Pseudocode

Table of Contents

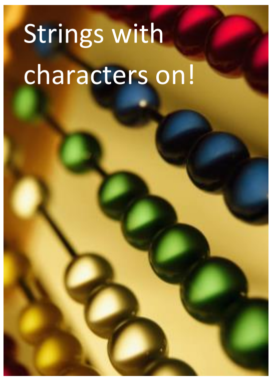
Summary

Pseudocode is a way of describing what happens in a computer program. It is set out with the same structure as programming languages but it is meant for human reading instead of machine reading. Pseudocode can be used to show people who don't know the programming language how the program works or programmers can use it to plan how they are going to make a program.

Variables

A variable is data item which has a quantity that is likely to change throughout a program.

Data Type	Description
Integer	An integer is a data type for representing whole numbers.
Boolean	A Boolean is a variable that can hold a single true or false value.
Real	A number that can be written as a decimal (a non-recurring decimal).
Character	A character is a unit or symbol, including letters, numerical digits and common punctuation.
String	A string is a data type which holds a series of characters.



How do you set the value of a variable?

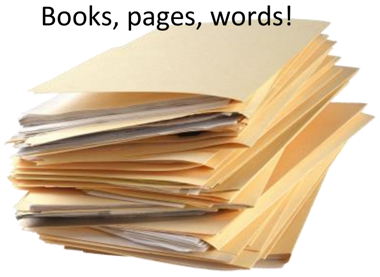
Shown in pseudocode:

```
var ← exp
```

This assigns the value of exp to var.

Arrays

An array is a data type that describes a collection of values or variables.



To set the values in an array using pseudocode:

```
myArr ← [1, 6, 23, 41]
```

If you have a variable called Book, one dimensional arrays can deal with the page numbers in the Book. Book [63]. Two dimensional arrays can deal with the page numbers and the words in the book – Book [63] [2].

Comments

It is important to write comments when writing pseudocode or real code because without comments, code can become confusing. Comments help you and other people understand your code and it will make it easier to change and fix things

How to write a comment in pseudocode:

Syntax:

```
# some text
```

Example:

```
# this is a comment! hurrah
```

Conditional Statements

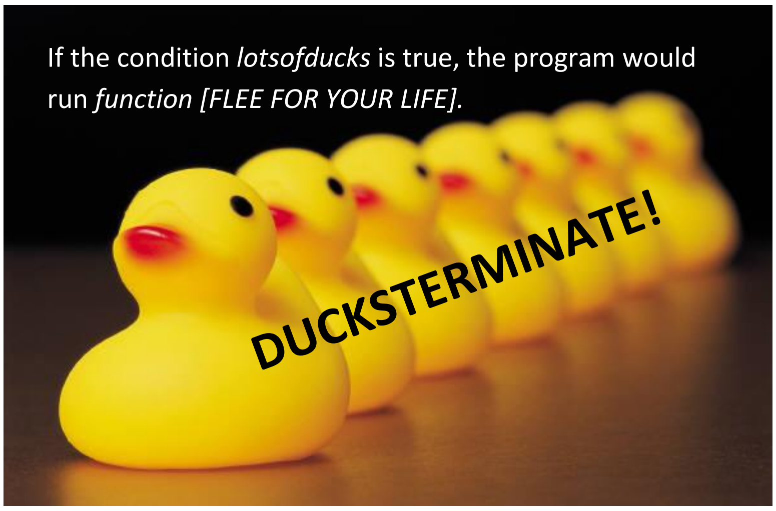
Conditional statements run depending on the value of a condition. For example, if a condition *lotsofducks* is true, then the program might do one thing, and if the condition was false, it would do a different thing.

