



Subroutines (sub-programs)





What is a sub-program?

Sub-programs are named blocks of code which can be run from within another part of the program.

When a sub-program is used like this we say it is “called”.

Sub-programs can be called from any part of the program and can be used over again.

A sub-program may be called several times during the execution of a single program





Why create sub-programs

Creating sub-programs makes the code more **modular** and readable.

Modular code allows sections of code to be self-contained.

Different sub-programs can be developed by different programmers without variable name clashes

Sub-programs can be re-used without any extra coding which saves time.



Example

This program works out the area of a room in a building.

Main_Program

```
SET length TO 0
SET breadth TO 0
SET area TO 0

REPEAT
    RECEIVE length FROM KEYBOARD
    IF length < 5 OR length > 50 THEN
        SEND "Please Re-enter" TO DISPLAY
    END IF
UNTIL length >=5 AND length <=50

REPEAT
    RECEIVE breadth FROM KEYBOARD
    IF breadth < 0 OR breadth > 20 THEN
        SEND "Please Re-enter" TO DISPLAY
    END IF
UNTIL breadth >=0 AND breadth <=20

SET area TO length * breadth

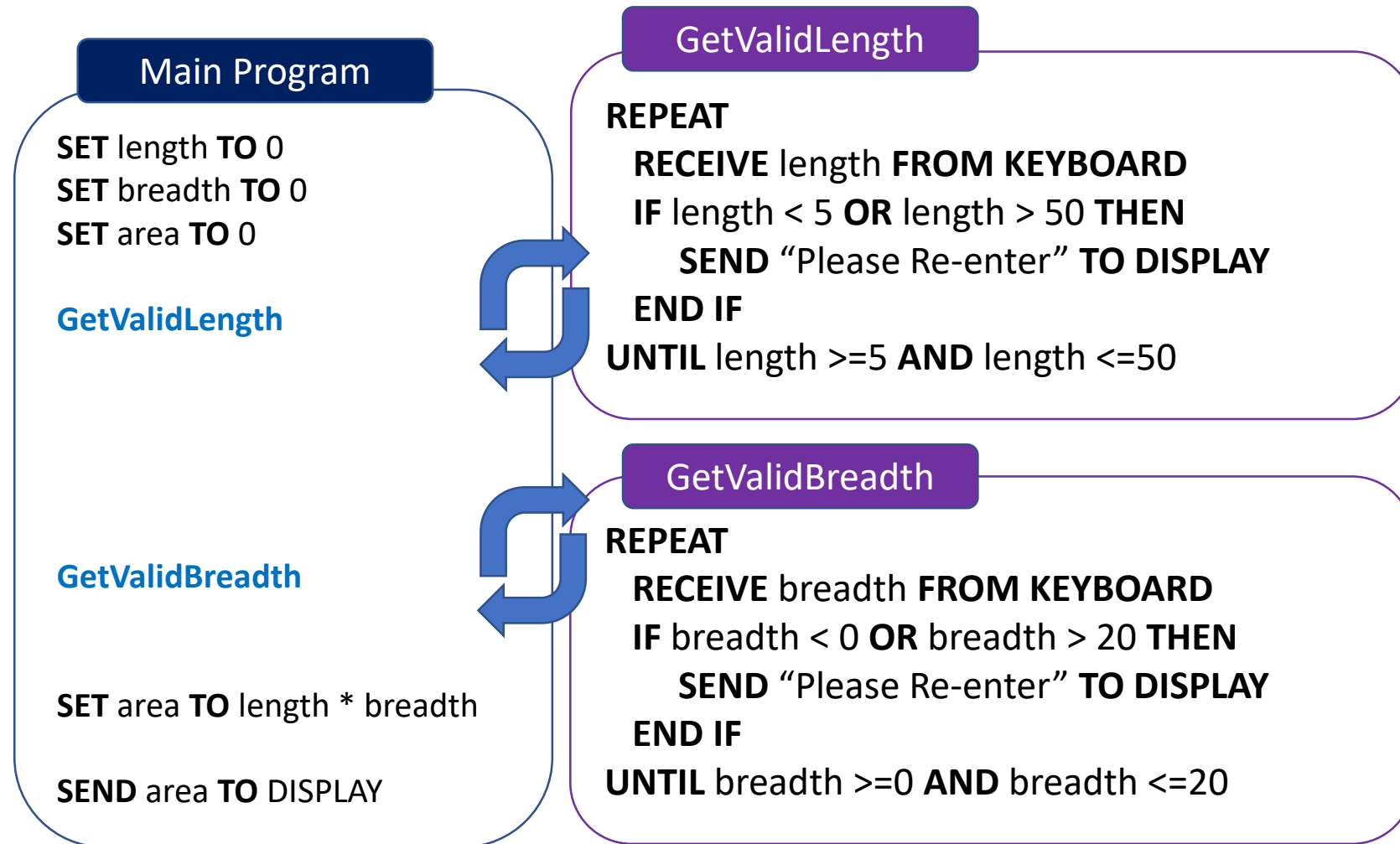
SEND area TO DISPLAY
```



