# DEBRE BRHAN UNIVERSITY

# COLLEGE OF COMPUTUNG

# DEPARTMENT OF COMPUTER SCIENCE

# ARTIFICIAL INTELLIGENCE LAB REPORT

NAME : Nebiyu Samuel

ID: 0764/13

SUBMISSION DATE: MARCH 14/2024

SUBMITTED TO: MR. MKRE

**An introduction to Python and its importance in the world of programming and artificial intelligence projects.**

# **Python:**

Python is a high-level programming language known for its simplicity and readability. It supports multiple programming paradigms, including procedural, object-oriented, and functional programming. Here's a simple example of Python code:

**Python for AI**

Python has become an indispensable language in the field of programming and Artificial Intelligence (AI) due to

* A great library ecosystem.
* A low entry barrier.
* Flexibility.
* Platform independence.
* Readability.
* Good visualization options.
* Community support.
* Growing popularity.

**Example:** Calculating the factorial of a number

def factorial(n):

"""Calculate the factorial of a number."""

if n == 0:

return 1

else:

return n \* factorial(n-1)

# Test the function

number = 5

result = factorial(number)

print(f"The factorial of {number} is {result}.")

# **PYTHON VARIABLES AND IDENTIFIERS**

In Python, variables are used to store data values, while identifiers are the names given to these variables, functions, classes, and other objects. Here's a breakdown of variables and identifiers in Python, along with examples:

**Variables**:

* A variable is a reserved memory location to store values.
* You can think of variables as containers that hold data which can be changed during the program execution.
* Python has no command for declaring a variable. A variable is created the moment you first assign a value to it.
* Variable names in Python can contain letters (both uppercase and lowercase), digits, and underscores (\_) but cannot start with a digit.
* Python is case-sensitive, so **myVar**, **myvar**, and **MYVAR** are all different variables.

**Example:**

x = 5

y = "Hello, world!"

Identifiers

Identifiers are the names given to variables, functions, classes, modules, and other objects in Python. They serve as labels to identify and reference these entities in your code. Identifiers must follow certain rules:

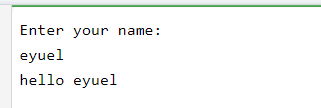
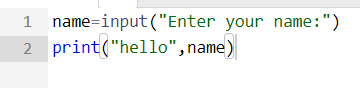
1. Identifiers can contain letters (both uppercase and lowercase), digits, and underscores (\_).
2. Identifiers must start with a letter or an underscore (\_). They cannot start with a digit.
3. Identifiers are case-sensitive, meaning myVar and myvar are two different identifiers.
4. Identifiers cannot be the same as Python keywords (e.g., `if`, `for`, `while`).It's recommended to use descriptive and meaningful names for identifiers to improve code readability.

# **Python Input and output functions**

In Python, input and output functions are used to interact with the user and display information to the user. Here's an overview of input and output functions in Python:

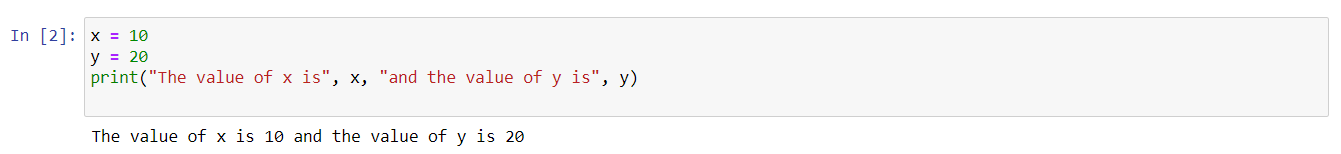
1. **Input Functions**:

* This function is used to take input from the user through the keyboard.
* It reads input as a string, so if you want to accept other types (like integers or floats), you need to convert the input using type casting.
* Example:

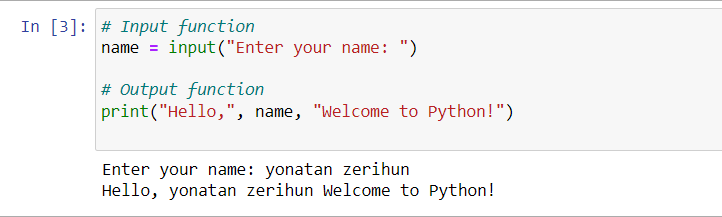


**Output Functions**:

* This function is used to display output to the console.
* It can take multiple arguments separated by commas and concatenates them with spaces in the output.
* Example:



Example combining input and output functions:



In this example, the **input()** function is used to take input from the user (their name), and the **print()** function is used to display a greeting message including the entered name.

