Python Programming

Karthikeyan Subramanian

Solution Architect,

Cognizant.

Chapter 1- Introduction to Python

Chapter 2- Python Installation & Setup + Visual Studio Code Installation

Chapter 3- Python Program

Chapter 4- Variables in Python

Chapter 5- Data Types in Python

Chapter 6- Type Casting in Python

Chapter 7- Input Function in Python

Chapter 8- Operators in Python

Chapter 9- Conditional Statements in Python

Chapter 10- Functions in Python

Chapter 11-Strings in Python (Part-1)

Chapter 12-Strings in Python (Part-2)

Chapter 13-Packages & Libraries in Python

Chapter 14-Pandas

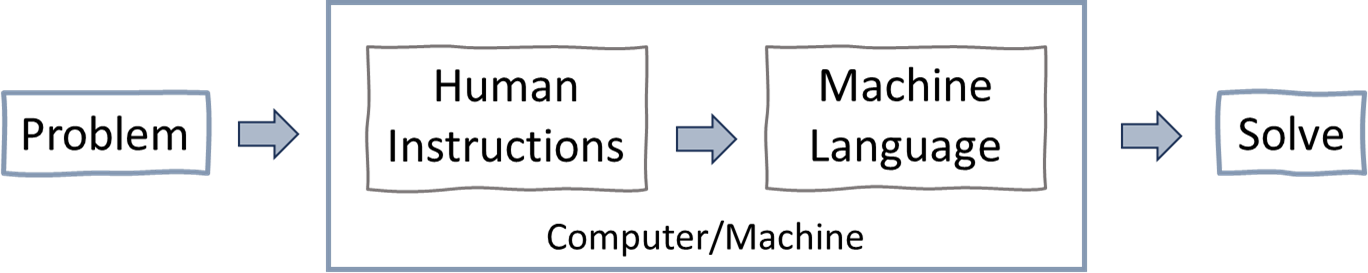
Chapter 15-Web Crawling using python

Chapter 16-You Tube with python

### Chapter - 01 Introduction to Python

* What is programming
* What is Python
* Popular programming languages
* Why Python
* Career with Python

### What is Programming Language?

**[](https://www.youtube.com/%40RishabhMishraOfficial/)**

Programming is the process of creating sets of instructions that tell a computer how to perform specific tasks. These instructions, known as code, are written in programming languages that computer understand and executes to carry out various operations, such as solving problems, analysing data, or controlling device.

Popular programming languages: Python, C, C++, Java, Go, C#, etc.

### What is Python?

Python is a high-level programming language known for its simplicity and readability.

Just like we use Hindi language to communicate and express ourselves,

Python is a language for computers to understand our instructions & perform tasks.

**Note:** Python was created by Guido van Rossum in 1991.

### Popular programming languages

As per survey, Python is the most popular programming language.

### Why Python?

Python is one of the easiest programming languages to learn and known for its versatility and user-friendly syntax, is a top choice among programmers.

Also, python is an open source (free) programming language and have extensive libraries to make programming easy. Python has massive use across different industries with excellent job opportunities.

### Python is Dynamically Typed Example

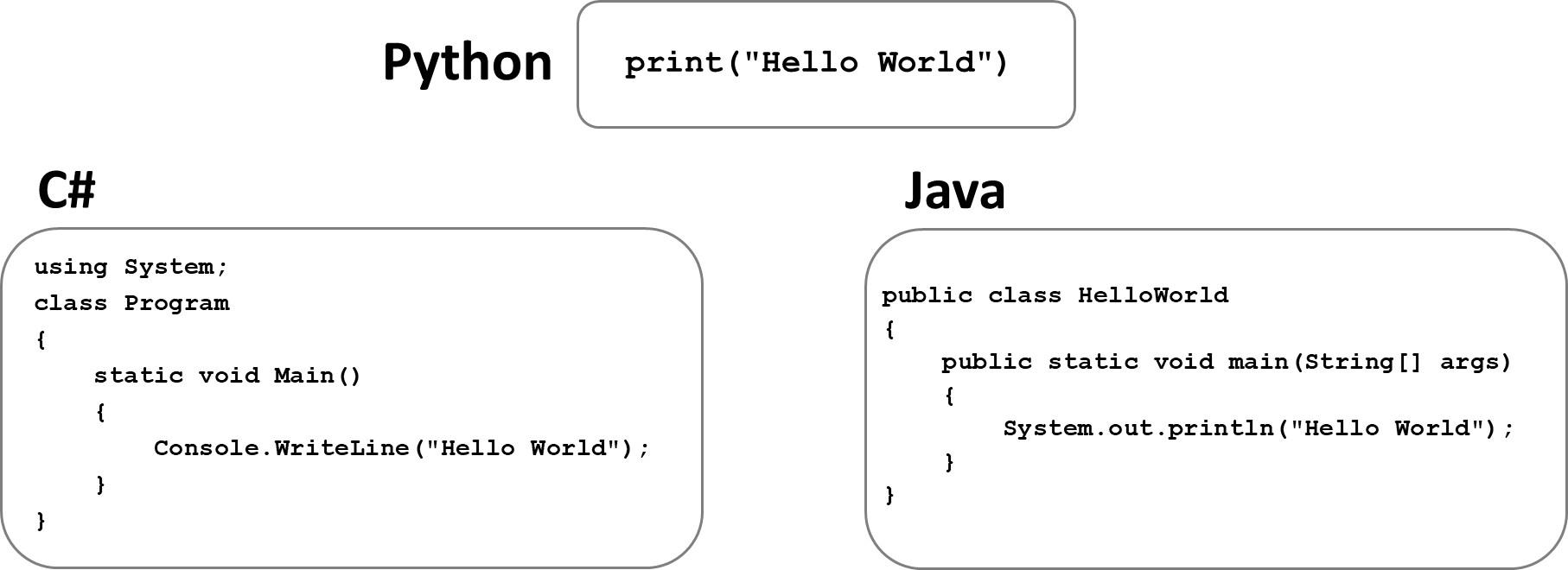
In Python there is no declaration of a variable, just an assignment statement. x = 8 *# here x is a integer*

x = "Python Programming" *# here x is a string*

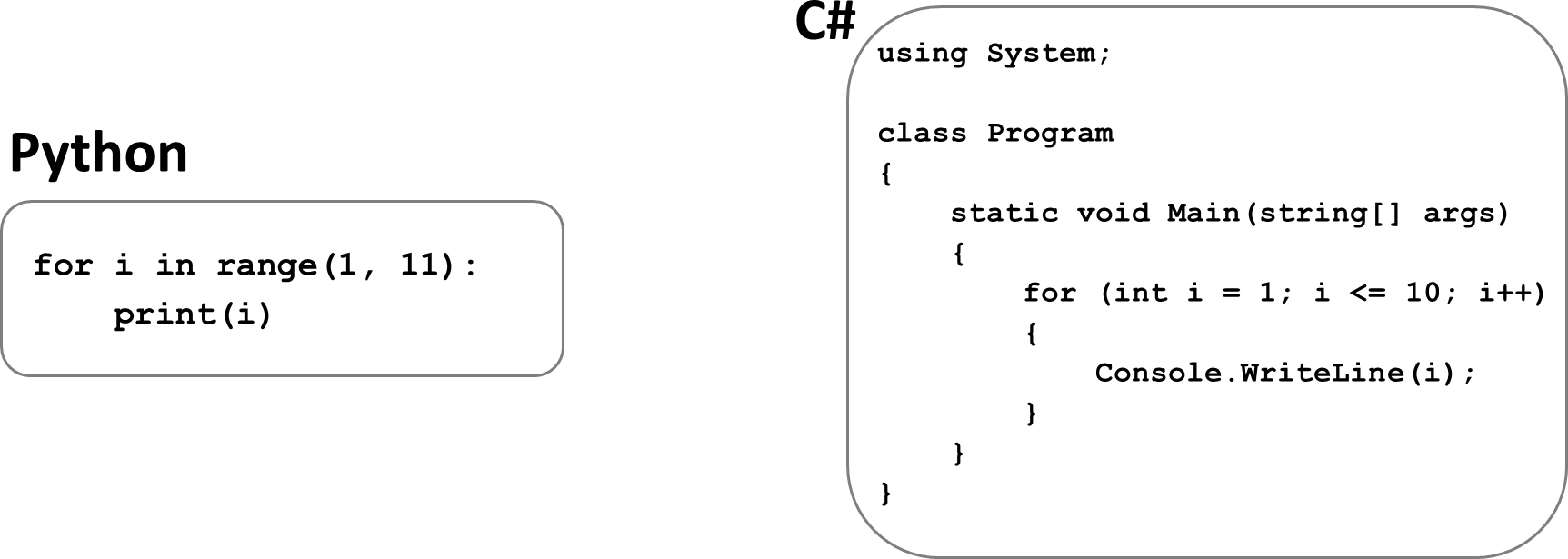
print(type(x)) *# you can check the type of x*

### Python - Easy to Read & Write

Ques1: Write a program to print “Hello World”

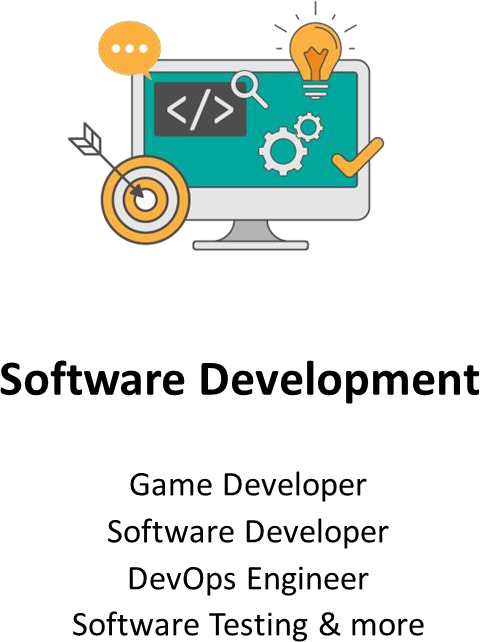
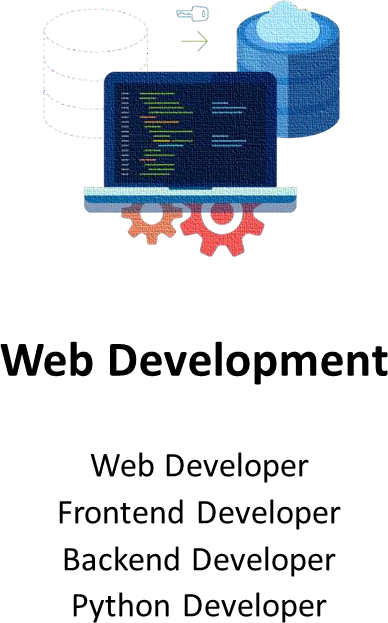
[](https://www.youtube.com/%40RishabhMishraOfficial/)

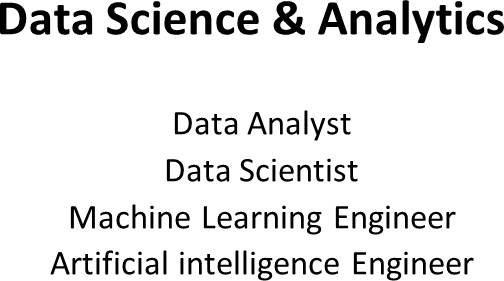
Ques2: Write a program to print numbers from 1 to 10

[](https://www.youtube.com/%40RishabhMishraOfficial/)

In above examples we can see that Python is simple in writing & reading the code.

