📘 Python Basics: Intro, Data Types & Operators

🔹 1. Introduction to Python

- **High-level programming language** – easy to read and write.
- **Interpreted** – code runs line by line (no need to compile).
- **Dynamically typed** – no need to declare data type explicitly.
- **Portable** – works on Windows, Mac, Linux.
- **Widely used in**: Data Science, AI/ML, Web Development, Automation, etc.

Example:

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

🔹 2. Data Types in Python

Python has **built-in data types**:

a) Numeric Types

- **int** → whole numbers
- x = 10
- **float** → decimal numbers
- y = 3.14
- **complex** → numbers with real + imaginary part
- z = 2 + 3j

b) Text Type

- **str** → sequence of characters (strings)
- name = "Python"

c) Sequence Types

- **list** → ordered, mutable
- fruits = ["apple", "banana", "cherry"]
- **tuple** → ordered, immutable
- point = (10, 20)
- **range** → sequence of numbers

- numbers = range(5) # 0 to 4

d) Mapping Type

- **dict** → key-value pairs
- student = {"name": "Tizu", "age": 21}

e) Set Types

- **set** → unordered, unique elements
- s = {1, 2, 3}
- **frozenset** → immutable set

f) Boolean Type

- **bool** → True / False
- is_active = True

g) Binary Types

- **bytes, bytearray, memoryview** → used for binary data

🔹 3. Operators in Python

Operators are symbols used to perform operations on variables.

a) Arithmetic Operators

- + Addition
- - Subtraction
- * Multiplication
- / Division (float)
- // Floor Division (integer result)
- % Modulus (remainder)
- ** Exponent (power)

b) Comparison (Relational) Operators

- == Equal
- != Not equal
- > Greater than

- < Less than
- >= Greater or equal
- <= Less or equal

```
print(5 > 3)  # True
```

c) Logical Operators

- and – True if both are True
- or – True if at least one is True
- not – Negates condition

```
print(True and False)  # False
```

d) Assignment Operators

- = Assign value
- += Add and assign
- -= Subtract and assign
- *= Multiply and assign
- /= Divide and assign
- //= Floor divide and assign
- %= Modulus and assign
- **= Exponent and assign

```
x = 5
x += 3  # x = x + 3 → 8
```

e) Bitwise Operators (work on binary level)

- & AND
- | OR
- ^ XOR
- ~ NOT
- << Left shift
- >> Right shift

```
print(5 & 3)  # 1
```

f) Membership Operators

- in → True if value exists in sequence
- not in → True if value does not exist

```
print("a" in "apple")  # True
```

g) Identity Operators

- is → True if both refer to the same object
- is not → True if not the same object

```
x = [1, 2]
y = [1, 2]
print(x is y)     # False (different objects)
print(x == y)     # True (same values)
```

Conditional Statements

- Conditional statements let a program **make decisions** and execute code **based on conditions (True/False)**.
- Used for **flow control**.

🔹 1. if Statement

Executes a block of code only if the condition is **True**.

```
age = 18
if age >= 18:
   print("You are eligible to vote.")
```

🔹 2. if-else Statement

Runs one block if condition is **True**, otherwise runs another block.

```
age = 16
if age >= 18:
   print("Eligible to vote")
else:
```

```
    print("Not eligible")
```

🔹 3. if-elif-else Ladder

Checks **multiple conditions** in sequence.

```
marks = 72

if marks >= 90:
    print("Grade A")
elif marks >= 75:
    print("Grade B")
elif marks >= 50:
    print("Grade C")
else:
    print("Fail")
```

🔹 4. Nested if

if inside another if. Useful for **multiple-level checks**.

```
num = 15
if num > 0:
    if num % 2 == 0:
        print("Positive Even")
    else:
        print("Positive Odd")
```

Iterative Statements & Functions

🔹 1. Iterative Statements (Loops)

Loops are used to **repeat a block of code multiple times**.

