# Chapter 3: Control Flow

## Contents



## Syllabus Learning Outcomes

1.1 Algorithmic Representation

Write algorithms in pseudo-code and flowchart for given problems.

- 1.1.1 Use appropriate techniques or tools such as pseudo-code and flowchart to show program flow.
- 1.1.2 Use standard flowchart symbols.
- 1.1.3 Use a combination of various control structures.
- 1.1.4 Use decision tables to explore the actions for combinations of different input conditions. Note: up to three conditions

2.2 Programming Elements and Constructs

Use programming language elements and constructs to write recursive and non-recursive programs to solve a variety of problems.

- 2.2.3 Apply the fundamental programming constructs to control the flow of program execution:
  – Sequence
  – Selection
  – Iteration

# 1 What is control flow?

In the examples that we have seen till now, there has always been a series of statements faithfully executed by Python in exact top-down order. What if you wanted to change the flow of how it works? For example, you want the program to take some decisions and do different things depending on different situations, such as printing 'Good Morning' or 'Good Evening' depending on the time of the day?

As you might have guessed, this is achieved using control flow statements. There are three control flow statements in Python - if, for and while.

## 1.1 Algorithm

An algorithm is a step-by-step procedure (well-defined instructions) to solve a given problem.

For instance, the algorithm to find the sum of two numbers is

1. Take two numbers
2. Add the numbers
3. Display the result

Or the algorithm to log in to VJC portal

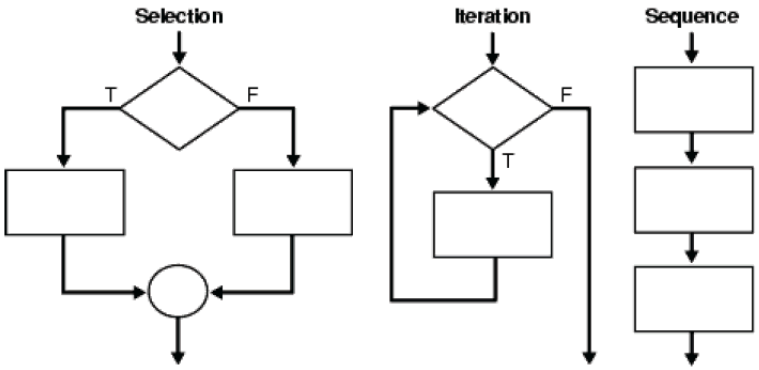
1. Go to https://portal.vjc.sg/
2. Enter vjc email and password
3. Click on Sign in button
4. Enter OTP
5. Click on Submit button

Sometimes there might be different steps depending on some other conditions. For example, if the day is a weekday, you will wake up at 6:00 am, else you will wake up at 10:00 am. Many problems have more than one solution. Sometimes it is a personal preference which solution to choose. Sometimes one solution will be measurably better than another.

We can write algorithm using pseudo code or flowchart.

## 1.2 Pseudo code

Pseudo code is a description of the steps in an algorithm using natural language (such as English) mixed with sequence, selection and iteration constructs.




Sequence construct is when a number of steps are performed one after the other.

Selection construct allows some steps to be performed under certain conditions, otherwise different steps (or no steps) are performed.

Repetition construct is when a sequence of steps is performed a number of times. This is also known as iteration or looping.

A general idea to solving a problem involves inputting data to the computer, processing the data and outputting results.

