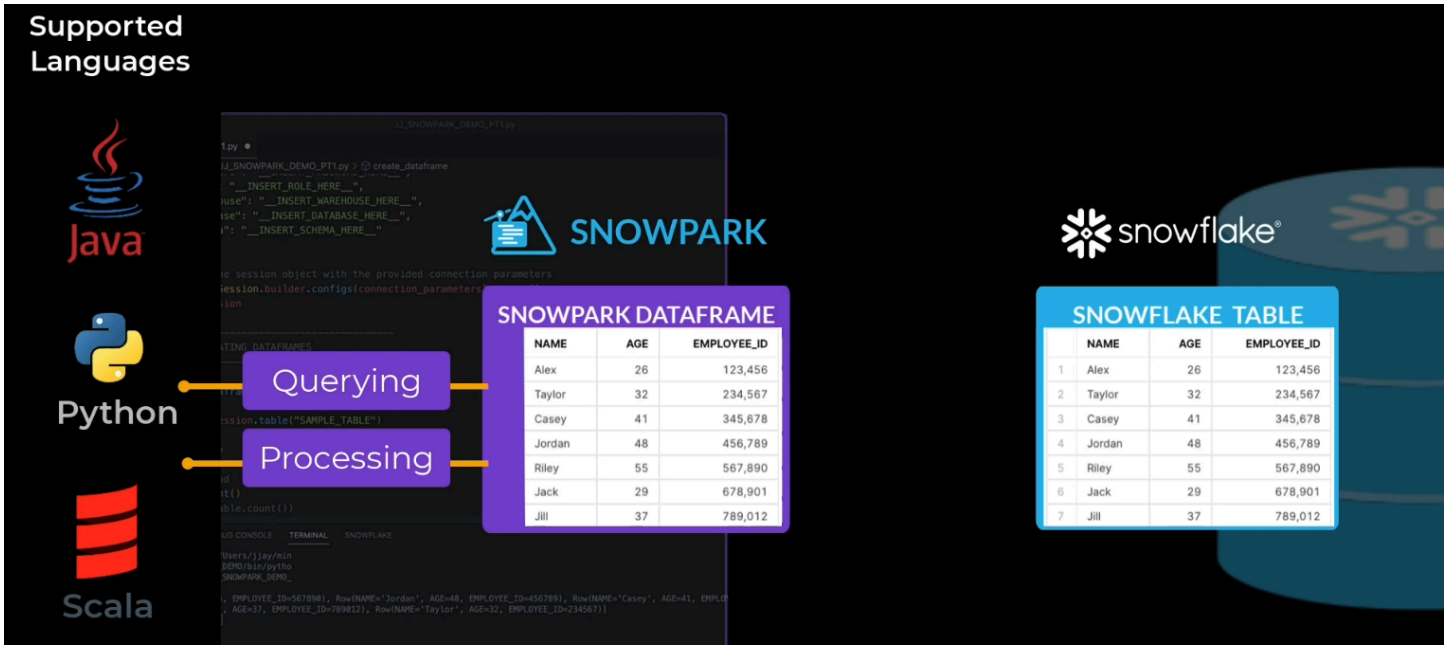
Prerequisites:

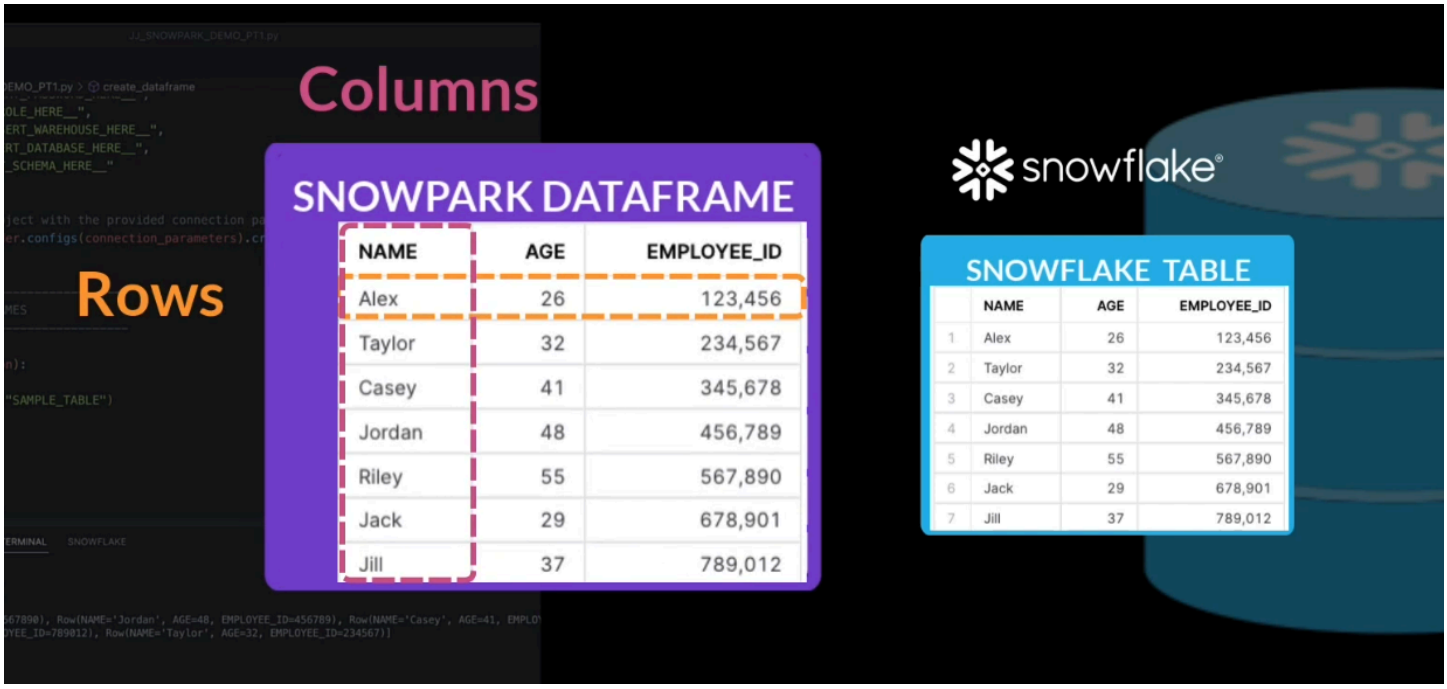
1. Basic understanding of Python programming language
2. Familiarity with Snowflake's data platform

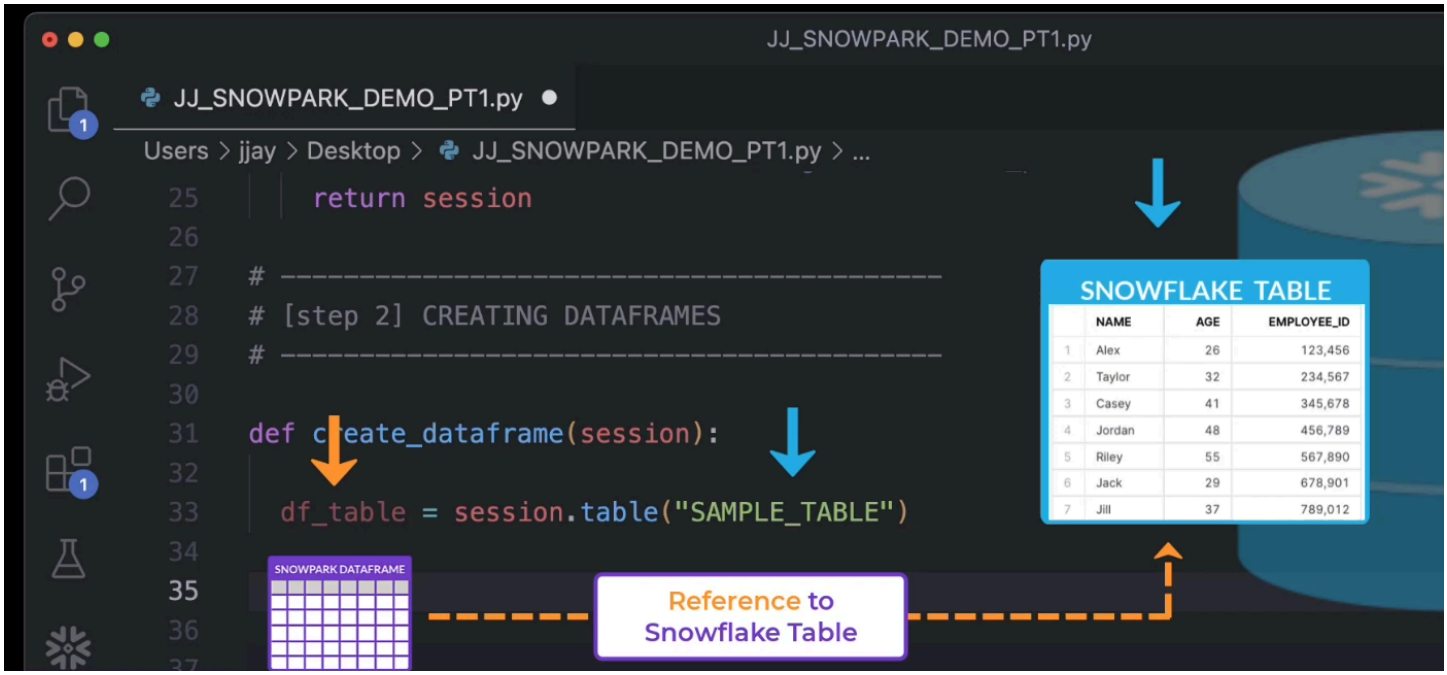
Learning Objectives:

By the end of this section, you will be able to:

1. Understand the fundamental concepts of DataFrames and their structure in Snowpark.
2. Create and manipulate DataFrames from a Snowflake Table.
3. Perform fundamental DataFrame operations, serving as a stepping stone towards executing more intricate operations.




Columns
Rows
SNOWPARK DATAFRAME
NAME AGE EMPLOYEE_ID
Alex 26 123,456
Taylor 32 234,567
Casey 41 345,678
Jordan 48 456,789
Riley 55 567,890
Jack 29 678,901
Jill 37 789,012
snowflake
SNOWFLAKE TABLE
NAME AGE EMPLOYEE_ID
1 Alex 26 123,456
2 Taylor 32 234,567
3 Casey 41 345,678
4 Jordan 48 456,789
5 Riley 55 567,890
6 Jack 29 678,901
7 Jill 37 789,012


JJ_SNOWPARK_DEMO_PT1.py
JJ_SNOWPARK_DEMO_PT1.py
Users > jjay > Desktop > JJ_SNOWPARK_DEMO_PT1.py > ...
25 return session
26
27 # ----------
28 # [step 2] CREATING DATAFRAMES
29 # ----------
30
31 def create_dataframe(session):
32
33 df_table = session.table("SAMPLE_TABLE")
34
35
36
SNOWPARK DATAFRAME
Reference to Snowflake Table
SNOWFLAKE TABLE
NAME AGE EMPLOYEE_ID
1 Alex 26 123,456
2 Taylor 32 234,567
3 Casey 41 345,678
4 Jordan 48 456,789
5 Riley 55 567,890
6 Jack 29 678,901
7 Jill 37 789,012

Data Frame Operations

Actions

- ☑ Information about the Data Frame
- ☑ Immediately returns a Value

Example Methods:

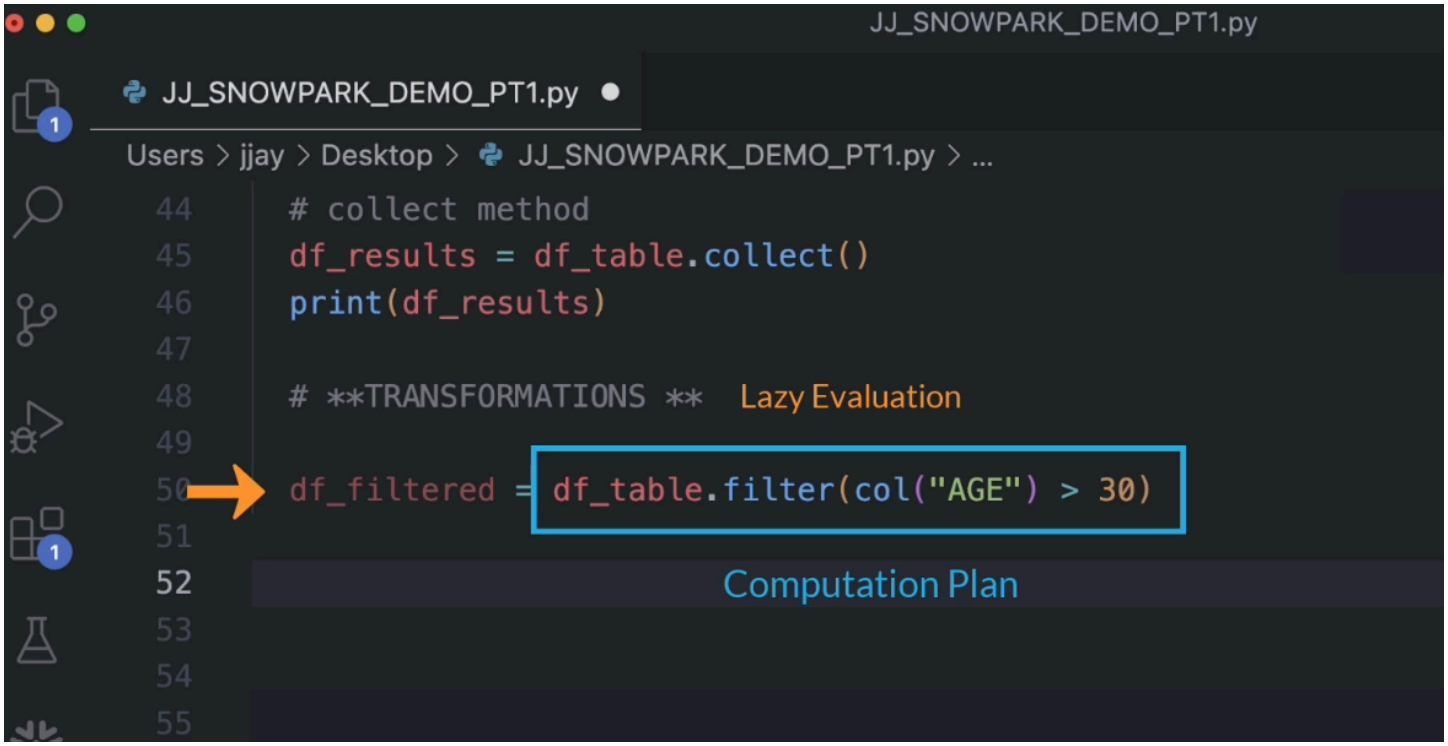
show ()
collect ()
...

