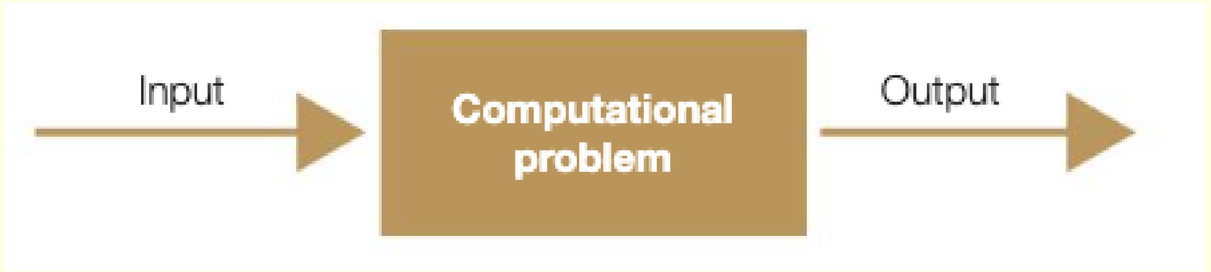
Learning Aims

Thinking Ahead

- Identify the inputs and outputs for a given situation
- Determine the preconditions for devising a solution to a problem
- Understand the need for reusable program components
- Understand the nature, benefits and drawbacks of caching

Computational problems

At its most abstract level, a computational problem can be represented by a simple diagram:



Input is the information relevant to the problem, which could for example be passed as parameters to a subroutine.

Output is the solution to the problem, which could be passed back from a subroutine.

A clear statement of exactly what the inputs and outputs of a problem are is a necessary first step in constructing a solution.

Example 1: Determine whether a given item is present in a list

On the face of it, this is a simple problem.

But do we know exactly what the inputs are? For example:

Is the list sorted? Are the items numeric or alphabetic?

What about the output – are we expecting it to be simply True or False, or should the output give the position in the list of the item if it is found?

The problem needs to be formally defined, stating the inputs and outputs. This can be done as follows:

Name:	SearchList
Inputs:	A list of strings S = (s1, s2,s3,sn)
	A target string t
Outputs:	A Boolean variable b

Now we can write pseudocode for the function SearchList:

```
function SearchList(s, t)
    found = False
    n = 0
    while found == False AND n < len(s)
        if t == s[n] then
            found = True
        else
            n = n + 1
        endif
    endwhile
    return found
endfunction
```

Write a pseudocode algorithm which initialises a list of string items, asks the user to enter an item to search for, calls the above function SearchList and prints an appropriate message depending on whether the function returns True or False.

```
aList = [‘banana, ‘pear’, ‘blackberry’, ‘rhubarb’]
searchItem = input(“Please enter item: “)
inList = SearchList(aList)
if (inlist == True)
  print(“Item in list”)
else
  print(“item not in list”
endif
```

Question

Using the headings given, specify the inputs and outputs to the following problems:

Sort a list of names into alphabetical order and return the sorted list, leaving the original list unchanged.

Name: sortList

Inputs: A list of strings listString **[s_1, s_2, s_3, … s_n]**

Outputs: A sequenced list of strings **[t_1, t_2, t_3, … t_n]**

Question

Find the average of a list of marks:

