Successful Pseudocoding
Edition 2.01 *alpha*

https://codeforschools.com/textbook

Table of Contents

Notes to Edition 2.1 *alpha*

Chapters 6 and 7 have been added which covers functions and lists.

Added a URL and QR code to the front page.

As a result, this resource now provides a complete course of tools and techniques for students to analyse and design mathematical algorithms required at sefnior secondary level.

Still to do:

- Deeper and clearer elaborations of concepts
- Links to animated / video walkthroughs of trace tables
- Fix errors in questions and solutions (of which I'm sure there are many!)

Any questions, comments, feedback, or errors please let the author know via toan@csinschools.com

Chapter 1 - Introduction to Pseudocode

An algorithm is the set of steps to solve a logical or mathematical problem.

Pseudocode is a structured way of representing instructions for humans to follow the algorithm.

Algorithms are generally written in pseudocode.

Like coding languages which you may have come across previously (Scratch, Python, Java, C++ etc.) pseudocode does have a set of rules to follow with regard to how to use its symbols (operators) and how to represent numbers etc. However, unlike these coding languages, the rules are more relaxed and pseudocode is designed for humans to read and understand.

Pseudocode also provides a common format for us humans to share algorithms between ourselves. For instance, when one person only knows how to program in Python and another person only knows how to program in Javascript, pseudocode provides a relaxed “common ground” for us both.

If we would like to have a computer run our algorithm, we will need to translate pseudocode into actual program code by using a particular programming language. Appendix A covers briefly how to convert the algorithms in this book into the Python programming language - so you can see these algorithms “come to life” on the computer.

Chapter 2 - Variables

2A - Variables

A variable is named location in the computer's memory which we use to store data values. We use variables to represent unknown quantities, represent quantities which change (vary-able) and to generally keep track of values.

To store a value inside a variable we use the assignment operator, ⟵.

variable_name ⟵ value

Whenever we see an ⟵ sign, a variable **must** be on the left hand side (because we're storing a value into it).

We may also see variables on the right hand side too (see later section).

We read this as "value is assigned to variable_name".
For example, for the instruction:

x ⟵ 10

We read this as "10 is assigned to x" or "x is assigned the value of 10"

Any valid mathematical expression can be found on the right hand side of the assignment operator ⟵. Any arithmetic expressions are evaluated using standard B(O/I)DMAS rules.

Examples:
Which of these are valid assignment statements?

	Assignment Statement	Valid?
a.	`c ← 42`	True
b.	`10 ← c`	False (the left hand side must be a variable)
c.	14 → `x`	False (the assignment operator is in the wrong direction)
d.	`y ← (10 + 2) × 3`	True
e.	`(20 - 8) / 3 → d`	False (the assignment operator is in the wrong direction)
g.	`x + 3`	False (there is no assignment operator - the value in the x variable does not change)
h.	`area ← 42`	True

Exercises:
Which of these are valid assignment statements?

	Assignment Statement	Valid?
1.	`d ← 0`	
2.	`x ← 30.0 - 2`	
3.	`0 → x`	
4.	`(20 - 8) / 3 → y`	
5.	`x + 2 ← 4`	
6.	`z + x ← (10 + 2) + 3`	
7.	`g ← ((10 / 2) × (3 + 5)) - 3.3`	
8.	`a / 3`	
9.	`x + z + 3`	
10.	`perimeter ← 2 * 10 + 2 * 5`	

2B - Variables on the Right Hand side of ←

When a variable is found on the right hand side of the ← , this means that the value currently assigned to the variable is **substituted** for that variable in the expression.

Variable on the right hand side: value is **read** from the variable.
Variable on the left hand side: value is **written** to the variable.

Any variable found on the right hand side **does not** change its value.
If we wish to update the value in a variable we must put it on the left hand side.

When there are two or more pseudocode instructions, they are executed in order, one line at a time, from top to bottom.

Any variable found on the right hand side of the ← , **must** already have a value assigned to it otherwise the pseudocode is invalid.
E.g.

```
x ← y + 3
```

The above pseudocode is invalid there has been no value assigned to the y variable.

```
y ← 3

x ← y + 3
```

The above pseudocode is now valid (read from top to bottom).

```
y ← x - 3

x ← y + 3
```

The above pseudocode is invalid because in the very first line, x does not have a value assigned to it (read from top to bottom).

```
x ← 0

y ← x - 3

x ← y + 3
```

The above pseudocode is now valid (read from top to bottom).

Examples:
What is the final value of x and y in each of the following pseudocode programs:

	Assignment Statement	Final values of x and y
a.	`y ← 42` `x ← 21`	x: 21 y: 42
b.	`y ← 42` `x ← y + 2`	x: 44 y: 42
c.	`y ← 10 + 2` `x ← y / 2`	x: 6 y: 12

Exercises:

What is the final value of x and y in each of the following pseudocode programs:

	Assignment Statement	Final values of x and y
1.	`y ← -3` `x ← -6`	
2.	`y ← 0.4` `x ← y / 2`	
3.	`y ← -3 + 2` `x ← y + 2`	

2C - Trace Tables

Trace tables are an extremely useful tool for keeping track of variable values throughout the execution of pseudocode. Once you learn how to use trace tables effectively, you will be able to deconstruct algorithms and see how they work.

The trace table below consists of 2 columns, one listing the variable names, and the other containing the values found within the corresponding variable.

Variable	**Value**
x	
y	

To use a trace table with pseudocode, we step through the pseudocode, one line at a time, and record any changes to variables and their values in the tracetable. It is important to keep the trace table updated with **every line** of pseudocode!

Here's a step-by-step example:

<table>
<tr>
<td>

```
y ← 0
y ← 3
x ← y - 2
y ← x
x ← y + 5
```

</td>
<td></td>
</tr>
<tr>
<td>

```
y ← 0
y ← 3
x ← y - 2
y ← x
x ← y + 5
```

</td>
<td>

Variable	Value
y	0

</td>
</tr>
<tr>
<td>

```
y ← 0
y ← 3
x ← y - 2
y ← x
x ← y + 5
```

</td>
<td>

Variable	Value
y	~~0,~~ 3

</td>
</tr>
<tr>
<td>

```
y ← 0
y ← 3
x ← y - 2
y ← x
x ← y + 5
```

</td>
<td>

Variable	Value
y	~~0,~~ 3
x	1

</td>
</tr>
</table>

```
y ← 0
y ← 3
x ← y - 2
y ← x
x ← y + 5
```

Variable	Value
y	~~0, 3,~~ 1
x	1

```
y ← 0
y ← 3
x ← y - 2
y ← x
x ← y + 5
```

Variable	Value
y	~~0, 3,~~ 1
x	~~1,~~ 6

Examples:
Using trace-tables, execute the following pseudocode and determine the final values of all the variables:

a.

Pseudocode	Trace-table

```
a ← 1
b ← a - 3
a ← b + 1
```

Variable	Value
a	~~1,~~ -1
b	-2

b.

```
x ← 3 - 2
y ← x / 2
z ← x + y
x ← z / y
```

Variable	Value
x	~~1,~~ 3
y	0.5
z	1.5

c.

```
x ← -3
y ← x + x
x ← y / x
y ← x * x
```

Variable	Value
x	~~-3,~~ 2
y	~~-6,~~ 4

Exercises:
Using trace-tables, execute the following pseudocode and determine the final values of all the variables:

	Pseudocode	Trace-table

1.	`a ← 10`
	`b ← 20`
	`a ← b`
	`b ← a`	

2.	`x ← 10`
	`y ← 20`
	`z ← x`
	`x ← y`
	`y ← z`	

3.	`x ← 8`
	`y ← x - x / 2`
	`x ← y + x`
	`y ← x * x`	

4.	`x ← 1`
	`y ← 1`
	`z ← x + y`
	`x ← y + z`
	`y ← x + z`
	`z ← x + y`	

2D - Self-assignment

The same variable can be found on **both** sides of the assignment operator.

E.g.

```
x ← 2
x ← x + 1
```

When this occurs, as usual, we evaluate the right hand side of the assignment operator first and **read** from the variable, operate on it, and then assign it back to the variable.

Remember that any operations on the right hand side of the assignment statement does not change the value of any variables. The values are only **read** from the variables.

Examples:
Using trace-tables, execute the following pseudocode and determine the final values of all the variables:

a.

```
x ← 0
x ← x + 1
```

Variable	Value
x	~~0,~~ 1

b.

```
x ← 2 * 3
x ← x * 2
x ← x * x
```

Variable	Value
x	~~6, 12,~~ 144

c.

```
a ← -3
b ← a + a
a ← a / b
```

Variable	Value
a	~~-3,~~ 0.5
b	-6

Exercises:
Using trace-tables, execute the following pseudocode and determine the final values of all the variables:

	Pseudocode	Trace-table
1.	`a ← 10` `a ← a / 2`	

2.	`x ← 8`
	`y ← (x + 4) / 3`
	`y ← y * 3`
	`y ← y - 4`	

3.	`x ← 4`
	`y ← 8`
	`y ← x + y`
	`x ← y - x`
	`y ← y - x`	

2E - Writing Pseudocode

Examples:
Write the pseudocode to achieve the goal provided.

	Goal	Code

a.	• Assign the value 10 to the variable, a
	• Square the value of a and assign it to the variable b
		`a ← 10`
		
		`b ← a * a`

b.	• Assign the value 3 to the variable, r
	• Calculate the area of a circle with the radius given by the value in the variable r
	• Assign the result to the variable, a
		`r ← 3`
		
		`a ← 3.14159 * r * r`

c.	• Assign the value 2 to the variable l
	• Assign the value 3 to the variable w
	• Calculate the area of a rectangle with a length given by l, and a width given by w
	• Assign the result to the variable a
		`l ← 2`
		
		`w ← 3`
		
		`a ← l * w`

d.	• Assign the value -2 to the variable, x
	• Double the value of x and assign it back to x
		`x ← -2`
		
		`x ← x * 2`

Exercises:

Write the pseudocode to achieve the goal provided.

	Goal	Code
1.	● Assign the value -20 to the variable, a ● Halve the value of a and assign it to the variable b	
2.	● Assign the value 0.6 to the variable, d ● Calculate the area of a circle with the diameter given by the value in the variable d ● Assign the result to the variable, a	
3.	● Assign the value 2 to the variable l ● Assign the value 3 to the variable w ● Assign the value 4 to the variable d ● Calculate the volume of a cuboid with a length given by l, a width given by w, and a depth given by d ● Assign the result to the variable v	
4.	● Assign the value 10 to the variable, x ● Double the value of x and assign it back to x ● Triple this value and assign it back to x	
5.	● Assign the value 180 to the variable, h ● Assign the value 80 to the variable, w ● Calculate the BMI given with a height (in cms) given by h, and a weight (in kgs) given by w ○ BMI = kgs / m^2 ● Assign the answer to the variable, bmi	

Chapter 3 - Boolean Expressions

3A - Boolean Expressions

Boolean expressions are expressions which evaluate to either True or False.

The following operators are used to compare the left and right hand operands.
They will always evaluate to True or False.

- > is greater than
- < is less than
- >= is greater than or equal to
- <= is less than or equal to
- == is equal to
- != is not equal to

The **not** operator will negate (flip between True and False) whatever boolean expression follows it.

Examples:

	Expression	**Evaluation**
a.	1 < 3	True
b.	3 == 1 + 2	True
c.	True	True
d.	3 != 1	True
e.	not False	True
f.	not (2 == 2)	False

Exercises:

	Expression	Evaluation
1.	5 == 2	
2.	10 < 10	
3.	2 != 2	
4.	32 >= 32	
5.	(5 + 3) < 8	
6.	7 + 2 * 2 == (11 - 0)	
7.	True == False	
8.	4.1 >= 4.01	
9.	False	
10.	not True	
11.	not (1 <= 0)	

3B - Compound Boolean Expressions

We can combine two or more boolean expressions together using the **and** and **or** operators.

The **and** operator will evaluate to True only when both the LHS and RHS are True. Any other combination will evaluate to False.

The **or** operator will evaluate to True when either the LHS or RHS are True. If both LHS and RHS are False, it will evaluate to False.

Examples:
Evaluate the following combined boolean expressions.

	Expression	**Evaluation**
a.	1 == 1 and 2 == 2	True
b.	4 == -4 and 10 == 10	False
c.	10 < 8 and 10 > 1	False
d.	4 >= 4 and 3 > 2	True
e.	6 == 5 or 7 < 8	True
f.	10 > 3 or 4 == 3	True
g.	8 < 2 or 4 >1	True
h.	(9 > 1 and 7 < 10) or (40 > 100)	True
i.	True and False	False

Exercises:
Evaluate the following combined boolean expressions.

	Expression	Evaluation
1.	3 <=3 and 4 <= 4	
2.	8 > 0 or 1 < 0	
3.	10 == (9 + 1) and 3 < 5	
4.	1 > 0 and 6 > 2	
5.	50 < 34 or 34 > 43	
6	10 == 10 and 10 < 10	
7.	True and False	
8.	False or True	
9.	(10 > 5 and 4 > 1) or (50 >= 50)	
10.	True or False	
11.	True and True	
12.	False or False	

3C - Boolean Expressions involving Variables

We can read values from variables within boolean expressions and also assign boolean expressions to variables.

