ABAP 101

New!

Exercises

Beginner - Starting from scratch

Learn how to create advanced ABAP applications
by hands on experience starting from scratch

- Learn how to use **Data Types and Data Objects**
- Create executable programs using **Imperative Logic**
- Split your program into many **Form Routines**
- Build **Selection Screens** to get user's input

Fábio Pagoti - Flávio Furlan - Jaime Freitas

ABAP 101 Exercises - Beginner Starting from scratch

Learn how to create advanced ABAP applications
by hands on experience starting from scratch

- Learn how to use **Data Types and Data Objects**
- Create executable programs using **Imperative Logic**
- Split your program into many **Form Routines**
- Build **Selection Screens** to get user's input

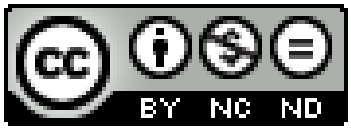
Freitas – Furlan – Pagoti

Jaime Freitas, Flávio Furlan and Fábio Pagoti

ABAP 101 Exercises - Beginner - Starting from scratch

Notes on Usage

Based on a work at http://abap101.com/ebook



Contents

Foreword by Fábio Pagoti

I read many technical books. Most of them are related with programming and not always ABAP. Usually the authors dedicate their books to their wife and kids. It seems like in order to become an author you must become a husband and a father beforehand.

So you might say I took the wrong direction somewhere as I'm writing my first book without even being engaged. Well, definitely if it weren't my parents I would never have written this book so I thank them on the first place.

Nevertheless, I do have other people to thank. First and most important, Flávio Furlan, a former work colleague and a permanent friend. We idealized this book long ago and after many conversations (some on Skype during almost the whole dawn) we could see a good progress.

Firstly this book would be written in Portuguese, as it's our native language and due to the lack of good SAP learning resources on such language. But we would rather reach as many people as possible writing it in English and making it free of charge (or at least asking you to pay with a single Tweet or Share on Facebook or LinkedIn).

I also must thank Jaime, who was one of my best ABAP students as he kept studying even after his course had finished. He patiently was the very first person to complete all 101 exercises. Moreover, he did a great job reviewing each question and giving awesome hints on how to make each question shorter and clearer.

Last but not least I thank you for reading these words and for having no shame on start something from scratch. I really hope this book can be your first steps on a great professional experience as an ABAP developer.

Fábio Pagoti

ABAP Consultant and Trainer @ Ka Solution

Foreword by Flávio Furlan

I taught ABAP for more than 5 years. Our training was based on four weeks of theory and simple exercises and one more week when students were requested to solve more difficult exercises. After all those years, I can assure to say that our students learn the basics on theory-exercise weeks and start to master in the last week, doing exercises by themselves.

You can read the help about WRITE, CONCATENATE and SELECT, but you will only learn when you use it. That is the spirit of this book! Practice, practice and practice!

When I was studying for college admission, my brother, a former ITA engineer (Aeronautics Institute of Technology in English, one of the most difficult college admissions in Brazil) used to show me several completed exercise books. 100% completed! Exercise by exercise. Step by step. That is what successful professionals have in common: hard work.

What you have in your hands looks silly in the beginning, but it gets more difficult and some exercises could represents truly challenges. Do not give up! If you are a beginner ABAPer do a favor to yourself, start from the exercise one and go further until the last one. It worth the effort!

Our first idea was just to present the exercises, but thanks to Fábio Pagoti and Jaime Freitas you can also count with solutions. Remember: it's not a crosswords book. There is no proud to complete it checking the answers. Don't lose your time checking the answers before actually solve (see that I didn't said try). Consider the words from Yoda Master for each exercise "Try not. Do... or do not. There is no try".

When you finish the last one, please generate a ZIP file with all solutions and e-mail me with subject "Take it, Furlan!". I dare you!

I really hope you enjoy and learn a lot with that book!

I also like to thank Fábio and Jaime! They really did a hard work to finish that! Thank you guys!

Flávio Furlan

Technical Architect @ Nestlé

Foreword by Jaime Freitas

The ebook “*ABAP 101 Exercises - Beginner - Starting from scratch*” is more than an extra learning resource for ABAP freshmen. It's also a reusable guide and an opportunity to review many techniques which are used on a daily basis during the development of Z programs or when enhancing standard SAP Programs.

Without a shadow of a doubt, no matter if you are just starting or if you have been working for some time with ABAP, this ebook deserves your attention.

Here, you will find the application of many concepts in an objective and dynamic way. This concepts can be consulted or deepened by you later so you can address day-to-day doubts related with a specific ABAP command, its use and variations.

I had access to this learning resource during its development phase when I had just finished an ABAP course. Particularly, it was very useful for reviewing concepts which I learned during the course. I truly believe this material will be useful in your daily job, being a young learner like me or someone already inside on SAP world.

Jaime Freitas

Senior System Analysis @ Lojas Riachuelo

Introduction

ABAP is not a difficult programming language to learn. Actually, it makes boring tasks like database queries, drawing a screen and handling user input very easy.

However, as in any other learning process, practice is essential. The whole idea of this e-book is to focus on the first steps you will need before deeping inside more complex ones.

Where is the theory?

Different than other books you might have read, this book only contains exercises. "How will I learn without theory?", you might ask. Definitely you won't. However times on which there was no documentation available for those who were about to work with SAP software have gone.

Nowadays you can learn ABAP by reading SAP Press books, downloading SAP Trial Versions, using ABAP Help, reading all different kinds of content from SAP Community Network, following other sites and blogs as abap101.com (if you speak Portuguese) or ZEvolving.com (if you speak English). Last but not least, there is Google. We recommend a mix of these information sources during the time you are doing all exercises described in this e-book.

Are 101 exercises enough?

101 exercises are a lot. However, don't expect them to be all you need in order to transform yourself into an ABAP developer. They are just an excellent starting point. In these exercises you will create executable programs using procedural logic (form routines). Although this is still entirely possible, be alert that development for SAP products is way more than that.

Nowadays, there are many other ways to create applications for SAP products (some don't even use ABAP, like OpenUI5). Also, object orientation becomes more and more used by SAP and its customers.

The authors of this book are totally willing to create other books like the one you are reading now. These other books might focus on anything else related with ABAP development: Module Pool programs, object orientation, SAP query, Data Dictionary, Smartforms, Enhancements, Web Dynpro or a mix of those and others. In order to happen that, your feedback is essential.

You can visit ABAP101's contact page to give your feedback to us. We would appreciate a lot such help.

What you should have/know before starting the exercises

- Access to a SAP server with development authorization (including SAP Logon or ABAP in Eclipse if you use a recent version of SAP NetWeaver)
- What ABAP is and its purpose
- Any programming language experience (if you don't have prior experience, study each exercise at least twice)
- SAP navigation (opening transactions, using the menu, etc)
- How to create an empty program (using SE38, SE80 or even Eclipse)

Answers

After having everything ready you are good to start the exercises. On the very first one you might wonder "Where do I start?". You can refer to the ABAP help but it might be hard to find what you need. That is the main reason why we have included answer and solutions for all exercises.

That's important - be honest with yourself! Even if you had to check the answer before doing yours, try to understand it (but always try to do first without the answer). Don't go further before reading all documentation available for each new keyword. Also, please understand that we always offer **possible** answers. If you have a totally different one, that's great! Another good exercise is to study the pros and cons of each possible one.

We also recommend to always look the answer before going to the next exercise, even if you don't need it to code a valid solution. By doing this you will also learn by looking at other person's code, which is always a good way to learn.

Exercises guidelines

To make this e-book useful for as many people as possible:

- SAP ERP tables are not used. In other words we always use the flight model and not tables like MARA, VBAK, BSEG, etc
- Exercises order is not based on difficult level but on what you should know before going further. **Don't skip exercises!!!**

General Instructions

- To make you organized, create a local package for everything you will create
- All exercises should be done
- The source code should be active
- Name your programs as Z_ABAP101_NNN, where NNN is the number of exercise

Part I - Data Types and Objects (1 - 40)

Declare a TYPE as a character with 10 positions.

Solution:

```
REPORT z_abap101_001.

TYPES customer_name TYPE c LENGTH 10.
```

Declare an integer.

Solution:

```
REPORT z_abap101_002.

DATA number_of_employees TYPE i.
```

Declare a type as a number with 7 positions.

Solution:

```
REPORT z_abap101_003.

TYPES number_of_unpaid_invoices TYPE n LENGTH 7.
```

Declare a date type.

Solution:

```
REPORT z_abap101_004.

TYPES creation_date TYPE d.
```

Declare a time type.

Solution:

```
REPORT z_abap101_005.

TYPES last_changed_at TYPE t.
```

Declare a structure type with 5 fields, each field with the same types from exercises 1 to 5.

Solution:

```
REPORT  z_abap101_006.

TYPES customer_name TYPE c LENGTH 10.

DATA number_of_employees TYPE i.

TYPES number_of_unpaid_invoices TYPE n LENGTH 7.

TYPES creation_date TYPE d.

TYPES last_changed_at TYPE t.

TYPES: BEGIN OF customer_structure,
name TYPE  customer_name,
n_employees LIKE number_of_employees,
unpaid_invoices TYPE number_of_unpaid_invoices,
creation_date TYPE d,
last_changed_at TYPE t,
END OF customer_structure.
```

Notes:

: (semicolon) is recommend in this case

name - This is a char with length 10

n_employees - This component (called ‘n_employees’ has the same type of the variable, but it is not reusable)

Declare a type using the global structure SFLIGHT.

Solution:

```
REPORT z_abap101_007.

TYPES same_type_of_sflight TYPE SFLIGHT.
```

Notes:

Double click on SFLIGHT

It's a global definition (it's inside the repository)

This is a TYPE, not a variable.

Declare a structure type with the following components of the global structure SFLIGHT: CARRID, CONNID, FLDATE, PRICE, CURRENCY, PLANETYPE, SEATSMAX and SEATSOCC.

Solution:

```
REPORT  z_abap101_008.

* 1st Option - Declaring several TYPES

TYPES BEGIN OF some_components_sflight.
TYPES carrid    TYPE sflight-carrid.
TYPES connid    TYPE sflight-connid.
TYPES fldate    TYPE sflight-fldate.
TYPES price     TYPE sflight-price .
TYPES currency  TYPE sflight-currency .
TYPES planetype TYPE sflight-planetype.
TYPES seatsmax  TYPE sflight-seatsmax .
TYPES seatsoccupied  TYPE sflight-seatsocc . " Different name for a component
TYPES END OF some_components_sflight.

* 2nd Option - Reusing TYPES keyword

TYPES: BEGIN OF some_components_sflight_2, " Semicolon ( : ) after TYPES
carrid    TYPE sflight-carrid, " Comma after each component
connid    TYPE sflight-connid,
fldate    TYPE sflight-fldate,
price     TYPE sflight-price,
currency  TYPE sflight-currency,
planetype TYPE sflight-planetype,
seatsmax  TYPE sflight-seatsmax,
seatsocc  TYPE sflight-seatsocc,
END OF some_components_sflight_2.
```

Declare a structure type with the following components of the global structure SBOOK: CARRID, CONNID, FLDATE, BOOKID, CUSTOMID.

Solution:

```
REPORT  z_abap101_009.

TYPES: BEGIN OF flight_booking,
carrid   TYPE sbook-carrid   ,
connid   TYPE sbook-connid   ,
fldate   TYPE sbook-fldate   ,
bookid   TYPE sbook-bookid   ,
customid TYPE sbook-customid ,
END OF flight_booking.
```

Declare a structure containing all the fields mentioned in exercises 8 and 9. Check it using the ABAP Debugger.

Solution:

```
REPORT  z_abap101_010.

* 2nd Option - Reusing TYPES keyword

TYPES: BEGIN OF some_components_sflight_2,
carrid    TYPE sflight-carrid,
connid    TYPE sflight-connid,
fldate    TYPE sflight-fldate,
price     TYPE sflight-price,
currency  TYPE sflight-currency,
planetype TYPE sflight-planetype,
seatsmax  TYPE sflight-seatsmax,
seatsocc  TYPE sflight-seatsocc,
END OF some_components_sflight_2.

TYPES: BEGIN OF flight_booking,
carrid   TYPE sbook-carrid   ,
connid   TYPE sbook-connid   ,
fldate   TYPE sbook-fldate   ,
bookid   TYPE sbook-bookid   ,
customid TYPE sbook-customid ,
END OF flight_booking.

* Note that some_components_sflight_2 and flight_booking have components with
the same name

* 'carrid', 'connid' and 'fldate'. 2 components cannot have the same name so

* in the structure below we add a suffix for each component originated in the
local

* structure flight_booking

TYPES: BEGIN OF sflight_sbook.
INCLUDE TYPE some_components_sflight_2.
INCLUDE TYPE flight_booking AS book RENAMING WITH SUFFIX _book.
TYPES END OF sflight_sbook.

START-OF-SELECTION. " F8 To Execute

DATA one_record TYPE sflight_sbook.

BREAK-POINT. " See one_record using the debugger
```

Declare a table type of integers.

Solution:

```
REPORT  z_abap101_011.

TYPES table_type_with_number TYPE TABLE OF i.

DATA odd_numbers TYPE table_type_with_number.
DATA even_numbers TYPE table_type_with_number.

START-OF-SELECTION.

APPEND: 1 TO odd_numbers,
3 TO odd_numbers,
5 TO  odd_numbers,
7 TO  odd_numbers,
9 TO odd_numbers.

APPEND: 2 TO even_numbers,
4 TO even_numbers,
6 TO even_numbers,
8 TO even_numbers,
10 TO even_numbers.
```

Declare a table type with all components of the global structure SFLIGHT.

Solution:

```
REPORT  Z_ABAP101_012.

TYPES table_type_sflight TYPE TABLE OF sflight.

DATA sflight_work_area TYPE LINE OF table_type_sflight.

DATA table_sflight TYPE table_type_sflight.

START-OF-SELECTION.

sflight_work_area-CARRID = 'AA'.

sflight_work_area-CONNID = '0017'.

sflight_work_area-FLDATE = 20131225. "Christmas

sflight_work_area-PRICE = '500.12'.

APPEND sflight_work_area TO table_sflight.

sflight_work_area-CARRID = 'AA'.

sflight_work_area-CONNID = '064'.

sflight_work_area-FLDATE = 20131225.

sflight_work_area-PRICE = '500.12'.

APPEND sflight_work_area TO table_sflight.
```

Declare a table type using the structure type created in exercise 8.

Solution:

```
REPORT  Z_ABAP101_013.

* 2nd Option - Reusing TYPES keyword

TYPES: BEGIN OF some_components_sflight_2,
carrid    TYPE sflight-carrid,
connid    TYPE sflight-connid,
fldate    TYPE sflight-fldate,
price     TYPE sflight-price,
currency  TYPE sflight-currency,
planetype TYPE sflight-planetype,
seatsmax  TYPE sflight-seatsmax,
seatsocc  TYPE sflight-seatsocc,
END OF some_components_sflight_2.

TYPES table_type_short_sflight TYPE TABLE OF some_components_sflight_2 WITH KEY
carrid connid fldate.
```

Declare a table type with the following components of the table SBOOK: CARRID, CONNID, FLDATE, BOOKID, CUSTOMID but using CUSTOMID as part of the table key.

