

# Introduction To Python

Lec-1

# What is Python ??

- Python is a **high-level, interpreted, general-purpose programming language** known for its **simplicity, readability, and versatility**.
- It allows you to write programs that are clear and logical even for complex problems.
- Python's **easy-to-learn syntax** makes it an **ideal choice for beginners** while its **powerful libraries and frameworks** support advanced applications in various domains like **web development, data science, and automation**.
- Python supports multiple programming paradigms, **including procedural, object-oriented, and functional programming**, making it a flexible tool for different programming needs and styles.



# A Brief History of Python

1

**Late 1980s**

Python was conceived by **Guido van Rossum**, a Dutch programmer, at CWI (Centrum Wiskunde & Informatica) in the Netherlands.

2

**1991**

First version **Python 0.9.0** was released. It already had exception handling, functions, and modules.

3

**2000**

**Python 2.0** came with new features like garbage collection and Unicode support.

4

**2008**

**Python 3.0** was released — not backward-compatible with Python 2, but designed for cleaner, modern syntax.

5

**2020**

**Python 2 officially retired**; now only **Python 3** is **maintained and developed**.

As of today, Python is one of the **most popular programming languages** worldwide — used by **Google, Netflix, NASA, YouTube**, and many others.



## Fun Fact:

Guido van Rossum named it “**Python**” after the British comedy show “*Monty Python’s Flying Circus*”, not after the snake

# Why to learn Python ??

**1** **Easy to Learn:** Python has a simple and straightforward syntax, making it an excellent language for beginners.

**2** **Versatility:** Python is a versatile language and is used in a wide range of applications- web development, scientific computing, data analysis, artificial intelligence and more.

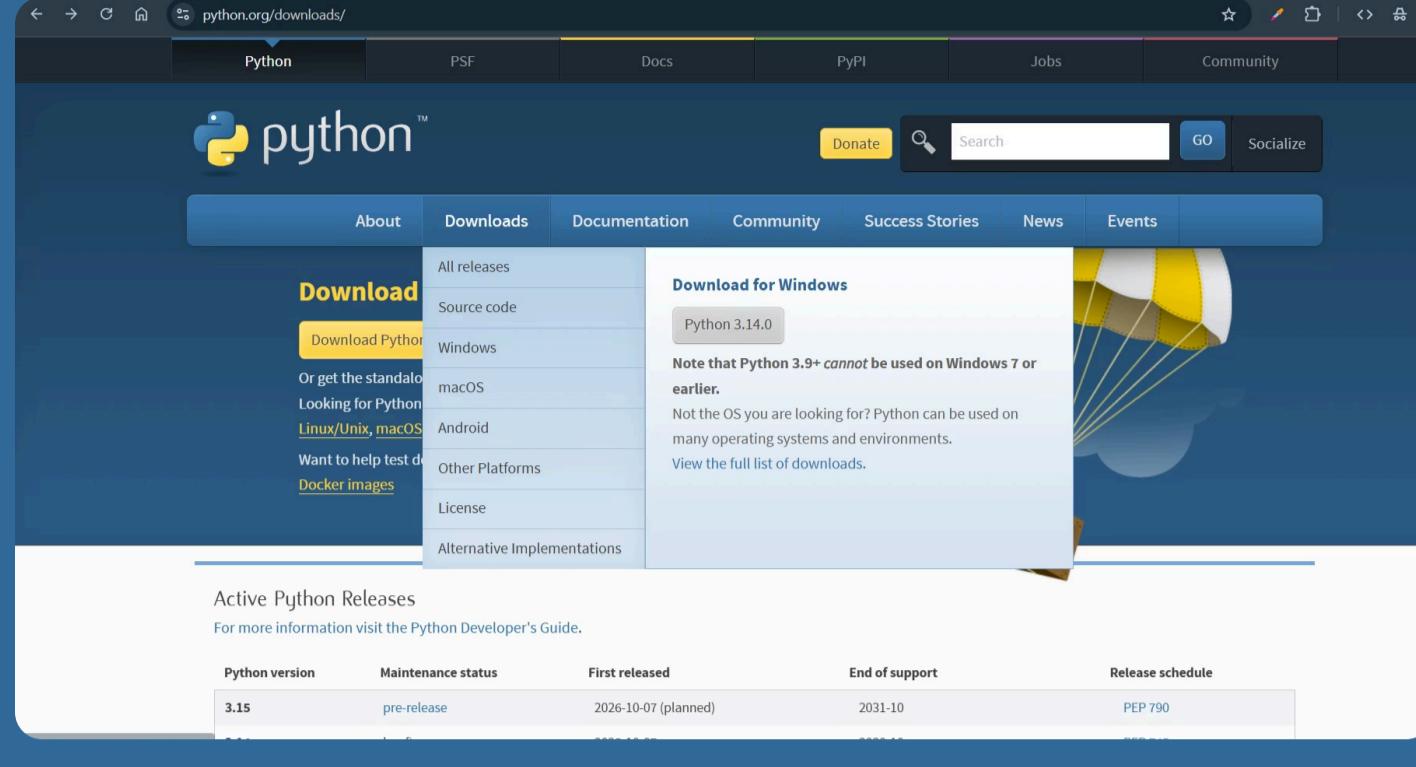
**3** **Large and Active Community:** Python has a large and active community of users and developers, which means that there is a wealth of resources and support available.