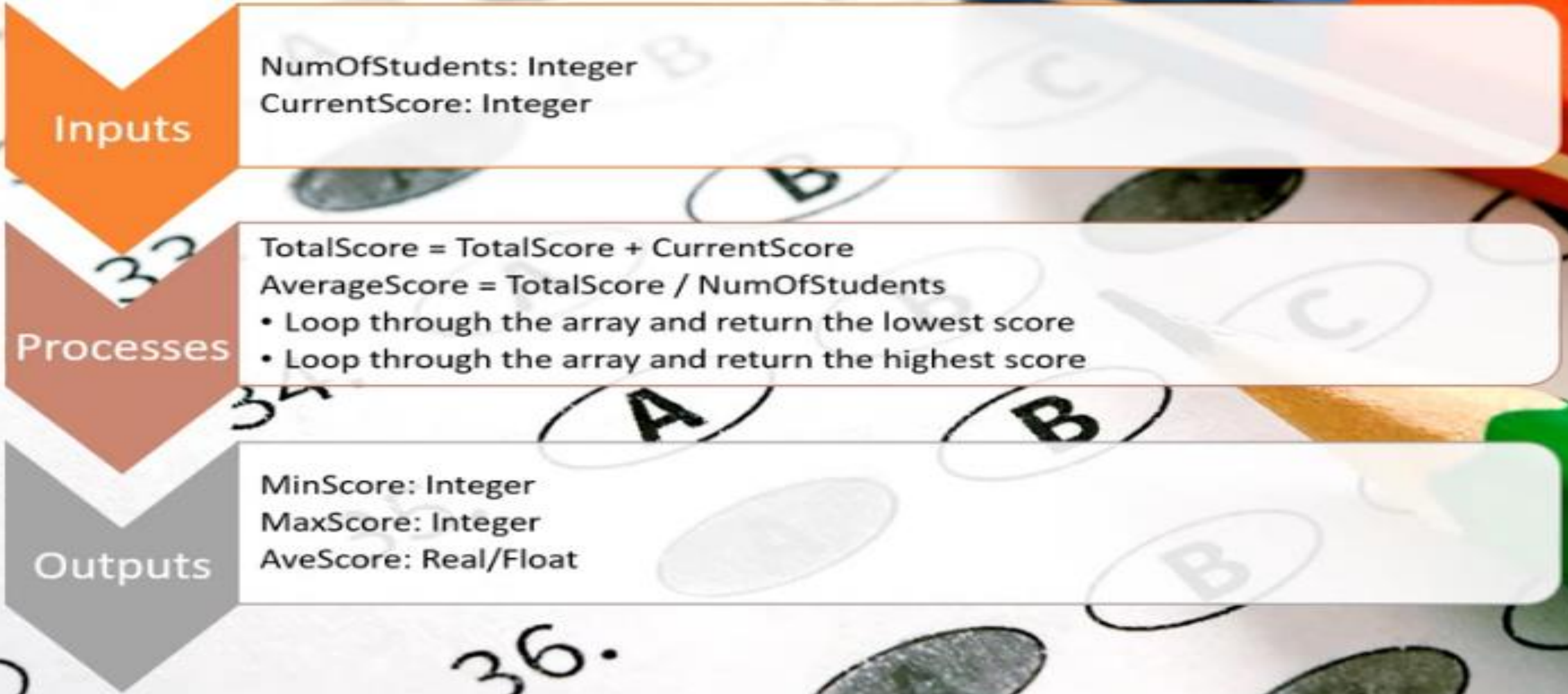
Name: averageNum

Inputs: A list of real numbers **listReals [r_1, r_2, r_3, … r_n]**

Outputs: A real number

Identifying inputs, processes and outputs

"Write a program that asks the user for the number of students in their class and prompts them to enter each student's test score within the range of 0 – 100. It should then output the highest, lowest and average score."



