Artificial Intelligence Diploma

Python Session 1

NeuroTech

Agenda

Programming

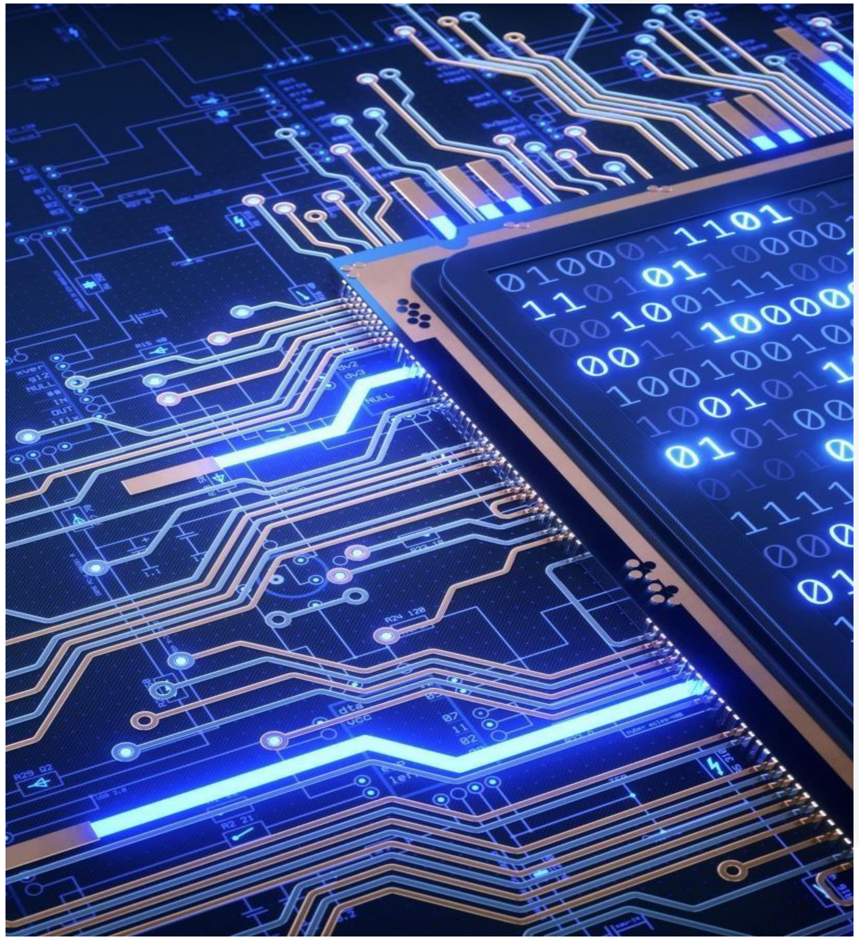
NeuroTech

What is Programming?

- **Programming** refers to the process of writing instructions or code that a computer can understand and execute. It involves designing, creating, testing, and maintaining sets of instructions to perform specific tasks or solve problems.

- In essence, programming enables us to communicate with computers and instruct them on what actions to take. Computers are essentially machines that operate based on specific sets of instructions, and programming allows us to provide those instructions.

Machine Code or Machine Language



- Machine code, also known as machine language, is the elemental language of computers. It is read by the computer's central processing unit (CPU), is composed of digital binary numbers and looks like a very long sequence of zeros and ones.

- Each CPU has its own specific machine language. The processor reads and handles instructions, which tell the CPU to perform a simple task. Instructions are comprised of a certain number of bits. If instructions for a particular processor are 8 bits, for example, the first 4 bits part (the opcode) tells the computer what to do and the second 4 bits (the operand) tells the computer what data to use.
  - 01001000 01100101 01101100 01101100 01101111 00100001

- Depending upon the processor, a computer's instruction sets may all be the same length, or they may vary, depending upon the specific instruction. The architecture of the particular processor determines how instructions are patterned.

NEUROTECH

Low Level Programming Vs High level Programming

Low-Level Programming: Low-level programming involves working directly with the hardware and writing code that is closely tied to the specific architecture of the computer system. It requires a deep understanding of computer architecture and the inner workings of the hardware. Low- level languages are often referred to as "close to the machine" languages. Examples of low-level languages include Assembly language and machine code.

High-Level Programming: High-level programming involves writing code that is more abstracted from the hardware and closer to human language. High-level programming languages provide a higher level of abstraction and are designed to simplify programming tasks.
Examples of high-level languages include Python,
Java, C++, and JavaScript.

NEUROTECH

Python Programming Language

NeuroTech

Python Programming Language

- **Python** is a high-level, interpreted programming language that emphasizes code readability and simplicity. It was created by Guido van Rossum and first released in 1991. Python has gained widespread popularity due to its ease of use, versatility, and a strong community of developers.

- **Python** is extensively used in various domains, including web development, data analysis, machine learning, artificial intelligence, scientific computing, automation, and more. Its simplicity, versatility, and extensive libraries make it a popular choice among programmers of all levels of expertise.

Key Features of Python

- **Easy to Learn and Read**: Python's syntax is designed to be clear and readable, resembling English-like statements. This makes it an ideal choice for beginners and enhances code readability, reducing development time and maintenance efforts.
- **Interpreted Language**: Python is an interpreted language, meaning that the code is executed line by line, without the need for compilation. This allows for rapid development and easy debugging, as errors can be identified and fixed on the go.
- **Cross-Platform Compatibility**: Python is available on various platforms, including Windows, macOS, Linux, and more. It provides a consistent programming experience across different operating systems, making it highly portable.
- **Extensive Standard Library**: Python comes with a comprehensive standard library that offers a wide range of modules and functions for common tasks. The standard library covers areas such as file I/O, networking, data manipulation, regular expressions, and more, which can significantly speed up development.
- **Dynamic Typing and Automatic Memory Management**: Python is dynamically typed, meaning that variable types are inferred at runtime, which provides flexibility and allows for quicker prototyping. Python also handles memory management automatically, relieving developers from managing memory explicitly.

Key Features of Python

- **Large and Active Community**: Python has a vibrant and active community of developers. This community contributes to the growth of the language by creating libraries, frameworks, and tools, and provides extensive documentation, tutorials, and support to help fellow programmers.
- **Multi-paradigm Language**: Python supports multiple programming paradigms, including procedural, object- oriented, and functional programming. It allows developers to choose the best approach for their project and encourages code organization and reusability.
- **Vast Ecosystem of Libraries and Frameworks**: Python has a rich ecosystem of third-party libraries and frameworks, which extend its capabilities and make it suitable for various domains. For web development, popular frameworks include Django and Flask, while for scientific computing and data analysis, libraries like NumPy, Pandas, and Matplotlib are widely used.
- **Integration and Extensibility**: Python can be easily integrated with other programming languages, enabling developers to leverage existing code written in languages like C, C++, or Java. This feature enhances Python's capabilities and allows for efficient use of resources.
- **Open Source**: Python is an open-source language, which means its source code is freely available. This promotes collaboration, encourages innovation, and allows developers to contribute to the language's development.

