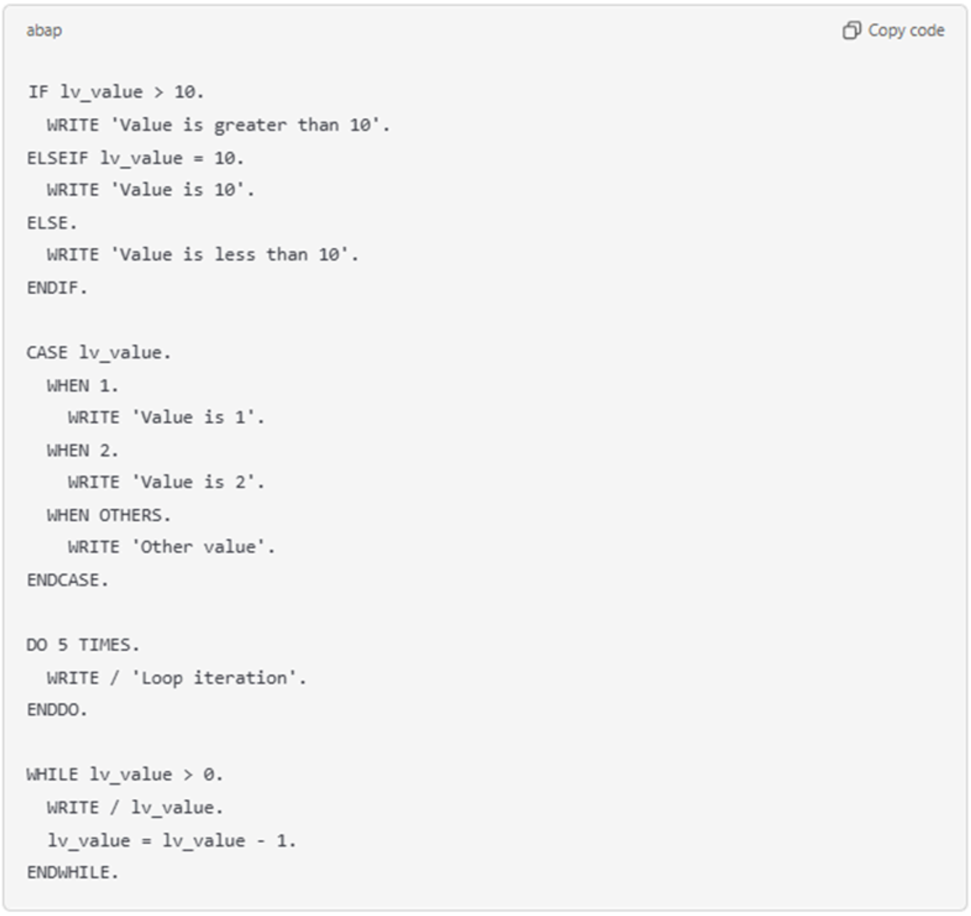
1. Data Declarations and Types

```abap
DATA lv_value TYPE i VALUE 10.
CONSTANTS lc_name TYPE string VALUE 'SAP'.
TYPES: BEGIN OF ty_person,
         name TYPE string,
         age  TYPE i,
       END OF ty_person.
DATA: lt_people TYPE TABLE OF ty_person.
FIELD-SYMBOLS: <fs_person> TYPE ty_person.
```

2. Control Structures



```abap
IF lv_value > 10.
  WRITE 'Value is greater than 10'.
ELSEIF lv_value = 10.
  WRITE 'Value is 10'.
ELSE.
  WRITE 'Value is less than 10'.
ENDIF.

CASE lv_value.
  WHEN 1.
    WRITE 'Value is 1'.
  WHEN 2.
    WRITE 'Value is 2'.
  WHEN OTHERS.
    WRITE 'Other value'.
ENDCASE.

DO 5 TIMES.
  WRITE / 'Loop iteration'.
ENDDO.

WHILE lv_value > 0.
  WRITE / lv_value.
  lv_value = lv_value - 1.
ENDWHILE.
```

The `/` symbol in `WRITE / 'Loop iteration'.` indicates that a **new line** is created **before** the text is printed.

Explanation:

- `SY-INDEX` holds the current iteration count of the loop.
- In this example, `SY-INDEX` starts at 1 and increments by 1 with each iteration until it reaches the loop's specified limit (5 in this case).

4. SY-UNAME

- **Purpose:** Contains the current user's username.
- **Usage:** Often used for tracking or logging purposes.
- **Example:**

```abap
WRITE: / 'Current user:', sy-uname.
```

- **Typical Value:** The username of the person logged into SAP.

5. SY-DATUM

- **Purpose:** Stores the current system date.
- **Usage:** Used to get the current date in ABAP programs.
- **Example:**

```abap
WRITE: / 'Current date:', sy-datum.
```

- **Typical Value:** Date in `YYYYMMDD` format.

6. SY-UZEIT

- **Purpose:** Contains the current system time.
- **Usage:** Often used for timestamping logs or tracking execution time.
- **Example:**

```abap
WRITE: / 'Current time:', sy-zeit.
```

- **Typical Value:** Time in `HHMMSS` format.

FUNCTION - FORM -

Step 1: Define the FORM

```abap
FORM calculate_sum USING iv_num1 TYPE i
                         iv_num2 TYPE i
                CHANGING ev_result TYPE i.

  ev_result = iv_num1 + iv_num2.
  WRITE: / 'The sum of', iv_num1, 'and', iv_num2, 'is', ev_result.

ENDFORM.
```

- **USING**: Defines the input parameters (`iv_num1` and `iv_num2` of type `i`).
- **CHANGING**: Defines an output parameter (`ev_result` of type `i`).
- **ENDFORM**: Marks the end of the subroutine.