The same rules for assignment statements apply.

Examples:
Using trace-tables, execute the following pseudocode and determine the final values of all the variables:

a.

```
age ← 10

result ← age < 18
```

Variable	Value
age	10
result	True

b.

```
height ← 1.83

six_feet ← height >= 1.83
```

Variable	Value
height	1.83
six_feet	True

c.

```
f ← 32.0

c ← (f - 32.0) * 5/9

freezing ← c <= 0
```

Variable	Value
f	32.0
c	0
freezing	True

Exercises:
Using trace-tables, execute the following pseudocode and determine the final values of all the variables:

	Pseudocode	Trace-table
1.	`age ← 18` `result ← age < 18`	

2.	`date ← 31/01/2001`
	`return_by ← 28/02/2001`
	`late ← date > return_by`	

3.	`height ← 180`
	`height ← height / 100`
	`six_feet ← height >= 1.83`	

4.	`t ← 200.0`
	`t ← (t - 32.0) * 5/9`
	`boiling ← t >= 100`	

5.	`hours ← 150`
	`age ← 19`
	`test_passed ← True`
	`licensed ← hours > 120 and age > 18 and`
	`Test_passed`	

6.

```
distance ← 1500

distance ← distance / 1000

fast_food ← True

rating ← 4

order ← distance < 2.0 and rating >= 4
          and not fast_food
```

3D - Writing Pseudocode

Complete the following programming challenges.

Examples:

	Requirements	Pseudocode

a.	● A student obtained a score of 17 out of 20
	● Write pseudocode to:
	○ Convert this score to a percentage
	○ Assign a boolean value to the variable, A, indicating whether or not the percentage was greater than or equal to 80
		`score ← 17 / 20`
		
		`score ← score * 100`
		
		`A ← score >= 80`

b.	● A rectangle has a length of 7, and a width of 8
	● Write pseudocode to:
	○ Determine whether the rectangle is a square
	○ Assign a boolean value to the variable, is_square, indicating whether or not the rectangle has sides of equal length
		`length ← 7`
		
		`width ← 8`
		
		`is_square ← length == width`

Exercises:

	Requirements	Pseudocode
1.	● A student obtained a score of 16 out of 30 ● Write pseudocode to: ○ Convert this score to a percentage ○ Assign a boolean value to the variable, pass, indicating whether or not the percentage was greater than or equal to 50	
2.	● A rectangle has a length of 3, and a width of 5 ● Write pseudocode to: ○ Determine the area ○ Assign a boolean value to the variable, is_large, indicating whether or not the rectangle has an area greater than 15	

Chapter 4 - Conditional Statements

4A - IF Statements Analysis

An IF statement is used to control which sequence of instructions is executed depending on the result of a boolean expression.

The IF statement must **always be followed by** a boolean expression (this is called the **condition**).

If the boolean expression (condition) evaluates to True, any sequence of instructions which are **indented underneath** the IF statement is executed.

If the condition evaluates to False, the sequence of instructions indented underneath the IF statement is **skipped** and the next instruction underneath the IF statement at the **same indentation level** is executed.

Examples:

a.

Code	Trace Table

`x ← 0`
`if x <= 0`
`  x ← 1`
	Variable | **Value**
	x | ~~0,~~ 1

b.

Code	Trace Table

`x ← 10`
`if x >= 5`
`  x ← x / 2`
`  x ← x + 10`
	Variable | **Value**
	x | ~~10, 5,~~ 15

c.

```
x ← 2

y ← 1

min ← x
if y < x

  min ← y
```

Variable	Value
x	2
y	1
min	~~2,~~ 1

* How to determine the smaller of 2 numbers.

* Notice how the minimum is set up "by default" to be x. It is only changed to y if the value in y is less than x.

d.

```
x ← 2

y ← 1

max ← x
if y > x

  max ← y
```

Variable	Value
x	2
y	1
max	2

* How to determine the larger of 2 numbers.

e.

```
x ← 6
if x > 2 and x < 10

  x ← x + 2

x ← x / 2
```

Variable	Value
x	~~6, 8,~~ 4

f.

```
x ← 12
if x > 0 and x < 5

  x ← x - 2

x ← x / 2
```

Variable	Value
x	~~12,~~ 6

g.

```
t ← 21

fan_on ← False
if t >= 20 and fan_on == False
  fan_on ← True
```

Variable	Value
t	21
fan_on	~~False,~~ True

h.

```
x ← 3
if x >= 0
  x ← -x
if x <= 0
  x ← x + 3
```

Variable	Value
x	~~3, -3,~~ 0

i.

```
x ← -3
if x >= 0
  x ← -x
if x <= 0
  x ← x + 3
```

Variable	Value
x	~~-3,~~ 0

Exercises:

	Code	Trace Table

1.	`y ← -3`
	`if y <= 0`
	`  y ← y * 2`	

2.	`x ← 1`
	`if x >= 1`
	`  x ← x + 2`
	`  x ← x - 10`	

3.	`x ← 4`
	`if x > 4 and x < 6`
	`  x ← x - 1`
	`x ← x / 2`	

4.	`z ← 10`
	`if z > 0 and z < 50`
	`  z ← z / 2`
	`z ← z - 2`	

5.	`t ← 21`
	`fan_on ← True`
	`if t >= 20 and fan_on == False`
	`  fan_on ← True`	

6.

```
t ← 24

fan_on ← False
if t >= 20 and fan_on == False
  fan_on ← True
if t <= 16 and fan_on == True
  fan_on ← False
```

7.

```
y ← 1
if y >= 0
  y ← -y
if y <= 0
  y ← y * -10
```

8.

```
x ← 10
if x > 5 and x < 15
  x ← -x
if x <= 0
  x ← x + 10

x ← x + 3
```

9.

```
day_hours  ← 10

night_hours  ← 2

is_night  ← False
if is_night and night_hours < 120
   night_hours ← night_hours + 1
if not is_night and day_hours <
120
   day_hours  ← day_hours + 1
```

10.

```
x ← 8

y ← 10

z ← 2

min ← x
if y < x and y < z
   min ← y
if z < x and z < y
   min ← z
```

11.

```
x ← 8

y ← 10

z ← 2

max ← x
if y > x and y > z
   max ← y
if z > x and z > y
   max ← z
```

4B - Writing Pseudocode

For the exercises below, write the pseudocode to fulfill the requirements

Examples:

	Requirements	Pseudocode

a.	● Assign 81 to the variable, score
	● Assign False to the variable, distinction
	● If score is greater than or equal to 75, then assign the value True to the variable distinction
		`score ← 81`
		
		`distinction ← False`
		`if score >= 75`
		
		`  distinction ← True`

b.	● Assign True to the variable, in_victoria
	● Assign 10 to the variable, time
	● If in_victoria then add 1 to the variable time
		`in_victoria ← True`
		
		`time ← 10`
		`if in_victoria`
		
		`  time ← time + 1`

c.	● Assign 18 to john_age
	● Assign 20 to jane_age
	● Assign 22 to jack_age
	● Calculate the largest age and assign it to the variable largest_age
		`john_age ← 18`
		
		`jane_age ← 20`
		
		`jack_age ← 22`
		
		`largest_age ← john_age`
		`if jane_age > jack_age and`
		`  jane_age > john_age`
		
		`  largest_age ← jane_age`
		`if jack_age > jane_age`
		`  and jack_age > john_age`
		
		`  largest_age ← jack_age`

Exercises:

	Requirements	Pseudocode
1.	● Assign 58 to the variable, score ● Assign False to the variable, credit ● If score is greater than or equal to 60, then assign the value True to the variable credit	
2.	● Assign False to the variable, in_WA ● Assign 12 to the variable, time ● If in_WA then subtract 1 from the variable time	
3.	● Assign 18 to john_age ● Assign 20 to jane_age ● Assign 22 to jack_age ● Calculate the middle age and assign it to the variable mid_age	

4C - IF / ELSE IF / ELSE Statements Analysis

When we would like the program to execute a sequence of instructions when the condition to an IF statement is **False**, we use the else statement.

All the instructions indented beneath the else statement will be executed when the IF condition is False.

Examples:

a.

```
x ← 0
if x > 0
  x ← 1
else
  x ← 2
```

Variable	Value
x	~~0,~~ 2

b.

```
x ← 10
if x > 20
  x ← x + 2
else
  x ← x - 2
x ← x / 2
```

Variable	Value
x	~~10, 8,~~ 4

* Unindented code after IF statements are always executed sequentially afterwards

c.

```
x ← 16
if x > 20
  x ← x + 2
else if x > 10
  x ← x - 2
else
  x ← 2
x ← x / 2
```

Variable	Value
x	~~16, 14,~~ 7

d.

```
x ← 8
if x > 20
  x ← x + 2
  x ← x + 4
else if x > 10
  x ← x - 2
  x ← x + 4
else
  x ← x + 4
  x ← x + 2
x ← x / 2
```

Variable	Value
x	~~8, 12, 14,~~ 7

* All indented pseudocode under an IF statement are executed in sequence.

e.

```
a ← 32
b ← 42
if a > 20 and b < 40
  a ← b + 2
  b ← a + 4
else if a > 20 and b < 50
  a ← b - 2
  b ← a - 4
else if x > 30 and b < 50
  a ← b + 4
  b ← a + 2
else
  a ← b - 4
  b ← a - 2
```

Variable	Value
a	~~32,~~ 40
b	~~42,~~ 36

f.

```
x ← 1
if x >= 1
  x ← x + 1
else if x >= 2
  x ← x - 1
else if x >= 3
  x ← x - 2
else
  x ← 0
```

Variable	Value
x	~~1,~~ 2

g.

```
x ← 1
if x >= 1
  x ← x + 1
if x >= 2
  x ← x - 1
if x >= 3
  x ← x - 2
else
  x ← 0
```

Variable	Value
x	~~1, 2, 1,~~ 0

* Notice how the sequential IF statements are always checked (contrast with the example above where the ELSE IFs are only checked when the IF statement before it is false)

Exercises:

	Code	Trace Table

1.	`y ← -2`
	`if y > 0`
	`  y ← y + 1`
	`else`
	`  y ← y - 2`	

2.	`z ← 20`
	`if z >= 30`
	`  z ← z - 20`
	`else`
	`  z ← z + 20`
	`z ← z / 4`	

3.	`x ← 2`
	`if x >= 2`
	`  x ← x + 2`
	`else if x < 0`
	`  x ← x - 2`
	`else`
	`  x ← 4`
	`x ← x * 2`	

4.

```
y ← 0
if y > 10
  y ← y + 2
  y ← y + 4
else if y > 5
  y ← y - 2
  y ← y + 4
else
  y ← y + 4
  y ← y + 2
y ← y * 0.5
```

5.

```
a ← 16
b ← 20
if a > 20 and b < 30
  a ← b + 2
  b ← a + 4
else if a > 10 and b < 30
  a ← b - 3
  b ← a - 2
else if a > 0 and b < 50
  a ← b + 4
  b ← a + 2
else
  a ← b - 4
  b ← a - 2
```

6.

```
x ← 2
if x >= 1
  x ← x - 1
else if x >= 2
  x ← x + 1
else if x >= 3
  x ← x - 2
else
  x ← 0
```

7.

```
z ← 2
if z >= 2
  z ← z + 1
if z >= 3
  z ← z - 1
if z >= 4
  z ← z - 2
else
  z ← z * 10
```

4D - Writing Pseudocode

For the exercises below, write the pseudocode to fulfill the requirements

Examples:

	Requirements	Pseudocode

a.	• Assign 61 to the variable, score
	• If score is greater than or equal to 80, then assign the value 1 to the variable grade
	• If score is between 70 inclusive and 80 exclusive, then assign the value 2 to the variable grade
	• If score is between 60 inclusive and 70 exclusive, then assign the value 3 to the variable grade
	• If score is between 50 inclusive and 60 exclusive, then assign the value 4 to the variable grade
	• If the score is below 50, assign the value 5 to the variable grade
		`score ← 61`
		
		`if score >= 80`
		`  grade ← 1`
		`else if score >= 70 and`
		`  score < 80`
		`  grade ← 2`
		`else if score >= 60 and`
		`  score < 70`
		`  grade ← 3`
		`else if score >= 50 and`
		`  score < 60`
		`  grade ← 4`
		`else`
		`  grade ← 5`
		`OR`
		`score ← 61`
		
		`if score >= 80`
		`  grade ← 1`
		`else if score >= 70`
		`  grade ← 2`
		`else if score >= 60`
		`  grade ← 3`
		`else if score >= 50`
		`  grade ← 4`
		`else`
		`  grade ← 5`

b.

