Pseudocode and **Flowcharts** are tools for designing and documenting *algorithms.*

An **algorithm** is: a step-by-step sequence of instructions to solve a specific problem in a finite amount of time.

Summary and Rules

*These rules are adapted from **Cambridge IGCSE Computer Science 0478 – Pseudocode Guide for Teachers**, for examinations from 2017 onwards.*

- In Pseudocode keywords are in uppercase, e.g. `IF`, `REPEAT`, `PROCEDURE`

Data Type	Description	Examples	Python equivalent
`INTEGER`	A whole number (positive or negative)	`5, -2, 10, 0`	`int`
`REAL`	A number capable of containing a fractional part	`1.23, -0.18, 747.0`	`float`
`CHAR`	A single character (letter, number, space, etc…)	`'a', 'B', '1', ' ', '-'`	No char type in Python
`STRING`	A sequence of zero or more characters strung together	`"stuff", "3.14", "", ":P"`	`str`
`BOOLEAN`	The logical values	`TRUE, FALSE`	`bool (True, False)`

- Variable names are often in `lowerCamelCase` or `UpperCamelCase` — Python convention is `snake_case` (underscore separated)
- Variables are **assigned** using an arrow: `count ← 5, count ← count + 1.` In Python: `count = 5, count = count + 1`
- Numerical variables can be *combined* using the **arithmetic operators**: `+, -, *, /`, and the integer division operators of `DIV` (//) and `MOD` (%)
- Other operators, e.g., String methods, numerical functions, etc…, can be described using common symbols or with words
- Variables can be *compared* using the operators below (that all return Booleans)

Name	Pseudo	Python	Examples yielding TRUE	Name	Pseudo	Python	Examples yielding TRUE
Equals	=	`==`	`5 = 5, 'a' = 'a'`	Not Equals	`<>`	`!=`	`1 <> 2, ":)" <> ":("`
Greater Than	`>`	`>`	`3 > 2, -1 > -2,`	Less Than	`<`	`<`	`1 < 2, 1.1 < 1.2,`
Greater Than or Equal To	`>=`	`>=`	`3 >= 3, 3 >= 2, 1.2 >= 1.1, 'A' >= 'A'`	Less Than or Equal To	`<=`	`<=`	`1 <= 2, 1 <= 1, 'a' <= 'a'`

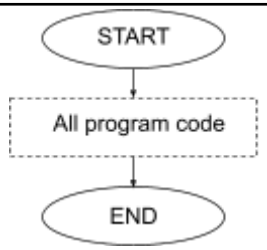
- You can combine Booleans using Logical Operators

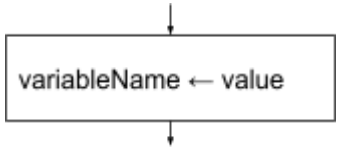
Logical Operators	Pseudocode	Python	equivalent to
`AND`	`(x < y) AND (y < z)`	`(x < y) and (y < z)`	`x < y < z`
`OR`	`(x < y) OR (x > 2*y)`	`(x < y) or (x > 2*y)`	`not (y <= x <= 2*y)`
`NOT`	`NOT (x < y)`	`not (x < y)`	`x >= y`

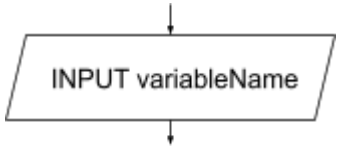
- Comments are preceded by two forward slashes // in pseudocode & Java and an octothorpe # in Python
    - Multiline comments are just a sequence of single line comments
- Lines inside blocks are indented by **four** spaces
- In Pseudocode, line continuations are denoted by a **two** space indentation
    - this includes the THEN and ELSE clauses in IF statements and the cases in a CASE statement
- Line numbers are used in examples where you need to refer to lines.
    - Line numbers are consecutive, unless numbers are skipped to indicate that part of the code is missing
- All code is contained between terminators
- Sequence is indicated by arrows between flowchart elements or by subsequent lines of code

