# Python Basics

Python is a high-level programming language which is:

- **Interpreted:** Python is processed at runtime by the interpreter.
- **Interactive**
- **Object-Oriented:** Python supports Object-Oriented technique of programming.
- **Beginner's Language:** Python is a great language for the beginner-level programmers and supports the development of a wide range of applications.
- Python is derived from **ABC** programming language, which is a general-purpose programming language.
- Rossum chose the name "**Python**", since he was a big fan of Monty Python's Flying Circus.

Release dates for the major and minor versions:

- ✦ **Python 1.0** - January **1994** :Python 1.5 - December 31, 1997
- ✦ **Python 3.0** - December 3, **2008** :Python 3.7 – June 27, **2018**

- Python IDEs : Vim ,Eclipse with PyDev ,Sublime Text ,Emacs ,Komodo Edit ,PyCharm
- Python Web Frameworks : Django ,Flask,Pylons,Pyramid,TurboGears ,Web2py

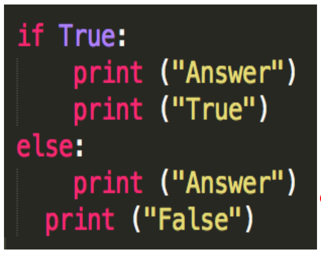
## Python Features

| | |
|---|---|
| **Easy to read** | ✓ **Python scripts have clear syntax, simple structure and very few protocols to remember before programming.** |
| **Easy to Maintain** | ✓ Python code is easy to write and debug. Python's success is that its source code is fairly easy-to-maintain. |

| | |
|---|---|
| **Portable** | ✓ Python can run on a wide variety of Operating systems and platforms and providing the similar interface on all platforms. |
| **Broad Standard Libraries** | ✓ Python comes with many prebuilt libraries apx. 21K |
| **High Level programming** | ✓ Python is intended to make complex programming simpler. Python deals with memory addresses, garbage collection etc internally. |
| **Interactive** | ✓ Python provide an interactive shell to test the things before implementation. It provides the user the direct interface with Python. |
| **Database Interfaces** | ✓ Python provides interfaces to all major commercial databases. These interfaces are pretty easy to use. |
| **GUI programming** | ✓ Python supports GUI applications and has framework for Web. Interface to tkinter, WXPython, DJango in Python make it. |

## Basic Syntax

- **Indentation** is used in Python to delimit blocks. The number of spaces is variable, but all statements within the same block must be indented the same amount.

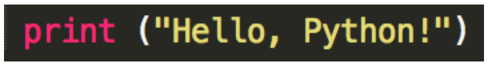


```
if True:
    print ("Answer")
    print ("True")
else:
    print ("Answer")
  print ("False")
```

àError!

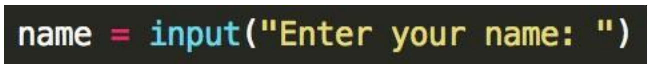
- The header line for compound statements, such as if, while, def, and
- class should be terminated with a colon ( : )
- The semicolon ( ; ) is optional at the end of statement
- # is used for comments

- Printing to the Screen:



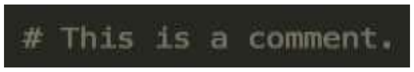
```python
print ("Hello, Python!")
```

- Reading Keyboard Input:



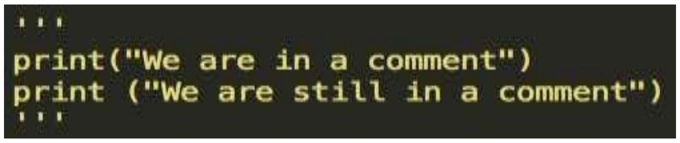
```python
name = input("Enter your name: ")
```

- **Comments**
- Single line:



```python
# This is a comment.
```

- Multiple lines:



```python
'''
print("We are in a comment")
print ("We are still in a comment")
'''
```

- Python files have extension **.py**

## Variables

### Creating Variables:

Unlike other programming languages, Python has no command for declaring a variable.

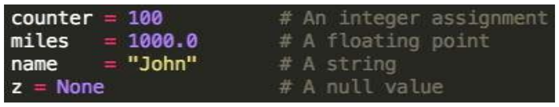
A variable is created the moment you first assign a value to it.

### Variable Names

A variable can have a short name (like x and y) or a more descriptive name (age, carname, total_volume).
Rules for Python variables:

- A variable name must start with a letter or the underscore character
- A variable name cannot start with a number
- A variable name can only contain alpha-numeric characters and underscores (A-z, 0-9, and _ )
- Variable names are case-sensitive (age, Age and AGE are three different variables)

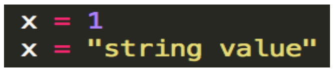
- Python is dynamically typed. You do not need to declare variables!



```
counter = 100          # An integer assignment
miles   = 1000.0       # A floating point
name    = "John"       # A string
z = None               # A null value
```

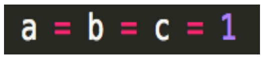
- The declaration happens automatically  when you assign a value to a variable.

