BIEM-BIEF

A.Y. 2024/2025

BLAB

HANDOUTS

COMPUTER SCIENCE

WRITTEN BY

TANCREDI SEQUI

TEACHING DIVISION

Computer Science

Reduced Exam

Tancredi Sequi– BIEM 16 – AY 2024-2025

Excel

Getting Started with Excel Basics

Terminology

- Dataset: Organized collection of data in table format
- Worksheet: A single page or tab within a workbook that contains a dataset
- Columns: Variables (Fields)
- Rows: Individual records

ERRORS

- ####### - The column isn’t wide enough to display all cell data.
- #NAME? - The text in the formula isn’t recognized.
- #VALUE! - There is an error with one or more formula arguments.
- #DIV/0 - The formula is trying to divide a value by 0.
- #REF! - The formula references a cell that no longer exists.

Cell References

- $ (Dollar Sign in Excel/Sheets): Locks a row, column, or both in a cell reference.
- $A$1 → Locks column A and row 1.
- A$1 → Locks row 1 only.
- $A1 → Locks column A only.

Selecting and Managing Cells

- Wrap Text in a Cell
    - Select the cell(s), then go to Home → Wrap Text.

- Merge Cells
    - Select the cells, then go to Home → Merge & Center.

- Name a Cell or Range
    - Select the cell(s), then click the Name Box in the Formula Bar, enter a name, press Enter.

Working with Formulas

- Reference Other Worksheets
  - To reference a different worksheet in a formula, add ! after the sheet name (Sheet2!A1).

- Display Worksheet Formulas
  - Go to Formulas → Formula Auditing → Show Formulas.

Formatting and Organizing Data

- Format a Cell Range as a Table
  - Select the cells. Go to Home → Styles group → Format as Table. Choose a table style.

- Sort Data
  - Select a cell in the column to sort.
    Go to Home → Sort & Filter and choose a sorting order or select Custom Sort.

- Filter Data
  - Click the filter arrow in a column header and uncheck items you want to hide.

Logical Functions

Logical Functions

- IF: Returns different values based on a condition.
    - IF(logical_test, [value_if_true], [value_if_false])
        - logical_test: The condition you want to test.
        - value_if_true: The value to return if the condition is true.
        - value_if_false: The value to return if the condition is false.
    - Example Case: If an item has more than 10 units in stock, it should be labeled as "High"; otherwise, it should be labeled as "Low".
        - IF(A1>10, "High", "Low")

- AND: Returns TRUE if all conditions are met.
    - AND(logical1, [logical2], ...)
        - logical1: The first condition you want to test.
        - [logical2]: Optional. Additional conditions to test.
    - Example Case: You are verifying whether an order qualifies for a discount. The order must have "Yes" in column A and a quantity greater than 50 in column B. If both conditions are met, it's "Valid"; otherwise, it's "Invalid".
        - =IF(AND(A1="Yes", B1>50), "Valid", "Invalid")

- OR: Returns TRUE if at least one condition is met.
    - OR(logical1, [logical2], ...)
        - logical1: The first condition you want to test.
        - logical2: Optional. Additional conditions to test. You can add multiple logical conditions.
    - Example Case: You are checking a product. If column A contains "Red" or column B contains "Blue", label it as "Match"; otherwise, it should be labeled as "No Match".
        - =IF(OR(A1="Red", B1="Blue"), "Match", "No Match")

- IFS: Checks multiple conditions and returns the first true result.
    - IFS(logical_test1, value_if_true1, logical_test2, value_if_true2, ...)
        - logical_test1: The first condition to test.
        - value_if_true1: The result to return if the first condition is true.
        - logical_test2: The second condition to test.
        - value_if_true2: The result to return if the second condition is true.
        - You can add more pairs of conditions and results as needed.
    - Example Case: You are converting scores. If a score is of 90 or higher, it gets an "A". If it's 80 or higher, it gets a "B". If it's 70 or higher, it gets a "C".
        - IFS(A1>=90, "A", A1>=80, "B", A1>=70, "C")

Mathematical & Statistical Functions

Math & Statistical Functions

- SUMPRODUCT: Multiplies and sums arrays.
  - SUMPRODUCT(array1, [array2], ...)
    - Array 1: The first array or range to multiply.
    - Array 2: The second array or range to multiply.
  - Remember that arrays must be of equal lengths.
  - Example Case: You are calculating the total cost of items in an order. Column A contains the quantity of each item, and column B contains the price per unit. The formula multiplies each quantity by its corresponding price and sums the results.
    - =SUMPRODUCT(A1:A5, B1:B5)

- RANK.EQ: Ranks a value within a list.
  - =RANK.EQ(number, ref, [order])
    - number: The number you want to rank.
    - ref: The range of numbers that you want to rank against.
    - [order]: Determines whether to rank in ascending (1) or descending (0) order.
  - Example Usage: If you want to rank the value in cell A1 against all values in the range A1:A10 in ascending order:
    - =RANK.EQ(A1, A1:A10, 1)

- COUNTIF: Counts cells that meet a condition.
  - =COUNTIF(range, criteria)
    - range: The range of cells to check for the condition.
    - criteria: The condition to count (can be a number, text, or other criteria).
  - Example Usage: To count how many values in the range A1:A10 are greater than 50:
    - =COUNTIF(A1:A10, ">50")

- COUNTIFS: Counts cells based on multiple conditions.
  - =COUNTIFS(range1, criteria1, [range2], [criteria2], ...)
    - range1: The first range of cells to check for the condition.
    - criteria1: The condition for the first range.
    - [range2]: Optional. The second range of cells to check for another condition.
    - [criteria2]: Optional. The condition for the second range.
  - To count how many times the values in the range A1:A10 are greater than 50, and the corresponding values in the range B1:B10 are less than 100:
    - =COUNTIFS(A1:A10, ">50", B1:B10, "<100")