How to write an IF statement:

```
IF myVar < 15 THEN
     myVar ← myVar + 1
ELSE

      OUTPUT myVar

ENDIF
```


If the condition *lotsofducks* is true, the program would run *function [FLEE FOR YOUR LIFE].*

How to write a CASE statement:

```
num ← 3 CASE num OF
1: OUTPUT "one"
2: OUTPUT "two"
3: OUTPUT "three"
4: OUTPUT "four"
ELSE
     OUTPUT "Out of range"
ENDCASE
```

How to decide between an IF statement and a CASE statement:

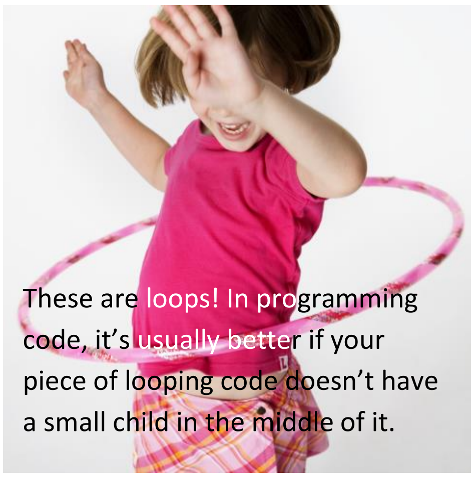
If you have only two different choices, it's easier to use an IF statement.

If you have more than two choices, it's easier to use a CASE statement (you can use an IF statement if there's more than two choices, but it gets messy and irritating.)

Loops

Sometimes, you might need to make a section of code run over and over again. For example, if you wanted a snowflake to continue falling throughout the program, you'd need the section of code that makes the snowflake fall to repeat, or *loop*.

Loops can be made using WHILE, FOR and REPEAT.


These are loops! In programming code, it's usually better if your piece of looping code doesn't have a small child in the middle of it.

While

```
WHILE myVar =/ 100
    OUTPUT myVar
     myVar ← myVar + 1

ENDWHILE
```

While the variable isn't equal to 100, the variable will be printed and the value of the variable will be increased by 1.

For

```
FOR i ← 1 TO 5
    OUTPUT i
ENDFOR
```

Will print the value of i, which is 1 to 5 – so it'll output 1, 2, 3, 4 and 5.

Repeat

```
REPEAT
    OUTPUT "Enter a number"
    num ← INPUT
    OUTPUT num
UNTIL num = 5
```

Will print "Enter a number". When a number is entered, this number will be repeated. This will be done until the number entered = 5.

Input and Output

Most programs will need to receive information from the user. For example, the program might need the user's name, which the user will have to type in. This type of information is called input information. Most programs will also have to output information to the user. For example, the user might need to know his score, which will be stored in one of the program's variables. The program would need to output the value of his variable. This is called output information.

Readline

Input
Magic box!
Output

```
if contents of file fruit.txt is:
L1 apple
L2 banana
L3 clementine
line2 ← READLINE(fruit.txt, 2)
```

This will read the second line of the document. The program will read "L2 banana".

Writeline

```
newfruit ← "L4 dragonfruit"
WRITELINE(fruit.txt, 4, newfruit)
WRITELINE(fruit.txt, 2, "empty")
```

```
Contents of fruit.txt is now:
L1 apple
empty
L3 clementine
L4 dragonfruit
```

Writeline will overwrite a line in the document.

Output

```
greeting ← ”hello”
OUTPUT “hi”
OUTPUT greeting

#OUTPUT greeting outputs “hello”
```

Output will print messages, either strings or variables.

Userprint

```
answer ← USERINPUT
```

Waits for the user to type something in and then assigns what they type in to the variable.

Functions and Procedures

Sometimes, it’s easier to reuse sections of code rather than write the whole thing out again. You can do this by using procedures and functions.

Procedures

Procedures are sections of code that can be reused.

```
PROCEDURE PoliteProc()
    OUTPUT “Enter your name”
    Name ← USERINPUT
    OUTPUT “Nice to meet you”
    OUTPUT Name
ENDPROCEDURE
```