Solution:

```
REPORT  Z_ABAP101_014.

TYPES: BEGIN OF flight_booking,
carrid   TYPE sbook-carrid   ,
connid   TYPE sbook-connid   ,
fldate   TYPE sbook-fldate   ,
bookid   TYPE sbook-bookid   ,
customid TYPE sbook-customid ,
END OF flight_booking .

TYPES table_type_booking TYPE TABLE OF flight_booking WITH KEY carrid connid
fldate customid.
```

Declare a variable of type character with 10 positions and give it ‘Hello ABAP’ as an initial value.

Solution:

```
REPORT  Z_ABAP101_015.

DATA message TYPE c LENGTH 10 VALUE 'Hello ABAP'.
```

Declare a variable of numeric type with 4 positions and initial value 1234.

Solution:

```
REPORT  Z_ABAP101_016.

DATA amount TYPE f VALUE 1234.
```

Declare a variable of type integer with initial value 42.

Solution:

```
REPORT  z_abap101_017.

DATA age TYPE i VALUE 42.
```

Declare a variable of type integer with initial value 12.72.

Solution:

```
REPORT  z_abap101_018.

DATA round_number TYPE i VALUE '12.72'.

WRITE round_number.
```

Declare a variable of type date and give it halloween day.

Solution:

```
REPORT  Z_ABAP101_019.

DATA any_day TYPE d.

any_day = '20131031'. " Halloween.
```

Declare a packed number variable with 7 decimal places.

Solution:

```
REPORT  z_abap101_020.

DATA many_decimals TYPE p LENGTH 10 DECIMALS 7.

many_decimals = '123456789.987654321'.

WRITE many_decimals. " What is printed?
```

Declare a variable of type S_CARR_ID.

Solution:

```
REPORT  Z_ABAP101_021.

DATA air_line_code TYPE s_carr_id.
```

Declare a variable of the same type of field carrid from table SPFLI.

Solution:

```
REPORT  Z_ABAP101_022.

DATA air_line_code TYPE spfli-carrid.
```

Declare a variable of the same type of field FLDATE table SFLIGHT.

Solution:

```
REPORT  z_abap101_023.

DATA flight_date TYPE sflight-fldate.
```

Declare a structure of the same type of SBOOK.

Solution:

```
REPORT  z_abap101_024.

DATA single_booking TYPE sbook.
```

Declare a structure with fields of the table SFLIGHT carrid, CONNID, FLDATE, PRICE, CURRENCY, PLANETYPE, and SEATSMAX SEATSOCC.

Solution:

```
REPORT  z_abap101_025.

DATA: BEGIN OF some_components_of_one_flight,
carrid    TYPE sflight-carrid   ,
connid    TYPE sflight-connid   ,
fldate    TYPE sflight-fldate   ,
price     TYPE sflight-price    ,
currency  TYPE sflight-currency ,
planetype TYPE sflight-planetype,
seatsmax  TYPE sflight-seatsmax ,
seatsocc  TYPE sflight-seatsocc ,
END OF some_components_of_one_flight.
```

Declare a structure with all fields of the table SBOOK and the field TELEPHONE from SCUSTOM table.

Solution:

```
REPORT  z_abap101_026.

DATA: BEGIN OF sbook_with_phone.
INCLUDE STRUCTURE sbook.
DATA phone TYPE scustom-telephone.
DATA END OF sbook_with_phone.
```

Declare an internal table with fields of the table SBOOK CARRID, CONNID, FLDATE, BOOKID, CUSTOMID.

Solution:

```
REPORT  z_abap101_027.

TYPES: BEGIN OF ty_sbook ,
carrid TYPE sbook-carrid,
connid TYPE sbook-connid,
fldate TYPE sbook-fldate,
bookid TYPE sbook-bookid,
customer_id TYPE sbook-customid,
END OF ty_sbook.

TYPES ty_itab_sbook TYPE TABLE OF ty_sbook WITH KEY carrid connid fldate bookid.

DATA itab_sbook TYPE ty_itab_sbook.
```

Declare an internal table with all table fields from table SCARR.

Solution:

```
REPORT  z_abap101_028.

DATA it_scarr TYPE STANDARD TABLE OF scarr.
```

Declare an internal table with all table fields SPFLI.

Solution:

```
REPORT  z_abap101_029.

TYPES ty_spfli TYPE spfli.

DATA itab_spfli TYPE TABLE OF ty_spfli.
```

Declare an internal table with all table fields from SCARR and the field TELEPHONE from table SCUSTOM.

Solution:

```
REPORT  z_abap101_030.

TYPES: BEGIN OF ty_scarr_telephone.
        INCLUDE TYPE scarr.
TYPES: phone TYPE scustom-telephone,
END OF ty_scarr_telephone,

ty_itab_scarr_telephone TYPE SORTED TABLE OF ty_scarr_telephone WITH UNIQUE KEY
carrid.

DATA itab TYPE ty_itab_scarr_telephone.
```

Declare a constant which contains your name.

Solution:

```
REPORT  z_abap101_031.

CONSTANTS c_full_name TYPE string VALUE 'ABAP101.com'.
```

Declare two constants which contain the values 'X' (true) and ' ' (false).

Note: This is a common practice as ABAP does not contain a boolean primitive type.

Solution:

```
REPORT  z_abap101_032.

TYPES ty_boolean TYPE c. " what is the length?

CONSTANTS c_true TYPE ty_boolean VALUE 'X'.
CONSTANTS c_false TYPE ty_boolean VALUE space.
```

Declare a constants which contains the 5 first decimals of Pi.

Solution:

```
REPORT  z_abap101_033.

CONSTANTS c_pi TYPE p LENGTH 7 DECIMALS 5 VALUE '3.14159'.
```

Declare a work area of constants. All components must be integers.

Solution:

```
REPORT  z_abap101_034.

CONSTANTS: BEGIN OF c_integers,
first TYPE i VALUE 1,
second TYPE i VALUE 2,
third TYPE i VALUE 3,
END OF c_integers.
```

Declare a work area of 5 constant components. All of them should have different primitive types.

Solution:

```
REPORT  z_abap101_035.

CONSTANTS: BEGIN OF c_primitives,
char TYPE c LENGTH 4 VALUE 'CHAR',
int TYPE i VALUE 101,
date TYPE d VALUE '20141225',
time TYPE t VALUE '112359',
string TYPE string VALUE `STRING`,
END OF c_primitives.
```

Is it possible to declare an internal table of constants?

Answer:

No. No initial value can be specified for internal tables and references

```
REPORT  z_abap101_036.

TYPES ty_integer TYPE i.
TYPES ty_integers TYPE TABLE OF ty_integer.

CONSTANTS itab_integers TYPE ty_integers VALUE 1. " Syntax error
```

Declare all types and constants from type-pools ABAP and ICON.

Solution:

```
REPORT  z_abap101_037.

TYPE-POOLS: icon, abap.

CONSTANTS c_favorite_icon LIKE icon_information VALUE '@0S@'.

DATA is_true TYPE abap_bool.
```

Declare a constant which type is the same of another constant.

Solution:

```
REPORT  z_abap101_038.

CONSTANTS c_a TYPE c VALUE 'A'.
CONSTANTS c_z LIKE c_a VALUE 'Z'.
```

Declare a type which is used in another type, variable, work area, internal table and constant.

Solution:

```
REPORT  z_abap101_039.

TYPES ty_reused_date TYPE d.

TYPES ty_creation_date TYPE ty_reused_date.

DATA v_creation_date TYPE ty_reused_date.
DATA v_update_date TYPE ty_reused_date.

DATA: BEGIN OF wa_document,
name TYPE string,
creation_date TYPE ty_reused_date,
update_date TYPE ty_reused_date,
END OF wa_document.

DATA: itab_documents LIKE TABLE OF wa_document.

CONSTANTS c_update_date TYPE ty_reused_date VALUE '99991231'.
```

Declare a variable which is used in another variable, type, work area, internal table and constant.

Solution:

```
REPORT  z_abap101_040.

DATA v_reused_time TYPE t.

DATA v_last_changed_at LIKE v_reused_time.
TYPES ty_last_changed_at LIKE v_reused_time.

DATA: BEGIN OF wa_employee,
name TYPE string,
next_meeting_at LIKE v_reused_time,
END OF wa_employee.

DATA: BEGIN OF itab_employees OCCURS 0,
name TYPE string,
next_meeting_at LIKE v_reused_time,
END OF itab_employees.

CONSTANTS c_lunch_time LIKE v_reused_time VALUE '130000'.
```

Part II - Imperative Logic (41 - 72)

Write an executable program that performs the following calculation: 2 + 3 * 5

Solution:

```
REPORT  z_abap101_041.

START-OF-SELECTION.

  DATA v_result TYPE i.

  v_result = 2 + 3 * 5. " 25 or 17?

  WRITE v_result.
```

Write an executable program that get two integers inside variables and perform the addition, subtraction, multiplication, division and power between them.

Solution:

```
REPORT  z_abap101_042.

DATA v_number_a TYPE i.
DATA v_number_b LIKE v_number_a VALUE 2.
DATA v_result TYPE f .

START-OF-SELECTION.
  v_number_a = 5.

  v_result = v_number_a + v_number_b.
  WRITE: 'Adition:', v_result EXPONENT 0 .
  NEW-LINE.

  v_result = v_number_a - v_number_b.
  WRITE: 'Subtraction:', v_result.
  NEW-LINE.

  v_result = v_number_a * v_number_b.
  WRITE: 'Multiplication:', v_result.
  NEW-LINE.

  v_result = v_number_a / v_number_b.
  WRITE: 'Division:', v_result.
  NEW-LINE.

  v_result = v_number_a ** v_number_b.
  WRITE: 'Power:', v_result.
```

Write an executable program that get two integers inside parameters and perform the addition, subtraction, multiplication, division and power between them.

Solution:

```
REPORT  z_abap101_043.

PARAMETERS p_num_a TYPE i.
PARAMETERS p_num_b LIKE p_num_a DEFAULT 2.
DATA v_result TYPE f.

START-OF-SELECTION.
  p_num_a = 5.

  v_result = p_num_a + p_num_b.
  WRITE: 'Addition:', v_result EXPONENT 0 .
  NEW-LINE.

  v_result = p_num_a - p_num_b.
  WRITE: 'Subtraction:', v_result.
  NEW-LINE.

  v_result = p_num_a * p_num_b.
  WRITE: 'Multiplication:', v_result.
  NEW-LINE.

  v_result = p_num_a / p_num_b.
  WRITE: 'Division:', v_result.
  NEW-LINE.

  v_result = p_num_a ** p_num_b.
  WRITE: 'Power:', v_result.
```

Write an executable program that concatenates two words and write the result.

Solution:

```
REPORT  z_abap101_044.

CONSTANTS c_abap TYPE c LENGTH 4 VALUE 'ABAP'.
DATA v_whole_text TYPE string.

START-OF-SELECTION.

  CONCATENATE c_abap '101' INTO v_whole_text .
  WRITE v_whole_text.
```

Write an executable program that concatenates two words and the current month, separating each part by a "-" and write the result.

Solution:

```
REPORT  z_abap101_045.

CONSTANTS c_abap TYPE c LENGTH 7 VALUE 'ABAP999'.
CONSTANTS c_separator TYPE c VALUE '-'.
DATA v_whole_text TYPE string.

START-OF-SELECTION.

  CONCATENATE c_abap(4) '101' sy-datum+4(2) INTO v_whole_text SEPARATED BY
c_separator.
  WRITE v_whole_text.
```

Write an executable program that reads the current system date and write it in your language in text format.

Ex: 20140727 should be written as July the Twenty-Seventh, 2014

Solution:

```
REPORT  z_abap101_046.

DATA v_day TYPE string.
DATA v_month TYPE string.
DATA v_year TYPE string.

START-OF-SELECTION.

* Handle month
  CASE sy-datum+4(2).

    WHEN '01'.
      v_month = 'January'.
    WHEN '02'.
      v_month = 'February'.
    WHEN '03'.
      v_month = 'March'.
    WHEN '04'.
      v_month = 'April'.
    WHEN '05'.
      v_month = 'May'.
    WHEN '06'.
      v_month = 'June'.
    WHEN '07'.
      v_month = 'July'.
    WHEN '08'.
      v_month = 'August'.
    WHEN '09'.
      v_month = 'September'.
    WHEN '10'.
      v_month = 'October'.
    WHEN '11'.
      v_month = 'November'.
    WHEN '12'.
      v_month = 'December'.

  ENDCASE.

* Handle day
  IF sy-datum+6(2) = '01'.
    v_day = 'first'.
  ELSEIF sy-datum+6(2) = '02'.
    v_day = 'second'.
```

```
ELSEIF sy-datum+6(2) = '03'.
  v_day = 'third'.

ELSE. " 04-31

  IF sy-datum+6(2) BETWEEN 04 AND 19. " 04-19

    CASE sy-datum+6(2).
      WHEN '04'.
        v_day = 'four'.
      WHEN '05'.
        v_day = 'fif'.
      WHEN '06'.
        v_day = 'six'.
      WHEN '07'.
        v_day = 'seven'.
      WHEN '08'.
        v_day = 'eigh'.
      WHEN '09'.
        v_day = 'nin'.
      WHEN '10'.
        v_day = 'ten'.
      WHEN '11'.
        v_day = 'eleven'.
      WHEN '12'.
        v_day = 'twelf'.
      WHEN '13'.
        v_day = 'thirteen'.
      WHEN '14'.
        v_day = 'fourteen'.
      WHEN '15'.
        v_day = 'fitteen'.
      WHEN '16'.
        v_day = 'sixteen'.
      WHEN '17'.
        v_day = 'seventeen'.
      WHEN '18'.
        v_day = 'eighteen'.
      WHEN '19'.
        v_day = 'nineteen'.
    ENDCASE.
    CONCATENATE v_day 'th' INTO v_day.

  ELSE.
    CASE sy-datum+6(2). " 20-31
      WHEN '20'.
        v_day = 'twentieth'.
      WHEN '21'.
        v_day = 'twenty-first'.
      WHEN '22'.
        v_day = 'twenty-second'.
```

```
        WHEN '23'.
          v_day = 'twenty-third'.
        WHEN '24'.
          v_day = 'twenty-fourth'.
        WHEN '25'.
          v_day = 'twenty-fifth'.
        WHEN '26'.
          v_day = 'twenty-six'.
        WHEN '27'.
          v_day = 'twenty-seventh'.
        WHEN '28'.
          v_day = 'twenty-eighth'.
        WHEN '29'.
          v_day = 'twenty-first'.
        WHEN '30'.
          v_day = 'thirtieth'.
        WHEN '31'.
          v_day = 'thirty-first'.
      ENDCASE.

    ENDIF.

  ENDIF.

* Handle Year
  v_year = sy-datum(4).

* print result
  WRITE: v_month, ' the ',  v_day, ', ', v_year. NEW-LINE.
```

Write an executable program that reads the current system time and write the time in 6 different zones (3 of them should be compulsorily Greenwich, Delhi and Brasilia).

