MIDTERM CHEAT SHEET

Daniel Palma

February 26, 2025

1 Linear Algebra

1.1 Definitions of Matrices and Vectors

A matrix is a rectangular array of numbers arranged in rows and columns, while a vector is a special case of a matrix with either one row (row vector) or one column (column vector).

Notation: We typically denote matrices with uppercase letters (A, B, C) and vectors with lowercase bold letters ($\mathbf{v}$, $\mathbf{x}$, $\mathbf{y}$).

Dimensions: An $m \times n$ matrix has m rows and n columns.

Example expanded:

$$A = \begin{bmatrix} 1 & 2 \\ 3 & 4 \end{bmatrix}, \quad \text{Row vector: } \mathbf{a} = \begin{bmatrix} 1 & 2 & 3 \end{bmatrix}, \quad \text{Column vector: } \mathbf{b} = \begin{bmatrix} 4 \\ 5 \\ 6 \end{bmatrix} \tag{1}$$

1.2 Addition, Subtraction, Multiplication

1.2.1 Matrix Addition and Subtraction

Two matrices can be added or subtracted if they have the same dimensions. The operation is performed element-wise.

For matrices A and B of the same dimensions:

- Addition: $(A + B)_{ij} = A_{ij} + B_{ij}$

- Subtraction: $(A - B)_{ij} = A_{ij} - B_{ij}$

Example:

$$A = \begin{bmatrix} 1 & 2 \\ 3 & 4 \end{bmatrix}, \quad B = \begin{bmatrix} 5 & 6 \\ 7 & 8 \end{bmatrix}, \quad A - B = \begin{bmatrix} -4 & -4 \\ -4 & -4 \end{bmatrix} \tag{2}$$

1.2.2 Scalar Multiplication

When multiplying a matrix by a scalar (a single number), each element of the matrix is multiplied by that scalar.

For a scalar c and matrix A:

- Scalar multiplication: $(cA)_{ij} = c \cdot A_{ij}$

Example:

$$A = \begin{bmatrix} 1 & 2 \\ 3 & 4 \end{bmatrix}, \quad 3A = \begin{bmatrix} 3 & 6 \\ 9 & 12 \end{bmatrix} \tag{3}$$

1.2.3 Matrix Multiplication

Matrix multiplication is more complex. For two matrices A and B to be multiplied (in the order $A \times B$), the number of columns in A must equal the number of rows in B.

If A is an $m \times n$ matrix and B is an $n \times p$ matrix, then $C = A \times B$ is an $m \times p$ matrix.

The (i, j) element of C is:

$$C_{ij} = \sum_{k=1}^{n} A_{ik} \cdot B_{kj} \tag{4}$$

Example with steps: For $A = \begin{bmatrix} 1 & 2 \\ 3 & 4 \end{bmatrix}$ and $B = \begin{bmatrix} 5 & 6 \\ 7 & 8 \end{bmatrix}$, the multiplication $A \times B$ is:

$$C_{11} = A_{11} \cdot B_{11} + A_{12} \cdot B_{21} = 1 \cdot 5 + 2 \cdot 7 = 5 + 14 = 19 \tag{5}$$
$$C_{12} = A_{11} \cdot B_{12} + A_{12} \cdot B_{22} = 1 \cdot 6 + 2 \cdot 8 = 6 + 16 = 22 \tag{6}$$
$$C_{21} = A_{21} \cdot B_{11} + A_{22} \cdot B_{21} = 3 \cdot 5 + 4 \cdot 7 = 15 + 28 = 43 \tag{7}$$
$$C_{22} = A_{21} \cdot B_{12} + A_{22} \cdot B_{22} = 3 \cdot 6 + 4 \cdot 8 = 18 + 32 = 50 \tag{8}$$

So $A \times B = \begin{bmatrix} 19 & 22 \\ 43 & 50 \end{bmatrix}$

1.3 Diagonal and Identity Matrices

A **diagonal matrix** has non-zero elements only on its main diagonal (from top-left to bottom-right).

