A·P·U
ASIA PACIFIC UNIVERSITY
OF TECHNOLOGY & INNOVATION

INTRODUCTION TO VISUAL AND INTERACTIVE PROGRAMMING

CT803-4-0-OIVIP

Topic: Procedures and Functions

Topic Learning Outcomes

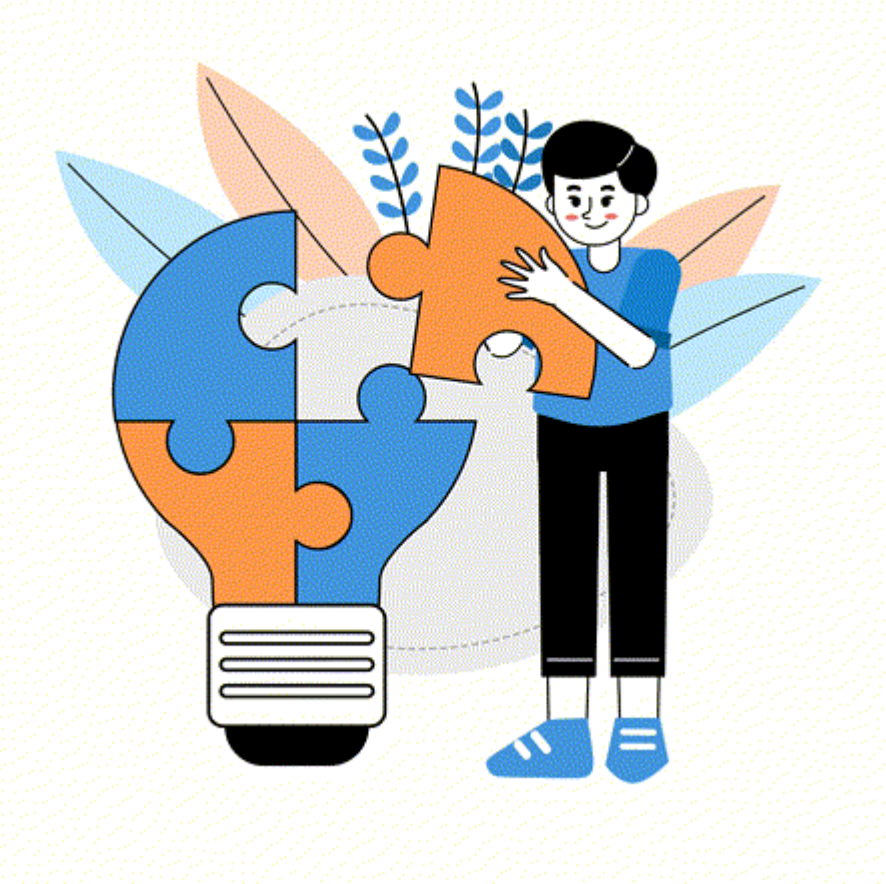
At the end of this topic, you should be able to:

- Explain procedures and function
- Identify the use of message broadcasting and make a block as procedure in block programming.
- Define the purpose of procedure in large program.
- Implement procedures in block programming

Contents & Structure

- Procedures
- Variables and Argument
- Messaging, Broadcasting and Receiving
- Create large program in small steps
- Working with procedures