Solution:

```
REPORT  z_abap101_047.

DATA v_timezone1 TYPE tznzone VALUE 'GMTUK'. " Greenwich
DATA v_timezone2 TYPE tznzone VALUE 'INDIA'. " Delhi
DATA v_timezone3 TYPE tznzone VALUE 'BRAZIL'. " Brasilia
DATA v_timezone4 TYPE tznzone VALUE 'CST'.
DATA v_timezone5 TYPE tznzone VALUE 'ISRAEL'.
DATA v_timezone6 TYPE tznzone VALUE 'RUS06'.

DATA v_timestamp TYPE tzonref-tstamps.
DATA v_timestamp_string TYPE string.

START-OF-SELECTION.

  CONCATENATE sy-datum sy-uzeit INTO v_timestamp_string.
  v_timestamp = v_timestamp_string.

  WRITE v_timestamp TIME ZONE v_timezone1. NEW-LINE.
  WRITE v_timestamp TIME ZONE v_timezone2. NEW-LINE.
  WRITE v_timestamp TIME ZONE v_timezone3. NEW-LINE.
  WRITE v_timestamp TIME ZONE v_timezone4. NEW-LINE.
  WRITE v_timestamp TIME ZONE v_timezone5. NEW-LINE.
  WRITE v_timestamp TIME ZONE v_timezone6. NEW-LINE.
```

Write an executable program that counts how many vowels are in the name of the user running the program and print the result

Solution:

```
REPORT  z_abap101_048.

DATA v_vowels_count TYPE i.
DATA v_vowels_total TYPE i.

DATA v_user LIKE sy-uname.

START-OF-SELECTION.

  v_user = sy-uname.
  TRANSLATE v_user TO UPPER CASE.

* One option
  FIND ALL OCCURRENCES OF 'A' IN v_user MATCH COUNT v_vowels_count.
  v_vowels_total = v_vowels_total + v_vowels_count.
  FIND ALL OCCURRENCES OF 'E' IN v_user MATCH COUNT v_vowels_count.
  v_vowels_total = v_vowels_total + v_vowels_count.
  FIND ALL OCCURRENCES OF 'I' IN v_user MATCH COUNT v_vowels_count.
  v_vowels_total = v_vowels_total + v_vowels_count.
  FIND ALL OCCURRENCES OF 'O' IN v_user MATCH COUNT v_vowels_count.
  v_vowels_total = v_vowels_total + v_vowels_count.
  FIND ALL OCCURRENCES OF 'U' IN v_user MATCH COUNT v_vowels_count.
  v_vowels_total = v_vowels_total + v_vowels_count.

* Another option
  FIND ALL OCCURRENCES OF REGEX 'A|E|I|O|U' IN v_user MATCH COUNT
v_vowels_count.

  WRITE v_vowels_total.
```

Write an executable program that counts a string length and if it's bigger than 2o characteres, write 'Too big'. If not, write the string length.

Solution:

```
REPORT  z_abap101_049.

DATA v_string TYPE string VALUE '1234567890ABCDEFGHIJ'.
DATA v_string_length TYPE i.

START-OF-SELECTION.

  v_string_length = strlen( v_string ).

  IF v_string_length > 20 .
    WRITE 'Too big'.
  ELSE.
    WRITE v_string_length.
  ENDIF.
```

Write an executable program that counts from 1 to 100 and for each multiple of 8, write the message: "The number [number] is a multiple of 8 ".

Solution:

```
REPORT  z_abap101_050.

DATA v_current_number TYPE i VALUE 1.

START-OF-SELECTION.

  WHILE v_current_number <= 100.

    IF ( v_current_number MOD 8 ) = 0.
      WRITE: 'The number', v_current_number, ' is a multiple of 8'.
      NEW-LINE.
    ENDIF.

    ADD 1 TO v_current_number.
  ENDWHILE.
```

Write an executable program that contains a routine which prints all usernames in the system. (Check table USR04 and its content in transaction SE11, SE16 or SE16N).

Solution:

```
REPORT  z_abap101_051.

TYPES ty_users TYPE TABLE OF usr04-bname.
DATA it_users TYPE ty_users.

START-OF-SELECTION.

  SELECT bname
  FROM usr04
  INTO TABLE it_users.

  PERFORM print_users USING it_users.

*&---------------------------------------------------------------------*
*&      Form  print_users
*&---------------------------------------------------------------------*
* Prints all usernames in the system
*----------------------------------------------------------------------*
*      -->US_T_USERS usernames
*----------------------------------------------------------------------*
FORM print_users USING us_t_users TYPE ty_users.

  DATA lwa_user TYPE LINE OF ty_users.

  LOOP AT us_t_users INTO lwa_user.
    WRITE lwa_user. NEW-LINE.
  ENDLOOP.
ENDFORM.                    "print_users
```

For this exercise, you should Read the help from command FORM completely. Then, write an executable program that has a routine that receives four global variables and change their value. Each variable will be received in a different way: 2 of them using the addition USING and the other 2 using the addition CHANGING from the FORM command. For each pair use and omit the adding VALUE. Print the contents of all global variables before the routine is called, at the beginning of the routine, at the end of the routine (after all values are changed) and after the PERFORM statement. See how the contents of variables behave using the debugger.

Solution:

```
REPORT  z_abap101_052.

DATA gv_a TYPE i VALUE 1.
DATA gv_b TYPE i VALUE 2.
DATA gv_c TYPE i VALUE 3.
DATA gv_d TYPE i VALUE 4.

*&---------------------------------------------------------------------*
*&      Form  form_parameters
*&---------------------------------------------------------------------*
* Get 4 parameters in different ways
*----------------------------------------------------------------------*
*      -->US_A       text
*      -->(USV_B)    text
*      -->CH_C       text
*      -->(CHV_D)    text
*----------------------------------------------------------------------*
FORM form_parameters
    USING   us_a TYPE i
            value(usv_b) TYPE i
    CHANGING ch_c TYPE i
        value(chv_d) TYPE i.

  WRITE 'Inside FORM.'. NEW-LINE.
  WRITE: 'us_a: ', us_a. NEW-LINE.
  WRITE: 'usv_b: ', usv_b. NEW-LINE.
  WRITE: 'ch_c: ', ch_c. NEW-LINE.
  WRITE: 'chv_d: ', chv_d. NEW-LINE.

  us_a = us_a + 10.
  usv_b = usv_b + 10.
  ch_c = ch_c + 10.
  chv_d = chv_d + 10.

  WRITE 'Inside FORM, after update local variables'. NEW-LINE.
  WRITE: 'us_a: ', us_a. NEW-LINE.
  WRITE: 'usv_b: ', usv_b. NEW-LINE.
  WRITE: 'ch_c: ', ch_c. NEW-LINE.
```

```
  WRITE: 'chv_d: ', chv_d. NEW-LINE.

  WRITE: 'gv_a: ', gv_a. NEW-LINE.
  WRITE: 'gv_b: ', gv_b. NEW-LINE.
  WRITE: 'gv_c: ', gv_c. NEW-LINE.
  WRITE: 'gv_d: ', gv_d. NEW-LINE.

ENDFORM.                    "form_parameters

START-OF-SELECTION.
  WRITE 'Before FORM'. NEW-LINE.
  WRITE: 'gv_a: ', gv_a. NEW-LINE.
  WRITE: 'gv_b: ', gv_b. NEW-LINE.
  WRITE: 'gv_c: ', gv_c. NEW-LINE.
  WRITE: 'gv_d: ', gv_d. NEW-LINE.

  PERFORM form_parameters
    USING
      gv_a
      gv_b
    CHANGING
      gv_c
      gv_d
    .

  WRITE 'After FORM'. NEW-LINE.
  WRITE: 'gv_a: ', gv_a. NEW-LINE.
  WRITE: 'gv_b: ', gv_b. NEW-LINE.
  WRITE: 'gv_c: ', gv_c. NEW-LINE.
  WRITE: 'gv_d: ', gv_d. NEW-LINE.
```

Write an executable program that has a routine that receives two numbers and returns the largest of them, If the numbers are equal return the number itself.

Solution:

```
REPORT  z_abap101_053.

DATA gv_largest TYPE f.

*&---------------------------------------------------------------------*
*&      Form  get_larger
*&---------------------------------------------------------------------*
*  Compares 2 numbers and returns the largest. If equal returns itself
*----------------------------------------------------------------------*
*      -->NUMBER_A        Number A
*      -->NUMBER_B        Number B
*      -->LARGEST_NUMBER  Largest Number
*----------------------------------------------------------------------*
FORM get_larger
    USING
        number_a TYPE f
        number_b TYPE f
    CHANGING
        largest_number TYPE f.

  IF number_a >= number_b.
    largest_number = number_a.
  ELSE.
    largest_number = number_b.
  ENDIF.

ENDFORM.                    "get_larger

START-OF-SELECTION.

  PERFORM get_larger USING 1 2 CHANGING gv_largest.
  WRITE gv_largest EXPONENT 0. NEW-LINE.

  PERFORM get_larger USING 4 3 CHANGING gv_largest.
  WRITE gv_largest EXPONENT 0. NEW-LINE.

  PERFORM get_larger USING 5 5 CHANGING gv_largest.
  WRITE gv_largest EXPONENT 0. NEW-LINE.

  PERFORM get_larger USING '6.2' '7.1' CHANGING gv_largest.
  WRITE gv_largest EXPONENT 0. NEW-LINE.
```

Write an executable program that has a routine that receives two numbers and return a flag (character with length 1). If the numbers are equal, set the flag with 'X. Otherwise set the flag to space.

Solution:

```
REPORT  z_abap101_054.

DATA gv_flag TYPE c.

*&---------------------------------------------------------------------*
*&      Form  get_larger
*&---------------------------------------------------------------------*
*  Compares 2 numbers and returns a flag (true) if they are equal
*----------------------------------------------------------------------*
*      -->NUMBER_A        Number A
*      -->NUMBER_B        Number B
*      -->FLAG            Equal numbers indicator
*----------------------------------------------------------------------*
FORM set_flag_if_equal
    USING
        number_a TYPE f
        number_b TYPE f
    CHANGING
        flag TYPE c.

  IF number_a = number_b.
    flag = abap_true.
    WRITE flag. NEW-LINE.
  ELSE.
    flag = abap_false.
  ENDIF.

ENDFORM.                    "get_larger

START-OF-SELECTION.

  PERFORM set_flag_if_equal USING 1 1 CHANGING gv_flag.

  PERFORM set_flag_if_equal USING 4 3 CHANGING gv_flag.

  PERFORM set_flag_if_equal USING 5 5 CHANGING gv_flag.

  PERFORM set_flag_if_equal USING '6.2' '7.1' CHANGING gv_flag.
```

Write an executable program that has a routine that takes two numbers and writes the result of the operation [higher_number / lower_number] if the numbers are different. If they are equal, write the result of the operation [number ^ 2].

Solution:

```
REPORT  z_abap101_055.

DATA gv_result TYPE f.

*&---------------------------------------------------------------------*
*&      Form  get_larger
*&---------------------------------------------------------------------*
*  Compares 2 numbers and returns the largest. If equal returns itself
*----------------------------------------------------------------------*
*      -->NUMBER_A        Number A
*      -->NUMBER_B        Number B
*      -->LARGEST_NUMBER  Largest Number
*----------------------------------------------------------------------*
FORM get_larger
    USING
        number_a TYPE f
        number_b TYPE f
    CHANGING
        largest_number TYPE f.

  IF number_a >= number_b.
    largest_number = number_a.
  ELSE.
    largest_number = number_b.
  ENDIF.

ENDFORM.                    "get_larger

*&---------------------------------------------------------------------*
*&      Form  get_larger
*&---------------------------------------------------------------------*
*  Compares 2 numbers and returns a flag (true) if they are equal
*----------------------------------------------------------------------*
*      -->NUMBER_A        Number A
*      -->NUMBER_B        Number B
*      -->FLAG            Equal numbers indicator
*----------------------------------------------------------------------*
FORM set_flag_if_equal
    USING
        number_a TYPE f
        number_b TYPE f
    CHANGING
```

```
        flag TYPE c.

  IF number_a = number_b.
    flag = abap_true.
*    WRITE flag. NEW-LINE.
  ELSE.
    flag = abap_false.
  ENDIF.

ENDFORM.                    "get_larger

*&---------------------------------------------------------------------*
*&      Form  division_or_power2
*&---------------------------------------------------------------------*
* takes two numbers and writes the result of the operation
* [higher_number / lower_number] if the numbers are different.
* If they are equal, write the result of the operation [number ^ 2].
*----------------------------------------------------------------------*
*      -->NUMBER_A   text
*      -->NUMBER_B   text
*      -->RESULT     text
*----------------------------------------------------------------------*
FORM division_or_power2
    USING
        number_a TYPE f
        number_b TYPE f
    CHANGING
        result TYPE f.

  DATA lv_number_equal TYPE c.

  PERFORM set_flag_if_equal
    USING
      number_a
      number_b
    CHANGING
      lv_number_equal.

  IF lv_number_equal = abap_true.
    result = number_a ** 2.
  ELSE.
    DATA lv_larger_number TYPE f.

    PERFORM get_larger USING number_a number_b CHANGING lv_larger_number.

    IF number_a = lv_larger_number.
      result = number_a / number_b.
    ELSE.
      result = number_b / number_a.
    ENDIF.
```

```
  ENDIF.

  WRITE result EXPONENT 0.
  NEW-LINE.
ENDFORM.                    "division_or_power2

START-OF-SELECTION.

  PERFORM division_or_power2
      USING 1 1
      CHANGING gv_result.

  PERFORM division_or_power2
      USING 3 3
      CHANGING gv_result.

  PERFORM division_or_power2
      USING 6 2
      CHANGING gv_result.

  PERFORM division_or_power2
      USING 2 6
      CHANGING gv_result.

  PERFORM division_or_power2
      USING 10 2
      CHANGING gv_result.

  PERFORM division_or_power2
      USING 2 10
      CHANGING gv_result.
```

Write an executable program that does NOT have a routine. The program should include a work area with 5 fields of different types or more. Then, it must be populated and its fields should be printed one per line, separated by one horizontal line. After testing your program, change the output separating each field by two lines. During this process, refactor your code to include a routine which handle the separation between each line.

Solution:

```
REPORT  z_abap101_056.

DATA: BEGIN OF work_area,
    str TYPE string,
    date TYPE d,
    time TYPE t,
    integer TYPE i,
    hex TYPE x LENGTH 8,
    END OF work_area.

START-OF-SELECTION.

  work_area-str = 'This is a string'.
  work_area-date = '20141225'. " Christmas
  work_area-time = '134059'.
  work_area-integer = 101.
  work_area-hex = '0123456789ABCDEF'.

* Before refactoring
*  WRITE work_area-str.
*  ULINE.
*  WRITE work_area-date DD/MM/YY.
*  ULINE.
*  WRITE work_area-time.
*  ULINE.
*  WRITE work_area-integer.
*  ULINE.
*  WRITE work_area-hex.
*  ULINE.

* After refactoring
  WRITE work_area-str.
  PERFORM separe_line.
  WRITE work_area-date DD/MM/YY.
  PERFORM separe_line.
  WRITE work_area-time.
  PERFORM separe_line.
  WRITE work_area-integer.
  PERFORM separe_line.
  WRITE work_area-hex.
```

```
  PERFORM separe_line.

*&---------------------------------------------------------------------*
*&      Form  separe_line
*&---------------------------------------------------------------------*
* Separe each output line
*----------------------------------------------------------------------*
FORM separe_line.
  DO 2 TIMES.
    ULINE.
  ENDDO.
ENDFORM.                    "separe_line
```

Write an executable program with a routine that receives a work area containing five different data types and count how many components are not filled. Finally, print result.

