AIMERZ.ai
Aim.Act.Achieve

NOTES

Introduction to Python

Introduction to Python

Overview

1. What is Python?

Definition:Python is a high-level programming language known for its readability and straightforward syntax. It's interpreted, meaning that Python code is executed line-by-line, which simplifies debugging and development.

Example:

Python code to print "Hello, World!":

```python
print("Hello, World!")
```

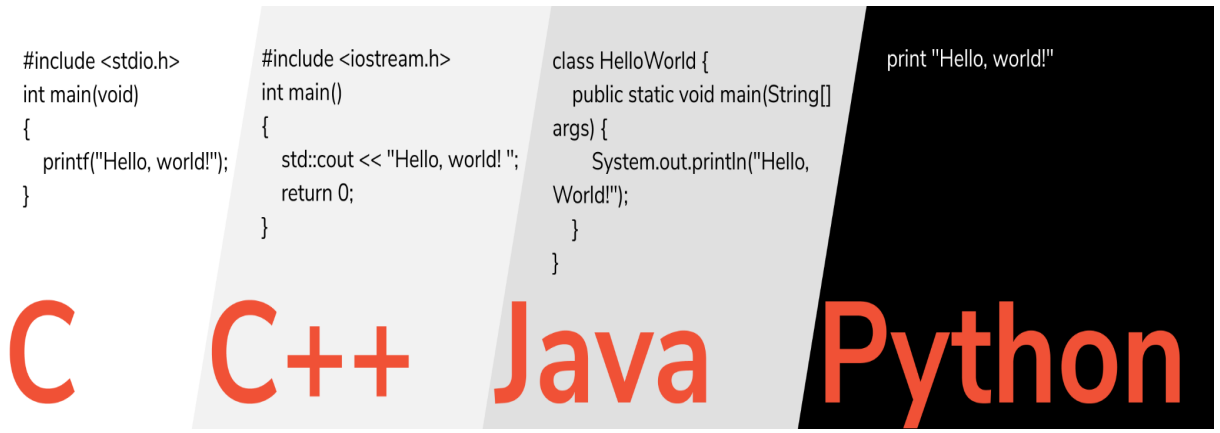
C code to print "Hello, World!":

```c
#include <stdio.h>

int main() {
   printf("Hello, World!");
   return 0;
}
```

Java code to print "Hello, World!":

```java
public class Main {
    public static void main(String[] args) {
        System.out.println("Hello, World!");
    }
}
```



History:

Python was created by Guido van Rossum and was first released in 1991. Its design emphasises code readability and simplicity, which has contributed to its widespread adoption. It's named after the British comedy group "Monty Python" and has since evolved into one of the most popular programming languages today.

2. Python Features

Easy to Learn and Use:

Python's syntax is clean and readable, making it accessible for beginners. It uses indentation to define code blocks, which encourages writing neat, organised code, and replaces the curly braces ({}) used in many other languages.

Example: Python's simple syntax for defining a function:

```python
def say_hello():
    print("Hello, World!")
```

In this example, there's no need for extra braces or semicolons, making it easy for non-tech learners to pick up.

Interpreted Language:

Python code is executed directly by the Python interpreter, without the need for compilation. This allows for immediate feedback and easier debugging.

Example: Use an IDE like PyCharm or an interpreter like Jupyter Notebook, where students can write code and instantly see results.

High-Level Language:

Python abstracts complex details of the computer's hardware. Users don't need to manage memory manually or deal with low-level operations.

Dynamic Typing:

In Python, you don't need to declare the type of a variable when you create it. The type is inferred at runtime, which simplifies coding but requires careful management to avoid type-related errors.

```python
x = 10       # x is an integer
x = "Hello"  # x is now a string
```

This allows more flexibility, but students should be aware of potential type-related errors as their programs grow larger.

Object-Oriented:

Python supports object-oriented programming (OOP), which means it can model real-world objects using classes. OOP principles such as inheritance, encapsulation, and polymorphism make it easier to model complex systems.

Example:

```python
class Person:
    def __init__(self, name):
        self.name = name  # Assigning the name attribute to the
person

    def say_hello(self):
        print(f"Hello, my name is {self.name}!")

# Creating an object of the Person class
person1 = Person("Alice")
person1.say_hello()  # Output: Hello, my name is Alice!
```

Extensive Libraries and Frameworks:

Python has a vast standard library and a rich ecosystem of third-party libraries and frameworks. This includes libraries for web development, data analysis, machine learning, and more.

Cross-Platform Compatibility:

Python can run on various operating systems, including Windows, macOS, and Linux. This makes it a versatile choice for development across different platforms.

Community Support:

Python has a large and active community that contributes to its development and offers extensive resources like documentation, tutorials, and forums.

Python Community

3. Applications of Python

Web Development:

Django: A high-level framework that encourages rapid development and clean, pragmatic design.

Flask: A lightweight Web Server Gateway Interface (WSGI) web application framework that's easy to use and extend.

Data Science and Machine Learning:

Pandas: Provides data structures and data analysis tools.

