Prepared by - PRIYANKA NEOGI

TOPICS COVERED :

1. PYTHON LANGUAGE FUNDAMENTALS
2. KEYWORDS
3. CASE SENSITIVE
4. CURRENT WORKING DIRECTORY
5. PACKAGE / LIBRARY
6. IMPORT
7. COMMENT
8. IDENTIFIERS
9. INDENTATION
10. STATEMENT
11. MULTILINE STATEMENT
12. VARIABLE DECLARATION

1. Python Language Fundamentals :

- Python is a general purpose high level programming language.
- Python is recommended as first programming language for beginners.
- **Guido developed Python language by taking almost all programming features from different languages :**

1. Functional Programming Features from C
2. Object Oriented Programming Features from C++
3. Scripting Language Features from Perl and Shell Script
4. Modular Programming Features from Modula-3
5. Most of syntax in Python Derived from C and ABC languages.

- Python is a versatile and high-level programming language that is suitable for a wide range of applications.
- It is particularly recommended as the first programming language for beginners due to its simplicity and readability.

Guido van Rossum, the creator of Python, designed the language by drawing inspiration from various programming languages:

1. **Functional Programming Features from C:**

Python incorporates functional programming concepts from the C programming language. Functional programming is a paradigm that treats computation as the evaluation of mathematical functions and avoids changing state and mutable data.

```
Think of a computer program as a recipe. In some recipes, you follow a
series of steps and, at the end, you have a delicious meal. Now,
```

```
imagine if you could only use ingredients once, and once you've used
them, you can't change them. In other words, once you've mixed the
ingredients, you can't unmix them, and you can't  add more or take any
away. You have to work with what you have.


So, in simple terms, functional programming in Python (and other
languages) means you work with your data     like it's a recipe that
you can't change. You use functions to create new results from the
original data,    much like a chef creates new dishes from ingredients
without altering the ingredients themselves. This     approach can
make your code more predictable and easier to understand.
```

Python's built-in functions are typically written in the C programming language. Python itself is implemented in C, and many of its core functionalities and built-in functions are part of the Python C API. These functions are highly optimized for performance and are an integral part of the Python language's standard library.

1. **Object-Oriented Programming Features from C++:**

Python inherits object-oriented programming features from C++. This means it supports the creation of classes and objects, which are fundamental concepts in object-oriented programming.


```
Building with Lego blocks. Each block can represent something, like a
car or a house. Now, in regular
play, just stack these blocks together however you want. You can
change them, add more, or take some away
at any time.

Object-oriented programming is like having special Lego instructions
that tell you exactly how to build a
specific thing using these blocks. Instead of changing the blocks, you
follow these instructions to
create objects, like a ready-made car or house. These objects are like
pre-built Lego sets, and they have
their own special features and abilities.

Python can use these Lego instructions to build things in a structured
way. It can create objects with
specific properties and actions, making it easier to work with complex
ideas and data in your code.
```

1. **Scripting Language Features from Perl and Shell Script:**

Python's scripting capabilities are influenced by languages like Perl and Shell Scripting. This means Python is well-suited for automating tasks and writing scripts to manipulate files and system resources.

```
Python, a popular programming language, has some features that it
borrowed from other languages like Perl
and Shell Scripting. These features make Python really good for
certain tasks.

1. Perl Influence:
Perl is known for its powerful text manipulation capabilities. Python
took some of
these abilities, so you can use Python to easily work with text in
files or data, like searching and
replacing text.

2. Shell Script Influence:
Shell scripts are used to automate tasks in the command-line interface
of a computer. Python learned from this
and can also be used for automating tasks, but with a more user-
friendly and versatile approach. This means
you can write Python scripts to make your computer do things
automatically, like moving files, processing data,
or managing system resources.

So, in simple terms, Python is
for automating tasks and working with files and your computer's
resources, thanks to its influences from Perl
and Shell Scripting. It makes it easier for people to write programs
that make their computer do what they
want without having to learn more complex languages or tools.
```