### Careers with Python

Python is not only one of the most popular programming languages in the world, but it also offers great career opportunities. The demand for Python developers is growing every year.



# PYTHON TUTORIAL FOR BEGINNERS

**Chapter - 02**

### Python Installation & Setup + Visual Studio Code Installation

* Python installation
* Visual Studio Code installation

### Install Python

Step 1: Go to website: <https://www.python.org/downloads/> Step 2: Click on “Download Python” button

(Download the latest version for Windows or macOS or Linux or other) Step 3: Run Executable Installer

Step 4: Add Python to Path

Step 5: Verify Python Was Installed on Windows

Open the command prompt and run the following command:

### python --version

**Install Python IDE (code editor)**

Step 1: Go to website: <https://code.visualstudio.com/download> Step 2: Click on “Download” button

(Download the latest version for Windows or mac or Linux or other) Step 3: Run Executable Installer

Step 4: Click “Add to Path”

Step 5: Finish & launch the app

**Popular Python IDE**: VS Code, PyCharm, Jupyter Notebook & more

# PYTHON TUTORIAL FOR BEGINNERS



**Chapter - 03**

**First Python Program**

* Print- Hello World!
* Python As a Calculator
* Running the Python code
* Python Execution Steps
* Interpreter v/s Compiler

## First Python Program - Hello World

Printing "Hello World" as the first program in Python.

**print** is a keyword word that has special meaning for Python. It means, "Display what’s inside the parentheses."

**print("Hello World")**

**Instructor = "Karthikeyan"**

**print("flython by", Instructor, sep="-")**

## Python As a Calculator

Python can be used as a powerful calculator for performing a wide range of arithmetic operations.

**2+5 ** **add two numbers print(10/5) ** **divide two numbers**

 **print sum of two numbers**

**a = 2**

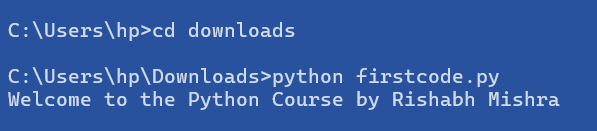
**b = 5 print(a+b)**

**Comments:** Comments are used to annotate codes, and they are not interpreted by Python. It starts with the hash character #

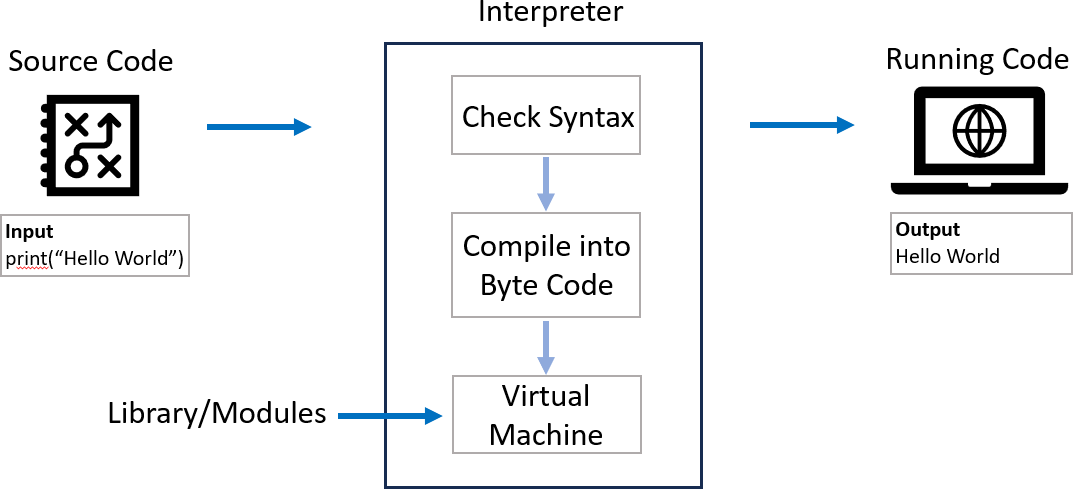
Comments are used as notes or short descriptions along with the code to increase its readability.

## Running the Python Code

* Create a new text file and inside it write – print(“Welcome to the Python Course”)
* Save file with extension .py – firstcode.py
* Open command prompt on windows (or Terminal on MacOS)
* Enter the location where firstcode.py file is saved – cd downloads
* Finally run the file as – python firstcode.py



## Python Execution Flow



**Python Code Execution Steps**

1. Lexical Analysis: The interpreter breaks down the code into smaller parts called tokens, identifying words, numbers, symbols, and punctuation.
2. Syntax Parsing: It checks the structure of the code to ensure it follows the rules of Python syntax. If there are any errors, like missing parentheses or incorrect indentation, it stops and shows a SyntaxError.
3. Bytecode Generation: Once the code is validated, the interpreter translates it into a simpler set of instructions called bytecode. This bytecode is easier for the computer to understand and execute.
4. Execution by PVM: The Python Virtual Machine (PVM) takes the bytecode and runs it step by step. It follows the instructions and performs calculations, assigns values to variables, and executes functions.
5. Error Handling and Output: If there are any errors during execution, like trying to divide by zero or accessing a variable that doesn't exist, the

interpreter raises an exception. If the code runs without errors, it displays any output, such as printed messages or returned values, to the user.

|  |  |
| --- | --- |
| **Interpreter** | **Compiler** |
| An interpreter translates and executes a source code line by line as the code runs. | A compiler translates the entire code into machine code before the program runs. |
| **Execution:** Line by line. | **Execution:** Entire program at once. |
| **Speed:** Slower execution because it translates each line on the fly. | **Speed:** Faster execution because it translates the entire program at once. |
| **Debugging:** Easier to debug as it stops at the first error encountered. | **Debugging:** Harder to debug  because errors are reported after the entire code is compiled |
| **Examples: Python**, Ruby, JavaScript, and PHP. | **Examples:** C, C++, Java, and Go. |

# PYTHON TUTORIAL FOR BEGINNERS



**Chapter - 04**

**Variables in Python**

* What is a Variable
* Variables - examples
* Variable Naming Rules

## Variables in Python

A variable in Python is a **symbolic name** that is a reference or pointer to an object.

In simple terms, variables are like **containers** that you can fill in with different types of data values. Once a variable is **assigned** a value, you can use that variable in place of the value.

We assign value to a variable using the **assignment operator** (=).

**Syntax: variable\_name = value Example: greeting = "Hello World"**

**print(greeting)**

## Variable Examples

Python can be used as a powerful calculator for performing a wide range of arithmetic operations.

|  |  |  |  |
| --- | --- | --- | --- |
| **flythonLevel** | **=** | **"Beginner"** | **pascal case** |
| **pythonLevel** | **=** | **"Beginner"** | **camel case** |
| **pythonlevel** | **=** | **"Beginner"** | **flat case** |
| **python\_level** | **=** | **"Beginner"** | **Snake case** |

**x = 10**

**print(x+1) ** **add number to a variable**

**a, b, c = 1, 2, 3**

**print(a, b, c) ** **assign multiple variables**

## Variable Naming Rules

1. Must start with a letter or an underscore ( **\_** ).
2. Can contain letters, numbers, and underscores.
3. Case-sensitive (my\_name and my\_Name are different).
4. Cannot be a reserved keyword (like for, if, while, etc.).

## \_my\_name = "Madhav" ⬛ for = 26 +

### ‘for’ is a reserved word in flython

# PYTHON TUTORIAL FOR BEGINNERS



**Chapter - 05**

**Data Types in Python**

* What are Data types
* Types of data types
* Data types examples

## Data Types in Python

In Python, a data type is a **classification** that specifies the **type of value** a variable can hold. We can check data type using type() function.