Functions

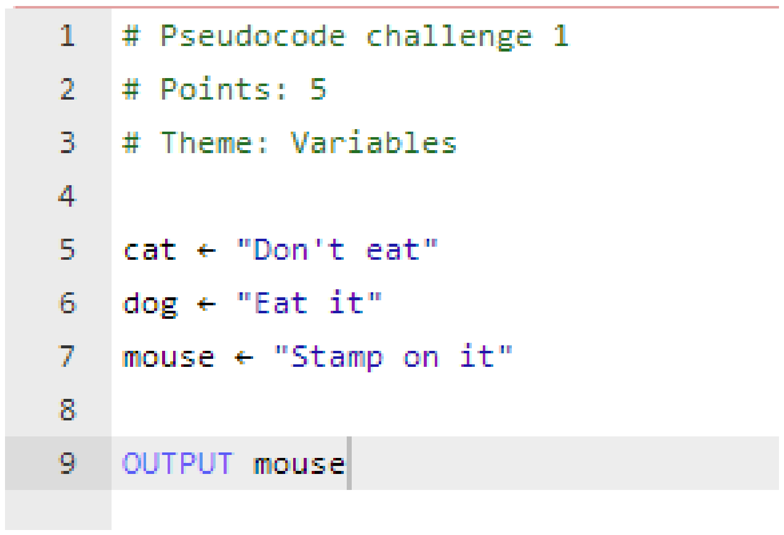
Functions are sections of code that can be reused and can return a value.

```
FUNCTION IsMember(myArr,val)
    FOR i ← 1 TO LEN(myArr)
        IF myArr[i] = val THEN
           RETURN true
                      ENDIF
                  ENDFOR
                  RETURN false

ENDFUNCTION
```

Examples

Example 1



```
1  # Pseudocode challenge 1
2  # Points: 5
3  # Theme: Variables
4
5  cat ← "Don't eat"
6  dog ← "Eat it"
7  mouse ← "Stamp on it"
8
9  OUTPUT mouse
```

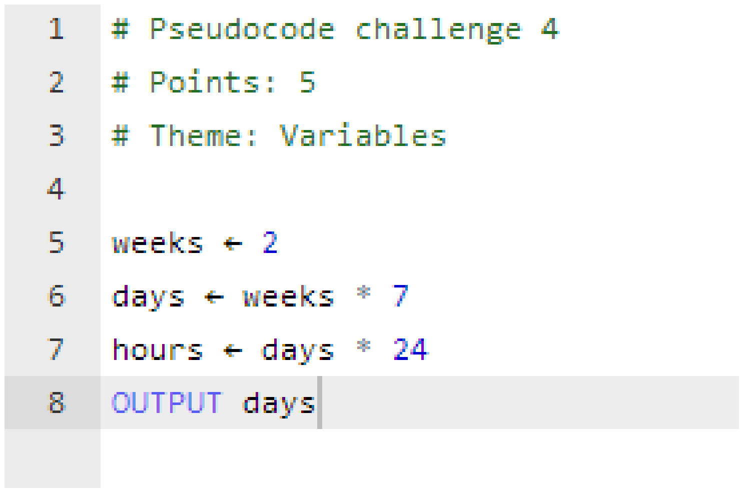
Line 5: The variable *cat* has the value of "Don't eat".

Line 6: The variable *dog* has the value of "Eat it".

Line 7: The variable *mouse* has the value of "Stamp on it".

Line 9: If you output *mouse*, the value "Stamp on it" will be outputted.

Example 2



```
1  # Pseudocode challenge 4
2  # Points: 5
3  # Theme: Variables
4
5  weeks ← 2
6  days ← weeks * 7
7  hours ← days * 24
8  OUTPUT days
```

