# 5. STRUCTURED PROGRAMMING

When you create your own programs, it is important to structure them so that they are:

1. Easy to read
2. Easy to understand
3. Easy to maintain

Programming problems are easier to solve by breaking them down into a series of smaller steps which are easier to understand and solve. The total solution is created when all the smaller sub-problems have been solved.

The three constructs that we use in structured programming are:

1. Sequence
2. Selection
3. Iteration

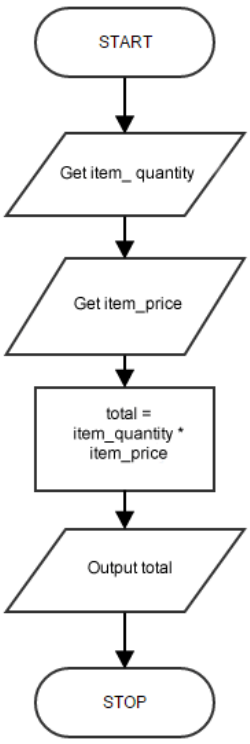
## 5.1 SEQUENCE

In structured programming, the SEQUENCE in which instructions are executed is the same order as they appear in the code:

```
# Sequence Example

item_quantity = int(input("Please enter the quantity required: "))
item_price = float(input("Please enter the item price: "))
total = item_quantity * item_price
print("The total cost is £{0}.".format(total))
```

When trying to solve problems we can also represent them using either pseudocode or diagrams:

Flow chart:




Your programs use **sequencing** when the solution can be broken down into a series of steps that are processed one after another.

Pseudocode:

```
item_quantity ← INPUT
item_price ← INPUT
total ← item_quantity * item_price
OUTPUT 'The total cost is £',total
```

## RELATIONAL OPERATORS

| Operator | What it means | Example |
|---|---|---|
| == | Equality operator; checks whether both values are the same | `>>> 7 == 6`<br>`False` |
| != | Not equal to | `>>> 8 != 2`<br>`True` |
| > | Greater than | `>>> 65 > 12`<br>`True` |
| < | Less than | `>>> 21 < 12`<br>`False` |
| >= | Greater than or equal to | `>>> 15 >= 12`<br>`True` |
| <= | Less than or equal to | `>>> 34 <= 35`<br>`True` |

## LOGICAL OPERATORS

| Operator | What it means | Example |
|---|---|---|
| AND | Logical AND checks whether both conditions are true or false | `>>> x = 6`<br>`>>> x > 0 and x < 7`<br>`True` |
| OR | Logical OR checks whether EITHER of the conditions is true | `>>> x = 5`<br>`>>> y = 8`<br>`>>> x/2 == 2 or y/4 == 2`<br>`True` |
| NOT | Logical NOT reverses a Boolean value. In the example, x > y evaluates to False, using the logical NOT reverses the evaluation to true. | `>>> x = 5`<br>`>>> y = 8`<br>`>>> not(x > y)`<br>`True` |

## EXERCISE 07: RELATIONAL AND LOGICAL OPERATORS

Use print statements to find the answers to the following:

1. Calculate the following (True or False)
   a. 23!=15 b. 5+3<10 c. 6 > 10 == 10< 2
2. If a = 3 and b = 8, what are the results of the following statements?
   a. a<b b. 6 >= a c. b > a == False d. True !=(a==b)
3. If c = True and d = False, what are the results of the following statements?
   a. c and d b. not d or c c. c == d and True

## 5.2 SELECTION

### IF STATEMENT

Selection or conditional statements execute a block of code based on the result of some test or condition we have set in the code. We are testing to see whether the result is TRUE or FALSE and we can set different actions depending on the result of our test.

Example:

```
if 76 > 23:
    print("76 is greater than 23")

if 15 > 23:
    print("15 is greater than 23")
```

```
>>>
76 is greater than 23
>>>
```

In this example, the test in the first block of code evaluates to **true** so the print statement is executed. The test in the second example evaluates to **false** so the print statement is NOT executed.

