# How to write Pseudocode

Pseudocode is an informal program description that does not contain code syntax or underlying technology considerations. Pseudocode summarizes a program's steps (or flow) but excludes underlying [implementation] details. -- techopedia.com

Pseudocode allows you to write easy to read but precise instructions for a program or algorithm. There are many ways to write pseudocode, to reduce uncertainty we will use the following conventions in this course when required.

---

## General Rules

Use upper case for all reserved words. All statements inside loops condition statement are to be indented.

```
CALL
CASE … ENDCASE
CREATE
DO … WHILE … ENDDO
FOR … ENDFOR
IF … THEN … ENDIF
IF … ENDIF
IF … THEN … ELSE … ENDIF
METOD
PRINT
READ
RETURN
WHILE … ENDWHILE
```

---

## Variables

Variables are used to store data. To assign a value to a variable use ← .

Windows/Linux
1. Check that Num Lock key is pressed to activate the numeric keyboard section.
2. Hold Alt key and type 27 on number keyboard

Mac
1. Pull down the Edit menu
2. Choose Emoji & Symbols or Special Characters
3. Locate and click on ←

| | |
|---|---|
| *letter grade* ← A | stores the value A in the variable *letter grade* |
| *grade* ← 87 | stores the value 87 in the variable *grade* |
| *grades* [3] ← 100 | stores the value 100 in the third element of the array *grades* |

| | |
|---|---|
| *grade* ← *grades* [3] | stores the value of the third element of the array *grades* in the variable *grade* |
| *grade with bonus* ← *grade* + 10 | stores the value *grade* + 10 in the *grade with bonus* variable; the value of the variable *grade* stays the same |

**Simple Actions**
A single line of instructions

READ the next grade
PRINT 'Hello!'
PRINT the average

---

## Conditions

Standard conditional statements:

- **IF … ENDIF**
- **IF … THEN … ENDIF**
- **IF … THEN … ELSE … ENDIF**
- **CASE … ENDCASE**

**Important Notes:** Any instructions that occur inside a selection must be indented.
The statement must show appropriate closing word

```
IF … ENDIF     not  IF … ENDCONDITION      not   IF … END
```

**IF … THEN … ELSE … ENDIF**
Binary choice on a given Boolean condition is indicated by the use of four keywords: IF, THEN, ELSE, and ENDIF. The general form is:

```
IF (condition) THEN
       statement block 1
ELSE
       statement block 2
ENDIF
```

The ELSE keyword and "statement block 2" are optional. If the condition is true, statement block 1 is performed, otherwise statement block 2 is performed.

**Example**

```
IF (HoursWorked > NormalMax) THEN
    Display overtime message
ELSE
    Display regular time message
ENDIF
```

**CASE Statement**
A CASE construct indicates a multiway branch based on conditions that are mutually exclusive. Four keywords, CASE, OF, OTHERS, and ENDCASE, and conditions are used to indicate the various alternatives. The general form is:

```
CASE expression OF
      condition 1 : statement block 1
      condition 2 : statement block 2
      ...
      condition n : statement block n
OTHERS:
      default statement block
ENDCASE
```

The OTHERS clause with its default statement block is optional. Conditions are normally numbers or characters indicating the value of "expression", but they can be English statements or some other notation that specifies the condition under which the given statement block is to be performed. A certain statement block may be associated with more than one condition.

**Example**

```
CASE Title OF
     Mr    : PRINT "Mister"
     Mrs   : PRINT "Missus"
     Miss  : PRINT "Miss"
     Ms    : PRINT "Mizz"
     Dr    : PRINT "Doctor"
ENDCASE
```

**Example**

```
CASE grade OF
     A  : points ← 4
     B  : points ← 3
     C  : points ← 2
     D  : points ← 1
     F  : points ← 0
ENDCASE
```

---

## Loops

Standard looping structures:

- **FOR … ENDFOR**
- **WHILE … ENDWHILE**
- **DO … WHILE … ENDDO**

**Important Notes:** Any instructions that occur inside an iteration must be indented. The statement must show appropriate closing word

```
FOR … ENDFOR        not   FOR … ENDLOOP      not    FOR … END
```

**WHILE Loop:**
The WHILE construct is used to specify a loop with a test at the top. The beginning and ending of the loop are indicated by two keywords WHILE and ENDWHILE. The general form is:

```
WHILE (condition = true)
       statement block
ENDWHILE
```

The loop is entered only if the condition is true. The "statement block" is performed for each iteration. At the conclusion of each iteration, the condition is evaluated and the loop continues as long as the condition is true.