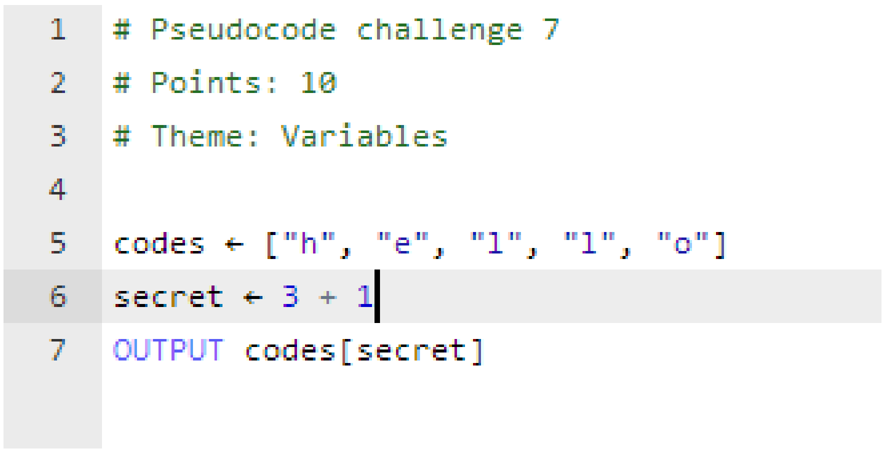
Line 5: The value of *weeks* is 2.

Line 6: The value of *days* is *weeks* multiplied by 7. Therefore, the value of *days* is 14.

Line 7: The value of *hours* is the value of *days* multiplied by 24.

Line 8: So, outputting *days* will output '14'.

Example 3



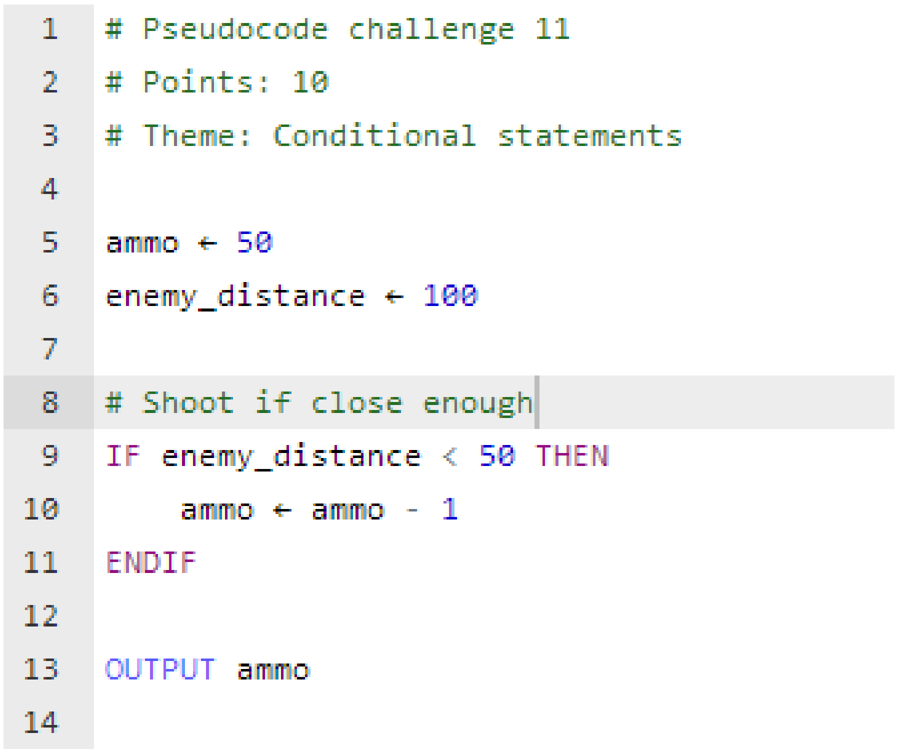
```
1  # Pseudocode challenge 7
2  # Points: 10
3  # Theme: Variables
4
5  codes ← ["h", "e", "l", "l", "o"]
6  secret ← 3 + 1
7  OUTPUT codes[secret]
```

Line 5: The value of *codes* is an array. So *codes* can be equal to h, e, l, l, or o.

Line 6: The value of *secret* is 3 + 1 - so *secret* is 4.

Line 7: Outputting *codes[secret]* is the same as saying *codes[4].* So, you output the fourth value of *codes* – which is 'l'.