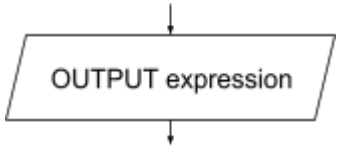
Translation Table

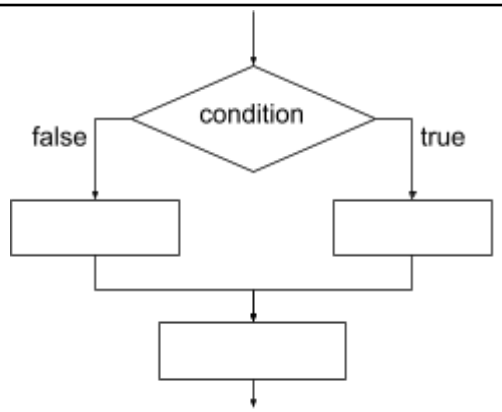
Type	Flowchart	Pseudocode	Python

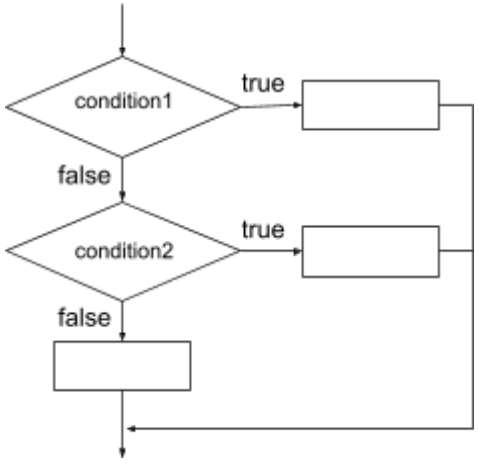
Terminators		`START`
		`    // All program code`
		`END`
			`# For the main function in Python,`
			`# just type your program code.`
			`# No need for START-END or any`
			`# public static void main() type things…`
			`# There is the if __name__ == "__main__" construct,`
			`# but it won't be needed in this course.`

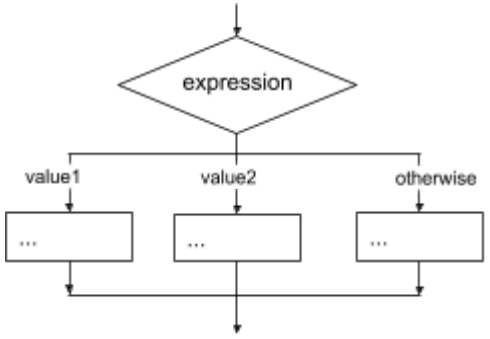
Assignment
(Data types do not need to be specified)
		`variableName ← value`	`variable_name = value`
			`# "basic" types: int, float, bool, str`
			`# n.b. Python has no char type`
			`type(variable_name) # returns the type`

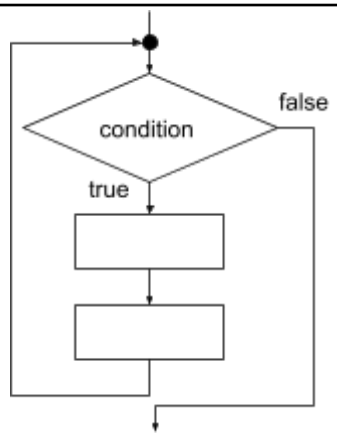
Input		`READ variableName`
		`// Or (just pick one and stick to it)`
		`INPUT variableName`
			`# input a string - note the built-in prompt`
			`variable_name = input("Prompt: ")`
			`# Or, e.g., convert to an integer (or float)`
			`num = int(input("How many? "))`

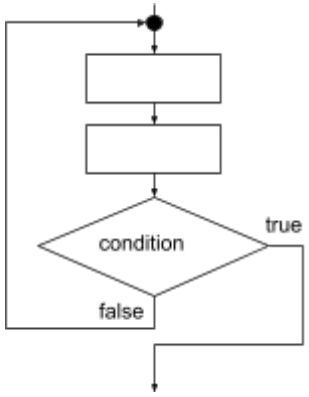
Output		`PRINT expression`
		`// Or (just pick one and stick to it)`
		`OUTPUT expression`
			`print(variable_name)`
			`print(num)`

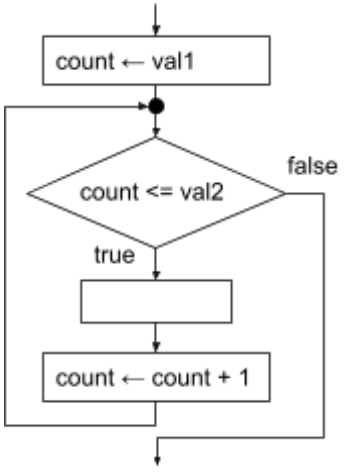
Selection (Binary)
If Statements
		IF condition
		  THEN
		  // code if true
		  ELSE
		  // code if false
		ENDIF
			if condition:
			  # code if condition is true
			else:
			  # code if condition is false
			