Transformations

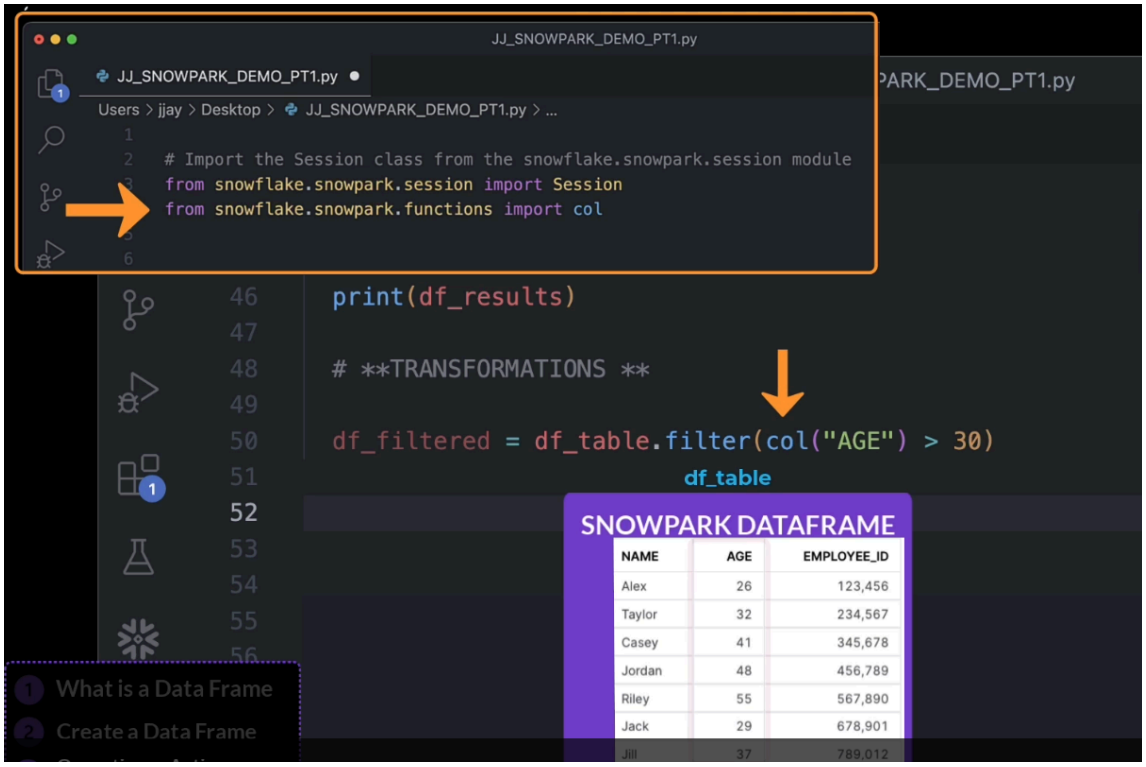
- ☑ Transforms the data
- ☑ Lazy Evaluation