A **identity matrix** (denoted I) is a special diagonal matrix with 1s on the main diagonal and 0s elsewhere.

Properties:

- For any matrix A, $A \cdot I = I \cdot A = A$ (identity property)
- Diagonal matrices commute under multiplication: if D_1 and D_2 are diagonal, then $D_1 \cdot D_2 = D_2 \cdot D_1$

Example of 3×3 diagonal and identity matrices:

$$D = \begin{bmatrix} 4 & 0 & 0 \\ 0 & 5 & 0 \\ 0 & 0 & 6 \end{bmatrix}, \quad I = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix} \tag{9}$$

1.4 Determinant and Eigenstructure

1.4.1 Determinant

The determinant is a scalar value associated with a square matrix that has important geometric and algebraic interpretations.

Properties of determinants:

- $\det(AB) = \det(A) \cdot \det(B)$
- $\det(A^{-1}) = \frac{1}{\det(A)}$
- $\det(A^T) = \det(A)$

Calculation for 3×3 matrix: For $A = \begin{bmatrix} a & b & c \\ d & e & f \\ g & h & i \end{bmatrix}$, the determinant is:

$$\det(A) = a(ei - fh) - b(di - fg) + c(dh - eg) \tag{10}$$

Example:

$$A = \begin{bmatrix} 2 & 0 & 1 \\ 3 & 1 & 2 \\ 1 & 0 & 3 \end{bmatrix} \tag{11}$$

$$\begin{aligned} \det(A) &= 2(1 \cdot 3 - 2 \cdot 0) - 0(3 \cdot 3 - 2 \cdot 1) + 1(3 \cdot 0 - 1 \cdot 1) && (12) \\ &= 2 \cdot 3 - 0 - 1 = 5 && (13) \end{aligned}$$

1.4.2 Eigenvalues and Eigenvectors

An eigenvector $\mathbf{v}$ of a square matrix A is a non-zero vector that, when multiplied by A, yields a scalar multiple of itself: $A\mathbf{v} = \lambda\mathbf{v}$, where λ is the corresponding eigenvalue.

Finding eigenvalues:

1. Set up the characteristic equation: $\det(A - \lambda I) = 0$
2. Solve for λ

Finding eigenvectors: For each eigenvalue λ, solve the homogeneous system: $(A - \lambda I)\mathbf{v} = \mathbf{0}$

Example: For $A = \begin{bmatrix} 3 & 1 \\ 1 & 3 \end{bmatrix}$

1. Characteristic equation:

$$\det\left(\begin{bmatrix} 3-\lambda & 1 \\ 1 & 3-\lambda \end{bmatrix}\right) = (3-\lambda)(3-\lambda) - 1 \cdot 1 = (3-\lambda)^2 - 1 = 0 \tag{14}$$

   Solving: $(3-\lambda)^2 = 1 \implies 3 - \lambda = \pm 1 \implies \lambda = 2$ or $\lambda = 4$

2. For $\lambda = 2$:

$$(A - 2I)\mathbf{v} = \begin{bmatrix} 1 & 1 \\ 1 & 1 \end{bmatrix} \mathbf{v} = \mathbf{0} \tag{15}$$

   This gives $v_1 = -v_2$, so an eigenvector is $\begin{bmatrix} 1 \\ -1 \end{bmatrix}$

3. For $\lambda = 4$:

$$(A - 4I)\mathbf{v} = \begin{bmatrix} -1 & 1 \\ 1 & -1 \end{bmatrix} \mathbf{v} = \mathbf{0} \tag{16}$$

   This gives $v_1 = v_2$, so an eigenvector is $\begin{bmatrix} 1 \\ 1 \end{bmatrix}$

1.5 Inverses and Singularity

A square matrix A is **invertible** if there exists a matrix A^{-1} such that $A \cdot A^{-1} = A^{-1} \cdot A = I$.

