Python

SARAVANA

What is Python?

Python is a high-level, interpreted programming language known for its simplicity and readability. It's widely used in various fields, including data science, web development, machine learning, and automation.

History of Python

Created by Guido van Rossum in the late 1980s, Python was designed with a focus on code readability and efficiency. Its emphasis on simplicity and a large standard library contributed to its rapid growth and popularity.

Installing and Setting Up Python

Choosing a Python version:

Python has two major versions: Python 2 and Python 3. It's recommended to use Python 3 for new projects.

Downloading and installing:

Visit the official Python website (https://www.python.org/downloads/) and download the appropriate installer for your operating system (Windows, macOS, Linux).

Verifying the installation:

Open a terminal or command prompt and type `python --version` or `python3 --version`. You should see the installed Python version.

Installing Options

Installing **Anaconda**

Download: Visit the Anaconda website (https://www.anaconda.com/products/individual) and download the appropriate installer.

Google Colab

Cloud-based Jupyter Notebook: Welcome To Colab - Colab (google.com)

Installing **PyCharm**

Visit the PyCharm download page: https://www.jetbrains.com/pycharm/download/

Basic Syntax: Variables and Data Types

Variables:

A variable is a named container for storing data.

Use the assignment operator = to assign a value to a variable.

Variable names can contain letters, numbers, and underscores but cannot start with a number.

Data Types

Python is dynamically typed, meaning you don't need to declare the data type of a variable beforehand.

Common data types:

- **Integers:** Whole numbers (e.g., 42, -10)
- **Floats:** Numbers with decimal points (e.g., 3.14, 2.5)
- **Strings:** Textual data (e.g., "Hello, world!", 'Python')
- **Booleans:** True or False values

Operators

Operators are symbols used to perform operations on values or variables.

Arithmetic operators:

+ (addition)

- (subtraction)

* (multiplication)

/ (division)

% (modulo - remainder)

// (floor division)

** (exponentiation)

Comparison operators:

- == (equal to)
- != (not equal to)
- < (less than)
- > (greater than)
- <= (less than or equal to)
- >= (greater than or equal to)

Logical operators

`and` (both conditions must be true)

`or` (at least one condition must be true)

`not` (negates a condition)

Exercise

Create variables to store your name, age, and whether you like Python.

Calculate the area of a rectangle with given length and width.

Check if a number is even or odd.

Variables, Keywords Statements and Comments

Variables: As discussed yesterday, variables are named containers for storing data. Python is dynamically typed, so you don't need to declare the data type beforehand.

Keywords: These are reserved words in Python that have special meanings and cannot be used as variable names. Examples include `if`, `else`, `for`, `while`, `def`, `class`, `import`, etc.

Statements: Instructions given to the Python interpreter. Each line of code is generally a statement.

Comments: Explanatory notes within the code. They are ignored by the interpreter.

- Single-line comments: `# This is a comment`
- Multi-line comments: `""" This is a multi-line comment """`

Indentation

Python uses whitespace indentation to define code blocks.

Consistent indentation is crucial for code readability and execution.

Incorrect indentation leads to `IndentationError`.

Input and Output

Input: Used to get user input.

Output: Used to display results or messages.

String Manipulation

Strings are sequences of characters.

Basic operations:

Concatenation: Combining strings using the + operator.

Indexing: Accessing individual characters using square brackets.

Slicing: Extracting a substring using `[start:end]`.

Length: Finding the length of a string using `len()`.

Operators

Arithmetic operators: +, -, *, /, %, //, **

Comparison operators: ==, !=, <, >, <=, >=

Assignment operators: =, +=, -=, *=, /=

Logical operators: and, or, not

Bitwise operators: &, |, ^, ~, <<, >> (used for bit-level operations)

Membership operators: `in`, `not in` (used to check if a value is present in a sequence)

Identity operators: `is`, `is not` (compare object identities)

Data Structures

Lists: Ordered collections of items.

Can contain elements of different data types.

Created using square brackets `[]`.

Accessed using indexing and slicing.

Tuples: Ordered collections of items, similar to lists but immutable (cannot be changed after creation).

Created using parentheses `()`.

Data Structures

Dictionaries - **Unordered** collections of key-value pairs.

Keys must be unique and immutable (strings, numbers, or tuples).

Values can be of any data type.

Created using curly braces `{}`.

Accessed using keys.

