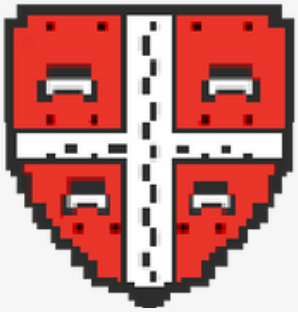

r/IGCSE Resources

Basic Programming Syntax for Cambridge IGCSE

Computer Science (0478)

Authored by

rud and puppy

1st edition, for examination until 2028

Information

This booklet explains the structures of Pseudocode and Python with clear examples to help the user understand the basics of programming using these programming languages.

All information provided in this document is aligned inclusively with the ***Cambridge IGCSE™*** and ***Cambridge O Level*** Computer Science syllabus covering examinations from 2023 to 2028.

For additional information or queries, please contact @neekh on Discord.

Contents

1 Comments 3

2 Data Types 4

3 Declarations 5
- 3.1 Variable Declaration . . . 5
- 3.2 Constant Declaration . . . 5
- 3.3 Assignment . . . 6

4 Functions 7
- 4.1 Input and Output . . . 7
- 4.2 Other Library Functions . . . 8

5 Operations 9
- 5.1 Arithmetic Operations . . . 9
- 5.2 Logical Operations . . . 10
- 5.3 String Operations . . . 10

6 Iterations and Loops 12
- 6.1 Count-controlled `FOR` Loop . . . 12
- 6.2 Post-conditional `REPEAT` Loop . . . 13
- 6.3 Pre-conditional `WHILE` Loop . . . 14

7 Selections 15
- 7.1 `IF` statements . . . 15
- 7.2 `CASE` statements . . . 16

8 Arrays 18
- 8.1 1D Arrays . . . 18
- 8.2 2D Arrays . . . 19

9 File Handling 20
- 9.1 Pseudocode . . . 20
- 9.2 Python . . . 21

10 Procedures and Functions 22
- 10.1 Procedures . . . 22
- 10.2 Functions . . . 23

Comments

Comments in code act as a form of note that programmers might add to explain sections of programs or other forms of logic. They are ignored by the compiler when the program is being executed. These comments help make the code maintainable and manageable both for the program author and other programmers.

Comments can be generalized into three forms, these are as follows:

- Multi-line comments
- Single-line comments
- Inline comments

In Pseudocode, all forms of comments are preceded with `//`.

Pseudocode

Multi-line comments

```
// This is an example of a multi-line comment
// It is normally used for larger explanations
```

Single-line comments

```
// This is an example of a single-line comment
```

Inline comments

```
OUTPUT Result // This is an example of an inline comment
```

In Python, multi-line comments are delimited by `'''` and, single-line and inline comments are preceded with `#`.

Python

Multi-line comments

```
'''
This is an example of a multi-line comment
It is normally used for larger explanations
'''
```

Single-line comments

```
# This is an example of a single-line comment
```

Inline comments

```
print(result) # This is an example of an inline comment
```

Data Types

In programming, there are a few keywords used to designate certain data types, these are as follows:

Pseudocode	Python
`INTEGER`	`int`
`REAL`	`float`
`CHAR`	`str` with single character
`STRING`	`str`
`BOOLEAN`	`bool`

Literals

Literals of these data types are written as follows:

`INTEGER`	written as normal in the denary numbering system, e.g: `5`, `-3`, `0`
`REAL`	written with at least one digit on either side of the decimal point (`0`s being added if necessary), e.g. `3.14`, `-4.5`, `0.0`
`CHAR`	written as a single character delimited by single quotes, e.g. `'x'`, `'@'`. In Python, characters are written as a string with a single character.
`STRING`	delimited by double quotes. Strings may be empty, e.g. `"An orange cat"`, `" "`.
`BOOLEAN`	written as either `TRUE` or `FALSE`.

Declarations

In programming, a declaration is a statement that introduces an element to a program. It provides the compiler with explicit information about the element before it is used. Additionally, by using declarations, programmers enhance the accessibility and maintainability of their code.