1. **Modular Programming Features from Modula-3:** Modular programming is a software design technique that promotes breaking code into manageable and reusable modules. Python adopted some of these concepts from Modula-3 to enhance code organization and maintainability.


```
**Modular Programming** is like building with LEGO blocks. It's a way
of organizing computer code into
smaller, reusable pieces, or "modules,".

Python, a popular programming language, learned some useful tricks
from another language called **Modula-3**
when it comes to organizing code.



1. **Breaking Code into Manageable Pieces:**
Instead of writing one giant block of code that's hard to understand,
Python encourages developers to split
their code into smaller, more manageable modules.

2. **Reusing Code:**
Just like you can use the same LEGO brick in different parts of your
```

```
LEGO creation, modular programming
allows developers to reuse their code modules in different parts of a
program. This makes it easier to build
and maintain software because you don't have to reinvent the wheel
every time you want to do something.

3. **Enhanced Organization:**
Think of modular programming as having different drawers for different
types of LEGO pieces. You can quickly
find the pieces you need because they're organized neatly. Similarly,
in Python, code is organized into
modules, making it easier to find and work with specific parts of the
program.

So, when we say Python adopted "Modular Programming Features from
Modula-3," we mean Python learned how to use the LEGO-like concept of
breaking code into pieces, reusing those pieces, and keeping
everything nicely organized. This makes it easier for programmers to
build and maintain software projects without everything turning into a
big, messy pile of code.
```

1. **Syntax Derived from C and ABC languages:** Python's syntax is inspired by both the C and ABC programming languages. This syntax choice contributes to Python's readability and ease of use.

```
Python, the programming language, learned some of its coding rules, or
"syntax," from two other languages:
C and ABC.



1. C Influence:
C is like the language that laid down some basic rules for writing
code. Python borrowed some of these
rules, making Python code look a bit like C code. This borrowing
helped Python become more powerful and
flexible.

2. ABC Influence:
ABC is like a language known for being simple and easy to read. Python
borrowed some of ABC's ideas to make
its code easy to understand and write. This is why Python is often
praised for being very readable.

So, when we say Python's syntax is "derived from C and ABC languages,"
we mean Python learned some helpful
coding rules from C to be versatile and from ABC to be easy to
understand. This combination makes Python
a friendly language for both beginners and experienced programmers
```

```
because it's like speaking a language
that's both powerful and clear.
```

Overall, Python's design philosophy emphasizes code readability and a clean, straightforward syntax, making it a popular choice for both beginners and experienced programmers across various domains. Its versatility and rich feature set make it a powerful tool for a wide range of programming tasks.

Where we can use Python:

1. **For developing Desktop Applications:** Python can be used to create software applications that run on desktop computers. It provides tools and libraries for creating graphical user interfaces (GUIs), making it suitable for developing applications with windows, buttons, and other interactive elements.

2. **For developing Web Applications:** Python is commonly used in web development. Frameworks like Django and Flask make it easier to build web applications, manage databases, and handle web requests, making Python a popular choice for web development.

3. **For developing Database Applications:** Python can connect to databases, retrieve and manipulate data, and create database-driven applications. This makes it useful for building software that interacts with databases, such as content management systems (CMS) and data analysis tools.

4. **For Network Programming:** Python has built-in libraries for network programming, which allows developers to create networked applications, manage network devices, and automate networking tasks.

5. **For Developing Games:** Python can be used in game development, particularly for creating 2D games. Libraries like Pygame provide game developers with the tools they need to build interactive and fun games.

6. **For Data Analysis Applications:** Python has a wide range of libraries and tools for data analysis, including pandas, NumPy, and Matplotlib. It is a popular choice for data scientists and analysts to analyze and visualize data.

7. **For Machine Learning:** Python's simplicity and a vast ecosystem of machine learning libraries, such as TensorFlow, Keras, and scikit-learn, make it a preferred language for developing machine learning models and applications.

8. **For Developing Artificial Intelligence Applications:** Python is widely used in artificial intelligence (AI) development. It is suitable for tasks like natural language processing, computer vision, and AI research.

9. **For IoT (Internet of Things):** Python is used in IoT projects to program and control devices like sensors, actuators, and microcontrollers. Its ease of use and support for various hardware interfaces make it a practical choice for IoT development.

In summary, Python's versatility and a vast ecosystem of libraries and frameworks make it a go-to language for a wide range of application areas, from web development to data analysis, machine learning, and even IoT projects. Its readability and ease of use also contribute to its popularity in these domains.

INDUSTRIES USING PYTHON :

1. **Google and YouTube:** Google has been known to use Python internally for various purposes. For instance, some of their web services and infrastructure components are built using Python. YouTube, which is owned by Google, also uses Python for certain tasks and services. However, they use a variety of languages and technologies across their vast ecosystem.

2. **NASA:** NASA has used Python for various scientific and data analysis tasks. Python's simplicity and a wide range of scientific libraries make it suitable for processing and analyzing data collected from space missions.

3. **New York Stock Exchange (NYSE):** While Python is not the primary language for trading platforms, it is used in financial institutions and trading firms for various purposes, including data analysis, risk assessment, and algorithmic trading.

4. **Top Software Companies:** Many top software companies, including Google, Microsoft, IBM, and Yahoo, use Python for different purposes. Python is often employed for web development, data analysis, machine learning, and automation tasks within these organizations.