Conditions for invertibility:

- A is invertible if and only if $\det(A) \neq 0$
- If A is not invertible, it is called **singular**

Calculating the inverse of a 2×2 matrix: For $A = \begin{bmatrix} a & b \\ c & d \end{bmatrix}$:

$$A^{-1} = \frac{1}{\det(A)} \begin{bmatrix} d & -b \\ -c & a \end{bmatrix} = \frac{1}{ad - bc} \begin{bmatrix} d & -b \\ -c & a \end{bmatrix} \tag{17}$$

Properties:

- $(AB)^{-1} = B^{-1}A^{-1}$
- $(A^{-1})^{-1} = A$
- $(A^T)^{-1} = (A^{-1})^T$

Example with a 3×3 matrix: For $A = \begin{bmatrix} 1 & 0 & 2 \\ 0 & 3 & 1 \\ 1 & 1 & 0 \end{bmatrix}$

You can use Gaussian elimination or the formula involving the adjugate to find:

$$A^{-1} = \begin{bmatrix} -\frac{2}{5} & \frac{2}{5} & \frac{3}{5} \\ \frac{1}{5} & \frac{1}{5} & -\frac{1}{5} \\ \frac{3}{5} & -\frac{1}{5} & -\frac{1}{5} \end{bmatrix} \tag{18}$$

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

1.6 Systems of Equations

A system of linear equations can be written in matrix form as $A\mathbf{x} = \mathbf{b}$.

Solving methods:

1. **Direct solution (if A is invertible): $\mathbf{x} = A^{-1}\mathbf{b}$**
2. **Gaussian elimination:** Transform the augmented matrix $[A|\mathbf{b}]$ into row echelon form
3. **Cramer's rule:** $x_j = \frac{\det(A_j)}{\det(A)}$, where A_j is A with column j replaced by $\mathbf{b}$

Example with solution: For the system:

$$2x + y - z = 8 \tag{19}$$
$$-3x + y + 2z = -2 \tag{20}$$
$$x + y + z = 3 \tag{21}$$

Matrix form:

$$\begin{bmatrix} 2 & 1 & -1 \\ -3 & 1 & 2 \\ 1 & 1 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ z \end{bmatrix} = \begin{bmatrix} 8 \\ -2 \\ 3 \end{bmatrix} \tag{22}$$

Using Gaussian elimination:

$$\left[\begin{array}{ccc|c} 2 & 1 & -1 & 8 \\ -3 & 1 & 2 & -2 \\ 1 & 1 & 1 & 3 \end{array}\right] \to \left[\begin{array}{ccc|c} 1 & 0 & 0 & 2 \\ 0 & 1 & 0 & 1 \\ 0 & 0 & 1 & 0 \end{array}\right] \tag{23}$$

So the solution is $x = 2$, $y = 1$, $z = 0$.

1.7 Singular Value Decomposition (SVD)

SVD states that any $m \times n$ matrix A can be decomposed as:

$$A = U\Sigma V^T \tag{24}$$

Where:

- U is an $m \times m$ orthogonal matrix ($UU^T = I$)
- Σ is an $m \times n$ diagonal matrix with non-negative entries (singular values)
- V is an $n \times n$ orthogonal matrix ($VV^T = I$)

Applications:

- Least squares approximation
- Image compression
- Principal Component Analysis
- Pseudoinverse calculation: $A^+ = V\Sigma^+ U^T$, where Σ^+ is obtained by taking the reciprocal of non-zero singular values

Example: For $A = \begin{bmatrix} 3 & 2 \\ 2 & 3 \end{bmatrix}$, the SVD is:

$$U = \begin{bmatrix} \frac{1}{\sqrt{2}} & \frac{1}{\sqrt{2}} \\ \frac{1}{\sqrt{2}} & -\frac{1}{\sqrt{2}} \end{bmatrix}, \quad \Sigma = \begin{bmatrix} 5 & 0 \\ 0 & 1 \end{bmatrix}, \quad V = \begin{bmatrix} \frac{1}{\sqrt{2}} & \frac{1}{\sqrt{2}} \\ \frac{1}{\sqrt{2}} & -\frac{1}{\sqrt{2}} \end{bmatrix} \tag{25}$$