Interpreter Vs Compiler

Interpreter	Compiler
An interpreter translates code written in a high-level programming language into machine code line-by-line as the code runs.	A compiler translates code from a high-level programming language into machine code before the program runs.
The interpreter takes a single line of code and very little time to analyze it.	A compiler takes in the entire program and requires a lot of time to analyze the source code.
Interpreted code runs slower.	Compiled code runs faster
The interpreter displays errors of each line one by one.	A compiler displays all errors after compilation. If your code has mistakes, it will not compile.
Example: Python, R, JavaScript	Example: C++, C, Java

Why python With Machine Learning?

- **Extensive Libraries and Frameworks:** Python has a rich ecosystem of machine learning libraries and frameworks that make it easier for developers to implement complex machine learning algorithms. Libraries like NumPy, Pandas, scikit-learn, TensorFlow, and PyTorch provide robust tools for data manipulation, model building, and deep learning.

- **Rapid Prototyping:** Python's quick development cycle and extensive libraries enable researchers and developers to rapidly prototype machine learning models and experiments. This allows for iterative experimentation and faster progress in the development of machine learning solutions.

- **Community Support:** Python's large and active community has contributed significantly to the development of machine learning libraries and tools. The availability of extensive documentation, tutorials, and community support makes it easier for newcomers to learn and apply machine learning concepts using Python.

Why python With Machine Learning?

- **Extensive Libraries and Frameworks:** Python has a rich ecosystem of machine learning libraries and frameworks that make it easier for developers to implement complex machine learning algorithms. Libraries like NumPy, Pandas, scikit-learn, TensorFlow, and PyTorch provide robust tools for data manipulation, model building, and deep learning.

- **Rapid Prototyping:** Python's quick development cycle and extensive libraries enable researchers and developers to rapidly prototype machine learning models and experiments. This allows for iterative experimentation and faster progress in the development of machine learning solutions.

- **Community Support:** Python's large and active community has contributed significantly to the development of machine learning libraries and tools. The availability of extensive documentation, tutorials, and community support makes it easier for newcomers to learn and apply machine learning concepts using Python.

How Install Python

- **To install Python on your computer, follow these steps:**

  1. **Check if Python is already installed**: Some operating systems come with Python pre-installed. Open a command prompt or terminal and type **python --version**or **python3 --version**to check if Python is already installed and to see the version number.
  2. **Download Python**: If Python is not installed or you need to update to a newer version, go to the official Python website (https://www.python.org/downloads/) and download the latest stable release of Python. Choose the appropriate installer for your operating system (Windows, macOS, or Linux).
  3. **Run the Installer**: After downloading the installer, run it and follow the installation wizard's instructions. On Windows, make sure to check the option "Add Python to PATH" during the installation process. This will allow you to use Python from the command prompt or terminal without specifying the full path.
  4. **Verify the Installation**: Once the installation is complete, open a new command prompt or terminal and type **python --version** or **python3 --version** to verify that Python has been installed correctly and to check the version number.

NEUROTECH

Anaconda

NeuroTech

What is Anaconda?

- **Anaconda** is a popular open-source distribution of the Python and R programming languages for data science and machine learning. It is widely used by data scientists, researchers, and developers for managing and deploying data-intensive applications. Anaconda simplifies the installation and management of data science packages and libraries, making it easier to work with complex data analysis and machine learning tasks.

- **Key features of Anaconda include:**
  - **Package Management**: Anaconda comes with a package manager called **conda**, which allows you to easily install, update, and manage various data science packages and libraries. **conda** handles package dependencies and ensures compatibility between different libraries, which can be challenging when installing packages individually.
  - **Data Science Libraries**: Anaconda includes a vast collection of pre-installed data science packages, including NumPy, Pandas, Matplotlib, SciPy, scikit-learn, TensorFlow, and PyTorch, among others. These libraries are essential for data manipulation, analysis, visualization, and machine learning.

Key Features of Anaconda

- **Virtual Environments**: Anaconda allows you to create isolated virtual environments for different projects. These environments help avoid conflicts between package dependencies, ensuring that your projects run smoothly and independently from one another.
- **Cross-Platform Support**: Anaconda is available for Windows, macOS, and Linux, making it a versatile choice for data scientists working on different operating systems.
- **Integrated Development Environment (IDE)**: Anaconda includes Jupyter Notebook, a web-based interactive computing environment that allows you to create and share documents containing live code, equations, visualizations, and narrative text. Jupyter Notebook is widely used for data exploration, prototyping, and presenting data analysis results.

How Install Anaconda ?

1. **Download Anaconda**: Go to the Anaconda website (https://www.anaconda.com/products/individual) and download the appropriate installer for your operating system (Windows, macOS, or Linux). Choose the installer for the latest version of Python 3, as Python 2 has reached its end of life and is no longer supported.

2. **Run the Installer**: After downloading the Anaconda installer, run it on your system. The installation wizard will guide you through the process.
   1. On Windows: Double-click the downloaded .exe file and follow the installation prompts. Make sure to check the box that says "Add Anaconda to my PATH environment variable" during the installation process. This allows you to use Anaconda from the command prompt without specifying the full path.
   2. On macOS: Open the downloaded .pkg file and follow the installation prompts. macOS may prompt you to enter your administrator password during the installation.
   3. On Linux: Open a terminal, navigate to the directory containing the downloaded .sh file, and run the installer using the command **bash Anaconda*.sh**. Follow the installation prompts.

NEUROTECH

How Install Anaconda ?

3. **Accept the License Agreement**: Read and accept the license agreement when prompted during the installation.
4. **Choose Installation Location**: The installer will ask you to choose a location for the Anaconda installation. You can
   leave the default location or select a custom directory.
5. **Initialize Anaconda**: After installation, you may need to initialize Anaconda to set up the necessary environment variables. On Windows, you can do this by opening the Anaconda Prompt from the Start menu. On macOS and Linux, open a terminal.

Virtual Environment

- In Anaconda, a virtual environment, also known as a virtual machine, is a self-contained directory that contains a specific version of Python along with a set of pre-installed packages and libraries. It allows you to create isolated environments for different projects, each with its own dependencies, without interfering with other projects or the system's Python installation.