It's important to note that these organizations often use a mix of programming languages and technologies to address their diverse needs. Python's popularity is attributed to its versatility, readability, and extensive libraries, making it a valuable tool for various applications. However, the specific usage of Python can vary between different projects and teams within these organizations.

Features of Python :

1. Simple and easy to learn:

Python is known for its simplicity as a programming language. When you read a Python program, it often feels like you're reading plain English sentences.

The syntax is straightforward, with only around 30 keywords available.

Compared to other programming languages, Python allows you to write programs with fewer lines of code, which enhances readability and simplicity.

This reduction in code length can also lead to cost savings in project development.

2. Freeware and Open Source:

1. **No License Required:** You don't need to pay for a license or any fees to use Python. It's freely available to anyone who wants to use it. This means you can download and use Python on your computer without any cost.

2. **Open Source:** Python is an open-source software. This means that the source code, which is the "recipe" that makes Python work, is freely available to the public. You can see how Python is built, and you're allowed to modify and customize it to suit your needs.
    - **Example:** Jython is a great example. It's a customized version of Python that's made to work seamlessly with Java applications. This customization was possible because Python's source code is open. Developers could adapt it to fit the specific requirements of working with Java.

3. High Level Programming language:

Python is considered a **high-level programming language**, and this comes with certain advantages, particularly for programmers. A "high-level language" is like a way for humans to talk to computers more easily. It's a bridge that helps us communicate with the computer without having to speak its complicated language directly.

1. **Programmer-Friendly:** Python is designed to be user-friendly for programmers. It emphasizes readability and simplicity, so when you write Python code, it often looks a lot like human-readable language, which makes it easier to understand and write.
2. **Abstraction of Low-Level Tasks:** One of the key benefits of a high-level language like Python is that it abstracts away many low-level activities that programmers would otherwise have to worry about. Two common examples are memory management and security:
    - **Memory Management:** In lower-level languages like C or C++, programmers need to manually allocate and deallocate memory for variables and data structures. In Python, this is handled automatically by the language itself, relieving the programmer from the responsibility of managing memory.
    - **Security:** Python has built-in security features and libraries that help protect against common security vulnerabilities. Programmers can focus on writing their code without needing to implement low-level security measures from scratch.

So, when you're using Python, you can concentrate on solving the specific problem you're working on, rather than getting bogged down in the nitty-gritty details of memory management and security.

4. Platform Independent:

Now, the interesting thing about Python is that once you've written your recipe (program), it can be used on different types of stoves (computers) without needing to change the recipe.

1. ** Platform Independence:** Python is designed to be "platform-independent," which means it doesn't care what kind of computer it's running on. Your Python program will work on Windows, Mac, or any other computer without needing to be rewritten.

2. **PVM - Python Virtual Machine:** Inside your computer, there's a special part called the Python Virtual Machine (PVM). Think of the PVM as a magical chef who takes your recipe and adjusts it to work perfectly on the specific stove (computer) it's running on. It does all the translation and conversion so that your Python program runs smoothly.

So, in simple terms, Python is like writing a universal recipe that can be cooked on any type of stove thanks to the Python Virtual Machine (PVM). You don't need to rewrite the recipe for each stove; the PVM takes care of it, making Python programs very versatile and easy to use across different computers.

5. Portability:

It works perfectly on any platform, whether it's your computer or a different computer.

1. **Portability:** Python programs are easy to carry around. You can take your Python program from one type of computer (like a PC) to another type (like a Mac) without much trouble. You don't need to change it; it just works.

2. **Consistency:** Python programs provide the same results no matter which type of computer or platform you use. This consistency is one of Python's strengths.

6. Dynamically Typed:

In Python, you don't need to explicitly specify the data type of a variable when you create it. This is in contrast to some other programming languages where you must declare the data type of a variable before using it.

In Python, the data type is determined automatically based on the value assigned to the variable. This is why Python is often referred to as a "dynamically typed" language.

1. **Type Inference:** When you assign a value to a variable in Python, the interpreter looks at the value and figures out what data type it is. For example, if you assign a number like `x = 5`, Python knows that `x` is an integer (int).

2. **Automatic Type Allocation:** Python then associates this data type information with the variable `x`. So, in this case, `x` is now of type integer. If you later assign a string to `x`, like `x = "Hello"`, Python will dynamically change the type of `x` to string (str).

```
x = 5   # x is an integer
x = "Hello"  # now x is a string
```

This dynamic type allocation makes Python flexible because you can use the same variable for different types of data as needed. However, it also requires careful coding to avoid unexpected type-related issues.