- SUMIF: Sums values based on a condition.
  - =SUMIF(range, criteria, [sum_range])
    - range: The range of cells to check for the condition.
    - criteria: The condition to sum based on (can be a number, text, or other criteria).
    - [sum_range]: The range of cells to sum.
  - Example Usage: To sum the values in the range B1:B10 where the corresponding values in the range A1:A10 are greater than 50:
    - =SUMIF(A1:A10, ">50", B1:B10)

- SUMIFS: Sums values with multiple conditions.
  - =SUMIFS(sum_range, range1, criteria1, [range2], [criteria2], ...)
    - sum_range: The range of cells to sum.
    - range1: The first range of cells to check for the condition.
    - criteria1: The condition for the first range.
    - [range2]: Optional. The second range of cells to check for another condition.
    - [criteria2]: Optional. The condition for the second range.
  - Example Usage: To sum the values in the range B1:B10 where the corresponding values in A1:A10 are greater than 50, and the corresponding values in C1:C10 are less than 100:
    - =SUMIFS(B1:B10, A1:A10, ">50", C1:C10, "<100")

- AVERAGEIF: Averages values meeting a condition.
  - =AVERAGEIF(range, criteria, [average_range])
    - range: The range of cells to check for the condition.
    - criteria: The condition to average based on (can be a number, text, or other).
    - [average_range]: The range of cells to average.
  - Example Usage: To calculate the average of the values in the range B1:B10 where the corresponding values in the range A1:A10 are greater than 50:
    - =AVERAGEIF(A1:A10, ">50", B1:B10)

- AVERAGEIFS: Averages values with multiple conditions.
  - =AVERAGEIFS(average_range, range1, criteria1, [range2], [criteria2], ...)
    - average_range: The range of cells to average.
    - range1: The first range of cells to check for the condition.
    - criteria1: The condition for the first range.
    - [range2]: Optional. The second range of cells to check for another condition.
    - [criteria2]: Optional. The condition for the second range.
  - Example Usage: To calculate the average sales in B1:B10 where: The sales targets in A1:A10 are greater than 50, The number of items sold in C1:C10 is less than 100:
    - =AVERAGEIFS(B1:B10, A1:A10, ">50", C1:C10, "<100")

Text Functions

Extracting Text

- LEFT: Extracts characters from the start of a string.
    - =LEFT(text, num_chars)
        - text: The text from which to extract characters.
        - num_chars: The number of characters to extract from the left.
    - Example Usage: To extract the first character from cell A8:
        - =LEFT(A8, 1)

- RIGHT: Extracts characters from the end of a string.
    - =RIGHT(text, num_chars)
        - text: The text from which to extract characters.
        - num_chars: The number of characters to extract from the right.
    - Example Usage: To extract the last two characters from cell A8:
        - =RIGHT(A8, 2)

- MID: Extracts characters from the middle of a string.
    - =MID(text, start_num, num_chars)
        - text: The text from which to extract characters.
        - start_num: The position of the first character to extract.
        - num_chars: The number of characters to extract.
    - Example Usage: To extract characters 4 and 5 of cell A8.
        - =MID(A8, 4, 2)

Merging Text

- CONCATENATE: Combines multiple strings.
    - =CONCATENATE(text1, text2, ...)
        - text1, text2, ...: The text or cell references you want to combine.
    - Example Usage: To combine the strings "John", "Smith", "lives in", and "London":
        - =CONCATENATE("John ", "Smith ", "lives in ", "London")
- "&" (Alternative to CONCATENATE)
    - "text1" & "text2" & ...
        - &: The ampersand combines text strings.
    - Example Usage: To combine the strings "John", "Smith", "lives in", and "London":
        - "John " & "Smith " & "lives in " & "London"

Modifying Text

- UPPER: Converts text to uppercase.
  - =UPPER(text)
    - text: The text to convert to uppercase.
  - Example Usage: To convert the text in cell A1 to uppercase:
    - =UPPER(A1)

- LOWER: Converts text to lowercase.
  - =LOWER(text)
    - text: The text to convert to lowercase.
  - Example Usage: To convert the text in cell A1 to lowercase:
    - =LOWER(A1)

- PROPER: Capitalizes the first letter of each word.
  - =PROPER(text)
    - text: The text to capitalize.
  - Example Usage: To convert the text in cell A1 to proper case (first letter capitalized):
    - =PROPER(A1)

- TRIM: Removes extra spaces from a string.
  - =TRIM(text)
    - Text: What you want to get trimmed.
  - Example Usage: To remove leading, trailing, and extra spaces between words in cell A1:
    - =TRIM(A1)

Getting Information from Text

- LEN: Returns the number of characters in a string.
  - =LEN(text)
    - text: The text to count the characters of.
  - Example Usage: To get the number of characters in cell A6:
    - =LEN(A6)

- SEARCH: Finds the position of a substring.
    - =SEARCH(find_text, within_text, [start_num])
        - find_text: The text you want to find.
        - within_text: The text to search within.
        - [start_num]: Optional. The position to start searching from.
    - Example Usage: To find the position of the first space in cell A6:
        - =SEARCH(" ", A6)
    - Possible Uses
        - Extracting First Name:
            - To extract the first name from a full name in A6:
                - =LEFT(A6, SEARCH(" ", A6)-1)
        - Extracting Last Name:
            - To extract the last name from a full name in A6:
                - =RIGHT(A6, LEN(A6)-SEARCH(" ", A6))

Working with Dates in Excel: Overview

In Excel, dates are stored as serial numbers, with each date corresponding to a unique number. The integer part represents the number of days since January 1, 1900 (the "base date" in Excel). For example:

- 1 represents January 1, 1900.
- 2 represents January 2, 1900.
- 43831 represents March 19, 2025.

The time is represented as a decimal part. For instance, 0.5 represents 12:00 PM (half of a 24-hour day).

Date Formats in Excel

Dates can be formatted in various ways, but they are fundamentally treated as numbers behind the scenes. The difference between two dates is simply the subtraction of their serial numbers, and you can calculate time differences in terms of days, months, or years.