1.8 Spectral Decomposition

For a square matrix A with n linearly independent eigenvectors, the spectral decomposition is:

$$A = PDP^{-1} \tag{26}$$

Where:

- P is the matrix of eigenvectors
- D is a diagonal matrix of eigenvalues

Applications:

- Computing matrix powers: $A^k = PD^kP^{-1}$
- Solving differential equations
- Analyzing dynamical systems

Example: For $A = \begin{bmatrix} 3 & 1 \\ 1 & 3 \end{bmatrix}$ from our eigenvalue example:

$$P = \begin{bmatrix} 1 & 1 \\ -1 & 1 \end{bmatrix}, \quad D = \begin{bmatrix} 2 & 0 \\ 0 & 4 \end{bmatrix}, \quad P^{-1} = \begin{bmatrix} \frac{1}{2} & -\frac{1}{2} \\ -\frac{1}{2} & \frac{1}{2} \end{bmatrix} \tag{27}$$

1.9 Properties and Derivations of Matrix Traces

The trace of a square matrix A, denoted $\text{tr}(A)$, is the sum of its diagonal elements.

Properties:

- $\text{tr}(A + B) = \text{tr}(A) + \text{tr}(B)$
- $\text{tr}(cA) = c \cdot \text{tr}(A)$ for scalar c

- $\mathrm{tr}(AB) = \mathrm{tr}(BA)$, even when A and B have different dimensions (as long as both products are defined)
- $\mathrm{tr}(A) =$ sum of eigenvalues of A
- For orthogonal matrices Q, $\mathrm{tr}(Q^T AQ) = \mathrm{tr}(A)$

Applications:

- In statistics: $\mathrm{tr}(\Sigma\Sigma^T)$ represents the total variance in multivariate analysis
- In optimization: many objective functions involve traces
- In physics: traces appear in quantum mechanics

Example with matrix products:

$$A = \begin{bmatrix} 1 & 2 \\ 3 & 4 \end{bmatrix}, \quad B = \begin{bmatrix} 5 & 6 \\ 7 & 8 \end{bmatrix} \tag{28}$$

$$\mathrm{tr}(A) = 1 + 4 = 5 \tag{29}$$

$$\mathrm{tr}(B) = 5 + 8 = 13 \tag{30}$$

$$\mathrm{tr}(AB) = \mathrm{tr}\left(\begin{bmatrix} 19 & 22 \\ 43 & 50 \end{bmatrix}\right) = 19 + 50 = 69 \tag{31}$$

Also:

$$BA = \begin{bmatrix} 5 & 6 \\ 7 & 8 \end{bmatrix} \begin{bmatrix} 1 & 2 \\ 3 & 4 \end{bmatrix} = \begin{bmatrix} 23 & 34 \\ 31 & 46 \end{bmatrix} \tag{32}$$

$$\mathrm{tr}(BA) = 23 + 46 = 69 = \mathrm{tr}(AB) \tag{33}$$

This demonstrates the property that $\mathrm{tr}(AB) = \mathrm{tr}(BA)$.