Remember:

* The **input()** function always returns a string, so if you're expecting a different data type, you'll need to convert it.
* The **print()** function can take multiple arguments, separated by commas. It will automatically convert each argument to a string and concatenate them together. You can also specify the separator (**sep**) between these values and the ending character (**end**) at the end of the printed output.

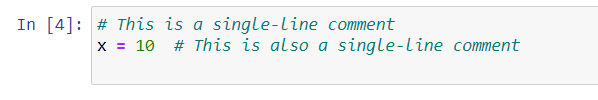
# **Python Comments**

In Python, comments are used to annotate code for better understanding or to temporarily disable certain lines of code. Python supports two types of comments: single-line comments and multi-line comments.

**Single-Line Comments:**

Single-line comments start with the **#** character and extend to the end of the line. They are commonly used for brief explanations or annotations.

Example:

 **Multi-line Comments**:

While Python doesn't have a built-in syntax for multi-line comments like some other programming languages, you can achieve multi-line comments using multi-line strings (also known as docstrings). Although these are technically strings, they are often used as comments because they are not assigned to a variable and thus have no effect on the program's execution.

Example:



# **PYTHON KEYWORDS**

# In Python, keywords are reserved words that have special meanings and purposes within the language. These keywords cannot be used as identifiers (such as variable names or function names). Here's a list of Python keywords:

False await else import pass

None break except in raise

True class finally is return

and continue for lambda try

as def from nonlocal while

assert del global not with

async elif if or yield

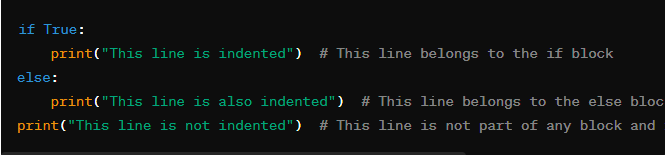
Here's a brief explanation of each keyword:

* **False, True, None**: These are Boolean values and represent false, true, and null values respectively.
* **and, or, not**: These are logical operators.
* **if, elif, else**: These are used for conditional branching.
* **for, while**: These are used for loop constructs.
* **break, continue**: These are used to alter the flow of loop execution.
* **pass**: This is a null operation (a placeholder) and is used when a statement is syntactically required but you don't want to execute any code.
* **def, return**: These are used in function definitions and to return values from functions.
* **class**: This is used to define a class.
* **try, except, finally**: These are used for exception handling.
* **raise**: This is used to raise exceptions.
* **import, from, as**: These are used for importing modules and parts of modules.
* **global**: This is used to declare that a variable inside a function is global.
* **nonlocal**: This is used to declare that a variable inside a nested function is not local.
* **lambda**: This is used to create anonymous functions.
* **yield**: This is used to return a value from a generator function.

# **PYTHON INDENTATION**

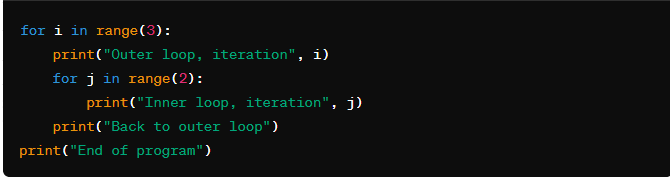
In Python, indentation plays a crucial role in defining the structure and scope of the code. Unlike many other programming languages where braces or other symbols are used to denote blocks of code, Python uses indentation to indicate the beginning and end of blocks. Here are some key points about Python indentation:

1. **Indentation Levels**:
   * Python code blocks such as those within functions, loops, conditionals, and classes are defined by indentation levels.
   * Blocks of code at the same indentation level are considered to be within the same block.
   * Indentation is typically done using four spaces for each level, although a tab character can also be used.
2. **Indentation Rules**:
   * The amount of indentation must be consistent throughout the code. Mixing tabs and spaces for indentation is not allowed and can lead to syntax errors.
   * A colon **:** typically indicates the beginning of a new block of code, and the following indented lines are considered part of that block.
3. **Example**:



1. **Nested Indentation**:
   * Indentation can be nested to represent nested blocks of code, such as nested loops or conditional statements.
2. **Indentation Errors**:
   * Incorrect indentation can lead to syntax errors or change the logic of the code unintentionally.
   * Python's syntax relies heavily on indentation, so proper indentation is essential for writing readable and maintainable code.
3. **Blank Lines**:
   * Blank lines and spaces within lines do not affect indentation. Only leading spaces or tabs determine the indentation level.
4. **Ending Indentation**:
   * The end of a block of code is indicated by a line with less indentation or a line with the same indentation level as the enclosing block.

