# SAP ABAP

# Complete Guide for Freshers

## A Comprehensive Introduction to SAP ABAP Programming

Generated: October 2025

# Table of Contents

# 1. Introduction to SAP and ABAP

## 1.1 What is SAP?

SAP (Systems, Applications, and Products in Data Processing) is a leading enterprise resource planning (ERP) software used by organizations worldwide to manage business operations and customer relations. SAP provides integrated solutions for finance, human resources, manufacturing, supply chain, and more.

## 1.2 What is ABAP?

ABAP (Advanced Business Application Programming) is a high-level programming language created by SAP for developing applications on the SAP platform. It is the primary language for customizing and extending SAP functionality. ABAP is used to create reports, interfaces, conversions, enhancements, forms, and workflows.

## 1.3 Why Learn ABAP?

• High demand in the job market with competitive salaries

• Opportunity to work with global organizations

• Stable and growing career path in enterprise software

• Integration with cutting-edge technologies like SAP HANA, Fiori, and Cloud

• Comprehensive ecosystem with extensive learning resources

## 1.4 SAP System Architecture

The SAP system follows a three-tier architecture:

• **Presentation Layer:** User interface (SAP GUI, Web browsers, Fiori apps)

• **Application Layer:** Business logic and ABAP programs run here

• **Database Layer:** Data storage (SAP HANA, Oracle, DB2, etc.)

# 2. Getting Started with ABAP

## 2.1 SAP Development Environment

ABAP development is done in the SAP system using transaction codes. The main development tools include:

• **SE38:** ABAP Editor - For creating and editing ABAP programs

• **SE11:** ABAP Dictionary - For creating database tables, structures, data elements

• **SE80:** Object Navigator - Integrated development environment

• **SE37:** Function Builder - For creating function modules

• **SE24:** Class Builder - For object-oriented programming

## 2.2 Your First ABAP Program

Let's create a simple 'Hello World' program:

```
REPORT Z_HELLO_WORLD.

WRITE: 'Hello World!'.
WRITE: / 'Welcome to SAP ABAP Programming!'.
```

To execute this program: Save it, Activate it (Ctrl+F3), and Execute it (F8).

## 2.3 ABAP Naming Conventions

SAP uses specific naming conventions:

• **Customer Objects:** Start with Y or Z (e.g., Z_CUSTOMER_REPORT)

• **Standard SAP Objects:** Do not start with Y or Z

• **Local Variables:** Start with l_ or lv_ (e.g., lv_counter)

• **Global Variables:** Start with g_ or gv_ (e.g., gv_total)

• **Field Symbols:** Use <fs_> prefix (e.g., <fs_line>)

# 3. ABAP Basics and Syntax

## 3.1 ABAP Program Structure

A typical ABAP program consists of:

• **REPORT statement:** Declares the program name

• **Data declarations:** TABLES, DATA, CONSTANTS, TYPES

• **Events:** INITIALIZATION, START-OF-SELECTION, END-OF-SELECTION

• **Processing blocks:** FORM routines, Methods, Function modules

## 3.2 ABAP Statements

Key points about ABAP statements:

• Each statement ends with a period (.)

• ABAP is not case-sensitive (but uppercase is conventional)

• Comments start with * (entire line) or " (inline)

• Multiple statements can be chained using colons (:)

Example of statement chaining:

```
DATA: lv_name TYPE string,
      lv_age TYPE i,
      lv_city TYPE string.
```

## 3.3 Comments

```
* This is a full-line comment
DATA: lv_var TYPE i. " This is an inline comment
```

# 4. Data Types and Variables

## 4.1 Elementary Data Types

ABAP provides various built-in data types:

| Type | Description | Example |
|---|---|---|
| C | Character (text) | DATA: lv_name TYPE c LENGTH 50. |
| N | Numeric text | DATA: lv_zip TYPE n LENGTH 6. |
| D | Date (YYYYMMDD) | DATA: lv_date TYPE d. |
| T | Time (HHMMSS) | DATA: lv_time TYPE t. |
| I | Integer | DATA: lv_count TYPE i. |
| P | Packed number | DATA: lv_amount TYPE p DECIMALS 2. |
| F | Floating point | DATA: lv_float TYPE f. |
| STRING | Dynamic string | DATA: lv_text TYPE string. |
| XSTRING | Byte string | DATA: lv_hex TYPE xstring. |

