Chapter 1:

Python is a versatile and beginner-friendly programming language known for its simplicity and readability. Whether you're new to programming or transitioning from another language, Python is an excellent choice to begin your journey. Here's how you can get started:

1.1 Install Python:

- Python is available for various operating systems including Windows, macOS, and Linux. You can download the latest version of Python from the official website (https://www.python.org/downloads/).
- Follow the installation instructions provided on the website to install Python on your system.
- Make sure to check the option to add Python to your system's PATH during the installation process. This allows you to run Python from the command line.

Choose a Text Editor or IDE:

- You'll need a text editor or an Integrated Development Environment (IDE) to write Python code.
- Popular choices include Visual Studio Code, PyCharm, Sublime Text, Atom, and IDLE (Python's built-in IDE).
- Choose the one that best suits your preferences and requirements.

Write First Python Program:

- Open your text editor or IDE and create a new file with a .py extension (e.g., hello.py).
- Write a simple Python program, such as a "Hello, World!" program, to get started:

```
# hello.py

print("Hello, World!")
```

Run Your Python Program:

- Open a terminal or command prompt and navigate to the directory where your Python file is saved.
- Type python hello.py and press Enter to run the program.
- You should see the output "Hello, World!" printed to the terminal.

Verify if Python is installed

To verify if Python is installed on your system, you can follow these steps:

Open a Terminal or Command Prompt:

- On Windows: Press Win + R, type cmd, and press Enter.
- On macOS: Press Cmd + Space to open Spotlight Search, type Terminal, and press Enter.
- On Linux: Press Ctrl + Alt + T to open a Terminal window.

1.2 Check Python Version:

- Type the following command and press Enter:

python –version

If Python is installed, this command will display the version number. For example: Python 3.9.6.

Check Python Interpreter:

- Type the following command and press Enter:
- python

If Python is installed, this command will launch the Python interpreter, and you'll see a prompt >>>

Exit Python Interpreter:

- To exit the Python interpreter, you can type exit() and press Enter, or press Ctrl + D (on macOS/Linux) or Ctrl + Z (on Windows).

If Python is installed, you'll see the version number displayed when checking the version, and you'll be able to access the Python interpreter. If Python is not installed, you'll likely receive an error message indicating that the command is not recognized.

1.3 How to create Hello, World program in IDLE in python

- Open IDLE:
- Open the IDLE application. You can find it in your system's applications menu or by searching for "IDLE" in the search bar.

Create a New File:

- Once IDLE is open, click on the "File" menu at the top-left corner.
- Choose "New File" from the dropdown menu or press Ctrl + N (Windows/Linux) or Cmd + N (Mac).

Write the Python Code:

- In the new file window that appears, type the following code:

```
print("Hello, World!")
```

Run the Program:

- After saving the file, go back to the file window where you wrote the code.
- Click on the "Run" menu at the top of the window.
- Choose "Run Module" from the dropdown menu or press F5.

View the Output:

- You should see the output "Hello, World!" printed in the Python Shell window, which is located at the bottom of the IDLE window.

1.4 Hello World Python file

- Open a text editor such as Notepad on Windows, TextEdit on macOS, or any code editor of your choice.
- Copy and paste the Python code into the editor.

```
# hello_world.py

# Print "Hello, World!" to the console
print("Hello, World!")
```

- Save the file with the name hello_world.py (make sure to include the .py extension).
- Open a terminal or command prompt.
- Navigate to the directory where you saved hello_world.py using the cd command.
- Type python hello_world.py and press Enter to execute the program.

You should see the output Hello, World! printed to the console

1.5 Launch an interactive Python shell

follow these steps for interactive Python shell

- Open Terminal or Command Prompt:

  Open the Terminal on macOS or Linux, or Command Prompt on Windows.

- Type the Command:

  Once you have the terminal or command prompt open, simply type python and press Enter.

  If you're using Python 3, you might need to type python3 instead of python.

- Start Interacting:

After you press Enter, you should see the Python prompt >>>, indicating that you are now in the Python interactive shell.

You can now start typing Python code directly into the shell and see the results immediately.

For example:

```
>>> print("Hello, World!")

Hello, World!
```

Exit the Shell:

To exit the Python shell, you can type exit() and press Enter, or press Ctrl + D (on macOS/Linux) or Ctrl + Z (on Windows).

