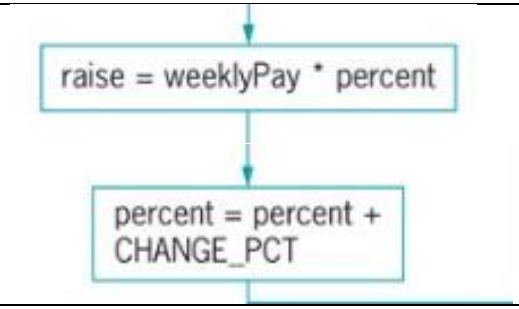
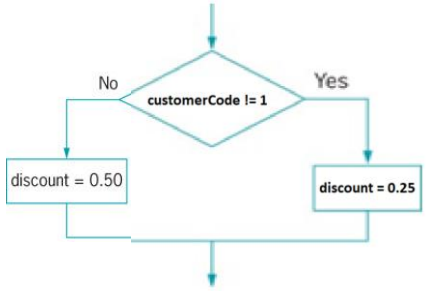
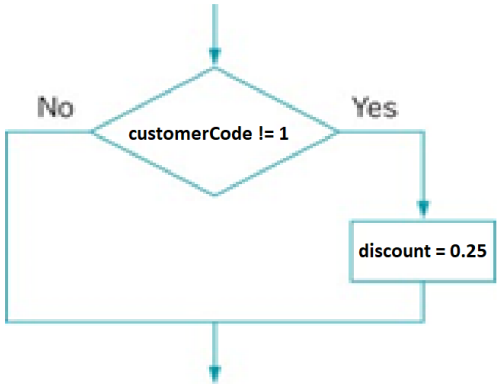
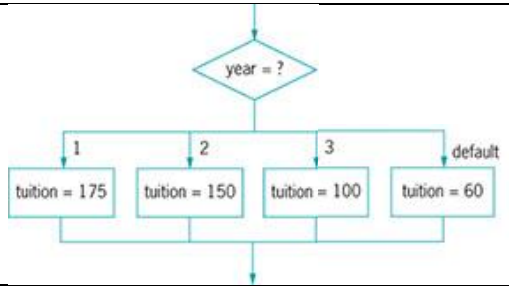
Cheat sheet: Intro into Programming Logic

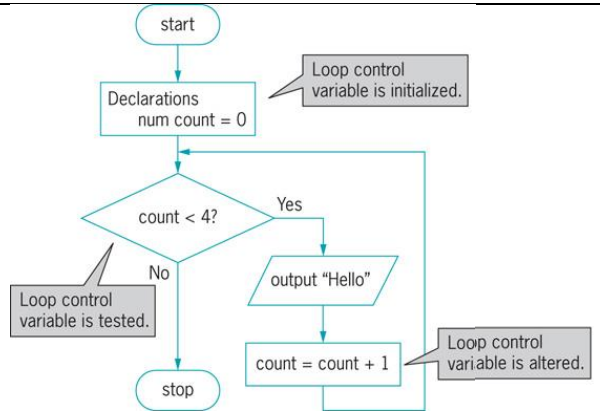
	Pseudocode	Flowchart	Key Elements
Sequence	`raise = weeklyPay * percent` `percent = percent + CHANGE_PCT`		

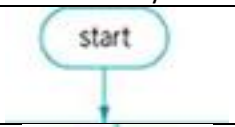
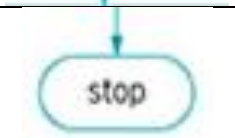
Selection	`if customerCode != 1`
	`  discount = 0.25`
	`else`
	`  discount = 0.5`
	`endif`
			• Starts with “if”
			• Has “else”
			• Ends with “endif”
			• Check indentation

	`if customerCode != 1`
	`  discount = 0.25`
	`endif`
			• Starts with “if”
			• End with “endif”
			• Check indentation

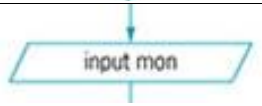
Switch	`case year`
	`  1: tuition = 175`
	`  2: tuition = 150`
	`  3: tuition = 100`
	`  default: tuition = 60`
	`endcase`
			

Cheat sheet: Intro into Programming Logic

Loop	<pre>start
	  Declarations
	  num count = 0
	  while count < 4
	  output "Hello"
	  count = count + 1
	  endwhile
	stop</pre>
			• Initialization
			• Evaluation
			• Incrementing or decrementing
			• Infinite loop never reaches an exit condition

	Pseudocode	Flowchart symbol	
start	start		Explicitly state start
end	end		Explicitly state end