NumPy: Supports large, multi-dimensional arrays and matrices, along with a collection of mathematical functions.

Scikit-learn: Offers simple and efficient tools for data mining and machine learning.

TensorFlow and PyTorch: Popular libraries for deep learning and neural networks.

Automation/Scripting:

Python is often used for scripting to automate repetitive tasks such as file management, data extraction, and system administration.

Software Development:

Python can be used to build desktop applications with frameworks like Tkinter and PyQt, as well as mobile apps with tools like Kivy.

Scientific Computing:

SciPy: Builds on NumPy and provides additional functionality for scientific and technical computing.

Matplotlib: A plotting library used to create static, animated, and interactive visualisations in Python.

Education:

Python's simplicity makes it an ideal language for teaching programming concepts to beginners, as well as advanced topics to more experienced learners.

Game Development:

Pygame: A library designed for writing video games, providing functionalities like creating windows, handling events, and drawing graphics.

Finance: Python is used in finance for tasks like data analysis, financial modelling, and algorithmic trading, thanks to its powerful libraries and ease of integration with financial data sources.

4. Python Ecosystem

IDEs and Text Editors:

PyCharm: A full-featured IDE for Python with tools for debugging, testing, and project management.

VS Code: A lightweight editor with Python support through extensions.

Jupyter Notebook: Perfect for data science and interactive computing.

Package Management:

pip: The package installer for Python, allowing users to install and manage Python libraries and dependencies.

Example: To install Pandas using pip:

```Python
pip install pandas
```

Python Versions:

Python 2 vs. Python 3: Python 3 is the current version with ongoing support and updates, while Python 2 reached its end of life in January 2020. Python 3 offers improved features and fixes from Python 2, and it's recommended to use Python 3 for new projects. The differences include improved syntax (e.g., print statements), type hinting, and more advanced data structures.

Example: Syntax differences between Python 2 and Python 3:

```python
# Python 2
print "Hello, World!"

# Python 3
print("Hello, World!")
```

Comparison with Other Programming Languages

1. Language Type

Python: Interpreted language.

High-level, meaning it abstracts many complex operations, making it easy to learn and use.

C: Compiled language (code is converted into machine code before execution).

Low-level language, giving close control over the hardware and memory.

Java: Compiled and Interpreted (compiled into bytecode and then run on the Java Virtual Machine (JVM)).

High-level like Python, but with more structured syntax.

2. Syntax Simplicity

Python: Very simple and easy to read. Uses indentation to define code blocks instead of curly braces `{}` or semicolons `;`.

Example:

```python
if age > 18:
    print("You are an adult.")
```

C: Requires more structure. Uses curly braces `{}` and semicolons `;` to define code blocks and statements.

Example:

```c
if (age > 18) {
    printf("You are an adult.");
}
```

Java: More verbose and structured than Python but simpler than C. Also uses braces `{}` and semicolons `;`.

Example:

```java
if (age > 18) {
    System.out.println("You are an adult.");
}
```

3. Memory Management

Python: Automatic memory management through garbage collection. The programmer doesn't have to manually allocate or free memory.

C: The programmer controls memory allocation (`malloc()`, `free()`) and has more control over how memory is used known as Manual memory management

Java: Automatic garbage collection, like Python, but still offers more control over memory than Python.

4. Performance

Python: Slower compared to C and Java because it's interpreted and dynamically typed (types are checked at runtime).

C: Fastest of the three, as it's compiled directly into machine code and gives low-level access to memory and CPU.

Java: Faster than Python, but generally slower than C. Java uses Just-In-Time (JIT) compilation, where bytecode is compiled to machine code at runtime.

5. Type System

Python: Dynamically typed, meaning you don't have to declare the data types of variables. Types are checked at runtime.

Example:

```python
x = 10      # No need to declare the type.
x = "Hello" # Python allows you to change the type at any time.
```

C: Statically typed, meaning you must declare the data type of each variable at the start, and it cannot be changed later.

Example:

```c
int x = 10; // Variable type must be declared.
x = "Hello"; // This would cause an error.
```

Java: Also statically typed like C. Variables need explicit type declaration, but Java is more flexible with object-oriented data types.

Example:

```java
int x = 10; // Variable type is declared.
x = "Hello"; // Causes an error.
```

6. Use Cases

Python: Known for data science, web development, automation, and scripting.

Common libraries include Pandas, NumPy, Django, and Flask.

C: Typically used for system programming (operating systems), embedded systems, and applications where performance is critical (like gaming engines).

Java: Popular for enterprise-level applications, Android app development, and large systems where portability and scalability are crucial.

Common libraries include Spring, Hibernate, and Android SDK.

7. Object-Oriented Programming (OOP)

Python: Supports object-oriented programming but doesn't force it. You can write procedural code without using objects.

C: Does not support object-oriented programming directly. It is procedural, meaning it follows a sequence of commands.

Java: Fully object-oriented, meaning almost everything in Java is treated as an object. You have to define your code around classes and objects.

8. Portability

Python: Highly portable. Python code can run on any platform that has a Python interpreter.