Solution:

```
REPORT  z_abap101_057.

TYPES: BEGIN OF ty_work_area,
    str TYPE string,
    date TYPE d,
    time TYPE t,
    integer TYPE i,
    hex TYPE x LENGTH 8,
    END OF ty_work_area.

DATA work_area TYPE ty_work_area.

*&---------------------------------------------------------------------*
*&      Form  count_initial_components
*&---------------------------------------------------------------------*
* Gets a work area, counts how many components are initial and write
* the result
*----------------------------------------------------------------------*
*      -->US_WORK_AREA  work area
*----------------------------------------------------------------------*
FORM count_initial_components USING us_work_area TYPE ty_work_area.
  DATA lv_initial_components_counter TYPE i.

  IF us_work_area-str IS INITIAL.
    lv_initial_components_counter = lv_initial_components_counter + 1.
  ENDIF.

  IF us_work_area-date IS INITIAL.
    lv_initial_components_counter = lv_initial_components_counter + 1.
  ENDIF.

  IF us_work_area-time IS INITIAL.
    lv_initial_components_counter = lv_initial_components_counter + 1.
  ENDIF.

  IF us_work_area-integer IS INITIAL.
    lv_initial_components_counter = lv_initial_components_counter + 1.
  ENDIF.

  IF us_work_area-hex IS INITIAL.
    lv_initial_components_counter = lv_initial_components_counter + 1.
  ENDIF.
```

```
  WRITE: 'Initial components: ', lv_initial_components_counter.
  NEW-LINE.

ENDFORM.                    "count_initial_components

START-OF-SELECTION.

  PERFORM count_initial_components USING work_area.
  work_area-str = 'This is a string'.
  PERFORM count_initial_components USING work_area.
  work_area-date = '20141225'. " Christmas
  PERFORM count_initial_components USING work_area.
  work_area-time = '134059'.
  PERFORM count_initial_components USING work_area.
  work_area-integer = 101.
  PERFORM count_initial_components USING work_area.
  work_area-hex = '0123456789ABCDEF'.
  PERFORM count_initial_components USING work_area.
```

Write an executable program with a routine that receives a work area with at least 4 components. All components can only be declared using numeric and different primitive types. Your routine should sum the values from all components and print the result.

Solution:

```
REPORT  z_abap101_058.

TYPES: BEGIN OF ty_work_area,
    integer TYPE i,
    float TYPE f,
    pack TYPE p LENGTH 8 DECIMALS 3,
    decfloat34 TYPE decfloat34,
    END OF ty_work_area.

*&---------------------------------------------------------------------*
*&      Form  sum_numeric_components
*&---------------------------------------------------------------------*
* Receives a work area with numeric components and sum them.
*----------------------------------------------------------------------*
*      -->US_WA      Work area with numeric components
*----------------------------------------------------------------------*
FORM sum_numeric_components USING us_wa TYPE ty_work_area.
  DATA lv_sum_result TYPE decfloat34.

  lv_sum_result = us_wa-integer + us_wa-float + us_wa-pack + us_wa-decfloat34.

  WRITE lv_sum_result.
  NEW-LINE.

ENDFORM.                    "sum_numeric_components

START-OF-SELECTION.

  DATA work_area TYPE ty_work_area.
  DATA work_area_doubled TYPE ty_work_area.

  work_area-integer = 2.
  work_area-float = '2.5'.
  work_area-pack = '2.12345'.
  work_area-decfloat34 = 10000000000000000000000000000000.

  PERFORM sum_numeric_components USING work_area.

  work_area_doubled-integer = work_area-integer * 2.
  work_area_doubled-float = work_area-float * 2.
  work_area_doubled-pack = work_area-pack * 2.
  work_area_doubled-decfloat34 = work_area-decfloat34 * 2.

  PERFORM sum_numeric_components USING work_area_doubled.
```

Write an executable program which has a routine that receives a work area with 3 char components and 3 numeric components. The routine should clear some component values according to the following rules:

1. Clear char components only if the sum of the numeric components is odd (ignoring possible decimal places)
2. Clear numeric components only if the sum of vowels in the three char components is even (ignoring lower/upper case)

Solution:

```
REPORT  z_abap101_059.

TYPES:
  BEGIN OF ty_char_and_numeric,
    char_comp1 TYPE string,
    char_comp2 TYPE c LENGTH 3,
    char_comp3 TYPE n LENGTH 10,
    num_comp1 TYPE i,
    num_comp2 TYPE f,
    num_comp3 TYPE decfloat16,
  END OF ty_char_and_numeric.

*&---------------------------------------------------------------------*
*&      Form  clear_char_or_numeric
*&---------------------------------------------------------------------*
*  This routine clears some component values according to the following rules:
*  a. Clear char components only if the sum of the numeric components is odd
(ignoring possible decimal places)
*  b. Clear numeric components only if the sum of vowels in the three char
components is even (ignoring lower/upper case)
*----------------------------------------------------------------------*
*      -->US_WA_CHAR_AND_NUMERIC  text
*----------------------------------------------------------------------*
FORM clear_char_or_numeric USING us_wa_char_and_numeric TYPE
ty_char_and_numeric.

  DATA lv_mod_result TYPE i.

  DATA lv_sum_numeric TYPE i.

  lv_sum_numeric =
    us_wa_char_and_numeric-num_comp1 +
    us_wa_char_and_numeric-num_comp2 +
    us_wa_char_and_numeric-num_comp3.

  lv_mod_result = lv_sum_numeric MOD 2.
```

```
  IF lv_mod_result <> 0.
    CLEAR:
      us_wa_char_and_numeric-char_comp1,
      us_wa_char_and_numeric-char_comp2,
      us_wa_char_and_numeric-char_comp3.
    RETURN.
  ENDIF.

  DATA lv_vowel_count TYPE i.
  DATA lv_current_vowel_count TYPE i.

  FIND ALL OCCURRENCES OF REGEX 'a|e|i|o|u|A|E|I|O|U' IN us_wa_char_and_numeric-
char_comp1 MATCH COUNT lv_current_vowel_count.
  lv_vowel_count = lv_vowel_count + lv_current_vowel_count.

  FIND ALL OCCURRENCES OF REGEX 'a|e|i|o|u|A|E|I|O|U' IN us_wa_char_and_numeric-
char_comp2 MATCH COUNT lv_current_vowel_count.
  lv_vowel_count = lv_vowel_count + lv_current_vowel_count.

  FIND ALL OCCURRENCES OF REGEX 'a|e|i|o|u|A|E|I|O|U' IN us_wa_char_and_numeric-
char_comp3 MATCH COUNT lv_current_vowel_count.
  lv_vowel_count = lv_vowel_count + lv_current_vowel_count.

  lv_mod_result = lv_vowel_count MOD 2.
  IF lv_mod_result = 0.
    CLEAR:
      us_wa_char_and_numeric-num_comp1,
      us_wa_char_and_numeric-num_comp2,
      us_wa_char_and_numeric-num_comp3.
    RETURN.
  ENDIF.

ENDFORM.                    "clear_char_or_numeric

START-OF-SELECTION.

  DATA wa_char_cleared TYPE ty_char_and_numeric.
  DATA wa_numeric_cleared TYPE ty_char_and_numeric.

  wa_char_cleared-char_comp1 = 'This should be clea'.
  wa_char_cleared-char_comp2 = 'red'.
  wa_char_cleared-char_comp3 = '0123456789'.
  wa_char_cleared-num_comp1 = 1.
  wa_char_cleared-num_comp2 = 10.
  wa_char_cleared-num_comp3 = 100.

  WRITE:
    wa_char_cleared-char_comp1,
    wa_char_cleared-char_comp2,
    wa_char_cleared-char_comp3,
    wa_char_cleared-num_comp1,
```

```
  wa_char_cleared-num_comp2,
  wa_char_cleared-num_comp3.
NEW-LINE.

PERFORM clear_char_or_numeric USING wa_char_cleared.

WRITE:
  wa_char_cleared-char_comp1,
  wa_char_cleared-char_comp2,
  wa_char_cleared-char_comp3,
  wa_char_cleared-num_comp1,
  wa_char_cleared-num_comp2,
  wa_char_cleared-num_comp3.
NEW-LINE.

ULINE.

wa_numeric_cleared-char_comp1 = 'aeiouAEIOU'.
wa_numeric_cleared-char_comp2 = 'BCD'.
wa_numeric_cleared-char_comp3 = '0123456789'.
wa_numeric_cleared-num_comp1 = 2. " even
wa_numeric_cleared-num_comp2 = 10.
wa_numeric_cleared-num_comp3 = 100.

WRITE:
  wa_numeric_cleared-char_comp1,
  wa_numeric_cleared-char_comp2,
  wa_numeric_cleared-char_comp3,
  wa_numeric_cleared-num_comp1,
  wa_numeric_cleared-num_comp2,
  wa_numeric_cleared-num_comp3.
NEW-LINE.

PERFORM clear_char_or_numeric USING wa_numeric_cleared.

WRITE:
  wa_numeric_cleared-char_comp1,
  wa_numeric_cleared-char_comp2,
  wa_numeric_cleared-char_comp3,
  wa_numeric_cleared-num_comp1,
  wa_numeric_cleared-num_comp2,
  wa_numeric_cleared-num_comp3.
NEW-LINE.
```

Write an executable program which contains three internal tables (their type must contain at least three components of different data types). Each table will have a different type (standard, sorted and hashed). Add 3 identical values in each table and view the contents of each table in the debugger.

Solution:

```
REPORT  z_abap101_060.

TYPES: BEGIN OF ty_line,
id TYPE c LENGTH 10,
name TYPE string,
value TYPE i,
END OF ty_line.

DATA it_standard TYPE STANDARD TABLE OF ty_line.
DATA it_sorted TYPE SORTED TABLE OF ty_line WITH UNIQUE KEY id.
DATA it_hashed TYPE HASHED TABLE OF ty_line WITH UNIQUE KEY id.

START-OF-SELECTION.

  DATA wa TYPE ty_line.

  wa-id = '3'.
  wa-name = 'John'.
  wa-value = 50.

  APPEND wa TO it_standard.
  INSERT wa INTO TABLE it_sorted.
  INSERT wa INTO TABLE it_hashed.

  wa-id = '2'.
  wa-name = 'Mary'.
  wa-value = 60.

  APPEND wa TO it_standard.
  INSERT wa INTO TABLE it_sorted.
  INSERT wa INTO TABLE it_hashed.

  wa-id = '1'.
  wa-name = 'Max'.
  wa-value = 30.

  APPEND wa TO it_standard.
  INSERT wa INTO TABLE it_sorted.
  INSERT wa INTO TABLE it_hashed.

  BREAK-POINT.
```

Write an executable program which has a routine that receives an internal table and print how many fields are filled with their default value (the line type of the table must have at least 4 fields).

Hint: each primitive type has a default value. For example, 0 (zero) is the default value of integers whereas space (' ') is the default value of characters.

Solution:

```
REPORT  z_abap101_061.

TYPES:
  BEGIN OF ty_line,
    id TYPE c LENGTH 10,
    name TYPE string,
    value TYPE i,
    creation_date TYPE d,
  END OF ty_line.

TYPES: ty_table TYPE STANDARD TABLE OF ty_line.

*&---------------------------------------------------------------------*
*&      Form  count_initial_fields_of_table
*&---------------------------------------------------------------------*
*  Counts how many fields are filled with their default value
*----------------------------------------------------------------------*
*      -->US_TABLE   internal table
*----------------------------------------------------------------------*
FORM count_initial_fields_of_table USING us_table TYPE ty_table.

  DATA lwa_line TYPE ty_line.
  DATA lv_initial_field_total TYPE i.

  LOOP AT us_table INTO lwa_line.

    IF lwa_line-id IS INITIAL.
      lv_initial_field_total = lv_initial_field_total + 1.
    ENDIF.

    IF lwa_line-name IS INITIAL.
      lv_initial_field_total = lv_initial_field_total + 1.
    ENDIF.

    IF lwa_line-value IS INITIAL.
      lv_initial_field_total = lv_initial_field_total + 1.
    ENDIF.

    IF lwa_line-creation_date IS INITIAL.
      lv_initial_field_total = lv_initial_field_total + 1.
    ENDIF.
```

```
  ENDLOOP.

  WRITE: 'Number of initial values: ', lv_initial_field_total.
  NEW-LINE.
ENDFORM.                    "count_initial_fields_of_table

START-OF-SELECTION.

  DATA itab TYPE ty_table.
  DATA wa TYPE ty_line.

  wa-id = '1'.
  wa-name = 'John'.
  wa-value = 50.
  wa-creation_date = '20140727'.

  APPEND wa TO itab.
  CLEAR wa.

  PERFORM count_initial_fields_of_table USING itab.

  wa-id = '2'.
  wa-name = 'Mary'.
  wa-value = 20.
*  wa-creation_date = ?.

  APPEND wa TO itab.
  CLEAR wa.

  PERFORM count_initial_fields_of_table USING itab.

  wa-id = '3'.
  wa-name = 'Max'.
*  wa-value = ?.
*  wa-creation_date = ?.

  APPEND wa TO itab.
  CLEAR wa.

  PERFORM count_initial_fields_of_table USING itab.

  wa-id = '4'.
*  wa-name = ?.
*  wa-value = ?.
*  wa-creation_date = ?.

  APPEND wa TO itab.
  CLEAR wa.

  PERFORM count_initial_fields_of_table USING itab.
```

Write an executable program which has a routine that receives an internal table and prints how many fields are blank by line (the type of table must have at least 4 fields). Output must be generated as:

```
Line [line number] => [number of blank fields] + " blank fields"
Total: [total number of blank fields]
```

Solution:

```
REPORT  z_abap101_062.

TYPES:
  BEGIN OF ty_line,
    id TYPE c LENGTH 10,
    name TYPE string,
    value TYPE i,
    creation_date TYPE d,
  END OF ty_line.

TYPES: ty_table TYPE STANDARD TABLE OF ty_line.

*&---------------------------------------------------------------------*
*&      Form  count_initial_fields_of_line
*&---------------------------------------------------------------------*
* Counts and prints how many fields are blank by line
*----------------------------------------------------------------------*
*      -->US_TABLE   internal table
*----------------------------------------------------------------------*
FORM count_initial_fields_of_line USING us_table TYPE ty_table.

  DATA lwa_line TYPE ty_line.
  DATA lv_initial_field_total TYPE i .
  DATA lv_initial_field TYPE i.

  LOOP AT us_table INTO lwa_line.
    CLEAR lv_initial_field .

    IF lwa_line-id IS INITIAL.
      lv_initial_field = lv_initial_field + 1.
    ENDIF.

    IF lwa_line-name IS INITIAL.
      lv_initial_field = lv_initial_field + 1.
    ENDIF.

    IF lwa_line-value IS INITIAL.
      lv_initial_field = lv_initial_field + 1.
    ENDIF.

    IF lwa_line-creation_date IS INITIAL.
```

```
      lv_initial_field = lv_initial_field + 1.
    ENDIF.

    WRITE: 'Line ', sy-tabix, ' => ', lv_initial_field, ' blank fields'.
    NEW-LINE.

    lv_initial_field_total = lv_initial_field_total + lv_initial_field.
  ENDLOOP.

  WRITE: 'Total: ', lv_initial_field_total.
  WRITE: sy-uline.
ENDFORM.                    "count_initial_fields_of_line

START-OF-SELECTION.

  DATA itab TYPE ty_table.
  DATA wa TYPE ty_line.

  wa-id = '1'.
  wa-name = 'John'.
  wa-value = 50.
  wa-creation_date = '20140727'.

  APPEND wa TO itab.
  CLEAR wa.

  PERFORM count_initial_fields_of_line USING itab.

  wa-id = '2'.
  wa-name = 'Mary'.
  wa-value = 20.
*  wa-creation_date = ?.

  APPEND wa TO itab.
  CLEAR wa.

  PERFORM count_initial_fields_of_line USING itab.

  wa-id = '3'.
  wa-name = 'Max'.
*  wa-value = ?.
*  wa-creation_date = ?.

  APPEND wa TO itab.
  CLEAR wa.

  PERFORM count_initial_fields_of_line USING itab.

  wa-id = '4'.
*  wa-name = ?.
*  wa-value = ?.
```

```
*  wa-creation_date = ?.

  APPEND wa TO itab.
  CLEAR wa.

  PERFORM count_initial_fields_of_line USING itab.
```

Write an executable program which has a routine that receives a standard internal table. The line type used in the internal table declaration must contain at least three components any text type. The routine should replace all occurrences of "space" by a "_" (underscore) using work areas (not field symbols). Print the table contents before and after calling the routine. The internal table must be populated with at least 10 records and contemplating some fields that have "space" in all field values, other records containing spaces in just a few fields and other records without spaces at all.