## 4.2 Variable Declaration

Variables can be declared in multiple ways:

```
" Direct type specification
DATA: lv_name TYPE string,
      lv_age TYPE i VALUE 25.

" Reference to ABAP Dictionary
DATA: lv_customer TYPE kna1-kunnr.

" Using LIKE (refer to existing variable)
DATA: lv_copy LIKE lv_name.

" Constants
CONSTANTS: gc_pi TYPE p DECIMALS 2 VALUE '3.14'.
```

## 4.3 System Fields

ABAP provides system fields (SY-* fields) with runtime information:

• **SY-DATUM:** Current date

• **SY-UZEIT:** Current time

• **SY-UNAME:** Current user

• **SY-SUBRC:** Return code (0 = success)

• **SY-TABIX:** Current table index

• **SY-REPID:** Current program name

# 5. Control Structures

## 5.1 Conditional Statements (IF-ELSE)

```
IF lv_age GE 18.
  WRITE: 'Adult'.
ELSEIF lv_age GE 13.
  WRITE: 'Teenager'.
ELSE.
  WRITE: 'Child'.
ENDIF.
```

## 5.2 CASE Statement

```
CASE lv_grade.
  WHEN 'A'.
    WRITE: 'Excellent'.
  WHEN 'B'.
    WRITE: 'Good'.
  WHEN 'C'.
    WRITE: 'Average'.
  WHEN OTHERS.
    WRITE: 'Needs Improvement'.
ENDCASE.
```

## 5.3 Loops

### 5.3.1 DO Loop (Count-based)

```
DO 5 TIMES.
  WRITE: / 'Iteration', sy-index.
ENDDO.
```

### 5.3.2 WHILE Loop (Condition-based)

```
WHILE lv_counter LT 10.
  lv_counter = lv_counter + 1.
  WRITE: / lv_counter.
ENDWHILE.
```

### 5.3.3 LOOP AT (Internal Table)

```
LOOP AT it_customers INTO wa_customer.
  WRITE: / wa_customer-name.
ENDLOOP.
```

## 5.4 Comparison Operators

| Operator | Symbol | Description |
|---|---|---|
| EQ | = | Equal to |
| NE | <> | Not equal to |
| GT | > | Greater than |
| GE | >= | Greater than or equal |
| LT | < | Less than |
| LE | <= | Less than or equal |

# 6. Internal Tables

## 6.1 What are Internal Tables?

Internal tables are temporary tables in memory used to store and process data. They are similar to arrays in other programming languages but more powerful and flexible.

## 6.2 Types of Internal Tables

• **Standard Table:** Most flexible, accessed by index, allows duplicates

• **Sorted Table:** Automatically sorted, accessed by index or key

• **Hashed Table:** Fastest for key access, unique keys required

## 6.3 Declaring Internal Tables

```
" Define structure
TYPES: BEGIN OF ty_employee,
       empid TYPE i,
       name TYPE string,
       salary TYPE p DECIMALS 2,
     END OF ty_employee.

" Declare internal table
DATA: it_employees TYPE TABLE OF ty_employee,
      wa_employee TYPE ty_employee.

" Modern syntax (ABAP 7.4+)
DATA(it_emp) = VALUE ty_employee_tt( ).
```

## 6.4 Working with Internal Tables

### Adding Data:

```
" Using work area
wa_employee-empid = 1.
wa_employee-name = 'John Doe'.
wa_employee-salary = 50000.
APPEND wa_employee TO it_employees.

" Direct insertion
INSERT VALUE #( empid = 2 name = 'Jane Smith' salary = 55000 ) INTO TABLE
it_employees.
```

### Reading Data:

```
" Read by index
READ TABLE it_employees INTO wa_employee INDEX 1.

" Read by key
READ TABLE it_employees INTO wa_employee WITH KEY empid = 1.

" Check success
IF sy-subrc = 0.
  WRITE: / wa_employee-name.
ENDIF.
```

### Modifying Data:

```
wa_employee-salary = 60000.
MODIFY it_employees FROM wa_employee INDEX 1.
```

### Deleting Data:

```
DELETE it_employees INDEX 1.
DELETE it_employees WHERE salary LT 50000.
```

## 6.5 Useful Table Operations

```
" Get table size
DATA(lv_lines) = lines( it_employees ).

" Clear all entries
CLEAR it_employees.
" Or
REFRESH it_employees.

" Sort table
SORT it_employees BY salary DESCENDING.

" Delete duplicates
DELETE ADJACENT DUPLICATES FROM it_employees COMPARING name.
```