Date Examples:

- Short Date (e.g., 3/19/25) → MM/DD/YY
- Long Date (e.g., Wednesday, March 19, 2025) → Fully spelled out date
- Custom Date (e.g., 2025 - March, 19 (Wednesday)) → Custom formatting
- Time (e.g., 7:29:09 PM) → Time format for hours, minutes, and seconds

Date & Time Functions

- DAY(A1) → Returns the day of the date in cell A1 (between 1 and 31).
- MONTH(A1) → Returns the month (1-12) from the date in A1.
- YEAR(A1) → Extracts the year from the date in A1.
- HOUR(A1), MINUTE(A1), SECOND(A1) → Extracts the respective components from a date-time value in A1.
- WEEKDAY(A1) → Returns the day of the week for the date in A1, where 1 = Sunday, 2 = Monday, and so on.
- TODAY() → Returns the current date (without the time).
- NOW() → Returns the current date and time.

Date Calculations

- Difference in Days:
    - Direct Subtraction: When subtracting dates, the result is the difference in days (including fractional days if times are involved).
        - Example: =B3-B5
        - This will output something like 6896.671168 days (if times are involved).
    - Using the DAYS Function: To find the difference in days between two dates, you can use the DAYS
        - Example: =DAYS(B3, B5)
        - This will give you the number of full days between the two dates (e.g., 6896 days).

- Difference in Years (Age Calculation)
    - Approximate Age (using days and dividing by 365):
    - Example: =(B3-B5)/365
        - This will give an approximate age of 18.89 years (including decimals). *Note that this is an approximation because it doesn't account for leap years.*
    - Simple Year Difference:
        - Example: =YEAR(B3)-YEAR(B5)
        - This formula gives the simple difference in years between two dates (e.g., 19 years), but it doesn't consider months and days.

- Exact Age with DATEDIF:
    - =DATEDIF(B5, B3, "Y")
    - This will give the exact number of years between two dates, accounting for months/days.

- Using Dates in a Logical Test
    - To check if a date is before or after a specific date:
        - Example: Is Today Before a Specific Date?
            - Using cell references for dynamic date checking:
                - =B5 < E35
            - Using the DATE function to reference a specific date:
                - =B5 < DATE(2025, 1, 1)
            - Using the TODAY() function for today's date:
                - =TODAY() < E35
    - In these examples, B5 is compared to a specific date (E35 or DATE(2025,1,1)). You can use TRUE or FALSE for conditional formatting or logic.
    - **Incorrect** Ways to Compare Dates (these formulas won't work):
        - =B5<1/1/2025 → This is not correct because Excel treats 1/1/2025 as a string and will not perform the comparison correctly.
        - =B5<"1/1/2025" → Also incorrect, as Excel interprets it as a string, not as a date.

Using Dates as Criteria

- You can use dates as criteria in functions like COUNTIF to count how many dates meet a certain condition.
  - Example: How Many Dates in B3:B5 Are Before 1/1/2025?
    - =COUNTIF(B3:B5, "<01/01/2025")
    - =COUNTIF(B3:B5, "<"&DATE(2025, 1, 1))
    - Both formulas count how many dates in the range B3:B5 are before January 1, 2025.
  - **Incorrect** Formula Examples (these will not work):
    - =COUNTIF(B3:B5,<01/01/2025) → Incorrect, as the operator should be inside the quotes.
    - =COUNTIF(B3:B5,B3<1/1/2025) → Incorrect syntax.

Advanced Text Functions

- CONCATENATE with Text Functions:
  - =CONCATENATE(UPPER(LEFT(text1, num_chars)), RIGHT(text2, num_chars), LOWER(text3), UPPER(LEFT(text4, num_chars)), "-", IF(logical_test, value_if_true, value_if_false))
    - UPPER(LEFT(text1, num_chars)): Takes the first num_chars from text1 and converts them to uppercase.
    - RIGHT(text2, num_chars): Takes the last num_chars from text2.
    - LOWER(text3): Converts text3 to lowercase.
    - UPPER(LEFT(text4, num_chars)): Takes the first num_chars from text4 and converts them to uppercase.
    - IF(logical_test, value_if_true, value_if_false): Returns different results based on a condition.
  - Example Usage:
    To extract the first two uppercase letters from B2, add the last four letters from A2, convert C2 to lowercase, extract the first three uppercase letters from D2, and add a suffix based on the value of E2:
    - =CONCATENATE(UPPER(LEFT(B2, 2)), RIGHT(A2, 4), LOWER(C2), UPPER(LEFT(D2, 3)), "-", IF(E2="Gold", "G", IF(E2="Silver", "S", IF(E2="Bronze", "B", "X"))))

Information & IS Functions

IS Functions

- ISBLANK: Checks if a cell is empty.
    - ISBLANK(value)
        - value: the cell you want to check
    - Example Usage: You are verifying if a required field in column A is empty. If the cell is blank, it should return "Empty"; otherwise, it should return "Filled".
        - =IF(ISBLANK(A1), "Empty", "Filled")

- ISTEXT: Checks if cell contains text.
    - ISTEXT(value)
        - value: the cell you want to check
    - Example: =IF(ISTEXT(E3),TODAY(),E3)-A3
        - ISTEXT(E3): Checks if E3 contains text.
            - IF(ISTEXT(E3), TODAY(), E3) - A3
                - If E3 contains text, it replaces it with today's date (TODAY()). Otherwise, it keeps the original value in E3.
                  (A3: Subtracts A3 from the resulting date.)

Error Handling

- IFERROR with AVERAGEIFS: Returns an average based on multiple criteria, handling errors.
    - =IFERROR(AVERAGEIFS(average_range, criteria_range1, criteria1, criteria_range2, criteria2), value_if_error)
        - value_if_error: The value to return if an error occurs.
    - Example Case: Want to find the average revenue (column H) for orders that match: Category in A6 (column D) and City "Milan" (column E).
        - =IFERROR(AVERAGEIFS(Products!H2:H474, Products!D2:D474, A6, Products!E2:E474, "Milan"), "No orders")
            - Averages H2:H474 (sales revenue) where:
                - Column D (category) matches A6.
                - Column E (city) equals "Milan".
            - If no matching data exists, it returns "No orders" instead of an error.