Step 2: Call the FORM in the Main Program

```abap
DATA: lv_num1 TYPE i VALUE 5,
      lv_num2 TYPE i VALUE 10,
      lv_result TYPE i.

" Call the FORM with parameters
PERFORM calculate_sum USING lv_num1 lv_num2 CHANGING lv_result.

" Output the result
WRITE: / 'The result from FORM is:', lv_result.
```

Explanation

- `PERFORM calculate_sum`: Calls the `calculate_sum` FORM, passing `lv_num1` and `lv_num2` as inputs, and `lv_result` as a changing parameter to receive the result.
- The FORM calculates the sum of `iv_num1` and `iv_num2` and assigns it to `ev_result`, which is then printed within the FORM.

Expected Output

```python
The sum of 5 and 10 is 15
The result from FORM is: 15
```

Key Points

- **USING** parameters are for input values to the `FORM`.
- **CHANGING** parameters allow you to return values from the `FORM` to the main program.
- `PERFORM` is used to call the `FORM`.

This modular structure makes the code more organized and reusable. You can call the `calculate_sum` FORM multiple times with different input values to perform the same operation.

4. Internal Tables

abap Copy code

```
DATA: lt_numbers TYPE TABLE OF i,
      lv_num     TYPE i.

APPEND 5 TO lt_numbers.
APPEND 10 TO lt_numbers.
INSERT 15 INTO lt_numbers INDEX 2.

LOOP AT lt_numbers INTO lv_num.
  WRITE / lv_num.
ENDLOOP.

DELETE lt_numbers WHERE table_line = 10.
SORT lt_numbers.

READ TABLE lt_numbers INDEX 1 INTO lv_num.
WRITE / lv_num.
```

5. Database Access

abap Copy code

```
DATA: lt_customers TYPE TABLE OF kna1,
      ls_customer TYPE kna1.

SELECT * FROM kna1 INTO TABLE lt_customers WHERE land1 = 'US'.

LOOP AT lt_customers INTO ls_customer.
  WRITE: / ls_customer-name1, ls_customer-ort01.
ENDLOOP.

UPDATE kna1 SET name1 = 'New Name' WHERE kunnr = '0001'.
INSERT INTO kna1 VALUES ls_customer.
DELETE FROM kna1 WHERE kunnr = '0001'.
```

Example with Field Symbols

abap

Copy code

```abap
TYPES: BEGIN OF ty_person,
         name TYPE string,
         age  TYPE i,
       END OF ty_person.

DATA: ls_person TYPE ty_person.

ls_person-name = 'John Doe'.
ls_person-age = 30.

" Declare a field symbol for the structure
FIELD-SYMBOLS <fs_person> TYPE ty_person.

" Assign the structure to the field symbol (pointer-like behavior)
ASSIGN ls_person TO <fs_person>.

" Accessing and modifying the structure fields through the field symbol
<fs_person>-name = 'Jane Doe'.
<fs_person>-age = 25.

WRITE: / 'Name:', ls_person-name, 'Age:', ls_person-age.
```

Key Components of the Data Dictionary

1. Database Tables:
   - Database tables in the SAP Data Dictionary are the primary storage structures where data is stored at the database level.
   - Types of Database Tables:
     - **Transparent Tables:** Store application data in a standard relational database table format. Each SAP table corresponds to an actual table in the database.
     - **Pooled Tables:** Store control data (like configuration data) in pooled tables within the database. Multiple pooled tables are stored in a single table at the database level.
     - **Cluster Tables:** Similar to pooled tables, cluster tables store control data in a compressed form. Several cluster tables are stored in one table at the database level.
   - **Primary and Foreign Keys:** Define relationships between tables by linking keys in a primary table to keys in a foreign table, maintaining referential integrity.
2. Views:
   - Views allow access to data across multiple tables without physically storing the data again. They represent a logical view of one or more tables.
   - Types of Views:
     - **Database Views:** Join data from multiple tables at the database level.
     - **Projection Views:** Show a subset of fields from a single table.
     - **Help Views:** Used in search helps to retrieve data from multiple tables.
     - **Maintenance Views:** Allow users to maintain data across multiple tables through a single view.
3. Data Elements:
   - A data element defines the semantic meaning of a database field or structure component. It contains descriptive information like field labels and documentation.
   - Each data element is associated with a domain, which defines the technical attributes of the data (like type, length, and possible values).

4. Domains:
    - Domains define the technical properties of a field, such as data type, length, and value ranges.
    - Domains provide value checks for fields (for example, only allowing values within a specific range or matching certain patterns).
    - Domains are reusable; multiple data elements can share the same domain.
5. Structures:
    - Structures are complex data types composed of fields from different data elements. Unlike database tables, structures do not store data persistently.
    - Structures are used to group related fields together, often for use in programs or as the basis for screen fields in ABAP reports.
6. Type Groups:
    - Type groups are collections of data types, constants, and structures. They provide reusable types that can be used in ABAP programs.
    - They help in organizing related types, especially for custom data types in development.
7. Search Helps:
    - Search helps provide user-friendly input assistance (F4 help) for entering field values.
    - Types of Search Helps:
        - **Elementary Search Help:** Based on a single data source, typically a table.
        - **Collective Search Help:** Combines multiple elementary search helps for a single field.