7. Both Procedure Oriented and Object Oriented:

Python is like a versatile toolbox for programmers. In this toolbox, it has tools for two different ways of building programs: **Procedure-Oriented** and **Object-Oriented**.

1. **Procedure-Oriented:** Imagine building a program like building a car. In the procedure-oriented way, you focus on each part of the car separately: the wheels, the engine, the seats, and so on. It's like putting together all the pieces step by step. This way is good for simple tasks and following a clear sequence of steps.

2. **Object-Oriented:** Now, think of building a program like creating a robot. In the object-oriented way, you think about the whole robot as one unit with its own behavior and abilities. You don't worry too much about how the robot's insides work; you just know it can do certain things. This way is good when you want to organize and reuse pieces of code, like having a library of robot parts that you can use in different robots.

Python gives you both of these ways to build programs. So, if you need to build a car, you can use the procedure-oriented tools. If you want to create a robot, you can use the object-oriented tools.

8. Interpreted:

1. **No Explicit Compilation:** In some programming languages like C or Java, you need to go through a process called compilation before you can run your code. It's like translating your code into a language the computer understands.
- However, in Python, you don't have to do this extra step. You can write your Python program directly without explicitly compiling it.
1. **Python Interpreter:** Instead of compiling, Python uses an interpreter.
- Think of the interpreter as a real-time translator for your code. When you write a Python program, the interpreter reads and executes your code line by line.
- It's like having a friend who speaks both English and another language (computer language) and can translate for you on the fly.
1. **Syntax Errors:** If you make a mistake in your Python code, like using the wrong word in a sentence, the interpreter will catch it and tell you there's an error. These are called syntax errors, and they're like grammar mistakes in your code.

2. **Python Virtual Machine (PVM):** Once the interpreter checks your code and finds it free of syntax errors, it hands it over to the Python Virtual Machine (PVM).

- The PVM is like the engine that makes your program run. It takes your Python code and executes it, making your program do whatever it's supposed to do.

So, in a nutshell, with Python, you write your code, the interpreter checks it for language errors, and if everything is okay, the Python Virtual Machine runs your program. You don't have to explicitly compile your code before running it, which makes Python code development quicker and more straightforward.

9. Extensible:

Python allows you to use programs written in other programming languages, and this approach has two significant advantages:

1. **Using Existing Legacy Code:** Sometimes, you may have valuable software components or libraries written in languages other than Python, often referred to as "legacy code."
- These components might be old, well-tested, and perform specialized tasks. By using Python's ability to integrate with other languages, you can incorporate and reuse this existing code within your Python programs.
- It's like adding tried-and-true ingredients to a new recipe, saving you time and effort in building new functionality.
1. **Performance Improvement:** Python is known for its simplicity and ease of use, but it may not always be the fastest language for certain computational tasks.
- In cases where performance is critical, you can use Python's integration with other languages to call highly optimized and efficient code written in languages like C or C++.
- This allows you to harness the speed of these languages while still benefiting from Python's high-level features for other parts of your application. It's like having a powerful sports car engine inside a comfortable and user-friendly car.

In essence, Python's ability to work with other languages enables you to leverage existing code assets and optimize performance where needed, making it a flexible and practical choice for a wide range of software development projects.

10. Embedded:

Python is like a versatile tool that you can use alongside other tools in your toolbox (other programming languages). It's not just a standalone tool; it can be embedded or used within programs written in other languages.

1. **Embedding Python:** Imagine you have a big project that's built with a different programming language like C++, Java, or C#.
- You can think of Python as a special tool that you can place inside your existing project. You can use Python code to perform specific tasks or calculations within your larger project.
1. **Integration:** Python can be seamlessly integrated into other programming languages.
- So, you can have a part of your program written in Python and another part in a different language. They can work together as if they speak the same language.
- It's like having two machines that can communicate and collaborate effectively.
1. **Versatility:** Python's ability to be embedded makes it extremely versatile.
- You can use Python for tasks like data analysis, scripting, or automation, even within projects primarily written in other languages.
- It's like having a multitool that fits perfectly into your other tools, making your work more efficient and flexible.

In summary, Python's versatility allows you to embed it within programs written in other languages, making it a valuable addition to your toolkit for a wide range of tasks, whether you're building large software projects or performing specific tasks within existing codebases.

11. Extensive Library:

Python comes with a treasure trove of tools and functions built right into it. These are like a vast collection of ready-made tools in a toolbox that are available for programmers to use.

1. **Rich Inbuilt Library:** Python has a large library of functions and modules that cover a wide range of tasks.
- These built-in functions can do things like handling files, working with data, connecting to the internet, and much more.
- Think of it as a library of pre-made functions that you can use in your programs.

