ABAP Important Notes

In ABAP there are 6 key words to write program in a Structured manner they are

1. Declerative keywords
2. Database keywords
3. Controlling Keywords
4. Defination keywords
5. Event keywords
6. Operation keywords

1. Declerative keywords

Declerative keywords are used to declare the different types of variables

- DATA:
- TYPE:
- CONSTANTS:
- TABLES:
- TYPES:

- **DATA:** Used to declare a variable.

SYNTAX:

```
DATA <variable_name> TYPE <data_type>.
```

EXAMPLE:

```
DATA lv_name TYPE string.  " LV REFERS LOCAL VARIABLE
```

- **TYPE:** Used to define custom data types or structures.

SYNTAX:

```
TYPE: <type_name> TYPE  <data_type>.
```

EXAMPLE:

```
types ty_employee TYPE STANDARD TABLE OF string.
```

- **CONSTANTS:** In ABAP, constraints are often set in Data Dictionary, but you can also define domain-specific constraints in Data Types.

  Syntax:

  ```
  CONSTANTS: stant_name> TYPE <data_type> VALUE
  <fixed_value>.
  ```

  Example:

  ```
  DATA lv_code TYPE c LENGTH 10.
  ```

- **TABLES:** Used to declare work areas for database tables (primarily in older styles).

  Syntax:

  ```
  TABLES <table_name>.
  ```

  Example:

  ```
  TABLES zemployee.
  ```

- **TYPES:** To define data types and structures.

  Syntax:

  ```
  types <type_name> TYPE <template_type>.
  ```

  Example:

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

2. Database keywords: Database Keywords in ABAP presented in the same structured manner, with syntax, example, and comments on common naming or Behavior.

- Create:
- Select:
- Insert:
- Modify:
- Update:
- Delete:

- **Create:** Creates an instance of a class.

  SYNTAX:

```
CREATE OBJECT <reference_variable> [EXPORTING <parameters>].
```

  EXAMPLE:

```
DATA lo_car TYPE REF TO zcl_car.
CREATE OBJECT lo_car EXPORTING name = 'BMW'.
```

- **Select:** Used to fetch data from a database table.

  SYNTAX:

```
SELECT <fields> FROM <table> INTO <work_area> [WHERE <condition>].
```

  EXAMPLE:

```
SELECT * FROM mara INTO lv_mara WHERE matnr = '1000'.
```

- **INSERT:** Used to insert new records into a database table.

  SYNTAX:

```
AbabINSERT <work_area> INTO <table>.
```

  EXAMPLE:

```
INSERT wa_employee INTO zemployee.
```

- **MODIFY:** Used to insert or update records in a database table (upsert)

  SYNTAX:

  ```
  MODIFY <table> FROM <work_area>.
  ```

  EXAMPLE:

  ```
  MODIFY zemployee FROM wa_employee.
  ```

- **UPDATE:** Used to update existing records in a database table.

  SYNTAX:

  ```
  UPDATE <table> SET <field1> = <value1> [<field2> = <value2>
  ...] WHERE <condition>.
  ```

  EXAMPLE

  ```
  UPDATE zemployee SET name = 'Sudheer' WHERE pernr = '1234'.
  ```

- **DELETE:** Used to delete records from a database table.

  SYNTAX:

  ```
  DELETE FROM <table> WHERE <condition>.
  ```

  EXAMPLE:

  ```
  DELETE FROM zemployee WHERE pernr = '1234'.
  ```

3. CONTROLLING KEYWORDS: Used to control the flow of the program

- If …. Elseif …. Else …. Endif
- Case …. When …. Endcase
- Loop …. Endloop
- Do ……. Enddo
- While … Endwhile

- **If …. Elseif …. Else …. Endif:** Used for conditional branching.

SYNTAX:

```
IF <condition>.
  <statements>.
ELSEIF <condition>.
  <statements>.
ELSE.
  <statements>.
ENDIF.
```

EXAMPLE:

```
IF lv_age > 18.
  WRITE 'Adult'.
ELSE.
  WRITE 'Minor'.
ENDIF.
```

- **CASE ... WHEN ... ENDCASE:** Used for multiple conditions branching.