**Example**

```
WHILE (Population < Limit)
      Population ← Population + Births - Deaths
ENDWHILE
```

**Example**

```
WHILE (employee.type !=  manager AND personCount < numEmployees)
       personCount ← personCount + 1
       CALL employeeList.getPerson with personCount RETURNING
employee
ENDWHILE
```

**REPEAT-UNTIL Loop:**
This loop is similar to the WHILE loop except that the test is performed at the bottom of the loop instead of at the top. Two keywords, REPEAT and UNTIL are used. The general form is:

```
  REPEAT
       statement block
  UNTIL condition
```

The "statement block" in this type of loop is always performed at least once, because the test is performed after the statement block is executed. At the conclusion of each iteration, the condition is evaluated, and the loop repeats if the condition is false. The loop terminates when the condition becomes true.

**FOR Loop:**
This loop is a specialized construct for iterating a specific number of times, often called a "counting" loop.  Two keywords, FOR and ENDFOR are used. The general form is:

```
FOR iteration bounds
    statement block
ENDFOR
```

In cases where the loop constraints can be obviously inferred it is best to describe the loop using problem domain vocabulary.

**Examples**

```
FOR each month of the year (good)
FOR month = 1 to 12 (ok)
FOR each employee in the list (good)
FOR empno = 1 to listsize (ok)
```

---

## Nested Constructs

The constructs can be embedded within each other, and this is made clear by use of indenting. Nested constructs should be clearly indented from their surrounding constructs.

**Example**

```
total ← zero
REPEAT
    READ Temperature
    IF Temperature > Freezing THEN
       total  ← total + 1
    ENDIF
UNTIL Temperature < zero
PRINT total
```

In the above example, the IF construct is nested within the REPEAT construct, and therefore is indented.

---

## Calling Subprograms or Methods

Use the CALL keyword. For example:

```
CALL AvgAge with StudentAges
CALL Swap with CurrentItem and TargetItem
CALL Account.debit with CheckAmount
CALL getBalance RETURNING aBalance
CALL SquareRoot with orbitHeight RETURNING nominalOrbit
```

**Example:**

**“Adequate”**

```
FOR X = 1 to 10
    FOR Y = 1 to 10
        IF gameBoard[X][Y] = 0
            Do nothing
        ELSE
            CALL theCall(X, Y) (recursive method)
            increment counter
        ENDIF
    ENDFOR
ENDFOR
```

**"Better"**

```
moveCount ← 1
FOR each row on the board
    FOR each column on the board
        IF gameBoard position (row, column) is occupied THEN
            CALL findAdjacentTiles with row, column
            INCREMENT moveCount
        ENDIF
    ENDFOR
ENDFOR
```

(Note: the logic is restructured to omit the "do nothing" clause)

**Example:**

**"Not So Good"**

```
FOR all the number at the back of the array
    SET Temp equal the addition of each number
    IF > 9 THEN
        get the remainder of the number divided by 10 to that
index
        and carry the "1"
    Decrement one
Do it again for numbers before the decimal
```

**"Good Enough (not perfect)"**

```
Carry ← 0
FOR each DigitPosition in Number from least significant to most
significant
   Total ← FirstNum[DigitPosition] + SecondNum[DigitPosition]
    IF Total > 10 THEN
       Carry ← 1
```

```
        Total ← Total - 10
    ELSE
        Carry ← 0
    ENDIF
    Result[DigitPosition] ← Total
ENDFOR
IF Carry = 1 THEN
    RAISE Overflow exception
ENDIF
```

**"Pretty Good"**

```
IF robot has no obstacle in front THEN
        Call Move robot
        Add the move command to the command history
        RETURN true
ELSE
        RETURN false without moving the robot
ENDIF
```

**Example:** Write pseudocode to read three positive numbers and printout their average.

```
METHOD FIND_AVERAGRE
BEGIN
   average ← 0
   FOR each of three numbers
       READ input_value
   sum ← sum + input_value
   ENDFOR
   average ← sum / 3.0
PRINT average
END FIND_AVERAGRE
```

**Example:** Write pseudocode to read ten numbers and find the maximum value.

```
METHOD FIND_MAX
BEGIN
   READ input_value
   max ← input_value  //assume first value is the max
   FOR each of next nine inputted numbers
       READ input_value
   IF max < input_value
      max ← input_value
   ENDIF
   ENDFOR
   PRINT max
END FIND_MAX
```

**Example:** Write pseudocode to read and store grades into an array (assume array size is 30), then print out their total followed by their average.

```
METHOD GRADES
BEGIN
   // read the grades into the array
   CREATE array grades[30]
   FOR each inputted grade
       READ grade_value
   grades[I] ← grade_value
   ENDFOR

   // add up all grades
   total ← 0;
   FOR each element in the array
   total ← total + grades[I]
   ENDFOR

   // determine their average
   average ← total / 30.0
   PRINT total
   PRINT average
END GRADES
```