Solution:

```
REPORT  z_abap101_063.

TYPES: BEGIN OF ty_line,
   comp1 TYPE c LENGTH 10,
   comp2 TYPE string,
   comp3 TYPE c LENGTH 5,
END OF ty_line.

TYPES: ty_tt_line TYPE TABLE OF ty_line.

*&---------------------------------------------------------------------*
*&      Form  replace_spaces
*&---------------------------------------------------------------------*
* Replaces all occurrences of "space" by a "_" (underscore)
*  using work areas (not field symbols).
*----------------------------------------------------------------------*
*      -->CH_ITAB    text
*----------------------------------------------------------------------*
FORM replace_spaces CHANGING ch_itab TYPE ty_tt_line.

  DATA lwa TYPE ty_line.

  LOOP AT ch_itab INTO lwa.
    REPLACE ALL OCCURRENCES OF REGEX '\s' IN lwa-comp1 WITH '_' IN CHARACTER
MODE.
    REPLACE ALL OCCURRENCES OF REGEX '[[:space:]]' IN lwa-comp2 WITH '_' IN
CHARACTER MODE.
    REPLACE ALL OCCURRENCES OF REGEX '\s' IN lwa-comp3 WITH '_' IN CHARACTER
MODE.
    MODIFY ch_itab INDEX sy-tabix FROM lwa.
  ENDLOOP.

ENDFORM.                    "replace_spaces

*&---------------------------------------------------------------------*
*&      Form  print_itab
*&---------------------------------------------------------------------*
* Prints internal table contents
*----------------------------------------------------------------------*
```

```
*      -->US_ITAB    text
*----------------------------------------------------------------------*
FORM print_itab USING us_itab TYPE ty_tt_line.

  DATA lwa TYPE ty_line.

  LOOP AT us_itab INTO lwa.
    WRITE: lwa-comp1 COLOR 1. NEW-LINE.
    WRITE: lwa-comp2 COLOR 2. NEW-LINE.
    WRITE: lwa-comp3 COLOR 3. NEW-LINE.
    WRITE /.
  ENDLOOP.

ENDFORM.                    "print_itab

START-OF-SELECTION.

  DATA itab TYPE ty_tt_line.
  DATA wa TYPE ty_line.

  wa-comp1 = 'ABAP 101'.
  wa-comp2 = 'One Two Three Four Five Six Seven Eight Nine'.
  wa-comp3 = '12345'.
  APPEND wa TO itab.
  CLEAR wa.

  wa-comp1 = 'ABAP101'.
  wa-comp2 = 'One/Two/Three/Four    Five/Six/Seven/Eight/Nine'.
  wa-comp3 = '12 45'.
  APPEND wa TO itab.
  CLEAR wa.

  wa-comp1 = ' '.
  wa-comp2 = 'One/Two/Three/Four+=_-Five/Six/Seven/Eight/Nine'.
  wa-comp3 = ''.
  APPEND wa TO itab.
  CLEAR wa.

  WRITE: 'Before replace'. NEW-LINE.
  PERFORM print_itab
    USING
      itab.

  PERFORM replace_spaces
    CHANGING
      itab.

  WRITE: /, 'After replace'. NEW-LINE.
  PERFORM print_itab
    USING
      itab.
```

Write an executable program which has a routine that receives a standard internal table. The line type used in the internal table declaration must contain at least three components any text type. The routine should replace all occurrences of "space" by a "_" (underscore) using field symbols (and not work areas). Print the table contents before and after calling the routine. The internal table must be populated with at least 10 records and contemplating some fields that have "space" in all field values, other records containing spaces in just a few fields and other records without spaces at all.

Solution:

```
REPORT  z_abap101_064.

TYPES: BEGIN OF ty_line,
   comp1 TYPE c LENGTH 10,
   comp2 TYPE string,
   comp3 TYPE c LENGTH 5,
END OF ty_line.

TYPES: ty_tt_line TYPE TABLE OF ty_line.

*&---------------------------------------------------------------------*
*&      Form  replace_spaces
*&---------------------------------------------------------------------*
* Replaces all occurrences of "space" by a "_" (underscore)
*  using field symbols (not work areas).
*----------------------------------------------------------------------*
*      -->CH_ITAB    text
*----------------------------------------------------------------------*
FORM replace_spaces CHANGING ch_itab TYPE ty_tt_line.

  FIELD-SYMBOLS <line> TYPE ty_line.
  LOOP AT ch_itab ASSIGNING <line>.
    REPLACE ALL OCCURRENCES OF REGEX '\s' IN <line>-comp1 WITH '_' IN CHARACTER
MODE.
    REPLACE ALL OCCURRENCES OF REGEX '[[:space:]]' IN <line>-comp2 WITH '_' IN
CHARACTER MODE.
    REPLACE ALL OCCURRENCES OF REGEX '\s' IN <line>-comp3 WITH '_' IN CHARACTER
MODE.
  ENDLOOP.

ENDFORM.                    "replace_spaces

*&---------------------------------------------------------------------*
*&      Form  print_itab
*&---------------------------------------------------------------------*
* Prints internal table contents
*----------------------------------------------------------------------*
*      -->US_ITAB    text
```

```
*---------------------------------------------------------------------*
FORM print_itab USING us_itab TYPE ty_tt_line.

  FIELD-SYMBOLS <line> TYPE ty_line.

  LOOP AT us_itab ASSIGNING <line>.
    WRITE: <line>-comp1 COLOR 1. NEW-LINE.
    WRITE: <line>-comp2 COLOR 2. NEW-LINE.
    WRITE: <line>-comp3 COLOR 3. NEW-LINE.
    WRITE /.
  ENDLOOP.

ENDFORM.                    "print_itab

START-OF-SELECTION.

  DATA itab TYPE ty_tt_line.
  DATA wa TYPE ty_line.

  wa-comp1 = 'ABAP 101'.
  wa-comp2 = 'One Two Three Four Five Six Seven Eight Nine'.
  wa-comp3 = '12345'.
  APPEND wa TO itab.
  CLEAR wa.

  wa-comp1 = 'ABAP101'.
  wa-comp2 = 'One/Two/Three/Four    Five/Six/Seven/Eight/Nine'.
  wa-comp3 = '12 45'.
  APPEND wa TO itab.
  CLEAR wa.

  wa-comp1 = ' '.
  wa-comp2 = 'One/Two/Three/Four+=_-Five/Six/Seven/Eight/Nine'.
  wa-comp3 = ''.
  APPEND wa TO itab.
  CLEAR wa.

  WRITE: 'Before replace'. NEW-LINE.
  PERFORM print_itab
    USING
      itab.

  PERFORM replace_spaces
    CHANGING
      itab.

  WRITE: /, 'After replace'. NEW-LINE.
  PERFORM print_itab
    USING
      itab.
```

Write an executable program which has a routine that receives an internal table of strings and concatenates their values in four different ways:

1. Way 1: concatenate internal table texts by the line order
2. Way 2: concatenate internal table texts by the text ascending order
3. Way 3: concatenate internal table texts by the text descending order
4. Way 4: concatenate internal table texts by the line reverse order

Solution:

```
REPORT  z_abap101_065.

*&---------------------------------------------------------------------*
*&      Form  concatenate_strings
*&---------------------------------------------------------------------*
* Concatenate strings from an internal table
*----------------------------------------------------------------------*
*      -->US_ITAB                 Internal table of strings
*      -->CH_CONCATENATED_STRING  Result
*----------------------------------------------------------------------*
FORM concatenate_strings
    USING
        us_t_strings TYPE table_of_strings
    CHANGING
        ch_concatenated_string TYPE string.

  DATA t_copied_strings LIKE us_t_strings.
  t_copied_strings = us_t_strings.

  FIELD-SYMBOLS <string_line> TYPE string.

  LOOP AT t_copied_strings ASSIGNING <string_line>.
    CONCATENATE ch_concatenated_string <string_line> INTO
ch_concatenated_string.
  ENDLOOP.

ENDFORM.                    "concatenate_strings

*&---------------------------------------------------------------------*
*&      Form  concatenate_strings_in_ways
*&---------------------------------------------------------------------*
*  Receives an internal table of strings and concatenates
*   their values in four different ways:
*  Way 1: concatenate internal table texts by the line order
*  Way 2: concatenate internal table texts by the text ascending order
*  Way 3: concatenate internal table texts by the text descending order
*  Way 4: concatenate internal table texts by the line reverse order
*----------------------------------------------------------------------*
```

```
*      -->US_T_STRINGS            Table of strings
*      -->US_V_CONCAT_LOGIC       Concatenation way # (1-4)
*      -->CH_CONCATENATED_STRING  Concatenated string
*---------------------------------------------------------------------*
FORM concatenate_strings_in_ways
    USING
        us_t_strings TYPE table_of_strings
        us_v_concat_logic TYPE c
    CHANGING ch_concatenated_string TYPE string.

  DATA t_copied_strings LIKE us_t_strings.
  t_copied_strings = us_t_strings.

  CASE us_v_concat_logic.
    WHEN '1'.

      PERFORM concatenate_strings
        USING
          t_copied_strings
        CHANGING
          ch_concatenated_string
        .

    WHEN '2'.
      SORT t_copied_strings.

      PERFORM concatenate_strings
        USING
          t_copied_strings
        CHANGING
          ch_concatenated_string
        .

    WHEN '3'.
      SORT t_copied_strings DESCENDING.

      PERFORM concatenate_strings
        USING
          t_copied_strings
        CHANGING
          ch_concatenated_string
        .

    WHEN '4'.

* reverse loop
      DATA vl_number_of_strings TYPE i.
      FIELD-SYMBOLS <string_line> TYPE string.
      DESCRIBE TABLE t_copied_strings LINES vl_number_of_strings.

      WHILE vl_number_of_strings > 0.
```

```
        READ TABLE t_copied_strings ASSIGNING <string_line> INDEX
vl_number_of_strings.
        IF sy-subrc IS INITIAL.
          CONCATENATE ch_concatenated_string <string_line> INTO
ch_concatenated_string.
        ENDIF.

        UNASSIGN <string_line>.

        vl_number_of_strings = vl_number_of_strings - 1.
      ENDWHILE.

  ENDCASE.

ENDFORM.                    "concatenate_strings_in_ways

START-OF-SELECTION.

  DATA it_strings TYPE table_of_strings.
  DATA gv_concatenated TYPE string.

  APPEND 'A' TO it_strings.
  APPEND 'B' TO it_strings.
  APPEND 'C' TO it_strings.
  APPEND 'D' TO it_strings.
  APPEND 'X' TO it_strings.
  APPEND 'Y' TO it_strings.
  APPEND 'Z' TO it_strings.
  APPEND 'M' TO it_strings.
  APPEND 'N' TO it_strings.
  APPEND 'O' TO it_strings.

  PERFORM concatenate_strings_in_ways
    USING
      it_strings
      '1'
    CHANGING
      gv_concatenated
    .
  WRITE: '1 - ', gv_concatenated, /.
  CLEAR gv_concatenated.

  PERFORM concatenate_strings_in_ways
    USING
      it_strings
      '2'
    CHANGING
      gv_concatenated
    .
  WRITE: '2 - ', gv_concatenated, /.
  CLEAR gv_concatenated.
```

```
PERFORM concatenate_strings_in_ways
  USING
    it_strings
    '3'
  CHANGING
    gv_concatenated
  .
WRITE: '3 - ', gv_concatenated, /.
CLEAR gv_concatenated.

PERFORM concatenate_strings_in_ways
  USING
    it_strings
    '4'
  CHANGING
    gv_concatenated
  .
WRITE: '4 - ', gv_concatenated, /.
CLEAR gv_concatenated.
```

Write an executable program with two parameters types as integers. The first represents a number to be printed and the second represents the length of the number to be printed. Place zeros to the left if necessary. Example:

- p_number = 15 p_length = 2. Output = 15
- p_number = 15 p_length = 6. Output = 000015
- p_number = 2014 p_length = 2. Output = 20
- p_number = 123456789 p_length = 10. Output = 0123456789
- p_number = 123456789 p_length = 4. Output = 1234

Solution:

```
REPORT  z_abap101_066.

PARAMETERS:
    p_number TYPE i,
    p_length TYPE i.

START-OF-SELECTION.

  DATA vl_number_string TYPE string.
  DATA vl_number_length TYPE i.
  DATA vl_number_with_left_zeros TYPE string.
  vl_number_string = p_number.
  CONDENSE vl_number_string NO-GAPS.
  vl_number_length = strlen( vl_number_string ).

  IF vl_number_length > p_length.
    WRITE: vl_number_string(p_length).
  ELSE.
    DATA vl_zeros TYPE i.
    DATA vl_left_zeros TYPE string.
    vl_zeros = p_length - vl_number_length.

    DO vl_zeros TIMES.
      CONCATENATE vl_left_zeros '0' INTO vl_left_zeros.
    ENDDO.
    CONCATENATE vl_left_zeros vl_number_string INTO vl_number_with_left_zeros.
    WRITE: vl_number_with_left_zeros.
  ENDIF.
```

Write an executable program with two parameters which represents a base and exponent. Print the result of exponentiation. As both parameters are required for the operation they should be mandatory.

Solution:

```
REPORT  z_abap101_067.

PARAMETERS p_base TYPE p LENGTH 5 OBLIGATORY.
PARAMETERS p_exp TYPE i OBLIGATORY.

START-OF-SELECTION.

  DATA v_result TYPE f.

  v_result = p_base ** p_exp.

  WRITE: v_result EXPONENT 0.
```

Write an executable program with two parameters (a string and a number) The number should be accepted only if it's less or equal to 25. The program should print the string as many times as the value of the numeric parameter. The output should be as following

String = "ABAPers are not crazy people." Number = 21.

```
Line [1]: A
Line [2]: AB
Line [3]: ABA
Line [4]: ABAP
Line [5]: ABAPe
(...)
Line [21]: ABAPers are not crazy
```

Solution:

```
REPORT  z_abap101_068.

PARAMETERS p_text TYPE string OBLIGATORY.
PARAMETER p_len TYPE i OBLIGATORY.

AT SELECTION-SCREEN ON p_len.

  IF p_len > 25.
    MESSAGE 'P_LEN should be less or equal 25' TYPE 'E'.
  ENDIF.

START-OF-SELECTION.

  DO p_len TIMES.
    WRITE: 'Line [', sy-index , ']: ', p_text(sy-index).
    NEW-LINE.
  ENDDO.
```

Write an executable program which has two internal tables, with a header line and the other without. Add five records in each table. In the case of the one with header line, use it embed work area. For the other one, use a work area declared explicitly. Print the contents of both internal tables.