**Examples:**

1. **my\_name = "Madhav"**

**>>> type(my\_name) O/fl: <class 'str’>**

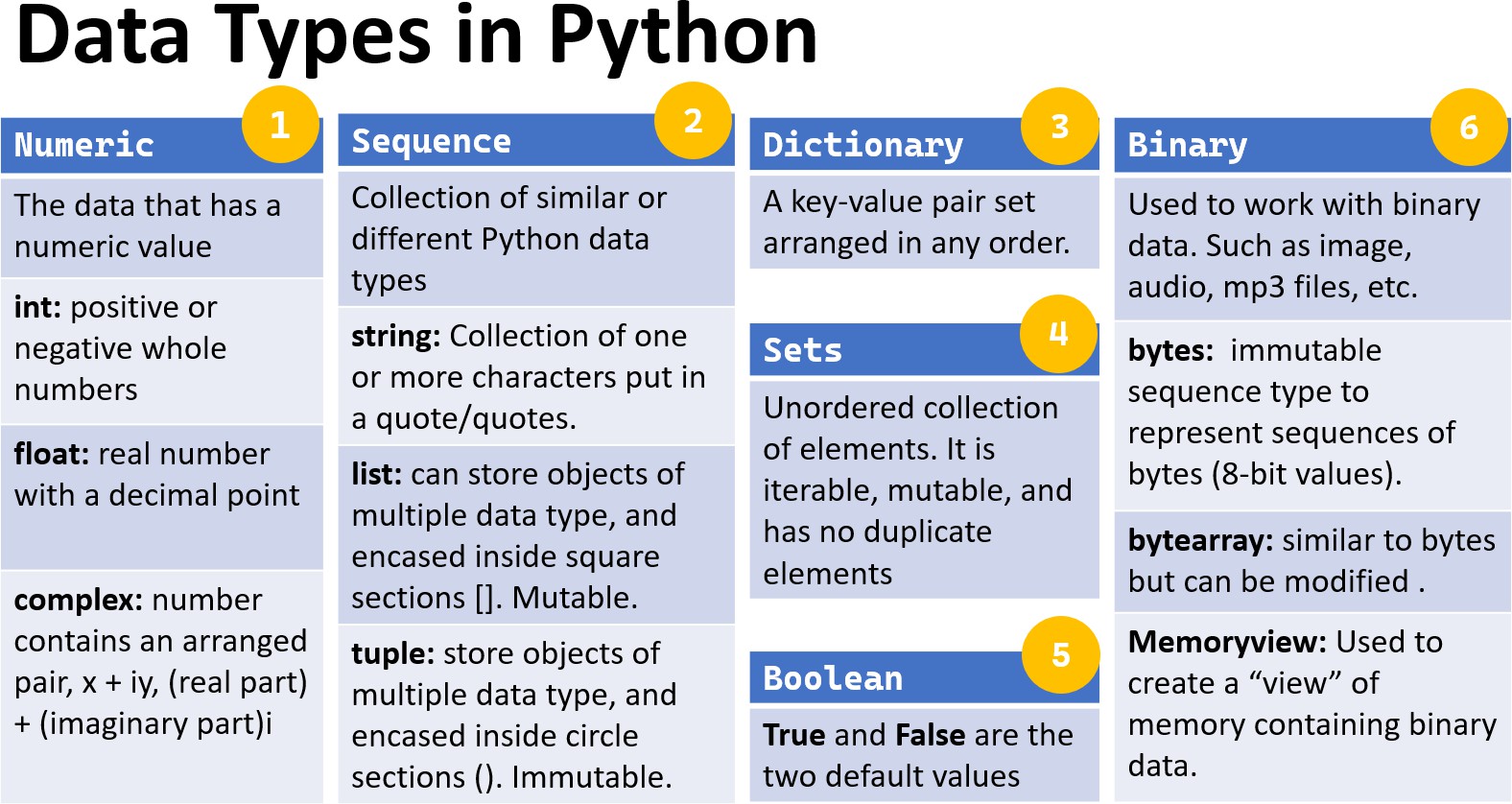
1. **value = 101**

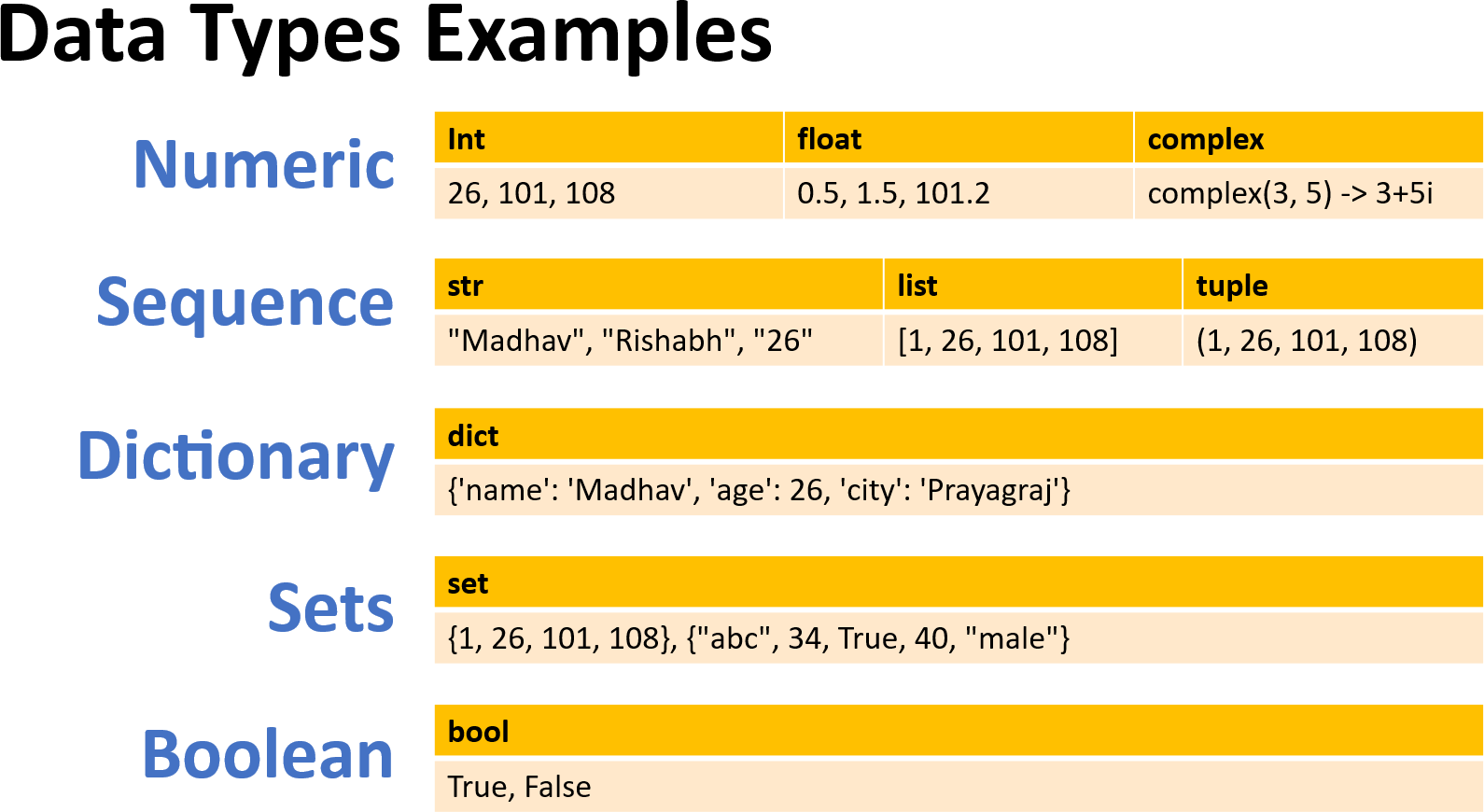
**>>> type(value) O/fl: <class 'int'>**

## Basic Data Types in Python

Python can be used as a powerful calculator for performing a wide range of arithmetic operations.

1. **umeric: Integer, Float, Complex**
2. **Sequence: String, List, Tuple**
3. **Dictionary**
4. **Set**
5. **Boolean**
6. **Binary: Bytes, Bytearray, Memoryview**



****

# PYTHON TUTORIAL FOR BEGINNERS



**Chapter - 06**

**Type Casting in Python**

* What is type casting
* Type Casting examples
* Type Casting - Types

## Type Casting

Type casting in Python refers to the process of **converting** a value from **one data type to another**. This can be useful in various situations, such as when you need to perform operations between different types or when you need to format data in a specific way. Also known as **data type conversion**.

Python has several built-in functions for type casting:

**int(): Converts a value to an integer.**

**float(): Converts a value to a floating-point number. str(): Converts a value to a string.**

**list(), tuple(), set(), dict() and bool()**

## Type Casting Examples

Basic examples of type casting in python:

 Converting String to Integer: str\_num = "26"

int\_num = int(str\_num) print(int\_num)  Output: 26

print(type(int\_num))  Output: <class 'int'>

 Converting Float to Integer:

float\_num = 108.56 int\_num = int(float\_num)

print(int\_num)  Output: 108 print(type(int\_num))  Output: <class 'int'>

## Types of Typecasting

There are two types of type casting in python:

* Implicit type casting
* Explicit type casting

## Implicit Type Casting

Also known as coercion, is performed automatically by the Python interpreter. This usually occurs when performing operations between different data types, and Python implicitly converts one data type to another to avoid data loss or errors.

 Implicit type casting from integer to float num\_int = 10

num\_float = 5.5

result = num\_int + num\_float  Integer is automatically converted to float

print(result)  Output: 15.5 print(type(result))  Output: <class 'float'>

## Explicit Type Casting

Also known as **type conversion**, is performed **manually** by the programmer using built-in functions. This is done to ensure the desired type conversion and to avoid unexpected behavior.

 Converting String to Integer: str\_num = "26"

int\_num = int(str\_num) print(int\_num)  Output: 26

print(type(int\_num))  Output: <class 'int’>

# PYTHON TUTORIAL FOR BEGINNERS



**Chapter - 07**

**Input Function in Python**

* Input Function – Definition
* Input Function – Example
* Handling Different Data Types

## Input Function in Python

The input function is an essential feature in Python that allows to take **input from the user**. This is particularly useful when you want to create interactive programs where the user can provide data during execution.

Also known as **user input function**. **How input Function Works:**

The input function waits for the user to type something and then press Enter. It reads the input as a string and returns it.

**Example:**

 flrompting the user for their name name = input("Enter your name: ")

 Displaying the user's input print("Hello, " + name + "!")

## Input Function – Add 2 Numbers

A simple program that takes two numbers as input from the user and prints their sum.

 flrompting the user for the first and second number num1 = input("Enter the first number: ")

num2 = input("Enter the second number: ")

 Since input() returns a string, we need to convert it to an integer

num1 = int(num1) num2 = int(num2)

 Calculating the sum and display the result

sum = num1 + num2

print("The sum of", num1, "and", num2, "is:", sum)

## Multiple Input from User & Handling different Data Types

 input from user to add two number and print result x = input("Enter first number: ")

y = input("Enter second number: ")

 casting input numbers to int, to perform sum print(f"Sum of {x} fi {y} is {int(x) + int(y)}")

## Home Work – User input and print result

Write a program to input student name and marks of 3 subjects. Print name and percentage in output.

 flrompting the user for their name and 3 subject marks name = input("Enter your name: ") hindi\_marks = input("Enter Hindi Marks: ") maths\_marks = input("Enter Maths Marks: ")

science\_marks = input("Enter Science Marks: ")

 Calculating percentage for 3 subjects

percentage = ((int(hindi\_marks) + int(maths\_marks) + int(science\_marks))/300)\*100

 flrinting the final results

print(f"{name}, have {percentage}%. Well done fi keep working hard!!")



**Chapter - 08**

**Operators in Python**

* What are Operators
* Types of Operators
* Operators Examples

## Operators in Python

Operators in Python are **special symbols or keywords** used to perform operations on operands (variables and values).

