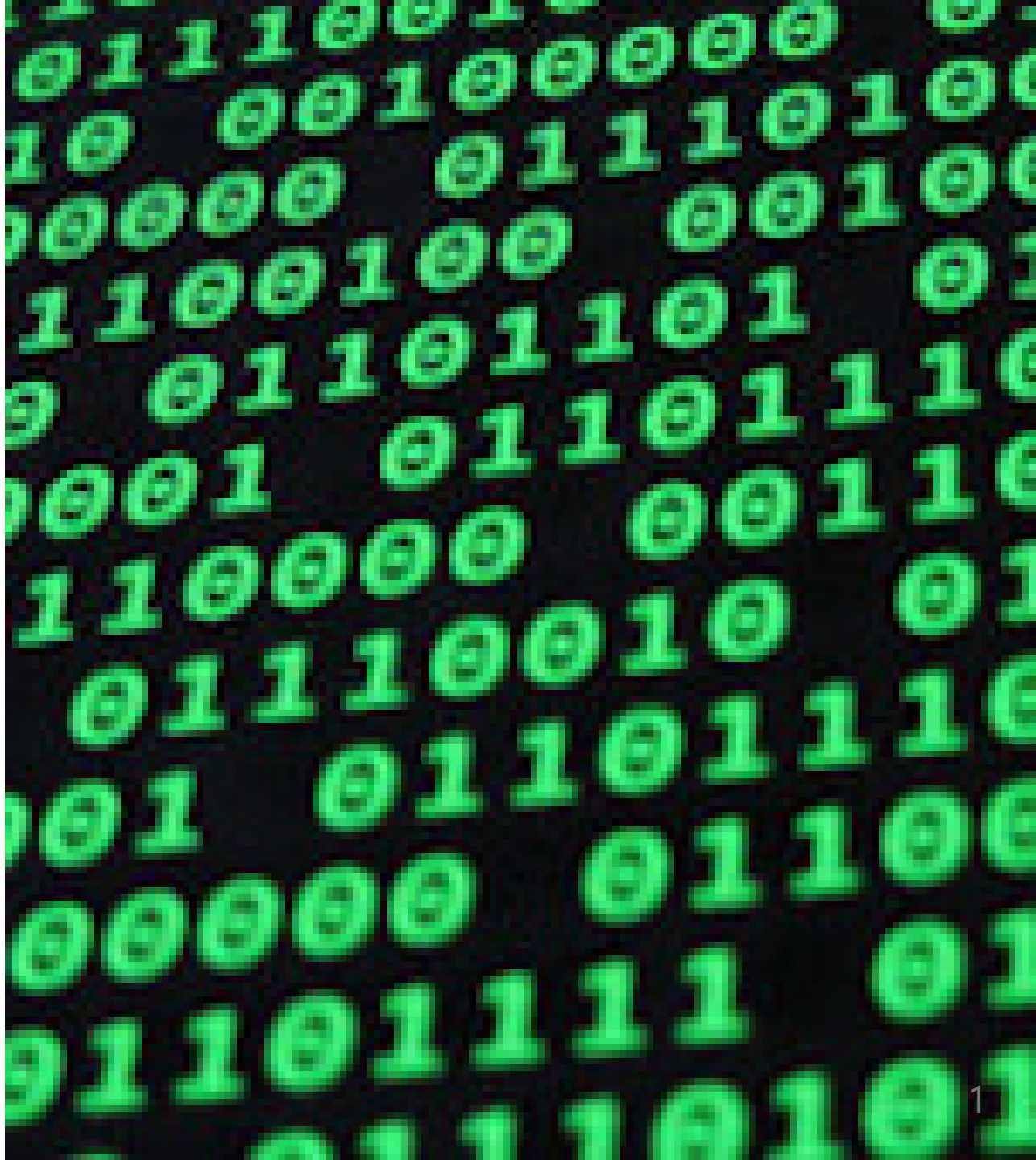
GCSE

Programming



Contents

Contents

Variables, Types and Strings

What is a Variable?

