# Google Colab

An Overview and Key Advantages

- **Website address:** https://colab.research.google.com

**Key advantages of using Google Colab for academic and research purposes:**

- Free online access to high-performance computing hardware, including Graphics Processing Units (GPUs) and Tensor Processing Units (TPUs), eliminating the need for expensive local infrastructure.
- Seamless storage and management of Jupyter notebooks and associated files directly in Google Drive, ensuring persistent access and automatic versioning.

# Google Colab

- Seamless storage and management of Jupyter notebooks and associated files directly in Google Drive, ensuring persistent access and automatic versioning.
- Pre-installed access to a comprehensive suite of widely used scientific computing and machine learning libraries — such as NumPy, Pandas, Matplotlib, TensorFlow, and PyTorch — without any local installation or environment configuration requirements.
- Facilitated collaboration through easy sharing and real-time simultaneous editing of the same notebook by multiple users, making it particularly suitable for team-based research, teaching, and academic projects.


def main()
printi"Hello Warld")
if _hame _== "_main"):
main()
HELLO WORLD

# Print statement

- print()
- Displays messages or output in the console
- Multiple arguments can be placed in a single print statement, separated by commas (,)
- f-string formatting can also be used inside print
- Let's look at a few examples: colab.research.google.com

# Variables

## Variables in Python

### What they do?

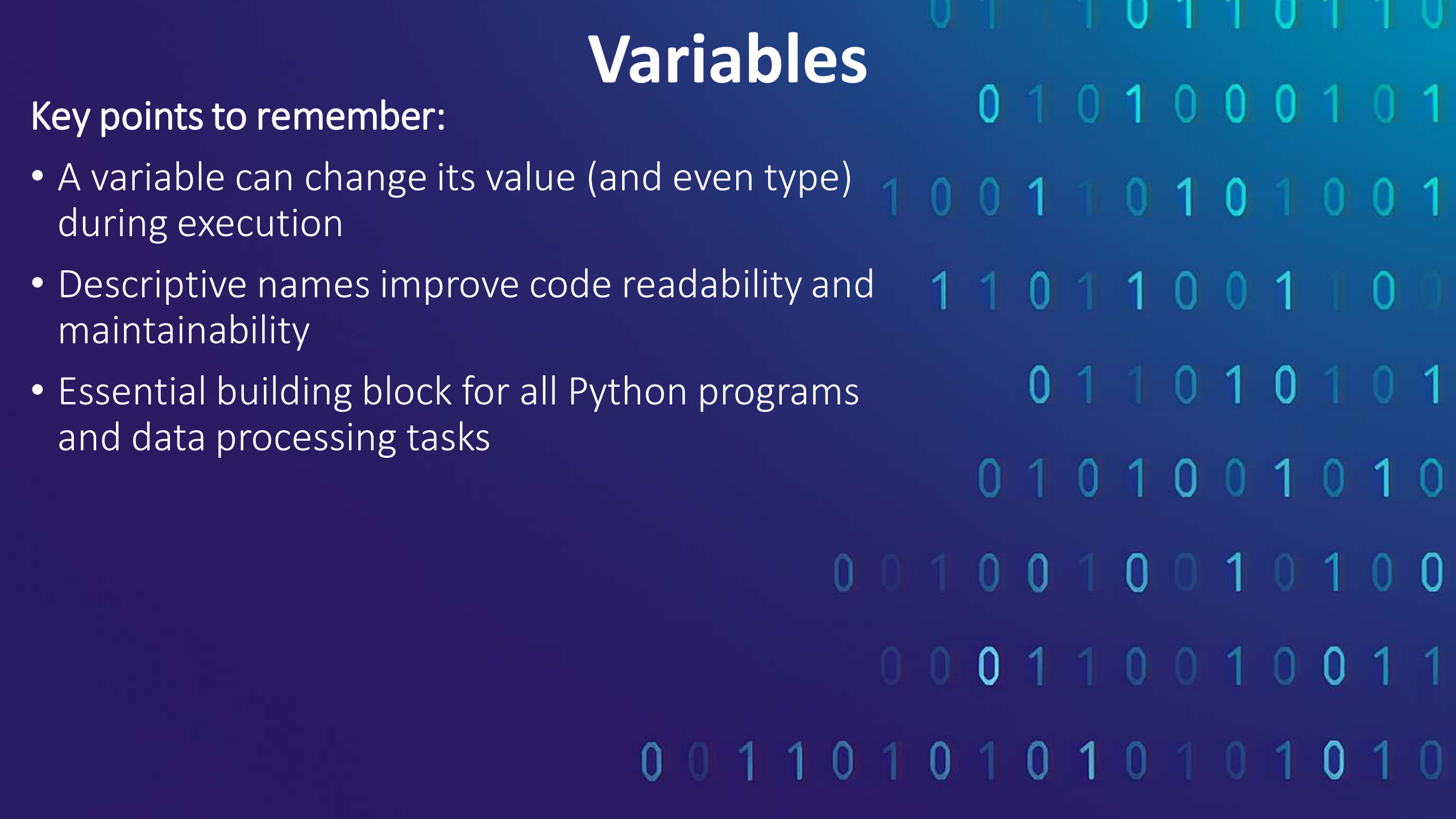
- Storing and Managing Data

- Variables are named containers used to store data values in memory
- No explicit type declaration is required – Python is dynamically typed
- Assignment is performed using the (=) operator
- Variable names are case-sensitive and should follow a special naming conventions

# Variables

Key points to remember:

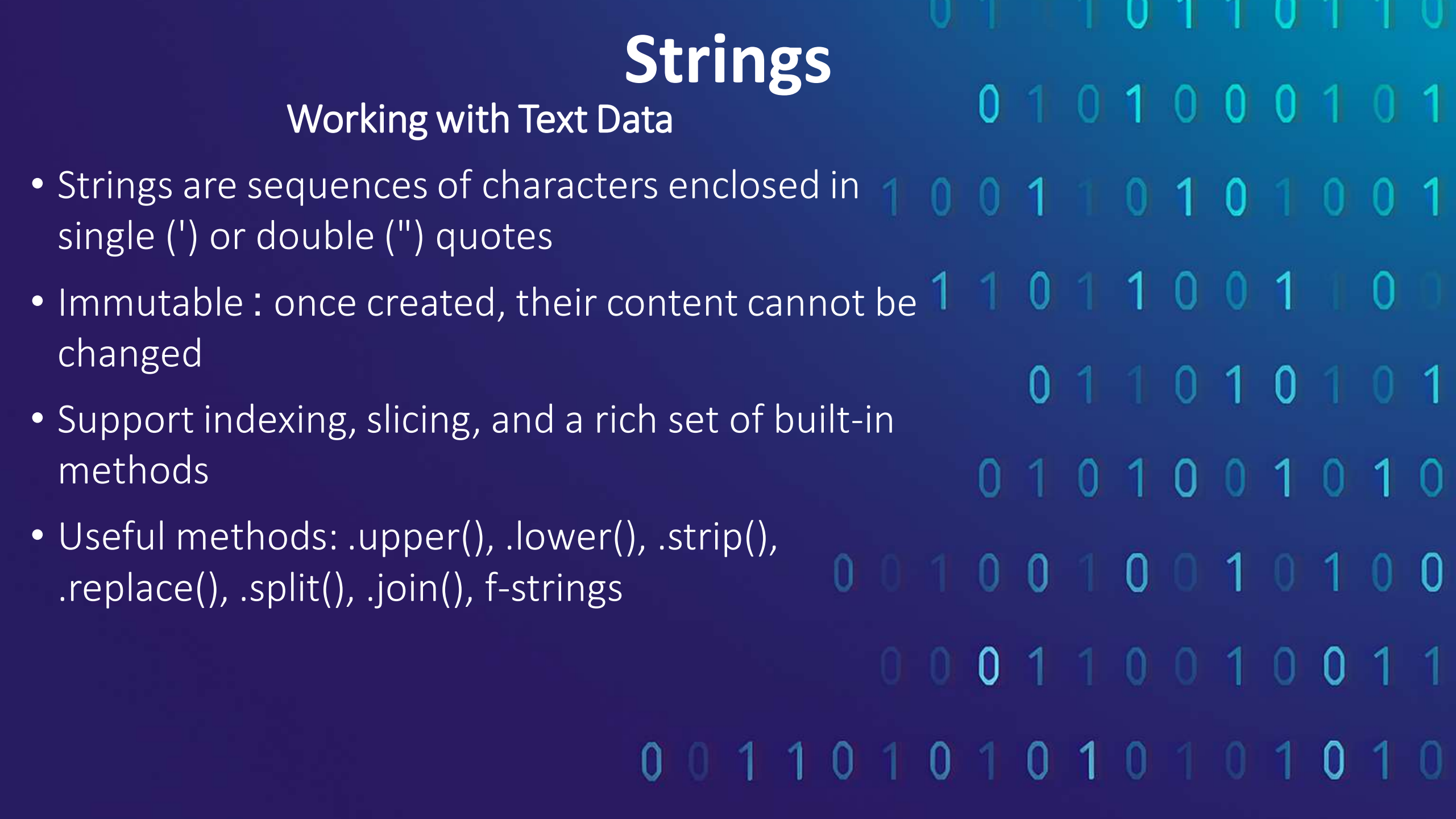
- A variable can change its value (and even type) during execution
- Descriptive names improve code readability and maintainability
- Essential building block for all Python programs and data processing tasks