Lookup & Reference Functions

These functions allow you to search for specific values, retrieve data from tables, and reference data across your spreadsheet.

- VLOOKUP: Searches for value in a table and returns a corresponding value from another column.
    - =VLOOKUP(lookup_value, table_array, col_index_num, [range_lookup])
        - lookup_value The value to search for (e.g., A3).
        - table_array The table range to search in (e.g., Data!B$2:C$56).
        - col_index_num The column number (relative to the table) to return from (e.g., 2).
        - range_lookup FALSE for an exact match, TRUE for an approximate match.
    - Example Usage:You have a price list in the Data sheet (columns B and C), where B contains product names and C contains their prices.
        - =VLOOKUP(A3, Data!B$2:C$56, 2, FALSE)

- MATCH: returns the position of a value in a row or column within a specified range.
    - MATCH(lookup_value, lookup_array, match_type)
        - lookup_value The value you are looking for (C1).
        - lookup_array The row or column to search within (Products!$A1:$I1).
        - match_type 0 for an exact match.
    - Example Usage: find the column number in Products!A1:I1 that matches C1.
        - MATCH(C1, Products!$A1:$I1, 0)

Financial Functions

- Function Calculate Mortgage Payment
  - =PMT(rate, nper, pv, [fv], [type])
    - rate Interest rate per period.
    - nper Total number of payment periods.
    - pv Present value (loan amount).
    - fv (optional) Future value (default is 0).
    - type (optional) 0 for end-of-period payments, 1 for beginning-of-period.
  - Example Usage:
    To calculate the monthly mortgage payment in cell B6 of the Mortgage worksheet:
    - =PMT(B2/12, B3*12, -B4)
      - B2/12 Converts annual interest rate to monthly.
      - B3*12 Converts years to months.
      - -B4 Loan amount (negative because its an outgoing payment).

- IPMT & PPMT Interest and Principal Breakdown
  - IPMT (Interest Portion of Payment):
    - =IPMT(rate, per, nper, pv, [fv], [type])
  - PPMT (Principal Portion of Payment):
    - =PPMT(rate, per, nper, pv, [fv], [type])
      - rate Interest rate per period.
      - per Payment number (e.g., first, second, etc.).
      - nper Total number of payments.
      - pv Loan amount.
  - Example Usage:
    To calculate the interest and principal components of each mortgage payment in columns B to G of the Repayment Plan worksheet:
    - =IPMT($B$2/12, A2, $B$3*12, -$B$4)
    - =PPMT($B$2/12, A2, $B$3*12, -$B$4)
      - A2 represents the current payment number.
      - $B$2/12 converts annual interest to monthly.
      - $B$3*12 converts years to months.
      - $B$4 is the loan amount.

- FV Function Future Value Calculation
  - =FV(rate, nper, pmt, [pv], [type])
    - rate Interest rate per period.
    - nper Total number of periods.
    - pmt Regular payment (if applicable).
    - pv Present value (optional, default is 0).
    - type 0 (end of period), 1 (beginning of period).
  - Example Usage: To calculate the future value of an investment in cells B5 and B6:

- =FV(B2/12, B3*12, -B4, 0)
  - B2/12 Converts annual interest to monthly.
  - B3*12 Converts years to months.
  - B4 Initial investment amount (negative because its outgoing).
    - If investing a fixed amount monthly:
      - =FV(B2/12, B3*12, -B5, -B4)
      - B5 represents monthly deposits.

Financial tools

- Goal Seek Find Needed Loan Amount
  - Steps to Use Goal Seek:
    - Select the mortgage payment cell (e.g. B6).
    - Go to Data What-If Analysis Goal Seek.
    - Set To value to the desired monthly payment.
    - Set By changing cell to B3 (loan term, for example).
    - Click OK to let Excel adjust B3 to achieve the target payment.
  - Example Use Case:
    You want to determine the required loan term (B3) for a fixed payment (B6). Goal Seek wil l adjust B3 until B6 equals the desired payment.

- Naming Ranges for Readability
  - Select the range in the worksheet.
  - Go to Formulas, Define Name.
  - Assign names.
  - Use named ranges in formulas instead of cell references.
    - Example Use Case:
      - Instead of: =FV(B2, B3, 0, -B4)
      - =FV(Interest_Rate, Years, 0, -Investment_Amount)
    - This makes formulas easier to read.

Pivot Tables and Macros

Pivot Table

A Pivot Table is an Excel feature used to quickly summarize, analyze, and organize large datasets. It allows users to group, filter, and perform calculations without modifying the original data. Pivot Tables are essential for financial analysis, sales reports, and trend tracking. Key concepts include Row Labels, Column Labels, Values, and Filters. **Practice creating Pivot Tables and using different aggregation functions to get comfortable with them.**

- Group pivot table values
  - Select a cell in the PivotTable containing a value you want to group by.
  - Go to Analyze → Group Field and specify the grouping criteria.

- Refresh a PivotTable
  - Select the PivotTable.
  - Go to Analyze → Data group → Refresh.

- Format a PivotTable
  - Select the PivotTable.
  - Go to Design → PivotTable Options & Styles.
  - Choose the desired formatting options.

- Create a PivotChart
  - Click any cell in the PivotTable.
  - Go to Analyze → Tools → PivotChart.
  - Select a PivotChart type.

- Modify PivotChart Data
  - Drag fields into and out of the field areas in the task pane.

- Modify PivotChart Elements
  - Select the PivotChart.
  - Go to Design → Chart Elements.
  - Click Add Chart Element and select the item(s) to add.

- Subtotals & Grand Totals
  - Subtotals: Show or hide subtotals and specify their location in the PivotTable.
  - Grand Totals: Add or remove grand total rows for columns and/or rows.