Here's a demonstration of indentation in a loop:



Indentation is a fundamental part of Python's syntax and is used to improve code readability and maintainability. By enforcing consistent indentation, Python code becomes more structured and easier to understand, especially for larger codebases and collaborative projects.

Operators

Python supports a wide variety of operators which act like functions, i.e.

they do something and return a value:

Arithmetic: + - \* / % \*\*

Logical: and or not

Comparison: > < >= <= != ==

Assignment: =

Bitwise: & | ~ ^ >> <<

Identity: is is not

Membership: in not in

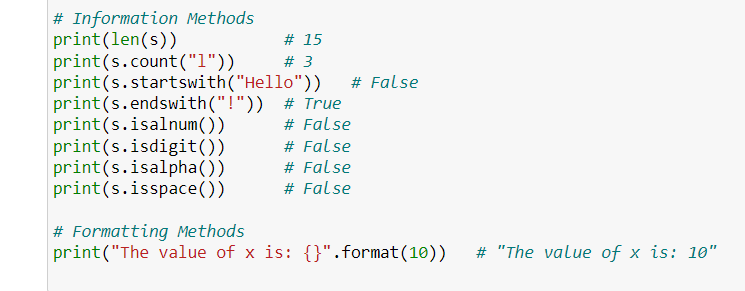
# **Python String Methods**

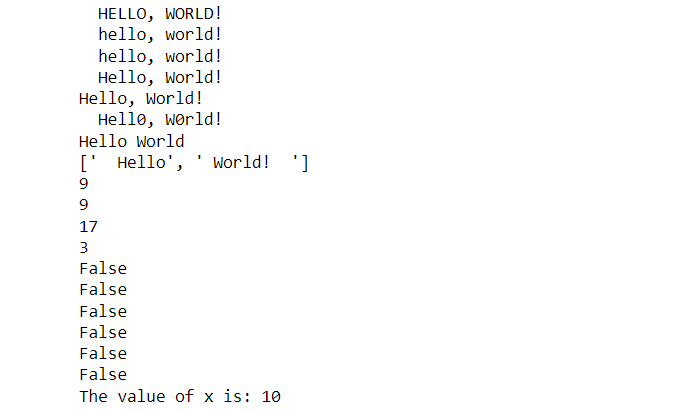
# In Python, strings are immutable sequences of characters, and Python provides a wide range of built-in methods to manipulate and work with strings efficiently. Here's an overview of some commonly used string methods:

# **conversion and Case Methods**:

* + **upper()**: Converts all characters in a string to uppercase.
  + **lower()**: Converts all characters in a string to lowercase.
  + **capitalize()**: Converts the first character of a string to uppercase and the rest to lowercase.
  + **title()**: Converts the first character of each word in a string to uppercase.

1. **Manipulation Methods**:
   * **strip()**: Removes leading and trailing whitespace characters from a string.
   * **lstrip()**: Removes leading whitespace characters from a string.
   * **rstrip()**: Removes trailing whitespace characters from a string.
   * **replace(old, new)**: Replaces occurrences of a substring with another substring.
   * **join(iterable)**: Concatenates elements of an iterable (such as a list) into a single string, with the string as the separator.
   * **split(sep=None, maxsplit=-1)**: Splits a string into a list of substrings based on a specified separator (default is whitespace). You can also specify the maximum number of splits to perform.
   * **find(sub[, start[, end]])**: Returns the lowest index of the substring **sub** in the string, or -1 if it's not found.
   * **index(sub[, start[, end]])**: Returns the lowest index of the substring **sub** in the string, or raises a **ValueError** if it's not found.
2. **Information Methods**:
   * **len()**: Returns the length of a string (number of characters).
   * **count(sub[, start[, end]])**: Returns the number of occurrences of a substring in a string.
   * **startswith(prefix[, start[, end]])**: Returns **True** if the string starts with the specified prefix; otherwise, returns **False**.
   * **endswith(suffix[, start[, end]])**: Returns **True** if the string ends with the specified suffix; otherwise, returns **False**.
   * **isalnum()**: Returns **True** if all characters in the string are alphanumeric (letters or numbers); otherwise, returns **False**.
   * **isdigit()**: Returns **True** if all characters in the string are digits; otherwise, returns **False**.
   * **isalpha()**: Returns **True** if all characters in the string are alphabetic (letters); otherwise, returns **False**.
   * **isspace()**: Returns **True** if all characters in the string are whitespace; otherwise, returns **False**.
3. **Formatting Methods**:
   * **format(\*args, \*\*kwargs)**: Formats a string by replacing placeholders with values.
   * **format\_map(mapping)**: Similar to **format()**, but takes a mapping (dictionary-like object) as its argument. Example usage of string methods 