1. **Direct Usage:** As a programmer, you don't have to reinvent the wheel.
- Instead of writing complex code from scratch to do common tasks, you can simply use these pre-built functions.
- It's like having access to ready-made tools for specific jobs, so you don't have to create them yourself.

1. **Saves Development Time:** This rich library saves you a lot of time and effort.
- You can focus on solving the unique challenges of your project, while Python's built-in functions handle routine tasks.
- It's like having an assistant who takes care of the everyday chores, leaving you free to work on the more important aspects of your program.

So, in essence, Python's rich built-in library is like having a vast collection of tools and helpers that programmers can use directly, without having to build everything from scratch. It simplifies programming and accelerates development by providing ready-made solutions for common tasks.

LIMITATIONS:

1. **Performance:** Python, being an interpreted language, can sometimes be slower compared to languages that are compiled into machine code.
- Python code is executed line by line, and this interpretation process can make it less efficient for certain high-performance tasks.
- However, Python offers ways to mitigate this issue, such as using optimized libraries and integrating with lower-level languages like C or C++ when performance is critical.

1. **Mobile Applications:** Python isn't as commonly used for mobile app development as languages like Java (for Android) or Swift (for iOS).
- While there are frameworks like Kivy and BeeWare that allow you to build mobile apps with Python, it's not the first choice for mobile development. Native app development on Android and iOS often requires languages and tools specifically designed for those platforms.

In summary, while Python is a versatile and powerful language, it may not be the best choice for all scenarios. It may face performance challenges for certain tasks, and it's not the primary choice for native mobile app development. However, it excels in many other areas and remains a popular choice for a wide range of applications.

PYTHON VERSIONS :

Python is continuously evolving, and there are different versions of Python available. Python 2 and Python 3 are two major versions of Python, and they have some differences.

One key thing to note is that Python 3 was designed to be an improvement over Python 2, but it made some changes that are not backward compatible.

1. **Python 2 and Python 3:** Python 3 introduced some improvements and changes to the language. While these changes were made to enhance Python's clarity and consistency.
- They also meant that some Python 2 code would not work in Python 3 without modifications.
1. **Lack of Guarantee:** Python's developers made a clear decision not to provide backward compatibility between Python 2 and Python 3.
- This means that there is no guarantee that Python 2 programs will run smoothly in Python 3 without making necessary updates and changes to the code.
- The transition from Python 2 to Python 3 may require modifications to adapt to the differences in the language.

In summary, if you have code written in Python 2, it may not work as-is in Python 3. You may need to update and modify your code to make it compatible with Python 3. This transition was made to improve the language, but it does require effort from developers to ensure their code works with the latest version of Python.

2. KEYWORDS :

1. Definition:
- Keywords in Python are reserved words that cannot be used as ordinary identifiers. They are used to define the syntax and structure of the Python language.
- Keywords in Python are specific words that have been reserved for exclusive use by the language itself.
- They cannot be freely employed anywhere within your code, whether as variable names or function names.
- Python has around 35 such keywords in Python 3.9, each with its own predefined functionality.

1. Immutability:
- Keywords are immutable. This means their meaning and definition can't be altered.
- The key words cannot be changed through any coding means.
1. Case Sensitivity:
- Keywords are case-sensitive. For example, `True` is a valid keyword, but true is not.
1. Total Number:
- As of Python 3.9, there are 35 keywords.
1. List of Keywords:
- The complete list of Python keywords are `False, None, True, and, as, assert, async, await, break, class, continue, def, del, elif, else, except, finally, for, from, global, if, import, in, is, lambda, nonlocal, not, or, pass, raise, return, try, while, with, yield`.
1. Special Keywords:
- async and await are used for handling asynchronous processing, and they became keywords in Python 3.7.
1. Usage:

- Each keyword has a specific meaning and usage in Python programming. For instance, `def` is used for defining functions, `if` is used for making conditional statements, `for` and while are used for loops, `class` is used for defining a class, and so on.
- One of the keywords can be `class`, which may belong to the `global` type category which is also a keyword. These are the specific keywords.

1. Identifying Keywords:

- You can get the list of all keywords in Python by using the following code:

```
# keyword is a package to check the keywords in the system.
import keyword

# Printing Keywords
print(keyword.kwlist)
```

```
['False', 'None', 'True', 'and', 'as', 'assert', 'async', 'await',
'break', 'class', 'continue', 'def', 'del', 'elif', 'else', 'except',
'finally', 'for', 'from', 'global', 'if', 'import', 'in', 'is',
'lambda', 'nonlocal', 'not', 'or', 'pass', 'raise', 'return', 'try',
'while', 'with', 'yield']
```

```
# Declaring a variable using keyword
None = 10
```

```
  Cell In[6], line 2
    None = 10
    ^
SyntaxError: cannot assign to None
```


- throws error: cannot assign to None , because `None` is a keyword and it has a specific use. Here keyword is assigned a value. A value can't be assigned to a keyword.