			# Consistent indentation is compulsory in
			# Python code. Note the common syntax of
			# **keyword expression:**
			# **indented**
			# **code block**

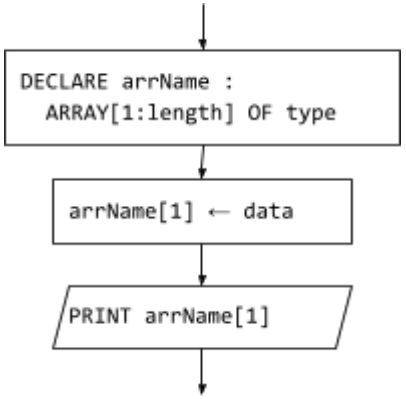
Nested If Statements used like Else-If
(The official pseudocode does not allow for the flattened **else if** structure of most programming languages)
		IF condition1
		  THEN
		  // code if 1st cond is true
		  ELSE
		  IF condition2
		  THEN
		  // code if 2nd cond true
		  ELSE
		  // code if both false
		  ENDIF
		ENDIF
			if condition1:
			  # code if 1st cond is true
			elif condition2:
			  # code if 2nd cond is true
			else:
			  # code if both false

Selection (Multiway)
Case Statements
		CASE OF variableName
		  value1:
		  // code 1
		  value2:
		  // code 2
		  // ...
		  OTHERWISE : // code
		ENDCASE
			There is no case/switch statement in Python.
			The recommendation is to use a
			if-elif-elif-...-else sequence or maybe use a
			dictionary lookup (not in this course)
			
			There is now PEP 634; a structural pattern
			matching match-case construct. See tutorial.

Iteration (Pre)
Pre-condition loops (WHILE-DO ENDWHILE)
 	
Zero or more times
		`WHILE condition DO`
		`    // code block`
		`    // inside loop`
		`ENDWHILE`
			`while condition:`
			`    # code block`
			`    # inside loop`

Iteration (Post)
Post-condition loops (REPEAT UNTIL)
		`REPEAT`
		`    // code block`
		`    // inside loop`
		`UNTIL condition`
			`There is no Do-WHILE or REPEAT-UNTIL loop in Python. Instead, use an infinite while loop with an explicit break at the end.`
			
			`while True:`
			`    # code block`
			`    # inside loop`
			`    if condition: break`

Iteration (Count)
Count-controlled loops (FOR)
		`FOR count ← value1 TO value2`
		`    OUTPUT count`
		`    // code block`
		`    // inside loop`
		`NEXT count`
			`for count in range(value1, value2+1):`
			`    print(count)`
			`    # code block inside loop`
			
			`# note: range(a, b) ~ [a, a+1, ..., b-2, b-1]`
			`# e.g., range(0, 5) ~ [0, 1, 2, 3, 4]`
			`# shorthand: range(5) == range(0, 5)`
			`# this is so that len(range(a, b)) = b - a`
			`# and range(a, b) ~ range(a, c) + range(c, b)`
			`# (no duplicate of the middle term)`

<table>
<tr>
<td rowspan="2">Arrays
(Can index from 1 or 0)</td>
<td rowspan="2"> 
</td>
<td colspan="2">

```
DECLARE ages : ARRAY[1:8] OF Integer // 8 slots to store integer ages
ages[1] ← 15
ages[2] ← 14
PRINT ages[1] + ages[2]
```

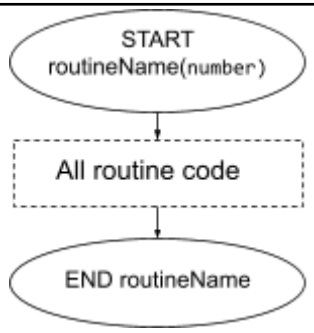
</td>
</tr>
<tr>
<td>

Python uses Lists, not Arrays (although arrays are available). Lists can contain elements of multiple types and are not of fixed length. Technically, they are Dynamic arrays of object refs

</td>
<td>

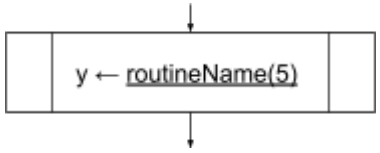
```
stuff = [1, 1.5, "ab"] # int, float, str
stuff[0] = 42          # set the 1st item
print(stuff[0])        # print 1st item

Can initialise empty arrays like
numbers = [0]*10
strings = [""]*10
```

Don’t use list-style features in your pseudocode. I.e., use a fixed data type and fixed size!