- The main benefits of using virtual environments in Anaconda are:

1. **Isolation**: Each virtual environment acts as a separate workspace, isolating the project's dependencies from other projects and the system. This helps prevent conflicts between packages and ensures that different projects can run smoothly without affecting each other.
2. **Dependency Management**: By creating a virtual environment, you can specify the exact versions of packages required for your project. This makes it easier to reproduce and share your work with others, ensuring that everyone is working with the same set of dependencies.

Virtual Environment

- To create a new virtual environment, open the Anaconda Prompt (Windows) or a terminal (macOS and Linux), and run the following command:

1. **conda create --name myenv python=3.9**
   - This will create a new virtual environment named "myenv" with Python 3.8. You can replace "myenv" with any name you prefer and choose a different Python version if needed.
2. **conda activate myenv**
   - **Activate the Virtual Environment**: After creating a virtual environment, you need to activate it to start using it

Python Basics

NeuroTech

Python Basics

- **Print statement:**
- In Python, you can use the **print()**function to display output to the console or terminal. The **print()** function allows you to output text, variables, and expressions. Here's the basic syntax of the **print()** function:
- print(*objects, sep=' ', end='\n')
    - **objects**: One or more objects to be printed. You can pass multiple objects separated by commas, and **print()**will automatically convert them to strings and concatenate them with a space as the default separator.
    - **sep**: Optional. Specifies the separator used between the objects. The default separator is a space **' '**.
    - **end**: Optional. Specifies the character that is printed at the end. The default value is a newline character **'\n'**, which prints each **print()**statement on a new line.
- Example:
- print "Hello, World!") ---- > Hello, World!)

NEUROTECH

Python Basics

Quiz

- If Output on Screen Is ---- > **Data Science Diploma**, What is statement for show this output?

1. Print('Data Science Diploma')
2. Print("Data Science Diploma")
3. print['Data Science Diploma']
4. print('Data Science Diploma')

NeuroTech

Python Basics

Quiz

- If Output on Screen Is ----- > **Data Science Diploma**, What is statement for show this output?

1. print('Data', ‘Science’ , 'Diploma')
2. print('Data', ‘Science ‘, 'Diploma',sep='')
3. print('Data', ‘Science’ , 'Diploma',sep='',end=' ')
4. print('Data', ‘Science’ , 'Diploma',sep='   ')

NeuroTech

Variable

- A variable is a name that refers to a value or an object in memory. It acts as a placeholder or container to store data during the execution of a program. Unlike some other programming languages, Python is dynamically typed, which means that you do not need to explicitly declare the type of a variable when you create it. The type of the variable is determined automatically based on the value assigned to it.

- **Variable Assignment:** To assign a value to a variable, you simply use the equal sign (**=**). For example:

```
x = 10
name = "John"
is_student = True
```

- In this example, we have created three variables: **x**, **name**, and **is_student**. **x**holds an integer value, **name** holds a string value, and **is_student**holds a boolean value (**True**).

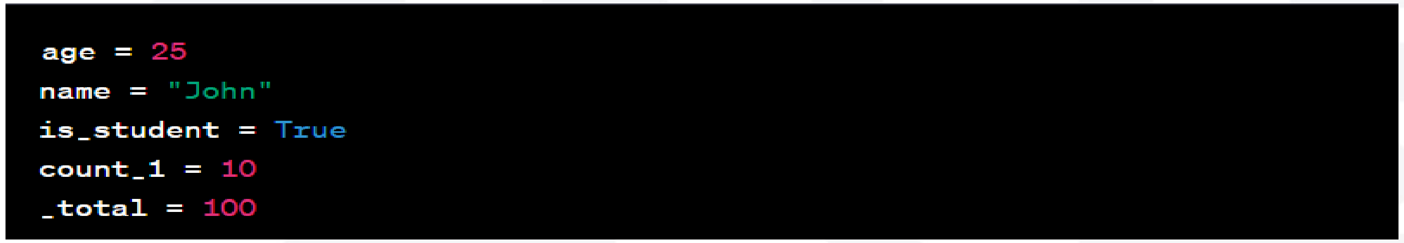
Variable

- **In Python**, a variable identifier is a unique name given to a variable. It acts as a reference or label for the memory location where the variable's value is stored. When you create a variable, Python associates it with a specific identifier, allowing you to access and manipulate its value in the code.

- **Rules for variable identifiers in Python:**
  1. The variable name must start with a letter (a-z, A-Z) or an underscore (_).
  2. It can be followed by letters, digits (0-9), or underscores.
  3. Variable names are case-sensitive, so **myVar**, **MyVar**, and **myvar** are considered different variables.
  4. Python keywords (reserved words like **if**, **else**, **for**, etc.) cannot be used as variable names.
  5. Variable names should be descriptive and follow a consistent naming convention for better code readability.

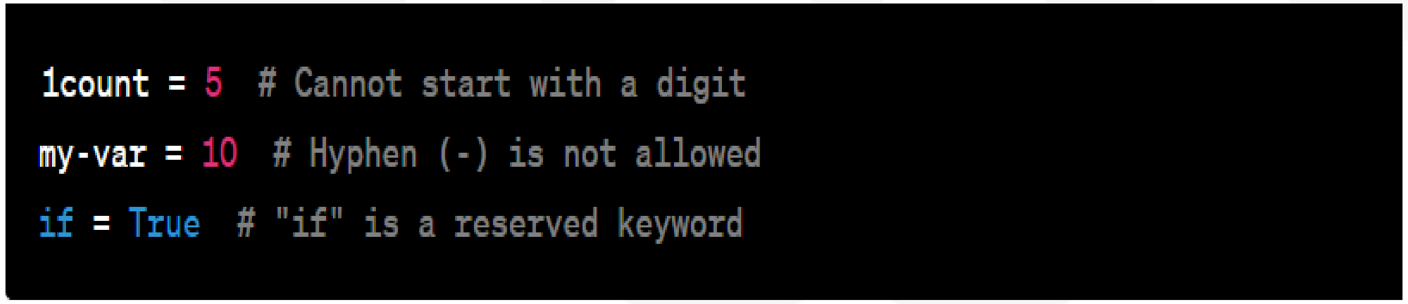
Variable Identifier (Variable Name)

- Examples of valid variable identifiers :



```python
age = 25
name = "John"
is_student = True
count_1 = 10
_total = 100
```

- Examples of invalid variable identifiers:



```python
1count = 5  # Cannot start with a digit
my-var = 10  # Hyphen (-) is not allowed
if = True  # "if" is a reserved keyword
```

Variable Types

- In Python, variables can hold various data types, which define the kind of data they can store. Python is a dynamically-typed language, meaning the data type of a variable is determined at runtime based on the value assigned to it. Here are some common data types in Python:

1. **Numeric Types**:
   - **int**: Integer values, e.g., 10, -5, 0.
   - **float**: Floating-point numbers, e.g., 3.14, -2.5, 0.0.
   - **complex**: Complex numbers with a real and imaginary part, e.g., 2 + 3j, -1 + 2j.
2. **Boolean Type**:
   - **bool**: Represents either True or False, used for logical operations.
3. **Text Type**:
   - **str**: Strings of characters, e.g., "Hello, Python!", '42', "Data Science".
4. **Sequence Types**:
   - **list**: Ordered, mutable collection of elements, e.g., [1, 2, 3], ['apple', 'banana', 'cherry'].
   - **tuple**: Ordered, immutable collection of elements, e.g., (1, 2, 3), ('red', 'green', 'blue').

