INTRODUCTION TO PYTHON

WHAT IS PYTHON?

Python is a high-level, interpreted programming language known for its simplicity and readability. It supports multiple programming paradigms including **procedural**, **object-oriented,** and **functional** programming.

WHY PYTHON?

Python's popularity stems from its easy syntax, vast **libraries**, strong **community**, **cross-platform compatibility**, and suitability for beginners and experts alike.

APPLICATIONS OF PYTHON

Used in **web** development, **data** analysis, **artificial** intelligence, **scientific** computing, **automation**, scripting, and more, making it a versatile tool across industries.

SALIENT FEATURES

Simple and **readable** syntax, **extensive** standard libraries, **dynamic** typing, **automatic** memory management, and strong **community** support.

INSTALLING PYTHOND

WINDOWS INSTALLATION

Download Python installer from the official website. Run the executable and ensure to check **'Add Python to PATH'** during installation to simplify command-line usage.

LINUX INSTALLATION

Use package managers like **apt** or **yum** to install Python. Example: **sudo apt-get install python3.** Verify installation with **python3 –version.**

MAC INSTALLATION

Install Python via Homebrew using **'brew install python'** or download the official installer. Verify installation with python3 –version.

SETTING ENVIRONMENT VARIABLES

Configure PATH environment variable to include Python directory. This enables running Python from any terminal or command prompt window.

PYTHON IDES

Popular IDEs include PyCharm, VS Code, and Jupyter Notebook. Select IDE based on feature needs like debugging, code completion, or data science support.

VARIABLES IN PYTHON

DOS AND DON'TS

Use meaningful **variable** names, avoid **reserved** keywords, and follow naming conventions (**snake_case**). Don't start names with **numbers** or use **special** characters.

COMMENTING

Use # for **single-line** comments and triple quotes ''' ''' or """ """ for **multi-line** comments to explain code clearly.

INDENTATION

Indent blocks consistently using **4 spaces**. Proper indentation defines code structure and avoids syntax errors.

PRINTING & INPUT

Use **print()** to display output. Use **input()** to take user input as strings, convert data types as needed.

DATATYPES

Common types: **int, float, str, bool, list, tuple, dict.** Choose datatypes based on the data's nature and operations needed.

HELPFUL FUNCTIONS

Use help() for documentation, dir() to list attributes/methods, and type() to check an object's datatype.

STRING MANIPULATIONS

COMMON STRING FUNCTIONS

Use len(), upper(), lower(), strip(), replace(), and split() for versatile string processing and formatting.

SLICING STRINGS

Extract substrings using syntax string[start:end:step], enabling precise control over which characters to select.

INDEXING & NEGATIVE INDEXING

Access characters by position: positive indexes start from 0; negative indexes start from -1 at the string end.

MATHEMATICAL FUNCTIONS

Apply functions like ord() and chr() to convert between characters and their Unicode integer representations.

PRACTICAL TIPS

Combine slicing and functions for efficient text manipulation; remember strings are immutable in Python.

PYTHON OPERATORS

COMMON PYTHON OPERATORS

- Logical Operators: and, or, not - used for boolean logic.
- Bitwise Operators: &, |, ^, ~, < <, > > - operate on bits of numbers.
- Assignment Operators: =, +=, -=, *=, /= - assign and update variable values.
- Arithmetic Operators: +, -, *, /, %, **, // - perform mathematical calculations.

ADDITIONAL OPERATORS

- Conditional Operator: Python uses if-else expressions instead of ?: ternary operator.
- Membership Operators: in, not in - check presence in sequences.
- Identity Operators: is, is not - compare object identities.
- Operators help control program flow and data manipulation efficiently.

DECISION MAKING IN PYTHON

Using IF Statement

Executes a block of code if a specified condition is true. Syntax: if condition: code_block

Using IF ELSE

Executes one block if condition is true, another block if false. Syntax: if condition: code_block else: alternative_block

Using IF ELIF ELSE

Allows multiple conditions to be checked sequentially. Syntax: if condition1: code elif condition2: code else: code

Using Ternary Operator

A compact syntax for simple conditional assignments. Format: value_if_true if condition else value_if_false

PROGRAM FLOW AND LOOPS

FOR LOOP

Iterates over a sequence (list, tuple, string) executing a block of code for each element. Useful for definite iteration.

RANGE() FUNCTION

Generates a sequence of numbers, commonly used with for loops to specify start, stop, and step values.

BREAK STATEMENT

Terminates the current loop immediately, exiting the loop even if conditions are not fully met.

CONTINUE STATEMENT

Skips the current iteration and proceeds to the next iteration of the loop without executing remaining code in the block.

PASS STATEMENT

A placeholder that does nothing; used when syntax requires a statement but no action is needed.

WHILE LOOP

Repeats a block of code as long as a specified condition remains true. Used for indefinite iteration.

DATA STRUCTURES IN PYTHON

Lists

Ordered, mutable collections. Support methods like append(), remove(), pop(), sort(). Useful for dynamic data storage.

Tuples & Packing/Unpacking

Immutable ordered collections. Packing groups multiple values; unpacking assigns them to variables efficiently.

Zip() Function

Combines multiple iterables element-wise into tuples, facilitating parallel iteration and data aggregation.

Dictionaries & Methods

Key-value pairs for fast lookups. Methods include **keys(), values(), items(), get(), update()** for flexible data access.

Sets & Methods

Unordered collections of unique elements. Support add(), remove(), union(), intersection() for set operations.

Comprehensions

Concise syntax for creating lists, dictionaries, and sets. Enhances readability and performance with inline loops and conditions.

FUNCTIONS AND ADVANCED FEATURES

Creating Functions

Define reusable blocks of code using the def keyword, enabling modular and organized programming.