**4** **Good for rapid prototyping:** Python's simplicity makes it possible to quickly test ideas and iterate on them, saving time and resources compared to compiled languages.

**5** **Job Opportunity:** Python has high demand in the job market, particularly in fields of data science, machine learning, web development, genAI and Agentic Ai development.

# Downloading and Installing Python

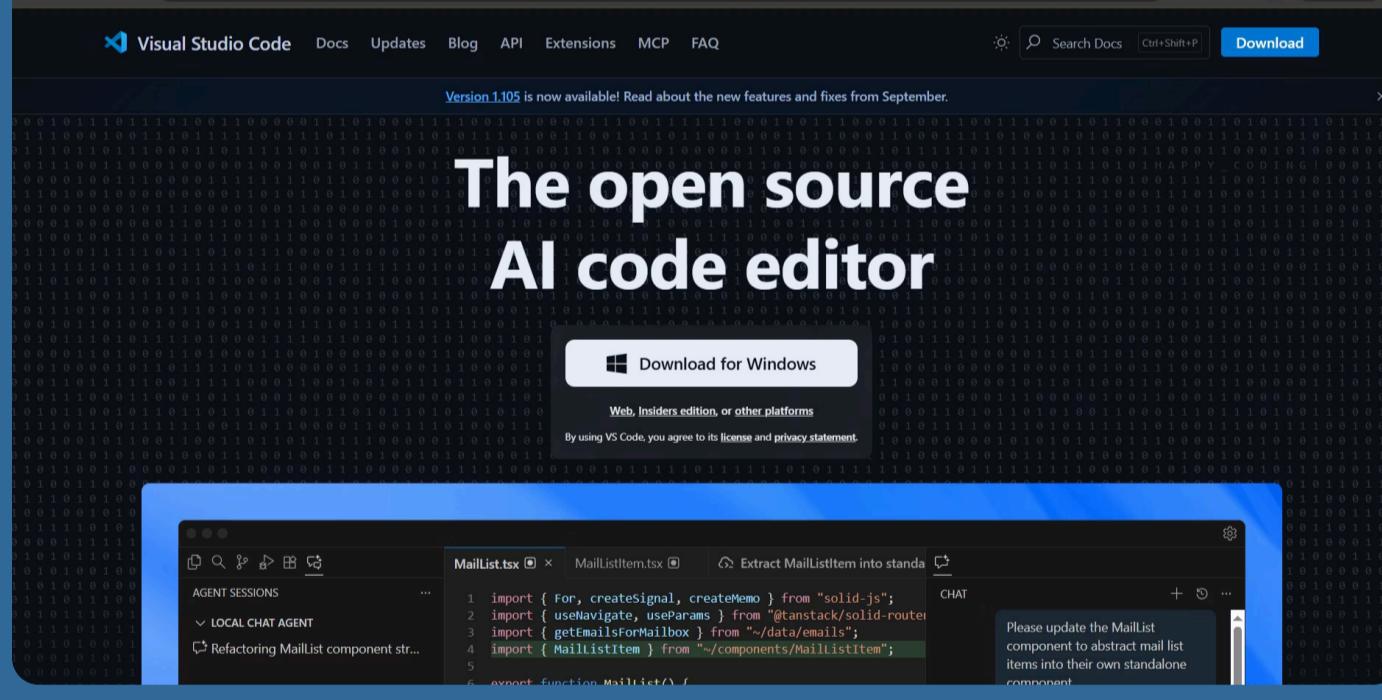
- **Step-1:** Visit- [Download Python](https://www.python.org/downloads/)
- **Step-2:** Hover over **Downloads** option and download the latest version according to your OS