# 7. Modularization Techniques

## 7.1 Why Modularization?

Modularization helps in creating reusable, maintainable, and organized code. ABAP provides several modularization techniques:

## 7.2 Subroutines (FORM-ENDFORM)

```
" Define subroutine
FORM calculate_tax USING p_salary TYPE p
                   CHANGING p_tax TYPE p.
  p_tax = p_salary * '0.10'.
ENDFORM.

" Call subroutine
DATA: lv_salary TYPE p DECIMALS 2 VALUE 50000,
      lv_tax TYPE p DECIMALS 2.

PERFORM calculate_tax USING lv_salary
                    CHANGING lv_tax.
```

## 7.3 Function Modules

Function modules are global and can be called from any program. They are created using transaction SE37.

```
" Calling a function module
CALL FUNCTION 'Z_CALCULATE_DISCOUNT'
  EXPORTING
    iv_amount = lv_amount
  IMPORTING
    ev_discount = lv_discount
  EXCEPTIONS
    invalid_amount = 1
    OTHERS = 2.

IF sy-subrc <> 0.
  " Error handling
ENDIF.
```

## 7.4 Methods (Object-Oriented)

Methods are part of classes and are used in object-oriented programming. We'll cover this in detail in Chapter 10.

## 7.5 Includes

Include programs allow code reuse across multiple programs:

```
" In main program
INCLUDE z_common_declarations.
INCLUDE z_common_subroutines.
```

# 8. Database Operations

## 8.1 ABAP Dictionary (SE11)

The ABAP Dictionary is a central repository for data definitions. It contains database tables, structures, data elements, domains, and more.

## 8.2 Database Tables

Common SAP tables:

• **MARA:** Material Master

• **KNA1:** Customer Master

• **LFA1:** Vendor Master

• **VBAK:** Sales Document Header

• **EKKO:** Purchasing Document Header

• **BKPF:** Accounting Document Header

## 8.3 SELECT Statement

The SELECT statement retrieves data from database tables:

### Basic SELECT:

```
" Select single record
SELECT SINGLE * FROM kna1 INTO wa_customer
  WHERE kunnr = '1000'.

" Select multiple records
SELECT * FROM kna1 INTO TABLE it_customers
  WHERE land1 = 'US'.

" Select specific fields
SELECT kunnr name1 land1
  FROM kna1
  INTO TABLE it_customers
  WHERE land1 = 'US'.
```

### Modern SELECT (ABAP 7.4+):

```
SELECT kunnr, name1, land1
  FROM kna1
  INTO TABLE @DATA(it_customers)
  WHERE land1 = @lv_country
  UP TO 100 ROWS.
```

## 8.4 INSERT, UPDATE, DELETE

### INSERT:

```
INSERT ztable FROM wa_data.
" Or
INSERT ztable FROM TABLE it_data.
```

### UPDATE:

```
UPDATE ztable SET field1 = 'value'
  WHERE key_field = '1000'.

" Or update from work area
UPDATE ztable FROM wa_data.
```

### DELETE:

```
DELETE FROM ztable WHERE key_field = '1000'.
" Or
DELETE ztable FROM TABLE it_data.
```

## 8.5 Database Commit

Don't forget to commit your database changes:

```
" After INSERT/UPDATE/DELETE
COMMIT WORK.
```

# 9. Reports and ALV

## 9.1 Classical Reports

Classical reports are simple programs that display data using WRITE statements:

```
REPORT z_employee_report.

TABLES: pa0001. " HR Master Record

DATA: it_emp TYPE TABLE OF pa0001,
      wa_emp TYPE pa0001.

SELECT * FROM pa0001 INTO TABLE it_emp
  UP TO 100 ROWS.

WRITE: / 'Employee Report'.
ULINE.

LOOP AT it_emp INTO wa_emp.
  WRITE: / wa_emp-pernr, wa_emp-ename.
ENDLOOP.
```

## 9.2 Interactive Reports

Interactive reports allow user interaction through events like AT LINE-SELECTION:

```
AT LINE-SELECTION.
  " Code to execute when user clicks a line
  READ TABLE it_emp INTO wa_emp INDEX sy-tabix.
  WRITE: / 'Details for:', wa_emp-ename.
```