In Pseudocode, there are two types of declarations, these are as follows:

- Variable declaration
- Constant declaration

Variable Declaration

Variables are declared in the following format:

```
DECLARE <identifier> : <data type>
```

Pseudocode

Declaration of the `INTEGER` data type

```
DECLARE NumTeams : INTEGER
```

Declaration of the `REAL` data type

```
DECLARE Score : REAL
```

Declaration of the `CHAR` data type

```
DECLARE Choice : CHAR
```

Declaration of the `STRING` data type

```
DECLARE Feedback : STRING
```

Declaration of the `BOOLEAN` data type

```
DECLARE Found : BOOLEAN
```

Constant Declaration

Constants are declared by stating the identifier and the literal value in the following format:

```
CONSTANT <identifier> ← <value>
```

Pseudocode

Constant declaration of integer

```
CONSTANT UpperBound ← 15
```

Constant declaration of decimal number

```
CONSTANT MaxTemp ← 123.4
```

Assignment

In Pseudocode, the assignment operator is ←

Assignment statements are written in the following format:

```
<identifier> ← <value>
```

Pseudocode

Assigning a literal value to a variable

```
MaxAttempt ← 4
```

Assigning an expression to a variable

```
ActualScore ← Score * 100
```

Assigning another variable to a variable

```
CurrentTemp ← Temp
```

In Python, the assignment operator is =

Assignment statements are written in the following format:

```
<identifier> = <value>
```

Python

Assigning a literal value to a variable

```
max_attempt = 8
```

Assigning a function to a variable

```
answer = input()
```

Assigning an expression to a variable

```
average_score = score / 8
```

Assigning another variable to a variable

```
current_score = score
```

Functions

In programming there are a few built-in library functions that may be provided.

Input and Output

In Pseudocode, values can be input as follows:

```
INPUT <identifier>
```

Values can be output as follows:

```
OUTPUT <value(s)>
```

Several values, separated by commas, can be output using the same function.

Pseudocode

Input function

```
INPUT Answer
```

Output function

```
OUTPUT Score
```

Output function outputting multiple values

```
OUTPUT AvgScore, TotalScore, HighestScore
```

In Python, values can be input as follows:

```
<identifier> = input(<prompt>)
```

Prompts are optional. They must be literals of the `STRING` data type.

Values can be output as follows:

```
print(<value(s)>)
```

Several values, separated by commas, can be output using the same function.

Python

Input function

```
answer = input()
```

Input function with prompt

```
num_players = input("Enter the number of players: ")
```

Output function

```
print(score)
```

Output function outputting multiple values

```
print(avg_score, total_score, highest_score)
```

Other Library Functions

In Pseudocode the following library functions are practiced:

`ROUND(<identifier>, <places>)`

Returns the value of the `identifier` rounded to `places` number of decimal places. The identifier should be any value that evaluates to the `REAL` data type whilst the number of places should be a positive integer.

`RANDOM()`

Returns a random decimal number between 0 and 1 inclusive.

Pseudocode

Rounding function
```
ROUND(3.1415, 1)
```
Returns `3.1`

Random function
```
RANDOM() * 5
```
Returns a random decimal number between 0 and 5 inclusive

```
ROUND(RANDOM() * 10), 0)
```
Returns a random integer between 0 and 10 inclusive

In Python these library functions can be written as:

`round(<identifier>, <places>)`

Returns the value of the `identifier` rounded to `places` number of decimal places. The identifier should be any value that evaluates to the `REAL` data type whilst the number of places should be a positive integer.

`random()`

Returns a random decimal number between 0 and 1 inclusive.

Python

Rounding function
```
round(3.1415, 1)
```
Returns `3.1`

Random function
```
random() * 5
```
Returns a random decimal number between 0 and 5 inclusive

```
round(random() * 10), 0)
```
Returns a random integer between 0 and 10 inclusive

Operations

In programming, there are three types of operations, these are as follows:

- Arithmetic operations
- Logical operations
- String operations

Arithmetic Operations

For arithmetic operations, standard mathematical operator symbols are used. These operator symbols are as follows:

	Pseudocode	Python
Addition	+	+
Subtraction	-	-
Multiplication	*	*
Division	/	/
Raised to the power of	^	**

Pseudocode

Calculating the area of a circle

```
Area ← Pi * Radius ^ 2
```

Integer division operations may also be used for calculations.

In Pseudocode, these integer division operations are written as follows:

`DIV(<dividend>, <divisor>)`

Returns the quotient of `dividend` divided by the `divisor` with the fractional part discarded.

`MOD(<dividend>, <divisor>)`

Returns the remainder of `dividend` divided by the `divisor`.