Inputs	NumOfStudents: Integer CurrentScore: Integer

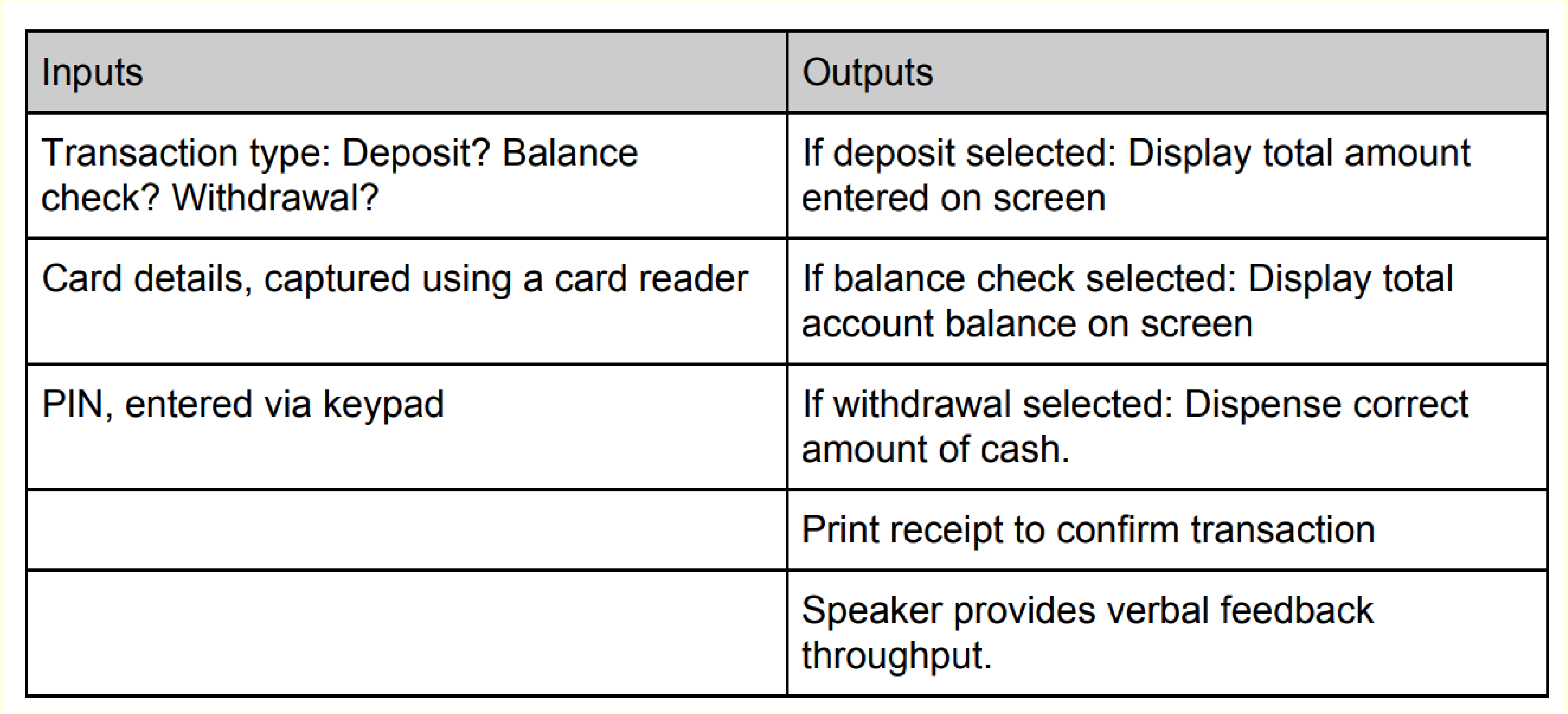
Processes	TotalScore = TotalScore + CurrentScore
	AverageScore = TotalScore / NumOfStudents
	• Loop through the array and return the lowest score
	• Loop through the array and return the highest score

Outputs	MinScore: Integer MaxScore: Integer AveScore: Real/Float

Example: Identifying the inputs & outputs

You must be able to identify the inputs and outputs that would be required to form a program given a scenario.

Take, for example, a program designed for an ATM.



Inputs	Outputs
Transaction type: Deposit? Balance check? Withdrawal?	If deposit selected: Display total amount entered on screen
Card details, captured using a card reader	If balance check selected: Display total account balance on screen
PIN, entered via keypad	If withdrawal selected: Dispense correct amount of cash.
	Print receipt to confirm transaction
	Speaker provides verbal feedback throughput.

Specifying preconditions

Suppose that a pseudocode algorithm has been written to find the maximum of a list of numbers.

```
function maxInt(listInt)
    maxNumber = listInt[0]
    for i = 1 to len(listInt) - 1
        if listInt[i] > maxNumber then
            maxNumber = listInt[i]
        endif
    next i
    return maxNumber
endfunction
```

In order to make sure the function never crashes, either the function must test for an empty list, or a precondition must be specified with the documentation for the function.