MACRO

An Excel Macro is a set of recorded actions that automate repetitive tasks in Excel. Macros are created using VBA (Visual Basic for Applications), allowing users to execute a sequence of steps with a single command. They are useful for automating data entry, formatting, and calculations, reducing manual effort and errors. However, macros can contain security risks, so it's important to enable them only from trusted sources. **Practice recording and editing macros in the Developer tab to understand how they work.**

- Record a Macro
  - Go to Developer → Record Macro.
  - Enter a name and description, then specify where to save it.
  - Complete the actions to be recorded.
  - Click Stop Recording on the Developer tab.

- Run a Macro
  - Go to Developer → Macros.
  - Select the macro and click Run.

- Edit a Macro
  - Go to Developer → Macros.
  - Select a macro and click Edit.
  - Modify the Visual Basic code as needed, then save.

- Delete a Macro
  - Go to Developer → Macros.
  - Select the macro and click Delete.

- Create a Button and Assign It to a Macro
  - Go to Developer → Insert → Button (Form Control).
  - Draw the button onto the worksheet.
  - In the Assign Macro dialog that appears, select the macro you want to run and click OK.
  - You can right-click the button to edit the text or change its formatting.

- **Important: Select the Correct Worksheet**
  - This ensures your macro always runs in the right place, especially if multiple sheets are open.

Python

Getting Started with Python Basics

The IDLE

1. The shell is used for interactive execution (e.g., user input).
2. The editor runs standalone scripts (e.g., writing to a file).

IDLE Color Coding

- Purple: Functions (e.g., print(), input())
- Red: Errors (e.g., SyntaxError, NameError)
- Blue: Output (e.g., printed text)
- Green: Comments (e.g., # This is a comment or '''Multi-line''')

Variables

- Variables cannot start with a number.
- Use = to assign values (e.g., name = "John").

Common Data Types:

- str (String): "Hello"
- int (Integer): 42
- float (Floating point): 3.14
- bool (Boolean): True / False

Basic Operations

Operator	Description	Examples
==	Equal to	4 == 4 -> True 4 == 73 -> False
!=	Different from	4 != 73 -> True 4 != 4 -> False
>	Greater than	73 > 4 -> True 4 > 73 -> False
>=	Greater than or equal to	73 >= 4 -> True 4 >= 4 -> True
<	Less than	4 < 73 -> True 73 < 4 -> False
<=	Less than or equal to	4 <= 73 -> True 4 <= 4 -> True

Basic Input and Output

Basic functions

- print() : Writes on the screen what you want.
  - Examples:
    - print("Hello!")
    - print(5 + 3)
  - Application: Used to display messages, debug programs, or show calculated results.

- input() : Displays the text inside the parentheses and waits for the user to type something.
  - Example:

    ```
    name = input("Enter your name: ")
    print("Hello, " + name + "!")
    ```
  - Application: Accepts user input for interactive programs.

- help() : Displays documentation on Python functions or objects.
  - Example:

    ```
    help(print)
    ```
  - Application: Used to learn more about built-in functions and modules.

Variables and Data Types

- Variable Assignment: Stores values in memory for later use.
  - Example:
    - age = 25
    - name = "Alice"
  - Application: Holds user inputs, calculations, or retrieved data.
  - Variables can change values (reassignment)
    - Example:
      - x = 10
      - x = x + 5 # Now x is 15
    - Application: Used to update values during program execution.

- Multiple Assignment: Assigns values to multiple variables in one line.
  - Example:
    - a, b, c = 1, 2, 3
  - Application: Reduces redundancy in variable initialization.

- type() : Returns the data type of a variable.
    - Example:
        - `print(type(5))  # Output: <class 'int'>`
        - `print(type("Hello"))  # Output: <class 'str'>`
    - Application: Helps in debugging and dynamic data type management.

- Data Type Conversion: Converts between types.
    - Example:
        - `x = int("10")  # Converts string to integer`
        - `y = float("3.14")  # Converts string to float`
        - `z = str(100)  # Converts number to string`
    - Application: Ensures proper data type usage in calculations or concatenation.

Mathematical Operations

- Basic Arithmetic (+, -, *, /) : Performs basic calculations.
    - Example:
        - `result = (5 + 3) * 2`
        - `print(result)  # Output: 16`
    - Application: Used for numerical computations in programs.

- Exponentiation (**) : Raises a number to a power.
    - Example:
        - `print(2 ** 3)  # Output: 8`
    - Application: Required for scientific calculations and formulas.

- Floor Division (//) : Divides and rounds down to an integer.
    - Example:
        - `print(10 // 3)  # Output: 3`
    - Application: Used for integer-based calculations.

- Modulus (%) : Returns the remainder of a division.
    - Example:
        - `print(10 % 3)  # Output: 1`
    - Application: Useful in checking even/odd numbers.

Conditional/Control Flow Statements

Conditional Statements

- IF Statement: Executes code based on a condition.
    - Example:

        -
          ```
          if age >= 18:
                  print("You are an adult.")
          ```
    - Application: Used for decision-making in programs.
    - Remember to indent the statements in rows bellow "IF" that belong to the condition.

- Else: used to define what happens when the if or elif conditions are not met.
    - Example:

        -
          ```
          if x > 10:
                  print("Greater than 10")
          else:
                  print("10 or less")
          ```
    - Application: Executes code when all preceding conditions (if, elif) are false.
    - Remember to indent (and indent further if nesting the functions.)

- Elif: lets multiple conditions be checked. It only runs if the previous conditions were not true.
    - Example:

        -
          ```
          if x > 10:
                  print("Greater than 10")
          elif x == 5:
                  print("Equal to 5")
          else:
                  print("Less than 5")
          ```
    - Application: Used when you want to check multiple conditions.

- In: checks if a value exists in a sequence (like a list, tuple, string, or range). It returns True if the value is found, otherwise False.
    - Example:
        - Checking in a String:
            - `print("a" in "apple")  # True`
            - `print("z" in "apple")  # False`
        - Checking in a Range:
            - `print(5 in range(10))  # True`
            - `print(15 in range(10)) # False`

- AND, OR, NOT : Combines conditions.
    - Example: (AND)

        -
          ```
          if x > 0 and x < 10:
                  print("x is between 0 and 10")
          ```
    - Application: Ensures multiple conditions are met before executing logic.
    - Example: (Or)

        -
          ```
          x = 15
          if x < 0 or x > 10:
                  print("x is either less than 0 or greater than 10")
          ```
    - Application: Ensures that at least one condition is met before executing logic.
    - Example: (NOT)

        -
          ```
          x = 5
          if not x > 10:
                  print("x is not greater than 10")
          ```
    - Application: Reverses a condition, ensuring logic is executed when the condition is false.

