- Easy Syntax

- **Dynamically Typed**: In Python, you don't have to declare the data type of a variable. The interpreter infers the type at runtime. This feature can speed up the development process, though it might cause runtime errors if not carefully handled.

- **Interpreted Language**: Python is an interpreted language, meaning that it executes code line by line, which aids in debugging. However, it also tends to be **slower** than compiled languages.

  In Python, a virtual environment is an isolated working copy of Python which allows you to work on a specific project without worrying about affecting other projects.

Setup.

Apache Spark is a unified computing engine and a set of libraries for distributed data processing. It was developed as a more efficient and capable successor to Apache Hadoop's MapReduce engine. Here are some of the key features of Apache Spark:

Speed: Spark is known for its speed. It can perform batch processing jobs from 10 to 100 times faster than MapReduce for a variety of workloads and up to 10 times faster on disk. This is mainly because it reduces the number of reads and writes to disk by keeping data in memory.

Ease of Use: Spark provides easy-to-use APIs for operating on large datasets. It supports APIs in Java, Scala, Python, and R, with Python and Scala being the most commonly used. It also supports over 100 high-level operators that make it easy to build parallel apps.

General-Purpose System: Unlike Hadoop, which is a batch processing system, Spark is a general-purpose computing engine. It supports batch processing, interactive queries (Spark SQL), streaming (Spark Streaming), machine learning (MLlib), and graph processing (GraphX), all within the same core computation engine.

Powerful Caching: Spark can hold intermediate data in memory or disk for quicker access during computation. This makes it particularly good at iterative algorithms, interactive data mining, and streaming tasks.

Fault Tolerance: Spark uses a data sharing model named Resilient Distributed Datasets (RDD) which is fault-tolerant. It can rebuild lost data using lineage information (a series of transformations that led to the RDD). This ensures that computations can continue even when individual nodes fail.

Real-Time Computation: Spark's capability to process real-time data makes it stand out from other big data processing frameworks. It can handle live data streams and perform real-time analytics.

Support for Complex Data Analysis: Spark not only supports 'Map' and 'Reduce' operations but also supports SQL queries, streaming data, and complex analytics such as machine learning and graph algorithms out-of-the-box.

Integration: Spark can easily be integrated with Hadoop and its modules. It can run on Hadoop clusters and process data stored in Hadoop's HDFS, as well as other data sources like HBase, Cassandra, etc.

Scalability: Spark can handle large amounts of data and can scale from a single server to thousands of machines.

Apache Spark and Hadoop MapReduce are both big data frameworks, but they differ in several ways. Here is a comparison of the two:

Data Processing Speed: Spark is known for its in-memory processing capabilities, which can significantly speed up the iterative part of computations like machine learning algorithms. It's often quoted as being able to process tasks 100 times faster in memory and 10 times faster on disk than MapReduce. MapReduce, in contrast, writes intermediary data back to disk, which can slow down computations significantly.

Ease of Use: Spark provides high-level APIs in Java, Scala, Python, and R and supports SQL queries, streaming data, and complex analytics such as machine learning and graph algorithms out-of-the-box. MapReduce primarily uses Java, and while other languages can be used (via Hadoop Streaming), they're often more challenging to work with.

Data Processing: MapReduce, as its name indicates, is built around two operations: Map and Reduce. Every job has to fit into this two-step paradigm. On the other hand, Spark provides more transformations and actions, allowing for more complex computations.

Fault Tolerance: Both Spark and MapReduce are fault-tolerant, meaning they can recover from failures. Spark achieves this through the use of RDDs (Resilient Distributed Datasets) that track data lineage information to rebuild lost data automatically. MapReduce writes intermediary data to disk, which makes it resilient against node failures.

Real-time Processing: Spark has the capability to handle live streams of data in real-time or near-real-time. This makes it well suited for use cases such as fraud detection in credit card transactions, where real-time processing is essential. MapReduce, on the other hand, is designed for batch processing and lacks the capability to process data in real-time.

General-Purpose Computation: Spark is a general-purpose computing engine, meaning it can handle a variety of data processing tasks within the same engine - batch processing, interactive queries, streaming, machine learning, and graph processing. MapReduce is specifically designed for batch processing tasks.

Resource Management: Both Spark and MapReduce can run on the YARN resource manager, allowing them to share resources with other Hadoop applications. However, Spark also comes with its standalone resource manager if YARN is not available.

Data Persistence: In Spark, users can persist an RDD in memory, allowing it to be reused across stages in a computation. This is particularly useful for iterative algorithms. In MapReduce, data must be stored in HDFS after each map and reduce operation, which can be less efficient.

Apache Spark has a master/worker architecture. It has a driver program (the master node), and many executor programs (worker nodes) that run across the cluster or locally on the same machine. Here's a brief explanation:

Driver Program: The driver program runs the main() function of the application and creates a SparkContext. The driver program defines one or more transformations and actions on the data and submits such operations to the cluster. The Driver Program also creates a SparkSession which acts as a single point of entry to interact with the underlying Spark functionality.

SparkContext: The SparkContext is a client of Spark’s execution environment and it acts as the master of the Spark application. When running with a cluster manager, SparkContext connects to it and acquires executors on nodes in the cluster. It sends your application code (defined by JAR or Python files passed to SparkContext) to the executors.

Cluster Manager: The Cluster Manager is an external service responsible for acquiring resources on the Spark cluster and allocating them to a Spark job. It can be one of several types, including Hadoop YARN, Apache Mesos, and the standalone Spark cluster manager.

Executors: Once connected, Spark acquires executors on nodes in the cluster, which are processes that run computations and store data for your application. Executors stay up for the duration of the Spark Application and run tasks in multiple threads across cores on a node. Executors communicate with the driver program and are responsible for executing the tasks assigned by the driver program.

Tasks: A task is a unit of work sent to one executor. The driver program splits a Spark application into a series of tasks that are distributed to executor nodes for execution.

Stages: Tasks are bundled together into stages. A stage is a group of tasks that all perform the same computation, but on different data subsets. Stages are the unit of scheduling in a Spark program.