A variable is a **container** or **label** used to store data in a program.

Examples:

- `age = 42`
- `answer = 'Y'`
- `full_name = 'Harry Jones'`
- `is_over_age = True`
- `area_of_circle = 3.142 * radius^2`

Pseudocode Examples: Variables

AQA Pseudocode

```
a <- 3
b <- 'Hello'
```

OCR Pseudocode

```
a = '3'
b = 'Hello'
```

- AQA uses a backward arrow `<-` for assignment, while OCR uses `=`.

Data Types

Data comes in different types:

- Numeric (integer, float)
- Text (string, char)
- Boolean (True/False)

Statically-Typed vs Dynamically-Typed

- **Statically-Typed:** Memory allocation is known at compile time.
- **Dynamically-Typed:** Memory is allocated at runtime.

Data Types and Memory

Data Type	Description	Memory Size	Example
Integer	Whole number (positive/negative)	4 bytes	`42`, `-89`
Float/Real	Number with fractional part	8 bytes	`3.142`
Char	Single character	1 byte	`'a'`, `';'`
String	Sequence of characters	1 byte per char	`'John'`
Boolean	True or False	1 byte	`True`, `False`

Constants

A **constant** is a variable whose value cannot change.

Examples:

- `const PI = 3.142`
- `const VAT_RATE = 0.2`
- In Python, constants are not enforced, but it is good practice to use uppercase for constant names.

Constants in Pseudocode

AQA Pseudocode

```
CONSTANT PI <- 3.142
CONSTANT CLASS_SIZE <- 30
```

- For OCR, uppercase identifiers are used to distinguish constants.

Input and Output

Python:

```
name = input("What is your name?")
print(name)
```

Pseudocode

- AQA

```
a <- USERINPUT
OUTPUT a
```

- OCR

```
name = input("Enter your name: ")
print(name)
```

Arithmetic Operators

Operator	Description	Example	Result
+	Addition	`7 + 3`	10
-	Subtraction	`7 – 3`	4
*	Multiplication	`7 * 3`	21
/	Division	`7 / 3`	2.33..
^	Exponentiation	`7 ^ 3`	343

Integer Division Operators

Operator	Description	Example	Result
DIV	Floor division	`7 // 3`	2
MOD	Modulo (remainder)	`7 % 3`	1

- Both AQA and OCR pseudocode use `MOD` and `DIV`.

Strings

A string represents text, while an integer represents a number.

Example:

- **String:** `'42'` in binary: `0111010101110011`
- **Integer:** `42` in binary: `00101010`

String Conversion Functions

AQA Pseudocode

Function	Description
STRING_TO_INT(str)	Convert string to integer
STRING_TO_REAL(str)	Convert string to float/real
INT_TO_STRING(num)	Convert integer to string
REAL_TO_STRING(num)	Convert real to string

OCR String Conversion Functions

Function	Description
`int(str)`	Convert string to integer
`float(str)`	Convert string to float
`str(num)`	Convert numeric value to string

String Concatenation

Strings can be joined using the `+` operator.

Examples:

```
full_name = first_name + ' ' + last_name
age = '4' + '2'
```

Common String Functions

Function	Description	Example	Output
`str.length`	Length of string	`'john'.length`	4
`str.substring(start, end)`	Slice of string	`'john'.substring(0,2)`	'jo'
`str.upper()`	Convert string to uppercase	`'john'.upper()`	'JOHN'
`str.lower()`	Convert string to lowercase	`'John'.lower()`	'john'

String Functions in Pseudocode

AQA Pseudocode:

```
LEN(str)                  // Get length of string
SUBSTRING(start, end, str) // Slice string
```

OCR Pseudocode:

```
str.length                // Get length of string
str.substring(start, num)  // Get substring
```

Comments in Code

Use comments to describe your code for others and for future reference.

Pseudocode Comments:

- **AQA:** Uses `#` (same as Python)
- **OCR:** Uses `//` (same as JavaScript)

GCSE

Summary

- Variables and Data Types
- Constants
- Input/Output in Pseudocode
- Operators and String Functions

Selection, Boolean, and Random

Selection

Programs execute sequentially unless flow control is changed using **selection** or **iteration**.