a) for loop

- Used for **iteration over a sequence** (list, tuple, string, range, etc.).

```
for i in range(5):
  print(i)   # 0 1 2 3 4
```

---

b) while loop

- Repeats as long as the condition is **True**.

```
count = 1
while count <= 5:
  print(count)
  count += 1
```

---

c) Loop Control Statements

1. **break** → exits the loop immediately

   ```
   for i in range(10):
     if i == 5:
       break
     print(i)
   ```

2. **continue** → skips the current iteration, moves to next

   ```
   for i in range(5):
     if i == 2:
       continue
     print(i)
   ```

3. **pass** → does nothing (placeholder)

   ```
   for i in range(3):
     pass  # future code can be added here
   ```

---

d) Nested Loops

- Loops inside loops.

```
for i in range(3):
    for j in range(2):
        print(i, j)
```

🔹 2. Functions in Python

Functions are **reusable blocks of code** that perform a specific task.

a) Defining a Function

```
def greet():
    print("Hello, welcome to Python!")
```

b) Calling a Function

```
greet()   # Output: Hello, welcome to Python!
```

---

c) Function with Parameters

```
def add(a, b):
    return a + b

print(add(5, 3))    # 8
```

---

d) Default Parameters

```
def greet(name="Guest"):
    print("Hello", name)

greet()           # Hello Guest
greet("Tizu")     # Hello Tizu
```

---

e) Keyword Arguments

```
def student(name, age):
    print(name, age)

student(age=21, name="Tizu")
```

---

f) Variable-length Arguments

1. ***args** → multiple positional arguments (tuple)

```
def add(*nums):
    return sum(nums)

print(add(1, 2, 3, 4))
```

2. ****kwargs** → multiple keyword arguments (dictionary)

```
def show_details(**info):
    print(info)

show_details(name="Tizu", age=21)
```

---

g) Return Statement

- Functions can return values.

```
def square(x):
    return x * x

print(square(4))   # 16
```

---

h) Lambda Functions (Anonymous Functions)

- Small one-line functions using lambda.

```
square = lambda x: x * x
print(square(5))   # 25
```

OOP

- **Programming paradigm** based on the concept of **objects**.
- Objects are **real-world entities** (like car, student, bank account) that have:
    - **Attributes (data/properties)**
    - **Methods (functions/behaviors)**

✅ OOP makes code **modular, reusable, and easier to maintain**.

🔹 Basic Concepts of OOP

a) Class

- A **blueprint** for creating objects.
- Defines attributes and methods.

```
class Student:
    def __init__(self, name, age):  # constructor
        self.name = name
        self.age = age

    def show(self):                 # method
        print("Name:", self.name, "Age:", self.age)
```

---

b) Object

- An **instance of a class**.

```
s1 = Student("Tizu", 21)
s1.show()   # Output: Name: Tizu Age: 21
```

---

c) Constructor (__init__)

- Special method called automatically when object is created.

```
class Car:
    def __init__(self, brand):
        self.brand = brand
```

---

d) Self

- Refers to the **current object** of the class.
- Used to access attributes and methods.

🔹 OOP Principles

1️⃣ Encapsulation

- **Bundling of data (attributes) and methods** inside a class.
- Access modifiers:

- public → accessible everywhere
- _protected → accessible inside class + subclasses
- __private → accessible only inside class

class Bank:

```
class Bank:
    def __init__(self, balance):
        self.__balance = balance

    def deposit(self, amount):
        self.__balance += amount

    def get_balance(self):
        return self.__balance
```

---

2 Inheritance

- **Child class** can inherit attributes & methods from **Parent class**.
- Promotes **code reusability**.

```
class Animal:
    def speak(self):
        print("Animal speaks")

class Dog(Animal):   # Dog inherits Animal
    def speak(self):
        print("Bark")
```

Types of Inheritance in Python:

- Single (one parent → one child)
- Multiple (many parents → one child)
- Multilevel (parent → child → grandchild)
- Hierarchical (one parent → many children)
- Hybrid (combination)

---

3 Polymorphism

- Same function name but different behavior.