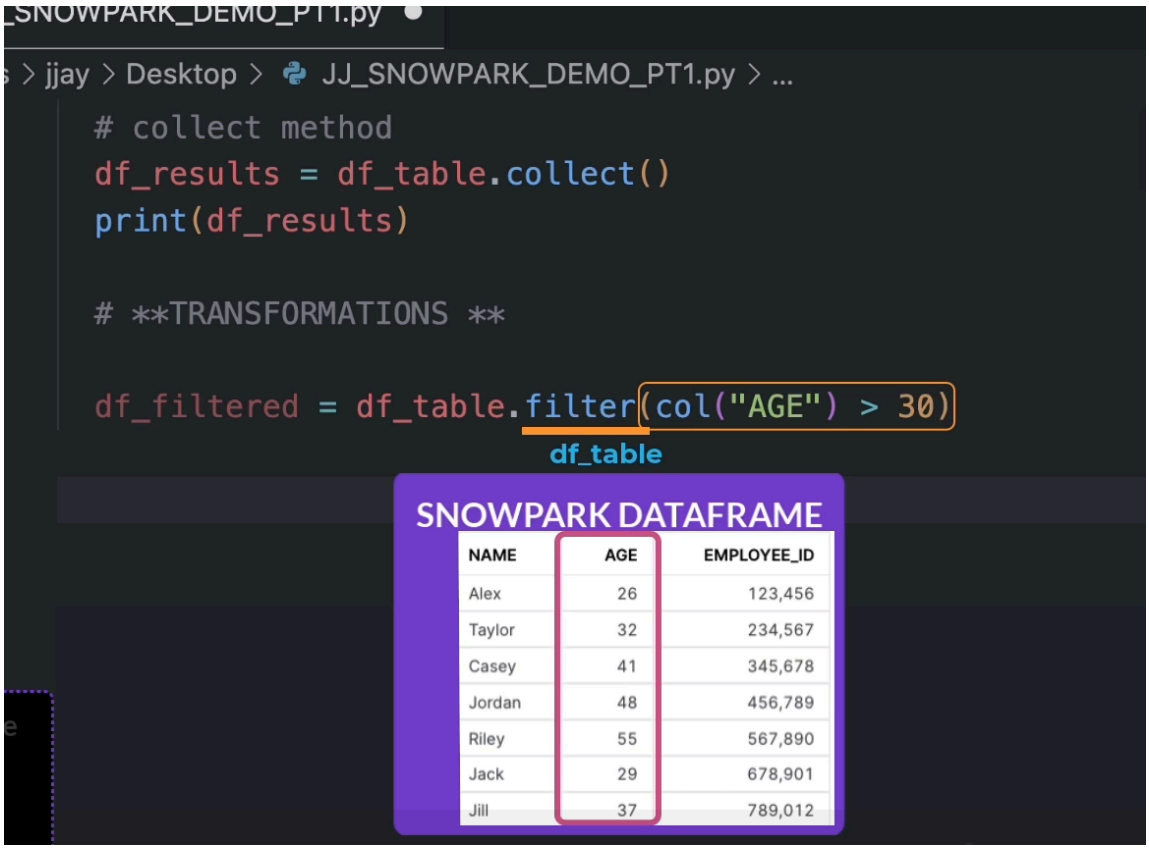
Example Methods:

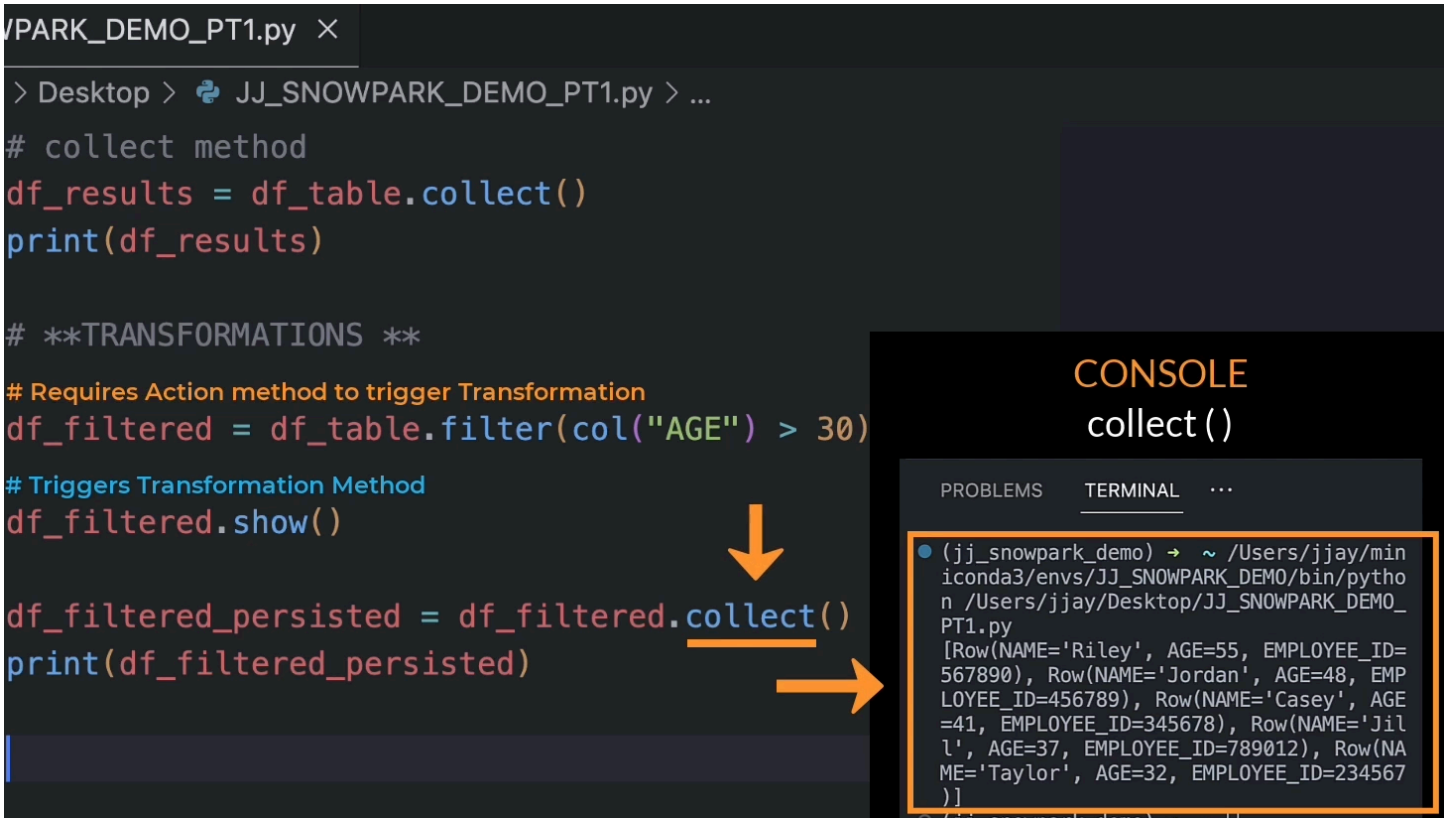
Filter ()
Sort ()
Agg ()


JJ_SNOWPARK_DEMO_PT1.py
JJ_SNOWPARK_DEMO_PT1.py
Users > jjay > Desktop > JJ_SNOWPARK_DEMO_PT1.py > ...
# collect method
df_results = df_table.collect()
print(df_results)
# **TRANSFORMATIONS **
Lazy Evaluation
df_filtered = df_table.filter(col("AGE") > 30)
Computation Plan


JJ_SNOWPARK_DEMO_PT1.py
JJ_SNOWPARK_DEMO_PT1.py
Users > jjay > Desktop > JJ_SNOWPARK_DEMO_PT1.py > ...
# collect method
df_results = df_table.collect()
print(df_results)
# **TRANSFORMATIONS **
df_filtered = df_table.filter(col("AGE") > 30)
Chain Transformation Methods
df_filtered = df_table.filter(col("AGE") > 30).sort(col("AGE").desc()).limit(10)
t is a Data Frame


JJ_SNOWPARK_DEMO_PT1.py
Users > jjay > Desktop > JJ_SNOWPARK_DEMO_PT1.py > ...
# Import the Session class from the snowflake.snowpark.session module
from snowflake.snowpark.session import Session
from snowflake.snowpark.functions import col
print(df_results)
# **TRANSFORMATIONS **
df_filtered = df_table.filter(col("AGE") > 30)
df_table
SNOWPARK DATAFRAME
NAME AGE EMPLOYEE_ID
Alex 26 123,456
Taylor 32 234,567
Casey 41 345,678
Jordan 48 456,789
Riley 55 567,890
Jack 29 678,901
Jill 37 789,012
What is a Data Frame
Create a Data Frame


jjay > Desktop > JJ_SNOWPARK_DEMO_PT1.py > ...
# collect method
df_results = df_table.collect()
print(df_results)
# **TRANSFORMATIONS **
df_filtered = df_table.filter(col("AGE") > 30)
df_table
SNOWPARK DATAFRAME
NAME AGE EMPLOYEE_ID
Alex 26 123,456
Taylor 32 234,567
Casey 41 345,678
Jordan 48 456,789
Riley 55 567,890
Jack 29 678,901
Jill 37 789,012