Selection Example (Pseudocode)

```
// in pseudocode
age = 15
if age > 17 then
    print("You are old enough to drive")
else
    print("You are not old enough to drive")
endif
```

- The condition `age > 17` evaluates to either **True** or **False**.
- In this case, `age = 15`, so the program prints the statement in the `else` block.

Multiple Branches

```
// Multiple conditions
if age > 18 then
    print("You may be at university")
elif age > 16 then
    print("You may be doing your A Levels")
elif age > 14 then
    print("You will be doing your GCSEs")
elif age > 11 then
    print("You will be at secondary school")
else
    print("You will be at primary school")
endif
```

AQA Pseudocode

```
// AQA
IF age > 18 THEN
    OUTPUT "You may be at university"
ELSE IF age > 16 THEN
    OUTPUT "You may be doing your A Levels"
ELSE IF age > 14 THEN
    OUTPUT "You will be doing your GCSEs"
ELSE IF age > 11 THEN
    OUTPUT "You will be at secondary school"
ELSE
    OUTPUT "You will be at primary school"
ENDIF
```

OCR Pseudocode

```
// OCR
if age > 18 then
    print("You may be at university")
elseif age > 16 then
    print("You may be doing your A Levels")
elseif age > 14 then
    print("You will be doing your GCSEs")
elseif age > 11 then
    print("You will be at secondary school")
else
    print("You will be at primary school")
endif
```

Nested `if` Statements

```
// Nested if
if age <= 16 then
    if day == 'Tuesday' then
        print("20% Discount is available")
    else
        print("10% Discount is available")
    endif
else
    print("Full price only")
endif
```

Switch/Case Statement

A **switch/case** statement simplifies multiple conditions:

```
// Pseudocode example
option = int(input("Enter an option: "))

switch option:
    case 1:
        print("You selected the first option")
    case 2:
        print("You selected the second option")
    case 3:
        print("You selected the third option")
    default:
        print("You made an invalid choice")
endswitch
```

- AQA pseudocode does not use switch/case statements.

Boolean Expressions

Boolean operators compare values.

Operator	Description	Example	Result
`==`	Equal to	`7 == 7`	True
`!=`	Not Equal	`7 != 6`	True
`>`	Greater than	`7 > 6`	True
`<`	Less than	`7 < 6`	False
`>=`	Greater or equal to	`7 >= 7`	True

- Don't confuse `==` (comparison) with `=` (assignment). E.g., `age == 17` asks "is age equal to 17?" while `age = 17` assigns the value 17 to age.

Complex Boolean Expressions

Boolean expressions can be combined with **AND**, **OR**, and **NOT**:

```
// Complex Boolean
if age > 16 AND age < 18 then
    print("You can claim a student discount")
endif
```

Random Number Generation

Random numbers can be generated in Python using the `random` library. In pseudocode, it is assumed to be available.

AQA Pseudocode

```
// AQA pseudocode
value = RANDOM_INT(first, second)  // returns a random integer between `first` and `second`
```

OCR Pseudocode

Random number generation is not included in OCR pseudocode.

Iteration

Remember:

There are three main **programming constructs** used to build algorithms:

- **Sequence**: Code statements are executed one after another.
- **Selection**: A condition is evaluated, deciding which statements to execute.
- **Iteration**: Statements are repeated until a condition is met.

Types of Iteration

1. `for ... next` loop (counted loop)
2. `while ... endwhile` loop (condition loop)
3. `do ... until` loop (repeat-until loop)

> **Note**: Python only supports the first two loops.

`for ... next` Loop (Counted Loop)

The `for ... next` loop repeats a set number of times. It is known as the **counted loop**.