## 9.3 ALV (ABAP List Viewer)

ALV is a powerful tool for displaying data in a grid format with built-in features like sorting, filtering, and export to Excel:

```
REPORT z_alv_demo.

DATA: it_customers TYPE TABLE OF kna1,
      it_fieldcat TYPE slis_t_fieldcat_alv,
      wa_fieldcat TYPE slis_fieldcat_alv.

" Fetch data
SELECT * FROM kna1 INTO TABLE it_customers UP TO 50 ROWS.

" Build field catalog
wa_fieldcat-fieldname = 'KUNNR'.
wa_fieldcat-seltext_m = 'Customer Number'.
APPEND wa_fieldcat TO it_fieldcat.

wa_fieldcat-fieldname = 'NAME1'.
wa_fieldcat-seltext_m = 'Customer Name'.
APPEND wa_fieldcat TO it_fieldcat.

" Display ALV
CALL FUNCTION 'REUSE_ALV_GRID_DISPLAY'
  EXPORTING
    it_fieldcat = it_fieldcat
  TABLES
    t_outtab = it_customers.
```

## 9.4 Modern ALV (CL_SALV_TABLE)

```
DATA: lo_alv TYPE REF TO cl_salv_table.

cl_salv_table=>factory(
  IMPORTING
    r_salv_table = lo_alv
  CHANGING
    t_table = it_customers ).

lo_alv->display( ).
```

# 10. Object-Oriented ABAP (OO ABAP)

## 10.1 Introduction to OO ABAP

Object-Oriented ABAP introduces concepts like classes, objects, inheritance, and polymorphism. It's essential for modern SAP development.

## 10.2 Classes and Objects

```
" Define a class
CLASS lcl_employee DEFINITION.
  PUBLIC SECTION.
    DATA: empid TYPE i,
          name TYPE string.

    METHODS: constructor IMPORTING iv_id TYPE i iv_name TYPE string,
             get_details.

  PRIVATE SECTION.
    DATA: salary TYPE p DECIMALS 2.
ENDCLASS.

" Implement the class
CLASS lcl_employee IMPLEMENTATION.
  METHOD constructor.
    empid = iv_id.
    name = iv_name.
  ENDMETHOD.

  METHOD get_details.
    WRITE: / empid, name.
  ENDMETHOD.
ENDCLASS.

" Create and use object
DATA: lo_emp TYPE REF TO lcl_employee.

CREATE OBJECT lo_emp
  EXPORTING
    iv_id = 1
    iv_name = 'John Doe'.

lo_emp->get_details( ).
```

## 10.3 Key OO Concepts

• **Encapsulation:** Hiding internal details using PUBLIC, PROTECTED, and PRIVATE sections

• **Inheritance:** Creating new classes based on existing ones using INHERITING FROM

• **Polymorphism:** Redefining methods in subclasses using REDEFINITION

• **Interfaces:** Defining contracts that classes must implement

## 10.4 Inheritance Example

```
CLASS lcl_manager DEFINITION INHERITING FROM lcl_employee.
  PUBLIC SECTION.
    DATA: department TYPE string.
    METHODS: get_details REDEFINITION.
ENDCLASS.

CLASS lcl_manager IMPLEMENTATION.
  METHOD get_details.
    CALL METHOD super->get_details.
    WRITE: / 'Department:', department.
  ENDMETHOD.
ENDCLASS.
```

## 10.5 Interfaces

```
INTERFACE lif_printable.
  METHODS: print.
ENDINTERFACE.

CLASS lcl_document DEFINITION.
  PUBLIC SECTION.
    INTERFACES: lif_printable.
ENDCLASS.

CLASS lcl_document IMPLEMENTATION.
  METHOD lif_printable~print.
    WRITE: / 'Printing document...'.
  ENDMETHOD.
ENDCLASS.
```

# 11. Best Practices and Tips for ABAP Freshers

## 11.1 Coding Standards

• Use meaningful variable names (e.g., lv_customer_name instead of lv_cn)

• Follow SAP naming conventions (Y/Z for custom objects)

• Add comments for complex logic

• Use proper indentation for readability

• Avoid using SELECT * - specify fields explicitly

• Always check SY-SUBRC after database operations

• Use constants instead of hard-coded values

• Implement error handling properly

## 11.2 Performance Tips

• Use SELECT SINGLE when fetching one record

• Avoid SELECT in loops (use FOR ALL ENTRIES instead)