1. **Method Overriding**

```python
class Bird:
    def fly(self):
        print("Some birds can fly")

class Parrot(Bird):
    def fly(self):
        print("Parrot flies high")
```

2. **Method Overloading**

```python
class Math:
    def add(self, a, b=0, c=0):
        return a + b + c

m = Math()
print(m.add(5))       # 5
print(m.add(5, 10))   # 15
```

🔽 Abstraction

- Hides **implementation details**, shows only **essential features**.
- Achieved using **abstract classes** and **interfaces** .

```python
from abc import ABC, abstractmethod

class Shape(ABC):
    @abstractmethod
    def area(self):
        pass

class Circle(Shape):
    def __init__(self, r):
        self.r = r
    def area(self):
        return 3.14 * self.r * self.r

c = Circle(5)
print(c.area())   # 78.5
```

🔹 Advantages of OOP

- Code reusability
- Easy maintenance & debugging

- Models real-world entities
- Supports modular programming

Statistics

- **Statistics – Measures of Central Tendency**

These measure the **"center"** or **typical value** in data.

a) Mean (Average)

a) Mean (Average)

$$\bar{x} = \frac{\sum x_i}{n}$$

Example: $(2, 4, 6, 8) \rightarrow$ Mean = $\frac{20}{4} = 5$

---

b) Median

- Middle value when data is arranged.
- If n is even → median = average of two middle values.

Example: (3,1,4,2,5) → Median = 3

---

c) Mode

- The **most frequent** value.
  Example: (2,3,4,4,5)→ Mode = 4

- **Statistics – Measures of Dispersion**

These measure **spread/variability** of data.

a) Range

$$\text{Range} = \text{Max} - \text{Min}$$

Example: $(2, 5, 9) \rightarrow$ Range = $9 - 2 = 7$

---

b) Variance

$$\sigma^2 = \frac{\sum (x_i - \bar{x})^2}{n}$$

- Average squared deviation from mean.

---

c) Standard Deviation (SD)

$$\sigma = \sqrt{\sigma^2}$$

- Square root of variance.
- Shows how much values deviate from mean.

---

d) Mean Absolute Deviation (MAD)

$$MAD = \frac{\sum |x_i - \bar{x}|}{n}$$

---

e) Coefficient of Variation (CV)

$$CV = \frac{\sigma}{\bar{x}} \times 100\%$$

- Used to compare variability between datasets.

f) Correlation

- **Correlation** measures the **strength and direction of a linear relationship** between two variables.
- Value ranges:

  $-1 \leq r \leq 1$

Formula:

$$r = \frac{\sum (x_i - \bar{x})(y_i - \bar{y})}{\sqrt{\sum (x_i - \bar{x})^2 \sum (y_i - \bar{y})^2}}$$

Where:

- x_i, y_i = individual data points
- $\bar{x}, \bar{y}$ = mean of x and y

Hypothesis Testing & Statistical Tests

- **1. P-value (P-test concept)**

🔸 **What is a P-value?**

- The **P-value** tells you the **probability** of getting the observed results (or more extreme) if the **null hypothesis (H_0)** is true.
- **Smaller P-value → stronger evidence against H_0.**

✅ **Decision Rule:**

- If **P-value ≤ α** (significance level, e.g., 0.05) → **Reject H_0.**
- If **P-value > α** → **Fail to reject H_0.**

🔹 **2. T-test (Student's t-test)**

Used when:

- **Sample size is small** (n < 30).
- **Population standard deviation (σ) is unknown**.
- Data is approximately **normally distributed**.

🔸 **a) One-sample t-test**

- Compares the **sample mean** to a known/population mean.

$$t = \frac{\bar{x} - \mu}{s/\sqrt{n}}$$

Where:

- $\bar{x}$ = sample mean
- μ = population mean
- s = sample standard deviation
- n = sample size

🔸 **b) Two-sample t-test**

- Compares **means of two independent samples**.

