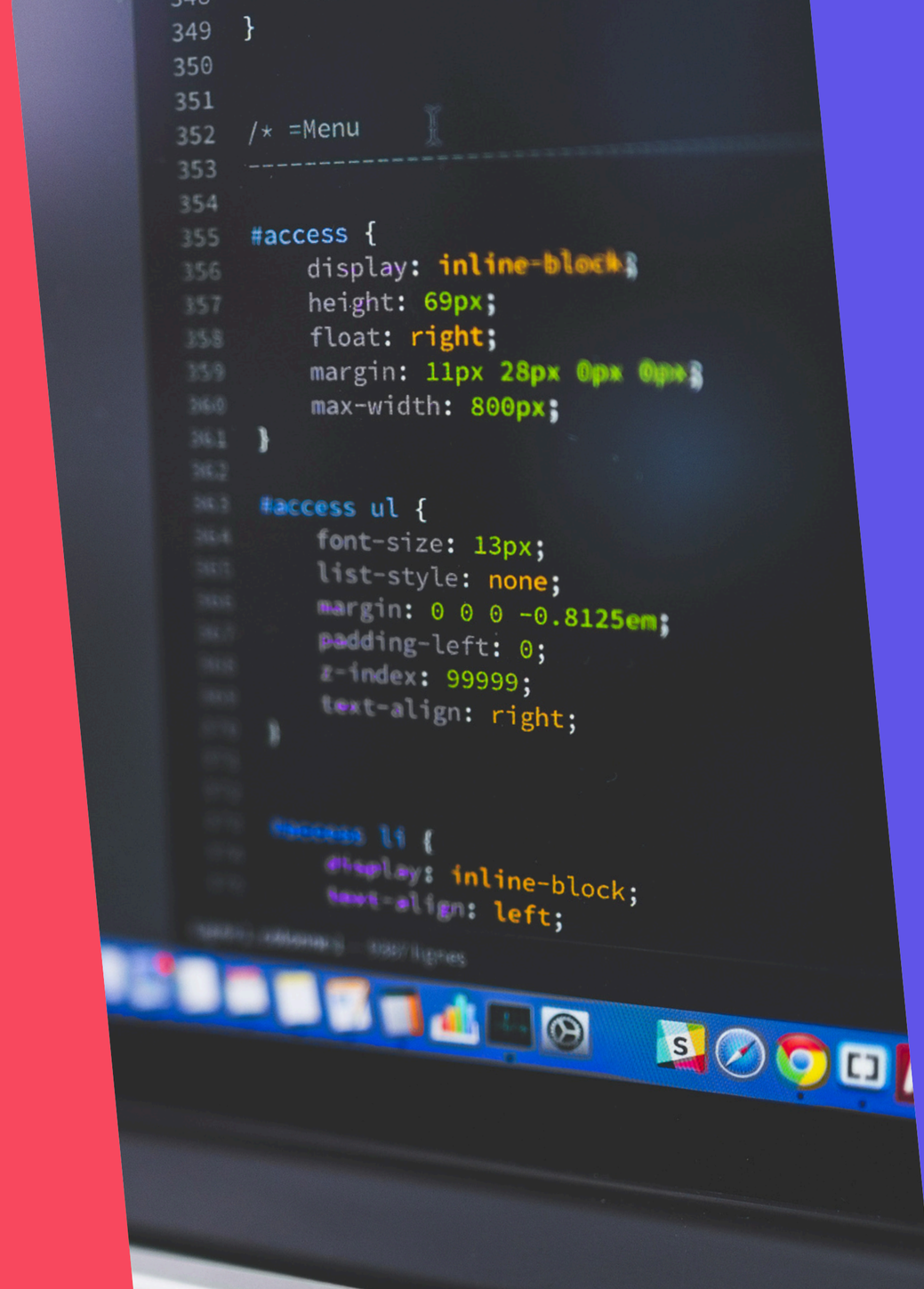


WBS CODING SCHOOL

Python: Basics

WBS CODING SCHOOL

Agenda

- What is Python?
- Data Types
- Variables
- Functions and Methods
- String Manipulation
- Operators