Variable Types

5. Set Types:

- **set**: Unordered, mutable collection of unique elements, e.g., {1, 2, 3}, {'apple', 'banana', 'apple'}.

6. Mapping Type:

- **dict**: A collection of key-value pairs, e.g., {'name': 'John', 'age': 25}.

7. Bytes Type:

- **bytes**: Represents a sequence of bytes, used for working with binary data.

8. Byte array Type:

- **Byte array**: Similar to bytes but mutable.

9. Range Type:

- **range**: Represents an immutable sequence of numbers.

Mutable vs Immutable

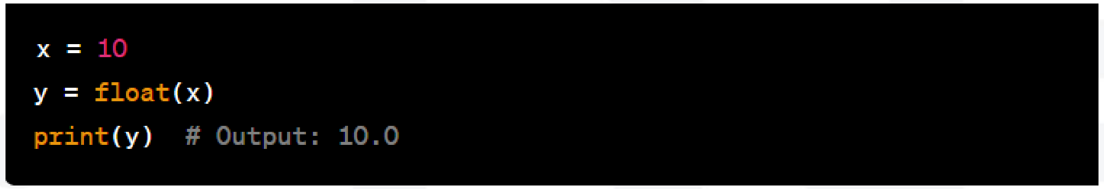
- In Python, objects are classified into two categories based on whether their values can be changed after creation:

1. **Mutable Objects**:
   - Mutable objects are objects whose values can be modified after they are created. This means you can change their contents, add or remove elements, or modify their properties in place without creating a new object.

2. **Immutable Objects**:
   - Immutable objects are objects whose values cannot be changed after they are created. Once an immutable object is created, you cannot modify its contents. Instead, any operation that seems to modify the object creates a new object with the updated value.

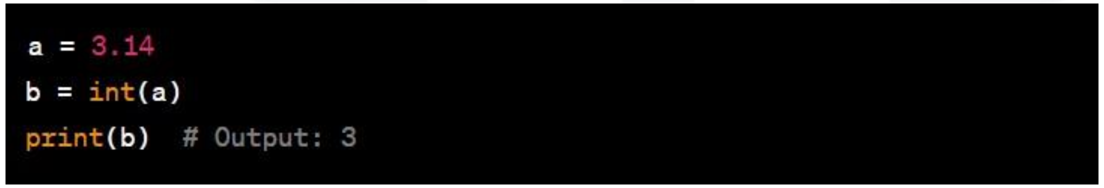
Convert between Different number Types

- You can convert between different number types using type casting functions or constructors. Type casting allows you to change the data type of a variable or value to another numeric type. Here are some common ways to convert between number types:
- **Integer to Float**:
  - You can convert an integer to a floating-point number using the **float()**constructor:



```
x = 10
y = float(x)
print(y)  # Output: 10.0
```

- **Float to Integer (Truncation)**:
  - You can convert a floating-point number to an integer using the **int()**constructor. Note that this truncates the decimal part:



```
a = 3.14
b = int(a)
print(b)  # Output: 3
```

NEUROTECH

Convert between Boolean and Numbers

- **Boolean to Integer**:
  - In Python, **True**is treated as 1, and **False**is treated as 0 when converted to integers:

```
bool_val = True
num_int = int(bool_val)

print(num_int)  # Output: 1
```

- **Integer to Boolean**:
  - You can convert an integer to a Boolean using the **bool()** constructor. Any non-zero integer is considered **True**, while 0 is considered **False**:

```
num = 42
bool_val = bool(num)

print(bool_val)  # Output: True
```

Arithmetic Operations

Operator	Description	Example
+	Addition	print(x + y) # Output: 13
-	Subtraction	print(x - y) # Output: 7
*	Multiplication	print(x * y) # Output: 30
/	Division (returns a float result)	print(x / y) # Output: 3.3333333333333335
//	Floor Division (returns an integer result, discards the decimal part)	print(x // y) # Output: 3
%	Modulo (returns the remainder after division)	print(x % y) # Output: 1
**	Exponentiation (raises a number to a power)	print(x ** y) # Output: 1000

NEUROTECH

Input Function

- In Python, the **input()**function is used to read user input from the console. It allows your program to interact with the user, prompting them to enter some data, which is then returned as a string.

- The syntax of the **input()**function is straightforward:

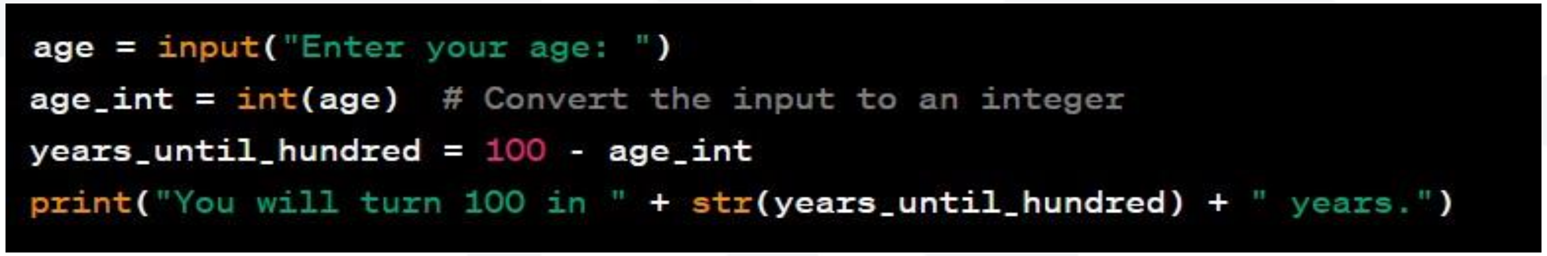
```
user_input = input("Enter something: ")
```

- The **input()**function takes an optional string argument, which serves as the prompt displayed to the user when asking for input. The user's input is read as a string and assigned to the variable **user_input**.