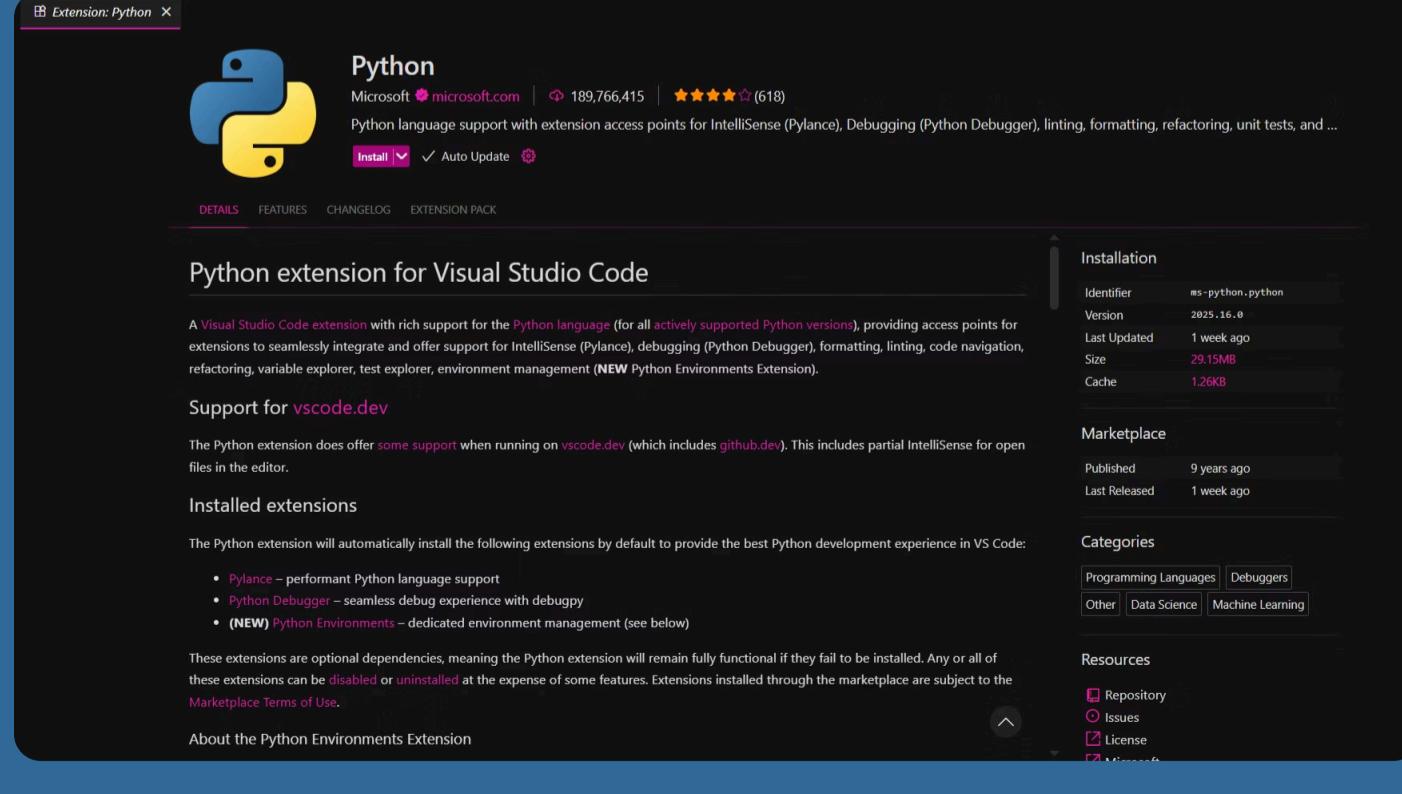
- **Step-3:** Click on **Install Now**



- **Step-4:** Go to <https://code.visualstudio.com/>, download and install it.