❄️ Try it yourself with a different database

Copy and paste the following code.

```
# Import Libraries
from snowflake.snowpark.session import Session
from snowflake.snowpark.functions import col
```

Step 1 CREATE SAMPLE TABLE

```
# [step 1] CREATE SAMPLE TABLE

CREATE TABLE sample_table AS
SELECT
       catalog_sales,
       customer,
       item,
       catalog_returns
  FROM
       SNOWFLAKE_SAMPLE_DATA.TPCDS_SF100TCL;
```

Step 2: CREATE SESSION OBJECT

```
# [step 1] CREATE SESSION OBJECT

def create_session_object():

    # Define the connection parameters
    connection_parameters = {
        "account": "__INSERT_ACCOUNT_HERE__",
        "user": "__INSERT_USERNAME_HERE__",
        "password": "__INSERT_PASSWORD_HERE__",
        "role": "__INSERT_ROLE_HERE__",
        "warehouse": "__INSERT_WAREHOUSE_HERE__",
        "database": "__INSERT_DATABASE_HERE__",
        "schema": "__INSERT_SCHEMA_HERE__"
    }

    # Create the session object with the provided connection parameters
    session = Session.builder.configs(connection_parameters).create()
    return session
```

Step 3: CREATING DATAFRAMES

```
def create_dataframe(session):

    # Create a dataframe
    df_table = session.table("SAMPLE_TABLE")

    #---------------------------------
    # **ACTIONS**
    #---------------------------------

    # count method
    df_table.count()
    # print(df_table.count())

    # show method
    df_table.show()

    # collect method
    df_results = df_table.collect()
    # print(df_results)

    #---------------------------------
    # **TRANSFORMATIONS **
    #---------------------------------

    df_filtered = df_table.filter(col("CS_BILL_CUSTOMER_SK") > 6000000)

    # Chaining method calls
    # df_filtered = df_table.filter(col("CS_BILL_CUSTOMER_SK") >
6000000).sort(col("CS_SHIP_ADDR_SK").desc()).limit(2000000)
```

```
    df_filtered.show()

    df_filtered.collect()

    df_filtered_persisted = df_filtered.collect()
    # print(df_filtered_persisted)
#
-------------------------------------------------------------------------------
--
```

FUNCTION CALLS

```
# call session object
session = create_session_object()

# call create dataframe
_ = create_dataframe(session)

# end your session
session.close()
```

Knowledge Check

1. How do you create a DataFrame from a Snowflake table in Snowpark?
   A. By using the row method provided by our session class.
   B. By using the index method provided by our session class.
   C. By using the table method provided by our session class.
   D. By using the column method provided by our session class.

2. What is the key difference between transformations and actions in Snowpark?
   A. Transformations are immediately evaluated while actions are lazily evaluated.
   B. Actions are immediately evaluated while transformations are lazily evaluated.
   C. Transformations are only used for data manipulation while actions are used for data retrieval.
   D. Actions are only used for data manipulation while transformations are used for data retrieval.

3. What is the advantage of lazy evaluation in Snowpark?
   A. It allows transformations to be performed immediately.
   B. It allows actions to be chained together.
   C. It allows transformations to be chained together, optimizing data processing.
   D. It allows actions to be performed on a DataFrame before it is fully loaded.

Explanation is available

4. When does the actual execution of transformations occur in Snowpark?
   A. When an action is performed on the DataFrame.
   B. When a transformation is performed on the DataFrame.
   C. When an index is applied to the DataFrame.
   D. When a row is added to the DataFrame.

5. What is the role of the *show* method in Snowpark?
   A. It is used to display the DataFrame to the console.
   B. It is used to hide certain data in the DataFrame.
   C. It is used to show the memory address of the DataFrame.
   D. It is used to show the unique identifiers of rows in the DataFrame.