These are just some of the many useful string methods available in Python. They provide powerful tools for manipulating and working with text data efficiently.

# **PYTHON DATA TYPES**

Python has several built-in data types that are classified into the following categories some of them are:

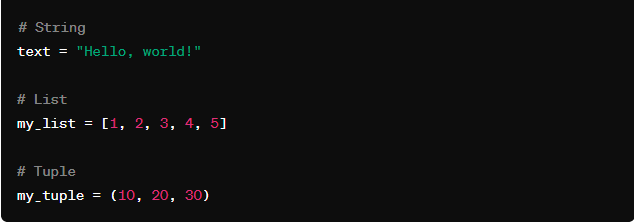
1 **Numeric Types**:

* **int**: Represents integer values.
* **float**: Represents floating-point numbers.
* **complex**: Represents complex numbers.



2: **Sequence Types:**

* str: Represents strings.
* list: Represents lists, which are ordered and mutable collections of items.
* tuple: Represents tuples, which are ordered and immutable collections of items.



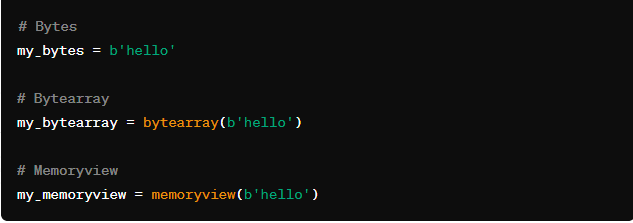
3: **Mapping Type:**

* **dict:** Represents dictionaries, which are unordered collections of key-value pairs.

****

4: **Binary Types**

* **bytes**: Represents sequences of bytes.
* **bytearray**: Represents mutable sequences of bytes.
* **memoryview**: Represents memory view objects.

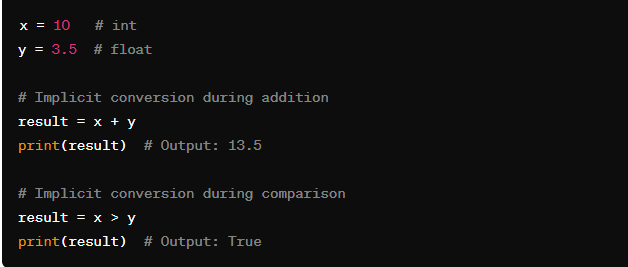
****

# **PYTHON TYPE CONVERSION**

Python provides several built-in functions and methods for type conversion, allowing us to convert data from one type to another

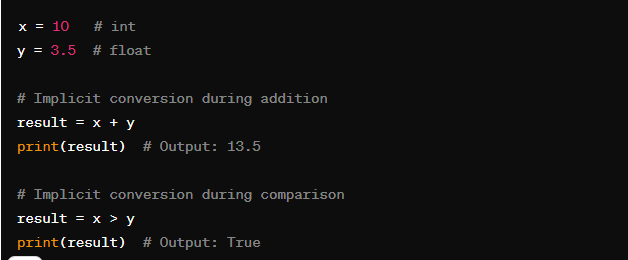
**Implicit Type Conversion**:

* Python automatically converts data types in some situations, such as during arithmetic operations or comparisons between different types.



**Explicit Type Conversion**:

* You can explicitly convert between data types using type conversion functions or constructors.

