Pseudocode

Representing an Algorithm: Pseudocode

* **What is Pseudocode?**

There are two main ways to represent an algorithm: Pseudocode and flowchart. Pseudocode is an informal high-level description of a computer program or algorithm. It is written in symbolic code which must be translated into a programming language before it can be executed. Using Pseudocode is similar to “writing in a programming language” and might look something like this:

OUTPUT 'What is your name?'

INPUT user inputs his/her name

STORE the user's input in the name variable

OUTPUT 'Hello' + name

OUTPUT 'How old are you?'

INPUT user inputs his/her age

STORE the user's input in the age variable

IF age >= 70 THEN

OUTPUT 'You are aged to perfection!'

ELSE

OUTPUT 'You are a spring chicken'

***INPUT*** asks a question. ***OUTPUT*** prints a message on the screen.

Pseudocode makes creating programs easier. Programs can be complex and long; preparation is the key. For years, flowcharts were used to map out programs before writing one line of code in a language. However, they were difficult to modify and with the advancement of programming languages, it was difficult to display all parts of a program with a flowchart. It is challenging to find a mistake without understanding the complete flow of a program. That is where pseudocode becomes more appealing.

To use pseudocode, all you do is write what you want your program to say in English. Pseudocode allows you to translate your statements into any language because there are no special commands and it is not standardized. Writing out programs before you code can enable you to better organize and see where you may have left out needed parts in your programs. All you have to do is write it out in your own words in short statements. Let's look at some examples.

Begin

INPUT hours

INPUT rate

pay = hours \* rate

OUTPUT pay

End

And this, slightly more complex example might calculate pay with overtime:

Begin

INPUT hours, rate

IF hours < 40

THEN

pay = hours \* rate

ELSE

pay = 40 \* rate + (hours - 40) \* rate \*1.5

OUTPUT pay

End

Most programs are developed using programming languages. These languages have a specific syntax that must be used so that the program will run properly. Pseudocode is not a programming language, it is a simple way of describing a set of instructions that does not have to use specific syntax.

Writing in a pseudocode is similar to writing in a programming language. Each step of the algorithm is written on a line of its own in sequence. Usually, instructions are written in uppercase, variables in lowercase and messages in sentence case.

* **Why use Pseudocode?**

A prototype is an early sample, model or release of a product created with the intention of concept testing and for learning purposes. They help us to learn without fully implementing our solutions. When developing user interfaces for our applications, we have several prototypes before the final interface. Some examples of these are wire-frames, graphical designs, and mock-ups. The same applies to writing technical code. Directly writing code for complex purposes might result in time wastage. The causes of this range from improper algorithms to ambiguous program flow. To prevent this, we can use Pseudocode.

Advantages of pseudocode:

* Pseudocode is understood by the programmers of all types.
* It enables the programmer to concentrate only on the algorithm part of the code development.
* It cannot be compiled into an executable program.

How to Write Pseudocode - 1

**Statements**

A statement is defined as an instruction that directs the computer to perform a specific action. In writing pseudocode, we will refer to singular instructions as statements. When writing pseudocode, we assume that the order of execution of the statements is from top to bottom. This changes when using control structures, functions and exception handling.

* **Mathematical operations** Mathematical operations are integral to solution development. They allow us to manipulate the values we have stored. Here are common mathematical symbols:

Assignment: ← or :=

Example: c ← 3, c := 2

Comparison: =, ≠, <, >, ≤, ≥

Arithmetic: +, −, ×, /, mod

Logical: and, or

* **Keywords** A keyword is a word that is reserved by a program because the word has a special meaning. Keywords can be commands or parameters. Every programming language has its own keywords (reserved words). Keywords cannot be used as variable names. In Pseudocode, they are used to indicate common input-output and processing operations. They are written fully in uppercase.

START, BEGIN: This is the start of your pseudocode.

INPUT: This is data retrieved from the user through the input device.

READ, GET: This is used when reading data from a data file.

PRINT, DISPLAY, SHOW, OUTPUT: This will show your output to a screen.

COMPUTE, CALCULATE: To calculate the result of the expression.

SET, INIT: To initialize values

INCREMENT, BUMP: To increase the value of a variable

DECREMENT: To reduce the value of a variable

END: This is the end of your pseudocode

**Conditionals**

During algorithm development, we need statements that evaluate expressions and execute instructions depending on whether the expression evaluated to True or False. Here are some common conditions used in Pseudocode:

* **IF — ELSE IF — ELSE**

This is a conditional that is used to provide statements to be executed if a certain condition is met. This also applies to multiple conditions and different variables. Here is an if statement with one condition

IF you are happy

THEN smile

ENDIF

Here is an if statement with an else section. Else allows for some statements to be executed if the “if” condition is not met.