*Note: Remember that it is important that your code is correctly indented to avoid syntax errors.*

This can be used in simple examples like this, where we are asking for input from the user and checking this against a present condition in our code:

```
mark = int(input("Enter test score: "))
if mark >= 50:
    print("Pass")
else:
    print("Test failed, resit please")
```

```
Enter test score: 57
Pass
>>>
```

What happens if I enter a value below 50?

We need the code to be able to deal with BOTH **true** and **false** inputs.

### IF ELSE STATEMENT

Using an If Else statement, I can now include a false code block so that something happens if the test condition does not evaluate to true.

Example:

```
mark = int(input("Enter test score: "))
if mark >= 50:
    print("Pass")
else:
    print("Test failed, resit please")
```

```
Enter test score: 49
Test failed, resit please
>>>
```

I may want my code to check several conditions and proceed with the condition which is true. For example, a different score in the test used above will result in a different grade.

### IF/ELIF STATEMENT

```
mark = int(input("Enter test score: "))

if mark <= 49:
    print("Grade D: Please attend resit")
elif mark > 50 and mark < 56:
    print("Grade C-needs improvement")
elif mark >= 56 and mark < 65:
    print("Grade B-good work")
elif mark >= 65 and mark < 70:
    print("Grade A-well done")
else:
    print("Grade A*- excellent!")
```

The structure of the IF/ELIF statement should follow these rules:

```
If Condition 1 = True:
    execute Code 1
elif Condition 2 = True:
    execute Code 2
else:
    execute Code 3
```

You can test any number of conditions using this method; your code does not have to include an ELSE but, if it does, it must be the last statement.

### NESTING IF STATEMENTS

We can also use IF/ELIF statements to check sub-conditions in a program:

```
x = 5
y = 8

if x == y:
    print("x and y are equal")
else:
    if x < y:
        print("x is less than y")
    else:
        print("x is greater than y")
```

```
>>>
x is less than y
>>>
```

In this example, condition 1 evaluates to false so the else part of the IF statement is executed. Condition 2, in the first part of the nested if statement, evaluates to true, as 5 is less than 8, and the print statement is executed.

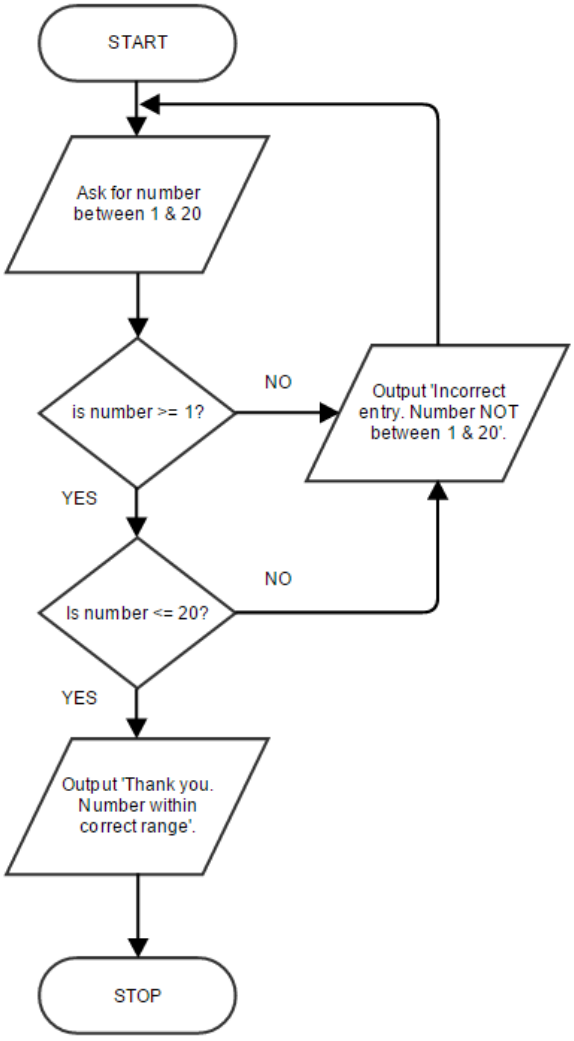
## EXERCISE 08: IF/ELIF STATEMENTS

1. Complete the program shown in the flow chart.

2. Write a **program** which asks for the names of two football teams playing against each other and their scores.

   Your program should calculate how many points each team gets (3 for a win, 1 for a draw, 0 if they lose).

