PYTHON LESSON 1

Agenda

- What, Why, and When of python?
- Python interpreter
- Python modules
- Variables, Values, and Types
- Objects
- Keywords
- Strings
- Indexing/Slicing

What is Python?

- A high-level, interpreted programming language.
- Dynamically-typed with an emphasis on code readability.
- Supports multiple programming paradigms: procedural, object-oriented, and functional programming.
- Has a comprehensive standard library that covers many areas of software development.
- Created by Guido van Rossum and first released in 1991.

Why use Python?

- **Readability**: Python's clean syntax makes code easy to read and write.
- **Versatility**: Can be used for web development, data analysis, artificial intelligence, scientific computing, automation, and more.
- **Extensive Libraries**: Python has a library for almost everything. Popular ones include Django (web development), Pandas (data analysis), TensorFlow (machine learning), and many more.
- **Community Support**: One of the largest programming communities. This ensures ample resources, solutions, and third-party packages. Github, Jira, Slack, Trello, Stripe, Salesforce, Dropbox, Zendesk, Mailchimp, Spotify, HubSpot, QuickBooks, Shopify, Square, etc.
- **Cross-Platform:** Python is portable and can run on various operating systems.
- **Integration:** Easily integrates with other languages like C, C++, Java, etc., and services like RESTful services.
- **Productivity:** Rapid development capabilities, which means less code and faster development.

When to use Python?

- Web Development: Frameworks like Django and Flask make building web applications easy.
- Data Analysis & Visualization: Tools like Pandas, NumPy, and Matplotlib make data manipulation and visualization straightforward.
- Machine Learning & AI: Libraries such as TensorFlow, Keras, and scikit-learn facilitate complex computations and algorithms.
- Automation: Writing scripts to automate repetitive tasks or batch process data.
- Scientific Computing: Ideal for researchers with SciPy, SymPy, etc.
- Education: It's often the first language taught in computer science courses due to its simplicity.
- Backend Development: Creating RESTful APIs and server-side applications.
- Internet of Things (IoT): Its simplicity and compatibility with hardware make it suitable for IoT solutions.

Python Interpreter

- Reads Python Code: When you run a Python script, the interpreter reads the code line by line.
- Compilation to Bytecode: Before execution, Python code is compiled to bytecode. This bytecode is a low-level, platform-independent representation of the source code.
- Python Virtual Machine (PVM): This bytecode is then interpreted and executed by the Python Virtual Machine. The PVM is the runtime engine of Python; it's an inner part of the interpreter.
- Interactive Mode: The Python interpreter can also be used interactively, which means you can type Python commands and see results immediately. This is done by just typing `python` (or `python3` depending on the installation) in the command line or terminal.

Lab - Python interpreter

Python modules

- file containing Python code, which can define functions, classes, and variables, and can also include runnable code.
- Purpose: Modules are used to organize and reuse code, enabling modular programming and code separation.
- File Extension: Typically, Python module files have a `.py` extension.
- Importing: Modules can be imported into other modules or into the main module using the `import` statement.
  - Example: `import math`
- Namespaces: Once imported, the functions, classes, and variables defined in that module can be accessed using the module's name as a prefix.
  - Example: `math.sqrt(4)`
- `from ... import ...` Statement: Allows specific functions, classes, or variables to be imported from a module, so you don’t need to use the module name as a prefix.
  - Example: `from math import sqrt`

Lab - dir() & help() function

Python variables

- reserved memory location to store values, acting as a reference to data.
- Naming Rules:
  - Must start with a letter (a-z, A-Z) or underscore (_).
  - Cannot start with a number (0-9).
  - Can contain letters, numbers, and underscores.
  - Variable names are case-sensitive (`age`, `Age`, and `AGE` are three different variables).
- Assignment: Variables are created through assignment using the equal sign (=).
  - Example: `x = 5`
- Dynamic Typing: Python is dynamically typed, meaning you don't explicitly declare a variable's type; it's determined at runtime.
  - Example: `x = 5` (Here, `x` is an integer)
- Reassignment: Variables can be reassigned to a new value or a value of a different type.
  - Example: `x = "hello"` (Now, `x` is a string)
- Multiple Assignment: Python allows multiple assignments in one line.
  - Example: `a, b, c = 5, 3.2, "Hello"`

Lab - variables

Data types

- Integers
- Floats (decimal numbers)
- Strings (text)
- Lists (ordered collections)
- Tuples (immutable ordered collections)
- Dictionaries (key-value pairs)
- Sets (unordered collections)
- Booleans (True or False)

Objects

- everything is an object. This means that every value in Python, be it a number, string, function, class, etc., is stored in memory as an object.
- Identity: Each object has a unique ID (address in memory), which can be retrieved using the `id()` function.
- Type: Each object has a type (like `int`, `str`, `list`, etc.), which determines the operations that the object supports. The type can be found using the `type(x)` function.
- Value: The content/data that the object represents.
- Immutable Objects: Objects whose values cannot be changed after creation. Examples include:
  - Integers
  - Strings
  - Tuples
- Mutable Objects: Objects whose values can be modified after creation. Examples include:
  - Lists
  - Dictionaries
  - Sets

Objects continued

- Reference Count: The number of references (or variables) pointing to the same object. When the reference count drops to zero, the object's memory is reclaimed by the garbage collector.
- Methods: Objects can have associated methods, which are functions tied to the object that can operate on the object's data. For example, strings have methods like `upper()` and `split()`.
- Attributes: Objects can also have attributes, which are variables that store data pertinent to the object. For instance, objects of a user-defined class might have attributes like `name` or `age`.

