Chapter 2
Downloading/Setting up Apache Spark

What is Homebrew?

- **Homebrew is a popular package manager for macOS.**
- **It allows users to easily install, update, and manage software packages.**

Benefits from Homebrew:

- ➔ **Streamlines software installation: No need to download and compile software manually.**
- ➔ **Offers a vast library of software packages for developers and users.**
- ➔ **Maintained by a community of contributors, ensuring up-to-date packages.**

Basic commands: **brew install**, **brew update**, **brew upgrade**, and more.

Step 1 - Downloading Apache Spark for Python

1. **Install Homebrew**
   Simplify the installation and management of various software packages on your system.

   Run the following command in your terminal:

   /bin/bash -c "$(curl -fsSL https://raw.githubusercontent.com/Homebrew/install/HEAD/install.sh)"

2. **Install Python with Homebrew**
   Ensure you have Python installed

   Run the following command: **brew install python**

3. **Install Apache Spark for Python**
   Enable integration of Apache Spark with Python

   Run the following command: **pip install pyspark**

   Update command: **pip install --upgrade pyspark**

Step 2 - Test

1. **Run the following command on the terminal:**

```
pyspark
# Used to start an interactive PySpark shell, which is an interactive
# Python shell with PySpark functionality enabled.
```

You should expect this:

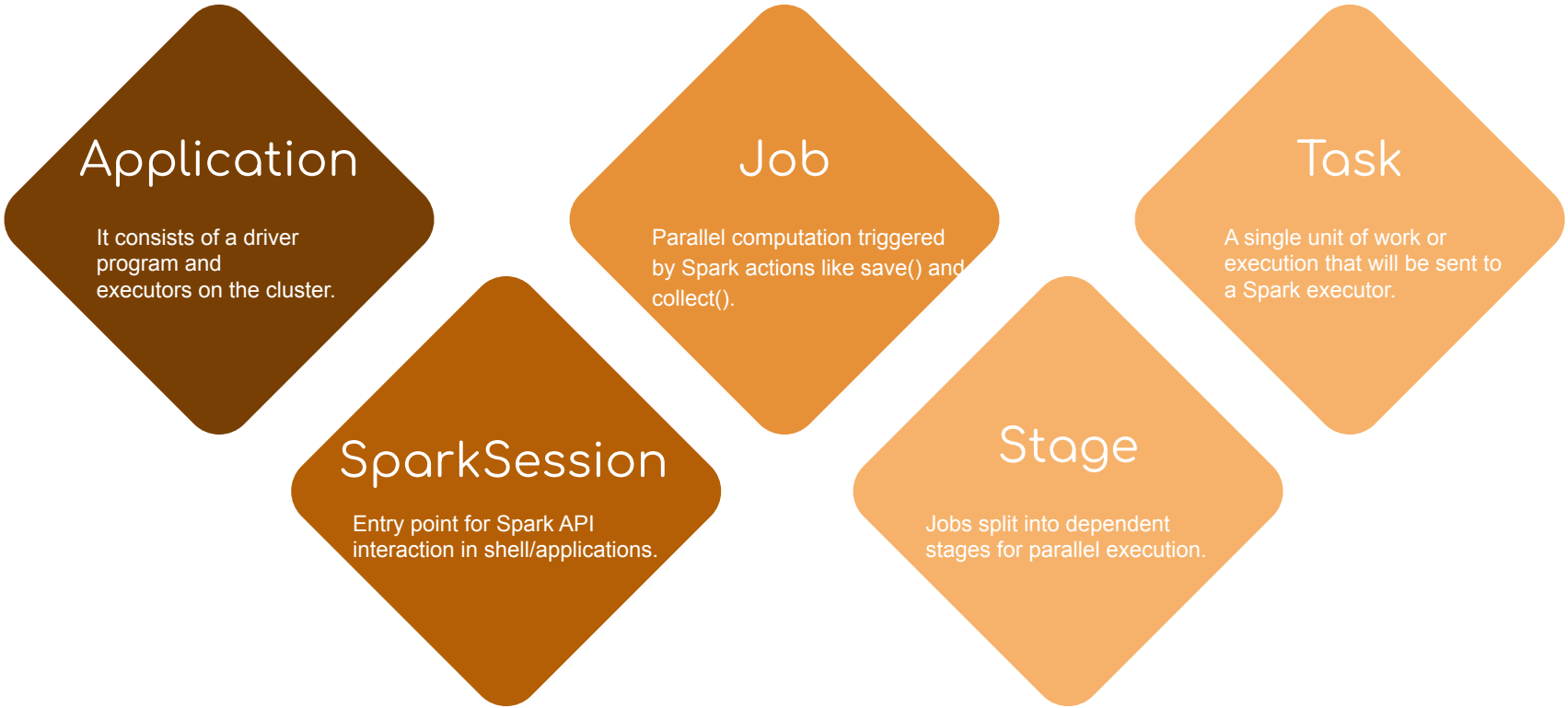
```
~ pyspark
Python 3.10.7 (v3.10.7:6cc6b13308, Sep  5 2022, 14:02:52) [Clang 13.0.0 (clang-1300.0.29.30)] on darwin
Type "help", "copyright", "credits" or "license" for more information.
Setting default log level to "WARN".
To adjust logging level use sc.setLogLevel(newLevel). For SparkR, use setLogLevel(newLevel).
23/10/01 16:54:08 WARN NativeCodeLoader: Unable to load native-hadoop library for your platform... using builtin-java classes where applicable
Welcome to
      ____              __
     / __/__  ___ _____/ /__
    _\ \/ _ \/ _ `/ __/  '_/
   /__ / .__/\_,_/_/ /_/\_\   version 3.5.0
      /_/

Using Python version 3.10.7 (v3.10.7:6cc6b13308, Sep  5 2022 14:02:52)
Spark context Web UI available at http://geralds-mbp.fritz.box:4040
Spark context available as 'sc' (master = local[*], app id = local-1696172049014).
SparkSession available as 'spark'.
```