</td>
</tr>
<tr>
<td>Recommended Extension:
Define a Subroutine
(Also known as functions, methods, etc…)</td>
<td> 
</td>
<td colspan="2">

```
START routineName(number)
     // More code
     RETURN number*number
END
```

```
# A simple example of a function that
# takes and returns an integer
def routine_name(number):
    # All function code
    return number*number
```

</td>
</tr>
<tr>
<td>Recommended Extension:
Use or Call a Subroutine</td>
<td> 
</td>
<td colspan="2">

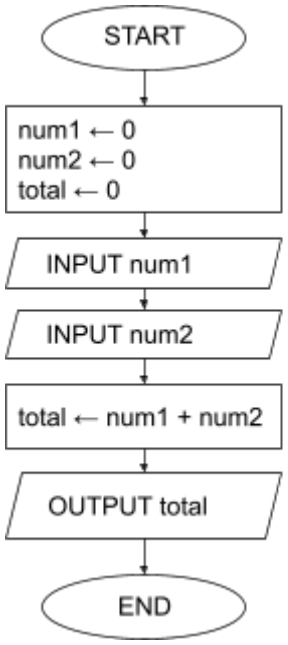
```
y ← routineName(5)
```

```
y = routine_name(5)
```

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

Examples

Add Two Numbers (a simple example with only Sequence, no branching)



These variables don't need initialising, but it is a good habit to have.

```
BEGIN
   num1  ← 0
   num2  ← 0
   total ← 0
   // Get the two numbers
   INPUT num1
   INPUT num2
   // Calculate & output their total
   total ← num1 + num2
   OUTPUT total
END
```

Note that in flowcharts and pseudocode, you often skip the niceties like prompting for input and contextualising output.

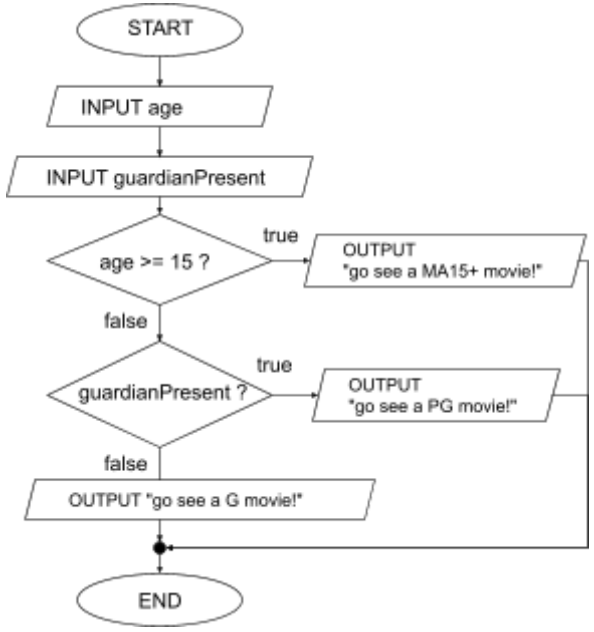
```
# Optionally initialize variables
num1, num2, total = 0, 0, 0

# Prompt and input
num1 = int(input("Enter a whole number: "))
num2 = int(input("Enter another whole number: "))

# Process
total = num1 + num2

# Output
print("Their sum is", total)
```

Movie Ratings: G, PG, MA15+ (Nested IF)

In Australia (http://www.classification.gov.au/Guidelines/Pages/Guidelines.aspx)

G is for general audience
PG is parental guidance recommended for children less than 15 years of age
(M is not recommended for children less than 15 years of age -- not used in this example)
MA15+ is legally restricted to persons 15 years of age and older