2 Machine Learning

2.1 Framework for Classical Machine Learning

- **Definition**: Classical machine learning involves training models on labeled data to make predictions or decisions without being explicitly programmed. These algorithms identify patterns in data through statistical techniques and use these patterns to make predictions on new, unseen data. The learning process often involves optimizing a mathematical function that measures how well the model performs.
- **Key Components**:
  - Data: Labeled examples used for training and testing
  - Features: Measurable properties or characteristics extracted from data
  - Algorithm: The method used to learn patterns from the data
  - Model: The representation learned from the data
  - Evaluation metrics: Methods to assess model performance (accuracy, precision, recall, F1 score)
- **Learning Paradigms**:
  - Batch learning: Model is trained on the entire dataset at once
  - Online learning: Model is updated incrementally as new data arrives
- **Examples**:
  - Linear regression for predicting continuous values
  - Decision trees for classification and regression

- Support vector machines for classification using optimal hyperplanes
  - Random forests for ensemble learning
  - k-nearest neighbors for instance-based learning
  - Naive Bayes for probabilistic classification

- **The Curse of Dimensionality**:
  - Definition: Refers to various phenomena that arise when analyzing data in high-dimensional spaces that do not occur in lower-dimensional settings
  - Key challenges:
    - Exponential growth in volume: As dimensionality increases, the volume of the space increases exponentially, making data increasingly sparse
    - Distance concentration: In high-dimensional spaces, distances between points become more similar, reducing the effectiveness of distance-based algorithms
    - Sample size requirements: The amount of data needed to achieve statistical significance grows exponentially with dimensions
    - Computational complexity: Processing high-dimensional data requires significantly more computational resources
  - Practical implications:
    - Models require exponentially more training data as dimensions increase
    - Feature selection and dimensionality reduction become essential
    - Models more prone to overfitting in high-dimensional spaces
    - Intuition from low-dimensional spaces often fails in high dimensions

2.2 Framework for Deep Learning

- **Definition**: Deep learning uses neural networks with multiple layers to learn complex patterns in data. It is a subset of machine learning that focuses on learning hierarchical representations directly from raw data, eliminating the need for manual feature engineering. The "deep" refers to the use of multiple layers in the neural network.

- **Key Components**:
  - Neural network architecture: Organization of neurons and connections
  - Activation functions: Non-linear transformations applied to neuron outputs (ReLU, sigmoid, tanh)
  - Loss functions: Measures how well the model performs (cross-entropy, mean squared error)
  - Optimizers: Algorithms that update model parameters (Adam, SGD, RMSprop)
  - Regularization techniques: Methods to prevent overfitting (dropout, batch normalization)

- **Types of Architectures**:
  - Feedforward Neural Networks: Basic architecture where information flows in one direction
  - Convolutional Neural Networks (CNNs): Specialized for grid-like data such as images
  - Recurrent Neural Networks (RNNs): For sequential data with LSTM/GRU variants
  - Transformers: Self-attention based models for sequence processing
  - Generative Adversarial Networks (GANs): For generating new data samples
  - Autoencoders: For unsupervised feature learning and dimensionality reduction

- **Applications**:
  - Computer vision (object detection, image recognition)
  - Natural language processing (translation, sentiment analysis)
  - Speech recognition and synthesis

- Game playing (AlphaGo, chess)
- Drug discovery and molecular design

- **Overfitting in Deep Learning**:
  - Definition: Overfitting occurs when a model learns the training data too well, capturing noise and specific details rather than general patterns, resulting in poor generalization to new data.
  - Characteristics of overfitting:
    - High accuracy on training data but poor performance on test data
    - Complex model with many parameters relative to the amount of training data
    - Model captures random fluctuations in the training data
  - Solutions to overfitting:
    - Regularization techniques: L1/L2 regularization to penalize large weights
    - Dropout: Randomly disabling neurons during training
    - Early stopping: Halting training when validation performance starts to degrade
    - Data augmentation: Creating variations of training examples
    - Batch normalization: Normalizing inputs to each layer
    - Transfer learning: Using pre-trained models as starting points