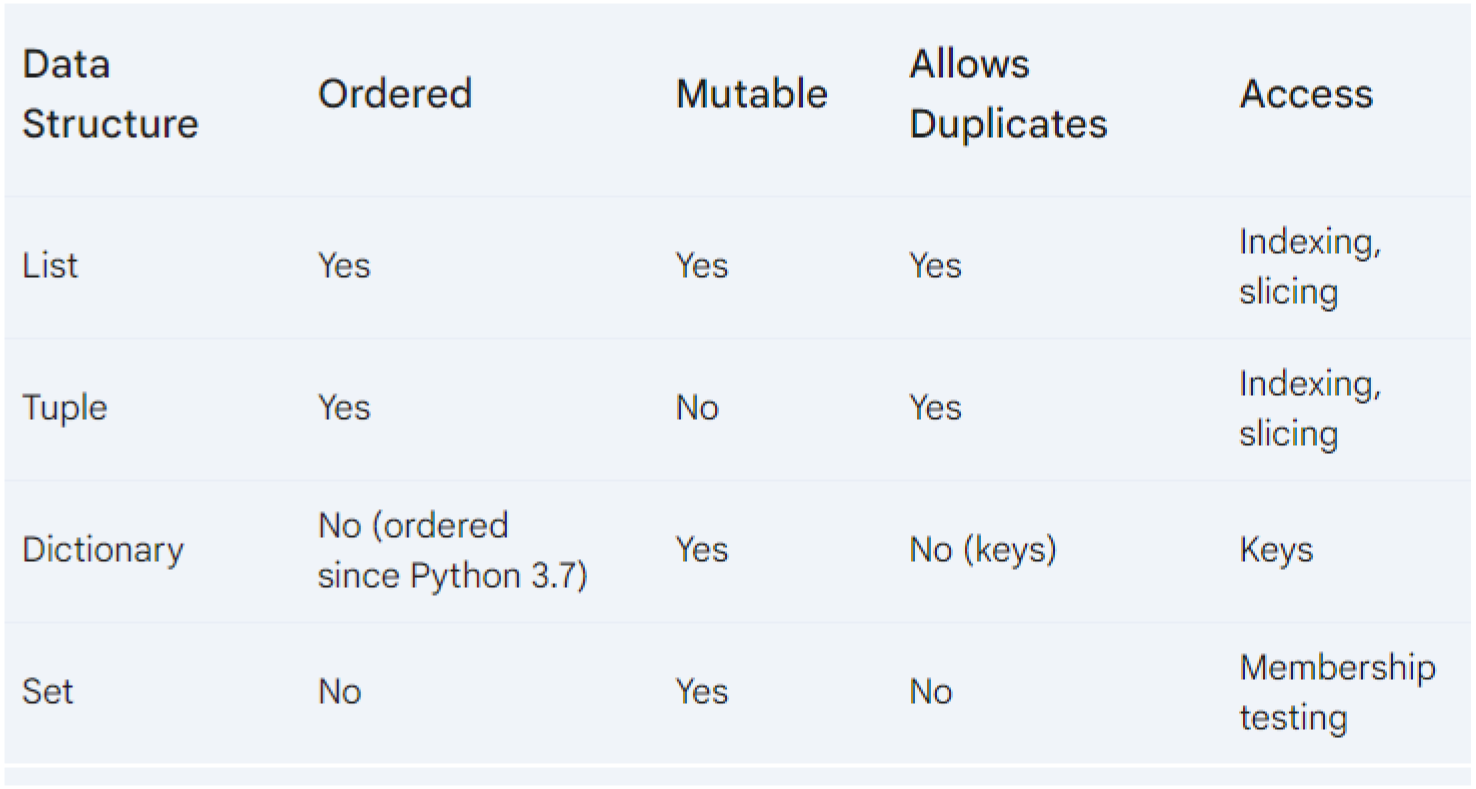
Sets - **Unordered** collections of unique elements.

No duplicate elements allowed.

Used for mathematical set operations (union, intersection, difference).

Created using curly braces `{}` or the `set()` constructor.

Key differences



Data Structure	Ordered	Mutable	Allows Duplicates	Access
List	Yes	Yes	Yes	Indexing, slicing
Tuple	Yes	No	Yes	Indexing, slicing
Dictionary	No (ordered since Python 3.7)	Yes	No (keys)	Keys
Set	No	Yes	No	Membership testing

Control Flow Statements

Control flow statements determine the order in which code is executed. They allow you to make decisions and repeat actions based on conditions.

If-Else Statements

Used to make decisions based on conditions.

Loops

Used to repeat a block of code multiple times.

For Loops

Iterate over a sequence (list, tuple, string, etc.).

While Loops

Execute a block of code as long as a condition is true.

Control Flow Statements

Nested Loops

Loops within loops - Used for tasks involving multiple iterations.

Break and Continue

Break: Terminates the loop entirely.

Continue: Skips the current iteration and moves to the next.

Exercise:

Write a program to check if a number is even or odd.

Create a program to print the multiplication table of a given number.

Find the factorial of a number using a loop.

Write a program to find the largest number in a list.

Understanding Functions

A function is a reusable block of code that performs a specific task. It helps to organize code, improve readability, and promote code reusability.

Defining Functions

Use the `def` keyword followed by the function name and parentheses.

Optionally, include parameters within the parentheses.

The function body is indented.

Use the `return` statement to specify the output.

Understanding Functions

Functions without parameters: Don't require any input values.

Functions with parameters: Accept input values.

Local and Global Variables

Local variables: Defined within a function and accessible only within that function.

Global variables: Defined outside of any function and accessible from anywhere in the program.

Lambda Functions

Anonymous functions defined using the `lambda` keyword.

Used for short, simple functions.

Understanding Functions

Calling Functions, Parameters, and Return Values

To call a function, use its name followed by parentheses.

Pass arguments (values) to the function through parameters.

A function can return a value using the `return` statement.

List Comprehension

A concise way to create lists based on existing lists or other iterable objects.

```
numbers = [1, 2, 3, 4, 5]
squares = [x**2 for x in numbers]
print(squares) # Output: [1, 4, 9, 16, 25]
```

Exercise

Write a function to calculate the factorial of a number.

Create a function to check if a number is prime.

Use list comprehension to create a list of even numbers from 1 to 20.

Write a lambda function to square a number.

Thanks