3. PYTHON IS CASE SENSITIVE LANGUAGE :

- Case sensitivity implies that uppercase and lowercase letters are treated as distinct and distinct from one another.

```
# taking the above example of None keyword
none = 10
print(none)
```

```
10
```

- No errors will occur now because all the characters in `none` used here are in lowercase, and case sensitivity is observed.
- There are no errors because the keyword `None` begins with an uppercase "N," and case sensitivity

```
# Incorrect use of keywords as variable names

class = 10     # 'class' is a reserved keyword
return = 5     # 'return' is a reserved keyword
if = "hello"   # 'if' is a reserved keyword
else = 3.14    # 'else' is a reserved keyword

print(class, return, if, else)
```

```
  Cell In[8], line 3
    class = 10     # 'class' is a reserved keyword
          ^
SyntaxError: invalid syntax
```

- In this context, all the variable names are reserved keywords in Python, which is an incorrect usage of keywords.

```
# Correct use of variable names.

class_number = 10
return_value = 5
if_string = "hello"
else_number = 3.14

print(class_number, return_value, if_string, else_number)
```

```
10 5 hello 3.14
```


- In this case, the variable names are not keywords, and they are used correctly without any issues.
- In this scenario, you are modifying the name of a specific keyword and making it a variable.

4. CURRENT WORKING DIRECTORY :

```
%pwd
```

```
'/Users/priyankaneogi/Desktop'
```

5. PACKAGE / LIBRARY :

A package can be likened to a container, within which numerous tools are stored. In the context of Python, packages serve as boxes filled with various functions.

For instance, consider a package named "math." Inside this package, you can find numerous tools such as addition, subtraction, and exponentiation, and there can be an unlimited number of tools like mean, average, and more. These tools are `functions` inside a package.

A package can encompass numerous functions. Take, for instance, the "math" package, which houses various functions like addition, subtraction, mode, median, etc

Question 1: Why is Python so successful in Data Science?

Python's success in data science is attributed to its extensive collection of packages. These packages essentially act as containers that house a multitude of tools, which are functions designed to perform specific tasks.

6 . IMPORT:

- "Importing a package loads it into my computer's RAM memory.

Question1: If the package is already installed, why do we need to load it into memory again?

When a package is installed, it's essentially stored on the hard drive, not in our RAM. Therefore, we have to explicitly call the package into our memory.

Question: Why can't packages be automatically present in RAM without requiring us to manually write 'import' followed by the package name?

In Python, we have a vast number of packages, let's say totaling 10 GB in size. Loading all of these packages into memory simultaneously would consume a significant amount of memory. This would be inefficient and wasteful because we typically use only a subset of these packages for a specific code. Keeping all packages in RAM would not only waste memory but also incur additional costs, as RAM comes at a price. Therefore, we should only load the packages that are necessary for the program.

In the data industry, where data sets can be enormous, efficient memory usage is crucial to avoid wastage. Hence, we only load the packages needed for a particular task." This results in a reduction of coding time since these functions already encompass the necessary tasks for completion, thereby also saving code length.

Question 2: What is the advantage of a package? Advantages of using packages include:

1. **Code Reduction:** Packages allow you to reduce the amount of code you need to write. Instead of recreating functions from scratch, you can leverage pre-built functions within packages, saving time and effort.
2. **Time Reduction:** It minimizes the coding duration.This results in a reduction of coding time since these functions already encompass the necessary tasks for completion, thereby also saving code length.

```
# package name - keyword
import keyword

# Package keyword contains a function 'kwlist'
print(keyword.kwlist)
```

7. COMMENTS :

- `#` Hash is used for commenting.

- Python treats comments as plain text and the Python interpreter disregards them when executing the code.
- Comments serve the purpose of explaining specific lines or pieces of code to others.
- They are a valuable tool for organizing and enhancing the comprehensibility of your code.
- Definition: A comment in Python is a piece of text in your code that is not executed. It's typically used to explain what the code is doing or leave notes for developers who will be reading or maintaining the code.

1. **Single-Line Comments:** Python uses the hash symbol (#) to denote a comment. Any text following the # on the same line will be ignored by the Python interpreter.
2. **Multi-Line Comments:** Python does not have a specific syntax for multi-line comments. Developers typically use a single # for each line to create multi-line comments.