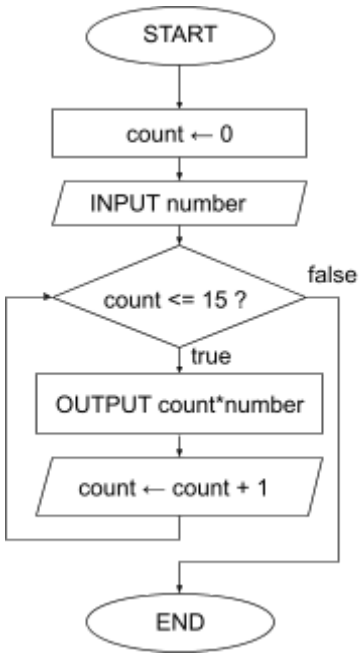
The algorithm below is probably a little simplistic for really choosing which movie to see…




```
BEGIN
    INPUT age // Integer
    INPUT guardianPresent // Boolean
    IF age >= 15
      THEN
        OUTPUT "Go see a MA15+ movie!"
      ELSE
        IF guardianPresent
          THEN
            OUTPUT "Go see a PG movie!"
          ELSE
            OUTPUT "Go see a G movie!"
      ENDIF
    ENDIF
END
```

```
age = int(input("Enter your age in years: "))
guardian_present = input("Is your guardian present Y/N? ")

if age >= 15:
    print("Go see a MA15+ movie")
elif guardian_present.upper() == "Y":
    print("Go see a PG movie")
else:
    print("Go see a G movie")
```

Print a Multiplication Table up to 15s (Using count controlled iteration)



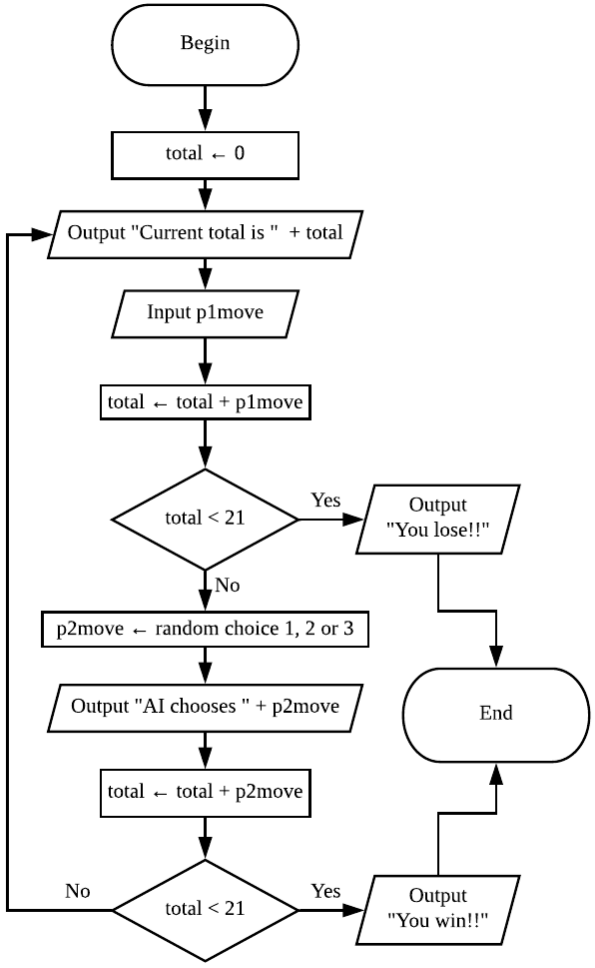

```
BEGIN
      INPUT number
      FOR multiple ← 0 TO 15:
          OUTPUT multiple*number
      NEXT multiple
END

// Alternative using repeated addition

BEGIN
      total = 0
      count = 0
      INPUT number
      WHILE count <= 15 DO
          OUTPUT total
          total = total + number
          count = count + 1
      ENDWHILE
END
```

```
number = int(input("Which times tables do you want? "))
for multiple in range(16):
    print(multiple*number)



# An Alternative using repeated addition and a while loop

total = 0
number = int(input("Which times tables do you want? "))
count = 0
while count <= 15:
    print(total)
    total = total + number
    count = count + 1
```

21 Game - Version 1

In this game the computer and the human player take turns giving a number between 1 and 3 which is then added to the total. The first one to bring the total **to or above** 21 wins the game. In this implementation, the computer chooses its number completely randomly. This is a variant of Nim (wiki).