1.6 Creating variables and assigning values in python

In Python, you can create variables and assign values to them using a straightforward syntax. Here's how you do it:

```python
# Assigning values to variables
name = "Ajay"
age = 30
is_student = False
height = 5.9

# Printing the values of variables
print("Name:", name)
print("Age:", age)
print("Is student?", is_student)
print("Height:", height)
```

Explanation:

- In the above example, four variables are created: name, age, is_student, and height.
- The = sign is used to assign values to the variables.
- Variables in Python do not require explicit declaration of data types. Python dynamically determines the data type based on the value assigned.
- You can assign values of different data types such as strings, integers, booleans, and floating-point numbers to variables.
- Finally, the print() function is used to display the values of the variables.

Output:

```
Name: Ajay

Age: 30

Is student? False

Height: 5.9
```

1.7 Block Indentation

In Python, block indentation is a fundamental aspect of the language syntax used to denote the structure of code blocks. Unlike many other programming languages that use curly braces {} or keywords like begin and end to indicate block structure, Python uses indentation.

Here's how block indentation works in Python:

Indentation Level:

- Python uses consistent indentation to define blocks of code.
- Each level of indentation typically consists of four spaces, although tabs can also be used. However, it's recommended to use spaces for indentation to ensure consistency across different environments.

Code Blocks:

- Code blocks such as function definitions, conditional statements (if-elif-else), loops (for, while), and class definitions are denoted by indentation.
- All statements within the same block must have the same level of indentation.

Example:

```
def greet(name):
   if name == 'AJAY':
      print('Hi, AJAY!')
   else:
      print('Hello, ' + name)
```

In this example:

- The def greet(name): line denotes the beginning of a function definition. The subsequent lines are indented to indicate that they belong to the function block.
- The if name == 'AJAY': line denotes the beginning of an if statement block. The subsequent line (print('Hi, AJAY!')) is indented to indicate that it is part of the if block.
- The else: line denotes the beginning of the else block, which is also indented.

End of Blocks:

- The end of a block is indicated by returning to a previous level of indentation.
- In the example above, the end of the function block is indicated by the return to the outer indentation level.

Consistency:

- Consistent indentation is crucial for Python code to be syntactically correct.
- Mixing tabs and spaces for indentation can lead to errors, so it's best to stick to one method throughout your codebase.

Block indentation is one of the features that contribute to Python's readability and clarity. By adhering to consistent indentation practices, you can ensure that your code is easy to understand and maintain.

1.8 **Datatypes in python**

In Python, data types represent the type or category of data that can be stored and manipulated within a program. Python supports several built-in data types, each serving a different purpose. Here are some commonly used data types in Python:

Numeric Types:

- int: Integer data type represents whole numbers, positive or negative, without any decimal point.
- float: Float data type represents floating-point numbers, which include decimal points.

- complex: Complex data type represents complex numbers in the form a + bj, where a and b are real numbers, and j is the imaginary unit.

Sequence Types:

- str: String data type represents sequences of characters enclosed within single quotes (') or double quotes (").
- list: List data type represents ordered collections of items enclosed within square brackets ([]). Lists can contain elements of different data types.
- tuple: Tuple data type represents immutable ordered collections of items enclosed within parentheses (()). Tuples can contain elements of different data types.
- range: Range data type represents a sequence of numbers generated by the range() function.

Mapping Type:

- dict: Dictionary data type represents unordered collections of key-value pairs enclosed within curly braces ({}). Each key-value pair in a dictionary maps the key to its corresponding value.

Set Types:

- set: Set data type represents unordered collections of unique elements enclosed within curly braces ({}). Sets do not allow duplicate elements.
- frozenset: Frozenset data type represents immutable sets, similar to sets but cannot be modified after creation.

Boolean Type:

- bool: Boolean data type represents the truth values True and False, used for logical operations and comparisons.

Binary Types:

- bytes: Bytes data type represents immutable sequences of bytes, typically used to store binary data.
- bytearray: Bytearray data type represents mutable sequences of bytes, similar to bytes but can be modified.

None Type:

- None: None data type represents the absence of a value or a null value in Python.