| | English | Pseudo code | |
|---|---|---|---|
| | | Example | General |
| Assignment and sequence | Set A to 34<br>Increment B | `A ← 34`<br>`B ← B + 1` | <identifier> ← <value> |
| Selection | If A is greater than B then do something, else do another thing | `IF A > B`<br>`  THEN`<br>`  Do something`<br>`  ELSE`<br>`  Do another thing`<br>`ENDIF` | `IF <condition>`<br>`  THEN`<br>`  <statements>`<br>`  ELSE`<br>`  <statements>`<br>`ENDIF` |
| Repetition | Repeat some steps until A is equal to B | `REPEAT`<br>`  Some steps`<br>`UNTIL A = B` | `REPEAT`<br>`  <statements>`<br>`UNTIL <condition>` |
| | While Num is smaller than 10, increment Num | `WHILE Num < 10 DO`<br>`  Num ← Num + 1`<br>`ENDWHILE` | `WHILE <condition> DO`<br>`  <statements>`<br>`ENDWHILE` |
| | For index 1 to 30, assign an empty string to Names | `FOR Index ← 1 TO 30`<br>`  Names[Index] ← ''`<br>`ENDFOR` | `FOR <identifier> ←`<br>`<value1> TO <value2>`<br>`  <statements>`<br>`ENDFOR` |
| Input | Take in an input and assign to A with and without a prompt | `INPUT A`<br><br>`INPUT "Enter name: " A` | `INPUT <identifier>` |
| Output | Output/print “Message”<br>Output B | `OUTPUT "Message"`<br>`OUTPUT B`<br>`OUTPUT "Your age is ", B` | `OUTPUT <values(s)>` |

A condition consists of at least one logic proposition. Logic propositions use the relational operators.

| Operator | Comparison |
|---|---|
| = | Is equal to |
| < | Is less than |
| > | Is greater than |
| <= | Is less than or equal to |
| >= | Is greater than or equal to |
| <> | Is not equal to |

The pseudo code presented in the table is the recommended style of writing pseudo code in examination papers. In the recommended style, keywords are in uppercase with indentation at appropriate parts of the algorithm.

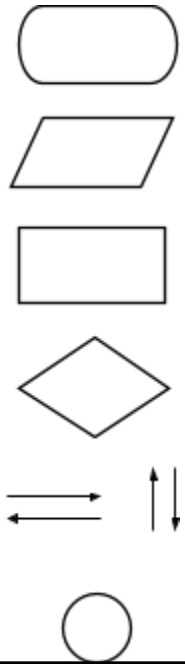
However, there is no fixed or defined syntax. It is acceptable as long as the logic correctly solves the problem.

### 1.3 Flow chart

A flowchart is the graphical or pictorial representation of an algorithm with the help of different symbols, shapes, and arrows to demonstrate a process or a program.

There are 6 basic symbols commonly used in Flowchart

(We focus only on the first 5)

| Symbols | Name | Description |
|---|---|---|
|  | Start/End | Indicates the starting or ending of the program |
| | Input/Output | Use for Input/Output(I/O) operation i.e. taking input and showing output |

| | | |
|---|---|---|
| | Process | Indicates any type of internal operations like initialization, calculation etc. |
| | Decision | Use for asking questions that can have either TRUE or FALSE (YES or NO) as an answer |
| | Control flow | Show direction of flow |
| | Connector | Connectors are used to connect breaks in the flowchart.<br><br>If a flowchart takes more than one page, then we use the connector to connect the flowchart between pages. (However, A-level questions on flowchart are usually within 1 page.) |

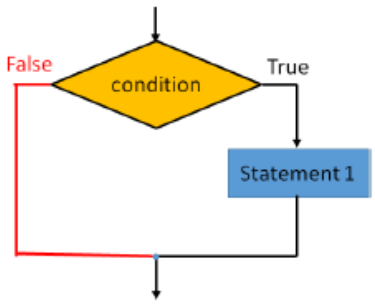
# 2 Control flow statements

There are three control flow statements in Python: `if`, `for` and `while`.

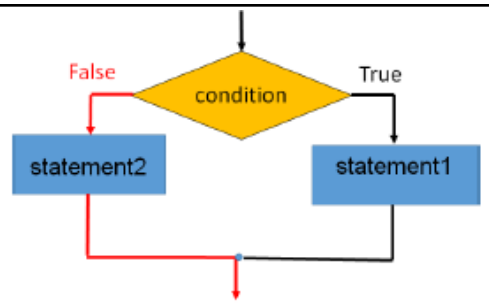
## 2.1 IF statement

The `if` statement is a selection construct that is used to check a condition: if the condition is true, we run a block of statements (called the if-block), else we process another block of statements (called the else-block). The else clause is optional.