NEUROTECH

Input Function

- Keep in mind that **input()** always returns the user's input as a string, regardless of what the user enters. If you want to perform calculations or operations with the user's input as numbers, you'll need to convert the input to the appropriate data type using functions like **int()**or **float()**.



```
age = input("Enter your age: ")
age_int = int(age)  # Convert the input to an integer
years_until_hundred = 100 - age_int
print("You will turn 100 in " + str(years_until_hundred) + " years.")
```

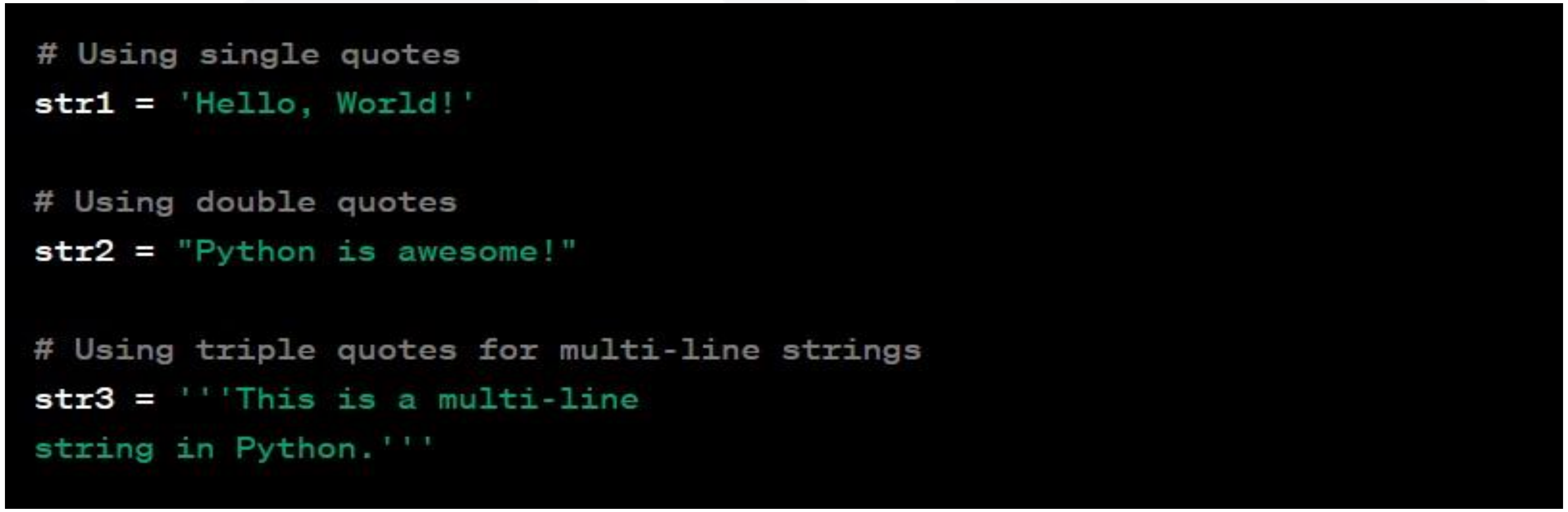
- In the above example, we convert the user's input (age) to an integer using **int()**so that we can perform arithmetic operations with it.
- Keep in mind that the **input()**function allows for direct interaction with users, making it a handy tool for building interactive applications and scripts in Python.

String

String

- In Python, a string is a sequence of characters enclosed in single quotes (' '), double quotes (" "), or triple quotes (''' ''' or """ """). Strings are one of the essential data types in Python and are used to represent text and textual data. They are immutable, meaning their content cannot be changed after creation, but you can create new strings by combining or manipulating existing ones.

- Some examples of strings in Python:

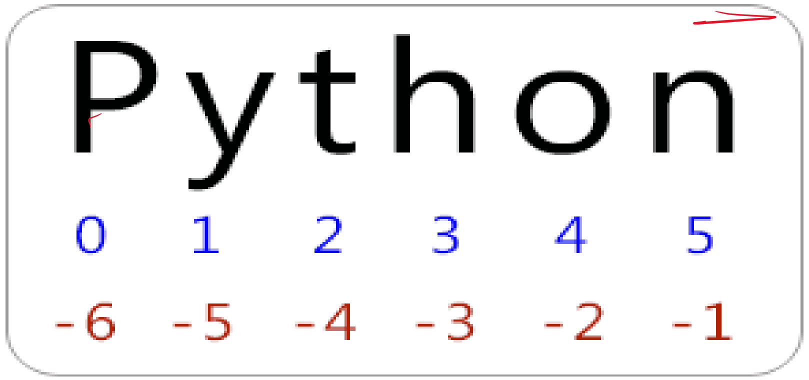


```python
# Using single quotes
str1 = 'Hello, World!'

# Using double quotes
str2 = "Python is awesome!"

# Using triple quotes for multi-line strings
str3 = '''This is a multi-line
string in Python.'''
```

NEUROTECH

String Indexing

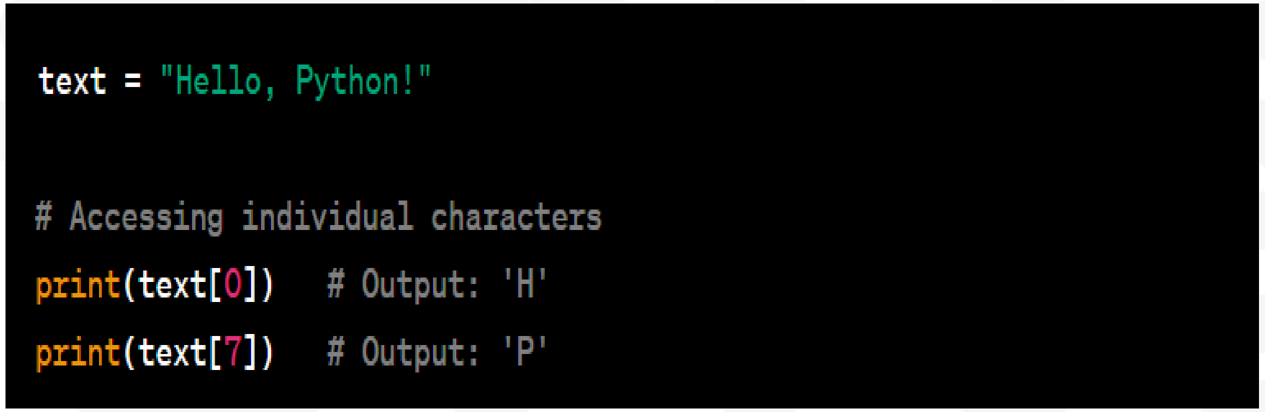
- String indexing in Python allows you to access individual characters in a string using their positions, or indices. Strings are sequences of characters, and each character in a string has a specific index associated with it.
- Indexing in Python starts at 0, meaning the first character is at **index 0**, the second character is at **index 1**, and so on.
- You can also use negative indices to count characters from the end of the string. **-1**represents the last character, **-2**represents the second-to-last character, and so on:



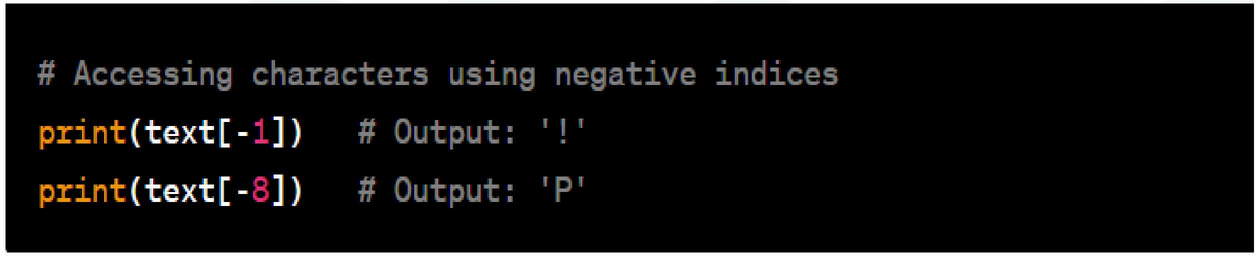
NEUROTECH

String Indexing

- Here's how you can use string indexing to access characters in a string:



```python
text = "Hello, Python!"

# Accessing individual characters
print(text[0])   # Output: 'H'
print(text[7])   # Output: 'P'
```



```python
# Accessing characters using negative indices
print(text[-1])   # Output: '!'
print(text[-8])   # Output: 'P'
```

String Indexing

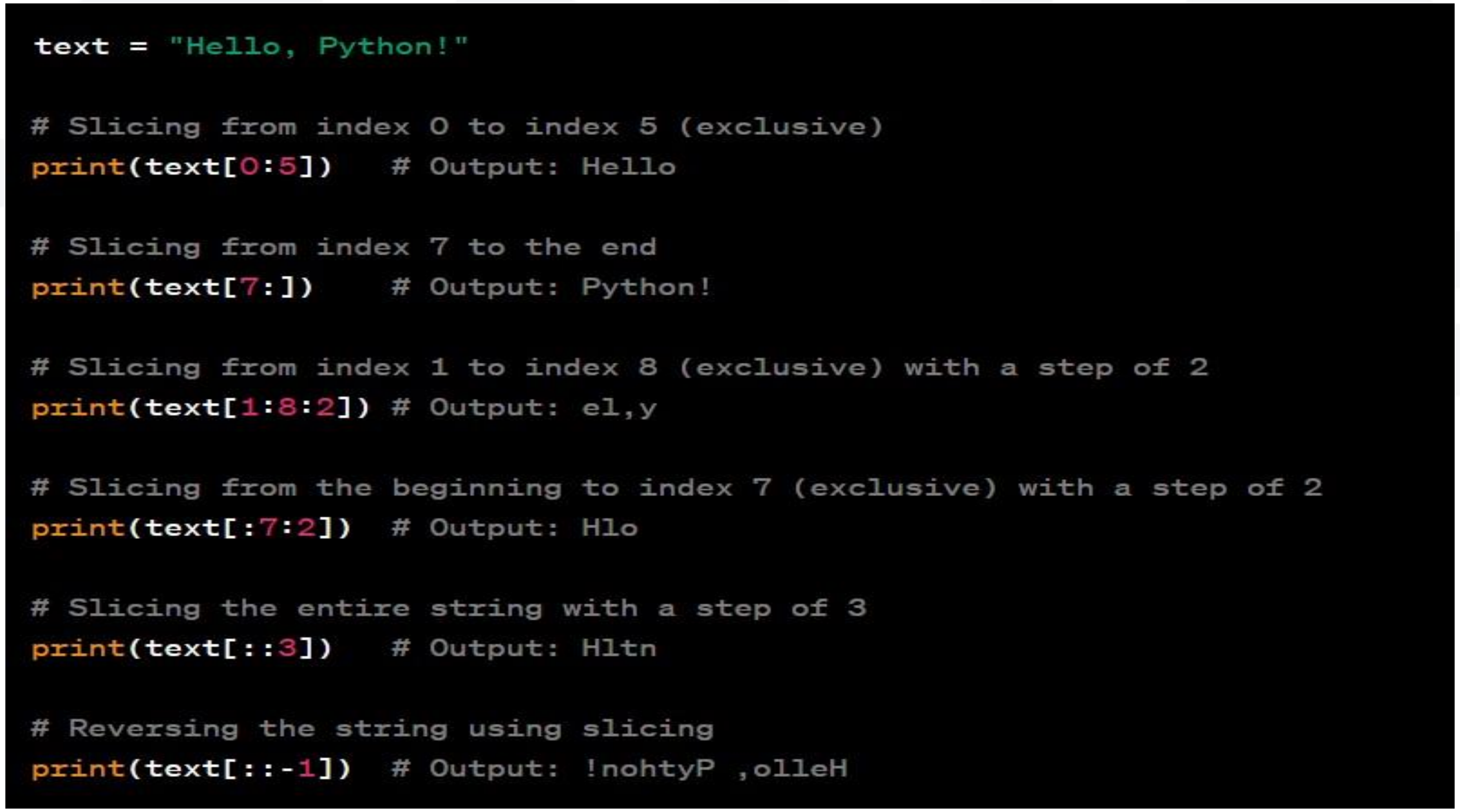
- string slicing allows you to extract a portion of a string by specifying a range of indices. Slicing creates a new substring without modifying the original string. The syntax for string slicing is as follows:

- String_name[start:stop:step]

  - **start**: The index (inclusive) from which the slice starts. If not specified, it defaults to 0 (the beginning of the string).
  - **stop**: The index (exclusive) at which the slice ends. If not specified, it defaults to the end of the string.
  - **step**: The number of positions to jump while slicing. If not specified, it defaults to 1.