- Assign 2 to the variable, x
- Is x is greater than 1, assign x to the variable y
- If x is less than -1, assign -x to the variable y
- If x is between -1 inclusive and 1 inclusive, assign 0 to the variable y

```
x ← 2

if x > 1

  y ← x
else if x < -1

  y ← -x
else

  y ← 0
```

Exercises:

	Requirements	Pseudocode
1.	● Assign 59 to the variable, score ● If score is greater than or equal to 85, then assign the value 4 to the variable GPA ● If score is between 70 inclusive and 85 exclusive, then assign the value 3 to the variable GPA ● If score is between 60 inclusive and 70 exclusive, then assign the value 2 to the variable GPA ● If score is between 50 inclusive and 60 exclusive, then assign the value 1 to the variable GPA ● If the score is below 50, assign the value 0 to the variable GPA	
2.	● Assign -3 to the variable, x ● Is x is greater than 2, assign x^2 to the variable y ● If x is less than -2, assign $-x^2$ to the variable y ● If x is between -2 inclusive and 2 inclusive, assign 0 to the variable y	
3.	● Assign 2 to the variable, x ● Assign 6 to the variable, y ● Assign 3 to the variable, z ● Determine the which value is the product of the other 2 values and assign that value into the variable, multiple ● If no values are a multiple of the other 2, then assign -1 to the variable, multiple	
4.	● Assign 4 to the variable, w ● Assign 2 to the variable, x ● Assign 6 to the variable, y ● Assign 3 to the variable, z ● Determine the which is the largest value and assign that value into the variable, max	

Chapter 5 - While Loops

5A - While Loops Analysis

When we need to repeat a sequence of instructions, we apply a concept called a loop.

There are a few types of loops in Python, in this section we will discuss the **while** loop.

A while loop operates similarly to an IF statement. The while keyword is followed by a boolean expression (called the condition), like the IF statement. If the boolean expression (condition) evaluates to True, the code indented underneath the while statement is executed (just like in the IF statement). We call this "entering the loop".

However, unlike the IF statement, when the indented sequence of instructions are completed, the while condition is **re-evaluated**. If the condition is True, then the indented code is executed again. After each execution of the sequence of indented code, the condition is always re-evaluated, and the indented block of code is always repeated *while* the condition is True. To avoid the loop going on forever (an infinite loop) we need to ensure that some part of the code within the indented sequence of instructions will eventually cause the while condition to become False.

When the condition is False, the next unidented instruction underneath the while statement is executed. We call this "exiting the loop".

Examples:

a.

Code	Trace Table

`x ← 0`
`while x < 3`
`  x ← x + 1`
	Variable | **Value**
	x | ~~0, 1, 2,~~ 3

b.

Code	Trace Table

`x ← 0`
`while x <= 3`
`  x ← x + 1`
	Variable | **Value**
	x | ~~0, 1, 2, 3,~~ 4

c.

```
i ← 1

t ← 0
while i <= 4

  t ← t + i

  i ← i + 1
```

Variable	Value
i	~~1, 2, 3, 4,~~ 5
t	~~1, 3, 6,~~ 10

* This is how we can sum all integers from 1 to 4

d.

```
a ← 0

b ← 1
while a <= 8

   a ← a + b

   b ← a + b
```

Variable	Value
a	~~0, 1, 3, 8,~~ 21
b	~~1, 2, 5, 13,~~ 34

* produces the fibonacci sequence

e.

```
x ← -2

sum ← 0
while x <= 2

   y ← x * x

   sum ← sum + y

   x ← x + 1
```

Variable	Value
x	~~-2, -1, 0, 1, 2,~~ 3
y	~~4, 1, 0, 1,~~ 4
sum	~~0, 4, 5, 5, 6,~~ 10

* sums all the values of x^2 for x ∈ Z and -2 <= x <= 2

f.

```
x ← -1

min ← ∞
while x < 3
   y ← (x - 1)² + 3
   if y < min
      min ← y

   x ← x + 1
```

Variable	Value
x	~~-1, 0, 1, 2,~~ 3
y	~~7, 4, 3,~~ 4
min	~~∞, 7, 4,~~ 3

* finds the minimum y value for -1 <= x <= 2

g.

```
a ← 0

b ← 1

f ← 0

n ← 1
while n < 6
  if n == 1

    f ← a
  else if n == 2

    f ← b
  else

    f ← a + b

    a ← b

    b ← f

  n ← n + 1
```

Variable	Value
a	~~0, 1,~~ 2
b	~~1, 2,~~ 3
f	~~0, 1, 1, 2,~~ 3
n	~~1, 2, 3, 4, 5,~~ 6

* f will be the n^{th} fibonacci number

h.

```
x1 ← 0

max_grad_x ← x1

max_grad ← -∞

while x1 <= 1
  y1 ← -x1^4 - 3x1^3 + 6x1^2 - x1 - 2
  x2 ← x1 + 0.5
  y2 ← -x2^4 - 3x2^3 + 6x2^2 - x2 - 2
  grad ← (y2 - y1) / (x2 - x1)
  if grad > max_grad
    max_grad ← grad
    max_grad_x ← (x1 + x2) / 2
  x1 ← x2
```

Variable	Value
x1	
max_grad_x	
max_grad	
y1	
x2	
y2	
grad	

* finds the maximum gradient for 0 <= x <= 1 in increments of 0.5

Exercises:

	Code	Trace Table

1.	`x ⟵ 3`
	`while x > 0`
	`  x ⟵ x - 1`	

2.	`x ⟵ 0`
	`while x <= 6`
	`  x ⟵ x + 2`	

3.	`i ⟵ 10`
	`t ⟵ 0`
	`while i <= 15`
	`  t ⟵ t + i`
	`  i ⟵ i + 1`	

4.	`a ⟵ 1`
	`b ⟵ 2`
	`while a <= 21`
	`  a ⟵ a + b`
	`  b ⟵ a + b`	

5.	`x ⟵ 0`
	`sum ⟵ 0`
	`while x <= 2`
	`  y ⟵ -x^2 + 2x`
	`  sum ⟵ sum + y`
	`  x ⟵ x + 0.5`	

6.

```
x ← 0

min ← ∞
while x <= 2

   y ← x^2 - 2x
   if y < min

      min ← y

   x ← x + 0.5
```

7.

```
x ← 0

max ← -∞
while x <= 2

   y ← -x^3 + 3x
   if y > max

      max ← y

   x ← x + 0.5
```

8.

```
a ← 2

b ← 1

L ← 0

n ← 1
while n < 6
  if n == 1

    L ← a
  else if n == 2

    L ← b
  else

    L ← a + b

    a ← b

    b ← L

  n ← n + 1
```

(What is the name of this sequence?)

5B - Writing Pseudocode

For the exercises below, write the pseudocode to fulfill the requirements

Examples:

	Requirements	Pseudocode

a.	• Find the sum of all the integers between 10 and 90	`i ← 10`
		`sum ← 0`
		`while i <= 90`
		`  sum ← sum + i`
		`  i ← i + 1`

b.	• Find the sum of the first 10 square numbers (ie. 1 + 4 + 25 + 36 … + 100)	`i ← 1`
		`sum ← 0`
		`while i <= 10`
		`  square ← i * i`
		`  sum ← sum + square`
		`  i ← i + 1`

c	• Find the minimum value of $y = 0.5x^4 - x^3 - 2x^2$ for $x \in [-1, 3]$, $x \in Z$	`x ← -1`
		`min ← ∞`
		`while x <= 3`
		`  y ← 0.5x^4 - x^3 -2x^2`
		`  if y < min`
		`  min ← y`
		`  x ← x + 1`

d.	• Find the x intercept for
	$y = 0.5x^3 - x^2 - 0.18x + 0.36$
	where $x \in [1, 1.5. 2, 2.5]$
		`x ⟵ 1`
		
		`x_intercept ⟵ x`
		`while x <= 2.5`
		
		`  y ⟵ 0.5x^3-x^2-0.18x+0.36`
		`  if y == 0`
		
		`    x_intercept ⟵ x`
		
		`  x ⟵ x + 0.5`

e.	• Find the sum of y values for
	$y = 0.5x^4 - x^3 - 2x^2$ for $x \in [-1, 3], x \in Z$
		`x ⟵ -1`
		
		`sum ⟵ 0`
		`while x <= 3`
		
		`  y ⟵ 0.5x^4 - x^3 -2x^2`
		
		`  sum ⟵ sum + y`
		
		`  x ⟵ x + 1`

Exercises:

	Requirements	Pseudocode
1.	● Find the sum of all the even numbers between 20 and 90	
2.	● Find the sum of the first 10 cube numbers (ie. 1 + 8 + 27 + … + 1000)	
3.	● Calculate 20! (20 factorial)	
4.	● Find the minimum value of $y = \sin x + \cos x$ for $x \in [0, 4], x \in Z$	
5.	● Find the sum of y values for $y = \sin x + \cos x$ for $x \in [0, 4], x \in Z$	
6	● The tribonacci numbers are like the Fibonacci numbers, but instead of starting with two predetermined terms, the sequence starts with three predetermined terms and each term afterwards is the sum of the preceding three terms. ● The first few tribonacci numbers are: 0, 0, 1, 1, 2, 4, 7, 13, 24, 44, 81 ● Find the 20th tribonacci number	

5C - Applications

For the exercises below, write the pseudocode to fulfill the requirements

Examples:

	Requirements	Pseudocode

a.	• Calculate 20! (factorial)	`n ← 20`
		`factorial ← 1`
		`while n >= 1`
		`  factorial ← factorial * n`
		`  n ← n - 1`
		`OR`
		`n ← 1`
		`factorial ← 1`
		`while n <= 20`
		`  factorial ← factorial * n`
		`  n ← n + 1`

b

The Leibniz formula for π, named after Gottfried Leibniz, states that

$$1 - \frac{1}{3} + \frac{1}{5} - \frac{1}{7} + \frac{1}{9} - \cdots = \frac{\pi}{4},$$

- Calculate the value for PI after 100 iterations

```
n ← 1

sum ← 0

sign ← 1

denom ← 1
while n <= 100

  term ← sign * 1 / denom

  sum ← sum + term

  sign ← -sign

  denom ← denom + 2

  n ← n + 1

PI ← sum * 4
```

c.

- Calculate $\sqrt{3}$ using bisection method, where a = 2 and b = -1

```
a ← -2

b ← -1

c ← (a + b) / 2

f_c  ← c² - 3

accuracy ← |f_c|

answer ← a
while accuracy > 0.01

  f_a ← a² - 3

  f_b ← b² - 3
  if f_a * f_c < 0

    b ← c
  else

    a ← c

  c ← (a + b) / 2

  f_c  ← c² - 3

  accuracy ← |f_c|

answer  ← c
```

d.

- Using the left-endpoint estimate rectangle rule for integration,

  calculate the area under the curve
  $y = 9 - 0.1x^2$

  between x = 1 and x = 6 using 10 rectangles

```
a ← 1
b ← 6
n ← 10
step ← (b - a) / n
sum ← 0
x ← a
while x < b
  f_x ← 9 - 0.1x^2
  sum ← sum + f_x * step
  x ← x + step
---------- OR ----------
a ← 1
b ← 6
n ← 10
step ← (b - a) / n
sum ← 0
x ← a
while x < b
  f_x ← 9 - 0.1x^2
  sum ← sum + f_x
  x ← x + step
sum ← sum * step
```

Exercises:

	Requirements	Pseudocode
1.	$\log_e 2$ = $\ln 2 = 1 - \frac{1}{2} + \frac{1}{3} - \frac{1}{4} + \frac{1}{5} - \frac{1}{6} + \cdots$ ● Calculate ln 2 using the above series up to 200 terms	
2.	● Calculate $\sqrt[3]{4}$ using the bisection method	
3.	● Using the Trapezoidal rule for integration, calculate the area under the curve $y = 9 - 0.1x^2$ between x = 2 and x = 5 using 6 trapeziums	

Chapter 6 - Functions

6A - Functions Analysis

Functions are chunks of pseudocode which have been grouped together under a name and perform a specific task or calculation, given some information.

For example, let's imagine we would like to add 1 to the value in a variable. We might write a function like the one below:

```
def addOne(x)
  result ← x + 1
  return result
```

This is the name of the function.
This is the body of the function.
This is the parameter to the function.

When we write a function like the one above, the pseudocode does not run… yet. In order to use the function, we must call it.

For example:

```
z ← addOne(10)
```

This is the name of the function we are calling.
This is the argument given to the function.

Notice a few points:

- To call the function, we must use the same name we defined the function as
- The number of arguments must match the number of parameters

Examples:
Complete a trace table for each of the pseudocode functions below.

a.

```
def add(x, y)
  total ← x + y
  return total

z ← add(10, 20)
```

Variable	Value
x	10
y	20
total	30
z	30

b.

```
def add(x, y)
  return x + y

z ← add(10, 20)
```

Variable	Value
x	10
y	20
z	30

c.

```
def square(x)
  sq ← x * x
  return sq

z ← square(5)
```

Variable	Value
x	5
sq	25
z	25

d.

```
def cube(x)
  cb ← x * x * x
  return cb

side ← 4

z ← cube(side)
```

Variable	Value
side	4
x	4
cb	64

		z: 64

e.	`def quadruple(x)`
	`  return x * x * x * x`
	
	`a ← 3`
	
	`z ← quadruple(a)`
		Variable / **Value**
		a: 3
		z: 81

f.	`def hypotenuse(a, b)`
	`  c ←` $\sqrt{a^2 + b^2}$
	`  return c`
	
	`side1 ← 3`
	
	`side2 ← side1 + 1`
	
	`z ← hypotenuse(side1, side2)`
		Variable / **Value**
		side1: 3
		side2: 4
		a: 3
		b: 4
		c: 5
		z: 5

Exercises:

Complete a trace table for each of the pseudocode functions below.