C: Less portable than Python or Java because C code is often compiled specifically for the operating system and hardware it's intended to run on.

Java: Very portable thanks to the JVM (Java Virtual Machine). Once compiled to bytecode, Java can run on any system that has the JVM installed.

9. Learning Curve

Python: Easiest to learn due to its readable syntax and flexibility.

Excellent choice for beginners.

C: Steeper learning curve due to manual memory management and complex syntax.

Good for those who want to understand the fundamentals of how computers work.

Java: Moderate learning curve. Easier than C but more complex than Python.

Requires understanding of OOP concepts early on.

Feature	Python	C	Java
Syntax	Very simple and clean.	Low-level, more complex syntax.	Moderate syntax, more structured than Python.
Length	Minimal code, one line is enough.	Requires additional elements like `#include`, `main()`.	Needs a class and method declaration (`public static void main`).
Compilation/Interpretation	Interpreted language (executes line-by-line).	Compiled language (must be compiled before execution).	Compiled to bytecode, then run by the JVM (Java Virtual Machine).
Memory Management	Automatic garbage collection (via reference counting).	Manual memory management (pointers and memory allocation).	Automatic garbage collection through JVM.
Execution Speed	Slower, due to being interpreted.	Very fast, close to machine code.	Faster than Python, slower than C (but faster than interpreted languages).
Type System	Dynamically typed (no need to declare variable types).	Statically typed (requires explicit type declaration)	Statically typed, with type declarations.
Ease of Use	Easiest, great for beginners.	More complex and low-level, harder for beginners.	Moderate difficulty; easier than C, more complex than Python.
Platform Dependency	Cross-platform (requires Python interpreter).	Platform-dependent (compiled to machine code for specific OS).	Cross-platform (runs on any platform with JVM).
Use Cases	Web development, scripting, data science, automation.	System programming, embedded systems, high-performance apps.	Enterprise applications, Android development, web apps.
Error Handling	Runtime errors are often caught during execution.	Compile-time errors are common due to stricter syntax.	Both compile-time and runtime error handling (exceptions).
Community Support	Large community and a wealth of libraries.	Strong community, but fewer high-level libraries.	Strong support for enterprise solutions and libraries.

Keywords, Identifiers, and Comments

1. Keywords in Python

Keywords are reserved words in Python that have special meanings. You cannot use them as variable names, function names, or identifiers. Python has around 35 keywords, and they are all lowercase except for True, False, and None.

Purpose: Keywords are predefined in Python and perform specific functions.

Rule: You cannot use keywords for naming variables, functions, or identifiers.

Examples of Python Keywords:

- if, else, elif: Used for conditional statements.
- for, while: Used for loops.
- True, False: Represent Boolean values.
- import: Used to include external modules.

Example:

```python
if True:                # 'if' and 'True' are keywords
    print("This is a keyword example.")
```

Keywords form the backbone of any program's structure, making it essential to know them to write Python code.

2. Identifiers in Python

Identifiers are the names given to variables, functions, classes, etc. They are created by the programmer to refer to different elements in the code.

Purpose: Identifiers give names to entities like variables and functions.

Rule: Identifiers cannot be Python keywords.

Syntax Rules: Identifiers must start with a letter (A-Z, a-z) or an underscore (_), but can include numbers (0-9).

No special characters like @, #, $ are allowed.

Identifiers are case-sensitive (Name and name are different).

Examples:

```python
name = "John"          # 'name' is an identifier
_age = 25              # '_age' is an identifier starting with an underscore
height2 = 175          # 'height2' is a valid identifier
```

Common Mistake:

```python
2name = "Alice"        # Invalid: cannot start with a number
```

Identifiers are like names given to things in real life (like people, places), and follow simple rules for naming.

3. Comments in Python

Comments are notes added to the code to explain what's happening, making it easier to understand for yourself and others. Comments are ignored by the Python interpreter, so they don't affect the code execution.

Types of Comments:

Single-Line Comment:

Created using a hash (#) symbol.

Used for brief explanations.

Example:

```python
# This is a single-line comment
print("Hello, World!")  # This prints a greeting
```

Multi-Line Comment:

Python doesn't have a built-in multi-line comment syntax, but you can use triple quotes (' ' ' or " " ") to simulate multi-line comments.

Often used to describe functions or large sections of code.

Example:

```python
'''
This is a multi-line comment.
It can span multiple lines.
'''
print("Hello, Python!")
```

Purpose: Comments make your code readable by describing its functionality.

Rule: Comments do not affect the execution of the program; they are only for humans reading the code.

Comments help both the writer and others understand what the code does. In real-world data science projects, comments ensure collaboration is smoother.

Example for all:

```python
# This program checks if someone is an adult or not

age = 20  # 'age' is an identifier for storing the person's age

if age >= 18:     # 'if' is a keyword
```

```
    print("You are an adult.")   # Comment: Printing adult status
else:             # 'else' is a keyword
    print("You are not an adult.")
```

THANK YOU

AIMERZ.ai
Aim.Act.Achieve