Example 4



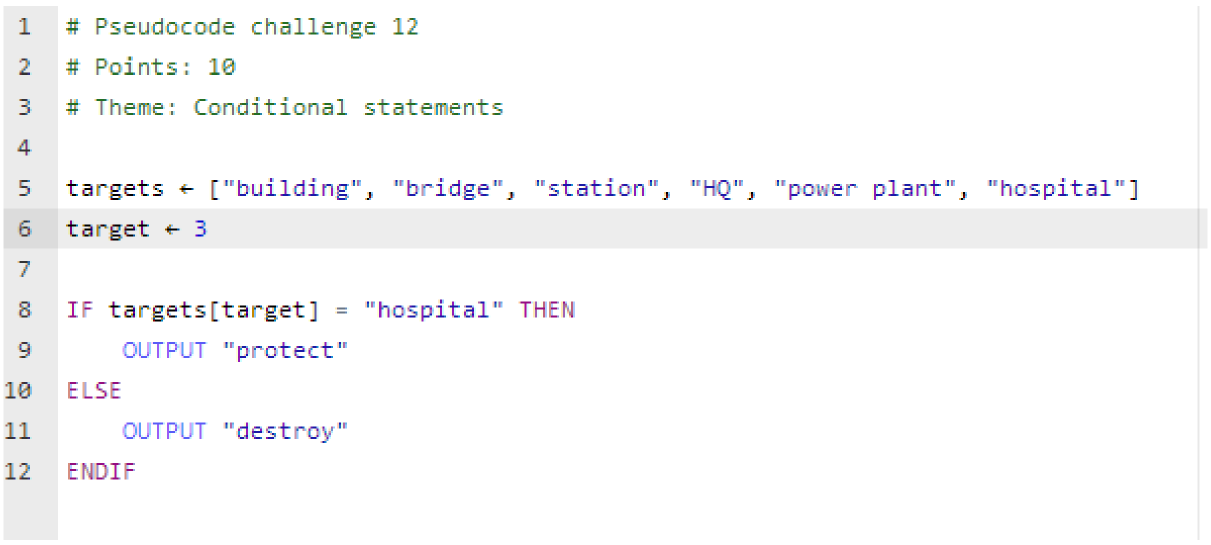
```
1   # Pseudocode challenge 11
2   # Points: 10
3   # Theme: Conditional statements
4
5   ammo ← 50
6   enemy_distance ← 100
7
8   # Shoot if close enough
9   IF enemy_distance < 50 THEN
10      ammo ← ammo - 1
11  ENDIF
12
13  OUTPUT ammo
14
```

Line 5 – 6: The value of *ammo* is 50 and the value of *enemy_distance* is 100.

Line 9 – 11: If the *enemy_distance* is less than 50, then the value of *ammo* is subtracted by one. Since the enemy_distance is more than 50, the value of *ammo* remains the same.

Line 13: Therefore, the value '50' is outputted.

Example 5



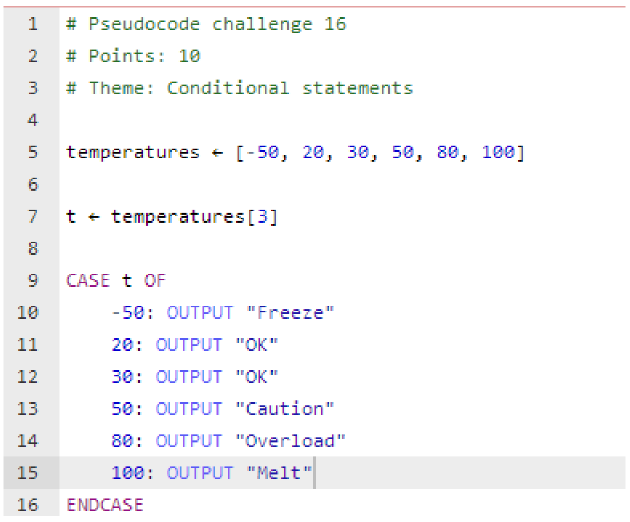
```
1   # Pseudocode challenge 12
2   # Points: 10
3   # Theme: Conditional statements
4
5   targets ← ["building", "bridge", "station", "HQ", "power plant", "hospital"]
6   target ← 3
7
8   IF targets[target] = "hospital" THEN
9       OUTPUT "protect"
10  ELSE
11      OUTPUT "destroy"
12  ENDIF
```

Line 5 – 6: The value of *targets* is an array. The value of *target* is 3.