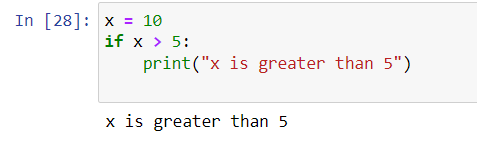


# **Python conditions**

1. In Python, conditions are used to control the flow of a program based on certain criteria. There are several ways to express conditions, primarily through the use of if, elif, and else statements. Let's explore different ways to use conditions in Python:

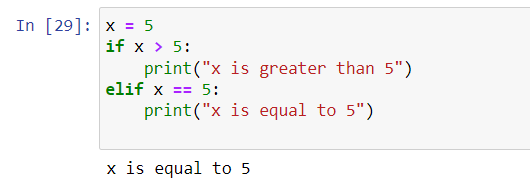
**Simple if statement**:

* Executes a block of code if a condition is true.



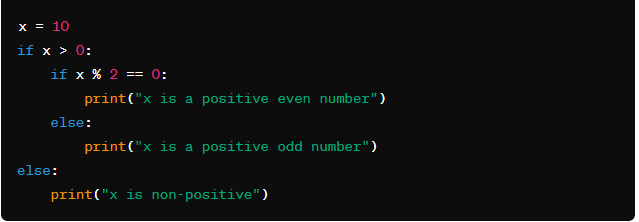
**if-else statement**:

* Executes one block of code if the condition is true and another block if the condition is false.



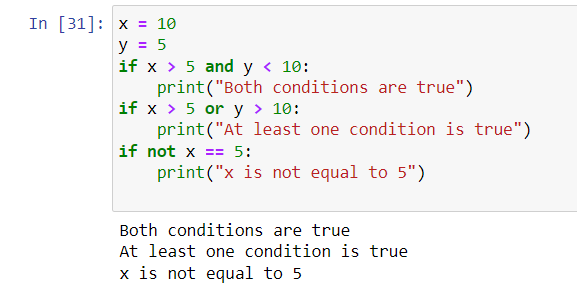
**Nested if statements**:

* **if** statements can be nested within each other to create more complex conditions.



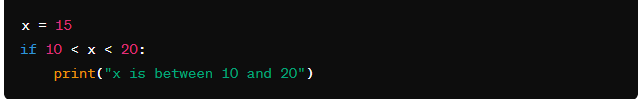
**3.Logical Operators**:

* **and**: Returns **True** if both conditions are true.
* **or**: Returns **True** if at least one of the conditions is true.
* **not**: Returns the opposite boolean value of the condition.



**4.Comparison Operators**:

* These operators are used to compare values.

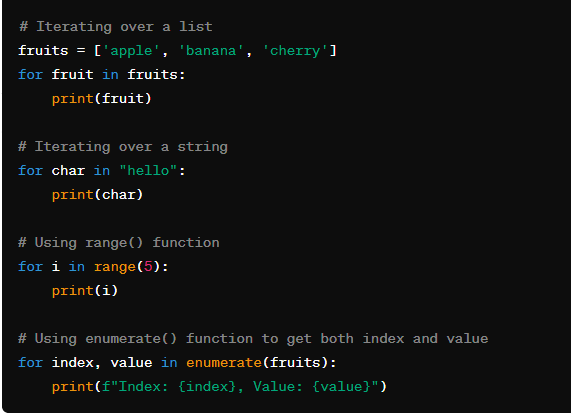


Conditions are fundamental for creating programs that can make decisions based on the state of the program or the data it is processing. They allow programs to react dynamically to different situations and inputs.

# **PYTHON LOOPS**

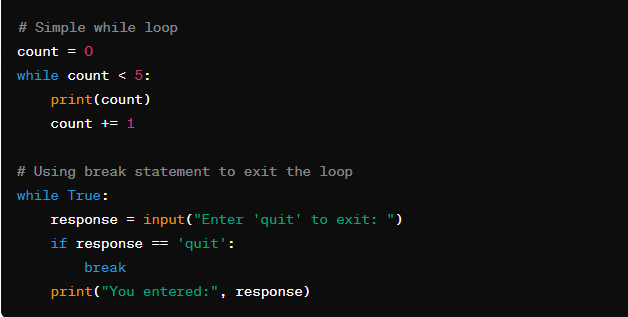
for Loop

* The for loop in Python is used to iterate over a sequence, such as a list, tuple, dictionary, set, or string.
* It works more like an iterator method found in other object-oriented programming languages, executing a set of statements for each item in the sequence.
* The range() function is commonly used with for loops to iterate a specified number of times, allowing customization of the starting value and increment



while Loop

* The while loop repeats a block of statements as long as a specified boolean condition is met.
* It is useful for situations where the number of iterations is not known beforehand.