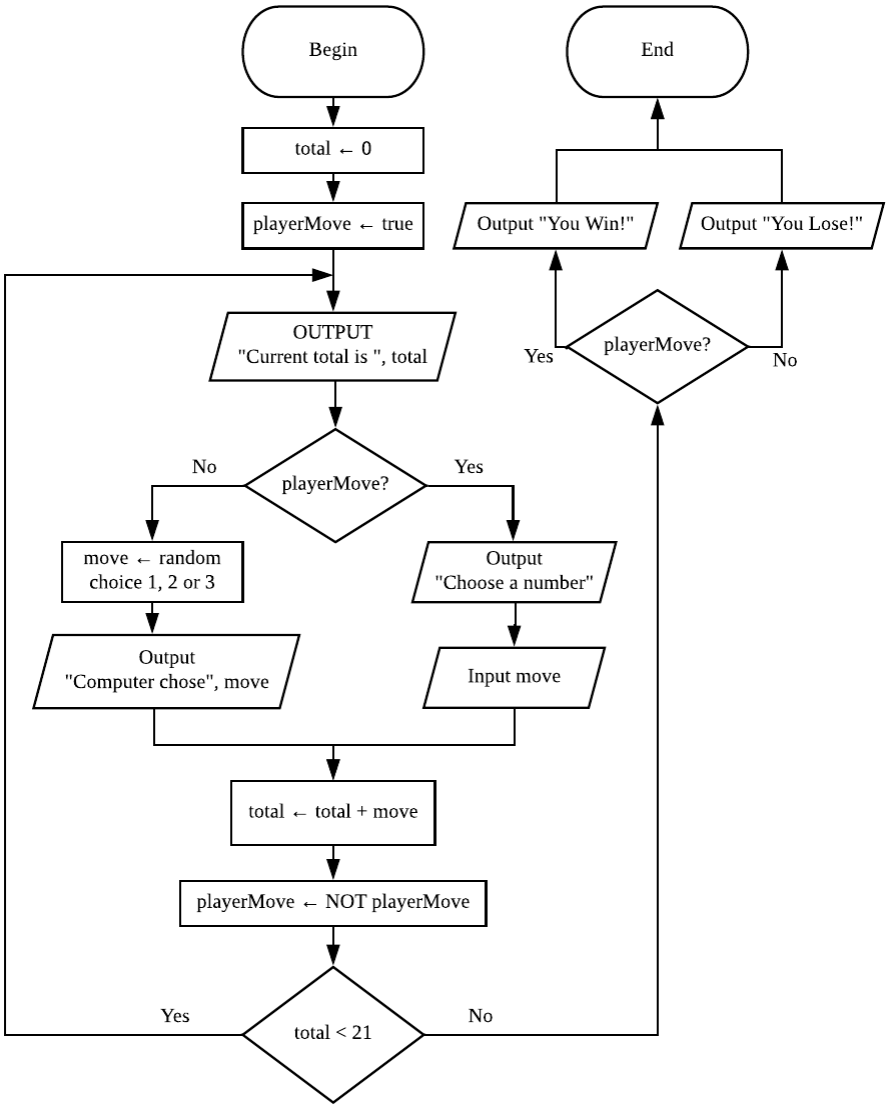
Note: There are two exit points for this loop, so it is probably most neatly implemented in Pseudocode using Break statements.
Alternatively, the computer move could be completely contained in the else clause of the first if statement...




```
BEGIN
      total ← 0
      WHILE true DO // loop until a break statement
            // 1, 2 or 3 - TODO add data validation
            OUTPUT "Total is ", total
            OUTPUT "Enter a number (1, 2 or 3) "
            INPUT p1move
            total ← total + p1move
            IF total >= 21
              THEN
                  OUTPUT "You lose!!"
                  BREAK
              END IF
            // TODO add smarter AI
            p2move ← random 1, 2, 3
            OUTPUT "computer chooses ", p2move
            total ← total + p2move
            IF total >= 21
              THEN
                  OUTPUT "You win!!"
                  BREAK
            END IF
      END WHILE
END
```

21 Game - Version 2

The previous version of the game had both the player and the computer move in each loop. This version treats the two players more symmetrically and only has one move per loop, the Boolean **flag** is used to track whose move it is. This would allow for a more flexible program where you can swap in different interfaces for human players or AIs for computer players.




```
BEGIN
   total ← 0
   playerMove ← true // Flag for player or computer turn

   REPEAT
      OUTPUT "Current total is ", total
      IF playerMove
        THEN
          OUTPUT "Choose a number (1, 2 or 3): "
          INPUT move
        ELSE // (not playerMove => computer's turn)
          move ← random number 1, 2, 3
          OUTPUT "Computer chose ", move
      END IF
      total ← total + move
      playerMove ← NOT playerMove
   UNTIL total >= 21

   IF playerMove
     THEN
      OUTPUT "You Win!"
     ELSE
      OUTPUT "You Lose!"
   END IF
END
```