Learning outcomes

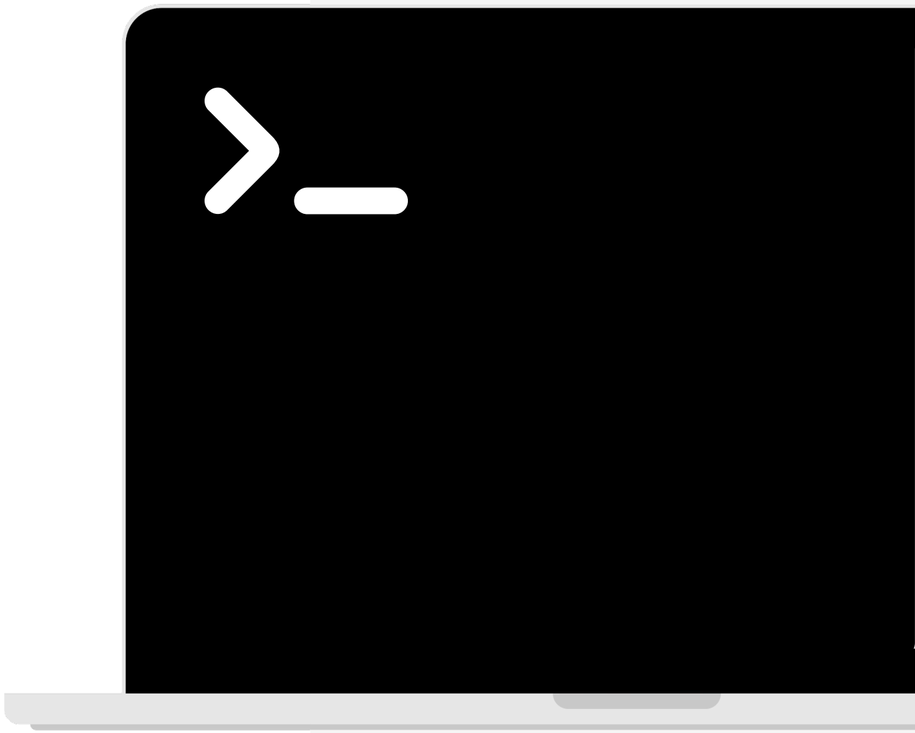
WBS CODING SCHOOL

Skills you will acquire:

1. Understand what is Python
2. Outline the basic data types in Python
3. Know how to declare variables
4. Understand how to manipulate strings

What is Python?

History and Characteristics



A Brief History of Python

- **Created by:** Guido van Rossum
- **Released:** In 1991
- **Origin:** Developed as a successor to a language called ABC. Guido wanted something more powerful and easier to use.
- **Why "Python"?** The name actually comes from the British comedy group Monty Python, as Guido was a fan of their show.

Since then, Python has grown into one of the most widely used programming languages in the world. It powers everything from web apps to artificial intelligence.

Key Characteristics of Python

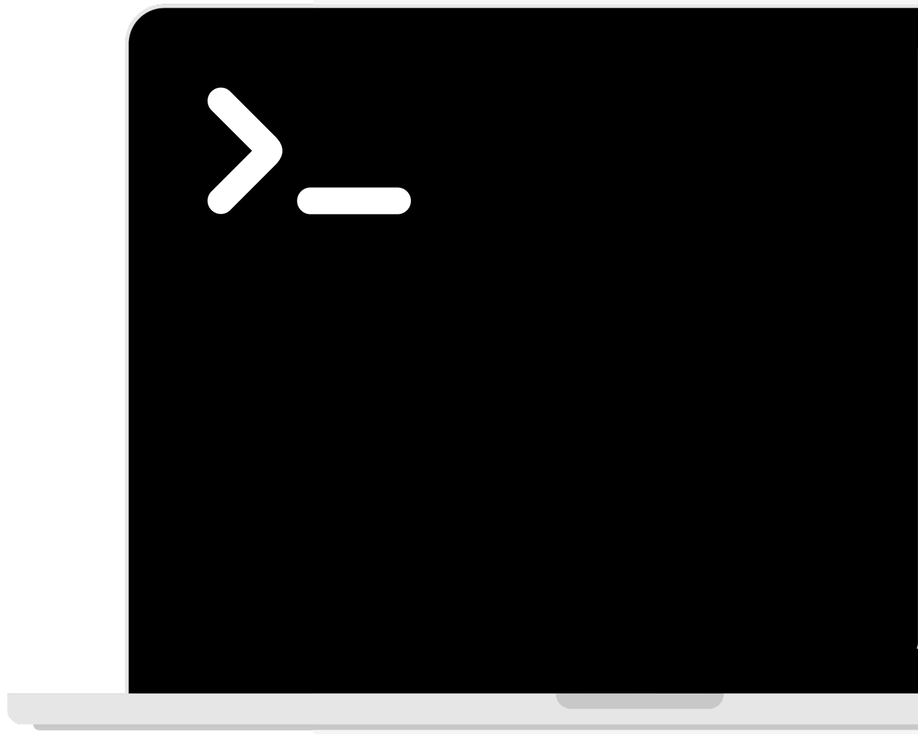
- **Easy to Read and Write** - Python looks like regular English.
- **Interpreted Language** - You don’t need to compile it before running. Python reads and runs the code line-by-line.
- **High-Level Language** - You don’t need to worry about complex details like memory management. Python handles that for you.
- Dynamicly typed Language -
- **Versatile and Multi-purpose** - Python is used in web development, data science, machine learning, automation, etc.
- **Huge Community and Library Support**
- **Cross-Platform** - Python runs on Windows, Mac, Linux — basically anywhere.