  SYNTAX:
  ```
  CASE <variable>.
    WHEN <value1>.
      <statements>.
    WHEN <value2>.
      <statements>.
    WHEN OTHERS.
      <statements>.
  ENDCASE.
  ```

  EXAMPLE
  ```
  CASE lv_color.
    WHEN 'RED'.
      WRITE 'Stop'.
    WHEN 'GREEN'.
      WRITE 'Go'.
    WHEN OTHERS.
      WRITE 'Wait'.
  ENDCASE.
  ```

- **Loop …. Endloop:** Loop through internal tables or ranges.

  SYNTAX:
  ```
  LOOP AT <internal_table> INTO <work_area>.
    <statements>.
  ENDLOOP.
  ```

  EXAMPLE:
  ```
  LOOP AT it_numbers INTO lv_num.
    WRITE lv_num.
  ENDLOOP.
  ```

- **Do …. Enddo:** Do used to execute a block of code a specified number of times(or until exited).

  SYNTAX:
  ```
  DO [n TIMES].
    " Statements executed repeatedly
  ENDDO.
  ```
  EXAMPLE:
  ```
   DO 5 TIMES.
    WRITE: / 'Hello ABAP!'.
  ENDDO.
  ```

- **While ….. Endwhile:** Used to execute a block of code repeatedly while a specified condition is true.

  SYNTAX:
  ```
  WHILE <condition>.

     " Statements to execute

  ENDWHILE.
  ```
  EXAMPLE :
  ```
  DATA counter TYPE i VALUE 1.

  WHILE counter <= 5.

    WRITE: / 'Counter value:', counter.

    counter = counter + 1.

  ENDWHILE.
  ```

4. Definition keywords: Defining keywords are used to create modular blocks that encapsulate code into reusable units or modules. These blocks end with specific END keywords to mark their boundaries.

- Form ……. Endform
- Function ….Endfunction
- Module ….. Endmodule
- Method ….. Endmethod

- **Form ……. Endform:** Defines a subroutine (a reusable procedural block).

  Syntax:

```
FORM <form_name>.
  " Code to execute
ENDFORM.
```

  Example:

```
FORM display_message.
  WRITE 'Hello from subroutine'.
ENDFORM.
```

- **FUNCTION ……Endfunction:** A function module in ABAP is a reusable procedure with a defined interface, used for modular and global code across programs.

  SYNTAX:

```
FUNCTION <module_name>.
      " Code logic
ENDFUNCTION.
```

  EXAMPLE:

```
FUNCTION Z_HELLO.
  WRITE 'Hello from function module'.
```

```
ENDFUNCTION.
```

- **Module ….. Endmodule:** Defines a dialog module in ABAP, which contains processing logic for SAP screens during events like PBO (Process Before Output) and PAI (Process After Input).

  Syntax:

```
MODULE <module_name> INPUT.
  " Code to process user input
ENDMODULE.

MODULE <module_name> OUTPUT.
  " Code to prepare screen output
ENDMODULE.
```

  Example:

```
MODULE status_check INPUT.
  IF sy-ucomm = 'SAVE'.
    WRITE 'Save button pressed'.
  ENDIF.
ENDMODULE.

MODULE status_display OUTPUT.
  WRITE 'Welcome to screen'.
ENDMODULE.
```

- **Method ….. Endmethod:** is used to define the implementation of a method inside a class. Methods represent the behavior or actions that can be performed by objects of the class.

  Syntax:

```
METHOD <method_name>.
  " Method code here
```

```
ENDMETHOD.
```

Example:

```
CLASS lcl_greet DEFINITION.
  PUBLIC SECTION.
    METHODS say_hello.
ENDCLASS.

CLASS lcl_greet IMPLEMENTATION.
  METHOD say_hello.
    WRITE 'Hello from method!'.
  ENDMETHOD.
ENDCLASS.
```

5. Event keywords: Event keywords are used to define event blocks that manage program flow based on specific runtime occurrences. These event blocks do not require explicit end statements; they end automatically when another block starts.