   *Hint: you will need to declare constants to use in calculating the points. Remember, one team will be the HOME team, the other the AWAY team.*

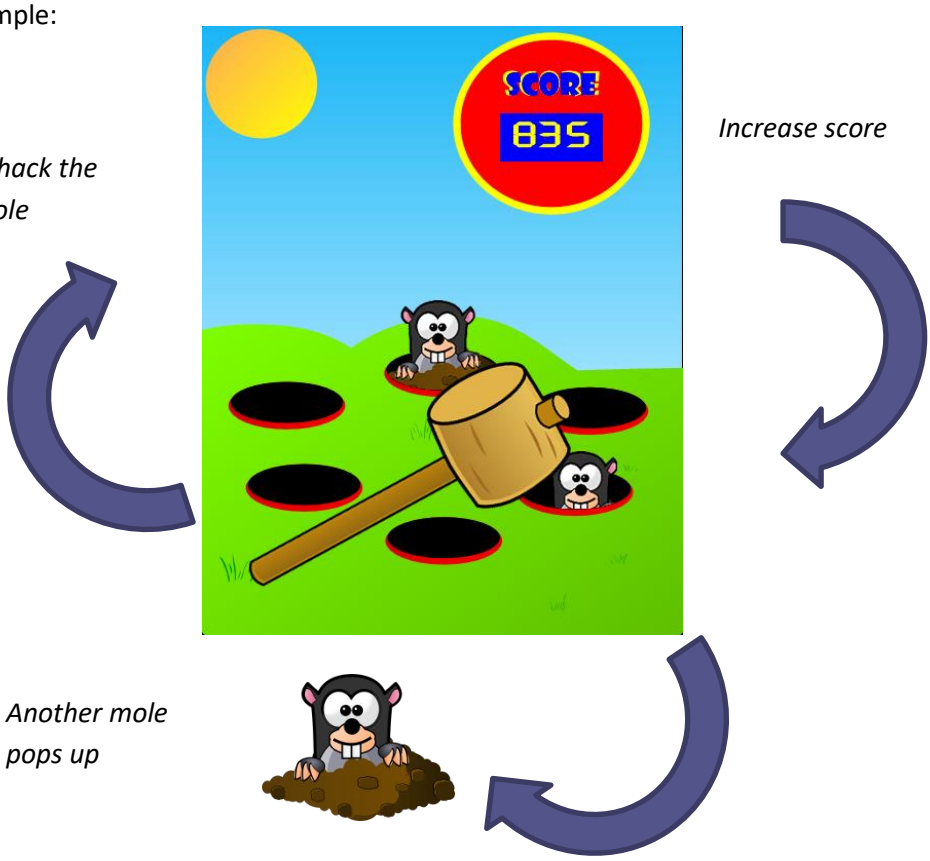


## 5.3 ITERATION

In programming, iteration means repeating instructions or processes over and over again. This is more commonly known as 'looping'.

Example:




If you do not hit the mole within a certain time limit, the game is over.

In Python, there are two types of loop that can be used, a FOR loop and a WHILE loop.

### WHILE LOOPS

While loops are also known as conditional loops as they will continue to iterate or loop until a condition, which you have set in your code, is met. It is important to make sure that you have written code that will allow your loop to finish and avoid an infinite loop.

Example:

```
def numberLoop():
    """ while loop example"""
    number = 1           # initial value of the variable
    while number <= 10: # the condition to exit the loop
        print(number)
        number += 1      # increments the value of 'number'

numberLoop()
```

If the value of the variable 'number' is not incremented by 1 each time the loop iterates, the condition to exit the loop will never be reached as 'number' will always be less than 10.

I can also check a condition entered by a user:

```
def while_Input():
    """while loop example input"""
    answer = 'n'
    while answer != 'y':
        answer = input("Are we there yet? Enter y/n: ").lower()
    print("At last!")

whileInput()
```

```
>>>
Are we there yet? Enter y/n: Almost
Are we there yet? Enter y/n: Nearly
Are we there yet? Enter y/n: Very soon
Are we there yet? Enter y/n: Y
At last!
>>>
```

As you can see, the loop continues until the condition is met, and the final print statement is then executed. Remember that use of the .lower() built-in function changes my capital letter Y into lower-case y.