AQA Pseudocode

```
// AQA
sum <- 0
FOR count <- 1 to 10
    sum <- sum + count
ENDFOR
```

OCR Pseudocode

```
// OCR
sum = 0
for count = 1 to 10
    sum = sum + count
next i
print(sum)
```

`while ... endwhile` Loop

The `while ... endwhile` loop runs **while** a condition is `True`. This is called a **top-tested loop**, as the condition is checked before each iteration.

AQA Pseudocode

```
// AQA
sum <- 0
count <- 1
WHILE count <= 10
    sum <- sum + count
    count <- count + 1
ENDWHILE
```

OCR Pseudocode

```
// OCR
sum = 0
count = 1
while count <= 10
    sum = sum + count
    count = count + 1
endwhile
print(sum)
```

Key Difference

- The `while` **loop** requires manual management of the loop counter (`count`).
- If the counter is not updated, an **infinite loop** occurs.

`do ... until` Loop (Bottom-Tested Loop)

The `do ... until` loop (or `repeat ... until`) checks the condition **after** each iteration. The loop always runs at least once.

AQA Pseudocode

```
// AQA
sum <- 0
count <- 1
REPEAT
    sum <- sum + count
    count <- count + 1
UNTIL count == 10
OUTPUT sum
```

OCR Pseudocode

```
// OCR
sum = 0
count = 1
do
    sum = sum + count
    count = count + 1
until count == 10
print(sum)
```

Key Concept

- **Top-tested loops**: Conditions are checked before each iteration (`while` loop).
- **Bottom-tested loops**: Conditions are checked after each iteration (`do ... until` loop).

> **Question**: What happens if the starting value of `count` is 11 in both loops?

Arrays

Python Lists

In Python, we use the `list` data structure to store collections of data.

```
my_shopping_list = ["Milk", "Eggs", "Tomatoes"]
my_values = [1, 6, 87, 34, 23]
my_mixed_list = ["Milk", 45, "Flour", 23.99]
my_sub_list = [["Milk", "Eggs"], [45, 3.142]]
```

Lists are **mutable**, meaning their contents can change during the program's run.

Accessing List Items

Items in the list can be referenced using their index, which starts at `0`.

- `my_shopping_list[0]` → "Milk"
- `my_shopping_list[2]` → "Tomatoes"

> **Note**: In pseudocode, the assignment operator `<-` is used instead of `=`.

Array vs List

- In Python, lists like `my_shopping_list` are referred to as **arrays**.
- Lists like `my_mixed_list` are not arrays because **arrays** require all elements to be of the same data type.

2D Arrays

Arrays can have multiple dimensions. For example:

```
marks = [[19, 16, 14, 16], [12, 8, 11, 14], [20, 17, 12, 8]]
```

This represents marks achieved by three students across four tests:

test 1	test 2	test 3	test 4
19	16	14	16
12	8	11	14
20	17	12	8

To access specific marks, we use two indices: `marks[0][1]`, `marks[2][0]`.

Processing a 2D Array

To display all the marks and calculate totals for each student, we can use a loop in pseudocode.

AQA Pseudocode

```
FOR student <- 0 to 2
    student_total <- 0
    FOR mark <- 0 to 3
        student_total <- student_total + marks[student][mark]
    ENDFOR
    OUTPUT student_total
ENDFOR
```

OCR Pseudocode

```
for student = 0 to 2
    student_total = 0
    for mark = 0 to 3
        student_total = student_total + marks[student, mark]
    next mark
next student
print(student_total)
```

Note: In OCR pseudocode, indices are separated by commas within the same brackets.

Functions and Procedures

Overview

A **subroutine** is a block of code intended to be reused or grouped together for manageability.

- **Function**: Returns a value to the calling program.
- **Procedure**: Does not return a value to the calling program.

