From the textbook—Chapter Two

ABAP Workbench
Object Navigator (SE80)
“Forward navigation”
Packages
Transports
Creating database tables
Field names, data elements, and data types

**Data element**: “object that describes the data type and semantic meaning of a table field” (SAP online glossary)

**Domain**: defines valid value ranges for fields. Similar fields can be grouped into a domain. Changing that domain changes all the fields.

Keyword Documentation
ABAP Variables and Data Types

**Data type**: description of the kind of data a variable may hold and the range of acceptable values based on storage allocated.

Technical and (potentially) semantic meaning

**Data object**: actual variable or constant (of a stated type) that has been defined.

Complete Data Types
- Fixed size, specified format data storage.

Incomplete Data Types
- Storage size can vary, so must be set upon variable declaration.

ABAP Standard Data Types

<table>
<thead>
<tr>
<th>Symbol</th>
<th>Type Description</th>
<th>Example</th>
</tr>
</thead>
<tbody>
<tr>
<td>i</td>
<td>integer</td>
<td>4 byte whole number +/- 2.1 billion</td>
</tr>
<tr>
<td>f</td>
<td>float</td>
<td>8 bytes, 15-16 significant digits</td>
</tr>
<tr>
<td>c</td>
<td>string</td>
<td>up to 65k characters</td>
</tr>
<tr>
<td>n</td>
<td>numeric string</td>
<td>up to 65k characters (non-math number)</td>
</tr>
<tr>
<td>string</td>
<td>string</td>
<td>dynamic length up to 2 GB long!</td>
</tr>
<tr>
<td>xstring</td>
<td>hex string</td>
<td>dynamic length byte sequence</td>
</tr>
<tr>
<td>x</td>
<td>byte sequence</td>
<td>up to 65k bytes</td>
</tr>
<tr>
<td>d</td>
<td>date</td>
<td>8 characters in form YYYYMMDD</td>
</tr>
<tr>
<td>t</td>
<td>time</td>
<td>6 characters in from HHMMSS</td>
</tr>
<tr>
<td>p</td>
<td>packed number</td>
<td>precise whole or floating number up to 16 bytes</td>
</tr>
</tbody>
</table>

Bold italics indicate incomplete data types (size set on declaration)
## Variable Declarations

**DATA varname TYPE [type specification]**

<table>
<thead>
<tr>
<th>Declaration</th>
<th>Example</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>DATA age TYPE i.</td>
<td>&quot;complete data type&quot;</td>
<td></td>
</tr>
<tr>
<td>DATA siblings TYPE i VALUE 7.</td>
<td>&quot;w/ initialization&quot;</td>
<td></td>
</tr>
<tr>
<td>DATA state TYPE c LENGTH 2 VALUE 'TN'.</td>
<td>&quot;incomplete&quot;</td>
<td></td>
</tr>
</tbody>
</table>

Use of chained statements (more typical):

<table>
<thead>
<tr>
<th>Chained Statement</th>
<th>Example</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>DATA: age TYPE i,</td>
<td></td>
<td></td>
</tr>
<tr>
<td>siblings TYPE i VALUE 7.</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Other variations (considered obsolete) exist.

LIKE can be used to declare a variable based on a prior declaration

<table>
<thead>
<tr>
<th>LIKE Declaration</th>
<th>Example</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>DATA state2 LIKE state.</td>
<td>&quot;only data type 'copied'&quot;</td>
<td></td>
</tr>
</tbody>
</table>

When specifying default values for numeric data with decimals, the value must be placed within single quotes.

## Declaring your own data types—program local

TYPES allows declaring **local data types** that are more specific than standard types.

**TYPES typename TYPE [type specification]**

This can be used to give a standard type a more descriptive name for your application or to more specifically define variable composition.

<table>
<thead>
<tr>
<th>TYPES Declaration</th>
<th>Example</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>TYPES: userval TYPE i,</td>
<td></td>
<td></td>
</tr>
<tr>
<td>usercode TYPE n LENGTH 10,</td>
<td></td>
<td></td>
</tr>
<tr>
<td>rate TYPE p LENGTH 3 DECIMALS 2.</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

TYPES cannot have a user-specified default value.

TYPES declarations are local to program. Global type declarations are possible through use of the **ABAP Dictionary**, allowing type management across entire system.
Declaring your own data types—global

ABAP Dictionary (SE11)
Enter name of type to be created in “Data type” field.
  To avoid name collision for class, start your type names with Znn_
Select Create and choose the class of data type.
Describe the data type and provide its technical specification.
Specify how the field will be labeled in reports.
Activate.
Use in programs.

Can generate a list of all repository elements that use the data type.
  Choose Where-Used List icon from SE11 initial screen.

What is the value of defining and using global data types?

Quick Practice
Constants

CONSTANTS allows specification of fixed data objects.

CONSTANTS name TYPE type VALUE value.

| CONSTANTS a_const TYPE d VALUE '19681214'. |
| CONSTANTS b_con TYPE p LENGTH 3 DECIMALS 2 VALUE '37.46'. |

A number of system-maintained constants exists. These are all within structure SYST. Can see list through dictionary, data type SYST.

Reference values by using the identifier SY-COMPONENT. For example, SY-MANDT will display the current client number.

Arithmetic and assignment

Valid arithmetic operators: +, -, *, /, ** (exponentiation), DIV (integer division), MOD.