- Variables can change type, simply by assigning them a new value of a different type.



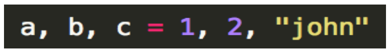
```
x = 1
x = "string value"
```

- Python allows you to assign a single value to several variables simultaneously.



```
a = b = c = 1
```

- You can also assign multiple objects to multiple variables.

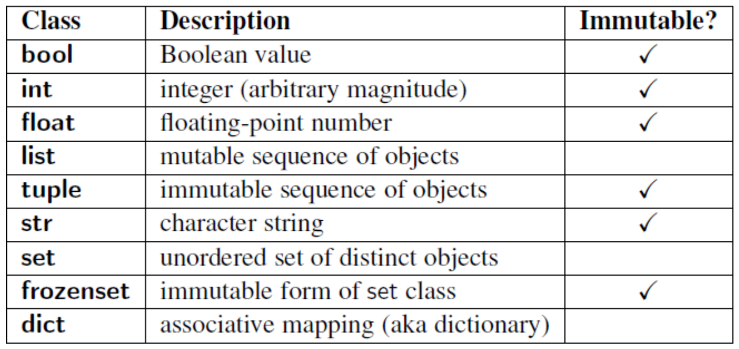


```
a, b, c = 1, 2, "john"
```

## Mutable and Immutable Objects in Python

Python objects can be either **mutable** or **immutable**. Since everything in Python is an Object, every variable holds an object instance. When an object is initiated, it is assigned a unique object id. Its type is defined at runtime and once set can never change, however its state can be changed if it is mutable. Simple put, a **mutable** object can be changed after it is created, and an **immutable** object can't.

*Objects of built-in types like (int, float, bool, str, tuple, unicode) are immutable. Objects of built-in types like (list, set, dict) are mutable. Custom classes are generally mutable. To simulate immutability in a class, one should override attribute setting and deletion to raise exceptions.*



| Class | Description | Immutable? |
|---|---|---|
| bool | Boolean value | ✓ |
| int | integer (arbitrary magnitude) | ✓ |
| float | floating-point number | ✓ |
| list | mutable sequence of objects | |
| tuple | immutable sequence of objects | ✓ |
| str | character string | ✓ |
| set | unordered set of distinct objects | |
| frozenset | immutable form of set class | ✓ |
| dict | associative mapping (aka dictionary) | |

**Python Operators and Types**

Operators are the constructs which can manipulate the value of operands.

Consider the expression 4 + 5 = 9. Here, 4 and 5 are called operands and + is called operator.

Types of Operators

Python language supports the following types of operators.

- Arithmetic Operators
- Comparison (Relational) Operators
- Assignment Operators
- Logical Operators
- Bitwise Operators
- Membership Operators
- Identity Operators

Python Arithmetic Operators

Assume variable ‘a’ holds 10 and variable ‘b’ holds 20, then –

| Operator | Description | Example |
|---|---|---|
| + Addition | Adds values on either side of the operator. | a + b = 30 |
| - Subtraction | Subtracts right hand operand from left hand operand. | a – b = -10 |

| | | |
|---|---|---|
| * Multiplication | Multiplies values on either side of the operator | a * b = 200 |
| / Division | Divides left hand operand by right hand operand | b / a = 2 |
| % Modulus | Divides left hand operand by right hand operand and returns remainder | b % a = 0 |
| ** Exponent | Performs exponential (power) calculation on operators | a**b =10 to the power 20 |
| // | Floor Division - The division of operands where the result is the quotient in which the digits after the decimal point are removed. But if one of the operands is negative, the result is floored, i.e., rounded away from zero (towards negative infinity) − | 9//2 = 4 and 9.0//2.0 = 4.0, -11//3 = -4, -11.0//3 = -4.0 |

Python Comparison Operators

These operators compare the values on either sides of them and decide the relation among them. They are also called Relational operators.

Assume variable a holds 10 and variable b holds 20, then −

| **Operator** | **Description** | **Example** |
|---|---|---|
| == | If the values of two operands are equal, then the condition becomes true. | (a == b) is not true. |
| != | If values of two operands are not equal, then condition becomes true. | (a != b) is true. |

| | | |
|---|---|---|
| <> | If values of two operands are not equal, then condition becomes true. | (a <> b) is true. This is similar to != operator. |
| > | If the value of left operand is greater than the value of right operand, then condition becomes true. | (a > b) is not true. |
| < | If the value of left operand is less than the value of right operand, then condition becomes true. | (a < b) is true. |
| >= | If the value of left operand is greater than or equal to the value of right operand, then condition becomes true. | (a >= b) is not true. |
| <= | If the value of left operand is less than or equal to the value of right operand, then condition becomes true. | (a <= b) is true. |

Python Logical Operators

Following logical operators supported by Python language. Assume variable a holds 10 and variable b holds 20 then

| **Operator** | **Description** | **Example** |
|---|---|---|
| and Logical AND | If both the operands are non-zero then condition becomes true. | (a and b) is true. |
| or Logical OR | If any of the two operands are non-zero then condition becomes true. | (a or b) is true. |
| not Logical NOT | Used to reverse the logical state of its operand. | Not(a and b) is false. |