2.3 Supervised Learning Formulations

- **Definition**: Supervised learning involves learning a mapping function from input variables (X) to output variables (Y) using labeled training data. The goal is to approximate this mapping function so well that when given new input data, the model can predict the output variables.
- **Mathematical Formulation**:
  - Input space: X (features)
  - Output space: Y (labels/targets)
  - Training data: $D = \{(x_1, y_1), (x_2, y_2), ..., (x_n, y_n)\}$
  - Hypothesis space: H (set of possible mapping functions)
  - Goal: Find h $\in$ H such that h: X $\rightarrow$ Y minimizes the expected loss
- **Loss Functions**:
  - Regression: Mean Squared Error (MSE), Mean Absolute Error (MAE)
  - Classification: Cross-Entropy Loss, Hinge Loss, Log Loss
- **Types of Supervised Learning**:
  - Classification: Predicting discrete class labels
    - Binary classification (spam detection, disease diagnosis)
    - Multi-class classification (digit recognition, species identification)
    - Multi-label classification (tagging images with multiple concepts)
  - Regression: Predicting continuous values
    - Simple linear regression (one feature)
    - Multiple linear regression (multiple features)
    - Polynomial regression (non-linear relationships)
    - Support vector regression (using kernel methods)
- **Examples**:
  - Predicting house prices based on features like size, location, age
  - Email spam detection
  - Medical diagnosis

  - Credit risk assessment
  - Image recognition

- **Stochastic Gradient Descent (SGD)**:
  - Definition: SGD is an optimization algorithm used to find the parameters that minimize the loss function in supervised learning.
  - Mathematical Definition:
    - For a loss function L(θ) with parameters θ:
      1. Initialize parameters θ_0
      2. For each iteration t = 1, 2, ..., T:
         - Randomly select a mini-batch of m samples
         - Compute gradient: $g_t = \nabla_\theta L(\theta_{t-1}; x_i, y_i)$ for samples in mini-batch
         - Update parameters: $\theta_t = \theta_{t-1} - \eta_t \cdot g_t$
         - where η_t is the learning rate at iteration t
  - Formula:
    - $\theta_t = \theta_{t-1} - \eta_t \cdot \nabla_\theta L(\theta_{t-1}; x_i, y_i)$
  - Key characteristics:
    - Uses a subset of training examples (mini-batch) for each update
    - More computationally efficient than batch gradient descent
    - Introduces noise in the gradient, which can help escape local minima
    - Requires tuning of hyperparameters like learning rate and batch size
    - Variants include momentum SGD, Adam, RMSprop, and AdaGrad
  - Advantages:
    - Faster convergence for large datasets
    - Lower memory requirements
    - May provide better generalization due to noise in updates
    - More likely to escape shallow local minima

2.4 Asymptotic Analysis

- **Definition**: Asymptotic analysis studies the behavior of algorithms and functions as the input size approaches infinity. It provides a mathematical framework for describing the efficiency of algorithms independent of hardware or implementation details.
- **Key Concepts**:
  - Big-O notation (O): Upper bound on growth rate
    - f(n) = O(g(n)) if there exist constants c ¿ 0 and n_0 such that 0 ≤ f(n) ≤ c·g(n) for all n ≥ n_0
  - Big-Omega notation (Ω): Lower bound on growth rate
    - f(n) = Ω(g(n)) if there exist constants c ¿ 0 and n_0 such that 0 ≤ c·g(n) ≤ f(n) for all n ≥ n_0
  - Big-Theta notation (Θ): Tight bound on growth rate
    - f(n) = Θ(g(n)) if f(n) = O(g(n)) and f(n) = Ω(g(n))
- **Common Complexities**:
  - O(1): Constant time (hash table lookups)
  - O(log n): Logarithmic (binary search)
  - O(n): Linear (linear search)
  - O(n log n): Log-linear (efficient sorting algorithms)
  - O(n^2): Quadratic (simple sorting algorithms)

  - $O(2^n)$: Exponential (brute force algorithms)
- **Applications in Machine Learning**:
  - Analyzing time complexity of training algorithms
  - Evaluating scalability with increasing data dimensions
  - Determining computational feasibility for large datasets
  - Comparing efficiency of different learning algorithms
- **Examples**:
  - k-nearest neighbors has O(nd) query time complexity, where n is the number of samples and d is the number of dimensions
  - Decision tree induction has O(d n log n) complexity for n samples with d dimensions
  - Neural network training with SGD has O(n d w) complexity per epoch, where n is the number of samples, d is the dimension, and w is the number of weights