	Code	Trace Table

1.	`def sub(x, y)`
	`  diff ⟵ x - y`
	`  return diff`
	
	`z ⟵ sub(100, 90)`	

2.	`def sub(x, y)`
	`  return x - y`
	
	`z ⟵ sub(10, 9)`	

3.	`def double(x)`
	`  db ⟵ x + x`
	`  return db`
	
	`z ⟵ double(9)`	

4.	`def halve(x)`
	`  return x / 2`
	
	`num ⟵ 10`
	
	`z ⟵ halve(num)`	

5.
```
def energy(m)
  return m * 300000000²

mass ⟵ 5

z ⟵ energy(mass)
```

6.
```
def area(l, w)

  a ⟵ l * w
  return a


x ⟵ 3

y ⟵ 4

z ⟵ area(x, y)
```

7.
```
def perimeter(l, w)
  return 2 * (l + w)


x ⟵ 3

y ⟵ 4

z ⟵ perimeter(x, y)
```

8.

```
def min(x, y)
  result ← x
  if y < x
    result ← y
  return result

z ← min(10, 5)
```

6B - Writing Functions

For the exercises below, write the pseudocode to fulfill the requirements

Examples:

	Requirements	Pseudocode

a.	Write a function:
	● named: **`third`**
	● with a parameter: **`x`**
	● which returns x / 3
	
	Then:
	● call the function
	● supply it an argument of 9,
	● and assign the returned value to the variable **`answer`**
		`def third(x)`
		`  return x / 3`
		
		`answer ← third(9)`

b.	Write a function:
	● named: multiply
	● with 2 parameters: **`a, b`**
	● which returns the product of a and b
	
	Then:
	● call the function
	● supply it arguments of -2 and 6
	● and assign the returned value to the variable **`product`**
		`def multiply(a, b)`
		`  return a * b`
		
		`product ← multiply(-2, 6)`

c	Write a function:
	● named: average
	● with 2 parameters: **`x, y`**
	● which returns the average of x and y
	
	Then:
	● call the function
	● supply it arguments of 4 and 6
	● and assign the returned value to the variable **`z`**
		`def average(x, y)`
		
		`  sum ← x + y`
		`  return sum / 2`
		
		`z ← average(4, 6)`

d. Write a function:
- named: isPositive
- with 1 parameters: **x**
- which returns True if x is 0 or positive, False otherwise

Then:
- call the function
- supply it an argument of 10
- and assign the returned value to the variable **p**

```
def isPositive(x)
  if x >= 0
    return True
  else
    return False

p ← isPositive(10)

-----Alternative-----

def isPositive(x)

  result ← True
  if x < 0

    result ← False
  return result


p ← isPositive(10)
```

e. Write a function:
- named: max
- with 2 parameters: **a, b**
- which returns a if a is larger than b, b otherwise

Then:
- call the function
- supply it arguments of -3, -4
- and assign the returned value to the variable **c**

```
def max(a, b)

  result ← a
  if b > a

    result ← b
  return result


c ← max(-3, -4)

-----Alternative-----

def max(a, b)
  if a > b
    return a
  else
    return b


c ← max(-3, -4)
```

Exercises:

Write the pseudocode to fulfill the requirements

	Requirements	Pseudocode

1.	Write a function:
	● named: **`circumference`**
	● with a parameter: **`r`**
	● which returns the circumference of a circle with radius, r
	● You may use the value of 3.1416 as an approximation for PI
	
	Then:
	● call the function
	● supply it an argument of 2,
	● and assign the returned value to the variable **`c`**
		`def circumference(r)`
		`  return 3.1416 * r * r`
		
		`c ⟵ circumference(2)`

2.	Write a function:
	● named: **`divide`**
	● with 2 parameters: **`x, y`**
	● which returns the quotient of x and y
	
	Then:
	● call the function
	● supply it arguments of 5 and -2
	● and assign the returned value to the variable **`quotient`**
		`def divide(x, y)`
		`  return x / y`
		
		`quotient⟵ divide(5, -2)`

3.	Write a function:
	● named: average
	● with 3 parameters: **`a, b, c`**
	● which returns the average of a, b and c
	
	Then:
	● call the function
	● supply it arguments of 1, 2, and 3
	● and assign the returned value to the variable **`d`**
		`def average(a, b, c)`
		`  return (a + b + c) / 3`
		
		`d ⟵ average(1, 2, 3)`

4. Write a function:
   - named: canDrive
   - with 1 parameters: **`age`**
   - which returns True if age is 18 or above, False otherwise

   Then:
   - call the function
   - supply it arguments of 17
   - and assign the returned value to the variable **`d`**

```
def canDrive(age)
  if age >= 18
    return True
  else
    return False

d ⟵ canDrive(17)

-----Alternative-----

def canDrive(age)

  result ⟵ True
  if x < 8

    result ⟵ False
  return result


d ⟵ canDrive(17)
```

5. Write a function:
   - named: max
   - with 2 parameters: **`x, y, z`**
   - which returns the largest of the values in the parameters

   Then:
   - call the function
   - supply it arguments of -3, -4, 0
   - and assign the returned value to the variable **`biggest`**

```
def max(x, y, z)
  if x >= y and x >= z
    return x
  else if y >= x and y >= z
    return y
  else
    return z

biggest ⟵ max(-3, -4, 0)
```

6C - Multiple Functions Analysis

We may define multiple functions in our pseudocode, and call each function separately and multiple times.

In some cases, we may also pseudocode within one function, call another function.

Examples:

a.

```
def sub(x, y)
  diff ← x - y
  return diff

def double(x)
  db ← x + x
  return db

a ← sub(6, 4)
z ← double(a)
```

Variable	Value
x	~~6,~~ 2
y	4
diff	2
a	2
db	4
z	8

b.

```
def sub(x, y)
  diff ← x - y
  return diff

def double(x)
  db ← x + x
  return db

z ← double(sub(10, 7))
```

Variable	Value
x	~~10,~~ 3
y	7
diff	3
db	6
z	6

d.

```
def area(l, w)
  return l * w

def perim(l, w)
  return 2 * (l + w)

def min(x, y)

  result ← x
  if y < x

    result ← y
  return result


length ← 4

width ← 3

a ← area(length, width)

p ← perim(length, width)

z ← min(a, p)
```

Variable	Value
length	4
width	3
l	4
w	3
a	12
p	14
z	12

e.

```
def area(l, w)
  return l * w

def perim(l, w)
  return 2 * (l + w)

def min(x, y)

  result ← x
  if y < x

    result ← y
  return result


a ← 4

b ← 5

z ← min(area(a, b), perim(a, b))
```

Variable	Value
a	4
b	5
l	4
w	5
z	18

f.

```
def square(x)
  s ← x * x
  return s

def area_circle(r)
  a ← 3.1416 * square(r)
  return a

z ← area_circle(2)
```

Variable	Value
r	2
x	2
s	4
a	12.5664
z	12.5664

Chapter 7 - Introduction to Lists

7A - Initialising Lists

Lists are collections of values referenced by a single variable name.

For example, if we wanted to store 3 numbers: -45.2, 0.25, and 2.333, we could initialise 3 variables as follows:

```
number1 ⟵ -45.2

number2 ⟵ 0.25

number3 ⟵ 2.333
```

Or we could use a list called names as follows:

```
numbers ⟵ [-45, 0.25, 2]
```

We use square brackets [] to contain the values inside a list, and we use commas to separate them.

Similarly, if we wanted to store 3 integers: 52, 21, 1 we could initialise 3 variables:

```
integer1 ⟵ 52

integer2 ⟵ 21

integer3 ⟵ 1
```

Or we could use a list called integers as follows:

```
integers ⟵ [52, 21, 1]
```

Examples:
Write code that would initialise the following lists:

	Initialisation goal	Code
a.	An empty list called heights.	`heights ← []`
b.	A list of numbers called heights with the values: 1.5, 1.47, 1.83, 1.76	`heights ← [1.5, 1.47, 1.83, 1.76]`
c.	A list of integers called ages with the values: 15, 19, 21, 65, 2	`ages ← [15, 19, 21, 65, 2]`
d.	A list of floating point numbers called heights with the values: 165.3, 182.45	`heights ← [165.3, 182.45]`
e.	A list of booleans called hasPaid with the values: True, True, False, True, True	`hasPaid ← [True, True, False, True, True]`
f.	A list of: 3 lists of: integers called collection where each of the inner list of integers contains the values: 1,2	`collection ← [[1, 2], [1, 2], [1, 2]]`

Exercises:
Write code that would initialise the following lists:

	Initialisation goal	Code
1.	An empty list called prices.	
2.	A list of numbers called areas with the values: 25, 30, 4	
3.	A list of integers called scores with the values: 150, 0, 100, 200	
4.	A list of floating point numbers called temperatures with the values: 15.3, 42.0, 32.1	
5.	A list of booleans called registered with the values: False, True, True, False	
6.	A list of: 2 lists of: numbers called invoices where each of the inner list of numbers contains the values: 100.0, 16.5	
7.	A list of: 3 lists called marks. Each inner list contains 3 floating point numbers recording the marks of a student. The details of each inner list are below: 100.0, 99.2, 95.7 76.2, 70.5, 78.0 85.0, 65.4, 72.2	

7B - List Elements

Definition: The values inside a list are called ‘elements’.

Examples:
Consider the following lists:

```
lengths = [23, 12, 10, 0, 22, 31]
```

1) What is the name of the list? lengths
2) What is the first element in the list? **23**
3) What is the last element in the list? 31
4) What is the third element in the list? **10**
5) What is the 5th element in the list? **22**

```
dimensions = [[23, 12], [11, 13], [10, 5]]
```

1) What is the name of the list? dimensions
2) What is the 2nd element in the list? **[11, 13]**
3) What is the last element in the list? **[10, 5]**
4) What is the 1st element of the 2nd element in the list? **11**
5) What is the 2nd element of the 1st element in the list? **12**
6) What is the 1st element in the last element in the list? **10**

Exercises:
Identify the elements in the following lists:

	Question
1.	`ages = [17,16, 21, 65, 6, 11]` 1) What is the name of the list? 2) What is the second element in the list? 3) What is the last element in the list? 4) What is the fourth element in the list? 5) What is the 2nd last element in the list?
2.	`squares = [[2, 4], [3, 9], [4, 16]]` 1) What is the name of the list? 2) What is the first element in the list? 3) What is the second element in the list? 4) What is the second element of the last element in the list? 5) What is the first element of the first element in the list? 6) What is the last element in the last element in the list?

7C - Appending to Lists

Lists can be expanded after they are initialised by ‘appending’ elements to them.

Appending an element will add that element to the **end** of the list.

So if we initialised a list as follows:

```
areas ← [24, 4, 6]
```

We can add the element 18 to the end of it by using the following pseudocode instruction:

```
areas.append(18)
```

And the list will be updated to:

```
[24, 4, 6, 18]
```

Examples:

Write down the code which will fulfil the requirements.
Then write down the final values in the list.

	Requirement	Code	Final List value

a.	Initialise an empty list called weights.
	Append the element 0.22.
	Append the element 1.45.
	Append the element 1.23.
		`weights ← []`
		`weights.append(0.22)`
		`weights.append(1.45)`
		`weights.append(1.23)`
			`[0.22, 1.45, 1.23]`

b.	Initialise an empty list called ages.
	Append the element 17.
	Append the element 21.
	Append the element 7.
		`ages ← []`
		`ages.append(17)`
		`ages.append(21)`
		`ages.append(7)`
			`[17, 21, 7]`

c.	Initialise an empty list called pairs.
	Append the element [1, 4]
	Append the element [2, 8]
	Append the element [4, 16]
		`pairs ← []`
		`pairs.append([1, 4])`
		`pairs.append([2, 8])`
		`pairs.append([4, 16])`
			`[[1, 4],`
			`[2, 8],`
			`[4, 16]]`

d.	Initialise a list called scores as [200, 300]
	Append the element 300.
	Append the element 250
		`scores ← [200, 300]`
		`scores.append(300)`
		`scores.append(250)`
			`[200, 300, 300, 250]`

e.	Initialise a list called cubes as
	[[2, 8], [4, 64]]
	Append the element [-3, -9]
		`cubes ← [[2, 8], [4, 64]]`
		`cubes.append([-3, -9])`
			`[[2, 8],`
			`[4, 64],`
			`[-3, -9]]`

f.	Initialise an empty list called nums.
	Using a loop, append all the integers from 1 to 100.
		`nums ← []`
		`i ← 0`
		`while i <= 100`
		`  nums.append(i)`
		`  num ← num + 1`
			`[1, 2, 3, 4… 100]`

g.	Initialise a list called countdown as [100, 99]
	Using a loop, append all the integers from 98 down to 1.
		`countdown ← [100 , 99]`
		`i ← 98`
		`while i >= 1`
		`  nums.append(i)`
		`  num ← num - 1`
			`[100, 99, 98, … 1]`

Exercises:

Write down the code which will fulfil the requirements.
Then write down the final values in the list.