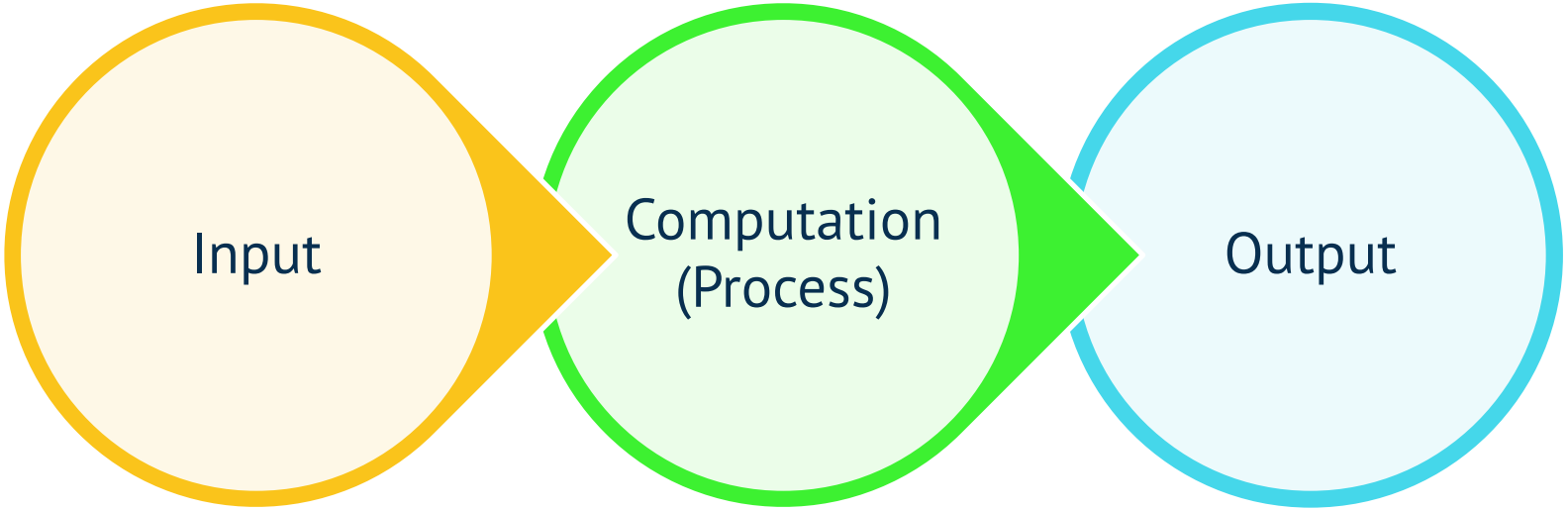
Introduction



HOW DO YOU SOLVE A COMPLEX PROBLEM?

Solve a Complex Problem

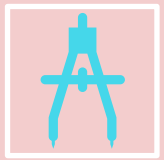
- In general, a problem is solved in 3 steps:



Introduction to Modular Programming

When a program becomes very large and complex, it becomes very difficult task for the programmers to design, test and debug such a program.

Therefore, a long program can be divided into a smaller program called modules called modules as the modules can be designed, tested and debugged separately, the task of programmer becomes easy and convenient.



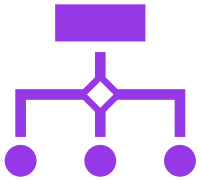
It makes your program easy to understand.