# Strings

## Working with Text Data

- Strings are sequences of characters enclosed in single (') or double (") quotes
- Immutable : once created, their content cannot be changed
- Support indexing, slicing, and a rich set of built-in methods
- Useful methods: .upper(), .lower(), .strip(), .replace(), .split(), .join(), f-strings



# Arithmetic Operators

## How to do simple math in python?

Performing Mathematical Calculations Operator:

**Operator Table:**

| Operator | Description | Example | Result |
|---|---|---|---|
| `+` | Addition | `8 + 5` | 13 |
| `-` | Subtraction | `20 - 7` | 13 |
| `*` | Multiplication | `4 * 3` | 12 |
| `/` | Division (float) | `10 / 3` | 3.333... |
| `//` | Floor division (integer) | `10 // 3` | 3 |
| `%` | Modulus (remainder) | `10 % 3` | 1 |
| `**` | Exponentiation | `2 ** 4` | 16 |

# Arithmetic Operators

## How to do simple math in python?

Performing Mathematical Calculations Operator:

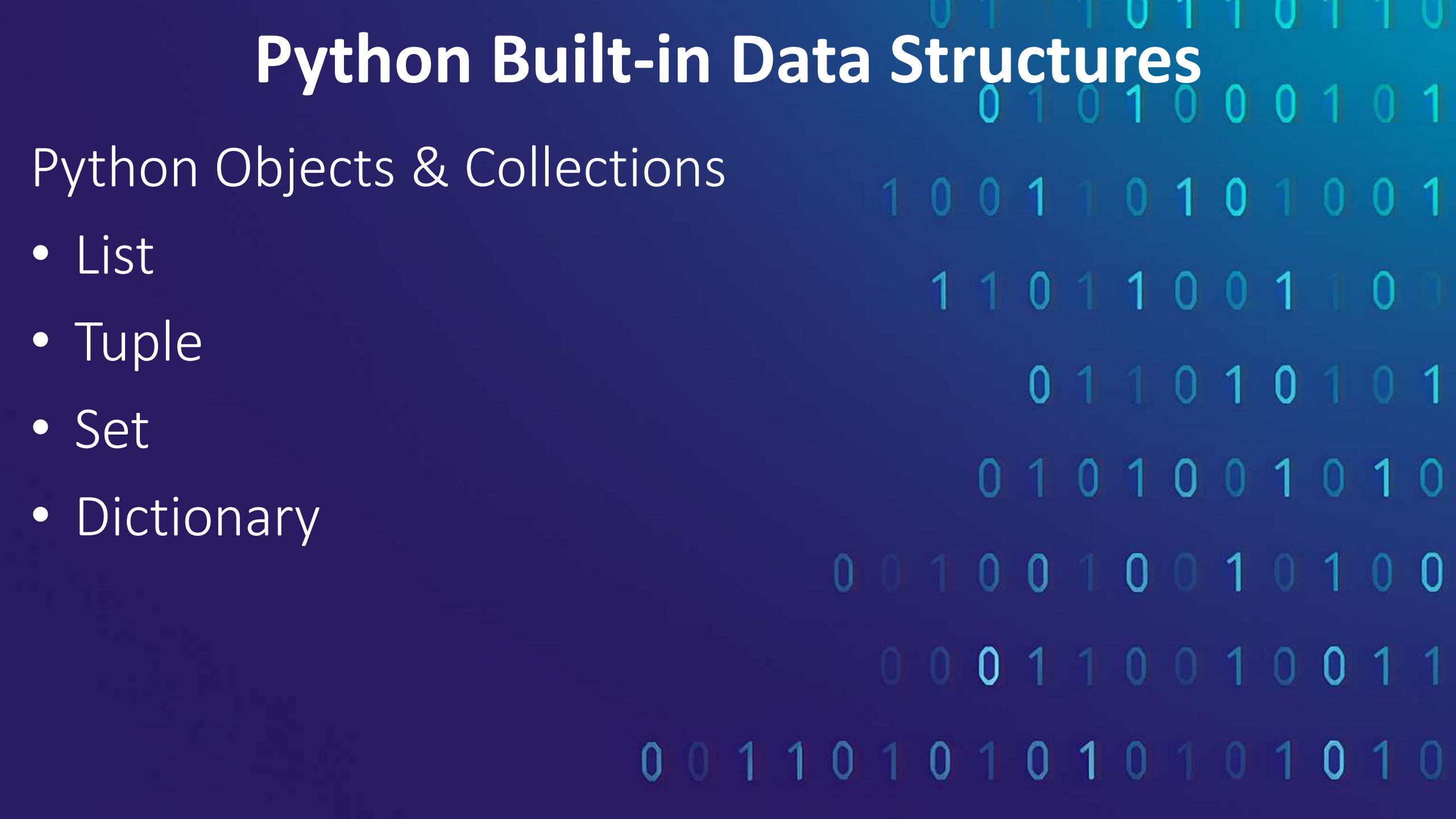
## IMPORTANT NOTE:

- Operator precedence follows standard mathematical rules (PEMDAS/BODMAS). Use parentheses () to control order explicitly.

# Python Built-in Data Structures

Python Objects & Collections

- List
- Tuple
- Set
- Dictionary



# Python Built-in Data Structures

## List

### An Ordered and Mutable Collection

- Lists are the most flexible data structure in Python
- They maintain the order of elements exactly as inserted
- Elements can be of any type — even mixed types in the same list

# Python Built-in Data Structures

## List

- Lists are mutable: you can add, remove, or modify elements after creation
- Ideal for sequences that need to grow or change during program execution
- Commonly used in data processing, machine learning pipelines, and result storage

# Python Built-in Data Structures

# Tuple

## An Ordered **but** Immutable Collection

- Tuples are similar to lists but cannot be changed after creation
- They preserve insertion order and allow duplicate elements
- Can contain mixed data types, including other containers

# Python Built-in Data Structures

# Tuple

- Immutability makes tuples perfect for fixed data records and as dictionary keys
- Faster than lists and provide integrity protection for constant data
- Widely used to represent records, coordinates, and return multiple values from functions