Examples:

- `print()` → Procedure (outputs text, no return value)
- `input()` → Function (returns user input)

Subroutines with Arguments

Subroutines can take **arguments**—values passed during the call:

```
print("Hello World")  # "Hello World" is the argument
your_name = input("Enter your name")  # "Enter your name" is the argument
```

Procedures

A common use case for procedures is displaying a menu:

```
procedure show_menu()
    print("      MAIN MENU      ")
    print("1. Play game")
    print("2. Show controls")
    print("3. Display high scores")
    print("4. QUIT")
endprocedure

show_menu()  # Call procedure to display the menu
```

Functions

Functions return a value to the calling program:

```
function get_user_choice()
    option = int("Select > ")
    return option
endfunction
```

This can be used with the `show_menu()` procedure:

```
procedure main()
    do
        show_menu()
        option = get_user_choice()
        process_choice(option)
    until option == 4
endprocedure
```

Parameters in Subroutines

When defining a subroutine, **parameters** are declared in parentheses:

```
function sum(a, b)
    return a + b
endfunction

total = sum(10, 12)  # Calling with arguments
```

Pseudocode Examples

AQA Function

```
SUBROUTINE sum(a, b)
    result <- a + b
    RETURN result
ENDSUBROUTINE
```

OCR Function

```
function sum(a, b)
    result = a + b
    return result
endfunction
```

Benefits of Subroutines

1. **Code Reusability**: Reduces duplication.
2. **Modularity**: Breaks down large programs.
3. **Easier Debugging**: Debug sections independently.
4. **Improved Readability**: Clear structure.
5. **Reduced Complexity**: Simplifies code.
6. **Abstraction**: Hides implementation details.
7. **Faster Development**: Speeds up coding.
8. **Collaboration**: Facilitates teamwork.

Variables and Scope

- **Local Variables**: Defined inside functions, accessible only within.
- **Global Variables**: Defined outside functions, accessible anywhere.

Example:

```
x = 10  # Global variable

procedure myProcedure()
    y = x + 20  # Local variable
    print(y)  # Outputs: 30
endprocedure

print(x)  # Outputs: 10
print(y)  # Error: y is not defined
```

File Handling

Overview

File handling allows programs to read and write data to/from **text files** (ASCII format).

Key steps:

1. Obtain a **handle** to the file.
2. Process the file by **reading** or **writing**.
3. **Close** the file.

File Example

A text file consists of a sequence of lines, like so:

```
// file1.txt
radar
rotor
madam
```

Or a series of **records**:

```
123, Harry, Jones, 2007
129, Ben, Davies, 2008
130, Mai Ling, Li, 2007
```

This format is also known as **Comma Separated Values** (CSV).

Records

A **record** contains multiple fields, which can be of different data types:

AQA Example:

```
type student = record
    studentId : integer
    first_name : string[15]
    last_name : string[25]
    year_born : integer
end;
```

Python Alternative:

```
student = {
    'studentId': 123,
    'first_name': 'Harry',
    'last_name': 'Jones',
    'year_born': 2007
}
```

Reading a Text File

To read a file `students.txt` containing records:

```
student_file = openRead('students.txt')
while not student_file.endOfFile()
    line = student_file.readLine()
    print(line)
endwhile
student_file.close()
```

Modes:

- **read**: Opens the file for reading.
- **write**: Opens the file for writing (overwrites existing content).
- **append**: Adds data to the end of the file.

Writing to a Text File

Example of writing data to a file:

```
student_file = openWrite('students.txt')
id = int(input("Enter student id: "))
first_name = input("Enter first name: ")
last_name = input("Enter last name: ")
year_of_birth = int(input("Enter year of birth: "))
student_file.writeLine(id + ", " + first_name + ", " + last_name + ", " + year_of_birth + "\n")
student_file.close()
```

> **Note**: The `\n` ensures new records are written on new lines.

File Closing

Always close a file after processing it:

```
student_file.close()
```

Closing a file:

- Frees up memory.
- Ensures the file is properly saved and can be accessed by other processes.

Pseudocode for File Handling

OCR pseudocode includes the following file handling functions:

Function	Description	Example
`openRead(file)`	Opens a file for reading	`file = openRead('students.txt')`
`readLine()`	Reads a line from the file	`line = file.readLine()`
`endOfFile()`	Checks if end of file is reached	`while not file.endOfFile()`
`openWrite(file)`	Opens a file for writing	`file = openWrite('students.txt')`