To exit the shell, type: **quit()**

An example of using the Python Interpretive shell:

```
~ pyspark
Python 3.10.7 (v3.10.7:6cc6b13308, Sep  5 2022, 14:02:52) [Clang 13.0.0 (clang-1300.0.29.30)] on darwin
Type "help", "copyright", "credits" or "license" for more information.
Setting default log level to "WARN".
To adjust logging level use sc.setLogLevel(newLevel). For SparkR, use setLogLevel(newLevel).
23/10/01 16:54:08 WARN NativeCodeLoader: Unable to load native-hadoop library for your platform... using builtin-java classes where applicable
Welcome to
      ____              __
     / __/__  ___ _____/ /__
    _\ \/ _ \/ _ `/ __/  '_/
   /__ / .__/\_,_/_/ /_/\_\   version 3.5.0
      /_/

Using Python version 3.10.7 (v3.10.7:6cc6b13308, Sep  5 2022 14:02:52)
Spark context Web UI available at http://geralds-mbp.fritz.box:4040
Spark context available as 'sc' (master = local[*], app id = local-1696172049014).
SparkSession available as 'spark'.
>>> strings = spark.read.text("README.md")
# The command reads the content of the "README.md" file into a PySpark DataFrame named "strings."

>>> strings.show(10, truncate=False)
# This command displays the first 10 rows of the "strings" DataFrame without truncating the text in each cell.

+------------------------------------------------------------------------------------------------------+
|value                                                                                                 |
+------------------------------------------------------------------------------------------------------+
|# Apache Spark                                                                                        |
|                                                                                                      |
|Spark is a unified analytics engine for large-scale data processing. It provides                      |
|high-level APIs in Scala, Java, Python, and R, and an optimized engine that                           |
|supports general computation graphs for data analysis. It also supports a                             |
|rich set of higher-level tools including Spark SQL for SQL and DataFrames,                            |
|pandas API on Spark for pandas workloads, MLlib for machine learning, GraphX for graph processing,   |
|and Structured Streaming for stream processing.                                                       |
|                                                                                                      |
|<https://spark.apache.org/>                                                                           |
+------------------------------------------------------------------------------------------------------+
only showing top 10 rows

>>> strings.count()
125
```

Important Concepts



Transformations, Actions, and Lazy Evaluation

Spark operations on distributed data fall into two categories:
Transformations and **Actions**.

Transformations create a Spark DataFrame starting from another one, without modifying the original data.
Example: Operations like `select()` or `filter()` return a new DataFrame.

Transformations are evaluated ***lazily*
Meaning that results are not computed immediately, but they are recorded or remembered as a lineage.

Actions RDD actions are operations that return the raw values

Example: Spark actions include count() and collect()

Transformations	Actions
orderBy()	show()
groupBy()	take()
filter()	count()
select()	collect()
join()	save()

Example of Transformation

```
>>> strings = spark.read.text("README.md")
>>> filtered = strings.filter(strings.value.contains("Spark"))
>>> filtered.count()
20
```

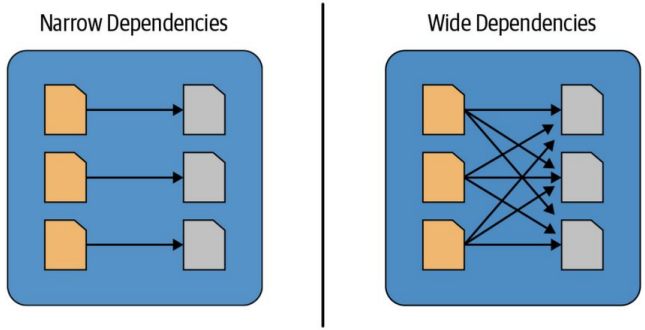
- It includes two transformations: `read()` (to read data) and `filter()` (to filter data).
- The key point is that nothing happens (in the terminal) when transformations are applied: execution is deferred until an action, such as `count()`, is called.
- In this example, the actual processing and filtering of data occur only when `filtered.count()` is executed in the Spark shell.