Python Membership Operators

Python’s membership operators test for membership in a sequence, such as strings, lists, or tuples. There are two membership operators as explained below −

| Operator | Description | Example |
|---|---|---|
| in | Evaluates to true if it finds a variable in the specified sequence and false otherwise. | x in y, here ‘in’ results in a 1 if x is a member of sequence y. |
| not in | Evaluates to true if it does not finds a variable in the specified sequence and false otherwise. | x not in y, here ‘not in’ results in a 1 if x is not a member of sequence y. |

Python Identity Operators

Identity operators compare the memory locations of two objects. There are two Identity operators explained below −

| Operator | Description | Example |
|---|---|---|
| is | Evaluates to true if the variables on either side of the operator point to the same object and false otherwise. | x is y, here **is** results in 1 if id(x) equals id(y). |
| is not | Evaluates to false if the variables on either side of the operator point to the same object and true otherwise. | x is not y, here **is not** results in 1 if id(x) is not equal to id(y). |

**Python Types**

- **Numeric Types:**
    - **int: Integers (whole numbers, e.g., 5, -10).**
    - **float: Floating-point numbers (decimal numbers, e.g., 3.14, -0.5).**
    - **complex: Complex numbers (e.g., 2 + 3j).**
- **Sequence Types:**
    - **str: Strings (sequences of characters, enclosed in single or double quotes, e.g., "hello", 'Python').**

    - **list: Ordered, mutable sequences of items (enclosed in square brackets, e.g., [1, 2, 'a']).**
    - **tuple: Ordered, immutable sequences of items (enclosed in parentheses, e.g., (1, 2, 'a')).**
    - **range: An immutable sequence of numbers, often used for looping.**
- **Set Types:**
    - **set: Unordered collections of unique, mutable items (enclosed in curly braces, e.g., {1, 2, 3}).**
    - **frozenset: Immutable version of a set.**
- **Mapping Type:**
    - **dict: Dictionaries (unordered collections of key-value pairs, enclosed in curly braces, e.g., {'name': 'Alice', 'age': 30}).**
- **Boolean Type:**
    - **bool: Represents truth values (True or False).**
- **None Type:**
    - **NoneType: Represents the absence of a value, with None being the only value of this type.**

If statement

This executes a block of code only if the specified condition is True.

if-else statement.

This provides an alternative block of code to execute if the if condition is False.

if-elif-else ladder.

This allows for checking multiple conditions sequentially. The first elif condition that evaluates to True will have its corresponding block executed. If none of the if or elif conditions are met, the else block (if present) will be executed.

Iterative statements

1. for loop:

The for loop is used for iterating over a sequence (such as a list, tuple, string, or range) or any other iterable object. It executes the code block for each item in the sequence.

2. while loop:

The while loop repeatedly executes a block of code as long as a specified condition remains true. The condition is evaluated at the beginning of each iteration.

FUNCTIONS

Key aspects of Python functions:

- **Definition:** Functions are defined using the def keyword, followed by the function name, parentheses (), and a colon :. The code block belonging to the function is indented.

```
Eg: def my_function():
        print("This is inside my function.")


        my_function()
```

- **Parameters and Arguments:** Functions can accept input values through parameters defined in the function signature. When calling the function, actual values passed are called arguments.

```
def greet(name):  # 'name' is a parameter
          print(f"Hello, {name}!")

 greet("Alice")
```

- **Return Values:** Functions can return a value using the return keyword. If no return statement is present, the function implicitly returns None.