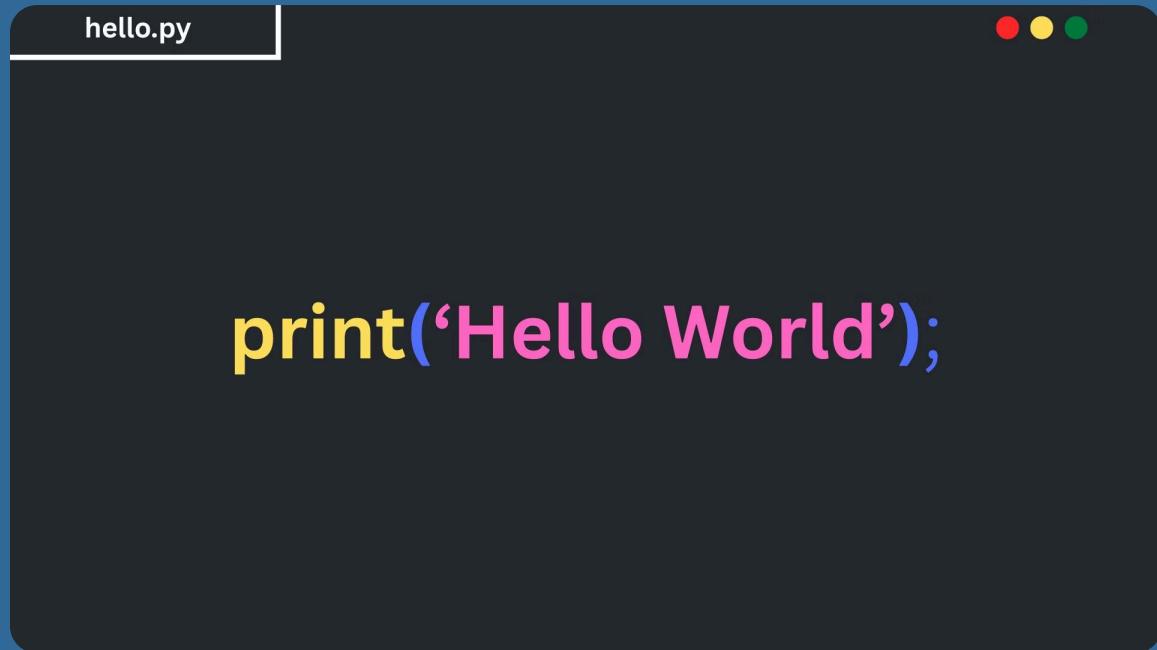
- **Step-5:** Download the following VS Code extension for Python in VS Code.



- **Step-6:** Open **Command Prompt** and run the below command in it, if it results the following way then your python installation is successful.

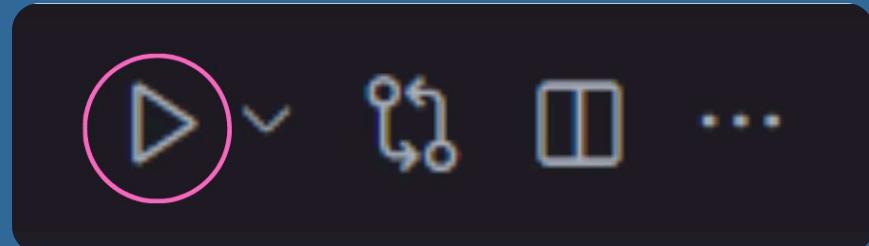
```
C:\Users\DELL>python --version
Python 3.14.0
```

# Hello World in Python



- To run Python program in Vs code there are 3 ways:

## 1. Use Run Button on Right Top Corner of VS Code editor



## **2. Inside terminal by running the command in following way**

```
PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS

● PS D:\Gen and Agentic AI\Python-tut-complete> cd 1_Intro-to-Python
● PS D:\Gen and Agentic AI\Python-tut-complete\1_Intro-to-Python> python hello.py
Hello world
○ PS D:\Gen and Agentic AI\Python-tut-complete\1_Intro-to-Python>
```

# Single and Multi Line Comments

- In Python, **comments are used to add explanatory notes within the code**, which are **ignored by the interpreter** during execution.
- Following is the way to give **Single Line Comment**:

```
# Single Line Comment
```

- Following is the way to give **Multi Line Comment**:

```
"""
Multi Line Comment
"""
```