Narrow and Wide Transformations

Transformations can be classified into two groups:
Narrow and **Wide** Transformations

Narrow transformations are those where each output can be computed from a single input.
They are *efficient* and and *fast.*
`filter()`, `map()`, `union()`, and `contains()`.

Wide transformations are those that require data from multiple input partitions to compute the output.
They are slower and less efficient as they require data shuffling to be performed.
`groupBy()`, `reduceByKey()`, `join()`, and `orderBy()`



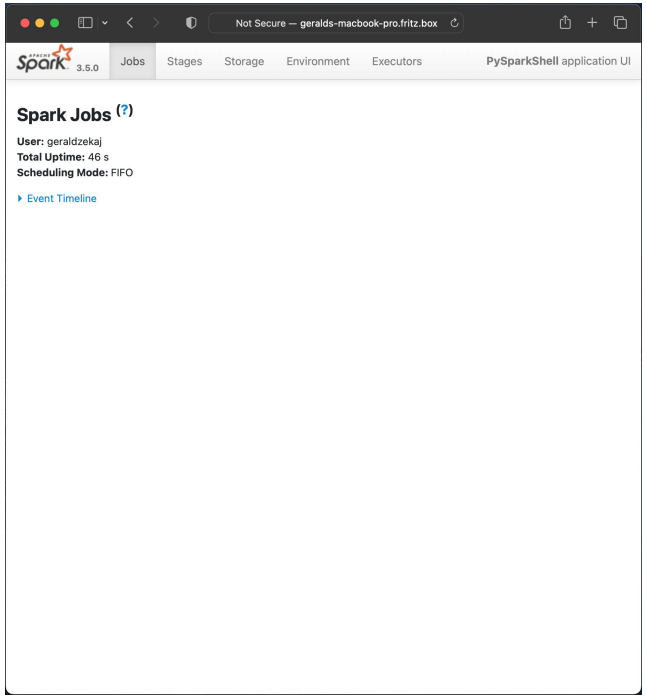
The Spark UI

Apache Spark provides a graphical user interface for monitoring applications at various stages. It can be found on the first message when you launch SPARK (http://mf0023.homenet.telecomitalia.it:4040).

The monitoring includes jobs, stages, and tasks.

Key Information Available

- **Scheduler stages and tasks**
- **Memory Usage**
- **Environment Details**
- **Executor Information**
- **Spark SQL Queries**



Code for Counting M&Ms

```python
import sys
from pyspark.sql import SparkSession
from pyspark.sql.functions import count

# Check if this script is the main program
if __name__ == "__main__":
    # Ensure that the correct number of command-line arguments are provided
    if len(sys.argv) != 2:
        print("Usage: python mnmcount.py <file>", file=sys.stderr)
        sys.exit(-1)

    # Create a SparkSession, which is the entry point to Spark functionality.
    # If a SparkSession doesn't exist, it will create a new one.
    spark = (SparkSession
             .builder
             .appName("PythonMnMCount")  # Assign a name to the Spark application
             .getOrCreate())

    # Retrieve the M&M data set filename from the command-line arguments
    mnm_file = sys.argv[1]

    # Read the specified file into a Spark DataFrame. We use the CSV format
    # and specify two options: "header" to indicate that the first row contains
    # column names and "inferSchema" to automatically detect data types.
    mnm_df = (spark.read.format("csv")
              .option("header", "true")
              .option("inferSchema", "true")
              .load(mnm_file))
```

```python
# We use the DataFrame high-level APIs to work with structured data. Note
# that we are exclusively using DataFrames and not RDDs (Resilient Distributed Datasets).
# In Spark, some functions return the same object, allowing us to chain function calls.

# Step 1: We select the fields "State," "Color," and "Count" from the DataFrame.
# Step 2: To group each state and its M&M color count, we use the groupBy() function.
# Step 3: We aggregate the counts of all colors and groupBy() "State" and "Color."
# Step 4: We order the results in descending order based on the "Total" count.
count_mnm_df = (mnm_df
    .select("State", "Color", "Count")
    .groupBy("State", "Color")
    .agg(count("Count").alias("Total"))
    .orderBy("Total", ascending=False))

# Show the resulting aggregations for all the states and colors;
# this displays the total count of each color per state.
# Note that "show()" is an action, which triggers the above query to be executed.
count_mnm_df.show(n=60, truncate=False)

# Print the total number of rows in the resulting DataFrame.
print("Total Rows = %d" % (count_mnm_df.count()))
```

```python
# Here we focus on a single state.

# Step 1: Select all rows from the DataFrame.
# Step 2: Filter only those rows where the state is "CA."
# Step 3: Group the data by "State" and "Color" just like before.
# Step 4: Aggregate the counts for each color.
# Step 5: Order the results in descending order.

ca_count_mnm_df = (mnm_df
    .select("State", "Color", "Count")
    .where(mnm_df.State == "CA")
    .groupBy("State", "Color")
    .agg(count("Count").alias("Total"))
    .orderBy("Total", ascending=False))

# Show the resulting aggregation for the state of California (CA).
# Similar to the previous example, the "show()" function is an action
# that will execute the computation and display the results.
ca_count_mnm_df.show(n=10, truncate=False)

# Stop the SparkSession.
spark.stop()
```