Why Modular Programming

Helps manage complexity

Smaller blocks of code

Easier to read



Encourages re-use of code

Within a particular program or across different programs

Allows independent development of code

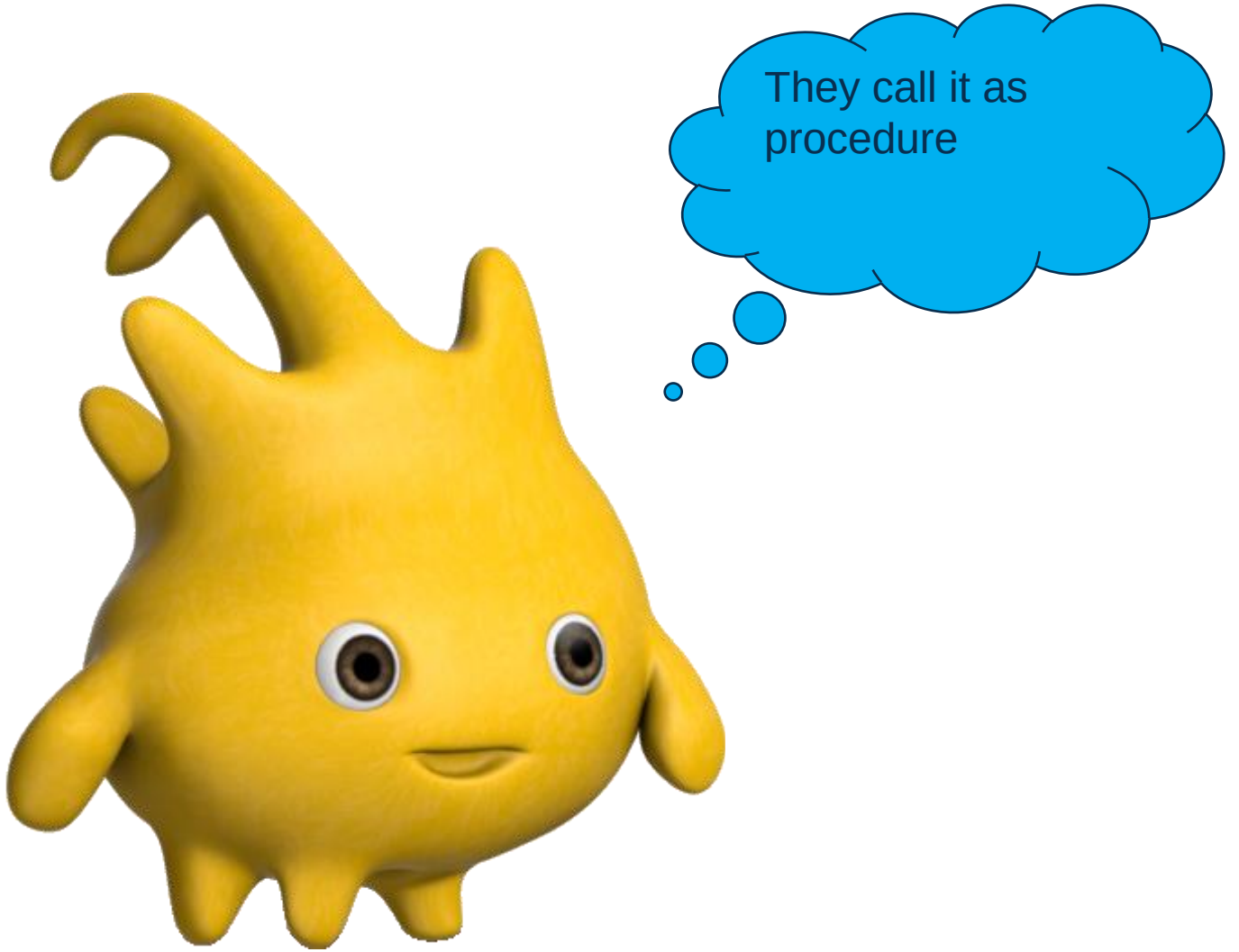
Advantages of Modular Programming

- Can be written and tested separately
- Can be reused
- Large projects can be developed in parallel
- Reduces length of program, making it more readable
- Promotes the concept of abstraction

Creating Large Programs in Small Steps

- The scripts that you've written up to this point are relatively short and simple.
- Eventually, you'll write longer, more complex scripts that contain hundreds of blocks, and understanding and maintaining them will become a real challenge.
- An approach known as *structured programming* was developed in the mid-1960s to simplify the process of writing, understanding, and maintaining computer programs.
- Instead of having you write a single large program, this approach calls for dividing the program into smaller pieces, each of which solves one part of the overall task.

What we have in Visual(graphical) Programming?



Procedure and Function in Snap!

Modules in Snap! are called procedure / function.

A procedure is a module performing one or more actions; it does not need to return any values.

A procedure may have 0 to many parameters.

Every procedure has two parts:

- The header and keyword (this contains the procedure name and the parameter list)
- The body, which is everything after the procedure or parameter list

Functions are a type of stored code and are very similar to procedures.

Functions can accept one, many, or no parameters, but a function must have a **report (return)** clause in the executable section of the function.

