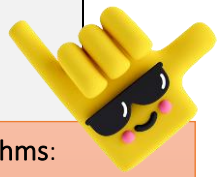
Pseudocode

```
BEGIN
    INPUT num1
    IF num1 > 0 THEN
        OUTPUT "POSITIVE ALREADY"
    ELSE
        WHILE num1 < 1
            CALCULATE num1 = num1 + 1
        ENDWHILE
        OUTPUT "NOW IT'S POSITIVE!"
    ENDIF
    FOR counter = 1 TO 5
        OUTPUT "hi"
    NEXT counter
    CALL module1(5)
END
```

Pseudocode to Python ↓

```
#BEGIN
num1 = int(input("num1: "))
if num1 > 0: #THEN
    print("POSITIVE ALREADY")
else:
    while num1 < 1:
        num1 = num1 + 1
    #ENDWHILE
    print("NOW IT'S POSITIVE!")
#ENDIF
for counter in range(1,5):
    print("hi")
#NEXT counter
module1(5)
#END
```



To solve or desk check algorithms:

1) Use your left index finger to point to the line of pseudocode you are executing.
2) Draw up a table to track values of variables. Add any new variables to a new column. Make value changes down rows.

Pseudocode

```
BEGIN module1(x)
    IF x >= 2 THEN
        REPEAT
            OUTPUT x
            CALCULATE x = x - 1
        UNTIL x == 1
    ENDIF
END module1
```

Pseudocode to Python →

```
def module1(x):
    if x >= 2: #THEN
        while True:
            print(x)
            x = x - 1
            if x == 1:
                break
    #ENDIF
#END module1
```

Python does not implement a REPEAT UNTIL loop. Post-test loops are arguably *poor programming practise*, as you are placing the user inside a loop, potentially forever, without testing a condition first.

For all algorithms:

- Use *consistent* wording. The following words could be substituted for each other depending on *context*, but once you pick a keyword or way of working, stick to it:
  - `INPUT`, `GET`, or `READ`
  - `OUTPUT`, `PRINT`, `DISPLAY`, or `WRITE` (possibly)
  - `SET`, `CALCULATE`, `ASSIGN` or `SET VARIABLE` (possibly)
  - `y = y + 1` can be OK too if it is obvious without a pseudocode keyword
- Selection and Repetition are both controlled by *test conditions*, which return a Boolean value (TRUE or FALSE). Loops will not execute once the value of a test condition has returned FALSE. An IF statement(s) will only execute if the value of the test condition is TRUE, otherwise the ELSE statement(s) will be executed (if there are any).
- **Selection** uses branching. Finish selection with `ENDIF` and maintain block indenting as you would with any Python program (but don't use `elif` – instead, nest `IF` statements within `IF` statements). If in doubt, treat it like Python.
- **Repetition** (loops) uses iteration. Maintain block indenting, and use the loop that best fits your cause:
  - Post-test loops (the loop condition is tested after the execution of the loop) use REPEAT UNTIL. These loops will execute a minimum of once. *These may be considered poor practise*.
  - Pre-test loops (the loop condition is tested before the execution of the loop) use WHILE ENDWHILE. These loops may not execute at all (minimum of 0) as the condition is tested first.
  - Counted loops use FOR NEXT
- **Modularisation** involves using `modules` of code to break up tasks into smaller, more manageable pieces. Other words for `modules` include `procedures`, `methods`, `subroutines` or `functions`.
- Modules may return values. Single equals = assigns a value. Double equals == compares values.

num1	counter	
-1	1	
0	2	
1	3	
	4	
	5	