🔸 **c) Paired t-test**

- Used for **before-after** (same group, two measurements).

🔹 3. Z-test

Used when:

- **Sample size is large** (n ≥ 30).
- **Population standard deviation (σ) is known**.

🔸 a) One-sample Z-test

$$z = \frac{\bar{x} - \mu}{\sigma / \sqrt{n}}$$

🔸 b) Two-sample Z-test

- Compares means of two independent samples (with known σ).

🔹 4. F-distribution & F-test

Used to:

- **Compare variances** of two or more samples.
- Used in **Analysis of Variance**.

🔸 Formula for F-test:

$$F = \frac{s_1^2}{s_2^2}$$

Where:

- s_1^2 and s_2^2 are sample variances.

- The **larger variance** is usually placed in the numerator.

✅ **F-distribution properties:**

- Always **positive** (since variance is squared).
- **Right-skewed**.
- Depends on **two degrees of freedom**:
  - df1=n1−1
  - df2=n2−1

Matrices, Eigenvalues, Eigenvectors, and Decompositions

🔹 **1. Matrix Operations**

1. **Addition & Subtraction**
   - **Performed element-wise.**
   - **Both matrices must have the same dimensions.**
   - **Example:**

$$\begin{bmatrix}1 & 2\\3 & 4\end{bmatrix}+\begin{bmatrix}5 & 6\\7 & 8\end{bmatrix}=\begin{bmatrix}6 & 8\\10 & 12\end{bmatrix}$$

Scalar Multiplication

   - **Multiply each element by a constant.**
   - **Example:**

$$2\times\begin{bmatrix}1 & 2\\3 & 4\end{bmatrix}=\begin{bmatrix}2 & 4\\6 & 8\end{bmatrix}$$

2. **Matrix Multiplication**
   - **Dot product of rows and columns.**
   - **The number of columns of first matrix = rows of second matrix.**
   - **Example:**

$$\begin{bmatrix}1 & 2\\3 & 4\end{bmatrix}\cdot\begin{bmatrix}2 & 0\\1 & 2\end{bmatrix}=\begin{bmatrix}4 & 4\\10 & 8\end{bmatrix}$$

3. **Transpose (A^T)**
    - **Flips rows into columns.**
    - **Example:**

$$\begin{bmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \end{bmatrix}^T = \begin{bmatrix} 1 & 4 \\ 2 & 5 \\ 3 & 6 \end{bmatrix}$$

4. **Determinant**
    - **Scalar value of a square matrix that gives info about area/volume scaling.**
    - **If determinant = 0, the matrix is singular (non-invertible).**
5. **Inverse (A^−1)**
    - **Only for square, non-singular matrices.**
    - **Property:**

$$A \cdot A^{-1} = I$$

    - **Used for solving linear systems.**

🔹 **2. Eigenvalues & Eigenvectors**

- **Defined by:**

$$Av = \lambda v$$

- **Eigenvalue (λ): scalar that shows how much the eigenvector is stretched/shrunk.**
- **Eigenvector (v): direction that does not change under transformation.**

Use in Data Science:

- **Dimensionality reduction (PCA)**
- **Feature importance in ML**
- **Stability analysis in systems**

🔹 **3. Singular Value Decomposition (UΣV)**

- **Any matrix A can be decomposed into:**

$$A = U\Sigma V^T$$

Where:

- **U → orthogonal matrix (left singular vectors)**
- **Σ→ diagonal matrix with singular values**
- **V→ orthogonal matrix (right singular vectors)**

📌 **Applications:**

- **PCA (Principal Component Analysis)**
- **Image compression (retain only top singular values)**
- **Noise reduction**
- **Recommender systems**
-

Probability – Introduction, Conditional Probability & Bayes' Theorem

🔹 **1. Probability**

Probability measures the likelihood of an event occurring.

$$P(E) = \frac{\text{Number of favorable outcomes}}{\text{Total number of outcomes}}$$

Values range between **0 and 1**.