- Alternatively, multi-line strings using triple quotes (''' or """) can also be used as multi-line comments because any string not assigned to a variable is ignored by Python.

1. Docstrings or documentation strings: Docstrings are a type of comment used to explain the purpose of a function or a class.

- They are created using triple quotes and are placed immediately after the definition of a function or a class.
- Docstrings can span multiple lines and are accessible at runtime using the .__doc__ attribute.


```
# This is a comment in Python
x = 5  # This is an inline comment

# This is a multi-line comment
# We are adding two numbers a and b.
a = 5
b = 3
c = a + b
print(c)
```

```
8
```

```
# Docstring Example
def add_numbers(a, b):
    """
    This function adds two numbers and returns the result.

    Parameters:
    a (int): The first number
    b (int): The second number

    Returns:
```

```
    int: The sum of the two numbers
    """
    return a + b

# You can access this docstring using the .__doc__ attribute.
# Here's how:
print(add_numbers.__doc__)


    This function adds two numbers and returns the result.

    Parameters:
    a (int): The first number
    b (int): The second number

    Returns:
    int: The sum of the two numbers
```

8. IDENTIFIERS :

Here are the key points related to identifiers in Python:

- **Definition:**
- An identifier is a name given to entities like class, functions, variables, etc. It helps to differentiate one entity from another.
- An identifier is a name assigned to an element or entity, such as a variable name, class name, or function name.

Rules for writing Identifiers :

- **Syntax:** Identifiers can be a combination of letters in lowercase (a to z) or uppercase (A to Z) or digits (0 to 9) or an underscore (_).
- **No digits:** They must start with a letter or the underscore character, but not with a digit.
- **Case-Sensitive:** Identifiers in Python are case-sensitive. For example, myVariable and myvariable are two different identifiers in Python.
- **No Special Characters:** Identifiers cannot have special characters such as

```
!, @, #, $, %, etc.
```

- **Reserved Words:** Python keywords cannot be used as identifiers. Words like for, while, break, continue, in, elif, else, import, from, pass, return, etc. are reserved words. You can view all keywords in your current version by typing help("keywords") in the Python interpreter.

- **Unlimited Length:** Python does not put any restriction on the length of the identifier. However, it's recommended to keep it within a reasonable size, to maintain readability and simplicity in the code.
- **Private Identifiers:** In Python, if the identifier starts with a single underscore, it indicates that it is a non-public part of the class, module, or function. This is just a convention and Python doesn't enforce it. If it starts with two underscores, it's a strongly private identifier. If the identifier also ends with two trailing underscores, the identifier is a language-defined special name.
- **Non-ASCII Identifiers:** Python 3 allows the use of non-ASCII letters in the identifiers. This means you can use letters like é, ñ, ö, я, etc. in your identifiers if you wish.

```
# 'marks' is a variable name storing the value 10.
marks = 10

# print is a function
print(marks)
```

```
10
```

```
# Python is case sensitive 'marks' in UPPERCASE
# 'marks' and 'MARKS' are different
MARKS = 20
print(MARKS)
```

```
20
```

```
# Identifier should not start with a number
10marks = 10
print(10marks)
```

```
  Cell In[13], line 2
    10marks = 10
     ^
SyntaxError: invalid decimal literal
```

```
_1a = 10
_1b = 20
c = _1a + _1b
print(c)
```

```
30
```

```
# Incorrect use of identifiers

1variable = 10      # Identifier starts with a digit
$second = 20        # Identifier contains special character
third variable = 30 # Identifier contains a space
```

```
for = 40            # Identifier is a reserved keyword

print(1variable, $second, third variable, for)
```

```
  Cell In[6], line 3
    1variable = 10      # Identifier starts with a digit
    ^
SyntaxError: invalid decimal literal
```

```
# Correct use of identifiers

variable1 = 10
second_variable = 20
third_variable = 30
variable_for = 40

print(variable1, second_variable, third_variable, variable_for)
```

```
10 20 30 40
```

IDENTIFIER NOTE :

1. **Private Identifier with a Single Underscore (_):**

- In Python, if an identifier (like a variable or a function name) starts with a single underscore, It's a convention that signals to other programmers that this identifier is intended for internal use within the module or class.
- It's like a gentle way of saying, "This is meant to be private, so be careful when using it, but I won't stop you." However, Python doesn't enforce strict privacy like some other languages, so it's more of a naming convention.

1. **Strongly Private Identifier with Double Underscores (__):**

- When an identifier starts with double underscores, it's a way of making it "strongly private."
- This means it's a signal to other programmers that they should not access this identifier directly from outside the class or module where it's defined.
- It's a stronger hint that you should avoid touching it unless you really know what you're doing. It's like saying, "Stay away unless you have a very good reason."