	Requirement	Code	Final List Value
1.	Initialise an empty list called pi_digits. Append the element 3. Append the element 1. Append the element 4.		
2.	Initialise an empty list called scores. Append the element 87. Append the element 93. Append the element 67..		
3.	Initialise an empty list called coords. Append the element [-2, -4] Append the element [0, 0] Append the element [3, 4]		
4.	Initialise a list called ages as [20, 14] Append the element 30. Append the element 22.		
5.	Initialise a list called students as [[145, 25], [146, 20]] Append the element [147, 22]		
6.	Initialise an empty list called nums. Using a loop, append all the integers from 100 to 150.		
7.	Initialise a list called countdown as [200, 198] Using a loop append all the integers from 196 down to 0, counting down by 2.		

7D - List Indices

Because lists now combine all the values inside it into just one variable (with one name), we need a way to refer to individual elements inside the list.

Definition: An index refers to the position of the element in the list. In our pseudocode convention, the first position has an index of 0.

Examples:
Consider the list and questions in the previous exercise

```
plots ← [4, 3, 1, 5, 0, 2]
```

1) What is the index of the first element? **0**
2) What is the index of the last element? **5**
3) What is the index of the 5th element? **4**
4) What is the index of the element with the value of 5? **3**

Exercises:

	Question
1.	**`scores = [78, 100, 65, 59, 98, 45]`** 1) What is the index of the first element? 2) What is the index of the last element? 3) What is the index of the 3rd element? 4) What is the index of 98? 5) What is the index of the highest score in the list?
2.	**`exams_scores = [ 99.2, 97.5, 80.3, 75.4, 99.8, 56 ]`** 1) What is the index of the first element? 2) What is the index of the last element? 3) What is the index of the 2nd element? 4) What is the index of 75.4? 5) What is the index of the element with the lowest value? 6) What is the index of the element with the highest value?

7E - Indexing Lists

To read an element from a list, we need to refer to it using its index. The index is contained between the [] brackets after the name of the list.

As with any variable, if we wish to read from it, it must be on the RHS of the assignment operator (⟵).

Remember that the first element has the index of 0. This will trip you up, guaranteed! It even trips up the most experienced of programmers sometimes!

Note that the [] is being used for a different purpose here than when it was used to initialise the list (see above section).

```
volumes ⟵ [8, 125, 64.8]
```

In the above code the square brackets are used to initialise the list. You can tell this is the case because the brackets and the values inside it appear to the RHS of the assignment operator.

```
volumes [2]
```

In the above code the square brackets are used to *index* the list. You can tell this is the case because the value inside the brackets is an integer and it does not appear to the RHS of the assignment operator.

Examples:
Consider the following list:

```
scores ⟵ [100, 94, 76, 45, 67, 82]
```

Write down the values of the following elements:

1) scores[1] **94**
2) scores[5] **82**
3) scores[0] **100**

Write down the expression to index the following elements in the list:

1) 100 **scores[0]**
2) 67 **scores[4]**
3) 76 **scores[2]**

Exercises:

	Question
1.	`marks ← [78, 100, 65, 59, 98, 45]` Write down the values of the following elements: 1) marks[2] 2) marks[0] 3) marks[5]
2.	`marks ← [78, 100, 65, 59, 98, 45]` Write down the expression to index the following elements in the list: 1) 100 2) 59 3) 78
3.	`diameters ← [ 3.4, 0, 1.2, 2.1, 3.5, 0.33 ]` Write down the values of the following elements: 1) members [1] 2) members [5] 3) members [0]
4.	`diameters ← [ 3.4, 0, 1.2, 2.1, 3.5, 0.33 ]` Write down the expression to index the following elements in the list: 1) 0.33 2) 0 3) 1.2

7F - Indexing Lists with Expressions

The index can also be an expression that evaluates to an integer.

Consider the following list:

```
points ⟵ [2, 0, 3, 4, 1, 5]

x ⟵ 5

points[x]
```

Will also be the same as points[5]

```
y ⟵ 6

x ⟵ 2

points[y / x]
```

Will evaluate to points[3]

In terms of order of operations, the index operator [] is at the same level as the parentheses ().

Note that list elements can also be used as indices.

```
points ⟵ [2, 0, 3, 4, 1, 5]

points[points[2]]
```

We evaluate the inner indexing operation first

```
points[points[2]]
points[3]
```

Then we evaluate the remaining index operation.

```
4
```

Exercises:
Evaluate the index and write down the value of the element indexed

	Question
1.	ids ⟵ [2, 4, 1, 5, 3, 0] n = 2 ids[n]
2.	ids ⟵ [2, 4, 1, 5, 3, 0] n = 10 ids[n/2]
3.	ids ⟵ [2, 4, 1, 5, 3, 0] ids[ids[0]]
4.	ids ⟵ [2, 4, 1, 5, 3, 0] ids[ids[1] / 2]
5.	ids ⟵ [2, 4, 1, 5, 3, 0] attempts ⟵ [3, 0, 1, 1, 1, 2] id ⟵ ids[5] attempts [id]
6.	ids ⟵ [2, 4, 1, 5, 3, 0] attempts = [3, 0, 1, 1, 1, 2] attempts [ids[3]]

7G - Looping Through Lists (Analysis)

We can use loops to inspect elements in a list by using the loop variable as the index into the list.

Note: the print() function will output its argument on a display.
E.g. print(3.14159) will display 3.14159, and
print(1 + 3) will display 4.

Examples:
In the exercises below, walk through the code, fill in the trace table and attempt to determine what the code is doing.

Note:
All questions below apply to the following list:

```
ages ← [17, 6, 21, 65, 6]
```

<table>
<tr><th></th><th>Code</th><th>Trace Table</th><th>Purpose</th></tr>
<tr><td>a.</td><td>n ← 0
while n < 5
  print(ages[n])
  n ← n + 1</td><td><table><tr><th>Variable</th><th>Value</th><th>Output</th></tr><tr><td>n

ages[n]</td><td><s>0, 1, 2, 3, 4,</s> 5

<s>17, 6, 21, 65,</s> 6</td><td>17
6
21
65
6</td></tr></table></td><td>Displays all the elements in the list in order.</td></tr>
<tr><td>b.</td><td>n ← 0
found ← False
while n < 5
  if ages[n] == 21
  found ← True
  n ← n + 1
print(found)</td><td><table><tr><th>Variable</th><th>Value</th><th>Output</th></tr><tr><td>n

ages[n]

found</td><td><s>0, 1, 2, 3, 4,</s> 5

<s>17, 6, 21, 65,</s> 6

<s>False,</s>
True</td><td>True</td></tr></table></td><td>Determines whether or not the element 21 is in the list.</td></tr>
</table>

<table>
<tr><th></th><th>Code</th><th>Trace Table</th><th>Purpose</th></tr>
<tr><td>c.</td><td>i ← 0
total ← 0
while i < 5
  total ← total + ages[i]
  i ← i + 1
print(total)</td><td><table><tr><th>Variable</th><th>Value</th><th>Output</th></tr><tr><td>n</td><td><s>0, 1, 2, 3, 4,</s> 5</td><td>115</td></tr><tr><td>ages[n]</td><td><s>17, 6, 21, 65,</s> 6</td><td></td></tr><tr><td>total</td><td><s>0, 17, 23, 44, 109,</s> 115</td><td></td></tr></table></td><td>Sums all the elements in the ages list and displays the sum.</td></tr>
<tr><td>d.</td><td>i ← 0
v ← ages[0]
while i < 5
  if ages[i] > v
  v ← ages[i]
  i ← i + 1
print(v)</td><td><table><tr><th>Variable</th><th>Value</th><th>Output</th></tr><tr><td>n</td><td><s>0, 1, 2, 3, 4,</s> 5</td><td>65</td></tr><tr><td>ages[n]</td><td><s>17, 6, 21, 65,</s> 6</td><td></td></tr><tr><td>v</td><td><s>17, 21,</s> 65</td><td></td></tr></table></td><td>Determines the maximum value in the list.</td></tr>
<tr><td>e.</td><td>n ← 0
v1 ← ages[0]
while n < 5
  if ages[i] > v1:
  v1 ← ages[i]
  n ← n + 1

n ← 1
v2 ← ages[0]
while n < 5
  if ages[i] > v2 and ages[i] < v1
  v2 ← ages[i]
  n ← n + 1
print(v2)</td><td><table><tr><th>Variable</th><th>Value</th><th>Output</th></tr><tr><td>n</td><td><s>0, 1, 2, 3, 4, 5, 0, 1, 2, 3, 4,</s> 5</td><td>21</td></tr><tr><td>ages[n]</td><td><s>17, 6, 21, 65, 6, 17, 6, 21, 65,</s> 6</td><td></td></tr><tr><td>v1</td><td><s>17, 21,</s> 65</td><td></td></tr><tr><td>v2</td><td><s>17,</s> 21</td><td></td></tr></table></td><td>Determines the second highest value in the list.</td></tr>
</table>

<table>
<tr><th></th><th>Code</th><th>Trace Table</th><th>Purpose</th></tr>
<tr><td>f.</td><td>a ⟵ []
n ⟵ 0
while n < 5
  a.append(ages[n])
  n ⟵ n + 2
n ⟵ 0
while n < 3:
  print(a[n])
  n ⟵ n + 1</td><td><table><tr><th>Variable</th><th>Value</th><th>Output</th></tr><tr><td>a</td><td>[17, 21, 6]</td><td></td></tr><tr><td>n</td><td><s>0, 1, 2, 3, 4, 5,</s>
<s>0, 1, 2,</s> 3</td><td></td></tr><tr><td>ages[n]</td><td><s>17, 6, 21, 65,</s> 6</td><td></td></tr><tr><td>a[n]</td><td><s>17, 21,</s> 6</td><td></td></tr></table></td><td>Creates a new list, a, which contains every 2nd element in the ages list.</td></tr>
<tr><td>g.</td><td>i ⟵ 0
dup ⟵ False
while i < 5
  j ⟵ i + 1
  while j < 5:
  if ages[i] == ages[j]
  dup ⟵ True
  j ⟵ j + 1
  i ⟵ i + 1
print(dup)</td><td><table><tr><th>Variable</th><th>Value</th><th>Output</th></tr><tr><td>i</td><td><s>0,1,2,3,4</s>,5</td><td>True</td></tr><tr><td>j</td><td><s>1, 2, 3, 4, 5, 2, 3,</s>
<s>4, 5, 3, 4, 5, 4, 5,</s>
5</td><td></td></tr><tr><td>dup</td><td><s>False,</s> True</td><td></td></tr><tr><td>ages[i]</td><td><s>17, 6, 21, 65,</s> 6</td><td></td></tr><tr><td>ages[j]</td><td><s>6, 21, 65, 6, 21,</s>
<s>65, 6, 65, 6,</s> 6</td><td></td></tr></table></td><td>Detects whether any element is duplicated in the list</td></tr>
</table>

Exercises:
In the exercises below, walk through the code, fill in the trace table and attempt to determine what the code is doing.

Note:
All questions below apply to the following list:

```
ages ← [17, 6, 21, 65, 6]
```

	Code	Trace Table	Purpose

1.	n ← 4
	while n >= 0:
	  print(ages[n])
	  n ← n - 1		

2.	n ← 0
	found ← False
	while n < 5:
	  if ages[n] == -32
	  found ← True
	  n ← n + 1
	print(found)		

3.	i ← 0
	a ← 0
	while i < 5
	  a ← a+ ages[i]
	  i ← i + 1
	a ← a / 5
	print(a)		

4.	i ← 0
	v ← ages[0]
	while i < 5
	  if ages[i] < v
	  v ← ages[i]
	  i ← i + 1
	print(v)		

	Code	Trace Table	Purpose

5.	n ⟵ 0
	v1 ⟵ ages[0]
	while n < 5
	  if ages[i] < v1:
	  v1 ⟵ ages[i]
	  n ⟵ n + 1
	
	n ⟵ 0
	v2 ⟵ ages[0]
	while n < 5:
	  if ages[i] < v2 and ages[i] > v1
	  v2 ⟵ ages[i]
	  n ⟵ n + 1
	
	print(v2)		

6.	r ⟵ []
	n ⟵ 4
	while n >= 0
	  r.append(ages[n])
	  n ⟵ n - 1		

7.	i ⟵ 0
	b ⟵ False
	while i < 5
	  j ⟵ i + 1
	  while j < 5:
	  if ages[i] + ages[j] == 86
	  b ⟵ True
	  j ⟵ j + 1
	  i ⟵ i + 1
	print(b)		

7H - Looping Through lists (Writing Code)

In the exercises below, write down the code to fulfill the requirements. You may wish to draw up your own trace table if you wish to help.