Name: maxInt

Inputs: A list of integers listInt = $(k_1, K_2, K_3 \ldots K_n)$

Outputs: An integer maxInt

Precondition: **length of listInt > 0**

If the function is called with an empty list, it will crash on the statement

maxNumber = listInt[0]

Specifying preconditions

Specify the input, output and any preconditions for a function sqrt(n) which finds the square root of an integer or floating point number.

Name: sqrt

Input: integer or floating point number *n*

Output: square root of n

Precondition: n >= 0

Advantages of specifying preconditions

- Specifying preconditions as part of the documentation of a subroutine ensures that the user knows what checks, if any, must be carried out before calling the subroutine.
- If there are no preconditions, then the user can be confident that necessary checks will be carried out in the subroutine itself, thus saving unnecessary coding. The shorter the program, the easier it will be to debug and maintain.
- Clear documentation of inputs, outputs and preconditions helps to make the subroutine reusable. This means that it can be put into a library of subroutines and called from any program with access to that library.

How Programmers Use Reusable Components in Large Programs

Programmers use **reusable components** to save time, reduce errors, and make development more efficient. These components include **functions, modules, libraries, and classes** that can be used multiple times in a program or across different projects.

Libraries & Modules

- Pre-built libraries (e.g., Python's math or random) are imported instead of rewriting common functions.
- This improves efficiency and reliability.

Functions & Subroutines

- Frequently used code (e.g., file handling, validation) is written as functions.
- This reduces repetition and makes debugging easier.

Modular Programming

- Large programs are split into **small, reusable modules** that can be used in different projects.
- Makes development more manageable.

Object-Oriented Programming (OOP)

- Classes are used to create reusable objects with properties and methods.
- Example: A Car class with attributes (speed, color) and methods (accelerate(), brake()).

Version Control & Code Repositories

- Tools like **GitHub** store reusable code, making it easy for teams to share and reuse components.

Reusability

Benefits	**Drawbacks**
Benefits: **Saves Time** – No need to rewrite common code. **Easier Maintenance** – Updates apply across all uses of the component. **Fewer Errors** – Well-tested components reduce bugs. **Better Collaboration** – Teams can work on separate modules and combine them. Using reusable components makes software **faster to develop, more reliable, and easier to maintain**.	Third-party components may not always be compatible with existing software. They may need modification, which can be **costly** and **time-consuming**, sometimes making in-house development a better option.

Nature and benefits of caching

Caching **temporarily stores** program instructions or data that were recently used and may be needed again soon.

For example:

The last few instructions of a program may be kept in **cache memory** for **quick access.**

Web caching is another example, where recently viewed **HTML pages and images** are stored. This allows **faster access** to previously visited pages and reduces unnecessary bandwidth usage by avoiding repeated downloads.

Benefits and Drawbacks of caching

Benefits	**Drawbacks**
Caching allows for **faster response times** and overall better device performance.	**Caching** depends on how well a caching algorithm is able to manage the cache. Larger caches still take a long time to search and so cache size limits how much data can be stored
Prefetching can be difficult to implement but can significantly improve performance if implemented effectively.	**Prefetching** - The biggest limitation is the **accuracy** of the algorithms used in prefetching, as they can only provide an informed **prediction** as to the instructions which are likely to be used and there is no guarantee that this will be right.

Worked Example

A user working on a PC at home has done several searches on the Internet and leaves the browser windows opens when she shuts down. The next day, her mother needs to use the PC. On opening the browser, all the windows that were previously open, appear automatically.

How does this happen? What are the advantages? What are the drawbacks? How can this situation be prevented?

The pages are cached by the operating system and reloaded when the browser is loaded again.

Advantage

Often very convenient for the user, who can continue work from where they left off, without having to search for pages again

Disadvantage

Possibly the user who loaded the pages may not want another user to see the pages they were looking at.

The situation can be avoided if each user has their own login password.