These are the built-in data types provided by Python. Additionally, Python also allows defining custom data types using classes, enabling developers to create objects with specific attributes and methods tailored to their requirements. Understanding and utilizing these data types effectively is essential for writing Python programs that are efficient, readable, and maintainable.

1.9 Built-in constants in python

In Python, there are a few built-in constants that are predefined and can be used directly in your code without the need for explicit declaration. Here are some commonly used built-in constants in Python:

True: Represents the boolean value True, used to denote true or on conditions in boolean expressions.

False: Represents the boolean value False, used to denote false or off conditions in boolean expressions.

None: Represents the absence of a value or a null value in Python. It is often used to indicate that a variable or expression has no value assigned to it.

Ellipsis: Represents an ellipsis (...) object, typically used in slicing syntax to indicate an unspecified number of elements in a sequence.

NotImplemented: Represents a special value used to indicate that a method or operation is not implemented or supported.

debug: Represents the boolean value True when Python is in debug mode. It is often used to conditionally execute code based on whether the interpreter is in debug mode or not.

These built-in constants are provided by Python and can be accessed and used directly in your code. They serve various purposes and are useful for writing clear, concise, and expressive Python programs.

1.10. **Converting between datatypes in python**

In Python, you can convert between different data types using built-in functions or constructors provided by the language. Here are some common ways to convert between data types:

Implicit Conversion:

- Python automatically performs implicit type conversion (also known as coercion) when required. For example, when performing arithmetic operations between different data types, Python will convert them to a common type if necessary.

```
# Implicit conversion example
x = 10
y = 3.5
result = x + y  # The integer 10 is implicitly converted to a float before the addition operation
```

Explicit Conversion using Constructors:

- Python provides constructors for each data type that can be used to explicitly convert values from one type to another. For example:

- int(): Converts a value to an integer.
- float(): Converts a value to a floating-point number.
- str(): Converts a value to a string.
- list(): Converts a sequence to a list.
- tuple(): Converts a sequence to a tuple.
- dict(): Converts a sequence of key-value pairs to a dictionary.
- set(): Converts a sequence to a set.

```
# Explicit conversion example
x = 10
y = float(x)  # Convert integer to float
z = str(x)    # Convert integer to string
```

Using Type-specific Conversion Functions:

- Some data types have specific conversion functions or methods that can be used to convert values. For example:
- int(), float(), and complex(): These functions can be used to convert values to integer, float, and complex data types, respectively.
- bool(): Converts a value to a boolean based on its truthiness.
- bytes(): Converts a value to a bytes object.
- bytearray(): Converts a value to a bytearray object.

```
# Type-specific conversion example
x = "123"
y = int(x)    # Convert string to integer
z = float(x)  # Convert string to float
```

Using the eval() Function (with Caution):

- The eval() function can be used to evaluate a string expression and convert it to the corresponding data type. However, it should be used with caution as it can execute arbitrary code and may pose security risks if used with untrusted input.

```
# Using eval() for conversion (with caution)
x = "123"
y = eval(x)  # Convert string to integer
```

These are some common methods to convert between data types in Python. Choosing the appropriate method depends on the specific requirements and context of your code. Always ensure that the conversion is done safely and accurately to avoid unexpected behavior or errors in your program.

1.11 **Collection Types in python**

In Python, collection types are data structures used to store and organize multiple elements or values into a single container. Python provides several built-in collection types, each with its own characteristics and use cases. Here are the main collection types in Python:

Lists:

- Lists are ordered collections of items.
- They are mutable, meaning you can modify their elements after creation.
- Elements in a list can be of any data type, and they can be heterogeneous.
- Lists are created using square brackets [].

Tuples:

- Tuples are ordered collections of items, similar to lists.
- They are immutable, meaning you cannot modify their elements after creation.
- Elements in a tuple can be of any data type, and they can be heterogeneous.
- Tuples are created using parentheses ().

Sets:

- Sets are unordered collections of unique elements.
- They do not allow duplicate elements.
- Sets are mutable, meaning you can add or remove elements after creation.
- Sets are created using curly braces {} or the set() constructor.