Procedures and Functions in Snap!

Functions and procedures are small sections of code used to perform a specific task.

They are used for two reasons:

They can be used to **avoid repetition** of commands within the program

Careful use of functions and procedures helps to define a **logical structure** for your program by breaking it down into a few smaller **modules**.

Function in Snap!

Is a series of statements enclosed by the certain module (of that system) and End Function statements.

The Function procedure performs a task and then returns control to the calling code.

When it returns control, it also may return a value to the calling module

Function that *report* (return) a value are called *reporters*.

Elements of Functions

Function definition:

Will defines how the function will perform the task

Function call:

Will call the function with or without arguments.

Type of Functions

Pre-defined (built-in) functions:

- Functions that pre-defined in the library (round(), sqrt(), pick random(), length()

Programmer-defined functions:

- Which is define by the user.

Procedures

In Block Programming

- Procedures that do something are called **commands**.
  - When making a block, you choose the name of the block, the input parameters, and the palette it should appear in (the color).
  - Then you will design the instructions that will run when the block is clicked.

Figure 7.1

- Any sprite can broadcast a message (you can call this message anything you like) using the **broadcast** or **broadcast and wait** blocks (from the *Control* palette) shown in Figure 7.1.

- This broadcast triggers all scripts in all sprites (including the broadcasting sprite itself) that begin with a matching **when I receive** trigger block.

- All sprites hear the broadcast, but they'll only act on it if they have a corresponding **when I receive** block.

Message Broadcasting and Receiving

Message, Broadcasting and Receiving

Hold a Conversation

- Have sprites chat with one another in an animated scene or story.
- Broadcasting can prompt a character to answer a question.
- Or cause a character to respond to something that was said.

Respond to Events

- Use broadcasting to make a sprite react to an event.
- For example, a character may move or change appearance when something happens.

Produce Multiple Actions at the Same Time

- Broadcasting can send a message to many sprites.
- This can cause several characters to do something at the same time.
- This enhances storytelling and holds viewer interest.

Control Game Play

- Direct when a game begins using broadcasting.
- After the instructions appear on the screen a broadcasted message can launch the game.
- Use it to make targets appear or start a timer.

End a Game

- Set what happens when a game is over.
- Use broadcasting to inform a player that the game has ended.
- For instance, you could display a message, such as GAME OVER.
- Broadcasting can also be used to stop game play.
- For example, you could hide targets to prevent the player from scoring more points.

Organize Scripts

- Long scripts in BYOB / SNAP! cannot display on one screen.
- This makes them difficult to debug.
- A solution is to divide the script into smaller chunks using broadcasting

What can you do with broadcasting?

Plan to Broadcast a Message in your Application

When using broadcasting it is a good idea to PLAN AHEAD:

01

Decide what you want to happen

02

Once you have an idea, pick the sprite that will send the message. Who is in control of when an action happens?

03

Study the sprite's script. To send the message at the right time, where should the broadcast coding block be placed?

04

Next, pick the sprite or sprites that will receive the message. What will they do when they receive the message?

Before Scratch 2, you couldn't build the ***Initialize*** block shown and then call it from your script.
The only way to emulate procedures and add some structure to a program was through BYOB message-broadcasting mechanism.
This has changed in Scratch 2 and BYOB, which added the powerful of ***"make a block"*** feature.