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

result = add_numbers(5, 3)
print(result)
```

- **Types of Functions:**
    - **Built-in Functions:** Functions provided by Python's standard library, such as print(), len(), int(), etc.
    - **User-defined Functions:** Functions created by the programmer to perform specific tasks.
    - **Functions from Modules:** Functions defined within modules (e.g., math.sqrt() from the math module) that need to be imported before use.

Object-Oriented Programming (OOP) in Python is a programming paradigm that structures code around the concept of "objects." These objects encapsulate both data (attributes) and the functions that operate on that data (methods).

Core Concepts of OOP in Python:

- **Classes:** A class acts as a blueprint or a template for creating objects. It defines the attributes (variables) and methods (functions) that all objects created from that class will possess.

- **Objects:** An object is an instance of a class. When you create an object from a class, it gets its **Encapsulation:**

  This principle involves bundling data and the methods that operate on that data within a single unit (the class). It helps in controlling access to data and preventing unintended modifications.

- **Inheritance:**

  Inheritance allows a new class (subclass or child class) to inherit attributes and methods from an existing class (superclass or parent class). This promotes code reusability and establishes a hierarchical relationship between classes.

- **Polymorphism:**

  Polymorphism means "many forms." In OOP, it allows objects of different classes to be treated as objects of a common type. This is often achieved through method overriding, where a subclass provides its own implementation of a method already defined in its superclass.

- **Abstraction:**