6. What does the collect method do in Snowpark?
   A. It retrieves all the data from the DataFrame and returns it as an array of row objects that can be stored in memory.
   B. It collects and stores the metadata of the DataFrame.

C. It combines multiple DataFrames into one.
D. It collects specific rows from the DataFrame based on certain conditions.

Explanation is available

7. How can one persist data in Snowpark?
   A. By using the show method.
   B. By using the count method.
   C. By using the collect method.
   D. By using the filter method.

8. Which of the following is NOT a component of a Snowpark DataFrame?
   A. Rows
   B. Columns
   C. Indexes
   D. Functions

Explanation is available

9. How can you reference columns in a DataFrame when using the filter method?
   A. By using the row function from the snowflake snowpark functions module.
   B. By using the col function from the snowflake snowpark functions module.
   C. By using the index function from the snowflake snowpark functions module.
   D. By using the table function from the snowflake snowpark functions module.

Explanation is available

10. In Snowpark, what happens when the *show* method is used after a transformation?
    A. The transformation is discarded and the original DataFrame is shown.
    B. The transformed DataFrame is displayed, and the original DataFrame is kept in memory.
    C. The transformed DataFrame is displayed, but not stored in memory.
    D. The transformation is stored in memory, but not displayed.

Explanation is available

Answer Keys

1. C
2. B
3. C
4. A
5. A
6. A
7. C
8. C
9. B
10. C

3🔍 Explanation:
In Snowpark for Snowflake, lazy evaluation means that:
Transformations (like filter, select, withColumn, etc.) are not executed immediately.
Instead, they are recorded in a logical execution plan.
The actual execution only happens when an action is triggered (like collect(), toPandas(), show(), etc.).

💡 Why is Lazy Evaluation Important?
✅ Optimized Query Planning: Snowflake can analyze the entire chain of transformations and push down operations efficiently as a single SQL query.
✅ Reduced Data Movement: Only necessary data is processed and transferred.
✅ Better Performance: Snowflake can use its cost-based optimizer to generate the most efficient execution plan.

6.🔍 Explanation:
In Snowpark, the .collect() method:
Triggers execution of the lazy transformation pipeline.
Retrieves all rows from the Snowflake table (or query result).
Returns them as a list of Row objects in the local Python (or Scala/Java) client memory.

💡 Important Notes:
It is an action, not a transformation.
Use with caution on large datasets — it pulls all rows into memory, which can lead to out-of-memory errors if the dataset is too large.
For large datasets, consider using .limit() or .to_pandas() with sample sizes instead.

8🔍 Explanation:
A Snowpark DataFrame represents a lazy, distributed SQL query plan — it doesn't behave like a Pandas DataFrame and does not support indexes.
✅ Components of a Snowpark DataFrame:
Component Description
Rows Represented as Row objects after actions like .collect().
Columns Represented by Column expressions used in transformations like .select() or .withColumn().
Functions Built-in Snowpark functions (like col(), lit(), sum()) are used to transform columns.
❌ Indexes:
Snowpark DataFrames are based on SQL tables and queries, which do not expose indexing the way Pandas or R DataFrames do.
You can't use index-based access like .iloc or .loc.
9🔍 Explanation:
In Snowpark, when you're filtering a DataFrame (e.g., using .filter() or .where()), you need to reference columns as Column objects, not as plain strings.
To do this, you use:
from snowflake.snowpark.functions import col
Then reference the column like:
df.filter(col("age") > 30)
This lets you write expressions that Snowpark can compile into SQL.

❌ Why Other Options Are Incorrect:

Option	Why it's incorrect
row	❌ Used to construct Row objects (e.g., for local data), not for referencing columns.
index	❌ There is no `index` function in Snowpark; Snowpark DataFrames do not use indexes like Pandas.
table	❌ Used to reference entire tables, not individual columns within a DataFrame.

✅ Best Practice:

```
from snowflake.snowpark.functions import col
df.filter(col("status") == "active")
```

10🔍 Explanation:
In Snowpark, when you call the .show() method after a transformation, here's what happens:

The entire transformation chain (because of lazy evaluation) is compiled into a SQL query.

That query is executed on Snowflake.

The result (default: top 10 rows) is retrieved and printed to your console/output.

❌ The result is not stored — either in memory or as a table — unless you explicitly call .cache_result() or .write.

✅ Behavior Summary:

Action	Description
Transformations	Lazy; they build a logical query plan.
.show()	Triggers execution, prints result, does not store data.
Memory	No in-memory caching unless explicitly done.
Persistence	Use .write.mode("overwrite").save_as_table("table_name") to store.