- Starting-of-selection:
- At selection-screen:
- At user-command:
- End-of-selection:
- Initialization:
- Top-of-page:
- End-of-page:

- **Start-of-selection:** Main processing block executed after selections screen.
  Syntax:
  ```
     START-OF-SELECTION.
   " Main logic
  ```
  Example:
  ```
     START-OF-SELECTION.
     WRITE 'Main processing'.
  ```

- **At selection-screen:** Triggers on selection screen input/validation
  Syntax:
  ```
  AT SELECTION-SCREEN.
  " Validation code
  ```
  Example:
  ```
  text
  AT SELECTION-SCREEN.
    MESSAGE 'Screen checked'.
  ```

- **At user-command:** Handles user actions (buttons, menus).

  Syntax:

  ```
  text
  ```

```
  AT USER-COMMAND.
  " Command processing
```

Example:

```
AT USER-COMMAND.
   IF sy-ucomm = 'SAVE'.
     WRITE 'Save pressed'.
ENDIF.
```

- **End-of-selection:** Runs after main processing completes.

  Syntax:

  ```
  text
  END-OF-SELECTION.
    " Final output
  ```

  Example:

  ```
  text
  END-OF-SELECTION.
    WRITE 'Processing complete'.
  ```

- **Initialization:** Sets defaults before selection screen displays.

  Syntax:

  ```
    INITIALIZATION.
     " Default values
  ```

  Example:

  ```
  INITIALIZATION.
    p_date = sy-datum.
  ```

- **TOP-OF-PAGE:** Prints list headers on new pages.

Syntax:

```
TOP-OF-PAGE.
 " Header content
```

Example:

```
TOP-OF-PAGE.
  WRITE 'Report Header'.
```

- **END-OF-PAGE**: Prints list footers on each page.

Syntax:

```
END-OF-PAGE.

" Footer content
```

Example:

```
END-OF-PAGE.

  WRITE 'Page Footer'.
```

6. OPERATION KEYWORDS: perform data processing like arithmetic, string manipulation, and assignments. They handle core operations on variables and data.

- Write
- Move
- Add
- Substract
- Multiply
- Divide
- Concatenate
- Clear

- **WRITE:** Outputs data to screen/list.

  Syntax:

  ```
  WRITE field.
  ```

  Example:

  ```
  WRITE 'ABAP Tutorial'.
  ```

- **MOVE:** Assigns data between variables.

  Syntax: `MOVE source TO target.`

  Example:

  ```
  DATA: a TYPE i VALUE 100, b TYPE i.

  MOVE a TO b.

  WRITE b.  "Output: 100"
  ```

- **ADD:** Adds numeric values.

 Syntax:

```
    ADD operand1 TO operand2.
```

Example:

```
    DATA: a TYPE i VALUE 10, b TYPE i VALUE 20.
    ADD a TO b.
    WRITE b.  "Output: 30"
```

- **SUBTRACT:** Subtracts numeric values.

 Syntax:

```
     SUBTRACT operand1 FROM operand2.
```

Example:

```
    DATA: x TYPE i VALUE 50, y TYPE i VALUE 30.
    SUBTRACT y FROM x.
    WRITE x.  "Output: 20"
```

- **MULTIPLY:** Multiplies numeric values.

Syntax: `MULTIPLY operand1 BY operand2.`

Example:

```
DATA: m TYPE i VALUE 5, n TYPE i VALUE 8.
MULTIPLY m BY n.
WRITE n.  "Output: 40”
```

- **DIVIDE:** Divides numeric values.

  Syntax: `DIVIDE operand1 BY operand2.`

  Example:

```
DATA: d TYPE i VALUE 100, e TYPE i VALUE 4.
DIVIDE d BY e.
WRITE d.  "Output: 25"
```

- **CONCATENATE:** Joins strings.

  Syntax: `CONCATENATE str1 str2 INTO result.`

  Example:

```
     DATA: s1 TYPE string VALUE 'ABAP', s2 TYPE string VALUE
‘Tutorial’.
     CONCATENATE s1 s2 INTO DATA(result).
     WRITE result.  "Output: ABAPTutorial"
```

- **CLEAR:** Resets variable to initial value.

  Syntax: `CLEAR variable.`

  Example:

```
DATA: num TYPE i VALUE 999.
CLEAR num.
WRITE num.  "Output: 0"
```