Pseudocode

`DIV` integer division operation

```
DIV(10, 3)
```

Returns `3`

`MOD` integer division operation

```
MOD(10, 3)
```

Returns `1`

In Python, these integer division operations are written as follows:

`<dividend> // <divisor>`

Returns the quotient of `dividend` divided by the `divisor` with the fractional part discarded.

`<dividend> % <divisor>`

Returns the remainder of `dividend` divided by the `divisor`.

Python

`DIV` integer division operation
`10 // 3`
Returns `3`

`MOD` integer division operation
`10 % 3`
Returns `1`

Logical Operations

The following symbols are used for logical operations:

	Pseudocode	Python
Equal to	=	==
Less than	<	<
Less than or equal to	<=	<=
Greater than	>	>
Greater than or equal to	>=	>=
Not equal to	<>	!=

These operations evaluate expressions to `BOOLEAN` data type values.

String Operations

String operations are used to manipulate values of the `STRING` data type.

In Pseudocode, the basic string operations are written as follows:

`LENGTH(<identifier>)`

Returns the integer value representing the length of the string.

`LCASE(<identifier>)`

Returns the string/character with all characters in lower case.

`UCASE(<identifier>)`

Returns the string/character with all characters in upper case.

`SUBSTRING(<identifier>, <start>, <length>)`

Returns a string of the length `length` starting at the position `start`. The `length` and `start` should be a positive integer.

Pseudocode

Length string operation
`LENGTH("Good evening")`
Returns `12`

Lower case string operation
`LCASE('H')`
Returns `'h'`

Upper case string operation
`UCASE("A blue bird")`
Returns `"A BLUE BIRD"`

Substring string operation
`SUBSTRING("Happy Days", 1, 5)`
Returns `"Happy"`

In Python, the basic string operations are written as follows:

`len(<identifier>)`

Returns the integer value representing the length of the string.

`(<identifier>).lower()`

Returns the string/character with all characters in lower case.

`(<identifier>).upper()`

Returns the string/character with all characters in upper case.

`<identifier>[<start>:<end>]`

Returns a string that starts from the position `start` and ends at the position `end` exclusive

Python

Length string operation
`len("Rud is the best")`
Returns `15`

Lower case string operation
`'X'.lower()`
Returns `'x'`

Upper case string operation
`"The orange painting".upper()`
Returns `"THE ORANGE PAINTING"`

Substring string operation
`"Hello Rud"[6:9]`
Returns `"Rud"`

Iterations and Loops

In programming, there are three types of iterations. These are as follows:

- Count-controlled `FOR` loop
- Post-conditional `REPEAT` loop
- Pre-conditional `WHILE` loop

Count-controlled `FOR` Loop

In count-controlled loop, the identifier must be a variable of the `INTEGER` data type.

For Pseudocode, count-controlled loops are written in the following structure:

```
FOR <identifier> ← <start> TO <finish>
    <statements>
NEXT <identifier>
```

The identifier is assigned each of the integer values from `start` to `finish` inclusive, executing the statements inside the loop after each assignment.

If `start` is equal to `finish`, the statements will only be executed once and if `start` is greater than `finish`, the statements will not be executed at all.

An increment can be specified as follows:

```
FOR <identifier> ← <start> TO <finish> STEP <increment>
    <statements>
NEXT <identifier>
```

The increment must be an expression that evaluates to an integer. The `identifier` will be assigned the values from `start` in successive increments of `increment` until it reaches `finish`. If it goes past `finish`, the loop will be terminated. `increment` can be negative.

Pseudocode

Count-controlled loop

```
FOR Index ← 1 TO 30
    OUTPUT Numbers[Index]
NEXT Index
```

Nested count-controlled loop

```
FOR Row ← 5 TO 10
    FOR Column ← 1 TO 3
        OUTPUT Value[Row, Column]
    NEXT Column
NEXT Row
```

Count-controlled loop with specified increment

```
FOR Index ← 1 TO 30 STEP 5
    OUTPUT Numbers[Index]
NEXT Index
```

In Python, count-controlled loops are written as follows:

```
for <identifier> in range(<start>, <finish>):
    <statements>
```

The range function creates a list from `start` to `finish` exclusively. If the `start` is not specified it will be set to the default `start` position, 0.

An increment can be specified as follows:

```
for <identifier> in range(<start>, <finish>, <increment>):
    <statements>
```

The increment must be an expression that evaluates to an integer. The identifier will be assigned the values from `start` in successive increments of `increment` until it reaches `finish`. If it goes past `finish`, the loop will be terminated. `increment` can be negative. `start` is mandatory to specify any step `increment`.