IF you are happy

THEN smile

ELSE

frown

ENDIF

We can add additional conditions to execute different statements if met.

IF you are happy

THEN smile

ELSE IF you are sad

THEN frown

ELSE

keep face plain

ENDIF

* **CASE**

Case structures are used if we want to compare a single variable against several conditions.

INPUT color

CASE color of

red: PRINT "red"

green: PRINT "green"

blue: PRINT "blue"

OTHERS

PRINT "Please enter a value color"

ENDCASE

The OTHERS clause with its statement is optional. Conditions are normally numbers or characters.

**Iteration**

To iterate is to repeat a set of instructions in order to generate a sequence of outcomes. We iterate so that we can achieve a certain goal.

* **FOR structure**

The FOR loop takes a group of elements and runs the code within the loop for each element.

FOR every month in a year

Compute number of days

ENDFOR

* **WHILE structure**

Similar to the FOR loop, the while loop is a way to repeat a block of code as long as a predefined condition remains true. Unlike the FOR loop, the while loop evaluates based on how long the condition will remain true.

To avoid a scenario where our while loop runs infinitely, we add an operation to manipulate the value within each iteration. This can be through an increment, decrement, et cetera.

PRECONDITION: variable X is equal to 1

WHILE Population < Limit

Compute Population as Population + Births — Deaths

ENDWHILE

**Functions**

When solving advanced tasks it is necessary to break down the concepts in a block of statements in different locations. This is especially true when the statements in question serve a particular purpose. To reuse this code, we create functions. We can then call these functions every-time we need them to run.

Function clear monitor

Pass In: nothing

Direct the operating system to clear the monitor

Pass Out: nothing

Endfunction

To emulate a function call in pseudocode, we can use the Call keyword

call: clear monitor

**Program Wrapping**

After writing several functions in our pseudocode, we find the need to wrap everything into one container. This is to improve readability and make the execution flow easier to understand.

To do this, we wrap our code as a program. A program can be defined as a set of instructions that performs a specific task when executed.

PROGRAM makeacupoftea

END

**Exception Handling**

An exception is an event which occurs during program execution that disrupts the normal flow of the instructions. These are events that are non-desirable.

We need to observe such events and execute code-blocks in response to them. This is called exception handling.

BEGIN

statements

EXCEPTION

WHEN exception type

statements to handle exception

WHEN another exception type

statements to handle exception

END

**Conclusion**

There are no technical rules for Pseudocode. It is meant to be human-readable and still convey meaning and flow.

There are different guide and tutorials which lean more towards language-specific pseudocode, examples of such are Fortran style pseudocode, Pascal style pseudocode, C style pseudocode and Structured Basic style pseudocode.

How to Write Pseudocode - 2

Now that you know what pseudocode looks like, it’s time to learn key aspects of how to write it.

We’re going to break down the process of writing pseudocode step by step so that you know exactly how to write it and how to use it effectively.

**1. Understand the Uses**

It’s difficult to use pseudocode if you don’t actually understand its many uses.

For starters, pseudocode simply makes the task of creating a new computer program more simple and straightforward.

Writing out the code in English enables you to create a verbal outline to follow during the programming stages of the project.

Pseudocode gives you the tools needed to ensure that everything you need will be included during programming. It lets you catch mistakes before they become mistakes.

It’s also highly beneficial to use pseudocode for group projects. It breaks the program down in a simple manner so that all programmers are on the same page.

**2. Pseudocode is Subjective**

The trickiest thing about pseudocode is that it is subjective.

There is no standard way to write pseudocode. The goal is simply to properly outline everything in your mind.

That said, there are certain structures and standard procedures you should use if you’re working with others. Follow these rules to ensure that everyone else on the team is on the same page.

Perhaps the most important rule is to place clarity first. Make your pseudocode as clear and concise as possible, so there is no question as to what you mean.

**3. Algorithms and Basic Constructs**

Two of the most important things you must understand when it comes to writing pseudocode are algorithms and basic algorithm constructs.

* **Understand Algorithms –** An algorithm is the steps you must take to achieve a specific goal.
* **Know Algorithm Flow –** The most basic algorithm construct or flow is “sequence,” “selection,” and “iteration.” These lay out the proper way to write the code.
* **Combine the Pieces –** Take the information you want to relay and use algorithm flow to create a straightforward outline.