Dictionaries:

- Dictionaries are unordered collections of key-value pairs.
- They are mutable and can be modified after creation.
- Keys in a dictionary must be unique and immutable (e.g., strings, numbers, tuples).
- Values in a dictionary can be of any data type and can be heterogeneous.
- Dictionaries are created using curly braces {} with key-value pairs separated by colons :.

Strings:

- Strings are immutable sequences of characters.
- They are commonly used to represent textual data.
- Strings support various operations and methods for manipulation.
- Strings are created using single quotes '' or double quotes "".

Bytes and Byte Arrays:

- Bytes and byte arrays are used to represent sequences of bytes.
- They are used for handling binary data, such as reading from and writing to files or network sockets.
- Bytes are immutable, while byte arrays are mutable.

These collection types in Python provide flexibility and versatility for storing and manipulating data in various ways. Choosing the appropriate collection type depends on factors such as mutability, ordering, uniqueness, and specific use case requirements.

1.12 **Built in Modules and Functions in python**

In Python, there are numerous built-in modules and functions available that provide a wide range of functionality for various tasks. These built-in modules and functions are part of the Python Standard Library and can be used directly in your code without the need for additional installation. Here are some commonly used built-in modules and functions in Python:

1. Built-in Modules:

    - math: Provides mathematical functions and constants, such as trigonometric functions, logarithms, and mathematical constants like π (pi) and e.
    - random: Allows generating random numbers, shuffling sequences, and selecting random elements from sequences.
    - datetime: Provides classes for working with dates, times, and time intervals.
    - os: Provides functions for interacting with the operating system, such as working with files and directories, environment variables, and process management.
    - sys: Provides access to system-specific parameters and functions, such as command-line arguments and the Python interpreter.
    - json: Allows encoding and decoding JSON data.
    - re: Provides support for regular expressions, allowing pattern matching and string manipulation.
    - csv: Provides classes and functions for reading and writing CSV files.
    - urllib: Allows working with URLs and making HTTP requests.
    - collections: Provides additional data structures beyond the built-in types, such as named tuples, deque, Counter, and defaultdict.
2. **Built-in Functions:**
    - print(): Prints the specified message or object to the standard output (usually the console).
    - input(): Reads input from the user via the keyboard.
    - len(): Returns the length (number of items) of an object, such as a sequence (e.g., list, tuple, string) or a collection (e.g., dictionary, set).
    - range(): Generates a sequence of numbers within a specified range.
    - type(): Returns the type of an object.
    - int(), float(), str(), list(), tuple(), dict(), set(): Constructors for creating objects of specific data types.
    - max(), min(): Returns the maximum or minimum value from a sequence.
    - sum(): Returns the sum of all elements in a sequence.
    - sorted(): Returns a new sorted list from the elements of the given iterable.
    - enumerate(): Returns an iterator that yields pairs of index and value from an iterable.

These are just a few examples of built-in modules and functions available in Python. The Python Standard Library provides a vast array of functionality for performing various tasks, making Python a powerful and versatile programming language.

1.13 **Creating a module in python**

Creating a module in Python is a simple process. A module in Python is essentially a file containing Python definitions and statements. These files have a .py extension and can contain functions, classes, or variables that can be imported and used in other Python scripts.

Here's how you can create a module in Python:

Create a Python File:

- Open your preferred text editor or IDE.
- Create a new file with a .py extension. For example, you can name it mymodule.py.

Write Python Code:

- Write the Python code you want to include in the module. This can be any valid Python code, such as function definitions, class definitions, or variable assignments.
- For example, let's create a simple module with a function that prints a greeting:

```python
# mymodule.py

def greet(name):
    print("Hello, " + name + "!")

```

Save the File:

- Save the Python file in the desired directory. Make sure to remember the location where you saved it

Using the Module:

- Once the module file is created, you can import it into other Python scripts using the import statement.
- For example, let's create another Python script and import the mymodule module:

```python
# main.py

import mymodule

mymodule.greet("AJAY")

```

In this example, we import the mymodule module using the import statement, and then we call the AJAY() function from the module to print a greeting.

Running the Script:

- To run the script that imports and uses the module, execute the Python script containing the import statement.
- Make sure that both the module file (mymodule.py) and the script file (main.py) are in the same directory or that the directory containing the module file is included in the Python path.