Looping Constructs

Looping Constructs

- FOR Loop: Automates repetitive tasks.
  - Example:
    ```
    for i in range(1,11):
        print(num,'x',i,'=',num*I)
    ```
  - Creates a multiplication table for a given integer from 1 to 10

- WHILE Loop : Repeats code while a condition is true.
  - Example:
    ```
    x = 1
    while x >=10:
        print(x)
        x=x+1
    ```
  - Counts if the resulting number is less than or equal to 10.
  - Application: Used when the number of iterations is unknown.

- Range(): function generates a sequence of numbers.
  - range(start, stop, step)
    - start (optional, default = 0): The starting number.
    - stop (required): The number where the sequence stops (not included).
    - step (optional, default = 1): The increment between numbers.
  - Example:
    ```
    print(list(range(5))) # [0, 1, 2, 3, 4]
    ```

- Continue: Creates a shortcut to the next shift of the while loop.
  - Example:
    ```
    for number in range(1,51):
        if number % 2 == 0:  # Check if the number is even
            continue  # Skip even numbers
    print("Odd number found: ", number) # Print only odd numbers
    ```

- Break: immediately exits the loop, as soon as a certain condition is met.
  - Example:
    ```
    for i in range(5):
        if i == 3:
            break  # Stops the loop when i is 3
        print(i)  # Prints i if it's not 3
    ```

Sequences

In Python, sequences are a type of collection that can hold multiple items. The most common sequence types are *strings, lists, and tuples.*

Sequence Operations

- You can access a slice of a sequence, by entering it as [n:m:o]
    - n is the start of the slice
    - m is the end of the slice
    - o is how often you pick elements from the string
  - Example:
    - MyQuote [5:10:2]
      Starting from the 6th element to the 11th, it picks out every other item.

- len(seq): Returns the number of elements in the sequence.
  - Example:
    - len([10, 20, 30]) # Output: 3

- max(seq) and min(seq): Returns the largest/smallest value in a sequence
  - The sequence must contain only strings or only numbers
  - Example:
    - max([4, 7, 2]) # Output: 7
    - min("apple") # Output: 'a'

- sorted(seq): Returns a new list with elements sorted in ascending order
  - The sequence must contain only strings or only numbers.
  - Example:
    - sorted([3, 1, 4]) # Output: [1, 3, 4]

- +: Concatenates/merges two sequences
  - Example:
    - [1, 2] + [3, 4] # Output: [1, 2, 3, 4]

- *: Repeats a sequence (creates multiple copies and merges them)
  - Example:
    - [1, 2] * 3 # Output: [1, 2, 1, 2, 1, 2]

- in: Returns true if an element is found in a sequence, otherwise returns False
  - Example:
    - 3 in [1, 2, 3, 4] # Output: True

- sum(seq): Sums the elements of a sequence (numbers only)
  - Example:
    - sum([2, 5, 8]) # Output: 15

Strings

Strings are created by enclosing a sequence of characters in either single quotes ' ' or double quotes " ". The characters are ordered, meaning the position of each character in the string is significant.

```
my string = "Hello, world!"
```

In this example, the string contains the characters "H", "e", "l", "l", "o", ",", " ", "w", "o", "r", "l", "d", and "!". The order of these characters matters.

String Operations

- Concatenation (+) : Joins strings together.
  - Example:
    - `first = "Hello"`
    - `second = "World"`
    - `print(first + " " + second)`
  - Application: Used to build messages, user prompts, and formatted text.

- Repetition (*) : Repeats a string multiple times.
  - Example:
    - `print("Python " * 3)        #Output: Python Python Python`
  - Application: Useful for text formatting and patterns.

- Escape Characters (\n, \t): Adds new lines and tabs.
  - Examples:
    - `print("Line1\nLine2")`
    - `print("Column1\tColumn2")`
  - Application: Formats text output for better readability.

- String Formatting (format()) : Formats numbers and text.
  - Example:

    -
      ```
      name = "Tancredi"
      age = 19
      print("My name is {} and I am {} years old.".format(name, age))
              #output: My name is Tancredi and I am 19 years old.
      ```
  - Application: Improves dynamic text output in applications.

String Methods

Strings are **immutable**: once they are created, their contents cannot be changed. Any operation that appears to modify a string, such as replacing characters or changing case, creates a new string rather than altering the original one.

- .upper(): Converts all characters in a string to uppercase.
  - Example:
    - "hello".upper() → "HELLO"
- .lower(): Converts all characters in a string to lowercase.
  - Example:
    - "HELLO".lower() → "hello"

- .find(sub): Returns the index of the first occurrence of sub in the string.
    - sub: The substring to search for.
  - Example:
    - "hello".find("l") → 2

- .startswith(prefix): Returns True if the string starts with prefix, otherwise returns False.
    - prefix: The substring to check.
  - Example:
    - "hello".startswith("he") → True

- .endswith(suffix): Returns True if the string ends with suffix, otherwise returns False.
    - suffix: The substring to check.
  - Example:
    - "hello".endswith("lo") → True

- .count(sub): Returns the number of times sub appears in the string.
    - sub: The substring to count.
  - Example:
    - "banana".count("a") → 3

- .split(): Splits a string into a list of words using spaces by default.
  - Example:
    - "hello world".split() → ["hello", "world"]

- .join(iterable): Concatenates all elements in iterable into a string, using the given string as a separator.
    - iterable: A list of strings.
  - Example:
    - "-".join(["hello", "world"]) → "hello-world"

Lists

Lists are created by enclosing items in square brackets [ ], with each item separated by a comma.

```
my_list = [1, 2, 3, "apple"]
```

In this example, the list contains integers and a string. The elements are ordered, meaning the position of each item is significant.