**4. Standard Procedure**

As mentioned above, there is no standard procedure for writing pseudocode. However, that doesn’t mean there aren’t certain rules you should follow.

Follow these basic rules to ensure that everyone you’re collaborating with understands your pseudocode.

* **One Statement Per Line –** Express each statement or action on its own line.
* **Capitalize Directions –** Capitalize directions to highlight their importance (for example, “READ”).
* **Focus on Meaning –** Write what the program will do. Don’t write how to program it.
* **Standard Programming Structures –** Follow the algorithm flow discussed above to create easy-to-follow structures.
* **Utilize Blocks –** Group similar actions together into blocks to separate the pseudocode into separate steps.

**5. Important Tips**

Once again, while pseudocode doesn’t have any hard and fast rules, there are certainly some things you should do to make understanding the pseudocode easier for all involved.

* **Keep It Simple –** Simplicity and clarity are key. Write down what the actions will be, not how to program them.
* **Explain Everything –** Don’t include information without explaining it. Add comments to explain your steps and reasoning if needed.
* **Practice Makes Perfect –** Just like learning a new programming language, learning how to write pseudocode takes time. Practice writing and reviewing it now.
* **Review the Pseudocode –** The biggest reason to write pseudocode is to catch any mistakes before programming. So, review the finished product thoroughly to nip errors in the bud.
* **Translate into Programming Language –** Implement the pseudocode by tracing it with your computer language. Compare the finished product to the pseudocode.

### Pseudocode: Practice

**Practice 1:** Write a pseudocode that reads two numbers and multiplies them together and print out their product.

Begin

INPUT number1, number2

SET answer to num1 \* num2

OUTPUT answer

End

**Practice 2:** Write a pseudocode that tells a user that the number they entered is not a 5 or a 6.

Begin

INPUT number

IF (number = 5)

OUTPUT "your number is 5"

ELSE IF (number = 6)

OUTPUT "your number is 6"

ELSE

OUTPUT "your number is not 5 or 6"

End

**Practice 3:** Write a pseudocode to print all multiples of 5 between 0 and 100 (including both 0 and 100).

Begin

SET x to 0

WHILE (x <= 20)

OUTPUT x \* 5

x = x + 1

End

**Practice 4:** Write a pseudocode that will count all the even numbers up to a user-defined stopping point.

For example, say we want to see the first 5 even numbers starting from 0.

Well, we know that evens numbers are 0, 2, 4, etc.

The first 5 even numbers are 0, 2, 4, 6, 8.

The first 8 even numbers are 0, 2, 4, 6, 8 ,10 ,12, 14

Begin

INPUT stopping\_point

SET x to 0

SET even to 0

WHILE (x < stopping\_point)

OUTPUT even

x = x + 1

SET even to even + 2

End

**Practice 5:** Write a pseudocode that will perform the following.

1. Read in 5 separate numbers.
2. Calculate the average of the five numbers.
3. Find the smallest (minimum) and largest (maximum) of the five entered numbers.
4. Write out the results found from steps 2 and 3 with a message describing what they are

Begin

OUTPUT "please enter 5 seperate numbers"

INPUT n1, n2, n3, n4, n5

OUTPUT "The average is:"

SET avg to (n1 + n2 + n3 + n4 + n5) / 5

OUTPUT avg

IF (n1 < n2)

SET max to n2

ELSE

SET max to n1

IF (n3 > max)

SET max to n3

IF (n4 > max)

SET max to n4

IF (n5 > max)

SET max to n5

OUTPUT "The max is:"

OUTPUT max

IF (n1 > n2)

SET min to n2

ELSE

SET min to n1

IF (n3 < min)

SET min to n3

IF (n4 < min)

SET min to n4

IF (n5 < min)

SET min to n5

OUTPUT "The min is:"