  Abstraction focuses on showing only essential information and hiding complex implementation details. In Python, this can be achieved through abstract classes and methods (using the abc module), which define interfaces without providing full implementations.

**Matrix Operations, Eigenvalues, Eigenvectors, and UV Decomposition**

**1. Matrix Addition**

You can only add matrices of the same size. Add corresponding elements.

Example:

A = [[1, 2], [3, 4]]
B = [[5, 6], [7, 8]]

A + B = [[1+5, 2+6], [3+7, 4+8]] = [[6, 8], [10, 12]]

**2. Matrix Multiplication**

Matrix multiplication is not element-wise. You take row × column dot product.

If A is (m × n) and B is (n × p), then AB is (m × p).

Example:

A = [[1, 2], [3, 4]]
B = [[5, 6], [7, 8]]

AB = [[1*5 + 2*7, 1*6 + 2*8], [3*5 + 4*7, 3*6 + 4*8]]

= [[19, 22], [43, 50]]

Note: Matrix multiplication is not commutative, i.e., AB ≠ BA.

**3. Eigenvalues and Eigenvectors**

Definition: An eigenvector v of a square matrix A satisfies Av = λv, where λ is the eigenvalue.

Example:
A = [[4, 2], [1, 3]]

Characteristic equation: det(A - λI) = 0

(4-λ)(3-λ) - 2*1 = 0 → $\lambda^2$ - 7λ + 10 = 0 → λ = 5, 2

Eigenvalues: 5 and 2
For λ=5, eigenvector ∝ [2, 1]
For λ=2, eigenvector ∝ [-1, 1]

**4. U–V (SVD) Decomposition**

Singular Value Decomposition (SVD) generalizes eigen-decomposition:

$A = U \Sigma V^T$

• U → left singular vectors (eigenvectors of $AA^T$)
• V → right singular vectors (eigenvectors of $A^TA$)

• $\Sigma$ → diagonal matrix of singular values (√ eigenvalues of $A^T A$)

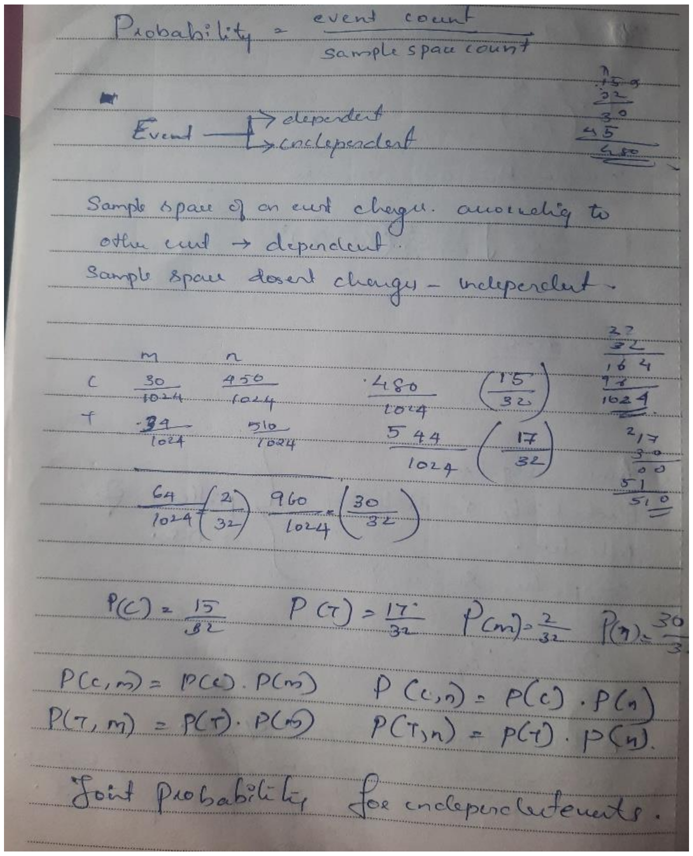
Applications: PCA, dimensionality reduction, recommender systems, NLP.

Statistics

- Percentile

  pthpercentile=P×(n+1)100
- Quartiles (Q1, Q2, Q3)
- Interquartile range (IQR = Q3 - Q1)
- Min
- Max

Lower bound= Q1-1.5*IQR)



$$\text{Probability} = \frac{\text{event count}}{\text{sample space count}}$$

15
32
30
45
480