Solution:

```
REPORT  z_abap101_069.

TYPES: BEGIN OF ty_line,
id TYPE n LENGTH 8,
name TYPE c LENGTH 20,
age TYPE i,
END OF ty_line.

DATA it_without_hat TYPE TABLE OF ty_line.
DATA it_with_hat TYPE TABLE OF ty_line WITH HEADER LINE. " Debug me to
understand my name

* Populating the internal table WITHOUT HEADER LINE (hat)
DATA wa_line TYPE ty_line.

wa_line-id = 1.
wa_line-name = 'The One'.
wa_line-age = 10.
APPEND wa_line TO it_without_hat.

wa_line-id = 2.
wa_line-name = 'Bob'.
wa_line-age = 20.
APPEND wa_line TO it_without_hat.

wa_line-id = 3.
wa_line-name = 'Mary'.
wa_line-age = 30.
APPEND wa_line TO it_without_hat.

wa_line-id = 4.
wa_line-name = 'Chris'.
wa_line-age = 40.
APPEND wa_line TO it_without_hat.

wa_line-id = 5.
wa_line-name = 'Janet'.
wa_line-age = 50.
APPEND wa_line TO it_without_hat.

* Populating the internal table WITH HEADER LINE (hat)
```

```
it_with_hat-id = 1.
it_with_hat-name = 'The One'.
it_with_hat-age = 10.
APPEND it_with_hat.

it_with_hat-id = 2.
it_with_hat-name = 'Bob'.
it_with_hat-age = 20.
APPEND it_with_hat.

it_with_hat-id = 3.
it_with_hat-name = 'Mary'.
it_with_hat-age = 30.
APPEND it_with_hat.

it_with_hat-id = 4.
it_with_hat-name = 'Chris'.
it_with_hat-age = 40.
APPEND it_with_hat.

it_with_hat-id = 5.
it_with_hat-name = 'Janet'.
it_with_hat-age = 50.
APPEND it_with_hat.

* Printing table WITHOUT header line
LOOP AT it_without_hat INTO wa_line.
  WRITE: wa_line-id, wa_line-name, wa_line-age.
  NEW-LINE.
ENDLOOP.

* Printing table WITH header line
LOOP AT it_with_hat.
  WRITE: it_with_hat-id, it_with_hat-name, it_with_hat-age.
  NEW-LINE.
ENDLOOP.

NEW-LINE.

*IF it_without_hat = it_with_hat. " Try to uncomment this IF
*  WRITE 'Maybe the tables are not so equal because...'.
*  NEW-LINE.
*ENDIF.

IF it_without_hat = it_with_hat[].
  WRITE ' ... without using []s we are using the work area and not the internal
table'.
ENDIF.
```

Have a routine that receives an internal table (with at least three columns) and the sort it by its first column.

Solution:

```
REPORT  z_abap101_070.

TYPES: BEGIN OF ty_person,
  id TYPE n LENGTH 8,
  name TYPE c LENGTH 20,
  age TYPE i,
END OF ty_person,
tt_people TYPE TABLE OF ty_person WITH KEY id.

DATA it_people TYPE tt_people.

FORM sort_1st_column CHANGING ch_itab_people TYPE tt_people.

  SORT ch_itab_people BY id ASCENDING.

ENDFORM.                    "sort_1st_column

START-OF-SELECTION.

* Populating the internal table WITHOUT HEADER LINE (hat)
  DATA wa_person TYPE ty_person.

  wa_person-id = 3.
  wa_person-name = 'The One'.
  wa_person-age = 30.
  APPEND wa_person TO it_people.

  wa_person-id = 2.
  wa_person-name = 'Bob'.
  wa_person-age = 20.
  APPEND wa_person TO it_people.

  wa_person-id = 1.
  wa_person-name = 'Mary'.
  wa_person-age = 10.
  APPEND wa_person TO it_people.

  wa_person-id = 5.
  wa_person-name = 'Chris'.
  wa_person-age = 50.
  APPEND wa_person TO it_people.

  wa_person-id = 4.
```

```
wa_person-name = 'Janet'.
wa_person-age = 40.
APPEND wa_person TO it_people.

WRITE 'Before SORT'. NEW-LINE.
LOOP AT it_people INTO wa_person.
  WRITE: wa_person-id, wa_person-name, wa_person-age.
  NEW-LINE.
ENDLOOP.

PERFORM sort_1st_column
  CHANGING
    it_people.

WRITE 'After SORT'. NEW-LINE.
LOOP AT it_people INTO wa_person.
  WRITE: wa_person-id, wa_person-name, wa_person-age.
  NEW-LINE.
ENDLOOP.
```

Have a routine that receives an internal table (with at least three columns) and a string with the name of a column. Sort the table by the specified column accordingly.

Solution:

```
REPORT  z_abap101_071.

TYPES: BEGIN OF ty_person,
  id TYPE n LENGTH 8,
  name TYPE c LENGTH 20,
  age TYPE i,
END OF ty_person,
tt_people TYPE TABLE OF ty_person WITH KEY id.

DATA it_people TYPE tt_people.

FORM sort_any_column
    USING us_v_column TYPE string
    CHANGING ch_itab_people TYPE tt_people.

  SORT ch_itab_people BY (us_v_column) ASCENDING.

ENDFORM.                    "sort_any_column

START-OF-SELECTION.

* Populating the internal table WITHOUT HEADER LINE (hat)
  DATA wa_person TYPE ty_person.

  wa_person-id = 3.
  wa_person-name = 'The One'.
  wa_person-age = 30.
  APPEND wa_person TO it_people.

  wa_person-id = 2.
  wa_person-name = 'Bob'.
  wa_person-age = 20.
  APPEND wa_person TO it_people.

  wa_person-id = 1.
  wa_person-name = 'Mary'.
  wa_person-age = 10.
  APPEND wa_person TO it_people.

  wa_person-id = 5.
  wa_person-name = 'Chris'.
  wa_person-age = 50.
  APPEND wa_person TO it_people.
```

```
wa_person-id = 4.
wa_person-name = 'Janet'.
wa_person-age = 40.
APPEND wa_person TO it_people.

WRITE 'Before SORT by ID'. NEW-LINE.
LOOP AT it_people INTO wa_person.
  WRITE: wa_person-id, wa_person-name, wa_person-age.
  NEW-LINE.
ENDLOOP.

PERFORM sort_any_column
  USING
    'ID'
  CHANGING
    it_people.

WRITE 'After SORT ID'. NEW-LINE.
LOOP AT it_people INTO wa_person.
  WRITE: wa_person-id, wa_person-name, wa_person-age.
  NEW-LINE.
ENDLOOP.

PERFORM sort_any_column
USING
  'NAME'
CHANGING
  it_people.

WRITE 'After SORT NAME'. NEW-LINE.
LOOP AT it_people INTO wa_person.
  WRITE: wa_person-id, wa_person-name, wa_person-age.
  NEW-LINE.
ENDLOOP.
```

Have a routine that receives an internal table (with at least three fields) and another internal table with the name of the columns to be ordered and order accordingly.

Solution:

```
REPORT  z_abap101_072.

TYPES: BEGIN OF ty_person,
  id TYPE n LENGTH 8,
  name TYPE c LENGTH 20,
  age TYPE i,
END OF ty_person,
tt_people TYPE TABLE OF ty_person WITH KEY id.

DATA it_people TYPE tt_people.

FORM print_people USING us_itab_people TYPE tt_people.
  DATA lwa_person TYPE ty_person.
  LOOP AT us_itab_people INTO lwa_person.
    WRITE: lwa_person-id, lwa_person-name, lwa_person-age.
    NEW-LINE.
  ENDLOOP.
ENDFORM.                    "print_people

FORM sort_any_columns
    USING us_t_columns TYPE table_of_strings
    CHANGING ch_itab_people TYPE tt_people.

  DATA lv_number_of_lines TYPE i.

  DESCRIBE TABLE us_t_columns LINES lv_number_of_lines.

  DATA lv_first_column TYPE string.
  DATA lv_second_column TYPE string.
  DATA lv_third_column TYPE string.

  CASE lv_number_of_lines.
    WHEN 1.
      READ TABLE us_t_columns INDEX 1 INTO lv_first_column.

      SORT ch_itab_people BY
      (lv_first_column) ASCENDING.

    WHEN 2.
      READ TABLE us_t_columns INDEX 1 INTO lv_first_column.
      READ TABLE us_t_columns INDEX 2 INTO lv_second_column.

      SORT ch_itab_people BY
```

```
      (lv_first_column) ASCENDING
      (lv_second_column) ASCENDING..

    WHEN 3.
      READ TABLE us_t_columns INDEX 1 INTO lv_first_column.
      READ TABLE us_t_columns INDEX 2 INTO lv_second_column.
      READ TABLE us_t_columns INDEX 3 INTO lv_third_column.

      SORT ch_itab_people BY
      (lv_first_column) ASCENDING
      (lv_second_column) ASCENDING
      (lv_third_column) ASCENDING.

  ENDCASE.

ENDFORM.                    "sort_any_columns

START-OF-SELECTION.

* Populating the internal table WITHOUT HEADER LINE (hat)
  DATA wa_person TYPE ty_person.

  wa_person-id = 3.
  wa_person-name = 'The One'.
  wa_person-age = 30.
  APPEND wa_person TO it_people.

  wa_person-id = 6.
  wa_person-name = 'Peter'.
  wa_person-age = 40.
  APPEND wa_person TO it_people.

  wa_person-id = 2.
  wa_person-name = 'Bob'.
  wa_person-age = 30.
  APPEND wa_person TO it_people.

  wa_person-id = 1.
  wa_person-name = 'Mary'.
  wa_person-age = 10.
  APPEND wa_person TO it_people.

  wa_person-id = 5.
  wa_person-name = 'Chris'.
  wa_person-age = 20.
  APPEND wa_person TO it_people.

  wa_person-id = 4.
  wa_person-name = 'Bob'.
  wa_person-age = 40.
  APPEND wa_person TO it_people.
```

```
DATA it_sort_columns TYPE table_of_strings.

WRITE 'Before SORT'. NEW-LINE.
PERFORM print_people USING it_people.

APPEND `NAME` TO it_sort_columns.

PERFORM sort_any_columns
  USING
    it_sort_columns
  CHANGING
    it_people.

REFRESH it_sort_columns.

WRITE 'After SORT NAME'. NEW-LINE.
PERFORM print_people USING it_people.

APPEND `AGE` TO it_sort_columns.
APPEND `ID` TO it_sort_columns.

PERFORM sort_any_columns
USING
  it_sort_columns
CHANGING
  it_people.

REFRESH it_sort_columns.

WRITE 'After SORT AGE/ID'. NEW-LINE.
PERFORM print_people USING it_people.

APPEND `AGE` TO it_sort_columns.
APPEND `NAME` TO it_sort_columns.
APPEND `ID` TO it_sort_columns.

PERFORM sort_any_columns
USING
  it_sort_columns
CHANGING
  it_people.

REFRESH it_sort_columns.

WRITE 'After SORT AGE/NAME/ID'. NEW-LINE.
PERFORM print_people USING it_people.
```

Part III – Selection Screens (73 – 101)

Write a program that...

Contains a select-options for numeric values and print the result of multiplying each number within the range of 3.

Solution:

```
REPORT  z_abap101_073.

DATA v_number TYPE i.
SELECT-OPTIONS s_number FOR v_number NO-EXTENSION.

START-OF-SELECTION.

  DATA v_difference TYPE i.
  DATA v_multiplication_result TYPE i.

  v_difference = s_number-high - s_number-low + 1.

  DO v_difference TIMES.
    v_multiplication_result = ( s_number-low + sy-index - 1 ) * 3.
    WRITE v_multiplication_result.
    NEW-LINE.
  ENDDO.
```

Contains a select-options for numeric values and print all search criteria separated ",".

Solution:

```
REPORT  z_abap101_074.

DATA v_number TYPE i.

SELECT-OPTIONS s_number FOR v_number.

START-OF-SELECTION.

  LOOP AT s_number.

    IF s_number-sign = 'I'.
      WRITE: 'Include'.

    ELSE.
      WRITE: 'Exclude'.
    ENDIF.

    CASE s_number-option.

      WHEN 'EQ'.
        WRITE: 'Equal', s_number-low.
      WHEN 'NE'.
        WRITE: 'Not equal', s_number-low.
      WHEN 'LT'.
        WRITE: 'Less than', s_number-low.
      WHEN 'LE'.
        WRITE: 'Less or equal', s_number-low.
      WHEN 'GT'.
        WRITE: 'Greater than', s_number-low.
      WHEN 'GE'.
        WRITE: 'Greater or equal', s_number-low.
      WHEN 'BT'.
        WRITE: 'between', s_number-low, ' and ', s_number-high.
      WHEN 'BT'.
        WRITE: 'not between', s_number-low, ' and ', s_number-high.
      WHEN 'CP'. " Patterns are used in char and string select-options
        WRITE: 'Contains pattern', s_number-low.
      WHEN 'NP'.
        WRITE: 'not the pattern', s_number-low.
    ENDCASE.

    NEW-LINE.

  ENDLOOP.
```

Declare a select-options for numeric values without ranges. Then, validate if the number zero is entered and if it is, show an error message.

Solution:

```
REPORT  z_abap101_075.

DATA v_number TYPE i.

SELECT-OPTIONS s_number FOR v_number NO INTERVALS.

AT SELECTION-SCREEN ON s_number.
  LOOP AT s_number.
    IF s_number-low = '0'.
      MESSAGE 'Number zero is not allowed' TYPE 'E'.
    ENDIF.
  ENDLOOP.
```

Declare a select-options for numeric values without multiple ranges. Then, validate if a range bigger than 100 is entered and if it is, show an error message.

Solution:

```
REPORT  z_abap101_076.

DATA v_number TYPE i.

SELECT-OPTIONS s_number FOR v_number NO-EXTENSION.

AT SELECTION-SCREEN ON s_number.

  DATA lv_range_size TYPE i.

  lv_range_size = s_number-high - s_number-low.

  IF lv_range_size > 100.
    MESSAGE 'Range is too big' TYPE 'E'.
  ENDIF.
```

Declare a parameter as a listbox containing all Airline codes com table SCARR.

Solution:

```
REPORT  z_abap101_077.

PARAMETER p_list TYPE scarr-carrid AS LISTBOX VISIBLE LENGTH 20.
```

Declare three parameters as checkboxes. Each of them will represent a different flight class (first, business and economy).

Solution:

```
REPORT  z_abap101_078.

PARAMETER p_first AS CHECKBOX.
PARAMETER p_busin AS CHECKBOX.
PARAMETER p_econo LIKE p_first AS CHECKBOX.
```

Declare three parameters as radio buttons. Each of them will represent a different flight class (first, business and economy).