1. **Magic Methods with Double Underscores at Both Ends (magic):**

- In Python, identifiers that start and end with double underscores, like `__init__` or `__str__`, are special names defined by the Python language itself.
- These are often referred to as "magic methods" or "dunder methods" (short for "double underscore").
- These methods have specific purposes and are automatically called by Python in certain situations.
- For example, `__init__` is called when an object is created, and `__str__` is used to represent an object as a string when you print it. Python automatically calls the `__str__` method to represent the object as a string.

- These magic methods provide customization points for developers to define how objects of a class should behave in specific situations.

So, in summary, underscores in Python identifiers are used to convey naming conventions and privacy levels, while double underscores at both ends are reserved for special names that have predefined meanings in the language, allowing you to customize certain aspects of your classes and objects.

9. Indentation:

Importance:

- In Python, indentation is not just for readability. It's a part of the syntax and is used to indicate a block of code.
- Space Usage: Python uses indentation to define the scope of loops, functions, classes, etc. The standard practice is to use four spaces for each level of indentation, but you can use any number of spaces, as long as the indentation is consistent within a block of code.
- Colon: Usually, a colon (:) at the end of the line is followed by an indented block of code. This is common with structures like if, for, while, def, class, etc.


```
def say_hello():
    print("Hello, Eeveryone!")

say_hello()
```

```
Hello, Eeveryone!
```

```
# Incorrect use of indentation

if True:
print("This is True!") # Error: expected an indented block

for i in range(3):
print(i) # Error: expected an indented block

def hello():
print("Hello, World!") # Error: expected an indented block

while False:
print("This won't print") # Error: expected an indented block
```

```
  Cell In[10], line 4
    print("This is True!") # Error: expected an indented block
    ^
IndentationError: expected an indented block after 'if' statement on
line 3
```

```
# Correct use of indentation

if True:
    print("This is True!")

for i in range(3):
    print(i)

def hello():
    print("Hello, World!")

while False:
    print("This won't print")
```

```
This is True!
0
1
2
```

10. Statements :

Definition:

- A statement in Python is a logical instruction that the Python interpreter can read and execute. In general, a statement performs some action or action.

Types:

- Python includes several types of statements including assignment statements, conditional statements, looping statements, etc.


```
# Conditional Statement
x = 2
if x > 0:
    print("Positive number")
```

```
Positive number
```

```
#
for i in range(5):
    print(i)
```

```
0
1
2
3
4
```

11. Multi-Line Statements:

- In Python, end of a statement is marked by a newline character.

- But we can make a statement extend over multiple lines with the line continuation character (), or within parentheses (), brackets [], braces {}, or strings.
- You can also write multiple statements on a single line using semicolons (;)

```
# Multi-Line Statements
# Using line continuation character
s = 1 + 2 + 3 + \
    4 + 5

# Using parentheses
s = (1 + 2 + 3 +
     4 + 5)

# Multiple Statements on a Single Line
x = 5; y = 10; print(x + y)
```

```
15
```

12. VARIABLE DECLARATION:

1. **Variable Names:**
   - Variable names can consist of letters (both uppercase and lowercase), digits, and underscores.
   - They must start with a letter (a-z, A-Z) or an underscore (_) followed by letters, digits, or underscores.
   - **Valid variable names:** `my_var`, `_value`, `temp123`, `data_file_2`
   - **Invalid variable names:** `123abc` (starts with a digit), `my-var` (contains a hyphen), `@special` (contains special characters)
2. **Case Sensitivity:**
   - Python is case-sensitive, which means that uppercase and lowercase letters are considered distinct.
   - So, `myVar` and `myvar` would be treated as two different variables.
3. **Reserved Keywords:**
   - You cannot use reserved keywords (also known as keywords or reserved words) as variable names because they have special meanings in Python.
   - Examples of reserved keywords include `if`, `else`, `while`, `class`, `def`, and many others. Attempting to use a reserved keyword as a variable name will result in a syntax error.
4. **Convention:**
   - While not a strict rule, it's a common convention in Python to use lowercase letters and underscores for variable names (e.g., `my_variable`, `count_of_items`). This is known as "snake_case" and is widely adopted in Python code to enhance readability.
5. **Descriptive Names:**

- Choose meaningful and descriptive variable names that convey the purpose of the variable. This makes your code more understandable to others (and to yourself) and is considered good programming practice.

Here are some variable declaration examples that adhere to these rules:

```
my_variable = 42
count_of_items = 10
user_name = "John"
```

Remember that following these rules and conventions not only helps avoid errors but also makes your code more maintainable and easier to understand for both you and others who may read your code.

```

```