Why Use Python?

- Easy for beginners
- In-demand skill in tech jobs
- Great for experimenting and learning programming concepts
- Widely used in industry and research

Data Types

How Python Handles Data



What Are Data Types?

In programming, a data type tells the computer what kind of value you're working with.

Think of it like labeling a box:

- A box labeled “numbers” holds numbers.
- A box labeled “text” holds words.
- A box labeled “list” holds multiple items.

Python needs to know the type of data so it knows how to store and handle it.

Main Data Types in Python

Data Type	Example	What It Is
int (Integer)	5	Whole numbers (positive or negative)
float	3.14	Decimal numbers (floating-point)
complex	2 + 3j	Real part + imaginary part
str (String)	"Hello"	Text (inside quotes)
bool (Boolean)	True, False	Represents Yes/No or On/Off
list	[1, 2, 3]	A collection of values (ordered,
tuple	(4, 5, 6)	Like a list, but **unchangeable**
dict (Dictionary)	{"name": "Alice", "age": 25}	Stores data in **key-value pairs**
set	{1, 2, 3}	A collection of unique values (unordered)
none	none	Absence of value

Example in Code

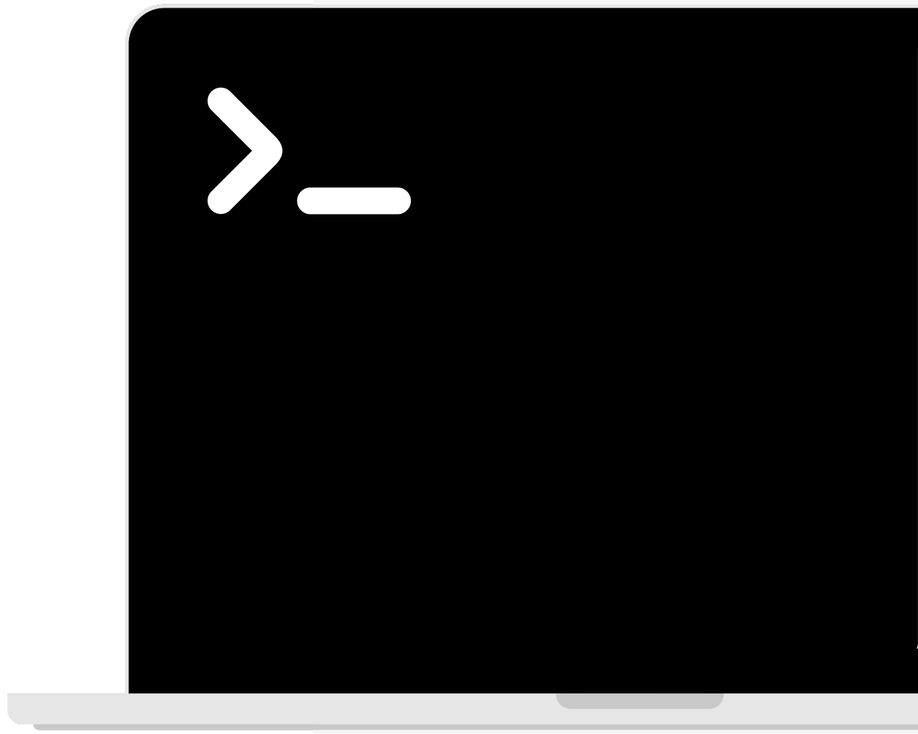
```
age = 25                    # int
pi = 3.14                   # float
name = "Alice"              # str
is_student = True           # bool
grades = [90, 85, 92]       # list
position = (1, 2)           # tuple
person = {"name": "Bob", "age": 30}  # dict
unique_numbers = {1, 2, 3}  # set
```

How to Check a Data Type in Python

```
print(type(name))  # Output: <class 'str'>
```

Variables

Usage And Conventions



What is a Variable?