Solution:

```
REPORT  z_abap101_079.

PARAMETER p_first RADIOBUTTON GROUP grp1.
PARAMETER p_busin RADIOBUTTON GROUP grp1.
PARAMETER p_econo LIKE p_first RADIOBUTTON GROUP grp1.
```

Declare three parameters as checkboxes. The first one should always be checked once the program is started. Moreover, if the current day is between 1 and 10, the other two checkboxes should be checked as well once the program is started.

Solution:

```
REPORT  z_abap101_080.

PARAMETER p_first AS CHECKBOX DEFAULT abap_true.
PARAMETER p_busin AS CHECKBOX.
PARAMETER p_econo LIKE p_first AS CHECKBOX.

INITIALIZATION.

  IF sy-datum+6(2) >= 1 AND sy-datum+6(2) <= 10.
    p_busin = 'X'.
    p_econo = p_busin.
  ENDIF.
```

Declare three radio buttons and an input field. If any radio button is selected, the input field should be cleared. Note: the field should be cleared as soon any radio buttons is selected and not after the program is executed.

Solution:

```
REPORT  z_abap101_081.

PARAMETER p_first RADIOBUTTON GROUP grp1 USER-COMMAND action.
PARAMETER p_busin RADIOBUTTON GROUP grp1.
PARAMETER p_econo LIKE p_first RADIOBUTTON GROUP grp1.
PARAMETER p_input TYPE string.

AT SELECTION-SCREEN.
  IF sy-ucomm = 'ACTION'.
    CLEAR p_input.
  ENDIF.
```

Declare three radio buttons and two input fields. If the first radio button is selected, both input fields should be displayed and ready for input. If the second one is chosen, the first input field should be mandatory and the second one should blocked for input. If the last radio button is chosen, both input fields should not be displayed in the screen.

Solution:

```
REPORT  z_abap101_082.

PARAMETER p_first RADIOBUTTON GROUP grp1 USER-COMMAND action.
PARAMETER p_busin RADIOBUTTON GROUP grp1.
PARAMETER p_econo LIKE p_first RADIOBUTTON GROUP grp1.
PARAMETER p_input1 TYPE string.
PARAMETER p_input2 TYPE i.

DATA v_last_action LIKE sy-ucomm.

AT SELECTION-SCREEN.
  v_last_action = sy-ucomm.

AT SELECTION-SCREEN OUTPUT.
  IF v_last_action = 'ACTION'.

    CASE 'X'.
      WHEN p_first.
        LOOP AT SCREEN.
          IF screen-name = 'P_INPUT1' OR screen-name = 'P_INPUT2'.
            screen-input = 1.
            MODIFY SCREEN.
          ENDIF.
        ENDLOOP.

      WHEN p_busin.
        LOOP AT SCREEN.
          IF screen-name = 'P_INPUT1'.
            screen-required = 1.
            MODIFY SCREEN.
          ENDIF.

          IF screen-name = 'P_INPUT2'.
            screen-input = 0.
            MODIFY SCREEN.
          ENDIF.

        ENDLOOP.

      WHEN p_econo.
        LOOP AT SCREEN.
```

```
        IF screen-name = 'P_INPUT1' OR screen-name = 'P_INPUT2'.
          screen-input = 0.
          screen-invisible = 1.
          MODIFY SCREEN.
        ENDIF.

      ENDLOOP.

  ENDCASE.

ENDIF.
```

Declare four parameters. The first two should have a character type and the last two a numeric type. Separate each pair in the selection screen using selection screen blocks. Both blocks should contain a frame so it's possible to see the separation between them.

Solution:

```
REPORT  z_abap101_083.

SELECTION-SCREEN BEGIN OF BLOCK b01 WITH FRAME.

PARAMETER p_text TYPE string.
PARAMETER p_char TYPE c LENGTH 10.

SELECTION-SCREEN END OF BLOCK b01.

SELECTION-SCREEN BEGIN OF BLOCK b02 WITH FRAME.

PARAMETER p_int TYPE i.
PARAMETER p_p TYPE p LENGTH 10.

SELECTION-SCREEN END OF BLOCK b02.
```

Declare four parameters. The first two should have a character type and the last two a numeric type. Separate each pair in the selection screen using selection screen blocks. Both blocks should contain a frame so it's possible to see the separation between them. Each frame should have a title. Also, define a text for each parameter label using text elements.

Solution:

```
REPORT  z_abap101_084.

SELECTION-SCREEN BEGIN OF BLOCK b01 WITH FRAME TITLE text-b01. " Define in text
elements

PARAMETER p_text TYPE string. " Define in text elements
PARAMETER p_char TYPE c LENGTH 10. " Define in text elements

SELECTION-SCREEN END OF BLOCK b01.

SELECTION-SCREEN BEGIN OF BLOCK b02 WITH FRAME TITLE text-b02. " Define in text
elements

PARAMETER p_int TYPE i. " Define in text elements
PARAMETER p_p TYPE p LENGTH 10. " Define in text elements

SELECTION-SCREEN END OF BLOCK b02.
```

Declare a parameter with a text element and translate it to a different language. Then log into your system using another language and check which text appears in your program.

Solution:

```
REPORT  z_abap101_085.

PARAMETER p_trans TYPE string. " Text element translated
```

Declare a checkbox and an input field side-by-side inside a frame.

Solution:

```
REPORT  z_abap101_086.

SELECTION-SCREEN BEGIN OF BLOCK b01 WITH FRAME.

  SELECTION-SCREEN BEGIN OF LINE.

    POSITION 1.
        PARAMETERS p_check AS CHECKBOX.

    SELECTION-SCREEN COMMENT 3(4) text-001 FOR FIELD p_text.

    POSITION 10.
        PARAMETERS p_text TYPE string.

  SELECTION-SCREEN END OF LINE.

SELECTION-SCREEN END OF BLOCK b01.
```

Declare a button inside a selection screen and show an information message when it is pressed.

Solution:

```
REPORT  z_abap101_087.

SELECTION-SCREEN PUSHBUTTON 10(8) text-001 USER-COMMAND press. " text-001 =
'Press me'

AT SELECTION-SCREEN.

  IF sy-ucomm = 'PRESS'.
    MESSAGE 'Button was pressed' TYPE 'I'.
  ENDIF.
```

Create a tabbed block with 3 tabs. Each of them should have a different content.

Solution:

```
REPORT  z_abap101_088.

SELECTION-SCREEN BEGIN OF SCREEN 1 AS SUBSCREEN.
PARAMETER p_1 TYPE string.
SELECTION-SCREEN END OF SCREEN 1.

SELECTION-SCREEN BEGIN OF SCREEN 2 AS SUBSCREEN.
PARAMETER p_2 TYPE d.
SELECTION-SCREEN END OF SCREEN 2.

SELECTION-SCREEN BEGIN OF SCREEN 3 AS SUBSCREEN.
PARAMETER p_3 TYPE t.
SELECTION-SCREEN END OF SCREEN 3.

SELECTION-SCREEN BEGIN OF TABBED BLOCK tb FOR 10 LINES.

SELECTION-SCREEN TAB (10) tab1 USER-COMMAND tab1_pressed DEFAULT SCREEN 1.
SELECTION-SCREEN TAB (10) tab2 USER-COMMAND tab2_pressed DEFAULT SCREEN 2.
SELECTION-SCREEN TAB (10) tab3 USER-COMMAND tab3_pressed DEFAULT PROGRAM
z_abap101_088 SCREEN 3.

SELECTION-SCREEN END OF BLOCK tb.

INITIALIZATION.

  tab1 = 'String'.
  tab2 = 'Date'.
  tab3 = 'Time'.
```

Declare three parameters. There should be a horizontal line separating the first two ones and a blank line separating the last two.

Solution:

```
REPORT  z_abap101_089.

PARAMETERS p_1.

SELECTION-SCREEN ULINE.

PARAMETER p_2 TYPE i.

SELECTION-SCREEN SKIP 1.

PARAMETER p_3 TYPE d.
```

Declare a selection screen with 8 parameters and 3 select-options of your choice. Execute your program and save a variant so you refer to your selection any time you want.

Solution:

```
REPORT  z_abap101_090.

SELECTION-SCREEN BEGIN OF BLOCK b01 WITH FRAME TITLE v_person.

PARAMETERS:
  p_name TYPE string LOWER CASE,
  p_born TYPE d,
  p_last TYPE t.

SELECTION-SCREEN SKIP 1.

PARAMETERS p_male RADIOBUTTON GROUP gend.
PARAMETERS p_female RADIOBUTTON GROUP gend.

SELECTION-SCREEN END OF BLOCK b01.

SELECTION-SCREEN SKIP 1.

SELECTION-SCREEN BEGIN OF BLOCK b02 WITH FRAME TITLE v_autho.
PARAMETER p_admin AS CHECKBOX.
PARAMETER p_user AS CHECKBOX.
PARAMETER p_guest AS CHECKBOX.
SELECTION-SCREEN END OF BLOCK b02.

SELECTION-SCREEN SKIP 1.

DATA v_luck TYPE i.
DATA v_bad_luck TYPE i.
DATA v_favourite TYPE p LENGTH 10.

SELECTION-SCREEN BEGIN OF BLOCK b03 WITH FRAME TITLE v_number.

SELECT-OPTIONS:
s_luck FOR v_luck,
s_bad FOR v_bad_luck NO INTERVALS,
s_favo FOR v_favourite NO-EXTENSION.

SELECTION-SCREEN END OF BLOCK b03.

INITIALIZATION.
  v_person = 'Personal Data'.
  v_autho = 'Authorization'.
  v_number = 'Numbers'.
```

Declare a selection screen with two date parameters. The first once should be typed with the primitive type. The second, with type SYST-DATUM. Is there any different between them when filling the selection screen? What about the documentation displayed when you hit F1 key?

Solution:

```
REPORT  z_abap101_091.

PARAMETER p_date1 TYPE d.
PARAMETER p_date2 TYPE syst-datum.
```

Declare a selection screen with two time parameters. The first once should be typed with the primitive type. The second, with type SYST-UZEIT. Is there any different between them when filling the selection screen? What about the documentation displayed when you hit F1 key?

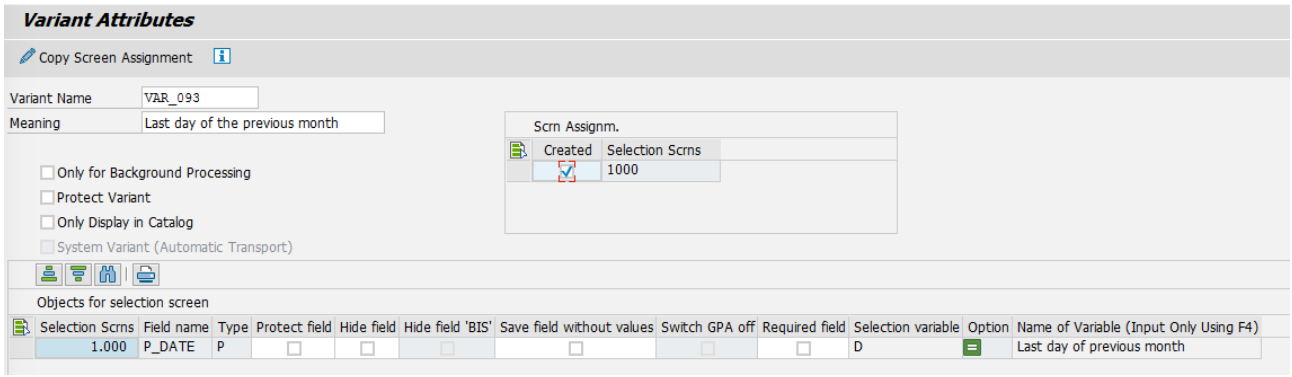
Solution:

```
REPORT  z_abap101_092.

PARAMETERS p_time1 TYPE t.
PARAMETERS p_time2 TYPE syst-uzeit.
```

Declare a selection screen with a parameter representing a date. Then, save a variant so that this field is filled with the last day of the previous month every time the variant is used.

Solution:

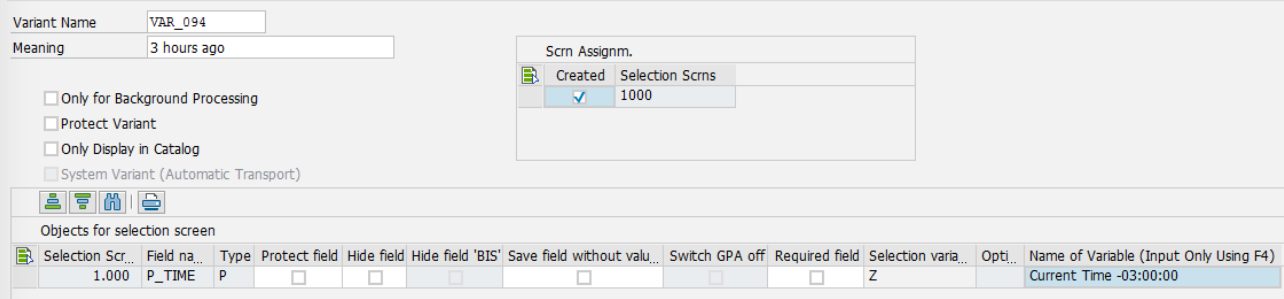
```
REPORT  z_abap101_093.

PARAMETER p_date TYPE d.
```



Declare a selection screen with a parameter representing a specific time. Then, save a variant so that this field is filled with the current time minus 3 hours every time the variant is used.

Solution:

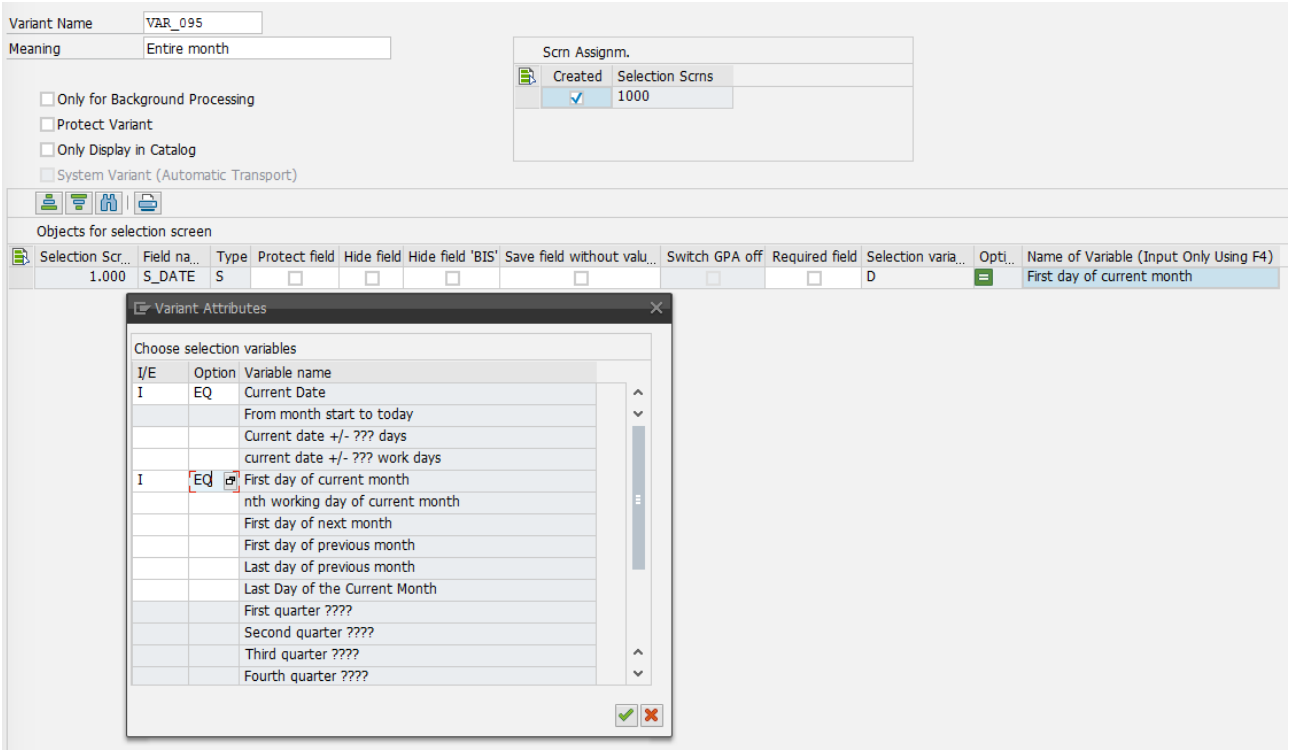
```
REPORT  z_abap101_094.

PARAMETERS p_time TYPE syst-uzeit.
```



Declare a selection screen with a select-options representing a date range. Then, save a variant so that it is filled with a range between the first day of the current month and the current date every time the variant is used.