Note:
All questions below apply to the following list:

```
scores ← [3, -3, 1, 5, 1, -3, 3]
```

	Requirement	Code
1.	Display every 2nd element in the list	
2.	Determine whether or not an integer stored in the number variable exists in the list (use a loop).	
3.	Create a new list, third, which contains only the first 3 elements. Use a loop to do this.	
4.	Calculate the product of all the elements in the list.	
5.	Determine whether or not all the elements in the list are in ascending order (smallest to largest).	
6.	Check whether the elements in the list is a palindrome - ie. the list is the same when the elements are reversed I.e. in this case, it is.	
7.	Determine whether there are any pair of numbers which sum to 0.	
8.	Find what is the largest product of any 2 numbers in the list.	

7I - Updating lists (Analysis)

To change specific elements in a list, we can assign to an indexed element in the list

Remember this means that the indexed element must appear on the LHS of the assignment operator (=).

marks[2] = 23

As with any variable, it can assigned to an expression.

marks[0] =10 + 3 + 4

temperatures = [24, 23, 20, 19, 34, 19]

temperatures[3] = temperatures[5]

_ = Read
_ = Write

temperatures[3] = temperatures[5]
temperatures[3] = 19

Examples:
In the exercises below, walk through the code, fill in the trace table and attempt to determine what the code is doing.

Note:
All questions below apply to the following list:

```
IDs ⟵ [4, 3, 1, 2]
```

a.

Code	Trace Table	Purpose

n ⟵ 0
while n < 4
  IDs[n] ⟵ 0
  n ⟵ n + 1
	Variable / Value
	n: ~~0, 1, 2, 3, 4~~
	IDs: [4,3,1,2] [0,3,1,2] [0,0,1,2] [0,0,0,2] [0,0,0,0]
		Sets all the elements in the list to 0.

	Code	Trace Table	Purpose

b.	n ⟵ 0
	while n < 3
	  IDs[n] ⟵ IDs[3]
	  n ⟵ n + 1
		<table><tr><th>Variable</th><th>Value</th></tr><tr><td>n</td><td>~~0, 1, 2, 3,~~ 4</td></tr><tr><td>IDs</td><td>[4,3,1,2]
		[2,3,1,2]
		[2,2,1,2]
		[2,2,2,2]</td></tr></table>
			Sets all the elements in the list to the value of the last element.

c.	i ⟵ 0
	while i < 2
	  j ⟵ 3 - i
	  temp ⟵ IDs[i]
	  IDs[i] ⟵ IDs[j]
	  IDs[j] ⟵ temp
	  i ⟵ i + 1
		<table><tr><th>Variable</th><th>Value</th></tr><tr><td>i</td><td>~~0, 1,~~ 2</td></tr><tr><td>j</td><td>~~3~~, 2</td></tr><tr><td>temp</td><td>~~4,~~ 3</td></tr><tr><td>IDs</td><td>[4,3,1,2]
		[2,3,1,4]
		[2,1,3,4]</td></tr></table>
			Reverses the order of elements in the list.

d.	x ⟵ 0
	while x < 3
	  y ⟵ x + 1
	  while y < 4
	  if IDs[y] < IDs[x]:
	  temp ⟵ IDs[x]
	  IDs[x] ⟵ IDs[y]
	  IDs[y] ⟵ temp
	  y ⟵ y + 1
	  x ⟵ x + 1
		<table><tr><th>Variable</th><th>Value</th></tr><tr><td>x</td><td>~~0, 1,~~ 2</td></tr><tr><td>y</td><td>~~1, 2, 3,~~
		~~2, 3,~~ 3</td></tr><tr><td>temp</td><td>~~4,3,4,3~~,4</td></tr><tr><td>IDs</td><td>[4,3,1,2]
		[3,4,1,2]
		[1,4,3,2]
		[1,3,4,2]
		[1,2,4,3]
		[1,2,3,4]</td></tr></table>
			Sorts the list in ascending element order.

Exercises:
In the exercises below, walk through the code, fill in the trace table and attempt to determine what the code is doing.

Note:
All questions below apply to the following list:

```
IDs ← [4, 3, 1, 2]
```

	Code	Trace Table	Purpose

1.	i ← 3
	while i >= 0
	  IDs[i] ← 1
	  i ← i - 1		

2.	n ← 0
	while n < 4
	  IDs[n] ← - IDs[n]
	  n ← n + 1		

3.	i = 0
	temp ← IDs[0]
	while i < :
	  IDs[i] ← IDs[i + 1]
	  i ← i + 1
	IDs[3] ← temp		

4.	i ← 3
	temp ← IDs[3]
	while i > 0
	  IDs[i] ← IDs[i - 1]
	  i ← i - 1
	IDs[0] ← temp		

5.	x ← 0
	while x < 3
	  y ← x + 1
	  while y < 4
	  if IDs[y] > IDs[x]:
	  temp ← IDs[x]
	  IDs[x] ← IDs[y]
	  IDs[y] ← temp
	  y ← y + 1
	  x ← x + 1		

	Code	Trace Table	Purpose

6.	x ⟵ 0
	swapped ⟵ True
	while swapped == True
	  x ⟵ 0
	  swapped ⟵ False
	  while x < 3
	  if IDs[x] > IDs[x + 1]:
	  temp ⟵ IDs[x]
	  IDs[x] ⟵ IDs[x + 1]
	  IDs[x + 1] ⟵ temp
	  swapped ⟵ True
	  x ⟵ x + 1		

7J - Updating lists (Writing Code)

In the exercises below, write down the code to fulfill the requirements. You may wish to draw up your own trace table if you wish to help.

Note:
All questions below apply to the following list:

```
scores ← [3, -3, 1, 5, 1, -3, 3]
```

	Requirement	Code
1.	Determine the highest valued element in the list. Copy this value to all the elements.	
2.	Determine the median value of the list.	
3.	**Extension:** Update the list so that all the negative numbers occur before the positive ones. There is no need to do a complete sort.	
4.	**Extension:** Write code to sort a list of numbers consisting only of 0s, 1s, and 2s **without** using a nested loop.	

Appendix A - Pseudocode to Python

Below are some notes on how to convert the pseudocode conventions used this in text into Python code:

- = instead of ←
- : at the end of IF and WHILE conditions
- ELIF instead of else if
- : at the end of ELIF and else
- Spaces between lines don't matter
- * instead of ×
- Must explicitly use * to multiply variables together.
    - E.g. ab as a multiplication between variables, a and b, in mathematics will need to be converted to a * b
    - Same for multipliers before and after parentheses
- ** is the index (raised to the power of) operator, not ^
    - E.g. x^y needs to be converted to x ** y
- Variable name rules:
    - Can't start with a number
    - Alphanumeric allowed
    - No spaces
    - Only special symbol allowed is underscore _

Other pseudocode conventions used elsewhere:

- No need for END WHILE, or END IF, or END FOR keywords to indicate that a block of pseudocode is complete. In Python, unindenting indicates the end of the previous indented block of code.
- No need for the THEN keyword after the IF condition
- = may be used for equality comparison operator, remember to convert to == for equality comparison in Python

Solutions to Exercises

2A

	Assignment Statement	Valid?
1.	`d ← 0`	True
2.	`x ← 30.0 - 2`	True
3.	`0 → x`	False (the assignment operator is in the wrong direction)
4.	`(20 - 8) / 3 → y`	False (the assignment operator is in the wrong direction)
5.	`x + 2 ← 4`	False (LHS can only contain a single variable)
6.	`z + x ← (10 + 2) + 3`	False (LHS can only contain a single variable)
7.	`g ← ((10 / 2) × (3 + 5)) - 3.3`	True
8.	`a / 3`	False (No assignment operator)
9.	`x + z + 3`	False (No assignment operator)
10.	`perimeter ← 2 * 10 + 2 * 5`	True

2B

	Assignment Statement	Final values of x and y
1.	`y ← -3` `x ← -6`	x: -6 y: -3
2.	`y ← 0.4` `x ← y / 2`	x: 0.2 y: 0.4
3.	`y ← -3 + 2` `x ← y + 2`	x: 1 y: -1

2C

	Pseudocode	Trace-table

1.	`a ← 10`
	`b ← 20`
	`a ← b`
	`b ← a`
		Variable / **Value**
		a / ~~10,~~ 20
		b / 20

2.	`x ← 10`
	`y ← 20`
	`z ← x`
	`x ← y`
	`y ← z`
		Variable / **Value**
		x / ~~10,~~ 20
		y / ~~20,~~ 10
		z / 10

3.

```
x ← 8
y ← x - x / 2
x ← y + x
y ← x * x
```

Variable	Value
x	~~8,~~ 12
y	~~4,~~ 144

4.

```
x ← 1
y ← 1
z ← x + y
x ← y + z
y ← x + z
z ← x + y
```

Variable	Value
x	~~1,~~ 3
y	~~1,~~ 5
z	~~2,~~ 8

2D

Pseudocode	Trace-table

1.

```
a ← 10
a ← a / 2
```

Variable	Value
a	~~10,~~ 5

2.

```
x ← 8
y ← (x + 4) / 3
y ← y * 3
y ← y - 4
```

Variable	Value
x	8
y	~~4, 12,~~ 8

3.

```
x ← 4

y ← 8

y ← x + y

x ← y - x

y ← y - x
```

Variable	Value
x	~~4,~~ 8
y	~~8, 12,~~ 4

2E

	Goal	Code
1.	• Assign the value -20 to the variable, a • Halve the value of a and assign it to the variable b	`a ← -20` `b ← a / 2`
2.	• Assign the value 0.6 to the variable, d • Calculate the area of a circle with the diameter given by the value in the variable d • Assign the result to the variable, a	`d ← 0.6` `a ← 3.14159 * d * d / 4`

3.	• Assign the value 2 to the variable l
	• Assign the value 3 to the variable w
	• Assign the value 4 to the variable d
	• Calculate the volume of a rectangle with a length given by l, a width given by w, and a depth given by d
	• Assign the result to the variable v
		`l ← 2`
		`w ← 3`
		`d ← 4`
		`v ← l * w * d`

4.	• Assign the value 10 to the variable, x
	• Double the value of x and assign it back to x
	• Triple this value and assign it back to x
		`x ← 10`
		`x ← x * 2`
		`x ← x * 3`

5.	• Assign the value 180 to the variable, h
	• Assign the value 80 to the variable, w
	• Calculate the BMI given with a height (in cms) given by h, and a weight (in kgs) given by w
	  ○ BMI = kgs / m^2
	• Assign the answer to the variable, bmi
		`h ← 180`
		`w ← 80`
		`bmi ← w / ((h / 100) * (h / 100))`

3A

	Expression	Evaluation
1.	5 == 2	False
2.	10 < 10	False
3.	2 != 2	False
4.	32 >= 32	True
5.	(5 + 3) < 8	False
6.	7 + 2 * 2 == (11 - 0)	True
7.	True == False	False
8.	4.1 >= 4.01	True
9.	False	False
10.	not True	False
11.	not (1 <= 0)	True

3B

	Expression	Evaluation
1.	3 <=3 and 4 <= 4	True
2.	8 > 0 or 1 < 0	True
3.	10 == (9 + 1) and 3 < 5	True
4.	1 > 0 and 6 > 2	True
5.	50 < 34 or 34 > 43	False
6	10 == 10 and 10 < 10	False
7.	True and False	False
8.	False or True	True
9.	(10 > 5 and 4 > 1) or (50 >= 50)	True
10.	True or False	True

11.	True and True	True
12.	False or False	False

3C

	Pseudocode	**Trace-table**
1.	`age ← 18` `result ← x < 18`	**Variable** / **Value** age / 18 result / False

2.	`date ← 31/01/2001`
	`return_by ← 28/02/2001`
	`late ← date > return_by`
		Variable / **Value**
		date / 31/01/2001
		return_by / 28/02/2001
		late / False

3.	`height ← 180`
	`height ← height / 100`
	`six_feet ← height >= 1.83`
		Variable / **Value**
		height / ~~180,~~ 1.80
		six_feet / False

4.	`t ← 200.0`
	`t ← (t - 32.0) * 5/9`
	`boiling ← t >= 100`
		Variable / **Value**
		t / ~~200.0,~~ 93.33
		boiling / False

5.

```
hours ← 150

age ← 19

test_passed ← True

licensed ← hours > 120 and age > 18 and
           test_passed
```

Variable	Value
hours	150
age	19
test_passed	True
licensed	True

6.

```
distance ← 1500

distance ← distance / 1000

fast_food ← True

rating ← 4

order ← distance < 2.0 and rating >= 4
        and not fast_food
```

Variable	Value
distance	~~1500,~~ 1.5
fast_food	True
rating	4
order	False

3D

	Requirements	Pseudocode

1.	• A student obtained a score of 16 out of 30
	• Write pseudocode to:
	  ○ Convert this score to a percentage
	  ○ Assign a boolean value to the variable, pass, indicating whether or not the percentage was greater than or equal to 50
		`score ← 16 / 30`
		
		`score ← score * 100`
		
		`pass ← score >= 50`

2.	• A rectangle has a length of 3, and a width of 5
	• Write pseudocode to:
	  ○ Determine the area
	  ○ Assign a boolean value to the variable, is_large, indicating whether or not the rectangle has an area greater than 15
		`length ← 7`
		
		`width ← 8`
		
		`area ← length * width`
		
		`is_large ← area > 56`

4A

	Code	Trace Table

1.	`y ← -3`
	`if y <= 0`
	`  y ← y * 2`
		<table><tr><th>Variable</th><th>Value</th></tr><tr><td>y</td><td>~~-3,~~ -6</td></tr></table>

2.	`x ← 1`
	`if x >= 1`
	`  x ← x + 2`
	
	`  x ← x - 10`
		<table><tr><th>Variable</th><th>Value</th></tr><tr><td>x</td><td>~~1, 3,~~ -7</td></tr></table>

3.

```
x ← 4
if x > 4 and x < 6
  x ← x - 1
x ← x / 2
```

Variable	Value
x	~~4,~~ 2

4.

```
z ← 10
if z > 0 and z < 50
  z ← z / 2
z ← z - 2
```

Variable	Value
z	~~10, 5,~~ 3

* How to check if z is between 0 and 50 (exclusive). Notice how the boolean condition is repeated.