$$\text{Event} \longrightarrow \begin{cases} \rightarrow \text{dependent} \\ \rightarrow \text{independent} \end{cases}$$

Sample space of on event changes according to other event → dependent.
Sample space dosent changes – independent.

| | m | n | | | |
|---|---|---|---|---|---|
| C | $\frac{30}{1024}$ | $\frac{450}{1024}$ | $\frac{480}{1024}$ | $\left(\frac{15}{32}\right)$ | |
| T | $\frac{34}{1024}$ | $\frac{510}{1024}$ | $\frac{544}{1024}$ | $\left(\frac{17}{32}\right)$ | |
| | $\frac{64}{1024}\left(\frac{2}{32}\right)$ | $\frac{960}{1024} = \left(\frac{30}{32}\right)$ | | | |

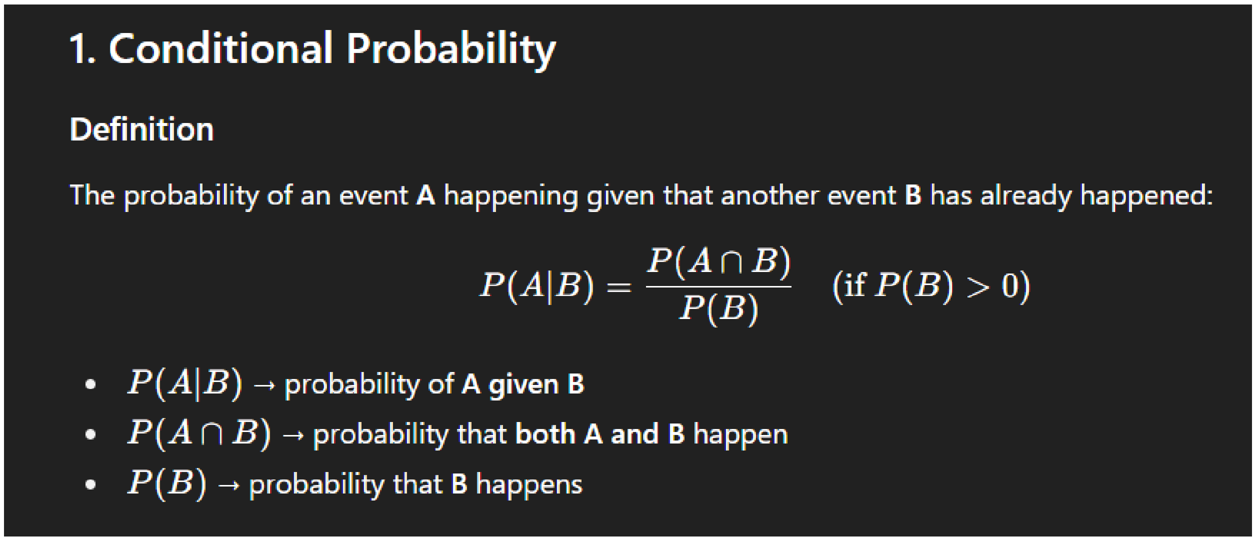
32
32
164
96
1024

17
30
00
51
510

$P(C) = \frac{15}{32}$ $P(T) = \frac{17}{32}$ $P(m) = \frac{2}{32}$ $P(n) = \frac{30}{3}$

$P(C, m) = P(C) \cdot P(m)$ $P(C, n) = P(C) \cdot P(n)$
$P(T, m) = P(T) \cdot P(m)$ $P(T, n) = P(T) \cdot P(n)$.

Joint probability for independent events.



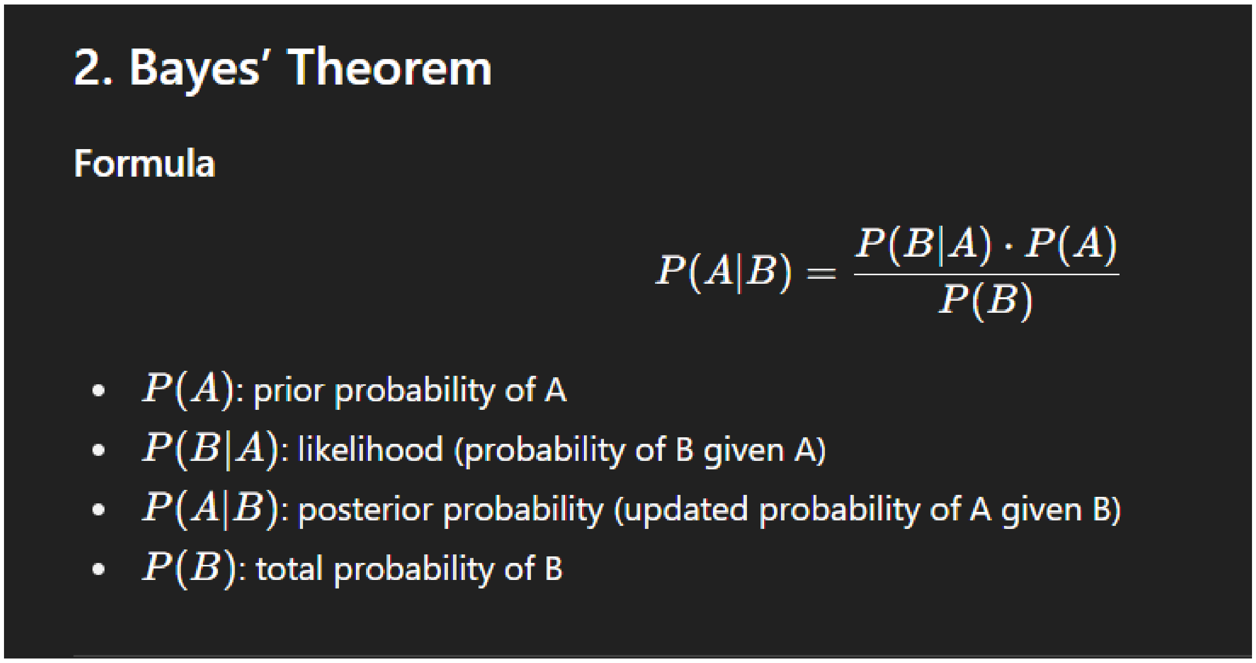
## 1. Conditional Probability

### Definition

The probability of an event **A** happening given that another event **B** has already happened:

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

- $P(A|B)$ → probability of **A given B**
- $P(A \cap B)$ → probability that **both A and B** happen
- $P(B)$ → probability that **B** happens



## 2. Bayes' Theorem

### Formula

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

- $P(A)$: prior probability of A
- $P(B|A)$: likelihood (probability of B given A)
- $P(A|B)$: posterior probability (updated probability of A given B)
- $P(B)$: total probability of B

**Z-test**

A Z-test is used to compare means when the sample size is large n>30 and the population variance is known. It follows the standard normal distribution (Z-distribution). The steps to do this as follows

1. State the Null Hypothesis

2. State the Alternate Hypothesis
3. Choose Your Significance level ($\alpha$)