Python

Count-controlled loop

```
for index in range(30):
    print(numbers[index])
```

Nested count-controlled loop

```
for row in range(5, 10):
    for column in range(3):
        print(numbers[row][column])
```

Count-controlled loop with specified increment

```
for index in range(1, 30, 5):
    print(numbers[index])
```

Post-conditional `REPEAT` Loop

In post-conditional loop, the condition must be an expression that evaluates to an `BOOLEAN` data type.

The statements in the post-conditional loop will be executed at least once. The condition is tested after the statements are executed and if the condition evaluates to `TRUE` the loop is then terminated, otherwise the process is repeated again.

In Pseudocode, post-conditional loops are written as follows:

```
REPEAT
    <statements>
UNTIL <conditions>
```

Pseudocode

Post-conditional loop

```
REPEAT
    INPUT InpPassword
UNTIL InpPassword = Password
```

Pre-conditional `WHILE` Loop

In pre-conditional loop, the condition must be an expression that evaluates to an `BOOLEAN` data type.

The condition is tested before the statements are executed, and the statements will only be executed if the condition evaluates to `TRUE`. After the conditions have been executed the conditions are tested again. The loop terminates when the condition evaluates to `FALSE`.

In Pseudocode, pre-conditional loops are written as follows:

```
WHILE <conditions> DO
    <statements>
ENDWHILE
```

Pseudocode

Pre-conditional loop

```
WHILE Number < 10 DO
    OUTPUT Number
    Number ← Number + 1
ENDWHILE
```

In Python, pre-conditional loops are written as follows:

```
while <conditions>:
    <statements>
```

Python

Pre-conditional loop

```
while number < 10:
    print(number)
    number = number + 1
```

Selections

There are two functions of selections this booklet practices, these are as follows:

- `IF` statements
- `CASE` statements

`IF` statements

In Pseudocode, an `IF` statement without an `ELSE` clause is written as follows:

```
IF <conditions>
    THEN
        <statements>
ENDIF
```

`IF` statement with an `ELSE` clause is written as follows:

```
IF <conditions>
    THEN
        <statements>
    ELSE
        <statements>
ENDIF
```

Pseudocode

`IF` statement without `ELSE` clause

```
IF Answer = CorrectAnswer
    THEN
        Score ← Score + 1
ENDIF
```

`IF` statement with `ELSE` clause

```
IF Answer = CorrectAnswer
    THEN
        Score ← Score + 1
    ELSE
        OUTPUT "Wrong Answer!"
ENDIF
```

In Python, an `IF` statement without any clauses is written as follows:

```
if <condition>:
    <statements>
```

An `IF` statement with an `elif` clause is written as follows:

```
if <condition>:
    <statements>
elif <condition>:
    <statements>
```

There can be multiple `elif` clauses in a `IF` selection statement. Please note that `elif` clauses do not completely align with the syllabus, candidates should use it with caution.

An IF statement with an ELSE clause is written as follows:

```
if <condition>:
    <statements>
else:
    <statements>
```

Python

IF statement with no clause

```
if valid:
    print("Permission granted")
```

IF statement with an elif clause

```
if red_score > blue_score:
    print("Red team have won!")
elif red_score == blue_score:
    print("Tie!")
```

IF statement with an ELSE clause

```
if found:
    score = score + 1
else:
    print("Failed!")
```

CASE statements

In Pseudocode, CASE statements allow one out of several branches of code to be executed depending on the value of the variable.

CASE statements are written as follows:

```
CASE OF <identifier>
    <value n> : <statements>
ENDCASE
```

An OTHERWISE clause can be added as follows:

```
CASE OF <identifiers>
    <value n> : <statements>
    OTHERWISE <statements>
ENDCASE
```

These clauses are tested in a sequence. When a value that applies is found, it's statement is executed and the CASE statement is completed. If present, an OTHERWISE clause must be the last case. It's statement will be executed if none of the preceding cases apply.