Argument Passing Types

Supports positional, keyword, default, and variable-length arguments for flexible function calls.

Lambda Functions

Anonymous, concise functions defined with lambda keyword, useful for simple, short operations.

Map, Filter, Reduce

Functional programming tools to apply functions over iterables, filter elements, and reduce sequences to single values.

Generators & Iterators

Efficiently produce items one at a time using yield in generators; iterators allow traversal over data collections.

Decorators

Functions that modify behavior of other functions or methods, enhancing code reuse and readability.

CORE PYTHON MODULES

IMPORTING MODULES

Use **'import'** to add built-in or custom modules to your project for extended functionality.

COMMON MODULES

Key modules include datetime for dates, os/sys for system tasks, random for randomness, math for calculations, and string for text operations.

CREATING OWN MODULES

Define reusable functions and classes in separate files, then import them to organize and modularize code.

__NAME__ == '__MAIN__'

Use this condition to run code only when the module is executed directly, not when imported.

MODULE BENEFITS

Modules promote code reuse, simplify maintenance, and provide access to powerful pre-built tools.

REGULAR EXPRESSIONS IN PYTHON

CREATING REGEX PATTERNS

Use raw strings **(r'pattern')** to define regex patterns. Patterns describe text sequences to match, allowing flexible search criteria.

WILDCARDS AND META CHARACTERS

Wildcards like '.' match any single character. Meta characters such as '*', '+', '?', '^', '$' control repetition, position, and optionality.

COMMON REGEX METHODS

Key methods include **re.match()** for start matching,

re.search() for anywhere in text,

re.findall() for all matches, and

re.sub() for replacements.

USE CASES IN PYTHON

Regex is used in validation, **parsing logs**, **extracting data**, and **complex text processing** efficiently within Python programs.

LOGGING IN PYTHON

INTRODUCTION TO LOGGING

Logging records runtime events, errors, and informational messages to help track program execution and diagnose problems.

LEVELS OF LOGGING

Common levels include DEBUG (detailed info), INFO (general events), WARNING (potential issues), ERROR (failures), and CRITICAL (severe errors).

SAVING LOGS TO FILES

Logs can be saved to files for persistence and later analysis, using file handlers that manage log file creation and rotation.

LOG STRING FORMATTERS

Formatters customize how log messages appear, including timestamps, log level, source module, and the message content itself.

FILE HANDLERS

Handlers direct logs to different destinations like files, consoles, or remote servers, enabling flexible log management.

BASIC FILE MANAGEMENT

OPENING FILES

Use open() function with modes like 'r', 'w', 'a' to open files for reading, writing, or appending respectively.

READING FILES

Read entire file with read(), line-by-line with readline(), or all lines as a list with readlines().

WRITING FILES

Write text using write() or writelines() methods; overwrites or appends based on mode used during opening.

USING 'WITH' KEYWORD

Automatically manages file closing, ensures resource release even if errors occur, promoting cleaner code.

FILE FUNCTIONS

Common functions include seek() to move file pointer, tell() to get current position, and truncate() to resize files.

CSV FILE HANDLING

Use csv module to read/write CSVs easily with reader(), writer(), DictReader(), and DictWriter() for structured data.

DATABASE MODULE BASICS

INSTALLING DATABASE MODULE

Use pip to install modules like sqlite3 or other database connectors needed for your project.

INTRODUCTION TO SQLITE

SQLite is a lightweight, serverless database engine, ideal for embedded applications and rapid development.

CREATING A CONNECTION

Establish a connection to the SQLite database file using sqlite3.connect() method.

CREATING A CURSOR

Use connection.cursor() to create a cursor object for executing SQL queries.

EXECUTING QUERIES

Perform CRUD operations by executing SQL commands through the cursor.execute() method.

COMMIT AND LASTROWID

Use connection.commit() to save changes; lastrowid retrieves the ID of the last inserted row for reference.

OBJECT ORIENTED PROGRAMMING

CLASSES & OBJECTS

Classes define blueprints for objects. Objects are instances of classes encapsulating data and behavior.

SELF & __INIT__()

self represents the instance within class methods. **__init__()** initializes object attributes when created.

ATTRIBUTE METHODS

setattr(), hasattr(), delattr() manage object attributes dynamically, allowing setting, checking, and deleting properties.

@CLASSMETHOD & @STATICMETHOD

@classmethod accesses the class itself, useful for factory methods. @staticmethod defines utility functions not tied to class or instance.

INHERITANCE & POLYMORPHISM

Inheritance allows a class to derive properties and behavior from another. Polymorphism enables methods to work with different object types.

ADVANCED OOP FEATURES

Operator overloading customizes operator behavior for classes. Access modifiers control attribute visibility: public, protected (_), private (__).

ERROR AND EXCEPTION HANDLING

Core Exception Concepts

Exceptions are unexpected errors that occur during program execution. Handling exceptions prevents crashes by using **try-except blocks** to **catch** and manage errors.

Custom Exceptions

Create user-defined exceptions by subclassing Exception. This provides specific error types to improve error identification and control flow.

Importance of Messaging

Clear, informative error messages help users and developers understand the cause of problems, aiding in debugging and improving usability.

AGENDA

- Introduction to Python and History
- Installation Steps
- Variables and Data Types
- String Functions and Operators
- Decision Making and Program Flow
- Data Structures Overview
- Functions and Advanced Features
- Core Python Modules
- Regular Expressions
- Logging Basics
- File Management
- Database Module
- Object Oriented Programming
- Error and Exception Handling

RESOURCES

1. https://github.com/Asabeneh/30-Days-Of-Python/tree/master
2. https://www.python.org/
3. https://www.youtube.com/watch?v=QoIRX37VZpo
4. https://www.youtube.com/watch?v=bdUqQidffPE

THANK YOU