At this point, you might ask, *"How do we create these function?"*

BUILDING YOUR OWN BLOCK

IMPORTANT
You Must Use
the Category
Other

If you leave the Category as Other your custom block will be grey color and found at the bottom of the Variables category

We will Only Look Here for custom blocks

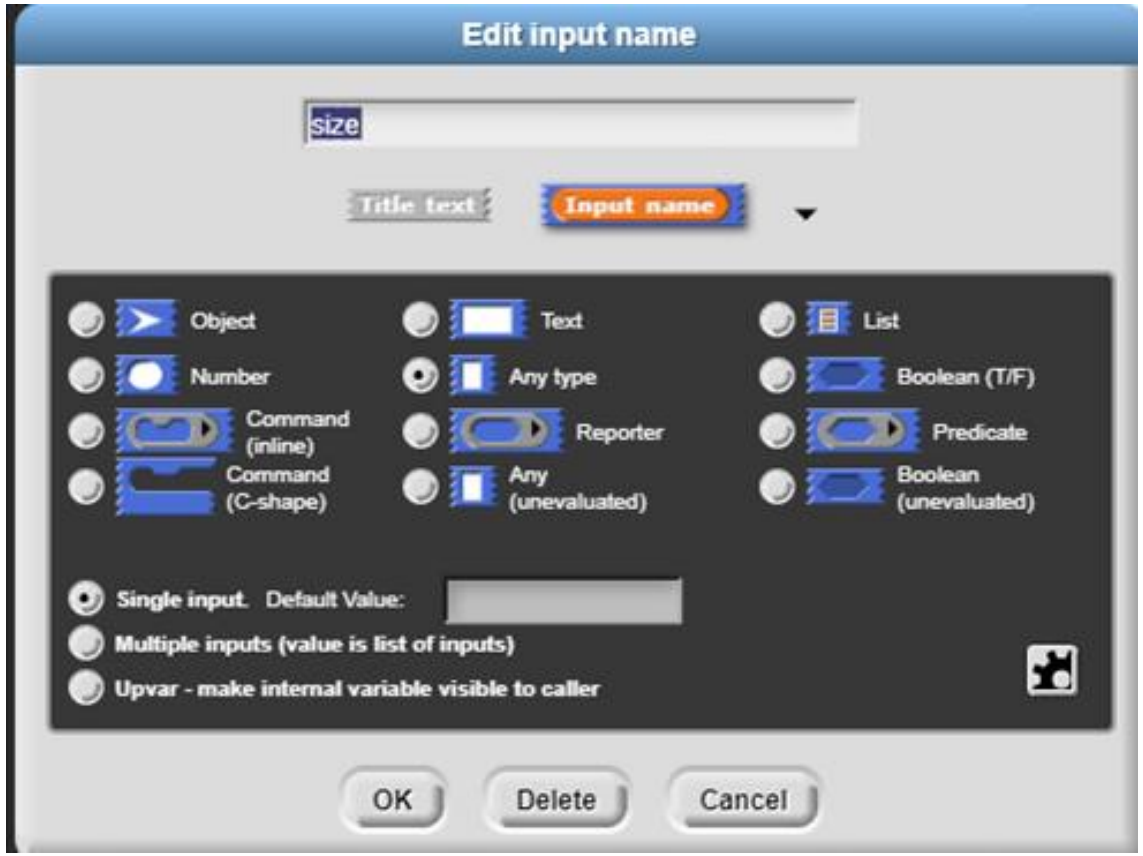
BUILDING YOUR OWN BLOCK

You **May** choose a **Category** so the block appears in that section

You **Must** choose a **Type** so the block has the right shape

You **Must** choose the **Scope**

Function – Input Type Dialog



- There are twelve input type shapes, plus three mutually exclusive categories, listed in addition to the basic
- Choose between title text and an input name.
- Default type - “Any,” meaning that this input slot is meant to accept any value of any type.
- If the size input in your block should be an oval-shaped numeric slot rather than a generic rectangle, click “Number.

Parameters

- Parameters are the means to pass values to and from the calling environment to the server.
- These are the values that will be processed or returned via the execution of the procedure/functions

Variables vs Argument

Variables

- Variables declared within functions or procedures are said to be local.
- Can only be used within that function, or other functions called by that function.
- This is called the scope of the variable. See variable slides.