Pseudocode

CASE statement

```
CASE OF Choice
    1 : OUTPUT "Birch wood selected"
    2 : OUTPUT "Spruce wood selected"
    3 : OUTPUT "Fir wood selected"
ENDCASE
```

CASE statement with an OTHERWISE clause

```
CASE OF Movement
    'w' : ycord ← ycord + 1
    'a' : xcord ← xcord - 1
    's' : ycord ← ycord - 1
    'd' : xcord ← xcord + 1
    OTHERWISE OUTPUT "Invalid choice!"
ENDCASE
```

In Python, CASE statements are written as follows:

```
match <identifier>:
    case <value n>:
        <statements>
```

An OTHERWISE clause (indicated by _) can be added as follows:

```
match <identifiers>:
    case <value n>:
        <statements>
    case _:
        <statements>
```

Python

CASE statement with an OTHERWISE clause

```
match movement:
    case 'w':
        ycord = ycord + 1
    case 'a':
        xcord = xcord - 1
    case 's':
        ycord = ycord - 1
    case 'd':
        xcord = xcord + 1
    case _:
        print("Invalid Choice!")
```

Arrays

Arrays are fixed-length structures of elements that have identical data type, accessible by consecutive index numbers. Square brackets, `[<index>]` are used to indicate the array indices.

In programming, arrays can have multiple dimensions, however this booklet only practices 1D and 2D arrays.

1D Arrays

In Pseudocode, 1D arrays are declared as follows:

```
DECLARE <identifier> : ARRAY[<l>:<u>] OF <data type>
```

Where `l` stands for lower bound and `u` stands for upper bound.

1D arrays are assigned in the following way:

```
<identifier>[<index>] ← <value>
```

Pseudocode

1D array declaration

```
DECLARE StudentName : ARRAY[1:30] OF STRING
```

1D array assignment

```
StudentName[19] ← "John Doe"
```

In Python, 1D arrays are initialized as follows:

```
<identifier> = [None] * <u>
<identifier> = [0] * <u>
```

Where `u` stands for upper bound.

1D arrays are assigned in the following way:

```
<identifier>[<index>] = <value>
```

Python

1D array initialization

```
student_marks = [0] * 30
```

1D array assignment

```
student_marks[15] = 88
```

2D Arrays

In Pseudocode, 2D arrays are declared as follows:

```
DECLARE <identifier> : ARRAY[<lr>:<ur>, <lc>:<uc>] OF <data type>
```

Where `r` stands for row, `c` stands for column, `l` stands for lower bound and `u` stands for upper bound.

2D arrays are assigned as follows:

```
<identifier>[<ri>, <ci>] ← <value>
```

Where `ri` stands for row index and `ci` stands for column index.

Pseudocode

2D array declaration

```
DECLARE Grade : ARRAY[1:30, 1:5] OF CHAR
```

2D array assignment

```
Grade[16, 3] ← 'A'
```

In Python, 2D arrays are initialized as follows:

```
<identifier> = [[None] * <ur>] * <uc>
<identifier> = [[0] * <ur>] * <uc>
```

Where `r` stands for row, `c` stands for column and `u` stands for upper bound.

2D arrays are assigned as follows:

```
<identifier>[<ri>][<ci>] ← <value>
```

Where `ri` stands for row index and `ci` stands for column index.

Python

2D array initialization

```
student_attendance = [[None] * 10] * 5
```

2D array assignment

```
student_attendance[8][3] = True
```

File Handling

Pseudocode

File Opening

When opening a file, the mode of operation of the file should be stated as follows:

```
OPENFILE <file identifier> FOR <file mode>
```

The file identifier will be the name of the file. The following file modes can be used:

`READ`	for data to be read from the file
`WRITE`	for data to be written to a file. In case the file does not exist, a new file will be created. Existing data is overridden.

A file can only be opened in one mode at a time.

Reading from File

Data is read from the file (after the file has been opened in the `READ` mode) as follows:

```
READFILE <file identifier>, <identifier>
```

When executed, a line of text is read from the file and assigned to the identifier.

Writing from File

Data is written to a file (after the file has been opened in `WRITE` mode) as follows:

```
WRITE <file identifier>, <identifier>
```

When executed, the value from the identifier is assigned to the file.

File Closing

When a file is no longer in use, it is closed as follows:

```
CLOSEFILE <file identifier>
```

Pseudocode

Opening file in read mode
```
OPENFILE "Names.txt" FOR READ
```

Opening file in write mode
```
OPENFILE "Remarks.txt" FOR WRITE
```

Reading from a file
```
READFILE "Names.txt", Data
```

Writing to a file
```
WRITEFILE "Remarks.txt", Remark
```

Closing file
```
CLOSEFILE "Names.txt"
```

Python

File Opening

When opening a file, the mode of operation of the file should be stated as follows:

```
<operator> = open(<file identifier>, <file mode>)
```

The operator is used as a reference to the file opened. The file identifier will be the name of the file. The following file modes are used:

- `r` for data to be read from the file
- `w` for data to be written to a file. In case the file does not exist, a new file will be created. Existing data is overridden.
- `a` for data to be appended to a file. Existing data will not be overridden

A file can only be opened in one mode at a time.