Line 8: *Targets[target]* is the same as *targets[3]* – the third thing in the *targets* array*,* which is 'station'.

Line 8 – 12: If *targets[target]* equalled 'hospital', then it would output 'protect'. However, *targets[target]* doesn't equal 'hospital', in which case the program would output 'destroy'.

Example 6

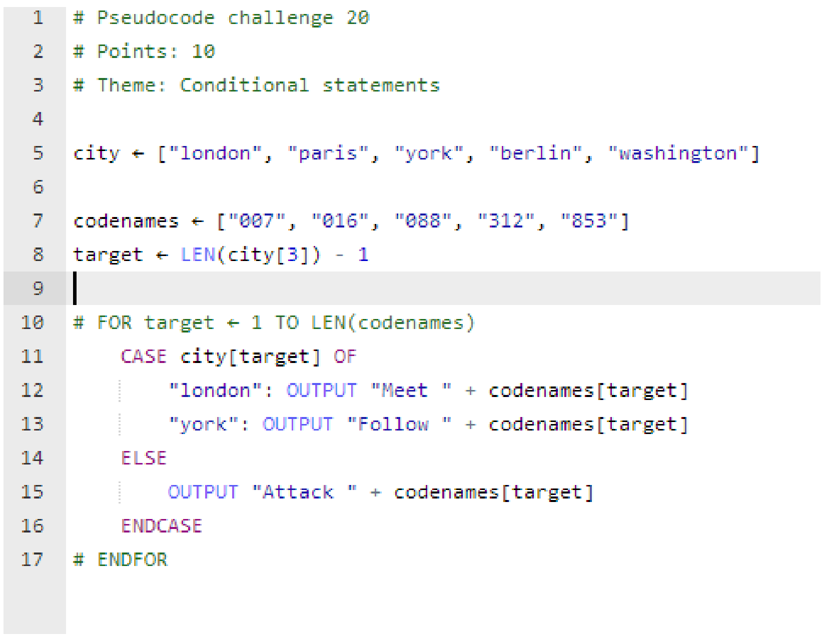


```
1   # Pseudocode challenge 16
2   # Points: 10
3   # Theme: Conditional statements
4
5   temperatures ← [-50, 20, 30, 50, 80, 100]
6
7   t ← temperatures[3]
8
9   CASE t OF
10      -50: OUTPUT "Freeze"
11      20: OUTPUT "OK"
12      30: OUTPUT "OK"
13      50: OUTPUT "Caution"
14      80: OUTPUT "Overload"
15      100: OUTPUT "Melt"
16  ENDCASE
```

Line 5 – 7: The value of *temperature* is an array. The value of *t* is the third value in the array of temperature, so, 30.

Line 9 – 16: The CASE bit says that if *t* is 30, the program should output "OK". So, the program outputs "OK".

Example 7



```
1   # Pseudocode challenge 20
2   # Points: 10
3   # Theme: Conditional statements
4
5   city ← ["london", "paris", "york", "berlin", "washington"]
6
7   codenames ← ["007", "016", "088", "312", "853"]
8   target ← LEN(city[3]) - 1
9
10  # FOR target ← 1 TO LEN(codenames)
11      CASE city[target] OF
12          "london": OUTPUT "Meet " + codenames[target]
13          "york": OUTPUT "Follow " + codenames[target]
14      ELSE
15          OUTPUT "Attack " + codenames[target]
16      ENDCASE
17  # ENDFOR
```

Line 5 – 7: The value of *city* and *codenames* is an array.

Line 8: The value of *target* is the length of the third thing in *city* take away one. The third thing in *city* is York, which is 4 letters long. 4 take away 1 is 3. So, the **value of target is 3!**

Line 11 – 14: In the CASE bit, if the *city[target]* is York (city[3] is York) , you output "Follow" and *codenames[target]. Codenames[target]* is the same as *codenames[3],* and the third thing in *codenames* is 088. Therefore, you output "Follow 088".