5.

```
t ← 21
fan_on ← True
if t >= 20 and fan_on == False
  fan_on ← True
```

Variable	Value
t	21
fan_on	True

6.

```
t ← 24
fan_on ← False
if t >= 20 and fan_on == False
  fan_on ← True
if t <= 16 and fan_on == True
  fan_on ← False
```

Variable	Value
t	24
fan_on	~~False,~~ True

7.

```
y ← 1
if y >= 0
  y ← -y
if y <= 0
  y ← y * -10
```

Variable	Value
y	~~1, -1,~~ 10

8.

```
x ← 10
if x > 5 and x < 15
  x ← -x
if x <= 0
  x ← x + 10
x ← x + 3
```

Variable	Value
x	~~-10, -10, 0,~~ 3

9.

```
day_hours  ← 10
night_hours  ← 2
is_night  ← False
if is_night and night_hours < 120
   night_hours ← night_hours + 1
if not is_night and day_hours < 120
   day_hours  ← day_hours + 1
```

Variable	Value
day_hours	~~10,~~ 11
night_hours	2
is_night	False

10.

```
x ← 8
y ← 10
z ← 2
min ← x
if y < x and y < z
   min ← y
if z < x and z < y
   min ← z
```

Variable	Value
x	8
y	10
z	2
min	~~8,~~ 2

11.

```
x ← 8

y ← 10

z ← 2

max ← x
if y > x and y > z
   max ← y
if z > x and z > y
   max ← z
```

Variable	Value
x	8
y	10
z	2
max	~~8,~~ 10

4B

	Requirements	Pseudocode

1.	• Assign 58 to the variable, mark
	• Assign False to the variable, credit
	• If score is greater than or equal to 60, then assign the value True to the variable credit
		`mark ← 58`
		
		`credit ← False`
		`if mark >= 60`
		
		`  credit ← True`

2.	• Assign False to the variable, in_WA
	• Assign 12 to the variable, time
	• If in_WA then subtract 1 from the variable time
		`in_WA ← True`
		
		`time ← 12`
		`if in_WA`
		
		`  time ← time - 1`

3.

- Assign 18 to john_age
- Assign 20 to jane_age
- Assign 22 to jack_age
- Calculate the middle age and assign it to the variable mid_age

```
john_age ← 18

jane_age ← 20

jack_age ← 22

mid_age ← john_age
if jane_age >= jack_age and
   jane_age <= john_age

   mid_age ← jane_age
if jane_age >= john_age and
   jane_age <= jack_age

   mid_age ← jane_age
if jack_age >= jane_age
   and jack_age <= john_age

   mid_age ← jack_age
if jack_age >= john_age
   and jack_age <= jane_age

   mid_age ← jack_age
```

4C

	Code	Trace Table

1.	`y ← -2`
	`if y > 0`
	`  y ← y + 1`
	`else`
	`  y ← y - 2`
		Variable | **Value**
		y | ~~-2,~~ -4

2.	`z ← 20`
	`if z >= 30`
	`  z ← z - 20`
	`else`
	`  z ← z + 20`
		Variable | **Value**
		z | ~~20, 40,~~ 10

```
z ← z / 4
```

3.

```
x ← 2
if x >= 2
  x ← x + 2
else if x < 0
  x ← x - 2
else
  x ← 4
x ← x * 2
```

Variable	Value
x	~~2, 4,~~ 8

4.

```
y ← 0
if y > 10
  y ← y + 2
  y ← y + 4
else if y > 5
  y ← y - 2
  y ← y + 4
else
  y ← y + 4
  y ← y + 2
y ← y * 0.5
```

Variable	Value
y	~~0, 4, 6,~~ 3

5.

```
a ← 16

b ← 20
if a > 20 and b < 30
  a ← b + 2

  b ← a + 4
else if a > 10 and b < 30
  a ← b - 3

  b ← a - 2
else if a > 0 and b < 50
  a ← b + 4

  b ← a + 2
else
  a ← b - 4

  b ← a - 2
```

Variable	Value
a	~~16,~~ 17
b	~~20,~~ 15

6.

```
x ← 2
if x >= 1
  x ← x - 1
else if x >= 2
  x ← x + 1
else if x >= 3
  x ← x - 2
else
  x ← 0
```

Variable	Value
x	~~2,~~ 3

7.

```
z ⟵ 2
if z >= 2
  z ⟵ z + 1
if z >= 3
  z ⟵ z - 1
if z >= 4
  z ⟵ z - 2
else
  z ⟵ z * 10
```

Variable	Value
z	~~2, 3, 2,~~ 20

4D

Requirements	Pseudocode

1.

- Assign 59 to the variable, score
- If score is greater than or equal to 85, then assign the value 4 to the variable GPA
- If score is between 70 inclusive and 85 exclusive, then assign the value 3 to the variable GPA
- If score is between 60 inclusive and 70 exclusive, then assign the value 2 to the variable GPA
- If score is between 50 inclusive and 60 exclusive, then assign the value 1 to the variable GPA
- If the score is below 50, assign the value 0 to the variable GPA

```
score ⟵ 59

if score >= 85
  GPA ⟵ 4
else if score >= 70 and
  score < 85
  GPA ⟵ 3
else if score >= 60 and
  score < 70
  GPA ⟵ 2
else if score >= 50 and
  score < 60
  GPA ⟵ 1
else
  GPA ⟵ 0
```

2.

- Assign -3 to the variable, x
- Is x is greater than 2, assign x^2 to the variable y
- If x is less than -2, assign $-x^2$ to the variable y
- If x is between -2 inclusive and 2 inclusive, assign 0 to the variable y

```
x ← -3

if x > 2
  y ← x * x
else if x < -2
  y ← -x * x
else
  y ← 0
```

3.

- Assign 2 to the variable, x
- Assign 6 to the variable, y
- Assign 3 to the variable, z
- Determine the which value is the product of the other 2 values and assign that value into the variable, multiple
- If no values are a multiple of the other 2, then assign -1 to the variable, multiple

```
x ← 2

y ← 6

z ← 3

multiple ← -1
if x == y * z
  multiple ← x
else if y == x * z
  multiple ← y
else if z == x * y
  multiple ← z
```

4.

- Assign 4 to the variable, w
- Assign 2 to the variable, x
- Assign 6 to the variable, y
- Assign 3 to the variable, z
- Determine the which is the largest value and assign that value into the variable, max

```
w ← 4

x ← 2

y ← 6

z ← 3

max ← w
if x > w and x > y and x > z

  max ← x
else if y > w and y > x and y > z

  max ← y
else if z > w and z > x and z > y

  max ← z
```

5A

	Code	Trace Table

1.	`x ← 3`
	`while x > 0`
	`  x ← x - 1`
		Variable | **Value**
		x | ~~3, 2, 1,~~ 0

2.	`x ← 0`
	`while x <= 6`
	`  x ← x + 2`
		Variable | **Value**
		x | ~~0, 2, 4, 6,~~ 8

3.

```
i ← 10

t ← 0
while i <= 15
  t ← t + i

  i ← i + 1
```

Variable	Value
i	~~10, 11, 12, 13, 14, 15,~~ 16
t	~~10, 21, 33, 46, 60,~~ 75

4.

```
a ← 1

b ← 2
while a <= 21
   a ← a + b

   b ← a + b
```

Variable	Value
a	~~0, 1, 3, 8, 21,~~ 55
b	~~1, 2, 5, 13, 34,~~ 89

5.

```
x ← 0

sum ← 0
while x <= 2
   y ← -x^2 + 2x

   sum ← sum + y

   x ← x + 0.5
```

Variable	Value
x	~~0, 0.5, 1, 1.5, 2,~~ 2.5
y	~~0, 0.75, 1, 0.75,~~ 0
sum	~~0, 0.75, 1.75,~~ 2.5

6.

```
x ← 0

min ← ∞
while x <= 2
   y ← x^2 - 2x
   if y < min
      min ← y

   x ← x + 0.5
```

Variable	Value
x	~~0, 0.5, 1, 1.5, 2,~~ 2.5
y	~~0, -0.75, -1, -0.75,~~ 0
min	~~∞, -0.75,~~ -1

7.

```
x ← 0

max ← -∞
while x <= 2
   y ← -x^3 + 3x
   if y > max
      max ← y
   x ← x + 0.5
```

Variable	Value
x	~~0, 0.5, 1, 1.5, 2,~~ 2.0
y	~~0, 1.375, 2, 1.125,~~ -2
min	~~-∞, 0, 1.375,~~ 2

8.

```
a ← 2

b ← 1

L ← 0

n ← 1
while n < 6
  if n == 1
    L ← a
  else if n == 2
    L ← b
  else
    L ← a + b
    a ← b
    b ← L
  n ← n + 1
```

(What is the name of this sequence?)

Variable	Value
a	~~2, 1, 3,~~ 4
b	~~1, 3, 4,~~ 7
L	~~0, 2, 1, 3, 4,~~ 7
n	~~1, 2, 3, 4, 5,~~ 6

5B

	Requirements	Pseudocode

1.	• Find the sum of all the even numbers between 20 and 90	`i ⟵ 20`
		
		`sum ⟵ 0`
		`while <= 80`
		
		`  sum ⟵ sum + i`
		
		`  i ⟵ i + 2`

2.	• Find the sum of the first 10 cube numbers (ie. 1 + 8 + 27 + … + 1000)	`i ⟵ 1`
		
		`sum ⟵ 0`
		`while i <= 10`
		
		`  sum ⟵ sum + i * i * i`
		
		`  i ⟵ i + 1`

3.	• Calculate 20! (20 factorial)	`i ⟵ 20`
		
		`factorial ⟵ 1`
		`while i >= 1`
		
		`  factorial ⟵ factorial * i`
		
		`  i ⟵ i - 1`

4.	• Find the minimum value of $y = \sin x + \cos x$ for $x \in [0, 4], x \in Z$	`x ⟵ 0`
		
		`min ⟵ ∞`
		`while x <= 4`
		
		`  y ⟵ sin x + cos x`
		`  if y < min`
		
		`    min ⟵ y`
		
		`  x ⟵ x + 1`

5.

- Find the sum of y values for
  y = sin x + cos x for x ∈ [0, 4], x ∈ Z

```
x ← 0

sum ← 0
while x <= 4

  y ← sin x + cos x

  sum ← sum + y

  x ← x + 1
```

6

- The tribonacci numbers are like the Fibonacci numbers, but instead of starting with two predetermined terms, the sequence starts with three predetermined terms and each term afterwards is the sum of the preceding three terms.
- The first few tribonacci numbers are: 0, 0, 1, 1, 2, 4, 7, 13, 24, 44, 81
- Find the 20th tribonacci number

```
a ← 0

b ← 0

c ← 1

t ← 0

n ← 1
while n <= 20
  if n == 1

    t ← a
  else if n == 2

    t ← b
  else if n == 3

    t ← c
  else

    t ← a + b + c

    a ← b

    b ← c

    c ← t

  n ← n + 1
```

5C
Exercises:

	Requirements	Pseudocode

1.	$\log_e 2$ =
	$\ln 2 = 1 - \frac{1}{2} + \frac{1}{3} - \frac{1}{4} + \frac{1}{5} - \frac{1}{6} + \cdots$
	● Calculate ln 2 using the above series up to 200 terms
		`n ← 1`
		`sum ← 0`
		`sign ← 1`
		`denom ← 1`
		`while n <= 200`
		`  term ← sign * 1 / denom`
		`  sum ← sum + term`
		`  sign ← -sign`
		`  denom ← denom + 1`
		`  n ← n + 1`
		`ln2 ← sum`

2.