When you run the script (main.py), it will import the module (mymodule.py) and execute the code within it, including the AJAY() function. This demonstrates how to create and use a simple module in Python.

1.14 **String function - str() and repr() in python**

In Python, both str() and repr() are built-in functions used for converting objects into string representations. However, they serve slightly different purposes and produce different outputs.

str() Function:

- The str() function is used to create a human-readable string representation of an object.
- It is intended to produce a user-friendly representation of the object's value.
- The output generated by str() may sacrifice precision or detailed information in favor of readability.
- It is commonly used for displaying objects to users or in situations where a concise, easy-to-read string representation is desired.

Example:

```
num = 10
str_representation = str(num)
print(str_representation)  # Output: '10'
```

repr() Function:

- The repr() function is used to create an unambiguous string representation of an object.
- It is intended to produce a representation that can be used to recreate the object exactly.
- The output generated by repr() is typically more detailed and precise compared to str().
- It includes information about the object's type and internal state, making it useful for debugging and development purposes.

Example:

```
num = 10
repr_representation = repr(num)
print(repr_representation)  # Output: '10'
```

When used on more complex objects, repr() provides additional information:

```
lst = [1, 2, 3]
repr_representation = repr(lst)
print(repr_representation)  # Output: '[1, 2, 3]'
```

Notice how repr() includes the square brackets and commas, providing a more detailed representation of the list.

In summary, str() is used for creating readable string representations, while repr() is used for creating unambiguous and detailed representations, especially for debugging and development purposes. Depending on the context, you may choose to use one or the other to obtain the desired output.

1.15 **How to Install external modules using pip in python**

To install external modules in Python using pip (Python's package installer), you can follow these steps:

Open a Terminal or Command Prompt:

- Open the terminal or command prompt on your computer.

Check if pip is Installed:

- Type the following command and press Enter:

```
pip --version
```

If pip is installed, you'll see its version information. If not, you'll need to install Python and ensure that pip is included during the installation process.

Install a Module:

- To install a specific module, use the following command:

```
pip install <module_name>
```

For example, to install the requests module, you would type:

```
pip install requests
```

Optional: Specify Module Version:

You can also specify a particular version of the module to install by appending == followed by the version number. For example:

```
pip install requests==2.26.0
```

View Installed Modules:

You can view the list of installed modules and their versions by typing:

```
pip list
```

Upgrade a Module:

- To upgrade a module to the latest version, use the following command:

```
pip install --upgrade <module_name>
```

For example, to upgrade the requests module, you would type:

```
pip install --upgrade requests
```

Uninstall a Module:

To uninstall a module, use the following command:

```
pip uninstall <module_name>
```

For example, to uninstall the requests module, you would type:

```
pip uninstall requests
```

Installing from a Requirements File:

- You can also install multiple modules listed in a requirements file by using the -r flag followed by the path to the requirements file. For example:

```
pip install -r requirements.txt
```

1.16 **Help Utility in python**

In Python, there are several ways to access help and documentation for modules, functions, classes, and methods. Here are some common methods to access the help utility in Python:

Using the help() Function:

- The help() function is a built-in Python function that provides interactive help on objects. You can pass any object (module, function, class, etc.) as an argument to help() to get information about it.

Example:

```
# Get help on the math module
help(math)

# Get help on the built-in print function
help(print)

# Get help on a specific method of a class
help(str.upper)
```

- When you run these commands interactively, you'll enter the help utility, where you can navigate through the documentation using keyboard commands. Press q to exit the help utility.

Using the dir() Function:

- The dir() function is a built-in Python function that returns a list of valid attributes and methods of an object.
- You can use dir() to list the attributes and methods of an object and then use help() to get information about a specific attribute or method.

Example:

```
# List attributes and methods of the math module
print(dir(math))

# Get help on a specific method of the math module
help(math.sqrt)
```

Using Documentation Strings (docstrings):

- Many Python modules, functions, classes, and methods include documentation strings, also known as docstrings. Docstrings are strings that appear as the first statement in a module, function, class, or method definition and provide information about its purpose, usage, and parameters.
- You can access the docstring of an object using the .__doc__ attribute.

Example:

```
# Get the docstring of the math module
print(math.__doc__)

# Get the docstring of the sqrt() function in the math module
print(math.sqrt.__doc__)
```