Declaration	<pre>num mon
	string MONTH[12] = "January", "February",
	"March", "April", "May", "June", "July",
	"August", "September", "October",
	"November", "December"</pre>
			• Explicitly state type of every variable
			• Upper case the entire variable name of constants

Input	Input mon		Explicitly state input
output	Output MONTH[mon]		Explicitly state output

Cheat sheet: Intro into Programming Logic

1-Dimensional Arrays

Num NUM_OF_ITEMS = 3 //constant

Declaration: type variableName[Size]

num itemPrice[NUM_OF_ITEMS]

Initialization: variableName[index]

Index = element_num – 1 // index starts at 0

itemPrice[0] = 5

itemPrice[1] = 65

itemPrice[2] = 30

Declaration & Initialization

num itemPrice = [5,65,30]

Accessing array elements

Output itemPrice[0] // 5

Output itemPrice[1] // 65

Output itemPrice[2] // 30

itemPrice			
Element	1st	2nd	3rd
Index	0	1	2
Value	5	65	30

Using a loop to print out elements in 1D array

```
start
    Num NUM_OF_ITEMS = 3
    num itemPrice[NUM_OF_ITEMS]
    itemPrice[0] = 5
    itemPrice[1] = 65
    itemPrice[2] = 30
    num index = 0

    while index < NUM_OF_ITEMS
        output itemPrice[NUM_OF_ITEMS]
        index = index +1
    endwhile
end
```

2-Dimensional Arrays

Num HORIZONTAL = 2 //constant

Num VERTICAL = 3 //constant

Declaration: type variableName[row size][column size]

num timeTable[HORIZONTAL][VERTICAL]

Initialization: variableName[row][column]

timeTable [0][0] = 1

timeTable [0][1] = 2

timeTable [0][2] = 3

timeTable [1][0] = 2

timeTable [1][1] = 4

timeTable [1][2] = 6

Declaration & Initialization

num timeTable = [

[1,2,3],

[2,4,6]

] //end of 2D array

Accessing array elements

Output timeTable [0][0] // 1

Output timeTable [0][1] // 2

Output timeTable [0][2] // 3

Output timeTable [1][0] // 2

Output timeTable [1][1] // 4

Output timeTable [1][2] // 6

timeTable			
Column/ Row	0	1	2
0	1	2	3
1	2	4	6

Cheat sheet: Intro into Programming Logic

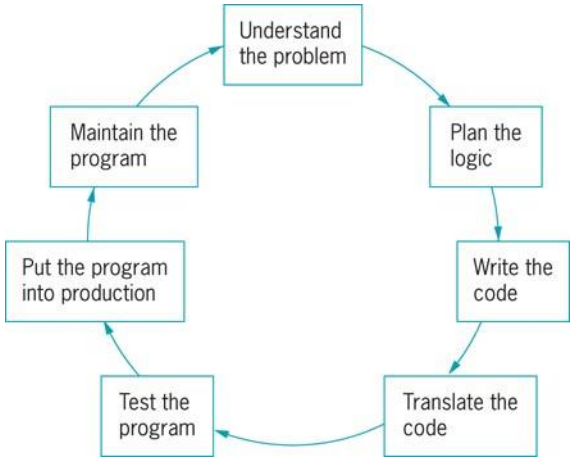
Using nested loops to initialize elements in a 2D array

```
start
    Num HORIZONTAL = 3 //constant
    Num VERTICAL = 2 //constant
    num timeTable[VERTICAL][ HORIZONTAL]
    itemPrice[0][0] = 1
    itemPrice[0][1] = 2
    itemPrice[0][2] = 3
    itemPrice[1][0] = 2
    itemPrice[1][1] = 4
    itemPrice[1][2] = 6

    num row = 0

    while row < VERTICAL
        num column = 0
        while column < HORIZONTAL
            timeTable[row][column] = (row+1)(column+1)
            column = column + 1
        endwhile
        row = row +1
    endwhile
end
```

Program development cycle



Arithmetic Operations			
Action	Sign	Example	Output
Add	+	5+3	8
Subtract	-	5-3	2
Multiply	*	5*2	10
Divide	/	12/3	4
Remainder	%	12%5	2

Logical Operators

op	*meaning*	*true*	*false*
==	*equal*	2 == 2	2 == 3
!=	*not equal*	3 != 2	2 != 2
<	*less than*	2 < 13	2 < 2
<=	*less than or equal*	2 <= 2	3 <= 2
>	*greater than*	13 > 2	2 > 13
>=	*greater than or equal*	3 >= 2	2 >= 3

Boolean operators

values	*true or false*		
literals	`true`	`false`	
operations	and	or	not
operators	&&	||	!