**Operators**: These are the special symbols/keywords. Eg: + , \* , /, etc.

**Operand**: It is the value on which the operator is applied.

 **Examples**

|  |  |  |
| --- | --- | --- |
| **Addition operator '+':** | **a** | **+ b** |
| **Equal operator '==':** | **a** | **== b** |
| **and operator 'and':** | **a** | **> 10 and b < 20** |

## Types of Operators

Python supports various types of operators, which can be broadly categorized as:

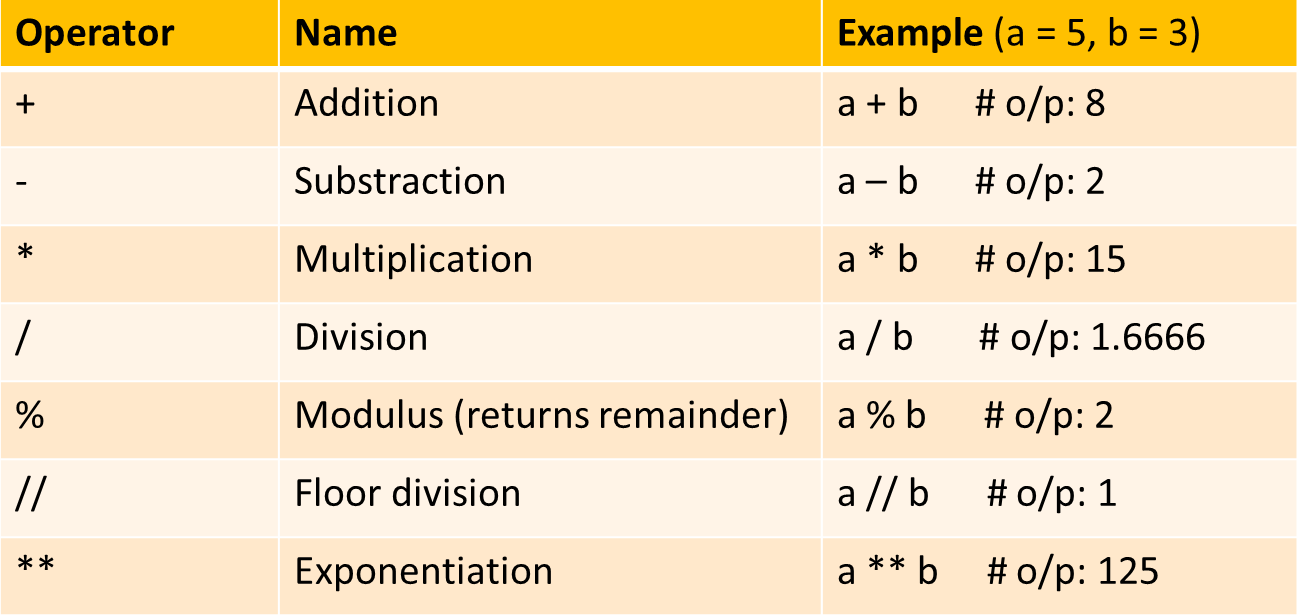
1. Arithmetic Operators
2. Comparison (Relational) Operators
3. Assignment Operators
4. Logical Operators
5. Identity Operators

## Operators Cheat Sheet

|  |  |
| --- | --- |
| **Operator** | **Description** |
| () | Parentheses |
| \*\* | Exponentiation |
| +, -, ~ | Positive, Negative, Bitwise NOT |
| \*, /, //, % | Multiplication, Division, Floor Division, Modulus |
| +, - | Addition, Subtraction |
| ==, !=, >, >=, <, <= | Comparison operators |
| is, is not, in, not in | Identity, **Membership** Operators |
| NOT, AND, OR | Logical NOT, Logical AND, Logical OR |

1. **Arithmetic Operators**

Arithmetic operators are used with numeric values to perform mathematical operations such as addition, subtraction, multiplication, and division.



Precedence of **Arithmetic Operators** in Python:

P – Parentheses

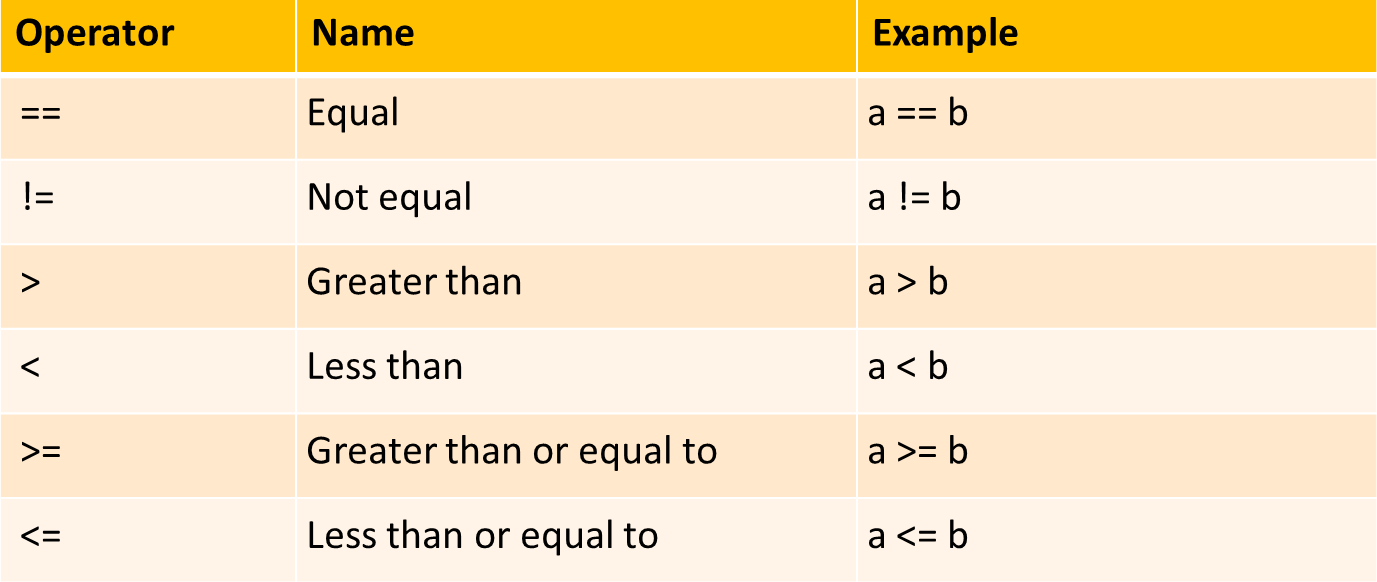
E – Exponentiation M – Multiplication D – Division

A – Addition

S – Subtraction

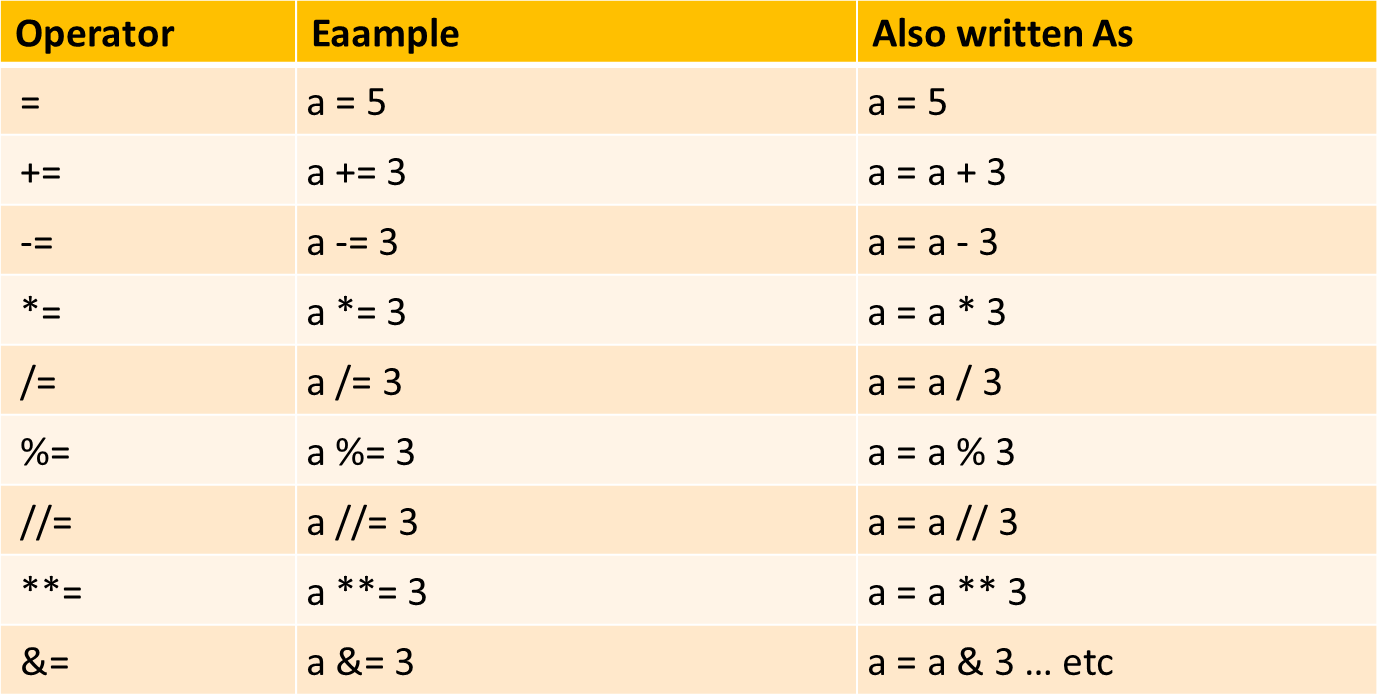
## Comparison (Relational) Operators

Comparison operators are used to compare two values and return a Boolean result (True or False).



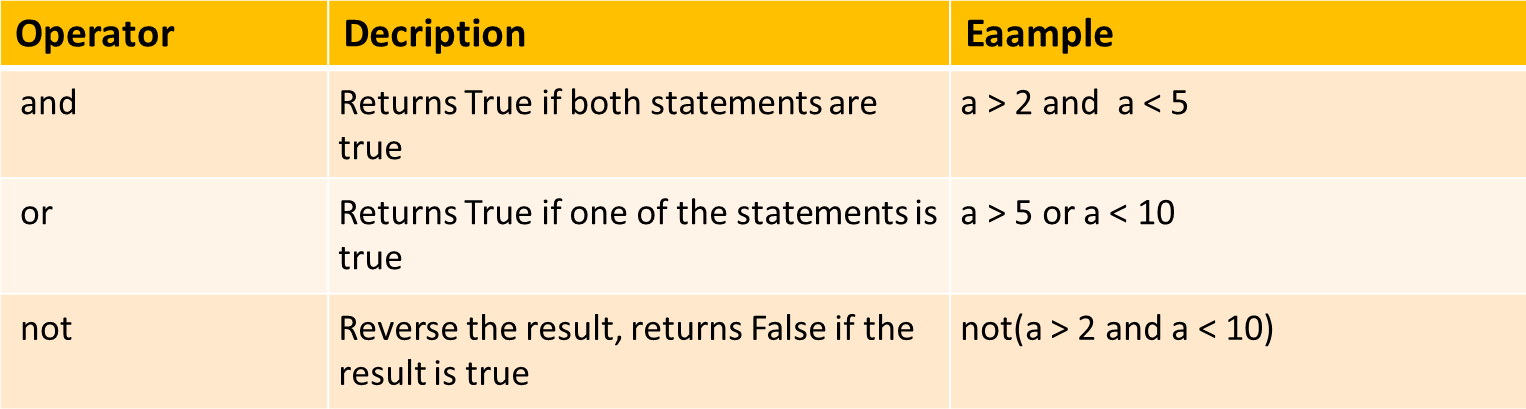
## Assignment Operators

Assignment operators are used to assign values to variables.