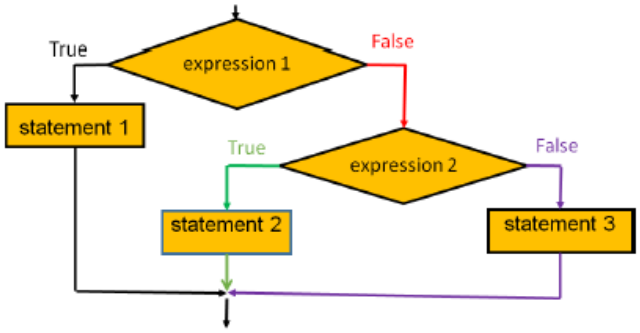
If we need to have another if-else statement in an else-block, we can simplify the code using if-elif-else statement.

| Syntax | Flowchart |
|---|---|
| `if <condition>:`<br>`  <statement>` |  |

```
if <condition>:
    <statement>
else:
    <statement>
```




```
if <condition>:
    <statement>
elif <condition>:
    <statement>
else:
    <statement>
```




```python
# Example of using if statement
a = int(input("Enter an integer: "))
b = int(input("Enter another integer: "))

if a < b:
    print('a is smaller than b!')
```

```
Enter an integer: 3
Enter another integer: 4
a is smaller than b!
```

```python
# Example of using if-else statement
a = int(input("Enter an integer: "))
b = int(input("Enter another integer: "))

if a < b:
    print('a is smaller than b!')
else:
    print('a is larger than b!')
```

```
Enter an integer: 3
Enter another integer: 2
a is larger than b!
```

```
# Example of using if-elif-else statement
a = int(input("Enter an integer: "))
b = int(input("Enter another integer: "))

if a < b:
    print('a is smaller than b!')
elif a > b:
    print('a is larger than b!')
else:
    print('a is equal to b!')
```

```
Enter an integer: 34
Enter another integer: 34
a is equal to b!
```

When there is more than one condition to be evaluated, logical operators can be used.

```
num = int(input("Enter an integer between 0 and 100 (inclusive): "))

if 0 <= num and num <= 100:
    print("Well done!")
else:
    print("Number is not between 0 and 100.")
```

```
Enter an integer between 0 and 100 (inclusive): 8
Well done!
```

### 2.1.1 Short hand if

If you have only one statement to execute, you can put it on the same line as the `if` statement.

```
a = 10
b = 3

if a > b: print("a is greater than b")
```

```
a is greater than b
```

### 2.1.2 Short hand if-else

If you have only one statement to execute, one for if, and one for else, you can put it all on the same line.

```
score = int(input("Enter your score: "))

print('pass') if score >= 50 else print('fail')
```

```
Enter your score: 60
pass
```

### 2.1.3 Evaluating non-zero numerical values

Non-zero values are evaluated to True
Zero (0 or 0.0) is evaluated to False

```
1  age = int(input("Enter your age: "))
2
3  if age:
4      print("You are " + str(age) + " years old!")
5  else:
6      print("You cannot be 0 years old!")
```

```
Enter your age: 0
You cannot be 0 years old!
```

### 2.1.4 Evaluating non-empty strings

Non-empty strings are evaluated to True.
Empty string is evaluated to False.

```
1  name = input("Enter your name: ")
2
3  if name:
4      print("Hi " + name + "!")
5  else:
6      print("You cannot have no name!")
```

```
Enter your name:
You cannot have no name!
```

## 2.2 WHILE

In general, statements are executed sequentially. There may be a situation when you need to execute a block of code a number of times.

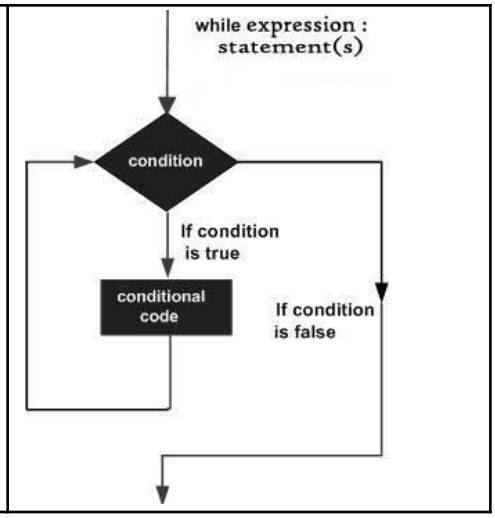
A loop statement allows us to execute a statement or group of statements multiple times.