String Slicing

- Some examples of string slicing :



```python
text = "Hello, Python!"

# Slicing from index 0 to index 5 (exclusive)
print(text[0:5])   # Output: Hello

# Slicing from index 7 to the end
print(text[7:])    # Output: Python!

# Slicing from index 1 to index 8 (exclusive) with a step of 2
print(text[1:8:2]) # Output: el,y

# Slicing from the beginning to index 7 (exclusive) with a step of 2
print(text[:7:2])  # Output: Hlo

# Slicing the entire string with a step of 3
print(text[::3])   # Output: Hltn

# Reversing the string using slicing
print(text[::-1])  # Output: !nohtyP ,olleH
```

NEUROTECH

String

Quiz

- What is output of the following statements:
  - Str='NeuroTech Academy'
  - print(Str[2:5])

- Output ?
  1. 'uro'
  2. 'uroT'
  3. 'euro'

String

Quiz

- What is output of the following statements:
  - Str='NeuroTech Academy'
  - print(Str[4::2])

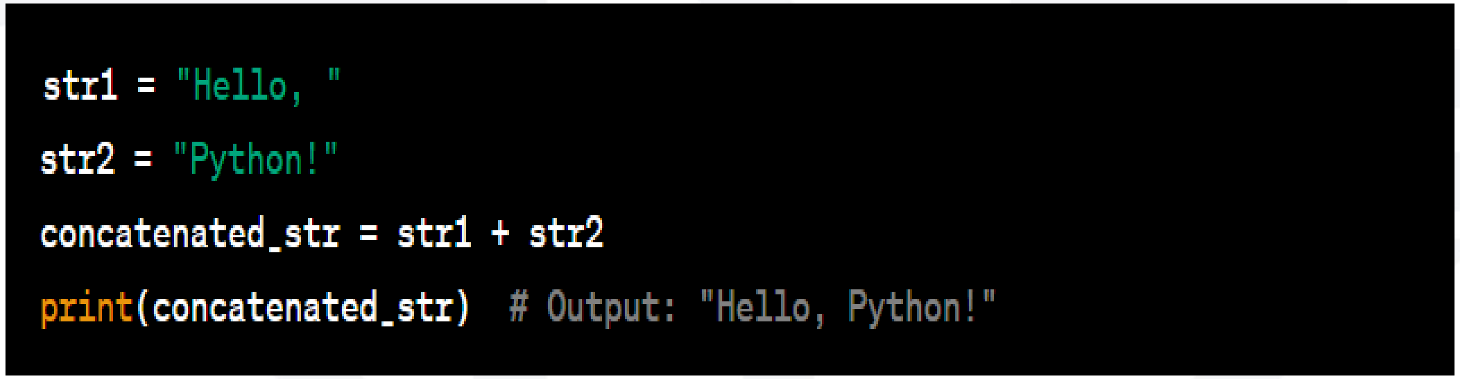
- Output ?
  1. 'oehcdm'
  2. 'ur'
  3. 'oehAaey'
  4. 'oTech Academy'

NEUROTECH

String Operation

1. Concatenation **(+)**:

- The '**+**'operator allows you to concatenate (combine) two or more strings into a single string.

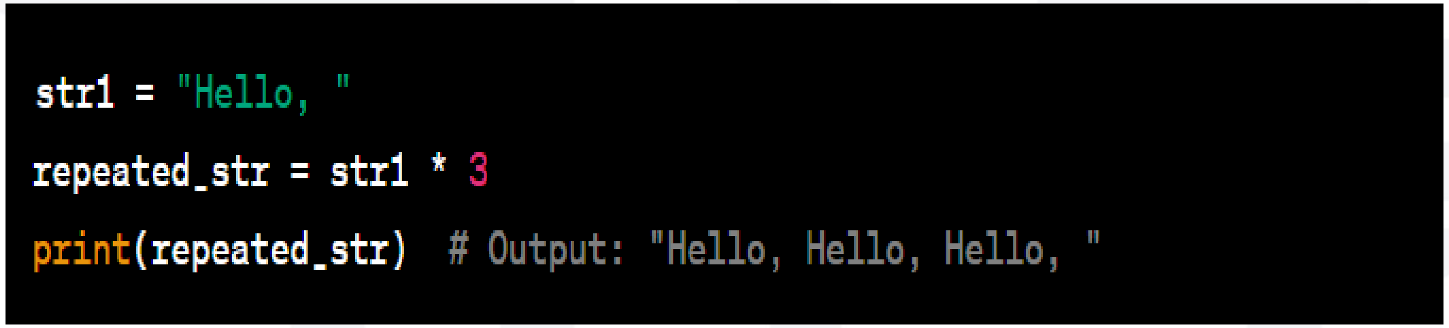


```python
str1 = "Hello, "
str2 = "Python!"
concatenated_str = str1 + str2
print(concatenated_str)  # Output: "Hello, Python!"
```

String Operation

2. Repetition **(*):**

- The '*****'operator allows you to repeat a string a certain number of times.

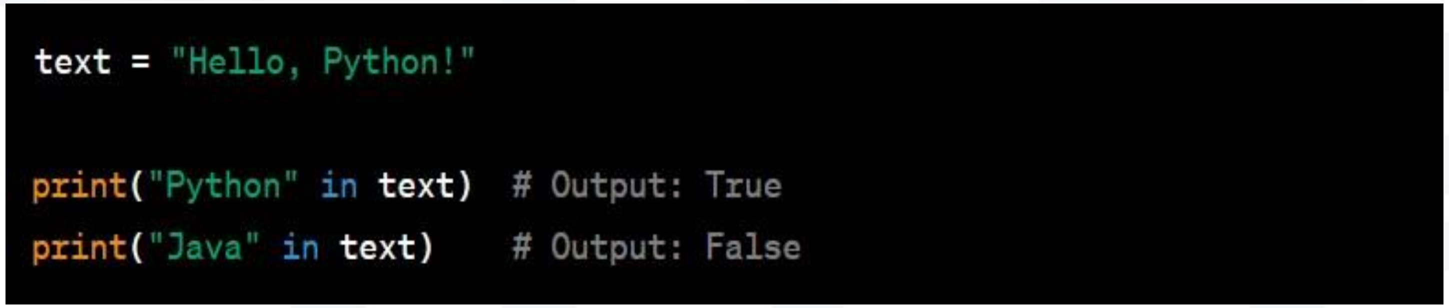


```python
str1 = "Hello, "
repeated_str = str1 * 3
print(repeated_str)  # Output: "Hello, Hello, Hello, "
```

String Operation

3. in Operator:

- The '**in**'operator checks if a substring exists within a string.