Working with Lists

The first element of the list is accessed with index 0 (e.g., myList[0]). Additionally, you can access elements from the end of the list using negative indices (e.g., myList[-1] for the last item).

- Lists : Stores multiple values in order.
    - Example:

        -
          ```
          fruits = ["apple", "banana", "cherry"]
          print(fruits[0])     # Output: apple
          ```
    - Application: Used for handling collections of data.

- List Slicing: Allows extracting a portion of a list by specifying a start and end index. The first index is inclusive, and the last index is exclusive.
    - Example:

        -
          ```
          fruits = ["apple", "banana", "cherry", "date", "elderberry"]
          print(fruits[1:4]) # Output: ['banana', 'cherry', 'date']
          ```
    - Application: Used for working with subsets of data within a list.

List Methods

lists are mutable, meaning their contents can be changed after they are created. You can modify, add, or remove elements from a list without creating a new one.

- .append(element): Adds element to the end of the list.
  - Example:
    - lst = [1, 2, 3]; lst.append(4) → [1, 2, 3, 4]

- .insert(index, element): Inserts element at the specified index in the list.
  - Example:
    - lst = [1, 3]; lst.insert(1, 2) → [1, 2, 3]

- .remove(element): Removes the first occurrence of element from the list.
  - Example:
    - lst = [1, 2, 3, 2]; lst.remove(2) → [1, 3, 2]

- .pop(index): Removes and returns the element at index. If index is not given, removes and returns the last element.
  - Example:
    - lst = [1, 2, 3]; lst.pop(1) → 2, lst becomes [1, 3]

- .index(element): Returns the index of the first occurrence of element in the list.
  - Example:
    - lst = [1, 2, 3]; lst.index(2) → 1

- .count(element): Returns how many times element appears in the list.
  - Example:
    - lst = [1, 2, 2, 3]; lst.count(2) → 2

- .sort(): Sorts the list in ascending order (default).
  - Example:
    - lst = [3, 1, 2]; lst.sort() → [1, 2, 3]

- .reverse(): Reverses the order of elements in the list.
  - Example:
    - lst = [1, 2, 3]; lst.reverse() → [3, 2, 1]

Tuples

A tuple is an ordered, immutable collection of items in Python. Unlike lists, tuples cannot be modified after creation (they are immutable). Tuples are commonly used to store a collection of items that should not change.

Syntax: close your items with parenthesis. For example: tuple_name = (item1, item2, item3, ...)

Tuple Methods

- index():is used to find the index of the first occurrence of a specific value in the tuple.
    - Returns the index of the first occurrence of the specified value.
    - If the value is not found, it raises a ValueError.
  - Example:

```
# Finding the index of an element in the tuple
person = ("John", 25, "Engineer")
index_of_name = person.index("John")
print(index_of_name) # Output: 0
```

- count(): counts how many times a specific value appears in the tuple.
    - Returns the count of the specified value in the tuple.
  - Example:

```
# Counting the occurrences of an element in the tuple
numbers = (1, 2, 3, 4, 2, 5, 2)
count_of_twos = numbers.count(2)
print(count_of_twos) # Output: 3
```

Dictionaries

Dictionaries are collections of elements that do not have a defined order. In a dictionary, each key must be unique—no two keys can be identical—while values can be repeated. *For example, in a phone contacts app, each phone number (key) must be unique, but multiple people (values) can have the same name.*

Dictionaries are created by enclosing key-value pairs in curly braces { }. Each key-value pair is separated by a colon :, and the pairs are separated by commas .

```
my_dict = {"name": "John", "age": 30}
```

In this example, the dictionary contains two key-value pairs: "name" is the key, and "John" is its corresponding value; "age" is the key, and 30 is its value. The order of the key-value pairs doesn't matter, but the keys must always be unique.

Working with Dictionaries

- Dictionaries : Stores key-value pairs.
    - Example:
        - `student = {"name": "Tim Cheese", "age": 25}`
        - `print(student["name"])  # Output: Tim Cheese`
    - Application: Efficient for structured data storage and retrieval.

Operations with Dictionaries

- `in`: Checks if a specific key exists in a dictionary.
        - key: The key you want to search for.
        - dictionary: The dictionary you're checking.
    - Example Usage:
        - To check if "name" is in the person dictionary:
        - `"name" in person`

- `del`: Deletes a key-value pair from the dictionary.
    - `del dictionary[key]`
        - dictionary: The dictionary you want to delete from.
        - key: The key of the item you want to remove.
    - Example Usage:
        - To remove the "age" entry from the person dictionary:
        - `del person["age"]`

- `len()`: Returns the number of key-value pairs in a dictionary.
    - `len(dictionary)`
        - dictionary: The dictionary to get the count from.
    - Example Usage:
        - To find out how many items are in the person dictionary:
        - `len(person)`

Methods with Dictionaries

Dictionaries are mutable: this means that once a dictionary is created, its contents can be changed. Unlike strings, where changes create new objects, dictionaries allow you to directly modify their keys and values.

- get(): Returns the value for a specified key in a dictionary. If the key doesn't exist, it returns None (or a default value if provided).
    - dictionary.get(key, default)
        - key: The key whose value you want to retrieve.
        - default (optional): A value to return if the key is not found.
    - Example Usage:
        - To get the value of "name" from the person dictionary:
        - person.get("name")
            - To return "Unknown" if "name" doesn't exist:
            - person.get("name", "Unknown")

- pop(): Removes the specified key and returns its value.
    - dictionary.pop(key, default)
        - key: The key to remove.
        - default (optional): Value to return if key is not found (prevents error).
    - Example Usage:
        - To remove "age" from the person dictionary and get its value:
        - person.pop("age")

- popitem(): Removes and returns the last inserted key-value pair as a tuple.
    - dictionary.popitem()
        - No arguments required.
    - Example Usage:
        - To remove and return the last item from the person dictionary:
        - person.popitem()