Reading from File

Data is read from the file (after the file has been opened in the `r` mode) as follows:

```
<identifier> = <operator>.read()
```

When executed, a line of text is read from the file and assigned to the identifier.

Writing from File

Data is written to a file (after the file has been opened in the `w` or `a` mode) as follows:

```
<operator>.write(<identifier>)
```

When executed, the value from the identifier is assigned to the file.

File Closing

When a file is no longer in use, it is closed as follows:

```
<operator>.close()
```

Python

Opening file in read mode

```
file_a = open("Names.txt", "r")
```

Opening file in append mode

```
file_b = open("StudentRegister.txt", "a")
```

Reading from a file

```
data = file_a.read()
```

Writing to a file

```
file_b.write("Jane Doe")
```

Closing file

```
file_a.close()
```

Procedures and Functions

Procedures and functions are always defined at the top of the program.

Procedures

In Pseudocode, a procedure with no parameters is defined as follows:

```
PROCEDURE <identifier>
    <statements>
ENDPROCEDURE
```

A procedure with parameters is defined as follows:

```
PROCEDURE <identifier>(<par n> : <data type>)
    <statements>
ENDPROCEDURE
```

When used, `par n` is the identifier for the parameters of the procedure. These will be used as variables in the statements of the procedure.

Procedures should be called as follows:

```
CALL <identifier>
CALL <identifier>(<val n>)
```

When a procedure is called, if any parameters are present, they are substituted by the values, and the statements are executed.

Pseudocode

Defining procedure with no parameter

```
PROCEDURE DisplayError
    OUTPUT "Invalid response! Try again"
ENDPROCEDURE
```

Defining procedure with parameter

```
PROCEDURE CalculateScore(RawScore:INTEGER)
    FullScore ← (RawScore/60) * 100
    HalfScore ← (RawScore/60) * 50
    OUTPUT FullScore, HalfScore
ENDPROCEDURE
```

Calling procedure with no parameter

```
CALL DisplayError
```

Calling procedure with parameter

```
CALL CalculateScore(48)
```

Returns `80` and `40`

Functions

In Pseudocode, functions behave the same way as procedures except it only returns one singular value. During definition the data type of the value returned has to be stated.

A function with no parameter is defined as follows:

```
FUNCTION <identifier> RETURNS <data type>
    <statements>
ENDFUNCTION
```

A function with parameter is defined as follows:

```
FUNCTION <identifier>(<par n>:<data type>) RETURNS <data type>
    <statements>
ENDFUNCTION
```

Function calls are not a complete program statement; the keyword `CALL` should not be used when calling a function. Instead, functions should only be called as part of an expression.

Pseudocode

Defining function

```
FUNCTION SumSquare(Num1:INTEGER, Num2:INTEGER) RETURNS INTEGER
    RETURN Num1 ^ 2 + Num2 ^ 2
ENDFUNCTION
```

Using function

```
OUTPUT "Sum of squares: ", SumSquare(5, 10)
```

In Python, functions behave the same way as procedures.

Function without parameters are defined as follows:

```
def <identifier>():
    <statements>
```

Function with parameters are defined as follows:

```
def <identifier>(<par n>):
    <statements>
```

When used, `par n` is the identifier for the parameters of the function. These will be used as variables in the statements of the function.

Function calls are not a complete program statement. Instead, functions should only be called as a part of an expression.

Python

Defining function

```
def sum_square(num_1, num_2):
    print(num_1 ** 2 + num_2 ** 2)
```

Using function

```
print("Sum of squares", sum_square(5, 10))
```

r/IGCSE Resources

r/IGCSE Resources repository | r/IGCSE subreddit | Our Discord server

igcse.reddit.com

discord.gg/igcse

Acknowledgments and Information:

The information on this guide has been generously prepared by current students/alumni; the authors have been acknowledged where possible. Links to external sites embedded in this document are not affiliated with or operated by r/IGCSE.

This guide is meant to be for educational purposes only and is to remain free of cost for the benefit of all students.

For any suggested changes or corrections to this document, please contact the r/IGCSE staff team via email or on the r/IGCSE Discord server.