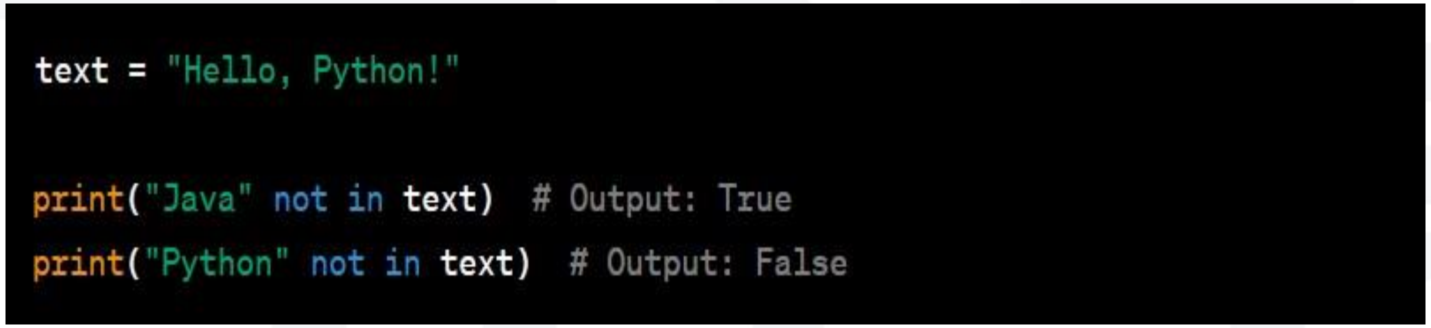
```python
text = "Hello, Python!"

print("Python" in text)  # Output: True
print("Java" in text)    # Output: False
```

NeuroTech

String Operation

4. **not in** Operator:

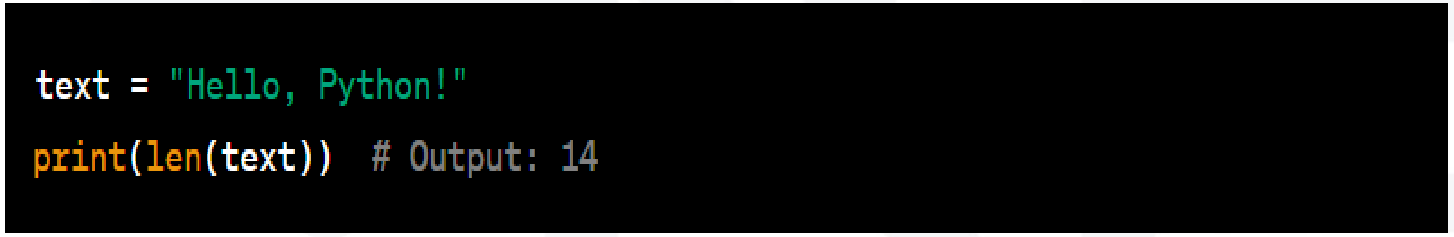
- The '**not in**'operator checks if a substring does not exist within a string.



```
text = "Hello, Python!"

print("Java" not in text)  # Output: True
print("Python" not in text)  # Output: False
```

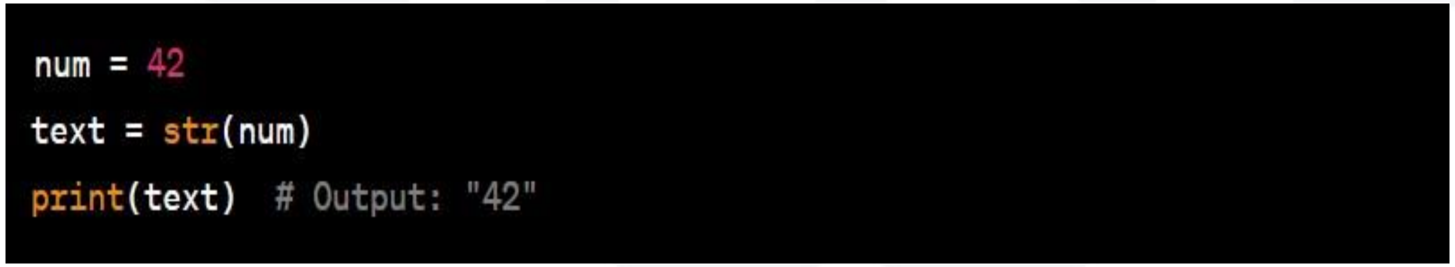
String Function

1. len(): returns the length of a string (the number of characters in the string).



```python
text = "Hello, Python!"
print(len(text))  # Output: 14
```

2. str(): Converts a value to a string.



```python
num = 42
text = str(num)
print(text)  # Output: "42"
```

NeuroTech

String Function

3. capitalize(): Converts the first character of the string to uppercase.

```python
text = "hello, python!"
capitalized_text = text.capitalize()
print(capitalized_text)  # Output: "Hello, python!"
```

4. upper(), **lower()**: Converts the entire string to uppercase or lowercase, respectively.

```python
text = "Hello, Python!"
upper_text = text.upper()
lower_text = text.lower()

print(upper_text)  # Output: "HELLO, PYTHON!"
print(lower_text)  # Output: "hello, python!"
```

String Function

5.split(): Splits a string into a list of substrings based on a specified separator.

```
text = "Hello, Python!"
words = text.split(", ")
print(words)  # Output: ['Hello', 'Python!']
```

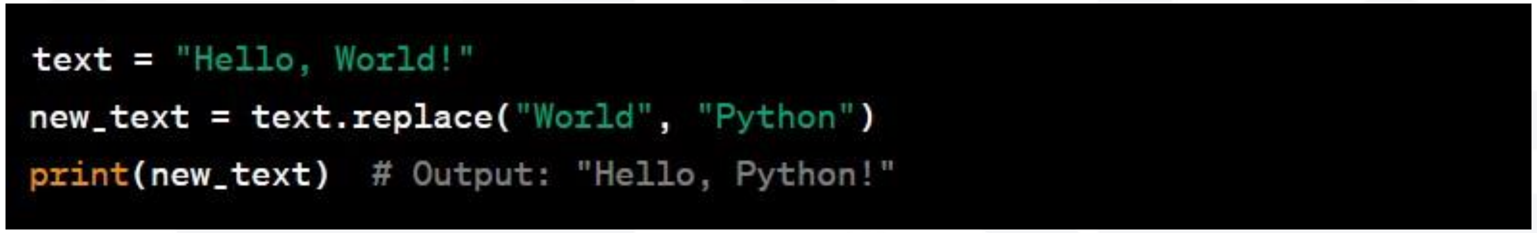
6. join(): Joins a list of strings into a single string using a specified separator.

```
words = ['Hello', 'Python!']
text = ", ".join(words)
print(text)  # Output: "Hello, Python!"
```

String Function

7. replace():

- Replaces occurrences of a substring with another substring.



```python
text = "Hello, World!"
new_text = text.replace("World", "Python")
print(new_text)  # Output: "Hello, Python!"
```

- You Can get info about more function from python documentation
  - https://docs.python.org/3/library/stdtypes.html#text-sequence-type-str

Any Question?

NeuroTech

Thanks

NeuroTech