Parenthesis can be used to set order of precedence.

Operators and parentheses are keywords and must be separated from other statement elements by at least one space.

Assignment syntax:

MOVE var2 TO var1.
var1 = var2.
varx = 3 + 7 * 2.

If variables of different type, automatic type conversion attempted.

CLEAR varx. Resets varx to default type-related value.
Writing output

**WRITE** is used for basic output.

**WRITE** 10 'Output'.
- Output is written beginning in column 10.

**WRITE** (3) 'Output'.
- Output is written in field of size 3. * used to show number truncation (if any). String truncation note noted.

**WRITE** (*) 'Output'
- Output is written in field of sufficient size, without extra spacing.

**WRITE** 10(3) 'Output'.
- Combination of above techniques.

**WRITE** / 'Output'.
- Output written on the next line

**WRITE** /10(3) 'Output'.
- Combination of above techniques.

Parameters

**PARAMETERS** prompts the user for runtime value at program start. (Called a selection screen.) Syntax is the same as **DATA**.

**PARAMETERS** age **TYPE** i.

**PARAMETERS**: var1 **TYPE** c **LENGTH** 8,
- var2 **TYPE** i.

Parameter name must be 8 characters or less.
User is prompted with the parameter name.
**DEFAULT** can be used to specify a default value in the field.

**PARAMETERS** var1 **TYPE** c **LENGTH** 8 **DEFAULT** 'NONE'.
Quick Practice

Literal text

ABAP is multilingual. Coding in string literals (prompting, etc.) defeats language independence and should only be done in testing.

The solution for this is **text symbols**.

Text symbols belong to each program in the **text pool** for that program.

The various texts are placed in the text pool and assigned a 3-character alphanumeric code (xxx).

This code is then used by specifying either

- `TEXT-xxx` instead of the literal.
- `'literal'(xxx)` where literal is the message in the native language.

Access the text pool to define text symbols by either:

- From editor select **Goto** → **Text Elements** → **Text Symbols**
- Using the syntax above to reference a text symbol in coding. Then double click on the text symbol entered.

**Goto** → **Translation** will translate the text symbols.
Logical Expressions

Logical expressions evaluate to true or false. In ABAP, logical expression cannot be assigned to variables (no boolean data type).

Logical expressions are used in conditional code sequences.

<table>
<thead>
<tr>
<th>Relational Operators</th>
<th>Boolean Operators</th>
</tr>
</thead>
<tbody>
<tr>
<td>Operator</td>
<td>Operation</td>
</tr>
<tr>
<td>=, EQ</td>
<td>Is equal to</td>
</tr>
<tr>
<td>&lt;&gt;, NE</td>
<td>Is not equal to</td>
</tr>
<tr>
<td>&lt;, LT</td>
<td>Is less than</td>
</tr>
<tr>
<td>&gt;, GT</td>
<td>Is greater than</td>
</tr>
<tr>
<td>&lt;=, LE</td>
<td>Is less than or equal to</td>
</tr>
<tr>
<td>&gt;=, GE</td>
<td>Is greater than or equal to</td>
</tr>
</tbody>
</table>

Conditional code execution: IF, ELSEIF, ELSE

IF logical_expression.  
  1 or more statements.  
ENDIF.

IF logical_expression.  
  1 or more statements.  
ELSE.  
  1 or more statements.  
ENDIF.

IF logical_expression.  
  1 or more statements.  
ELSEIF logical_expression.  
  1 or more statements.  
ENDIF.
Conditional code execution: CASE

CASE data_object.
  WHEN value.
    1 or more statements.
  WHEN value.
    1 or more statements.
  WHEN OTHERS.
    1 or more statements.
ENDCASE.

CASE var_a.
  WHEN 1.
    WRITE \ "uno".
  WHEN 2.
    WRITE \ "dos".
  WHEN OTHERS.
    WRITE \ "other".
ENDCASE.

Quick Practice
Using the Debugger to test programs

Program must be saved and activated before debugging.
Set a breakpoint indicating where you want the program to stop in the execution and display the debugger.
Use Direct Processing to begin execution. Program runs to breakpoint.
In the source code display, double click any variable you wish to watch.
Add/remove additional breakpoints by single clicking to left of line.
Step through code using controls in upper left—single step, execute, return, continue (runs until next breakpoint).
Watched variable values can be changed.
A watchpoint can be set of a variable and the program will run until that variable's value changes

Looping

ABAP supports definite iteration, pre-test iteration, and post-test iteration.

```
DO value TIMES.
  1 or more statements.
ENDDO.

WHILE condition.
  1 or more statements.
ENDWHILE.

DO.
  1 or more statements.
  IF abort_condition.
    EXIT.
  ENDIF.
ENDDO.
```

Within loops sy-index is a system managed loop counter.
Exiting Loops

EXIT: Loop exits immediately. Jump to statement that follows loop block.

CONTINUE: Restart next loop iteration.

```
WHILE sy-index < 10.
  WRITE / sy-index.
  IF sy-index < 3.
    CONTINUE.
  ENDIF.
  WRITE 'After ENDIF'.
ENDWHILE.
```

CHECK condition: Restart next loop iteration (continue) if false.

```
WHILE sy-index < 10.
  WRITE / sy-index.
  CHECK sy-index < 3.
  WRITE 'After CHECK'.
ENDWHILE.
```

Quick Practice