# Python Built-in Data Structures

## set

An Unordered Collection of Unique Elements

- Sets contain only unique items — duplicates are automatically removed
- No guaranteed order of elements (unordered collection)
- Highly optimized for membership testing and eliminating duplicates

# Python Built-in Data Structures

## set

- Support powerful mathematical set operations: union, intersection, difference, symmetric difference
- Perfect for removing duplicates from data, checking belonging, and performing logical operations
- Commonly used in data cleaning and exploratory data analysis

# Python Built-in Data Structures

# Dictionary

Mapping of Unique Keys to Values

- Dictionaries store data as key–value pairs
- Keys must be unique and immutable (strings, numbers, tuples)
- Values can be of any type and can be duplicated

# Python Built-in Data Structures

# Dictionary

- Extremely fast lookup by key
- Order of insertion is preserved (Python 3.7+)
- The standard way to represent structured and labeled data (e.g., JSON-like objects, database records, configuration settings)

# Python Built-in Data Structures

## Python Objects & Collections

| Structure | Syntax | Ordered? | Mutable? | Allows Duplicates? | Typical Use Case |
|---|---|---|---|---|---|
| **List** | [ ] | Yes | Yes | Yes | General-purpose sequences, dynamic arrays |
| **Tuple** | ( ) | Yes | No | Yes | Fixed data, heterogeneous records |
| **Set** | { } | No | Yes | No | Unique items, membership testing, set operations |
| **Dictionary** | {k: v} | Yes* | Yes | Keys: No | Mapping/lookup tables, structured data |

# Python Built-in Data Structures

## Introduction to len()

### Measuring the Size of Python Objects

- Returns the number of items or length of an object
- One of the most frequently used built-in functions in Python
- Behavior depends entirely on the type of object
- Essential for loops, conditionals, input validation, and debugging
- Works with all major built-in container types

# Python Built-in Data Structures

## len() with Sequence Types

Length of Ordered Sequences
(Strings ▪ Lists ▪ Tuples)

- String: counts individual characters (including **spaces** and **punctuation**)
- List: counts all elements, regardless of type or duplication

# Python Built-in Data Structures

## len() with Sequence Types

- Tuple: counts all contained elements (order is preserved)
- Duplicates are fully counted in all three sequence types

# Python Built-in Data Structures

## len() with Unordered & Mapping Types

Length of Sets and Dictionaries

### ❖ len() counts Unique Elements and Keys

- Set: counts only unique elements (automatic deduplication)
- Dictionary: counts the number of keys (values are ignored)
- Even if a value is a long list or complex object, it still counts as only one key

# Python Built-in Data Structures

## Introduction to len()

| Data Type | What len() Counts |
| --- | --- |
| String | Number of characters |
| List | Total number of elements |
| Tuple | Total number of elements |
| Set | Number of unique elements only |
| Dictionary | Number of keys (not values) |

## Conditional Execution

# The if Statement

## Making Decisions in Python

- Allows the program to execute certain blocks only when specific conditions are true
- Fundamental building block of program logic and control flow

## Conditional Execution

# The if Statement

- Supports elif (else-if) and else clauses for multiple branches
- Conditions can be combined using and, or, not logical operators
- Indentation defines the block of code that belongs to each condition

# Conditional Execution

# Loops

## Repeating Actions

- Automating Repetitive Tasks
- Loops let a program execute a block of code multiple times
- Two main types: for loops (iterate over sequences) and while loops (repeat while condition is true)

# Conditional Execution

## Loops

- Essential for processing datasets, training models, and numerical simulations
- Both types can be controlled with break, continue, and else clauses
- Proper loop design prevents infinite execution and improves performance

# Conditional Execution

# The For Loops

## Iteration over Sequences

- Designed to iterate over elements of any iterable (list, tuple, string, range, etc.)
- Automatically handles the iteration process – no manual index management needed
- Clean, readable syntax – preferred whenever the number of repetitions is known or finite
- Commonly used in data analysis, file processing, and machine learning pipelines