## COUNTING AND TOTALLING WITH A WHILE LOOP

We can use a while loop to iterate (loop) and count up (increment) or count down (decrement) until a preset condition is true or false.

Example:

```
1    # print 5 times table
2
3    i = 1
4    while i <= 12:
5        print(i * 5)
6        i += 1
```



```
5
10
15
20
25
30
35
40
45
50
55
60
```

It is important to set the initial value of 'i' on Line 3. The preset condition for exiting the while loop is then set on Line 4 with the incrementing value of i set on Line 6 with Line 5 printing out the resulting 5 times table.

Example 2:

```
1    # count down example
2
3    x = 10
4    while x >= 0:
5        print("Count... " + str(x))
6        if x == 0:
7            print("Lift off!")
8        x -= 1
```

```
Count... 10
Count... 9
Count... 8
Count... 7
Count... 6
Count... 5
Count... 4
Count... 3
Count... 2
Count... 1
Count... 0
Lift off!
```

In this example of a countdown, the initial value of x, on Line 3, must be greater than 0 as we are counting down from 10. The condition for exiting the while loop is set on Line 4 and the value of x is concatenated into a print statement on Line 5.

An additional check on Line 6 checks when the count has reached 0 for an extra print statement and the value of x is decremented on Line 8.

Example 3:

```
1    # adding to a total
2
3    total = 0
4    x = 0
5
6    while x <= 10:
7        total = total + x
8        print("x = " + str(x) + "  Total = " + str(total))
9        x += 1
```

```
x = 0  Total = 0
x = 1  Total = 1
x = 2  Total = 3
x = 3  Total = 6
x = 4  Total = 10
x = 5  Total = 15
x = 6  Total = 21
x = 7  Total = 28
x = 8  Total = 36
x = 9  Total = 45
x = 10  Total = 55
```

In this example, the while loop counts from 0 to 10; look at Line 6. In Line 7, the value of x is added to the running total and the results are printed on Line 8. The value of x is then incremented by 1 on Line 9.

The value of x can be incremented by any value; for example, if you want to loop through from 0 to 25 in steps of 5 you would do this:

```
1    # increment by 5
2
3    x = 0
4    while x <= 25:
5        print(x)
6        x += 5
```

```
0
5
10
15
20
25
```


We can also use a while loop to check whether a valid input has been entered; this is an ideal way to ensure that your code is robust.

*Screenshot shown in PyCharm IDE to show line numbers*

```
4   def menu():
5       """Display menu choice: Enter Player Names, Play Game ,Quit"""
6
7       menu_options =['E', 'P', 'Q']
8       invalid_menu_choice = True
9       print('''Please choose from the following options:\n
10          \tPress'E' to enter players names\n
11          \tPress 'P' to play the game\n
12          \tPress 'Q' to quit\n''')
13      menu_choice = input('>> ').upper()
14
15      while invalid_menu_choice:
16          if menu_choice in menu_options:
17              invalid_menu_choice = False
18          else:
19              print('That was not in the menu,\nplease choose E,P or Q to continue')
20              menu_choice = input('>> ').upper()
21
22      return menu_choice
23
24
25  def main():
26      """runs all functions"""
27
28      menu_choice = menu()
29      if menu_choice == 'E':
30          print("You have chosen to enter player names")
31      elif menu_choice == 'P':
32          print("You have chosen to play the game")
33      else:
34          print("You have chosen to quit the game")
35
36
37  main()
```

```
>> x
That was not in the menu,
please choose E,P or Q to continue
>> y
That was not in the menu,
please choose E,P or Q to continue
>> q
You have chosen to quit the game
>>>
```

- On Line 8, the variable 'invalid_menu_choice' is set to True
- On line 15, the while loop will continue while the value of the variable remains True. This could also be written as:
    - while invalid_menu_choice == True:
- Line 16 checks whether the input is in the list 'menu_ options'. If that condition evaluates to True then a valid menu choice has been entered.
- Line 17 then changes the value of the variable 'invalid_menu_choice' to False so that the program will now stop looping through lines 15 to 20.