- items(): Returns a view object containing all key-value pairs as tuples.
    - dictionary.items()
        - dictionary: The dictionary you want to inspect.
    - Example Usage:
        - To get all key-value pairs in the person dictionary:
        - person.items()

- keys(): Returns a view object of all the keys in the dictionary.
    - dictionary.keys()
        - dictionary: The dictionary to get keys from.
    - Example Usage:
        - To list all keys in the person dictionary:
        - person.keys()

- values(): Returns a view object of all the values in the dictionary.
    - dictionary.values()
        - dictionary: The dictionary to get values from.
    - Example Usage:
        - To list all values in the person dictionary:
        - person.values()

Traversing dictionaries

- You can use a for loop to go through each key in a dictionary.
    - for key in dictionary:
        - key: The current key in the loop.
        - dictionary: The dictionary you're iterating over.
    - Example Usage:
        - To print all keys in the person dictionary:

        -
          ```
          for key in person:
                   print(key)
          ```

- You can use .items() with a for loop to access both keys and values at the same time.
    - for key, value in dictionary.items():
        - key: The current key in the loop.
        - value: The value associated with the current key.
    - Example Usage:
        - To print all key-value pairs in the person dictionary:

        -
          ```
          for key, value in person.items():
                   print(key, value)
          ```

Functions

Functions can be void or value returning. Void functions perform actions but don't return a value, while value-returning functions give back a result using the return keyword.

Functions basics

- Defining Functions : Creates reusable blocks of code.
- The **def** keyword starts the definition, followed by the function name and parentheses which may include parameters.
- Parameters are the values that the function needs to perform its task, and you can set default values for parameters in case they are not provided when the function is called.
  - Example:

    -
      ```
      def greet(name="Guest", age=100):
          return f"Hello, {name}, you are {age} years old."
      print(greet("John", 25))        # Output: Hello, John, you are 25 years old.
      print(greet())                  # Output: Hello, Guest, you are 100 years old.
      ```
  - Application: Helps organize code and avoid repetition.

Modules

Modules in Python are files that contain reusable code, like functions or variables, which help organize your program. You can use the import statement to bring in a module and access its features in your code.

- Importing Modules : Imports extra functionality.
  - Example:

    -
      ```
      import math
      print(math.sqrt(25))
      ```
  - Application: Extends Python's capabilities with external libraries.

Main Modules

- Math module: Offers access to mathematical operations.
  - Example:

    -
      ```
      import math
      area = math.pi * (5 ** 2)
      math.pi             # 3.141592653589793 (The constant π)
      math.e              # 2.718281828459045 (Euler's number)
      math.sqrt(16)         # 4.0 (Square root)
      math.factorial(5)       # 120 (5!)
      math.floor(3.9)         # 3 (Round down)
      math.ceil(3.1)          # 4 (Round up)
      math.sin(math.pi / 2)   # 1.0 (Sine of 90°)
      math.cos(0)             # 1.0 (Cosine of 0°)
      math.degrees(math.pi)   # 180.0 (Convert radians to degrees)
      math.radians(180)       # 3.141592653589793 (Convert degrees to radians)
      # Area of a circle with radius 5
      area = math.pi * (5 ** 2)  # 78.53981633974483
      ```

- Random module: Generates random values.
  - Example:

    -
      ```
      import random
      random.random()          # Float between [0, 1)
      random.randint(1, 10)    # Integer between 1 and 10
      random.randrange(0, 10, 2) # Random even number <10
      random.choice(['a', 'b'])  # Random element from list
      ```

Error Handling

Types of errors

1. Syntax Errors: Occur when there is a mistake in how the code is written.
   - Features:
     - Python displays a traceback message in the shell, indicating the error occurs.
     - The message does not suggest how to fix the error.
     - The error message always starts with SyntaxError.
   - Example:

     -
       ```
       if True:
           Print("Hello")
       ```
     - Error Output: `SyntaxError: expected ':'`

2. Runtime errors: Occur when the program is running
   - Examples
     - We try to convert to an integer the string “Hello”
     - We try to divide by 0
     - We can handle these errors as if they were exceptions

3. Semantic Errors: Occur when the program runs without errors but produces incorrect or unexpected results.
   - Features:
     - Caused by wrong code design (also called logic errors).
     - Do not trigger error messages, making them difficult to detect.
     - Require step-by-step analysis or debugging tools to find and fix.
   - Example:

     -
       ```
       def calculate_area(width, height):
               return width + height  # Incorrect formula, should be width * height
       print(calculate_area(5, 3))  # Expected: 15, Output: 8
       ```
     - The program runs, but the result is incorrect due to a logic mistake.

Exceptions handling:

- Try: write code that might throw an error and gives you a way to handle that error.
  - Example:

    -
      ```
      try:
              x = 10 / 0  # This will raise a ZeroDivisionError
      except ZeroDivisionError:
              print("Can't divide by zero!")  # Properly indented
      ```

    Application: Prevents the program from crashing when an error occurs.

- Except: follows a try block and handles the error if one occurs.
    - Example:

        -
          ```
          try:
              x = 10 / 0  # This will raise a ZeroDivisionError
          except ZeroDivisionError:
              print("Can't divide by zero!")  # Properly indented
          ```
    - Application: Catches and handles specific exceptions or errors in your code.

FOR DOUBTS OR SUGGESTIONS ON THE HANDOUTS

TANCREDI SEQUI

tancredi.sequi@studbocconi.it

@tancredisequi

+39 3922892129

FOR INFO ON THE TEACHING DIVISION

VITTORIA NASONTE

vittoria.nasonte@studbocconi.it

@_vittorian_

+39 3274441476

ELENA CACIOLI

elena.cacioli@studbocconi.it

@elenacaciolii_

+39 3928931605

TEACHING DIVISION

OUR PARTNERS

Wall Street English®

TEGAMINO'S

ETHAN
SUSTAINABILITY

700+
CLUB

DELIVERY VALLEY
NO GENDER KITCHEN

LA PIADINERIA

LA PIADINERIA
dal 1994