A variable is like a labeled box where you can store data (like a number, word, or list). You can name the box and reuse it throughout your code.

Python is a dynamically typed language, which means when declaring a variable you don’t have to specify its data type, but you can:

- **Implicit Typing:** Python assigns the type automatically
- **Explicit Typing:** You specify the type using casting functions like int(), float(), or str() → **Casting**

Declaring A Variable

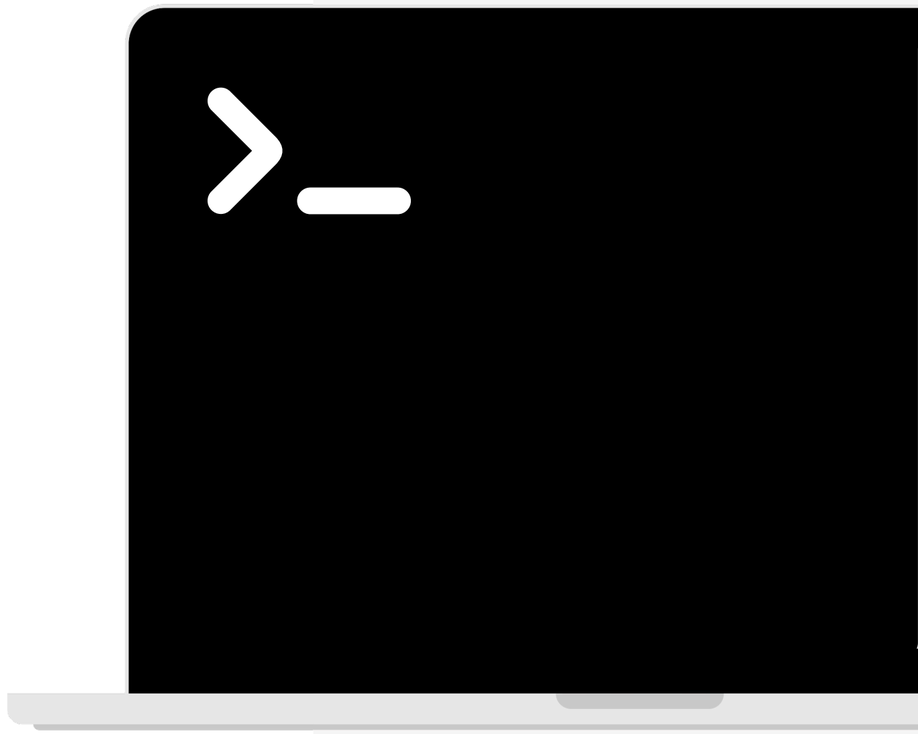
- **Give it a name following the conventions:**
  - Snake_case: separating words with an underscore
  - It must start with a letter or _
  - It cannot start with a number
  - It cannot use spaces or symbols
- **Give it a value using =**

Declaring A Variable

```
age = 25                    # int
pi = 3.14                   # float
name = "Alice"              # str
is_student = True           # bool
grades = [90, 85, 92]       # list
position = (1, 2)           # tuple
person = {"name": "Bob", "age": 30}  # dict
unique_numbers = {1, 2, 3}  # set
```

Utilities

Imported And In-built Functions



Modules

Utilities or libraries that can be imported in and provide pre-created functions or methods. A good example is the random module:

- **random.randint(1, 10)** → creates a random integer between the two numbers you pass inside the parentesis separated by a coma, including those numbers.
- **random.random()** → creates a random float between the 0 and 1.
- **random.uniform(1, 5)** → creates a random float between the two numbers you pass inside the parentesis separated by a coma, including those numbers.

In-Built Functions

Built-in functions in Python are predefined functions that come with the language — you can use them without importing any libraries.