## Logical Operators

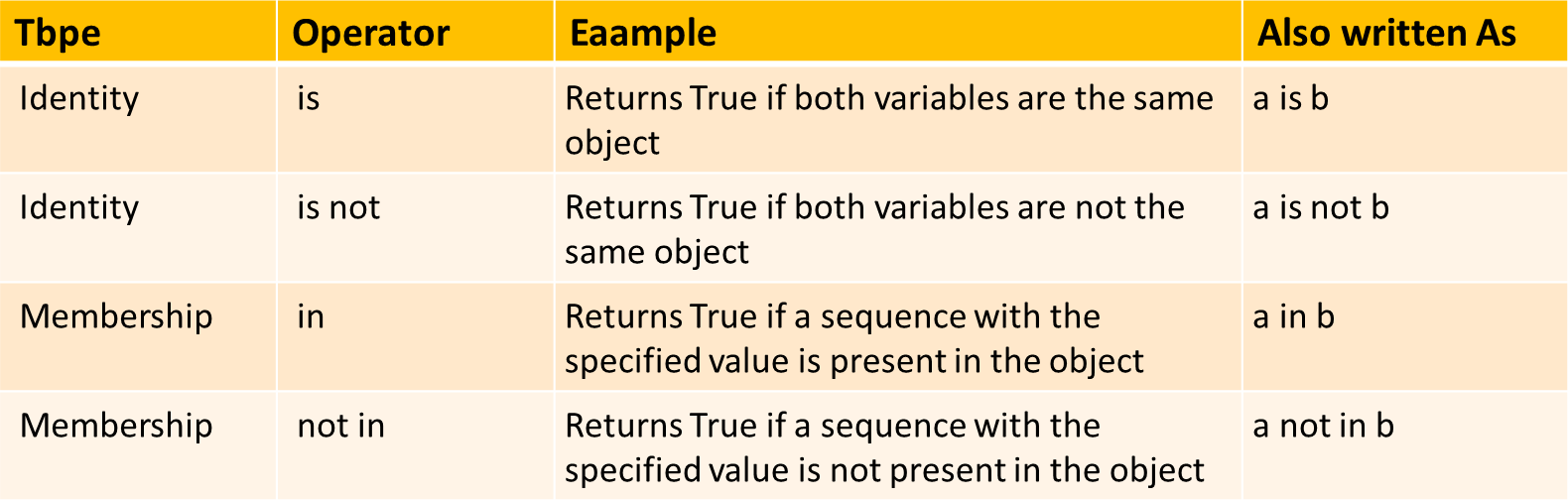
Logical operators are used to combine conditional statements.



## Identity & Membership Operators

Identity operators are used to compare the memory locations of two objects, not just equal but if they are the same objects.

Membership operators checks whether a given value is a member of a sequence (such as strings, lists, and tuples) or not.



# PYTHON TUTORIAL FOR BEGINNERS



**Chapter - 09**

**Conditional Statements in Python**

* Conditional Statement definition
* Types of Conditional Statement
* Conditional Statement examples

## Conditional Statements in Python

Conditional statements allow you to execute code based on **condition** evaluates to True or False. They are essential for **controlling the flow** of a program and making decisions based on different inputs or conditions.

 **Examples**

a = 26

b = 108

**if** b > a**:**

print("b is greater than a")

 *Indentation - whitespace at the beginning ofi a line*

## Types of Conditional Statements

There are 5 types of conditional statements in Python:

* 1. 'if' Statement
  2. 'if-else' statement
  3. 'if-elif-else' statement
  4. Nested 'if else' statement
  5. Conditional Expressions (Ternary Operator)

## 'if' Conditional Statement

The **if** statement is used to test a condition and execute a block of code **only if the condition is true**.

### Syntax:

if condition:

 Code to execute if the condition is true

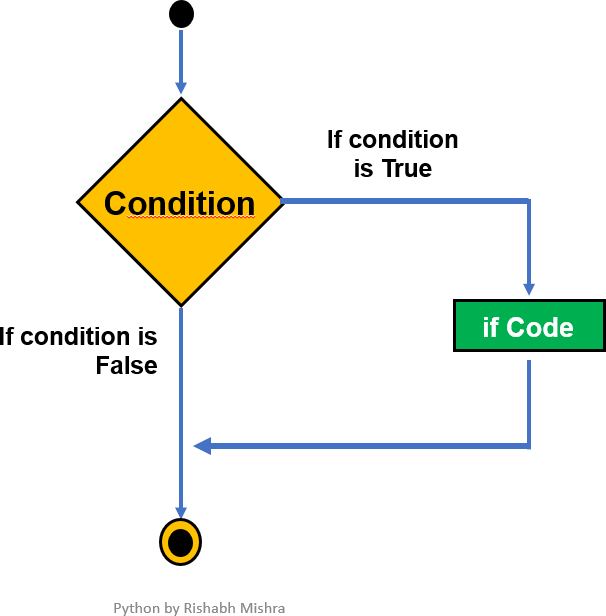
### Example:

age = 26

if age > 19**:**

print("You are an adult")

## 'if' statement flow diagram:

****

1. **'if-else' Conditional Statement**

The **if-else** statement provides an alternative block of code to execute if the condition is **false**.

### Syntax:

if condition**:**

 Code to execute if the condition is true

else**:**

 Code to execute if the condition is false

### Example:

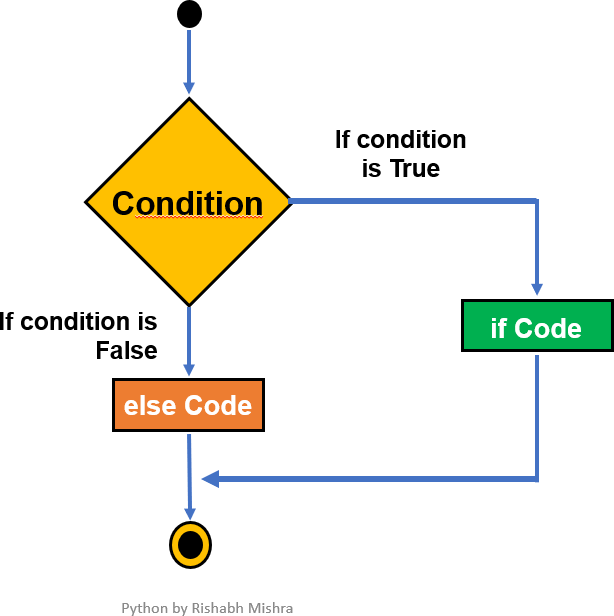
temperature = 30

if temperature > 25**:** print("It's a hot day.")

else**:**

print("It's a cool day.")

## 'if-else' statement flow diagram:

****

1. **'if-elif-else' Conditional Statement**

The **if-elif-else** statement allows to check **multiple conditions** and execute different blocks of code based on which condition is true.

### Syntax:

if condition1**:**

 Code to execute if condition1 is true elif condition2**:**

 Code to execute if condition2 is true

else**:**

 Code to execute if none of the above conditions are true

### Example:

**Grading system:** Let’s write a code to classify the student’s grade based on their total marks (out of hundred).

score = 85

if score >= 90: print("Grade - A")

elif score >= 80: print("Grade - B")

elif score >= 70: print("Grade - C")

else:

print("Grade - D")

## ested 'if-else' Conditional Statement

A **nested if-else** statement in Python involves placing an if-else statement **inside** another if-else statement. This allows for more complex decision-making by checking **multiple conditions that depend on each other**.

### Syntax:

if condition1**:**

 Code block for condition1 being True if condition2**:**

 Code block for condition2 being True

else**:**

****

else**:**

Code block for condition2 being False

 Code block for condition1 being False

**... ..**

### Example:

**Number Classification:** Let's say you want to classify a number as positive, negative, or zero and further classify positive numbers as even or odd.

number = 10

if number > 0:  *First check ifi the number is positive*

if number % 2 == 0:

print("The number is positive and even.") else:

print("The number is positive and odd.") else:  *The number is not positive*

if number == 0:

print("The number is zero.") else:

print("The number is negative.")

## Conditional Expressions

Conditional expressions provide a shorthand way to write simple if-else statements. Also known as Ternary Operator.

### Syntax:

value\_if\_true **if** condition **else** value\_if\_false

### Example:

age = 16

status = "Adult" **if** age >= 18 **else** "Minor" print(status)

# PYTHON TUTORIAL FOR BEGINNERS



**Chapter - 10**

**Functions in Python**

* Functions definition
* Types of Functions
* Function examples

## Functions in Python

A function is a block of code that **performs a specific task**. You can use it whenever you want by calling its name, which saves you from writing the same code multiple times.