• Use database indexes wisely

• Minimize database calls by fetching data in bulk

• Use SORT and BINARY SEARCH for internal table operations

• Avoid nested loops when possible

• Use field symbols for better performance with internal tables

• Clear internal tables when no longer needed

## 11.3 Debugging Tips

• Use /h in command field to enter debug mode

• Set breakpoints by double-clicking line numbers

• Use watchpoints to monitor variable changes

• Check the debugger's call stack to trace program flow

• Use F5 (step into), F6 (step over), F7 (return) for navigation

• Utilize the 'Display' button to view variable values

## 11.4 Important Transaction Codes

| T-Code | Description |
|---|---|
| SE38 | ABAP Editor |
| SE11 | ABAP Dictionary |

| | |
|---|---|
| SE80 | Object Navigator |
| SE37 | Function Builder |
| SE24 | Class Builder |
| SE93 | Transaction Code Maintenance |
| SM30 | Table Maintenance |
| SE16 | Data Browser |
| ST22 | Runtime Error Analysis |
| SE09 | Transport Organizer |

# 12. Common ABAP Interview Questions for Freshers

## 12.1 Basic Concepts

### Q: What is ABAP?

A: ABAP (Advanced Business Application Programming) is a high-level programming language created by SAP for developing applications on the SAP platform.

### Q: What is the difference between APPEND and INSERT in internal tables?

A: APPEND adds a record at the end of the table, while INSERT adds a record at a specific index position.

### Q: What is SY-SUBRC?

A: SY-SUBRC is a system field that contains the return code of the last operation. 0 indicates success, non-zero values indicate errors or specific conditions.

### Q: What is the difference between TYPE and LIKE?

A: TYPE refers to a data type defined in ABAP Dictionary, while LIKE refers to an existing variable's type.

### Q: What are the different types of internal tables?

A: Standard tables, Sorted tables, and Hashed tables.

### Q: What is a field symbol?

A: A field symbol is a placeholder for a data object. It's similar to a pointer and is used for dynamic data references and better performance.

### Q: What is the difference between DELETE and CLEAR?

A: DELETE removes entries from an internal table, while CLEAR initializes a variable to its default value but doesn't remove table entries.

### Q: What is ALV?

A: ALV (ABAP List Viewer) is a tool for displaying data in a grid format with built-in features like sorting, filtering, and exporting.

### Q: What is the difference between classical and interactive reports?

A: Classical reports simply display data, while interactive reports allow user interaction through events like AT LINE-SELECTION.

### Q: What is a function module?

A: A function module is a reusable procedure that can be called from any ABAP program. It's created using transaction SE37.

## 12.2 Advanced Topics

### Q: What is the difference between CALL METHOD and functional method calls?

A: CALL METHOD is the traditional syntax, while functional method calls (using ->) are the modern, more concise syntax introduced in newer ABAP versions.

### Q: What is BAdI?

A: BAdI (Business Add-In) is an enhancement technique that allows adding custom functionality to standard SAP programs without modifying the original code.

### Q: What is the difference between synchronous and asynchronous RFC?

A: Synchronous RFC waits for the called function to complete before continuing, while asynchronous RFC continues processing without waiting.

### Q: What is a secondary index?

A: A secondary index is an additional index created on a database table to improve query performance for specific fields.

### Q: What is FOR ALL ENTRIES?

A: FOR ALL ENTRIES is used to fetch data from a database table based on entries in an internal table, avoiding SELECT statements in loops.

## Learning Resources

To continue your ABAP learning journey, explore these resources:

• **SAP Learning Hub:** Official SAP training platform

• **SAP Community:** Forums and blogs from SAP experts

• **SAP Help Portal:** Official documentation

• **OpenSAP:** Free online courses

• **ABAP Development Tools (ADT):** Eclipse-based IDE for modern ABAP

• **SAP Fiori:** Learn about modern SAP UX

• **Practice:** Set up a free SAP system or use SAP Cloud Platform trial

## Conclusion

This guide covers the essential concepts of SAP ABAP that every fresher should know. ABAP is a powerful language with extensive capabilities for enterprise application development. As you continue learning, focus on:

• Building strong fundamentals in ABAP syntax and data structures

• Practicing with real-world scenarios and use cases

• Understanding SAP's best practices and coding standards

• Exploring modern ABAP features and tools

• Learning integration with other SAP modules

• Staying updated with SAP's evolving technology stack

Remember, becoming proficient in ABAP takes practice and hands-on experience. Don't hesitate to experiment, make mistakes, and learn from them. The SAP community is vast and supportive, so leverage available resources and connect with fellow developers.

**Good luck on your SAP ABAP journey!**