2.5 Non-asymptotic Analysis

- **Definition**: Non-asymptotic analysis focuses on the performance of algorithms for finite input sizes, providing explicit bounds that hold for any sample size rather than just asymptotic behavior.
- **Key Concepts**:
  - Concentration inequalities: Bounds on how a random variable deviates from its expected value
  - High-probability bounds: Guarantees that hold with high probability but not necessarily always
  - Uniform convergence: Convergence that applies simultaneously to all functions in a class
  - Generalization error bounds: Limits on the difference between training and test performance
- **Important Tools**:
  - Hoeffding's inequality: Bounds the probability that the sum of bounded random variables deviates from its expected value
  - Markov's inequality: Provides bounds for non-negative random variables
  - Chebyshev's inequality: Relates variance to probability of deviation from the mean
  - McDiarmid's inequality: Bounds for functions of independent random variables
  - VC dimension: Measure of complexity for hypothesis classes
- **Applications in Machine Learning**:
  - Deriving sample complexity: Number of samples needed to achieve a specific error rate
  - Bounding generalization error: Gap between training and test performance
  - Model selection: Choosing between different model complexities
  - Early stopping criteria: Deciding when to halt the training process
  - Regularization parameter selection: Determining optimal strength of regularization
- **Examples**:
  - For empirical risk minimization, the generalization error is bounded by $O(\sqrt{\frac{\log(1/\delta)}{n}})$ with probability 1-δ
  - VC dimension-based bounds show that the sample complexity scales linearly with the VC dimension
  - Rademacher complexity provides data-dependent bounds on the generalization error
  - PAC learning theory provides bounds on the sample complexity needed to learn a concept with high probability

3 Python Data Types

3.1 Overview of Python Data Types

- Python provides a variety of built-in data types, categorized into mutable and immutable types.
- Mutable types can be modified after creation, while immutable types cannot.

3.2 Immutable Data Types

- **Integers (int)**
  - Whole numbers, positive or negative.
  - Arbitrary-precision arithmetic.
  - Example: `x = 42`
- **Floating-Point Numbers (float)**
  - Decimal numbers with fractional parts.
  - Example: `y = 3.14`
- **Complex Numbers (complex)**
  - Numbers with real and imaginary parts.
  - Example: `z = 3 + 4j`
- **Strings (str)**
  - Sequence of Unicode characters.
  - Immutable.
  - Example: `text = "Hello"`
- **Tuples (tuple)**
  - Ordered collection of elements.
  - Immutable.
  - Example: `t = (1, 2, 3)`
- **Booleans (bool)**
  - Represents True or False values.
  - Example: `flag = True`

3.3 Mutable Data Types

- **Lists (list)**
  - Ordered collection of elements.
  - Mutable (can be modified after creation).
  - Example: `lst = [1, 2, 3]`
- **Dictionaries (dict)**
  - Key-value pairs.
  - Mutable.
  - Example: `d = {'a': 1, 'b': 2}`
- **Sets (set)**
  - Unordered collection of unique elements.
  - Mutable.
  - Example: `s = {1, 2, 3}`

3.4 Type Conversion

- Python supports explicit and implicit type conversion.
- Example:
  - Explicit: `float(5) # Converts integer 5 to float 5.0`
  - Implicit: `x = 5 + 3.2 # Integer 5 is converted to float`

3.5 Special Data Types

- **Bytes (bytes)**
  - Immutable sequence of bytes.
  - Example: `b = b’hello’`
- **Byte Arrays (bytearray)**
  - Mutable sequence of bytes.
  - Example: `ba = bytearray(5)`
- **NoneType (None)**
  - Represents absence of value.
  - Example: `x = None`