The while statement allows you to repeatedly execute a block of statements as long as a condition is true. A while statement is an example of a looping statement.

It is important that the condition will eventually become False, else the program will go into an infinite loop.

| Syntax | Flowchart |
|---|---|

```
while condition:
    statement(s)
    update condition
```




Here, statement(s) may be a single statement or a block of statements. The loop iterates and execute the statement(s) while the condition is true. The condition may be any expression.

When the condition becomes false, program control passes to the line immediately following the loop.

All the statements indented by the same number of character spaces after a programming construct are considered to be part of a single block of code.

```
1 count = 1
2 while count <= 5:
3     print(count)
4     count = count + 1
```

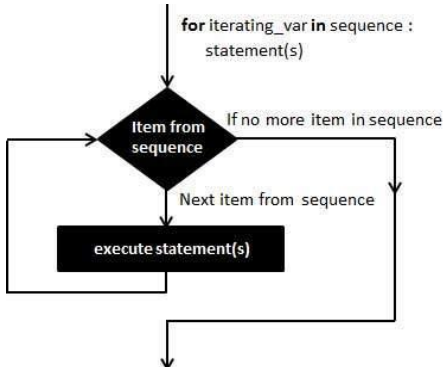
```
1
2
3
4
5
```

The block here, consisting of the print and increment statements, is executed repeatedly until count is no longer less than and equal to 5.

Note that there will be a possibility of a while loop running into an infinite loop. A loop becomes an infinite loop if a condition never becomes FALSE. Use while loop with caution and always check that the condition will be updated to become TRUE for the program control to exit the while loop.

## 2.3 FOR

A `for` loop is used for iterating over the items of any sequence, such as a strong or a list etc.

| Syntax | Flowchart |
| --- | --- |
| `for iterating_var in sequence:`<br>`    statements(s)` |  |



The first item in the sequence is assigned to the iterating variable `iterating_var`. Next, the statements block is executed. Each item in the sequence is assigned to `iterating_var`, and the statement(s) block is executed until the entire sequence is exhausted.

```
1  # for each letter in the string "banana"
2  # print the letter on a new line
3  for letter in "banana":
4      print(letter)
```

```
b
a
n
a
n
a
```

### 2.3.1 The range() function

To loop through a set of code a specified number of times, we can use the `range()` function

The range() function returns a sequence of numbers, starting from 0 by default, and increments by 1 (by default), and ends at a specified number.

```
1 for x in range(5):
2     print(x)
```

```
0
1
2
3
4
```

Note that `range(5)` is not the values of 0 to 5, but the values 0 to 4.

The `range()` function defaults to 0 as a starting value, however it is possible to specify the starting value by adding a parameter: `range(2, 5)`, which means values from 2 to 4 (but not including 5):

```
1 for x in range(2,5):
2     print(x)
```

```
2
3
4
```

The `range()` function defaults to increment the sequence by 1, however, it is possible to specify the increment value by adding a third parameter: range(2, 11, 2):

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

```
2
4
6
8
10
```

### 2.3.2 Iterating by Sequence Index

An alternative way of iterating through each item in a sequence is by index offset into the sequence itself.

```
word = "banana"

for index in range(len(word)):
    print(word[index])
```

```
b
a
n
a
n
a
```

Generally, if we do not know in advance how many times to execute a block of statements, we can choose to use `while` loop. If we know the number of times to execute the code, we can use `for` loop.

## 2.4 Break, continue and pass

Using loops in programming allows automation of tasks efficiently. However, there may be cases when we want to exit the loop prematurely, skip an iteration or we are not sure what we want to do (or should do) in the loop. In this section, we will introduce 3 statements to achieve the actions stated above.

### 2.4.1 Break

In Python, the break statement provides you with the opportunity to exit a loop prematurely before looping through a sequence completed.

```
for x in "spiderman":
    if x == "m":
        break
    print(x)
```

```
s
p
i
d
e
r
```

In the example above, the program control exited the loop when x is 'm', before 'm' is being printed.