OUTPUT min

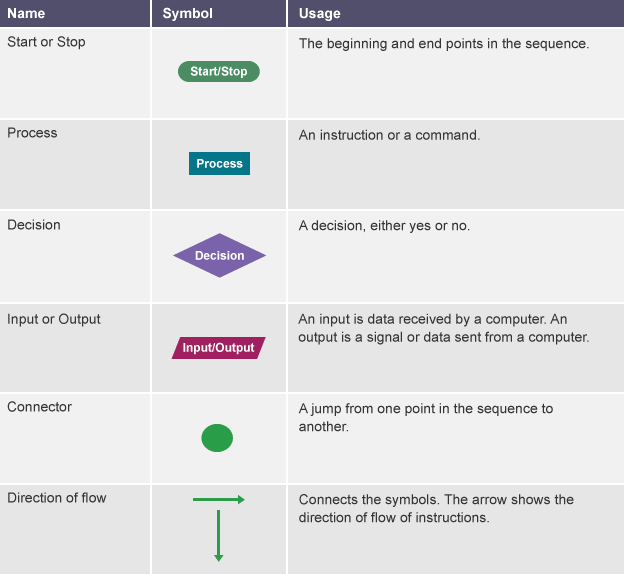
End

## Flowchart

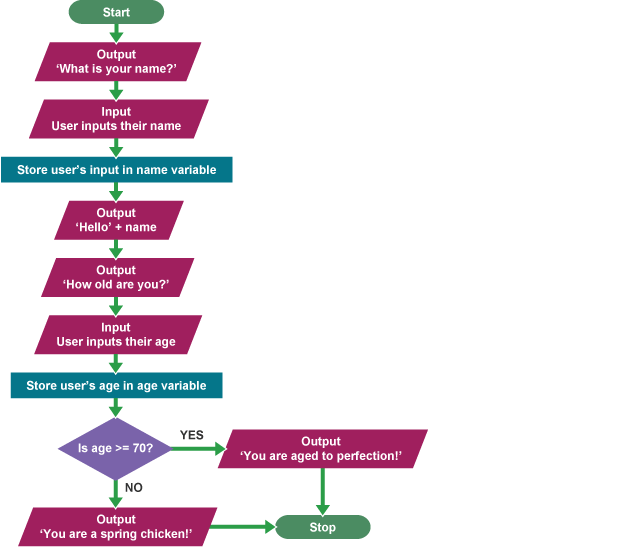
### Representing an Algorithm: Flowchart

A flowchart is a diagram that represents a set of instructions. Flowcharts normally use standard symbols to represent different instructions. There are few real rules about the level of detail needed in a flowchart. Sometimes flowcharts are broken down into many steps to provide a lot of detail about exactly what is happening. Sometimes they are simplified so that a number of steps occur in just one step.

**Flowchart symbols**

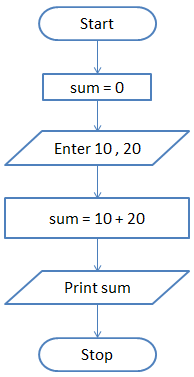


A **simple** program could be created to ask someone their name and age, and to make a comment based on these. This program represented as a flowchart would look like this:



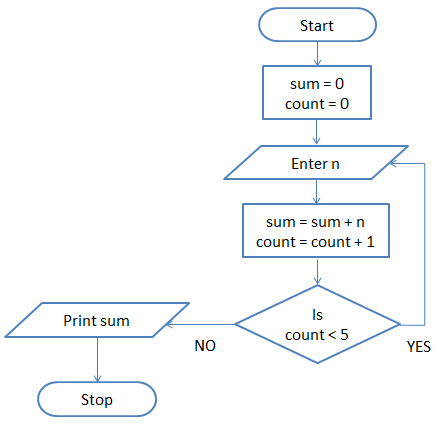
### Flowchart: Practice

**Practice 1:** Add 10 and 20 then display the sum.



**Practice 2:** Take 5 numbers from the user and display the sum of the numbers.

In this question, we are asked to find the sum of 5 numbers. So, we will take two variables - sum and count and set both of them to zero. The sum variable will store the result while the count variable will keep track of how many numbers we have read. To solve this problem we will use the concept of a loop. In loop or iterative operation, we execute some steps repeatedly as long as the given condition is TRUE. In this case, we will keep reading the input until we have read 5 numbers. So, we first initialize sum and count to zero. Then we will take the input and store it in a variable n. Next, we will add the value stored in n to sum and save the answer in sum. i.e., sum = sum + n Then we will increment the count by 1 and check if the count is less than 5. If this condition is TRUE then we will take another input. If the condition is FALSE then we will print the value stored in the variable sum.



**Practice 3:** Print Hello World 10 times.

This problem is also solved using the loop concept. We take a variable count and set it to zero. Then we print "Hello World" and increment count by 1. i.e., count = count + 1 Next, we check if the count is less than 10. If this is TRUE then we again print "Hello World" and increment the variable count. On the other hand, if the condition is FALSE then we will stop.