Nested Loops

* Python supports nested loops, where one loop is inside another loop.
* Nested loops are useful for iterating over multiple sequences or creating complex patterns.

 # Outer loop

for i in range(1, 4):

    # Nested inner loop

    for j in range(1, 4):

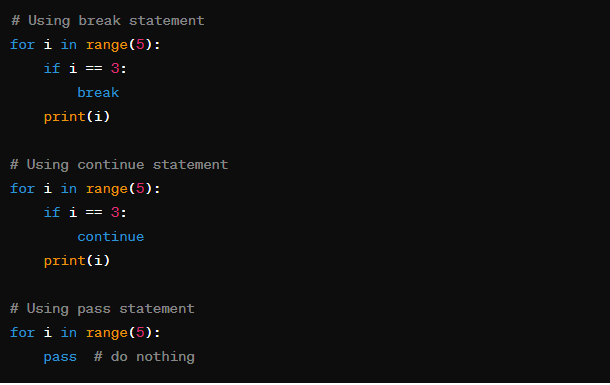
        print(f"Outer loop: {i}, Inner loop: {j}")

Loop Control Statements

Python provides break and continue statements to control loop execution flow.

* break is used to exit a loop, while continue skips the current iteration and moves to the next.

Example:

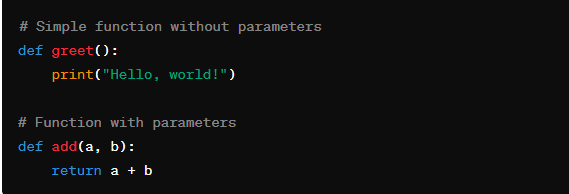


# **PYTHON FUNCTIONS**

Functions in Python are blocks of code that perform a specific task and can be reused throughout a program. Here are different ways to define and use functions in Python:

**Defining a Function:**

You can define a function using the def keyword followed by the function name and parentheses containing parameters. The body of the function is indented.

****

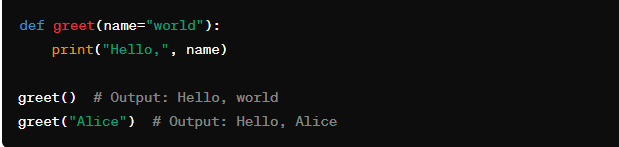
**Calling a Function**

To use a function, you call or invoke it by using the function name followed by parentheses with any required arguments



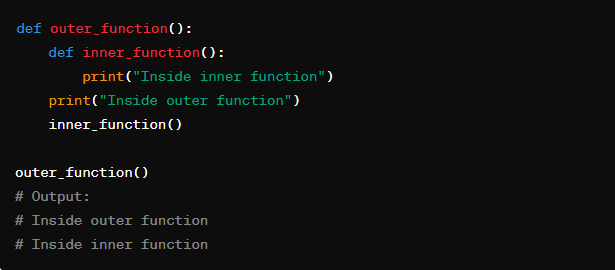
**Default Arguments:**

* You can provide default values for function parameters. If a parameter is not provided when the function is called, it will use the default value.



**Nested Functions:**

* Functions can be defined inside other functions, known as nested functions.



**Docstrings**:

* Docstrings are used to document functions, providing information about what the function does, its parameters, and return value.

**Default Parameters**:

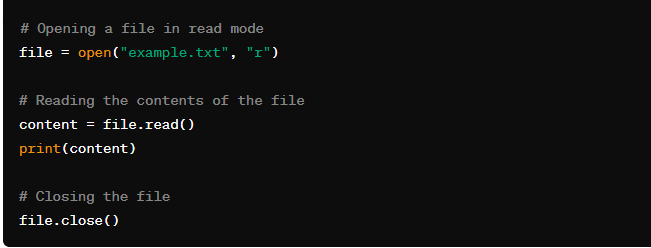
* You can specify default values for parameters in a function. If a value is not provided for a parameter when the function is called, the default value will be used.

**Variable-length Arguments**:

* You can use **\*args** and **\*\*kwargs** to handle variable-length arguments (arbitrary number of positional and keyword arguments) in a function.