- Python **does not really have a syntax for multiline comments** but Python will **ignore string literals that are not assigned to a variable**, you can add a multiline string (triple quotes) in your code, and place your comment inside it:
- As long as the string is not assigned to a variable, **Python will read the code, but then ignore it, and you have made a multiline comment.**

# Data Types in Python

- Data Types are used to define **Type of a Data** stored inside any variable.
- In Python **built-in data types** are categorized in the following way:

<b>Text Type</b>	str
<b>Numeric Type</b>	int, float, complex
<b>Sequence Type</b>	list, tuple, range
<b>Mapping Type</b>	dict
<b>Set Types</b>	set, frozenset
<b>Boolean Type</b>	bool
<b>Binary Types</b>	bytes, bytearray, memoryview
<b>None Type</b>	NoneType

# Variables in Python

- Variables are **named memory locations** used to **store information within a program**.

## Creating Variables

- In Python, you create a variable **just by assigning it something**. No need to declare it beforehand.

```
a = 10  
d = "Joe"  
print(a) # will print 10  
print(d) # will print Joe
```

- Variable names are **Case Sensitive**.

```
a = 4  
A = "Borris"  
#This will create 2 different variables and a will not override A.
```

- We **don't have to declare a particular datatype while declaring a variable**, variables **automatically adapt to the type of data they hold**, and **can be reassigned to data of any other type**.

```
x = 20 # x is now a variable of type int  
x = 'Harry' # x is now a variable of type string  
print(x)
```

- String variables can be **declared either by using single or double quotes**.

## Casting

- If you want to **specify the data type of a variable**, this can be done with casting using **constructor function**.

```
x = str(3) # x will be '3'  
y = int(3) # y will be 3  
z = float(3) # z will be 3.0
```

## Get the type of variable

- We can get the actual data type of any variable using **type()** function.

```
name = 'Donald'  
empid = 100  
print(type(name)) # will print <class 'str'>  
print(type(empid)) # will print <class 'int'>
```

# Taking Input From User in Python

- Python allows to ask the user for input in the following way using **input()** function.

Eg:

```
print("What's your name ?")
name = input() # input: John
print("Hello" + name) # output: HelloJohn
```

- In above example, as we saw the user had to input their name on a new line to assign it to variable name. But the **input()** also have a **prompt** parameter, **which acts as a message you can put in front of the user input, on the same line.**

Eg:

```
age = input("Enter your age: ")
print('The Age is ' + age)
```

output:

```
Enter your age: 20
The Age is 20
```

- ❑ • Generally Python stops executing when it comes to the **input()**, and continues when the user has given some input.

## Input Number

- The **input()** treats **input from user by default as a String**.
- In order to **convert it** of some different type we have to use **constructor functions** which discussed above.

Eg:

```
birth_year = input("Enter your birth year") # This will consider the type of input as Str.
age = 2025 - int(birth_year) # This will convert it into int
print(age) # will print the result
```

# Strings in Python

- Strings in python are surrounded by either **single quotation marks**, or **double quotation marks**.
- You can display a string literal with the `print()` function:

```
Eg: print('Hello')  
     print("Hi")
```

- String is assigned to a variable by writing it's name and then followed by **assignment operator (=)**

```
Eg: a = "Hello World"
```

- Multiline string is assigned to a variable by using **three single or double quotation marks (" ") / (' ')**

```
Eg: z = """ My name  
       is  
       John  
       Doe """
```

## **type() method**

```
Eg: a = "hello"  
     print(type(a)) # will print <class 'str'>
```

## **Length of an String / len() method**

- In order to get length of any string, we use **len()** method.

```
Eg: a = 'John Doe'  
     print(len(a)) # will print 8
```

# String slicing

- Using String Slicing we can return a **character** or **range of character** including in the given string.
- In order to get our desired range of a string- we provide **starting index : ending index** inside a **square bracket**.
- It always **returns the string up till a character before the declared ending index i.e. (ending index-1)**.

```
Eg: a = 'John Doe'  
print(a[2:6]) # will print 'hn D'
```

## Slice from start

- If we **do not give starting index** of a string while slicing, **it would by default consider starting index as 0th index** and would go up till the **index just before the declared ending index**.

```
Eg: a = 'John Doe'  
print(a[:5]) # will print John_
```

## Slice from End

- If we **do not give ending index** of a string while slicing, **it would get the characters from the declared starting index to all the way to the end of the string**.

```
Eg: a = 'John Doe'  
print(a[2:]) # will print 'hn Doe'
```

## Negative Indexing

- In case of negative indexing, **it considers the last character of the string as the first index (-1) and goes further towards left**.

```
Eg: a = 'John Doe'  
print(a[-6:-2]) # will print 'hn D'
```

# Formatted Strings

- In python Strings and numbers cannot be directly combined.
- Thus, in order to overcome this we use **Formatted Strings**.