## FOR LOOPS

The for loop will loop for a set number of times, which can be a number range, e.g. from 1 to 10, or items in a sequence, such as a string or a list.

A common way to use a for loop with numbers is to use the range() built-in function. When we use this built-in function we usually supply the starting number in the range and the number to go up to, **but not include** in the range. Look at these examples:

```
for x in range(6):
    print(x)
```

```
>>>
0
1
2
3
4
5
>>>
```

```
for x in range(1,11):
    print(x)
```

```
>>>
1
2
3
4
5
6
7
8
9
10
>>>
```

```
for x in range(1,11,2):
    print(x)
```

```
>>>
1
3
5
7
9
>>>
```

The first example has no starting point for the range of numbers so uses the default of 0 to print out six numbers from 0 to 5.

In the second example, I have supplied the starting point 1 and the end of the range as 11; this means up to, BUT not including, the last number.

The third example sets the start and end of the range but also specifies that the numbers must increase in steps of 2.

## COUNTING AND TOTALLING WITH FOR LOOPS

We can use the same techniques as counting and totalling with the while loop in a for loop.

```
1   # counting with a for loop
2
3   total = 0
4
5   for i in range(5):
6       total = total + i
7       print(" Value of i = " + str(i) + " Total = " + str(total))
```

```
Value of i = 0 Total = 0
Value of i = 1 Total = 1
Value of i = 2 Total = 3
Value of i = 3 Total = 6
Value of i = 4 Total = 10
```

Using the range() method, we already know it will iterate up to, but not including, 5. Line 6 simply adds the current value of i to the running total.

Another common method of iteration and counting or totalling uses arrays. Look at this example:

```
1   # counting with a for loop
2   items = 0
3   shopping = ["eggs", "milk", "bread", "cheese", "jam"]
4
5   for each in shopping:
6       items += 1
7   print("Number of items in shopping = " + str(items))
```

The variable 'each' is used to iterate through the array 'shopping'. Each time the FOR loop iterates, the value of 'items' is incremented by 1.

This method can also be used to count the characters in a string. Look at this example:

```
1   # counting with a for loop
2
3   word = "programming"
4   vowels = ['a', 'e', 'i', 'o', 'u']
5   v_count = 0
6   l_count = 0
7
8   for letter in word:
9       l_count += 1
10      if letter in vowels:
11          v_count += 1
12  print("The word has " + str(l_count) + " letters and " + str(v_count) + " vowels")
```

```
The word has 11 letters and 3 vowels
```

### FOR LOOPS USING STRINGS AND LISTS

Sometimes we want to iterate over data structures like a string or a list. We can do this by using a variable 'i' or 'item' to iterate over the characters in the string or the elements in the list/array.

```
word = "programming"

for i in word:
    print(i)
```

```
>>>
p
r
o
g
r
a
m
m
i
n
g
>>>
```

```
shopping = ["eggs","milk","bread","cheese","jam"]

for item in shopping:
    print(item)
```

```
>>>
eggs
milk
bread
cheese
jam
>>>
```

### NESTED LOOPS

Sometimes we want to loop through two sets of integers to compare them.