Example



The Input Validation lines of code can be put into **sub-programs** and **called** by the main program





Types of sub-program

There are two types of sub-program that can be used in procedural languages.

- **Procedures**
- **Functions**



Procedures

A procedure is a self-contained section of code that executes a set of commands.

Procedures are given a meaningful name which is used to call them

When procedures are called, variables (parameters) can be passed into the procedure and back out again.

Procedures can be called with any number of parameters passing in or out (or sometimes none)



Example



Consider creating the GetValidLength sub-program as a **procedure**

Main Program

SET *UserLen* TO 0

CALL **GetValidLength** (*UserLen*)

The *UserLen* variable is declared in the main program

When the procedure is called, *UserLen* is **passed** to it as a **parameter**



GetValidLength

When it is called, the **GetValidLength** procedure executes its lines of code in order.

The *length* parameter is changed and **passed back** to the main program.

PROCEDURE GetValidLength (*length*)

REPEAT

RECEIVE *length* FROM KEYBOARD

IF *length* < 5 OR *length* > 50 THEN

SEND "Please Re-enter" TO DISPLAY

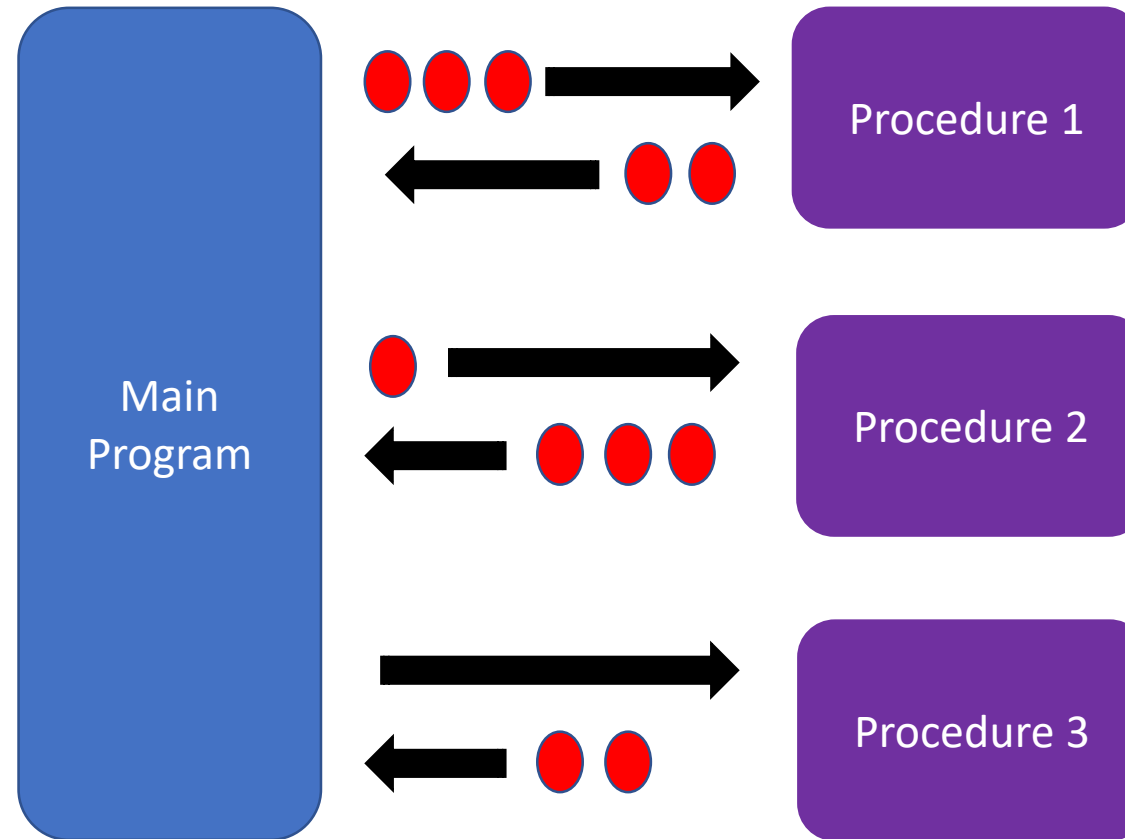
END IF

UNTIL *length* >=5 AND *length* <=50

END PROCEDURE



Procedures



Any number of parameters (variables) can be passed **in or out** of procedures.



Functions

A function is also a self-contained section of code that executes a set of commands.

Functions are also given a meaningful name which describes its purpose (usually a verb)

When functions are called, variables (parameters) the function needs can be passed into it.

Functions **return** a **single value** which is stored in a variable



Example



Consider creating the GetValidLength sub-program as a **function**

MainProgram

SET *UserLen* TO 0

UserLen = GetValidLength()

The *UserLen* variable is declared in the main program

When the function is called, *UserLen* is used to store the **returned integer**

When it is called, the **GetValidLength** function executes its lines of code in order.

The *length* variable is **returned** to the main program and stored in *UserLen*