4. Calculate Your Z-test Statistic

   $Z=\bar{x}-\mu(\sigma n)$

- $\bar{x}$ = sample mean
- $\mu$ = population mean
- $\sigma$ = population standard deviation
- $n$ = sample size

5. Find p-value using Z-Table and Z-test statistic computed
6. if p-value > $\alpha$, then we fail to reject null Hypothesis
7. There are one-tailed and two-tailed tests

**One Sample T-test**

Compares the sample mean to a known population mean

The steps are as follows

1. State the Null Hypothesis
2. State the Alternate Hypothesis
3. Choose Your Significance level ($\alpha$)
4. Calculate Your T-test Statistic

   $t=\bar{x}-\mu(sn)$

- $\bar{x}$ = sample mean
- $\mu$ = population mean
- $s$ = sample standard deviation

s=1n−1∑i=1N(xi−x—)2

- n = sample size

5. Find the critical t-value from T-Table using significance (α) level
6. if t-statistic < critical t-value, we fail to reject Null Hypothesis

**Independent Sample T-test**

Compares means of two independent groups

The steps are as follows

1. State the Null Hypothesis
2. State the Alternate Hypothesis
3. Choose Your Significance level (α)
4. Calculate Your T-test Statistic

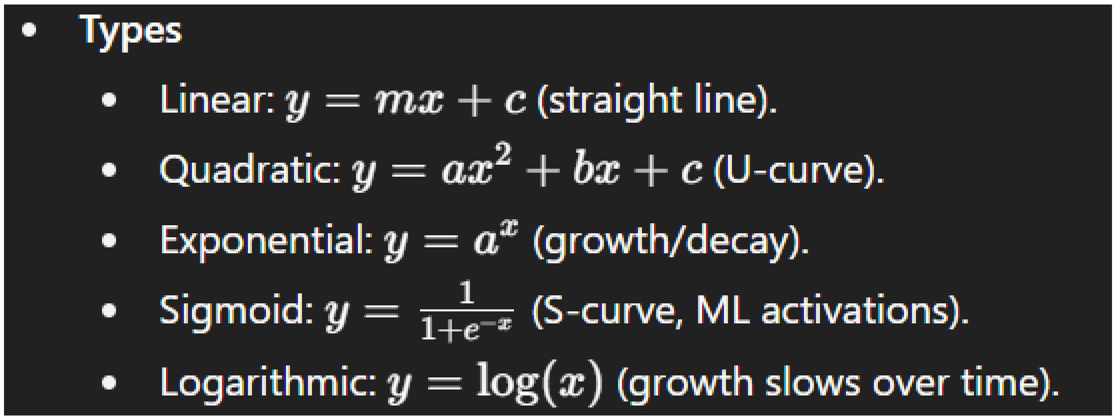
- t=x1—−x2—(s12n1+s22n2); for one tailed
- t=x1—−x2—Sp(1n1+1n2); for two tailed
  - Sp=(n1−1)s12+(n2−1)s22n1+n2−2; known as pooled standard deviation
- x1¯ = sample mean of group 1
- x2¯ = sample mean of group 2
- s1 = sample standard deviation of group 1
- s2 = sample standard deviation of group 2
- n1 = sample size of group 1
- n2 = sample size of group 2

**What is a function?**

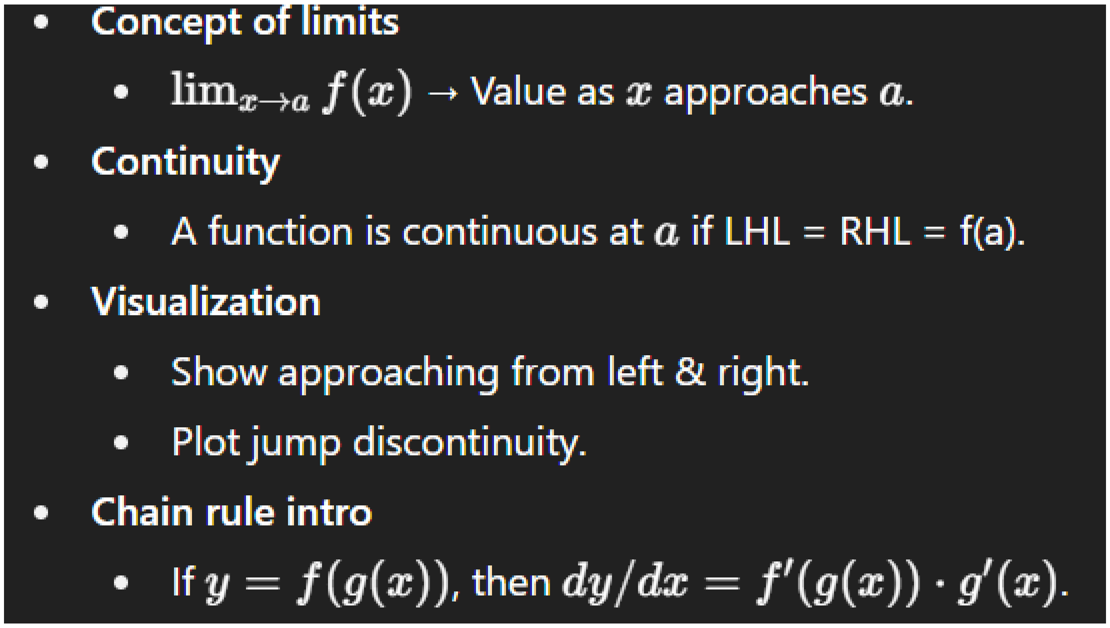
- A rule mapping inputs → outputs.
- f(x)=f(x) = f(x)=y → Input xxx gives output y.

**Independent vs Dependent variables**

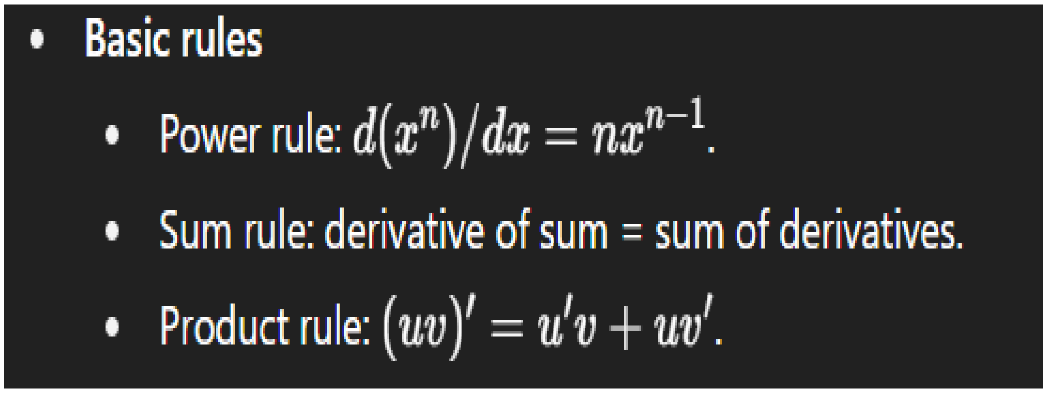
- Independent: Input we control , Dependent: Output/result



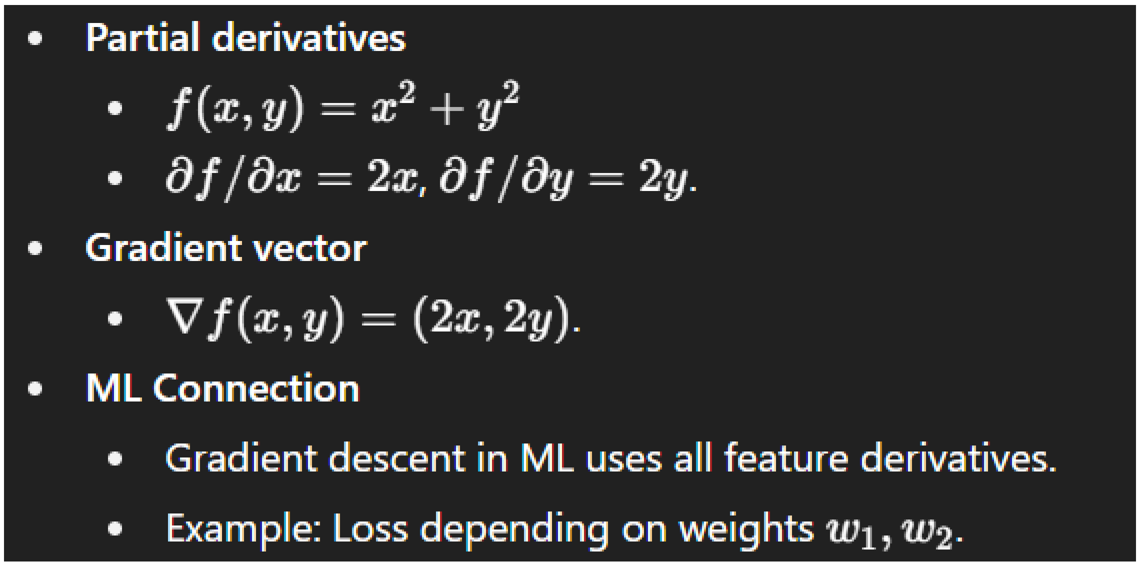
- **Types**
  - Linear: $y = mx + c$ (straight line).