# **PYTHON FILE HANDLING**

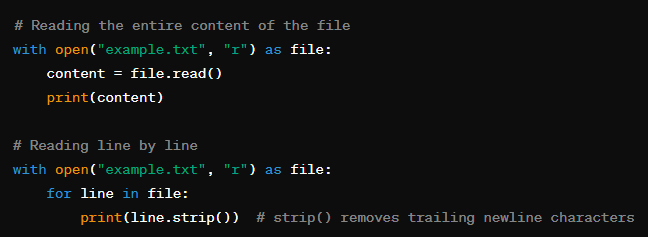
* File handling in Python allows you to work with files on the filesystem, reading from them, writing to them, and performing various other operations"r" - Read mode (default)
* "w" - Write mode (overwrites existing file)
* "a" - Append mode (adds to end of file)
* "x" - Create mode (creates new file, fails if exists)
* "b" - Binary mode (for non-text files like images)



Reading Files

Once a file is opened, you can read its contents using methods like:

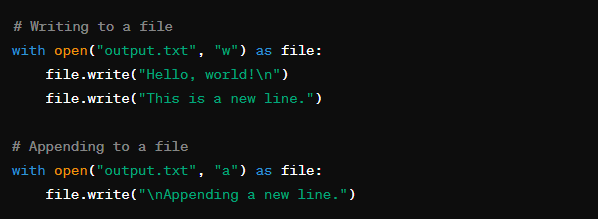
* read() - Reads the entire file content into a string
* readline() - Reads one line from the file
* readlines() - Reads all lines into a list of strings



Writing to Files

To write or append to a file, open it in "w" (write) or "a" (append) mode.Then use methods like:

* write(string) - Writes the string to the file
* writelines (list\_of\_strings) - Writes a list of strings to the file



Closing Files

It's important to close files after you're done using them to free up system resources.You can use the close() method or the with statement

 file = open("file.txt", "r")

# Do operations on file

file.close()

# Or use with statement

with open("file.txt", "r") as file:

    # File is automatically closed after this block

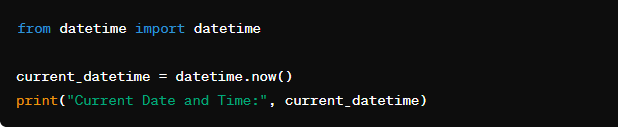
    pass

# **Python date and time**

In python, the 1datetime` module provides for working with dates and times. It allows you to create, manipulate, format, and perform arithmetic operations on dates and times easily. Additionally, the 1time1 module provides functions for working with time-related tasks, such as measuring time intervals and working with timestamps. Here’s an overview of the `datetime` module;

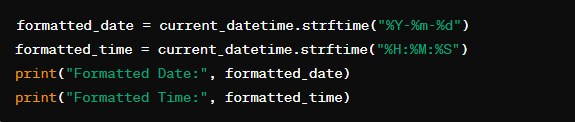
 Current Date and Time:

You can get the current date and time using the datetime.now() method.



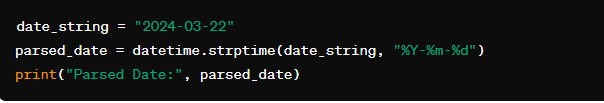
 Date Formatting:

You can format dates and times using the strftime() method with format codes.



 Parsing Dates from Strings:

You can parse strings into datetime objects using the strptime() method.



# PYTHON LIBRARIES

A python library is a collection of modules or packages that provides pre-written functionalities to perform specific tasks. These libraries as=re designed to simplify development by offering ready-to-use functions, classes, and constants that can be imported and used in python programs. Libraries in python serve various purposes, ranging from data manipulation and analysis to web development, machine learning, and scientific computing. Here are some common types of python libraries

* Standard library: the python standard library is a comprehensive collection of modules and packages that come with the python installation.
* Third-party libraries: third-party libraries are developed by individuals or organizations outside the python core development team. Third-party libraries are usually distributed using package managers like pip (python package installer) and can be installed easily using package management commands
* Numpy: is a fundamental library for numerical computing in python
* Pandas: pandas is powerful library for data manipulation and analysis in python
* Matplotlib: is a plotting library for creating static, interactive, and animated visualizations in python
* Requests: is a popular library for making HTTP requests in python
* Django and Flask: are web development frameworks for building web applications in pyhton