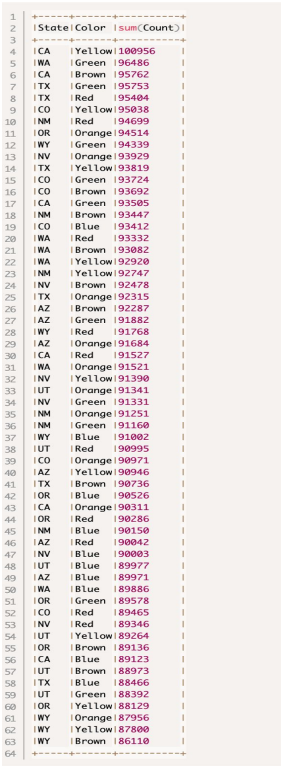
Counting M&Ms for the Cookie Monster

```
+-----+------+-----+
|State|Color |Count|
+-----+------+-----+
|TX   |Red   |20   |
|NV   |Blue  |66   |
|CO   |Blue  |79   |
|OR   |Blue  |71   |
|WA   |Yellow|93   |
+-----+------+-----+
only showing top 5 rows
```

Fig. 1

```
+-----+------+----------+
|State|Color |sum(Count)|
+-----+------+----------+
|CA   |Yellow|100956    |
|CA   |Brown |95762     |
|CA   |Green |93505     |
|CA   |Red   |91527     |
|CA   |Orange|90311     |
|CA   |Blue  |89123     |
+-----+------+----------+
```

Fig. 3



```
+-----+------+----------+
|State|Color |sum(Count)|
+-----+------+----------+
|CA   |Yellow|100956    |
|WA   |Green |96486     |
|CA   |Brown |95762     |
|TX   |Green |95753     |
|TX   |Red   |95404     |
|CO   |Yellow|95038     |
|NM   |Red   |94699     |
|OR   |Orange|94514     |
|WY   |Green |94339     |
|NV   |Orange|93929     |
|TX   |Yellow|93819     |
|CO   |Green |93724     |
|CO   |Brown |93692     |
|CA   |Green |93505     |
|NM   |Brown |93447     |
|CO   |Blue  |93412     |
|WA   |Red   |93332     |
|WA   |Brown |93082     |
|WA   |Yellow|92920     |
|NM   |Yellow|92747     |
|NV   |Brown |92478     |
|TX   |Orange|92315     |
|AZ   |Brown |92287     |
|AZ   |Green |91882     |
|WY   |Red   |91768     |
|AZ   |Orange|91684     |
|CA   |Red   |91527     |
|WA   |Orange|91521     |
|NV   |Yellow|91390     |
|UT   |Orange|91341     |
|NV   |Green |91331     |
|NM   |Orange|91251     |
|NM   |Green |91160     |
|WY   |Blue  |91002     |
|UT   |Red   |90995     |
|CO   |Orange|90971     |
|AZ   |Yellow|90946     |
|TX   |Brown |90736     |
|OR   |Blue  |90526     |
|CA   |Orange|90311     |
|OR   |Red   |90286     |
|NM   |Blue  |90150     |
|AZ   |Red   |90042     |
|NV   |Blue  |90003     |
|UT   |Blue  |89977     |
|AZ   |Blue  |89971     |
|WA   |Blue  |89886     |
|OR   |Green |89578     |
|CO   |Red   |89465     |
|NV   |Red   |89346     |
|UT   |Yellow|89264     |
|OR   |Brown |89136     |
|CA   |Blue  |89123     |
|UT   |Brown |88973     |
|TX   |Blue  |88466     |
|UT   |Green |88392     |
|OR   |Yellow|88129     |
|WY   |Orange|87956     |
|WY   |Yellow|87800     |
|WY   |Brown |86110     |
+-----+------+----------+
```

Fig. 2