- Calculate $\sqrt[3]{4}$ using the bisection method

```
a ← -2

b ← -1

c ← (a + b) / 2

f_c  ← c^3 - 4

accuracy ← |f_c|

answer ← a
while accuracy > 0.01

  f_a ← a^3 - 4

  f_b ← b^3 - 4
  if f_a * f_c < 0

    b ← c
  else

    a ← c

  c ← (a + b) / 2

  f_c  ← c^3 - 4

  accuracy ← |f_c|

answer  ← c
```

3.

- Using the Trapezoidal rule for integration,

  calculate the area under the curve
  $y = 9 - 0.1x^2$

  between x = 2 and x = 5 using 6 trapeziums

```
a ← 2

b ← 5

n ← 6

step ← (b - a) / n

sum ← 0

x ← a
while x < b

  f_x ← 9 - 0.1x^2

  x1 ← x + step

  f_x1 ← 9 - 0.1x1^2

  sum ← sum+(f_x+f_x1)/2*step

  x ← x1
---------------- OR --------------
a ← 2

b ← 5

f_a ← 9 - 0.1a^2

f_b ← 9 - 0.1b^2

n ← 6

step ← (b - a) / n

sum ← 0

x ← a + step
while x < b

  f_x ← 9 - 0.1x^2

  sum ← sum + 2 * f_x

  x ← x + step

sum ← (sum + f_a + f_b) * step/2
```

6A

	Code	Trace Table

1.	`def sub(x, y)`
	`  diff ⟵ x - y`
	`  return diff`
	
	`z ⟵ sub(100, 90)`
		Variable / **Value**
		x: 100
		y: 90
		diff: 10
		z: 10

2.	`def sub(x, y)`
	`  return x - y`
	
	`z ⟵ sub(10, 9)`
		Variable / **Value**
		x: 10
		y: 9
		z: 1

3.	`def double(x)`
	`  db ⟵ x + x`
	`  return db`
	
	`z ⟵ double(9)`
		Variable / **Value**
		x: 9
		db: 18
		z: 18

4.	`def halve(x)`
	`  return x / 2`
	
	`num ⟵ 10`
	
	`z ⟵ halve(num)`
		Variable / **Value**
		num: 10
		x: 10
		z: 5

5.

```
def energy(m)
  return m * 300000000^2

mass ← 5

z ← energy(mass)
```

Variable	Value
mass	5
m	5
z	4.5×10^{17}

6.

```
def area(l, w)
  a ← l * w
  return a

x ← 3

y ← 4

z ← area(x, y)
```

Variable	Value
x	3
y	4
l	3
w	4
a	12
z	12

7.

```
def perimeter(l, w)
  return 2 * (l + w)

x ← 3

y ← 4

z ← perimeter(x, y)
```

Variable	Value
x	3
y	4
l	3
w	4
z	14

8.

```
def min(x, y)
  result ← x
  if y < x
    result ← y
  return result

z ← min(10, 5)
```

Variable	Value
x	10
y	5
result	~~10,~~ 5
z	5

6B

	Requirements	Pseudocode

1.	Write a function:
	● named: **`circumference`**
	● with a parameter: **`r`**
	● which returns the circumference of a circle with radius, r
	● You may use the value of 3.1416 as an approximation for PI
	
	Then:
	● call the function
	● supply it an argument of 2,
	● and assign the returned value to the variable **`c`**
		`def circumference(r)`
		`  return 3.1416 * r * r`
		
		`c ← circumference(2)`

2.	Write a function:
	● named: **`divide`**
	● with 2 parameters: **`x, y`**
	● which returns the quotient of x and y
	
	Then:
	● call the function
	● supply it arguments of 5 and -2
	● and assign the returned value to the variable **`quotient`**
		`def divide(x, y)`
		`  return x / y`
		
		`quotient← divide(5, -2)`

3. Write a function:
   - named: average
   - with 3 parameters: **`a, b, c`**
   - which returns the average of a, b and c

   Then:
   - call the function
   - supply it arguments of 1, 2, and 3
   - and assign the returned value to the variable **`d`**

```
def average(a, b, c)
  return (a + b + c) / 3

d ⟵ average(1, 2, 3)
```

4. Write a function:
   - named: canDrive
   - with 1 parameters: **`age`**
   - which returns True if age is 18 or above, False otherwise

   Then:
   - call the function
   - supply it arguments of 17
   - and assign the returned value to the variable **`d`**

```
def canDrive(age)
  if age >= 18
    return True
  else
    return False

d ⟵ canDrive(17)

-----Alternative-----

def canDrive(age)

  result ⟵ True
  if x < 8

    result ⟵ False
  return result


d ⟵ canDrive(17)
```

5. Write a function:
   - named: max
   - with 2 parameters: **`x, y, z`**
   - which returns the largest of the values in the parameters

   Then:
   - call the function
   - supply it arguments of -3, -4, 0
   - and assign the returned value to the variable **`biggest`**

```
def max(x, y, z)
  if x >= y and x >= z
    return x
  else if y >= x and y >= z
    return y
  else
    return z

biggest ⟵ max(-3, -4, 0)
```

7A

1. prices ⟵ []

2. areas ⟵ [25, 30, 4]

3. scores ⟵ [150, 0, 100, 200]

4. temperatures ⟵ [15.3, 42.0, 32.1]

5. registered ⟵ [False, True, True, False]

6. invoices ⟵ [[100.0, 16.5], [100.0, 16.5]]

7. marks ⟵ [[100.0, 99.2, 95.7], [76.2, 70.5, 78.0], [85.0, 65.4, 72.2]]

7B

1. 1) ages
   2) 16
   3) 11
   4) 65
   5) 16

2. 1) `squares`
   2) `[2, 4]`
   3) `[3, 9]`
   4) `16`
   5) `2`
   6) `16`

7C

1.

```
pi_digits ⟵ []
letters.append(3)
letters.append(1)
letters.append(4)
```

2.

```
scores⟵ []
scores.append(87)
scores.append(93)
scores.append(67)
```

3.

```
coords⟵ []
coords.append([-2, -4])
coords.append([0, 0])
coords.append([3, 4])
```

4.

```
ages ⟵ [20, 14]
ages.append(30)
ages.append(22)
```

5.

```
students ⟵[ [145, 25], [146, 20] ]
students.append([147, 22])
```

6.

```
nums ⟵ []

i ⟵ 100
while i <= 150
  nums.append(i)
  i ⟵ i + 1
```

7.

```
countdown ⟵ [200, 198]

n ⟵ 196
while n >= 0
  countdown.append(n)
  n ⟵ n - 2
```

7D

1.
1) 0
2) 5
3) 2
4) 4
5) 1

2.
1) 0
2) 5
3) 1
4) 3
5) 5
6) 4

7E

1.
1) 65
2) 78
3) 45

2.
1) marks[1]
2) marks[3]
3) marks[0]

3.
1) `0`
2) `0.33`
3) `3.4`

4.
1) members[5]
2) members[0]
3) members[2]

7F

1. 1

2. 0

3. 1

4. 1

5. 3

7. 2

7G

1.

Variable	Value	Output
n	~~4, 3, 2, 1, 0,~~ -1	6, 65, 21, 6, 17
ages[n]	~~6, 65, 21, 6,~~ 17	

Displays all the elements in the list in reverse order.

2.

Variable	Value	Output
n	~~0, 1, 2, 3, 4,~~ 5	False
ages[n]	~~17, 6, 21, 65,~~ 6	
found	False	

Searches for the element -32.

3.

Variable	Value	Output
i	~~0, 1, 2, 3, 4,~~ 5	23
ages[i]	~~17, 6, 21, 65,~~ 6	
a	~~17, 23, 44, 109, 115~~, 23	

Calculates the average of all the elements in the list.

4.

Variable	Value	Output
i	~~0, 1, 2, 3, 4,~~ 5	6
ages[i]	~~17, 6, 21, 65,~~ 6	
v	~~17,~~ 6	

Determines the lowest element in the list.

5.

Variable	Value	Output
n	~~0, 1, 2, 3, 4, 5,~~ ~~0, 1, 2, 3, 4,~~ 5	17
ages[i]	~~17, 6, 21, 65, 6,~~ ~~17, 6, 21, 65,~~ 6	
v1	~~17,~~ 6	
v2	17	

Determines the 2nd lowest element in the list.

6.

Variable	Value	Output
n	~~4, 3, 2, 1, 0,~~ -1	
ages[i]	~~6, 65, 21, 6,~~ 17	
r	[6] [6, 65] [6, 65, 21] [6, 65, 21, 6] [6, 65, 21, 6, 17]	

Creates a new list, r, which contains the elements in ages in reversed order.

7.

Variable	Value	Output
i	~~0, 1, 2, 3,~~ 4	True
j	~~1, 2, 3, 4, 5,~~ ~~2, 3, 4, 5,~~ ~~3, 4, 5,~~ ~~4, 5,~~ 5	
ages[i]	~~17, 6, 21,~~ 65	
ages[j]	~~6, 21, 65, 6,~~ ~~21, 65, 6,~~ ~~65, 6,~~ 6	
b	~~False~~, True	

Determines whether any pair of elements sum to 86.

7H

1.
```
n ⟵ 0
while n < 7
  print(scores[n])
  n ⟵ n + 2
```

2.
```
i ⟵ 0
found ⟵ False
while i < 7
  if number == scores[i]
    found ⟵ True
  i ⟵ i + 1
print(found)
```

3.
```
third ⟵ []
n ⟵ 0
while n < 3
  third.append(scores[n])
  n ⟵ n + 1
```

4.

```
i ← 0
product ← 1
while i < 7
  product ← product * scores[i]
  i ← i + 1
print(product)
```

5.

```
ascending ← True
n ← 0
while n < 6
  if scores[n + 1] < scores[n]
    ascending ← False
  n ← n + 1
print(ascending)
```

6.

```
palindrome ← True
n ← 0
while n < 3
  if scores[n] != scores[6 - n]
    palindrome ← False
  n ← n + 1
print(palindrome)
```

7.

```
zero_sum ← False
x ← 0
while x < 6
  y ← x + 1
  while y < 7
    if scores[x] + scores[y] == 0
      zero_sum ← True
    y ← y + 1
  x ← x + 1
print(zero_sum)
```

8.

```
max_product ⟵ scores[0] * scores[1]
x ⟵ 0
while x < 6:
  y ⟵ x + 1
  while y < 7
    if scores[x] * scores[y] > max_product
      max_product ⟵ scores[x] * scores[y]
    y ⟵ y + 1
  x ⟵ x + 1
print(max_product)
```

7I

1.

Variable	Value
i	~~3, 2, 1, 0,~~ -1
IDs	**[4,3,1,2]** **[4,3,1,1]** **[4,3,1,1]** **[4,1,1,1]** **[1,1,1,1]**

Overwrites all the list elements with the value 1, from the highest index down to the first index.

2.

Variable	Value
n	~~0, 1, 2, 3,~~ 4
IDs	**[4,3,1,2]** **[-4,3,1,2]** **[-4,-3,1,2]** **[-4,-3,-1,2]** **[-4,-3,-1,-2]**

Negates all the integers in the list.

3.

Variable	Value
i	~~0, 1, 2,~~ 3
temp	**4**
IDs	**[4,3,1,2]** **[3,3,1,2]** **[3,1,1,2]** **[3,1,2,2]** **[3,1,2,4]**

“Rotates” all the elements in the list one position to the left.

4.

Variable	Value
i	~~3, 2, 1,~~ 0
temp	2
IDs	[4,3,1,2] [4,3,1,1] [4,3,3,1] [4,4,3,1] [2,4,3,1]

"Rotates" all the elements in the list one position to the right.

5.

Variable	Value
x	~~0, 1, 2,~~ 3
y	~~1, 2, 3, 4,~~ ~~2, 3, 4,~~ ~~3, 4,~~ 4
temp	1
IDs	[4,3,1,2] [4,3,2,1]

Sorts the list in descending order.
(This sorting algorithm is known as a selection sort).

6.

Variable	Value
x	~~0, 1, 2, 3, 4,~~ ~~0, 1, 2, 3, 4,~~ ~~0, 1, 2, 3,~~ 4
swapped	~~True~~, ~~False~~, ~~True,~~ ~~False, True,~~ False
temp	**4,3**
IDs	**[4,3,1,2]** **[3,4,1,2]** **[3,1,4,2]** **[3,1,2,4]** **[1,3,2,4]** **[1,2,3,4]**

Sorts the list in ascending order.
(This sorting algorithm is known as a bubble sort).

7J

1.

```
#find the max value first

max_value ⟵ scores[0]
i ⟵ 1
while i < 7
  if scores[i] > max_value
    max_value ⟵ scores[i]
  i ⟵ i + 1

# then copy it to the rest

i ⟵ 0
while i < 7
  scores[i] ⟵ max_value:
  i ⟵ i + 1
```

2.

```
# sort the list first

x ⟵ 0
while x < 6
  y ⟵ x + 1
  while y < 7
    if scores[y] > scores[x]
      temp ⟵ scores[x]
      scores[x] ⟵ scores[y]
      scores[y] ⟵ temp
    y ⟵ y + 1
  x ⟵ x + 1

# the median is just the middle value

print(scores[3])
```