### 2.4.2 Continue

With the `continue` statement we can stop the current iteration of the loop, and continue with the next.

```
count = 0

while count < 5:
    count += 1
    if count == 3:
        continue
    print(count)
```

```
1
2
4
5
```

In the example above, all the numbers from 1 to 5 are printed except when the number is 3.

### 2.4.3 Pass

The `pass` statement acts as a "placeholder" in the development of code, when we are not sure how (or what) to do when a condition is met.

```
for n in range(5):
    if n == 2:
        pass
    print(n)
```

```
0
1
2
3
4
```

Will the above code still run if `pass` is omitted?

# 3 Decision table

A decision table is a precise way of modelling logic. A decision table is a tool for documenting complicated logic, which is a part of some business problem.

The aim is to state all possible combinations of conditions and the respective actions that result.

A Decision Table is usually divided into four quadrants as in the table below.

| | |
|---|---|
| Condition statements (Conditions being tested) | Condition alternatives |
| Action statements (Possible actions to take) | Action entries |

Let's look at an example.

A shop gives some customers a discount on goods totaling more than $20. The discounts are:

- 5% for goods totaling more than $100
- 5% with a discount card
- 10% with a discount card and goods totaling more than $100

**Step 1: Analyse the requirements and create the first column**

Identify the conditions that allows customers to get a discount.

| **Conditions** |
|---|
| Goods totaling more than $20 |
| Goods totaling more than $100 |
| Has discount card |
| **Actions** |
| No discount |
| 5% discount |
| 10% discount |

**Step 2: Add columns**

Calculate how many columns are needed in this table. The number of columns depends on the number of conditions and the number of alternatives for each condition. If there are two conditions and each condition can be either true or false, you need four columns. If there are three conditions there will be eight columns, and so on. Mathematically, the number of columns is $2^{conditions}$. In the example above, $2^3 = 8$ columns.

| No of conditions | No of columns |
|---|---|
| 1 | 1 |
| 2 | 4 |
| 3 | 8 |
| 4 | 16 |
| 5 | 32 |

| | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|
| **Conditions** | Goods totaling more than $20 | Y | Y | Y | Y | N | N | N | N |
| | Goods totaling more than $100 | Y | Y | N | N | Y | Y | N | N |
| | Has discount card | Y | N | Y | N | Y | N | Y | N |
| **Actions** | No discount | | | | | | | | |
| | 5% discount | | | | | | | | |
| | 10% discount | | | | | | | | |

**Step 3: Determine the actions**

Enter actions for each column in the table. You will be able to find this information in the requirement.

| | | **Rules** | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|
| **Conditions** | Goods totaling more than $20 | Y | Y | Y | Y | N | N | N | N |
| | Goods totaling more than $100 | Y | Y | N | N | Y | Y | N | N |
| | Has discount card | Y | N | Y | N | Y | N | Y | N |
| **Actions** | No discount | | | | X | X | X | X | X |
| | 5% discount | | X | X | | | | | |
| | 10% discount | X | | | | | | | |

**Step 4: Reduce the table**

Simplify the solution by removing redundancies in the decision table.
Mark the insignificant values with values “-“.

| | | **Rules** |
|---|---|---|

| | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|
| **Conditions** | Goods totaling more than $20 | Y | Y | Y | Y | N | N | N | N |
| | Goods totaling more than $100 | Y | Y | N | N | Y | Y | N | N |
| | Has discount card | Y | N | Y | N | Y | N | Y | N |
| **Actions** | No discount | | | | X | X | X | X | X |
| | 5% discount | | X | X | | | | | |
| | 10% discount | X | | | | | | | |

| | | **Rules** | | | | |
|---|---|---|---|---|---|---|
| **Conditions** | Goods totaling more than $20 | Y | Y | Y | Y | N |
| | Goods totaling more than $100 | Y | Y | N | N | - |
| | Has discount card | Y | N | Y | N | - |
| **Actions** | No discount | | | | X | X |
| | 5% discount | | X | X | | |
| | 10% discount | X | | | | |

Note that as long as the goods do not total up to more than $20, there will be no discount despite having a discount card. Hence, the only determining condition for having no discount is when the goods do not total up to $20. We can remove the redundancies by using dashes for the other conditions.