**Benefits** of Using Function**:** Increases code **Readability** & **Reusability**. **Basic Concepts:**

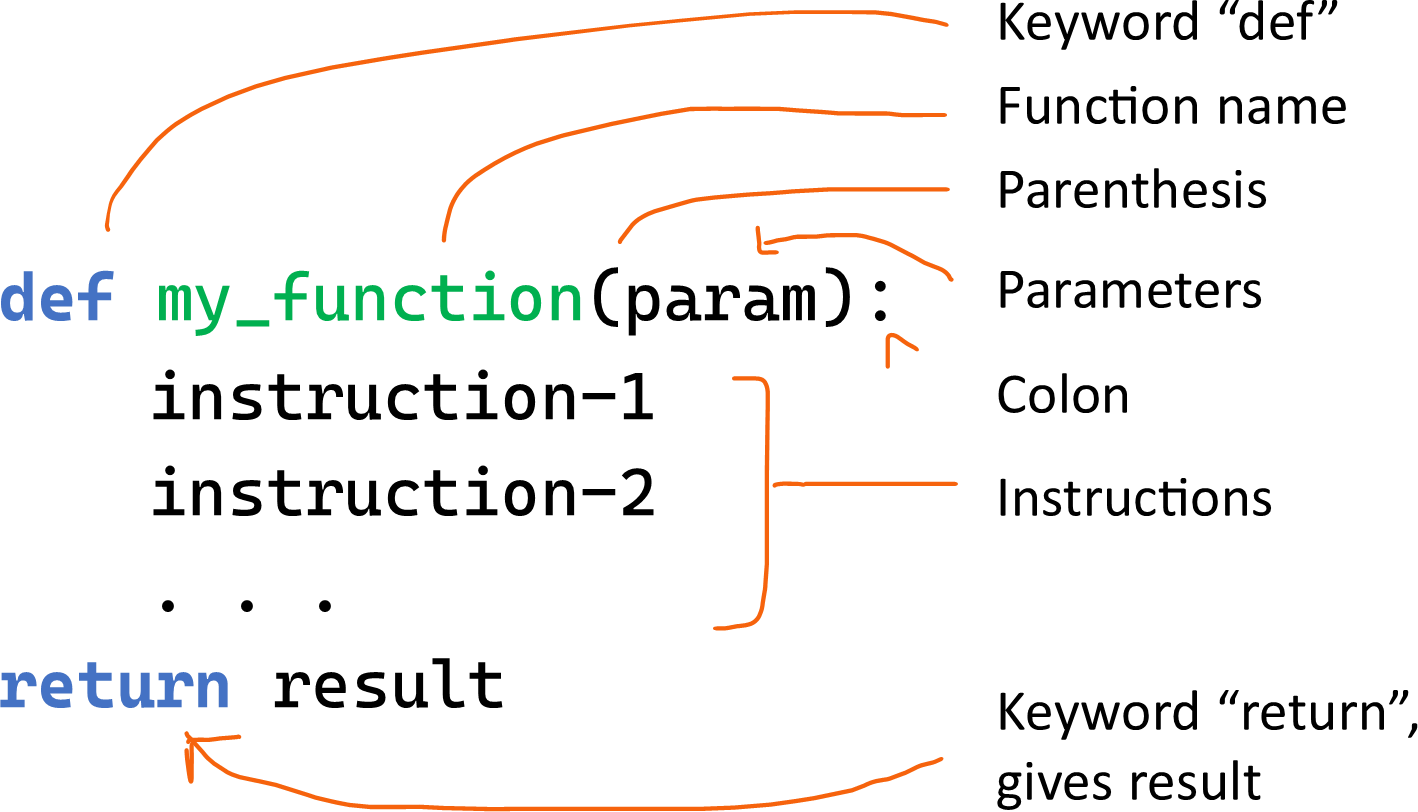
* + **Create** function: Use the def keyword to define a function.
  + **Call** function: Use the function's name followed by () to run it.
  + **Parameter**: The variable listed inside parentheses in function definition.
  + **Argument**: The actual value you pass to function when you call it.

## Types of Functions

Below are the two types of functions in Python:

1. Built-in library function:
   * These are Standard functions in Python that are available to use.
   * Examples: print(), input(), type(), sum(), max(), etc
2. User-defined function:
   * We can create our own functions based on our requirements.
   * Examples: create your own function :)

**Syntax:**

****

**# *return*** *result is optional, Use if you want the function to give back a value*

## Function without Parameters

**Example:1**

 *Create or Defiine Function*

def greetings():

print("Welcome to flython tutorial by Rishabh")

 *Use or call this Function*

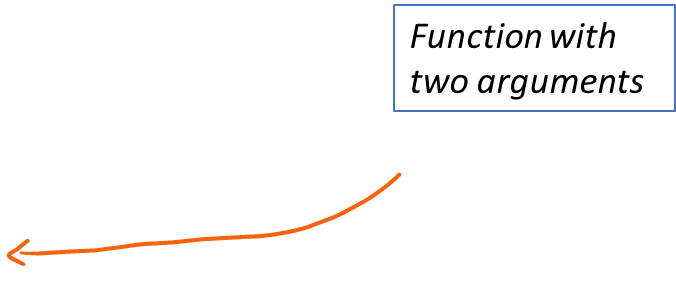
greetings()

 Output: Welcome to flython tutorial by Rishabh

## Function with Parameters

**Example:2**

 function to adds two numbers fi print result. def add2numbers(a, b):

result = a + b

print("The sum is:", result)

 Calling this function with arguments add2numbers(**5**, **3**)

 Output: The sum is: 8

## The return Statement

The return statement is used in a function to **send a result back** to the place where the function was called. When return is executed, the function **stops running** and immediately returns the specified value.

**Example:**

def add(a, b):

**return** a + b  *This line sends back sum ofi a and b*

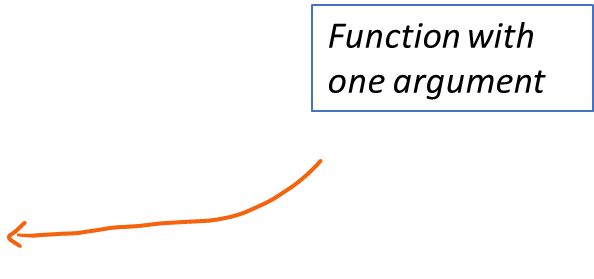
result = add(3, 5) print(result)

 *Output: 8*

## Function with a Return value

**Example:3**

 *fiunction to convert Celsius to Fahrenheit*

def celsius\_to\_fahrenheit(celsius): fahrenheit = (celsius \* 9/5) + 32 return Fahrenheit

*value*

 *Calling this fiunction to return a*

temp\_f = celsius\_to\_fahrenheit(**25**) print("Temperature in Fahrenheit:", temp\_f)

 Output: Temperature in Fahrenheit: 77.0

## Functions – HW

Exercise 1: Write a function that takes two numbers as arguments and returns their sum.

Exercise 2: Create a function that takes a number as input and returns both its square and cube

Exercise 3: Write a function that takes a person's name and age as keyword arguments and prints a greeting message.

Exercise 4: Create a function that prints a message. The function should have a default argument to greet "Hello, World!" if no message is passed.

Exercise 4:Write a Python program to create a **calculator** that can perform at least **five**

different mathematical operations such as addition, subtraction, multiplication, division and average. Ensure that the program is user-friendly, prompting for input and displaying the results clearly.



**Chapter - 12**

**Strings in Python (Part-1)**

* Strings and Examples
* Formatted Strings
* Escape Characters
* String Operators

## Strings in Python

A string is a sequence of **characters**. In Python, strings are enclosed within single (') or double (") or triple (""") quotation marks.

### Examples:

print('Hello World!')  *use type() to check data type*

print("Won’t Give Up!")

print('''"Quotes" and 'single quotes' can be tricky.''') print("\"Quotes\" and 'single quotes' can be tricky.")

## Types of Function Arguments

A formatted string in Python is a way to **insert variables** or expressions inside a string. It allows you to format the output in a readable and controlled way.

There are multiple ways to format strings in Python:

1. Old-style formatting (% operator)
2. str.format() method
3. F-strings (formatted string literals)

## Formatted String - % Operator

### Old-style formatting (% operator)

This approach uses the **%** operator and is similar to string formatting in languages like C.

**Syntax:** "string % value"

### Example:

name = "Madhav" age = 16

print("My name is %s and I’m %d." **%** (name, age))

 *%s, %d are placeholders fior strings and integers*

## Formatted String - str.format()

### str.format() method

In Python 3, the **format**() method is more powerful and flexible than the old-style **%**

formatting.

**Syntax:** "string {}"**.format**(value)

### Example:

name = "Madhav" age = 16

|  |  |  |
| --- | --- | --- |
| print("My | name | is {} and I’m {}."**.format**(name, age)) |
| *You can* | *also* | *refierence the variables by index or keyword:* |
| print("My | name | is {0} and I’m {1}."**.format**(name, age)) |
| print("My | name | is {name} and I’m {age}."**.format**(name="Madhav", |
| age=28)) |  |  |

## Formatted String – F-strings

### F-strings (formatted string literals)

In Python 3.6, F-strings are the most concise and efficient way to format strings. You

prefix the string with an f or F, and variables or expressions are embedded directly within curly braces **{}**.

**Syntax:** f"string **{**variable**}**"

### Example:

name = "Madhav" age = 16

print(f"My name is {name} and I’m {age}.")

 *You can also perfiorm expressions inside the placeholders:*

print(f"In 5 years, I will be {age + 5} years old.")

## Escape Characters

Escape characters in Python are **special** characters used in strings to represent

whitespace, symbols, or control characters that would otherwise be difficult to include. An escape character is a **backslash \** followed by the character you want to insert.

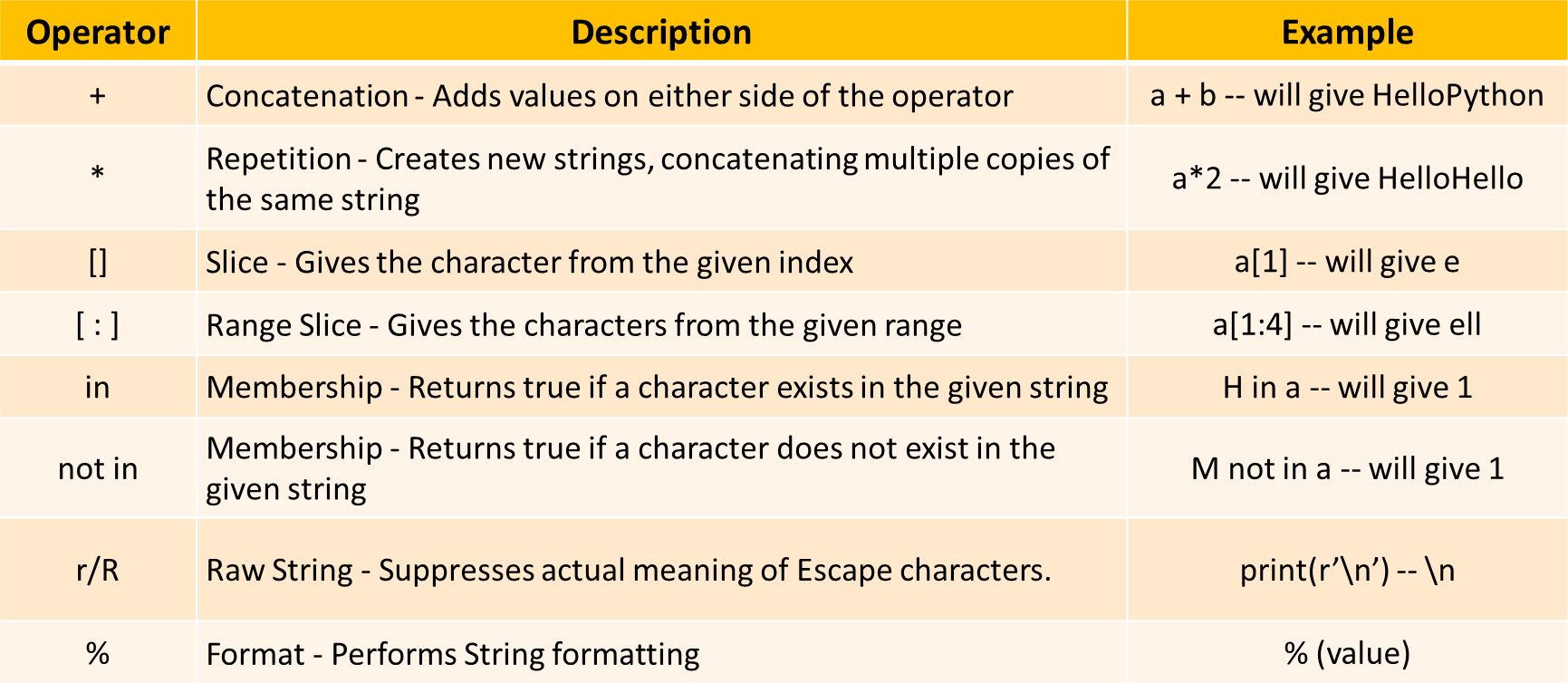
### Examples:

print('Hello\nWorld!')  *\n fior new line*

print('Hello\tWorld!')  *\t fior tab*

print("\"Quotes\" and 'single quotes' can be tricky.")  *print single and double quotes*

## String Operators

****

# PYTHON TUTORIAL FOR BEGINNERS



**Chapter - 13**

**Strings in Python (Part-2)**

* String Indexing
* String Slicing
* String Methods

## String Indexing

You can access individual characters in a string using their **index**. Python uses **zero-based**

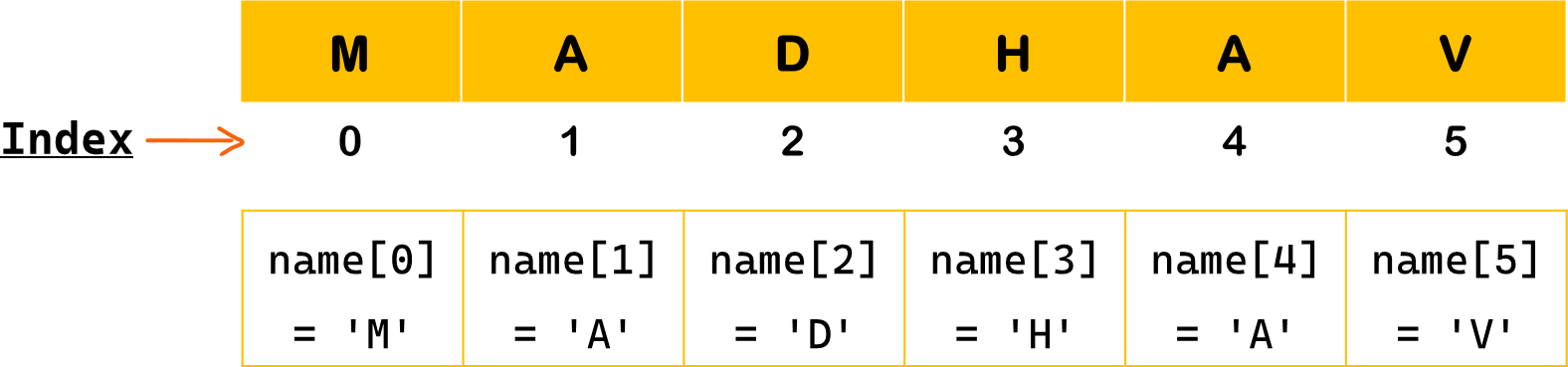
indexing, meaning the first character has an index of **0**. **Index:** Position of the character.

**Syntax:**

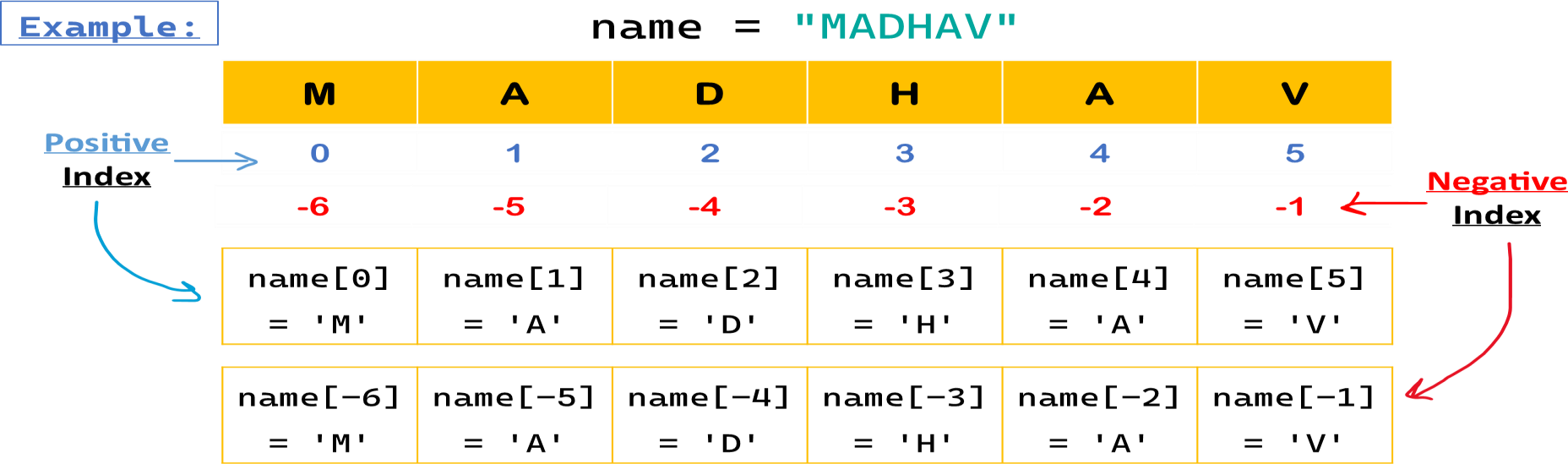
**string[Index\_Value]**

**Example:**

name = "MADHAV"



## String Indexing – Positive & Negative Index

****

**String Slicing**

Slicing in Python is a feature that enables **accessing parts** of the sequence. String slicing allows you to get subset of characters from a string using a specified **range of indices**.

### Syntax:

**string[start : end : step]**

* **start** : The index to start slicing (inclusive). Default value is 0.
* **end** : The index to stop slicing (exclusive). Default value is length of string.
* **Step** : How much to increment the index after each character. Default value is 1.

### Example:

name = "MADHAV" name[0:2] = 'MA'

name[0:5:2] = 'MDA'

## String Slicing - Examples

### Example:

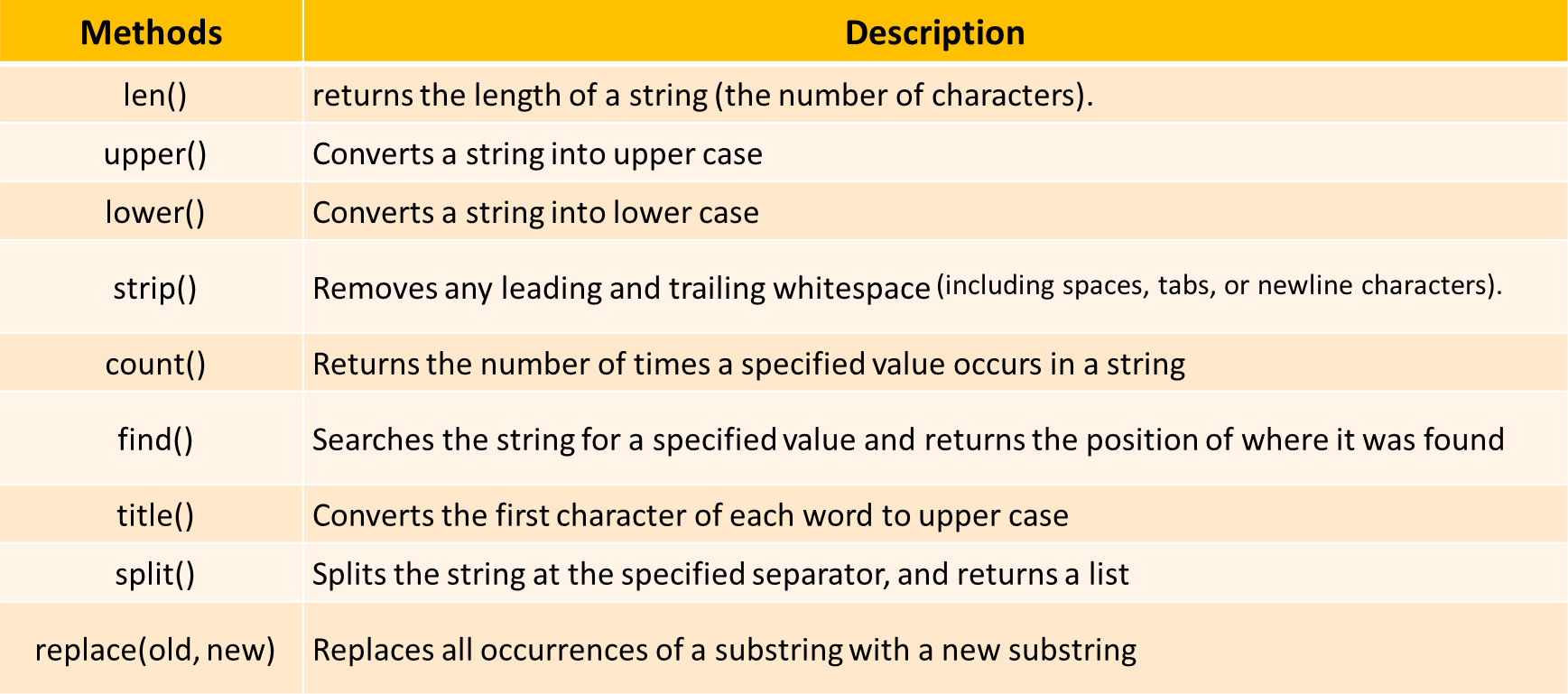
name = "MADHAV"

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| name[0:1] | = | name[:1] | = | 'M' | *fiirst* | *Char* |
| name[0:2] | = | name[:2] | = | 'MA' | *fiirst* | *2 chars* |
| name[2:5] | = | 'DHA' |  |  | *third* | *to fiifith chars* |
| name[5:] = name[-1:] | | | = | 'V' | *last char* | |
| name[4:] = name[-2:] | | | = | 'AV' | *last 2 chars* | |

name[0:5:2] = name[0::2] = 'MDA'  *every second chars* name[1:-1] = 'ADHA'  *exclude fiirst & last chars* name[:] = name[::] = 'MADHAV'  *all chars*

name[::-1] = 'VAHDAM'  *reverse the string*

## String Methods

****

# PYTHON TUTORIAL FOR BEGINNERS



**Chapter - 14**

**Loops in Python**

* Loops & Types
* While Loop
* For Loop
* Range Function
* Loop Control Statements

## Loops in Python

Loops enable you to perform **repetitive tasks** efficiently without writing redundant code. They

iterate over a sequence (like a list, tuple, string, or range) or execute a block of code as long as a specific **condition is met**.