Function	Description	Example
print()	Displays output to the screen	print("Hello")
len()	Returns the length of a collection	len("Python") → 6
type()	Returns the type of an object	type(42) → <class 'int'>
int(), str(), float()	Converts values between types	int("7") → 7
input	Takes user input as a string	name = input("Your name: ")

In-Built Functions

Function	Description	Example
max(), min()	Finds the largest/smallest value	max([3, 7, 1]) → 7
sum()	Adds up values in a collection	sum([1, 2, 3]) → 6
range()	Creates a range of numbers	range(5) → 0 to 4
abs()	Returns absolute value of a number	abs(-10) → 10
round()	Rounds a number to the nearest integer or decimal	round(3.1415, 2) → 3.14

In-Built Methods in Python

In-built **methods** are **functions** that belong to specific objects (like strings, lists, dictionaries, etc.). These methods are "built into" the object types and are used to perform common operations directly on the object.

Think of them as actions that an object knows how to perform.

String Methods

Method	Description	Example
.upper()	Converts to uppercase	"hi".upper() → "HI"
.lower()	Converts to lowercase	"Hi".lower() → "hi"
.strip()	Removes whitespace	" hello ".strip() → "hello"
.replace()	Replaces part of a string	"a b".replace(" ", "") → "ab"

List Methods

Method	Description	Example
.append(x)	Adds item to end	[1,2].append(3) → [1,2,3]
.pop()	Removes and returns last item	[1,2,3].pop() → 3
.sort()	Sorts the list in-place	[3,1,2].sort() → [1,2,3]

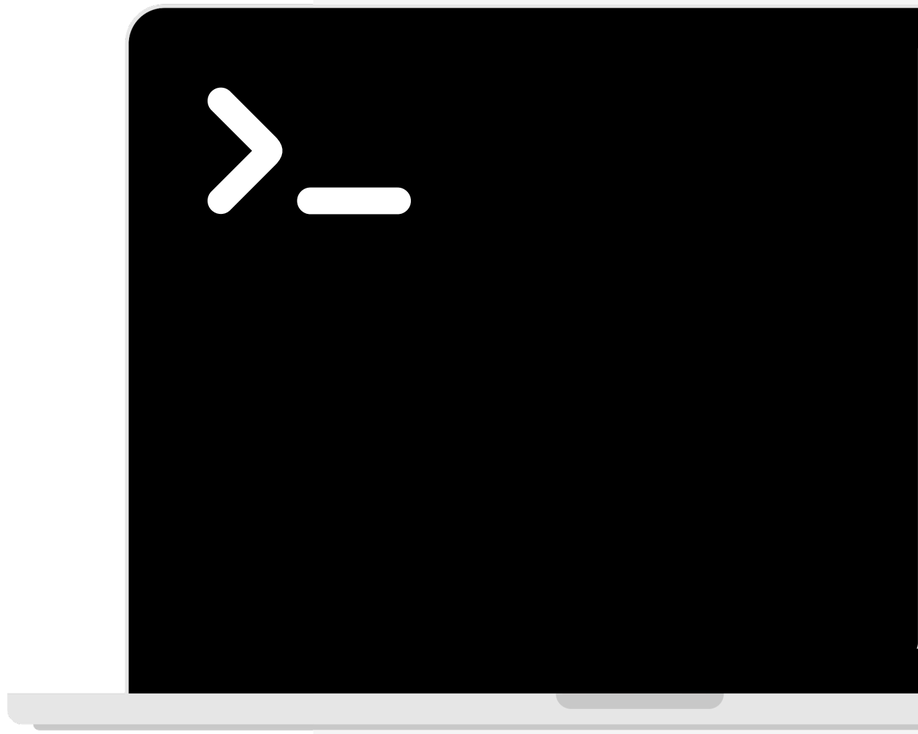
Dictionary Methods

Method	Description	Example
.keys()	Returns a list of keys	{"a": 1}.keys()
.values()	Returns a list of values	{"a": 1}.values()
.get(key)	Returns value for key safely	{"a":1}.get("a") → 1

Chapter 2: Python

String Manipulation

How To Declare And Alter Strings



Strings are...