Eg: age = 24

```
print("My age is: " + 24)  
# The above line will result in error as 24 (int) cannot be directly  
combined with "My age is" (string)
```

- To specify a string as an **f-string**, simply put an **f in front of the string literal**, and **add curly braces {} as placeholders for variables** and other operations.

Eg: age = 24

```
print(f"My age is: {age}") # This will print "My age is: 24"
```

- A **placeholder can contain variables, operations, functions, and modifiers** to format the value.

Eg: price = 2000

```
print(f"The Price of this product is: {price})  
# Above line will print- "The Price of this product is: 2000"  
print(f"The Price after 50% discount is: {price/2})  
# Above line will print- "The Price after 50% discount is: 1000.0
```

# String Methods in Python

- Python contains a list of **built-in methods** which can be used on **strings**.

## Note:

- All these string methods **returns new value**. They **do not change the original string**.
- These methods are accessible using **dot operator(.)**.

### 1. .upper() :

- This method **returns a string where all characters are in upper case**.
- Symbols and Numbers** are **ignored** by this method.

Eg:

```
a = "John"  
print(a.upper()) # will print: JOHN
```

### 2. .capitalize() :

- This method **returns a string where the first character is upper case, and the rest is lower case**.

Eg:

```
a = 'john'  
print(a.capitalize()) # will print: John
```

### 3. .lower() :

- This method **returns a string where all characters are lower case**.
- Symbols and Numbers** are **ignored** by this method.

Eg:

```
a = "John"  
print(a.lower()) # will print: john
```

### 4. .find() :

- This method **finds the first occurrence of the specified value**.
- It **returns the index at which the value is**.
- It **returns -1 if the value is not found**.
- It is a **case-sensitive** method.

Eg:

```
a = 'John Doe'  
print(a.find('o')) # will print: 1  
print(a.find('O')) # will print: -1  
print(a.find('s')) # will print: -1  
print(a.find('Doe')) # will print: 6
```

### 5. .replace() :

- This method **replaces a specified phrase with another specified phrase**.
- It is a **case-sensitive** method.

Eg:

```
a = "Michael Edward"  
print(a.replace("Edward","Jackson")) # prints: "Michael Jackson"
```

- We can also **specify how many occurrences of the old value we want to replace** as the **third parameter** to the replace method.

Eg:

```
a = "A friend in need is a friend in deed"  
print(a.replace('friend','brother', 1))  
# Above line will print: "A brother in need is a friend in deed".
```

# String Methods in Python

## 6. `.split()`:

- This method **splits a string into a list**.

Eg:

```
a = 'Apple Banana Orange'  
print(a) # print: "Apple Banana Orange"  
print(a.split()) # print: ['Apple', 'Banana', 'Orange']
```

- Separator can be specified, **default separator in any whitespace**.

Eg:

```
a = 'apple@banana@orange'  
print(a) # print: apple@banana@orange  
print(a.split('@')) # print: ['apple', 'banana', 'orange']
```

## 7. `.startswith()`:

- This method **returns True if the string starts with the specified value, otherwise returns false**.
- This is a **case-sensitive** method.

Eg:

```
a = "rock and roll"  
print(a) # will print "rock and roll"  
print(a.startswith('rock')) # returns: True  
print(a.startswith('roll')) # returns: False  
print(a.startswith('Rock')) # returns: False
```

## 8. `.endswith()`:

- This method **returns True if the string ends with the specified value, otherwise returns false**.
- This is a **case-sensitive** method.

Eg:

```
a = "rock and roll"  
print(a) # will print "rock and roll"  
print(a.endswith('rock')) # returns: False  
print(a.endswith('roll')) # returns: True  
print(a.endswith('Roll')) # returns: False
```

## 9. `.strip()`:

- This method **removes whitespaces at the beginning or either at the end of the string**.

Eg:

```
a = " banana "  
print(f"My favorite fruit is: {a.strip()}")  
# above line will print- "My favorite fruit is banana"
```