  - Quadratic: $y = ax^2 + bx + c$ (U-curve).
  - Exponential: $y = a^x$ (growth/decay).
  - Sigmoid: $y = \frac{1}{1+e^{-x}}$ (S-curve, ML activations).
  - Logarithmic: $y = \log(x)$ (growth slows over time).



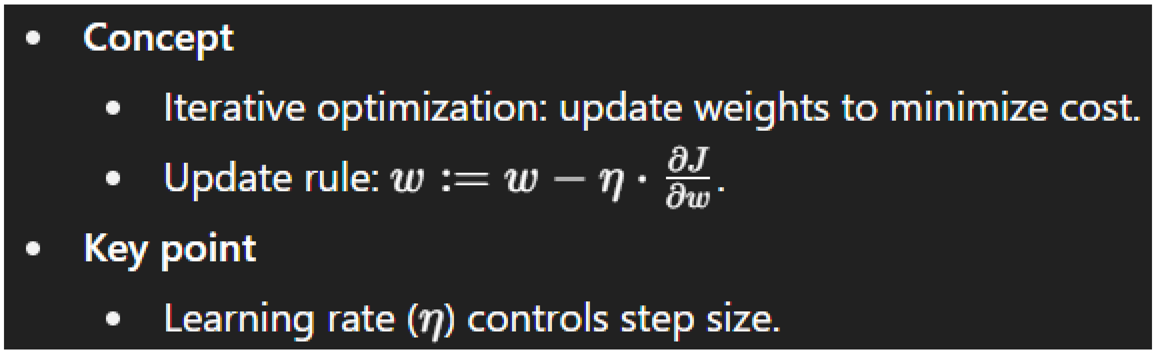
- **Concept of limits**
  - $\lim_{x \to a} f(x)$ → Value as $x$ approaches $a$.
- **Continuity**
  - A function is continuous at $a$ if LHL = RHL = f(a).
- **Visualization**
  - Show approaching from left & right.
  - Plot jump discontinuity.
- **Chain rule intro**
  - If $y = f(g(x))$, then $dy/dx = f'(g(x)) \cdot g'(x)$.



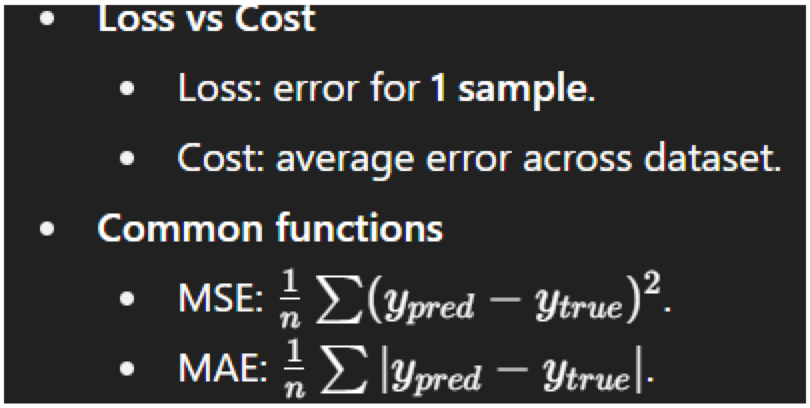
- Basic rules
  - Power rule: $d(x^n)/dx = nx^{n-1}$.
  - Sum rule: derivative of sum = sum of derivatives.
  - Product rule: $(uv)' = u'v + uv'$.



- **Partial derivatives**
  - $f(x, y) = x^2 + y^2$
  - $\partial f/\partial x = 2x, \partial f/\partial y = 2y$.
- **Gradient vector**
  - $\nabla f(x, y) = (2x, 2y)$.
- **ML Connection**
  - Gradient descent in ML uses all feature derivatives.
  - Example: Loss depending on weights $w_1, w_2$.



- **Concept**
  - Iterative optimization: update weights to minimize cost.
  - Update rule: $w := w - \eta \cdot \frac{\partial J}{\partial w}$.
- **Key point**
  - Learning rate ($\eta$) controls step size.



- **Loss vs Cost**
  - Loss: error for **1 sample**.
  - Cost: average error across dataset.
- **Common functions**
  - MSE: $\frac{1}{n}\sum(y_{pred} - y_{true})^2$.
  - MAE: $\frac{1}{n}\sum|y_{pred} - y_{true}|$.

NumPy – Numerical python

NumPy is a Python library used for working with arrays. It also has functions for working in domain of linear algebra, fourier transform, and matrices

Used for applications of machine learning and deep learning

Import numpy as np

Numpy arrays are ndarray N – dimensional array

Numpy element wise operations

Data manipulation analysis

ML