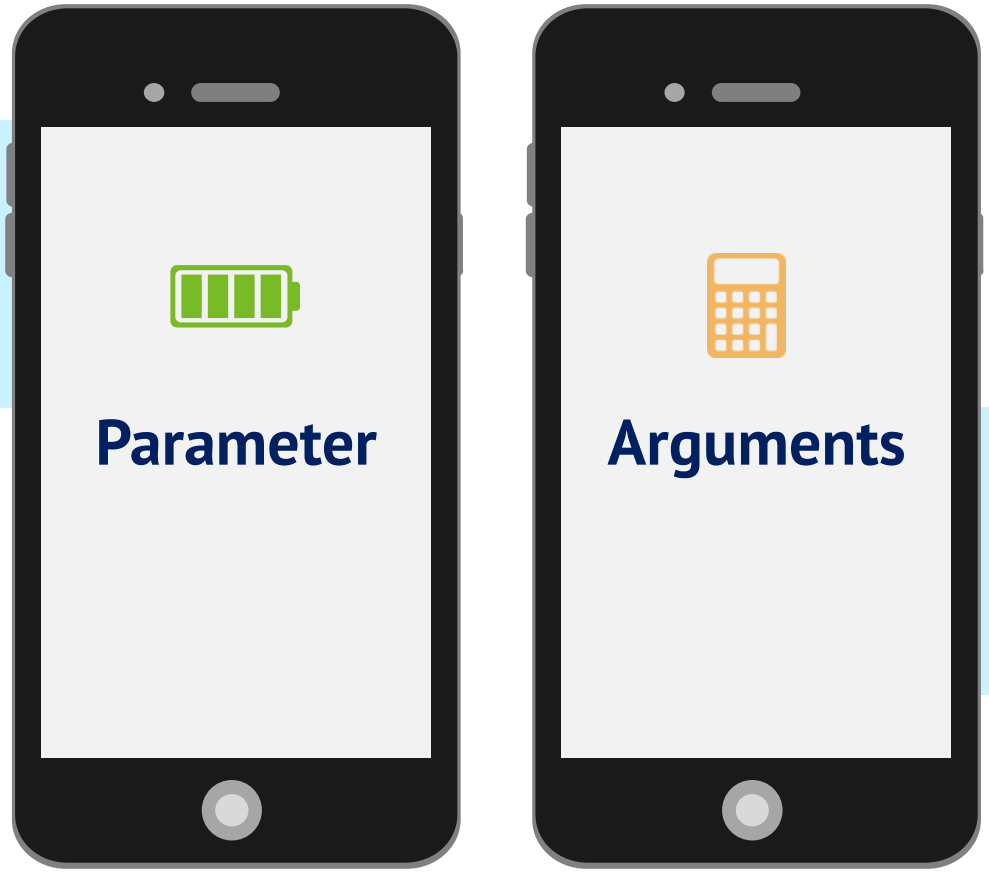
Argument

- Some functions will require arguments - values upon which the operation performed by the function will be based.
- If you are using a typed language, then you will need to give the argument a type in your function/procedure declaration.

The difference is subtle, but not difficult:

Defined by the programmer when the method is defined, and given names which do not change

As a general rule, if we are talking about ***using*** a method we are talking about **arguments**, and if we are talking about a method ***definition*** we are talking about **parameters**.



Running program can provide different values for the arguments each time the method is called

For example, in a **max of (x) and (y)** method, x and y are **parameters**, but when it is used you have something like **max of (9) and (3)**, where 9 and 3 are **arguments**

Function with parameters

- In Snap!, a function that report (return) a value are called reporters.
- Reporter blocks have a rounded shape, and they can either be clicked to report a value to the programmer or they can be dropped into an empty input slot of another block to be used as input.
- When you make a reporter block, the block definition automatically includes a report block.
- The value of the expression in the input slot of this report block is the value that is returned when the reporter is run.
- Here is an example of what a real reporter block definition might look like in Snap!:

Summary / Recap of Main Points

- Procedures
- Variables and Argument
- Messaging, Broadcasting and Receiving
- Create large program in small steps