GetValidLength

FUNCTION GetValidLength () **RETURNS INTEGER**

REPEAT

RECEIVE *length* FROM KEYBOARD

IF length < 5 OR *length* > 50 THEN

SEND "Please Re-enter" TO DISPLAY

END IF

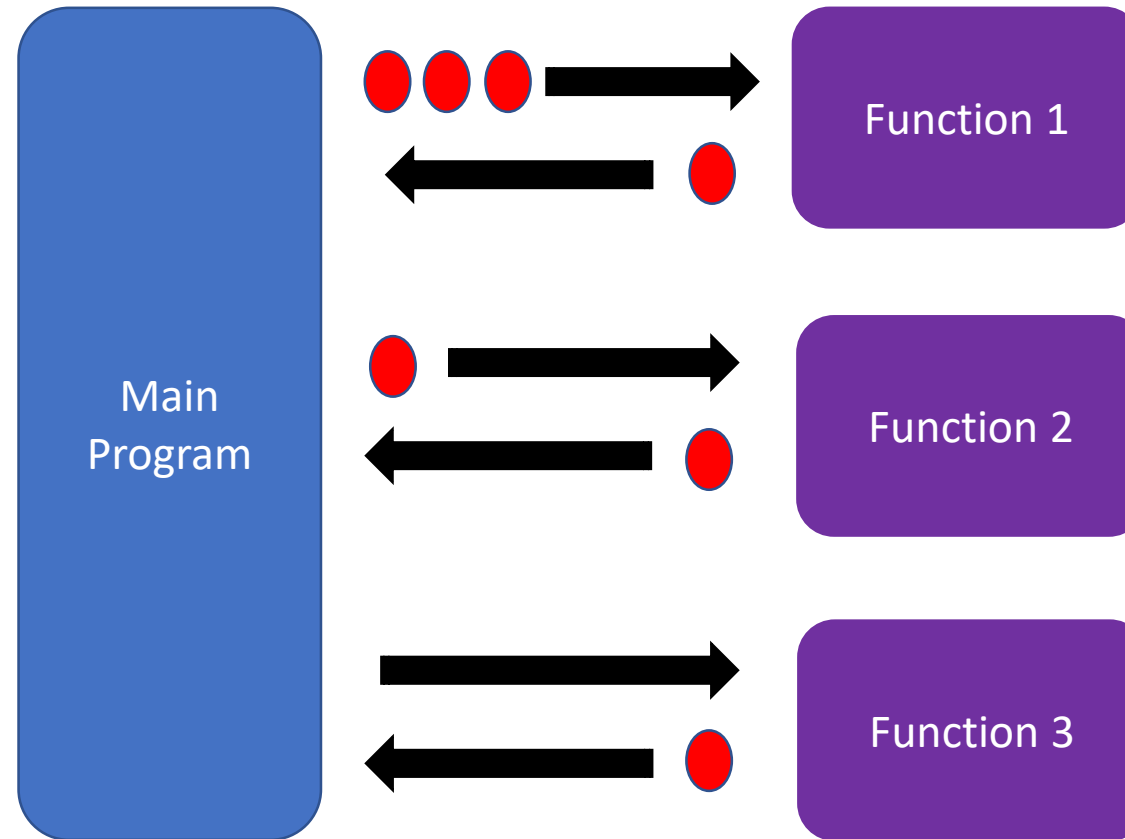
UNTIL *length* >=5 AND *length* <=50

RETURN *length*

END FUNCTION



Functions



Any number of parameters (variables) can be passed in or but only **one value** is returned.



Re-using sub-programs

The most efficient use of sub-programs is when they can be re-used .

When coding procedures and functions, consideration should be given to making them able to solve **any related problem** rather than **one specific problem**.

e.g. a calculator that can only solve the calculation $2+2$ would be very limited.



Re-using procedures

The procedures below are almost identical except for the range of values they validate.

GetValidLength

```
PROCEDURE GetValidLength (length)  
REPEAT  
  RECEIVE length FROM KEYBOARD  
  IF length < 5 OR length > 50 THEN  
    SEND "Please Re-enter" TO DISPLAY  
  END IF  
UNTIL length >= 5 AND length <= 50  
END PROCEDURE
```

GetValidBreadth

```
PROCEDURE GetValidBreadth (breadth)  
REPEAT  
  RECEIVE breadth FROM KEYBOARD  
  IF breadth < 0 OR breadth > 20 THEN  
    SEND "Please Re-enter" TO DISPLAY  
  END IF  
UNTIL breadth >= 0 AND breadth <= 20  
END PROCEDURE
```



Re-using procedures