Keywords

- Definition: Keywords are reserved words in Python that have a predefined meaning and cannot be used as identifiers like variable names, function names, or class names.
- Purpose: They are used to define the structure and flow of a Python program.
- Immutable: Keywords cannot be altered or redefined.
- Case Sensitive: Keywords must be used exactly as they are. For example, `True` is a keyword, but `true` is not.

Keyword examples

- `if`, `else`, `elif`: Conditional statements.
- `while`, `for`: Loop constructs.
- `break`: Breaks out of a loop.
- `continue`: Skips the rest of the current loop iteration.
- `return`: Returns a value from a function.
- `import`, `from`: Used to include external modules.
- `def`: Defines a function.
- `class`: Defines a class.
- `try`, `except`, `finally`: Exception handling.
- `pass`: Null statement, a placeholder.
- `global`, `nonlocal`: Variable scope specifiers.
- `True`, `False`: Boolean literals.
- `None`: Represents the absence of a value.
- `and`, `or`, `not`: Logical operators.
- `in`, `is`: Membership and identity operators.
- `lambda`: Creates anonymous functions.
- `yield`: Used in generator functions.
- `raise`: Raises an exception.
- `with`: Simplifies exception handling by cleaning up resources.
- `async`, `await`: Used for asynchronous programming.

Strings

- Definition: A string in Python is a sequence of characters enclosed within single (`' '`), double (`" "`), or triple (`''' '''` or `""" """`) quotes.
- Immutable: Once created, the content of a string cannot be changed.
- Indexing: Characters in a string can be accessed using indexing.
  - Example: `str[0]` returns the first character.
- Slicing: Substrings can be obtained by slicing.
  - Example: `str[1:4]` returns characters from index 1 to 3.
- Escape Sequences: Special characters can be added using backslashes (`\`), like `\n` for a new line or `\\` for a backslash.
- Concatenation: Strings can be joined using the `+` operator.
  - Example: `'Hello' + ' World'` results in `'Hello World'`.
- Repetition: Strings can be repeated using the `*` operator.
  - Example: `'A' * 3` results in `'AAA'`.

Strings continued

- Length: The `len()` function returns the number of characters in a string.
- Built-in Methods: Strings come with many built-in methods for manipulation and inquiry, such as:
  - `upper()`: Converts the string to uppercase.
  - `lower()`: Converts the string to lowercase.
  - `split()`: Breaks the string into a list of substrings.
  - `replace()`: Replaces parts of the string.
  - `find()`: Searches for a substring and returns its starting index.
  - `strip()`: Removes whitespace from the start and end of the string.
- Multiline Strings: Triple quotes (`''' '''` or `""" """`) are used for strings that span multiple lines.
- Formatted Strings:
  - Using `format()`: `"Hello, {}!".format('Alice')` results in `'Hello, Alice!'`.
  - F-strings (Python 3.6+): `name = "Alice"; f"Hello, {name}!"` results in `'Hello, Alice!'`.

Lab - strings

Indexing/Slicing

- Basic Slicing:
  - Extract characters from index 2 to 4 (end index is exclusive): `str[2:5]`
- Without Start or End Index:
  - Get all characters from index 2 to the end: `str[2:]`
  - Get all characters from the beginning to index 4 (exclusive): `str[:4]`
  - Get all characters of the string: `str[:]`
- Negative Indexing:
  - Get the last character: `str[-1]`
  - Get the last three characters: `str[-3:]`
  - Get characters from third last to the last: `str[-3:]`
- Step/Stride in Slicing:
  - Get every second character from the string: `str[::2]`
  - Get characters from index 2 to 8, stepping by 2: `str[2:8:2]`

Indexing/Slicing continued

- Reversing a String:
  - Using slicing to reverse a string: `str[::-1]`
- Skipping Initial Characters:
  - Skip the first 3 characters and fetch the rest: `str[3:]`
- Fetching a Middle Section:
  - Extract characters from index 3 to 6 (exclusive of end index): `str[3:6]`
- Using Slicing with Data Structures:
  - For lists (works similarly to string slicing): `list[1:4]`

Lab - indexing/slicing

Strings

j	a	r	e	d
0	1	2	3	4