Solution:

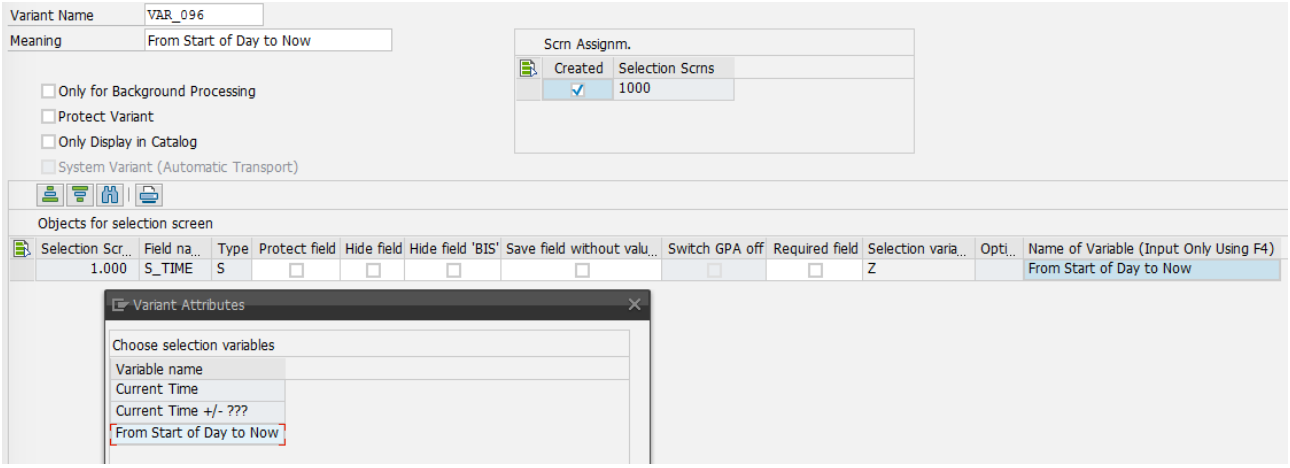
```
REPORT z_abap101_095.

DATA v_date TYPE d.

SELECT-OPTIONS s_date FOR v_date.
```



Declare a selection screen with a select-options representing a time range. Then, save a variant so that these fields are filled with the start of day until the current time every time the variant is used.

Solution:

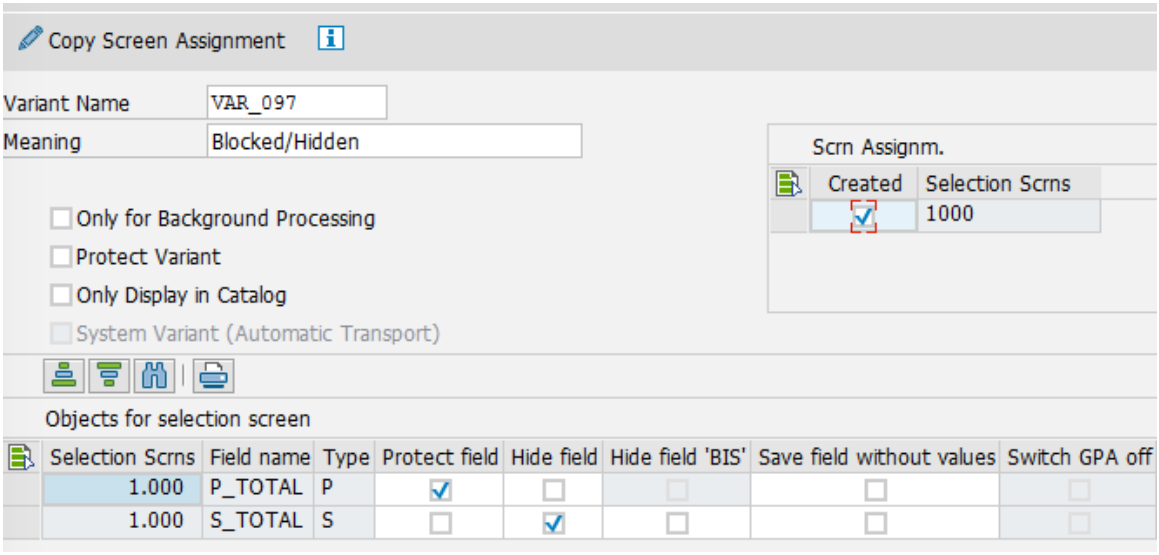
```
REPORT z_abap101_096.

DATA v_time TYPE t.

SELECT-OPTIONS s_time FOR v_time.
```



Declare a selection screen with one parameter and a select-options. Save a variant so that the parameter is blocked for input and the select-options is hidden.

Solution:

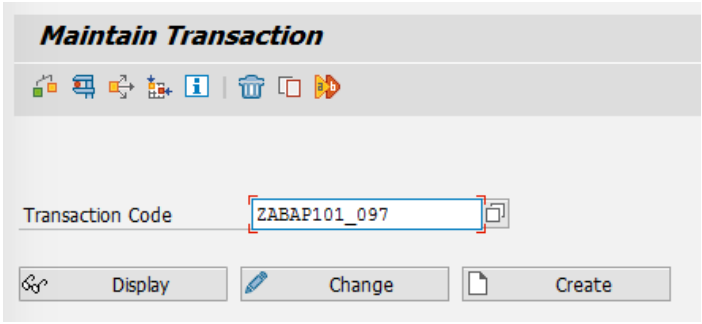
```
REPORT z_abap101_097.

PARAMETERS p_total TYPE p LENGHT 5 DECIMALS 2.

SELECT-OPTIONS s_total FOR p_total.
```



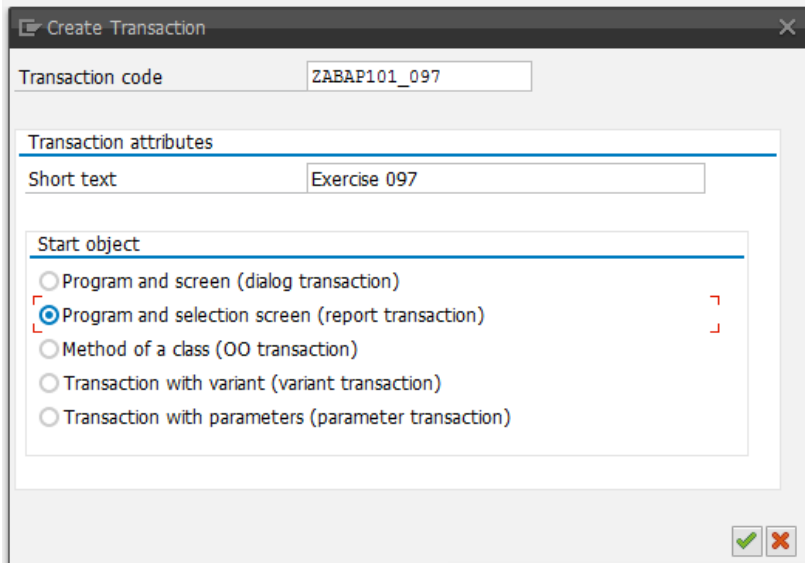
Create a transaction for any of your programs already created.

Solution:

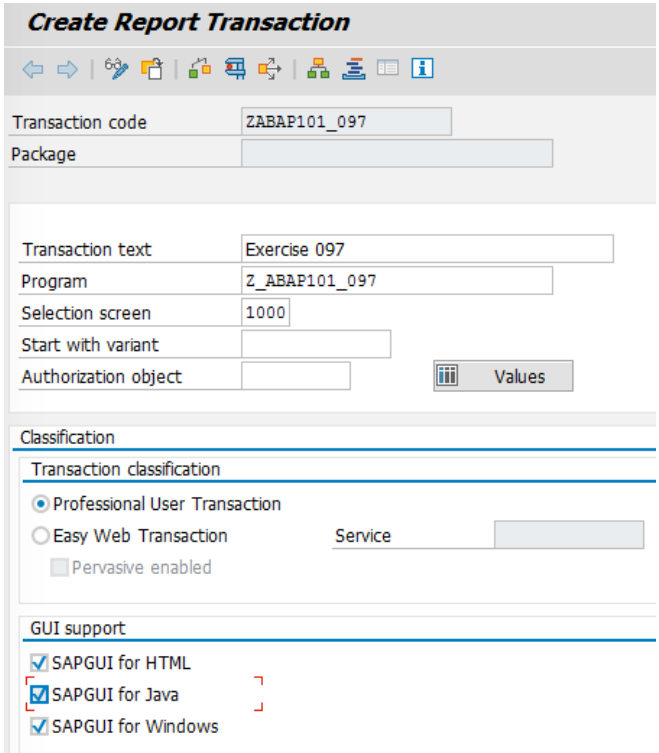
Open transaction SE93 and follow as below.



Click on create and fill the popup with the transaction name and description. It must be a report transaction.

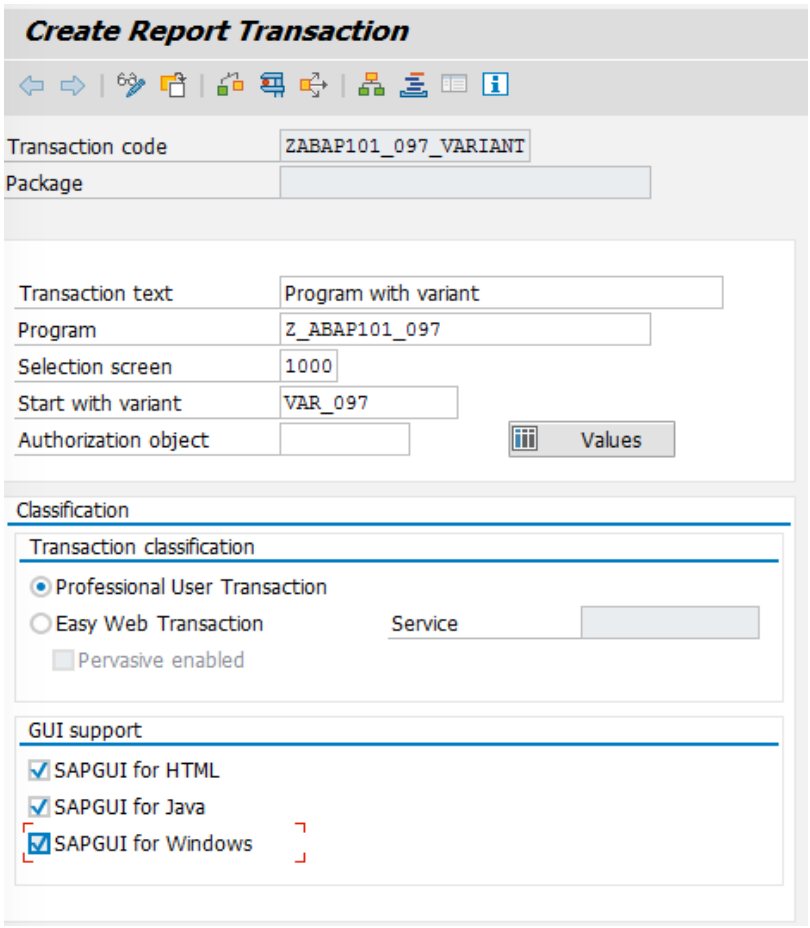


Link your program with the transaction and save.



Create a transaction which points to a program and fills its selection screen automatically using a predefined variant.

Solution:



Create a program with one parameter representing an executable program name (also known as report). The program should execute the program entered in the parameter.

Solution:

```
REPORT Z_ABAP101_100.

PARAMETERS p_report TYPE char20 DEFAULT 'Z_ABAP101_'.

START-OF-SELECTION.

SUBMIT (p_report) VIA SELECTION-SCREEN AND RETURN.
```

Create a program with one select-options representing an executable program name (also known as report). The program should execute all programs entered in the select-options one-by-one. Keep in mind that after running a program, the execution should return back to the original program.

Solution:

```
REPORT z_abap101_101.

DATA v_report TYPE char20.
SELECT-OPTIONS s_report FOR v_report NO INTERVALS.

START-OF-SELECTION.

  LOOP AT s_report.

    SUBMIT (s_report-low) VIA SELECTION-SCREEN AND RETURN.

  ENDLOOP.
```

Appendices

Publishing your answers on the Internet

Doing 101 exercises is great. Showing the world (including recruiters) your answers is even better.

We recommend using GitHub.com to publish you answers. Learning how to use git (a software which helps you versioning files) and GitHub (a website where you can publish and share code) is beyond the scope of this ebook.

ABAP programs are not stored locally on developer's computer. In order to export your programs to a local file you can use the “export program” option inside SE38 and SE80 or do it massively using an open source project called SAPLink (saplink.org), Learning how to use SAPLink is also beyond the scope of this ebook.

Check all these websites if you are interested on sharing the work you have done.

The Authors

Fábio Pagoti is a software developer and ABAP Instructor at Ka Solution working with SAP software since 2009 when he joined Nestlé Brazil as an intern. He is an author of ABAP101.com and creator of the website HanaBrasil.com.br. As the last name implies, Fábio is also focusing on development for SAP Hana and UI5.

Flávio Furlan is a SAP certified ABAP developer and instructor, and founder of website ABAP101.com. He has more than 14 years of experience in ABAP language and was ABAP instructor for more than 5 years. Today he's working for Nestlé Purina USA as Technical Architect and live with his wife and three kids in Saint Louis, MO, USA.

Jaime Freitas is graduate on Information Systems by Faculdades Integradas Rio Branco. He has been building a career on Information Technology since he was 17 years-old adquiring experience on managements, analysis, development, implementing and supporting software solutions. Recently, he worked as a Project Manager at Walmart Brazil and nowadays he is a senior system analysis at Lojas Riachuelo, a Brazilian department store with over than 40 thousand employees.