## Conditional Execution

# The While Loops

- Repeats as long as a given Boolean condition remains true
- Useful when the number of iterations is not known in advance
- Requires careful condition updates to avoid infinite loops
- Frequently used in optimization algorithms, user input loops, and real-time simulations

# Real-World Examples

## if Statement

- Grading system: determine letter grade from numerical score
- Model selection: choose best hyperparameter set based on validation accuracy
- Data cleaning: keep or discard samples according to missing value threshold

# Real-World Examples

## If Loops

- Calculating mean and standard deviation over a dataset of measurements
- Processing every image in a folder for preprocessing and augmentation
- Training a neural network for a fixed number of epochs

# Real-World Examples

## While Loops

- Gradient descent optimization until convergence criterion is met
- Reading sensor data continuously until user terminates the experiment
- Simulating physical systems until equilibrium or maximum time is reached

# REVIEW

| Feature | if Statement | for Loop | while Loop |
| --- | --- | --- | --- |
| When to use | Decision making | Known/fixed number of items | Unknown number, condition-based |
| Risk of infinity | None | Very low | High (must update condition) |
| Typical in ML | Hyperparameter choice | Training epochs, batch loops | Optimization until convergence |

# Matplotlib

- Python’s Standard Plotting Library **Subtitle:** Importing and Basic Setup
- Matplotlib is the most widely used plotting library in the Python scientific ecosystem
- Provides publication-quality 2D (and basic 3D) figures

# Matplotlib

- Highly customizable and integrates seamlessly with NumPy, Pandas, and Jupyter/Colab
- Standard convention: import the pyplot submodule as plt
- Essential for data visualization in research, reports, and academic publications

# Matplotlib

## Why Matplotlib Matters in Academic Work?

What is the Role of Matplotlib in Scientific Python?

From Exploration to Publication

- Enables exploratory data analysis through quick plots
- Creates reproducible, high-quality figures for papers and theses

# Matplotlib

- Supports a vast range of plot types: line, scatter, bar, histogram, contour, heatmap, 3D surfaces, etc.
- Full control over every visual element (colors, labels, fonts, legends, annotations)
- Output formats: PNG, PDF, SVG, EPS – perfect for LaTeX and journal submissions

# Standard Import Pattern

- Almost every notebook, script, and paper uses the same import line

```
import numpy as np
import matplotlib.pyplot as plt
```

- Short alias plt makes code more readable and concise
- Often combined with NumPy import (np) since arrays are the primary data source
- Once imported, plt functions are available throughout the session

# Matplotlib

## What Comes Next After Import

- After Importing Matplotlib we use Prepare data (usually NumPy arrays or Pandas DataFrames)
- Create figure and axes objects (explicit style recommended)
- Plot data using various plt or axes methods
- Customize appearance (titles, labels, grid, legend, limits)
- Display inline (notebooks) or save to file (scripts and papers)
- Matplotlib is the cornerstone of scientific visualization in Python – mastering it is essential for every researcher and data scientist.

# Functions

## Why Functions Exist?

- The Key to Clean, Reusable Code
- Eliminate repeated code
- Give meaningful names to operations
- Make programs easier to read, test, and debug
- Essential building block of all serious scientific and research code

# Functions

## Flexibility Features:

- Default values → optional arguments
- Keyword arguments → call in any order
- Variable number of arguments (*args, **kwargs) for advanced use
- Clear function signatures make code self-documenting

# Functions

Functions You Will Write Daily:

- Data loading and preprocessing
- Model training and evaluation steps
- Statistical calculations and metrics
- Plotting routines with consistent style
- Simulation steps and result analysis

# Functions

*Golden Rule of Programming:*
*If you copy-paste the same block more than twice*
*→ turn it into a function.*

*Presented by*

**Zahra Garoosy**

Content & *Slides prepared by*

**Mohammad Sadat Rasoul**

***QUST***

*fall of 2025*

انجمن علمی کوانتوم

دانشگاه علم و صنعت ایران

*Thank you for your attention!*