- P(E)=0 → Impossible event
- P(E)=1 → Certain event

🔹 **2. Conditional Probability**

Definition: Probability of an event A given that another event B has already occurred.

$$P(A|B) = \frac{P(A \cap B)}{P(B)} \quad (\text{if } P(B) > 0)$$

Example:

A card is drawn from a deck of 52. Find P(King | Face card).

- Face cards = J, Q, K → 12 cards.
- Kings = 4 (all are face cards).

$$P(\text{King} \mid \text{Face card}) = \frac{4}{12} = \frac{1}{3}$$

🔹 3. Bayes' Theorem

Used to **update probabilities** when new information is available.

$$P(A|B) = \frac{P(B|A) \cdot P(A)}{P(B)}$$

Where:

- P(A) = prior probability of A
- P(B|A) = likelihood
- P(A|B) = posterior probability

Calculus for Data Science

🔹 1. Why Calculus in Data Science?

- **Optimization** → Training ML models involves minimizing loss/cost functions (gradient descent).
- **Understanding change** → Rates of change in data (slopes, trends).
- **Probability & Statistics** → Continuous distributions (Normal, Exponential, etc.) use integration/differentiation.
- **Neural Networks** → Backpropagation uses derivatives.

🔹 2. Functions & Limits

a) Function

A function maps input → output:

f(x)=y

Example: $f(x) = x^2 + 3x$

b) Limit

The value a function approaches as input approaches a point.

$$\lim_{x \to a} f(x)$$

Example:

$$\lim_{x \to 0} \frac{\sin x}{x} = 1$$

🔹 **3. Derivatives (Differentiation)**

Definition: The derivative of f(x) measures the **rate of change** of f w.r.t x.

$$f'(x) = \frac{dy}{dx} = \lim_{h \to 0} \frac{f(x+h) - f(x)}{h}$$

Basic Rules:

1. $\frac{d}{dx}(c) = 0$ (constant rule)
2. $\frac{d}{dx}(x^n) = nx^{n-1}$ (power rule)
3. $\frac{d}{dx}(e^x) = e^x$
4. $\frac{d}{dx}(\ln x) = \frac{1}{x}$
5. Sum, Product, Quotient, Chain rules.

Application in Data Science:

- **Gradient Descent** → find slope of loss function and move towards minimum.
- Example:

$$L(w) = (y - wx)^2,$$

$$\frac{dL}{dw} = -2x(y - wx)$$

This derivative guides weight updates.

🔹 4. Partial Derivatives

- Used when function has **multiple variables**.
- Important in **multivariable optimization** (ML models with many features).

$$\frac{\partial f}{\partial x}, \quad \frac{\partial f}{\partial y}$$

Example:

If $f(x, y) = x^2y + y^2$,

$$\frac{\partial f}{\partial x} = 2xy, \quad \frac{\partial f}{\partial y} = x^2 + 2y$$

🔹 5. Gradient

- A **vector of partial derivatives**.
- Points in the direction of steepest increase.

$$\nabla f(x, y) = \left(\frac{\partial f}{\partial x}, \frac{\partial f}{\partial y} \right)$$

In ML: Gradient is used in **gradient descent** to update model parameters.

🔹 6. Integration

Definition: Reverse process of differentiation (area under the curve).

$$\int f(x)dx$$

Rules:

1. $\int x^n dx = \frac{x^{n+1}}{n+1} + C$
2. $\int e^x dx = e^x + C$
3. $\int \frac{1}{x} dx = \ln|x| + C$

🔹 7. Optimization in Data Science

- **Local minima/maxima** → where derivative = 0.
- **Second derivative test:**
  - If f''(x)>0→ minimum
  - If f''(x)<0→ maximum