- The text data type in Python
- Written by using “” or ‘’ for one line, three ‘ for multi-line
- They are a linear data structure where all elements are arranged sequentially.
- They are immutable

```
greeting = 'Hello, "Class 24"!'
name = "Alice"
age = '25'
presentation = f"I'm {name} and I'm {age} old."
multi_line = """Hi my name is Maria
and I am writing this in several
lines"""
```

Strings as Linear Data Structures

- Each element in a string has an **index** or position.
- The index of the 1st element is always 0.
- By using indexing you can:
  - select individual characters
  - select a sequence of characters
- Using negative numbers allows you to get the values at the end of the string (-1 is the last character)
- Each element in a sequence can be looped.
- The elements inside the sequence can be counted by using the len() function

Strings as Linear Data Structures

```
greeting = 'Hello, "Class 24"!'

gretting[0] # -> selects the element on the first position (H)
gretting[-1] # -> selects the last element (!)
gretting[8:13] # -> selects a portion of the string from index 8 to 12 (Class)
gretting[:5] # -> selects a portion of the string starting from index 0 to 4 (Hello)
gretting[7:] # -> selects a portion of the string starting from index 4 to the end ("Class 24"!)
```

String Concatenation

- You can concatenate one string after another by using:
  - the operator + between one string and the other
  - an fstring

```
class_name = 'Class 24'

greeting = 'Hello, "' + class_name +'"!'

greeting2 = f'Hello, {class_name}'
```

String as Immutable

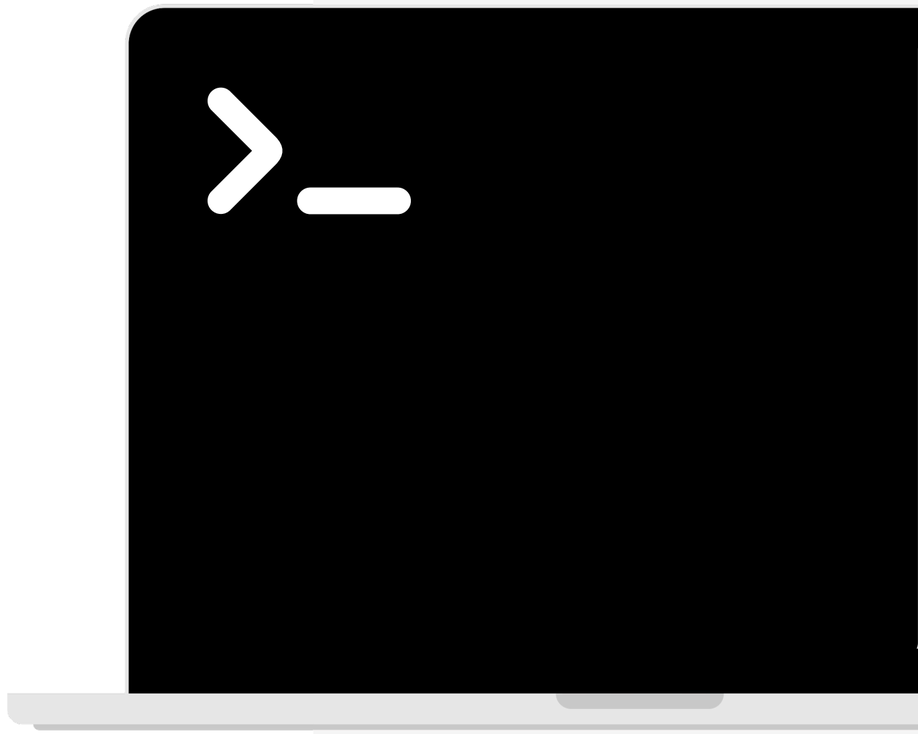
Strings are immutable objects in Python, they cannot change. However through **string methods** we can create **copies** of the original string with the desired changes:

```
class_name = 'Class 24'

class_name.upper()
class_name.lower()
class_name.strip()
```

Operators

What Are Operators in Python?



What are operators?

Operators are special symbols or keywords that perform operations on values or variables. They are used to perform tasks like arithmetic, comparison, assignment, and more.

Operators are essential tools for:

- Doing calculations
- Making decisions
- Managing values and logic
- Writing clean and efficient code

Type of operators

- **Arithmetic Operators** → Used for basic math operations.
- **Assignment Operators** → Used to assign values to variables.
- **Comparison Operators** → Used to compare values.
- **Logical Operators** → Used to combine conditional statements.
- **Identity Operators** → Check if two variables refer to the same object.
- **Membership Operators** → Check if a value is in a sequence (like list, string).
- **Bitwise Operators (Advanced)** → Operate on bits (0s and 1s).

WBS CODING SCHOOL

Lesson completed