**Example 1:**

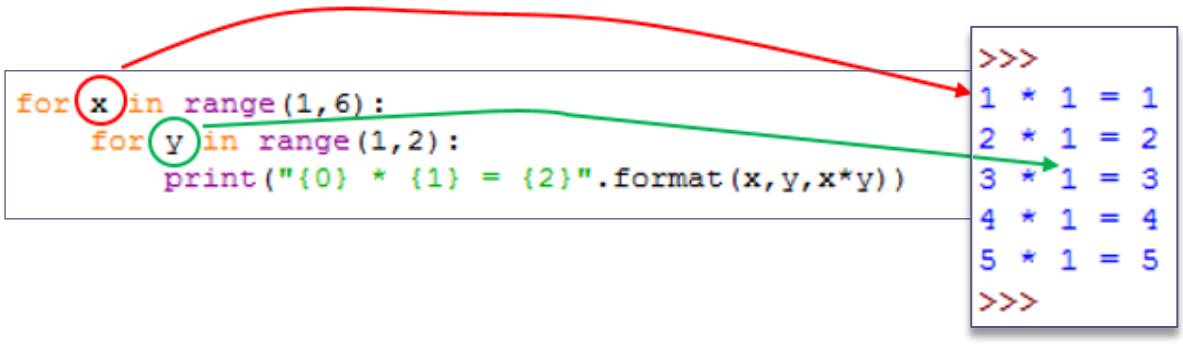
```
nums1 = [5,3,7,9,15,12]
nums2 = [7,5,9,11,16,8]

for x in nums1:
    for y in nums2:
        if x == y:
            print("{0} in both arrays".format(x))
```

```
>>>
5 in both arrays
7 in both arrays
9 in both arrays
>>>
```

This example finds the integers that appear in both lists; although they are not in the same order, we can identify which are duplicated. We may be writing a program to find duplicate values, so these can be processed in some way to solve a problem.

**Example 2:**

Here, x is used to count through the range from 1 to 5, y is used to count through the range from 1 to 2. The string format() built-in function has been used to print out a simple times table.



**Example 3:**

This example shows how you might run a game loop in a program.

```
4   no_winner = True
5   player_go = 1
6
7   while no_winner:  # outer loop
8       while player_go == 1:    # inner loop
9           name = input("Please enter your name: ")
10          choice = input("Please enter a vowel {0}: ".format(name))
11          if choice.upper() == "I":
12              no_winner = False
13              print("Well done {0}, you have won!".format(name))
14              break
15          else:
16              print("You have not won the game, your turn is over")
17              player_go = 2    # player_go variable changed to 2
18      while player_go == 2:
19          name = input("Please enter your name: ")
20          choice = input("Please enter a vowel {0}: ".format(name))
21          if choice.upper() == "I":
22              no_winner = False
23              print("Well done {0}, you have won!".format(name))
24              break
25          else:
26              print("You have not won the game, your turn is over")
27              player_go = 1    # player_go variable changed to 1
28
29  print("Game over")
```

*Screenshot shown in PyCharm IDE to show line numbers*

The two conditions that are being checked in each while loop, no_winner and player_go, are set at the start on lines 4 and 5.

- Line 7 – the outer loop continues to check whether no_winner is still true.
- Line 8 – the inner while loop to control the turn for each player asks for their name and their choice. This is then compared with the answer on line 11.
- If the choice entered matches the answer then the game has been won.
- Line 12 – the variable no_winner is set to false, a message is printed and the 'break' command forces the code out of the inner loop.
- The condition for the outer loop is no longer true and the code jumps to line 29.
- If the choice entered does not match the answer, the 'else' part of the selection statement is executed, a message is printed on line 16 and the player_go variable is changed to 2 on line 17.
- The second inner loop then works in exactly the same way.

### BREAKING OUT OF LOOPS

A break statement will allow us to 'break' out of a while loop if a test condition is met. In the example 3 above this happens if the player correctly guesses the vowel.

Here are two more examples:

**Example 1:**

```
for i in range(0,10,2):
    if i == 6:
        break
    print(i)
```

```
>>>
0
2
4
>>>
```

This for loop should print from 0 to 8 in steps of 2 but is set to break out of the loop if the iterator 'i' is equal to 6.

**Example 2:**

```
while True:
    print("Hello world!")
    x = input(">>> ")
    if x == "x":
        break
print("You entered 'x'!")
```

```
>>>
Hello world!
>>>
Hello world!
>>>
Hello world!
>>> x
You entered 'x'!
```

In this example, the use of TRUE in the while loop means that until the input meets the criteria to break out of the loop, the program will continue to ask for an input. In the example above, no input was entered apart from hitting the ENTER key.

## EXERCISE 09: WHILE AND FOR LOOPS

1. Write a program, using at least two functions AND parameter passing, which will allow the user to input a number between 1 and 12 and print the times table for that number.

2. Write a program, using at least 2 functions AND parameter passing, that will add together a series of numbers until the user enters 0. The program will then display the total.