They could instead be made more generic by allowing the range of values to be changed each time it is called.

GetValidValue

```
PROCEDURE GetValidValue (low, high, userVal)  
REPEAT  
  RECEIVE userVal FROM KEYBOARD  
  IF userVal < low OR userVal > high THEN  
    SEND "Please Re-enter" TO DISPLAY  
  END IF  
UNTIL userVal >=low AND userVal <=high  
END PROCEDURE
```



Re-using procedures

Main Program

```
SET UserLen TO 0  
SET UserBre TO 0  
  
CALL GetValidValue (5, 50, UserLen)  
  
CALL GetValidValue (0, 20, UserBre)
```

The GetValidValue procedure can now be called to obtain a value within **any range** specified.

GetValidValue

```
PROCEDURE GetValidValue (low, high, userVal)  
REPEAT  
  RECEIVE userVal FROM KEYBOARD  
  IF userVal < low OR userVal > high THEN  
    SEND "Please Re-enter" TO DISPLAY  
  END IF  
UNTIL userVal >=low AND userVal <=high  
END PROCEDURE
```



Re-using functions

Main Program

```
SET UserLen TO 0  
SET UserBre TO 0  
  
UserLen = GetValidValue (5, 50)  
  
UserBre = GetValidValue (0, 20)
```

The implementation of a reusable function would look like this.

GetValidValue

```
FUNCTION GetValidLength (low, high) RETURNS INTEGER  
REPEAT  
  RECEIVE userVal FROM KEYBOARD  
  IF userVal < low OR userVal > high THEN  
    SEND "Please Re-enter" TO DISPLAY  
  END IF  
UNTIL userVal >= low AND userVal <= high  
RETURN userVal  
END FUNCTION
```