### Types of Loops in Python

1. While loop
2. For loop
3. Nested loop

## While Loop

The while loop **repeatedly** executes a block of code as long as a given condition remains

True. It checks the condition before each iteration.

**Syntax:**

**while** condition**:**

 *Code block to execute*

**Example:** *Print numbers firom 0 to 3*

count = 0

**while** count < 4:  *Condition*

print(count) count += 1

 *Output: 0 1 2 3*

## While Loop Example

**else Statement:** An else clause can be added to loops. It executes after the loop finishes normally (i.e., not terminated by break). ***Example:***

count = 3

**while** count > 0:  *Condition* print("Countdown:", count) count -= 1

### else:

print("Liftoff!")  *Run afiter while loop ends*

## For Loop

The for loop in Python is used to iterate over a sequence (such as a list, tuple, dictionary, set, or string) and execute a block of code for **each element** in that sequence.

**Syntax:**

**for** variable **in** sequence**:**

 *Code block to execute*

**Example:** *iterate over each character in language*

lat:ua:v = 'flQtkot’

**for** x **in** language:

print(x)  *Output: P y t h o n*

## Using range() Function

To **repeat** a block of code a specified number of times, we use the range() function. The range() function returns a sequence of numbers, starting from 0 by default,

increments by 1 (by default), and stops before a specified number.

### Syntax:

**range**(stop) **range**(start, stop) **range**(start, stop, step)

* + **start**: (optional) The beginning of the sequence. Defaults is 0. (inclusive)
  + **stop**: The end of the sequence (exclusive).
  + **step**: (optional) The difference between each number in the sequence. Defaults is 1.

## range() Function Example

**Example1:** *Basic usage with One Argument - Stop*

**for** i in range(5): print(i)

 *Output: 0 1 2 3 4*

**Example2:** *Basic usage with Start, Stop and Step*

**for** i in range(1, 10, 2): print(i)

 *Output: 1 3 5 7 9*

## For Loop Example

**else Statement:** An else clause can be added to loops. It executes after the loop finishes normally (i.e., not terminated by break).

**Example:**

**for** i **in** range(3): print(i)

### else:

print("Loop completed")

 *Output: 0 1 2 Loop Completed*

## while loop VS for loop

### while loop

* + - A while loop keeps running as long as a **condition is true**.
    - It is generally used when you **don’t know** how many iterations will be needed beforehand, and loop continues based on a condition.

### for loop

* + - A for loop **iterates over a sequence** (like a strings, list, tuple, or range) and runs the loop for each item in that sequence.
    - It is used when **you know** in advance how many times you want to repeat a block of code.

## Loop Control Statements

Loop control statements allow you to **alter** the normal flow of a loop. Python supports 2 clauses within loops:

* + - **break Statement**
    - **continue Statement**

## Loop Control - break Statement

**break Statement:** The break statement terminates the loop entirely, exiting from it immediately.**Example:**

**for** i **in** range(5):

**if** i == 3:

### break

print(i)  *Output: 0 1 2*

Above example, the loop **terminated** when condition met true for i == 3

## Loop Control - continue Statement

**continue Statement:** The continue statement **skips** the current iteration and moves to the next one.

**Example:**

**for** i **in** range(5):

**if** i == 3:

### continue

print(i)  *Output: 0 1 2*

Above example, the loop **skips** when condition met true for i == 3

# PYTHON TUTORIAL FOR BEGINNERS



**Chapter - 13**

**Modules, Packages & Libraries in Python**

* What is Module
* Create & Use a Module
* What is Package
* What is Library
* Python pip
* Most used Libraries

## Modules in Python

A module is a single Python file (.py) containing Python code. It can include functions, classes, and variables that you can reuse in other programs.

### Why use modules?

* To organize code into smaller, manageable chunks.
* To reuse code across multiple programs.

#### # Create a module:

* Save the following as mymodule.py

def say\_hello(name):

return print(f"Hello, {name}!")

#### # Use the module:

**import** mymodule greetings.say\_hello("Madhav") *# Output: Hello, Madhav!*

## Packages in Python

A package is a **collection** of modules organized in **directories** (folders) with an init .py file. It allows you to structure your Python projects logically.

### Why use packages?

* To group related modules together.
* To create larger applications or libraries.

#### # Structure Example:

my\_package/

init .py math\_utils.py string\_utils.py

#### # Use the package:

Syntax: **from** my\_package **import** <package\_name>

Example: **from** my\_package **import** math\_utils, string\_utils

## Libraries in Python

A library is a collection of modules and packages that provide pre-written functionality for your program. Libraries are typically larger and more feature-rich than packages or modules.

### Why use libraries?

To avoid writing common functionality from scratch.

To leverage powerful tools developed by the community.

**Example**: Python has many popular libraries, such as:

* Pandas: For data manipulation.
* Matplotlib: For plotting and visualization.

#### # Using a library (Pandas):

**import** pandas as pd

## Python PIP

pip stands for "Pip Installs Packages". It is the package manager for Python that allows you to install, update, and manage Python libraries (packages) from the Python Package Index (PyPI).

*Think of pip as an app store for Python libraries. You use it to search, install, and manage Python tools, just like downloading apps on your phone.*

When you use pip install <package\_name>, it:

* Connects to PyPI (Python Package Index) online.
* Downloads the specified library or package.
* Installs it into your Python environment.

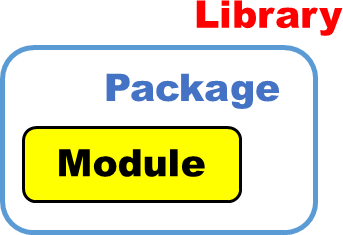
To install packages, we use: **pip install** *<library\_name>*

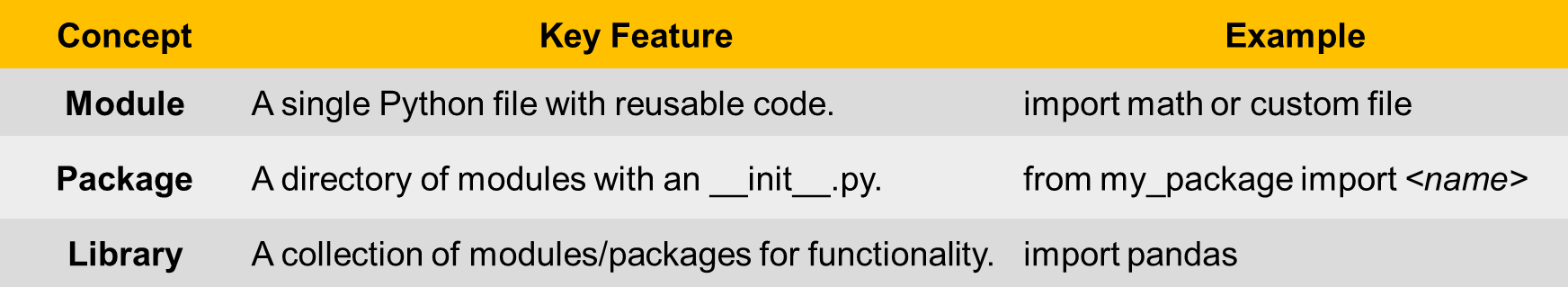
**Example**: installing pandas to work on dataframe:

pip install pandas

## Summary: Module, Package and Library

* **Module:** A single page.
* **Package:** A book containing multiple pages.
* **Library:** A book store with many books.





## Most Used Python Libraries

### Data Analytics, data visualization and ML

**Application Library Description Install Command**

**Data Analytics**

**Pandas** Data manipulation and analysis.

**NumPy** Numerical computing with array support.

**SciPy** Scientific computing and technical computing.

**Statsmodels** Statistical modeling and

testing.

**Dask** Parallel computing for large datasets.

pip install pandas pip install numpy pip install scipy

pip install statsmodels pip install dask

**Data Visualization**

**Matplotlib** Basic plotting and visualization. pip install matplotlib

**Seaborn** Statistical data visualization. pip install seaborn

**Plotly** Interactive graphs and dashboards.

**Scikit-learn** Classic machine learning

algorithms.

pip install plotly

pip install scikit-learn

**Machine Learning**

**TensorFlow** Deep learning and ML models. pip install tensorflow

Deep learning with dynamic

**fi Deep Learning**

**PyTorch**

pip install torch torchvision

computation.

**Keras** High-level deep learning API. pip install keras

**XGBoost** Gradient boosting for structured data.

pip install xgboost

### Web Scraping, web development and game development

|  |  |  |  |
| --- | --- | --- | --- |
| **Application** | **Library** | **Description** | **Install Command** |
|  | **BeautifulSoup** | Parsing HTML and XML for data pip install beautifulsoup4  extraction. | |
|  | **Scrapy** | Advanced web scraping framework. | pip install scrapy |
| **Web Scraping** | **Selenium** | Browser automation for scraping dynamic sites. | pip install selenium |
|  | **Requests** | HTTP library for fetching web pages. | pip install requests |
|  | **Lxml** | Fast XML and HTML parsing. | pip install lxml |
|  | **Django** | Full-stack web framework. | pip install django |
| **Web Development** | **Flask** | Lightweight web framework. | pip install flask |
|  | **FastAPI**  **Pygame** | High-performance API framework.  Game development library. | pip install fastapi  pip install pygame |
|  |
| **Game Development** | **Arcade** | Advanced 2D game development library. | pip install arcade |
|  | **Panda3D** | Real-time 3D rendering and game creation. | pip install panda3d |