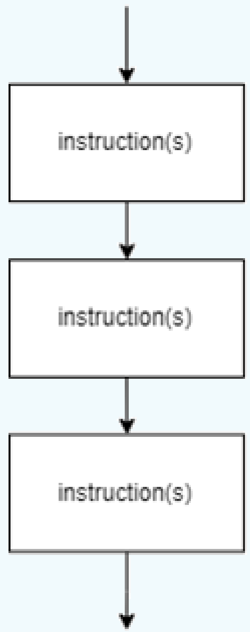
Topic 05

Control Structures

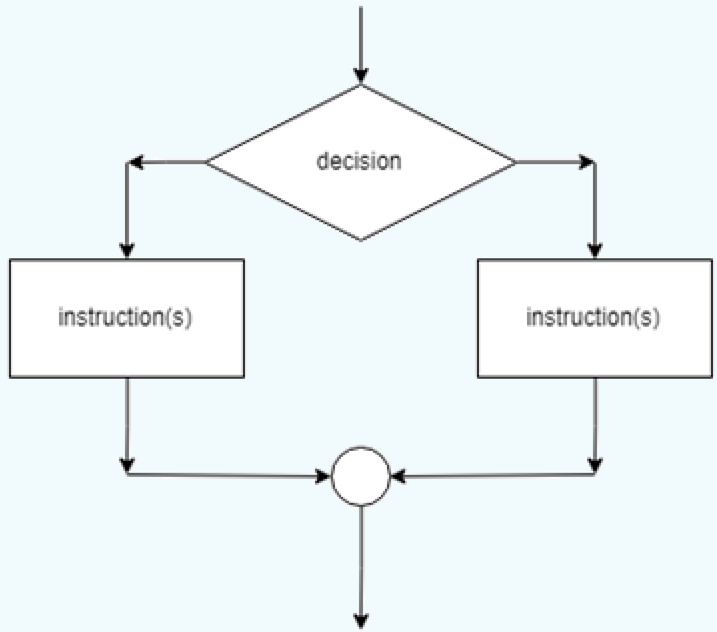
Prepared by: Suhaini Nordin

Acknowledgement: Some contents in this notes are edited from original notes prepared by Ban Kar Weng (William)

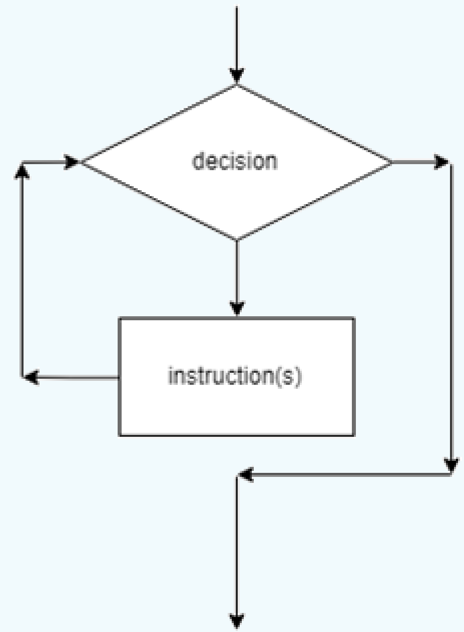
Control Structures



Sequence



Selection



Repetition

Selection Control Structure

Objectives

1. To be able to develop solution using selection logic structure and sequential logic structure.
2. To know and understand the differences between straight-through, positive, and negative selection logic structure.
3. To be able to solve problems using nested selection logic structure.

Decision Logic Structure

- Use ***If/Then/Else*** instruction

If <condition> Then

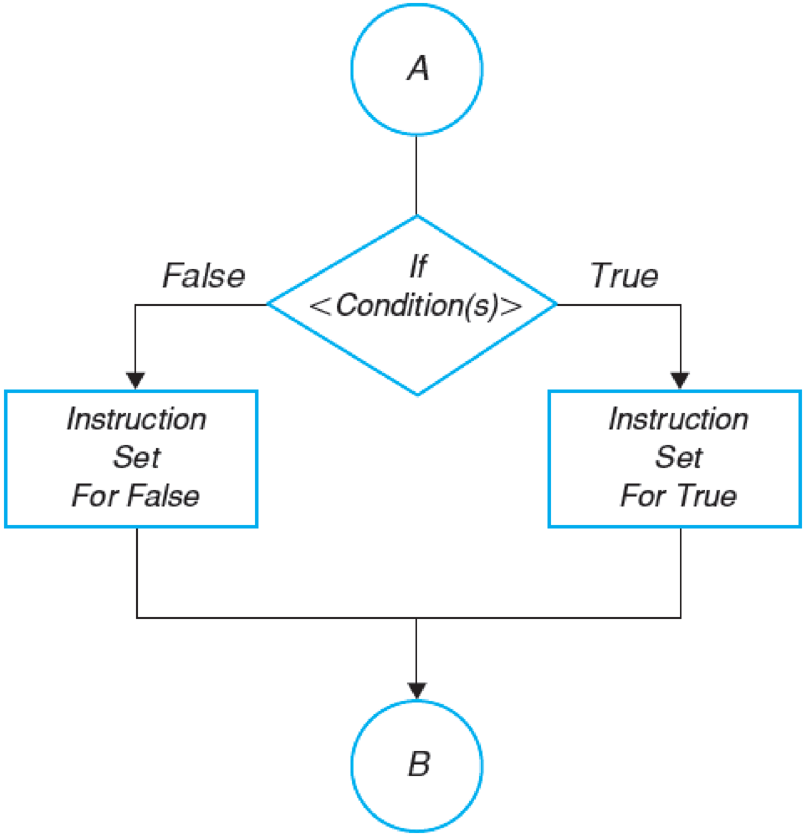
True <True instructions>

Else

False <False instructions>

EndIf

Flowchart Diagram for Decision Structure



** It is best to be consistent in choosing the point of the diamond for these branches to be placed*

Single Condition

- A simple decision with only one condition and one action/ set of actions
- Example:
  - If **Mark < 50** then Fail the course
    - → That one condition is about the mark
  - If **Income > 10000** then Can purchase house
    - → That one condition is about the income
  - If **Height < 120** then Eligible for free balloon and cannot ride the Space Mountain
    - → That one condition is about the height

Example

- Student's status will be "Pass" if his/her mark is 50 and above.

Input	Process	Output

mark	If mark >= 50
	    status = "Pass"
	Else
	    status = "Fail"
	End If
		status

1. Start
2. Get mark
3. If mark >= 50

       status = "Pass"

   Else

       status = "Fail"

   End If
4. Print status
5. End

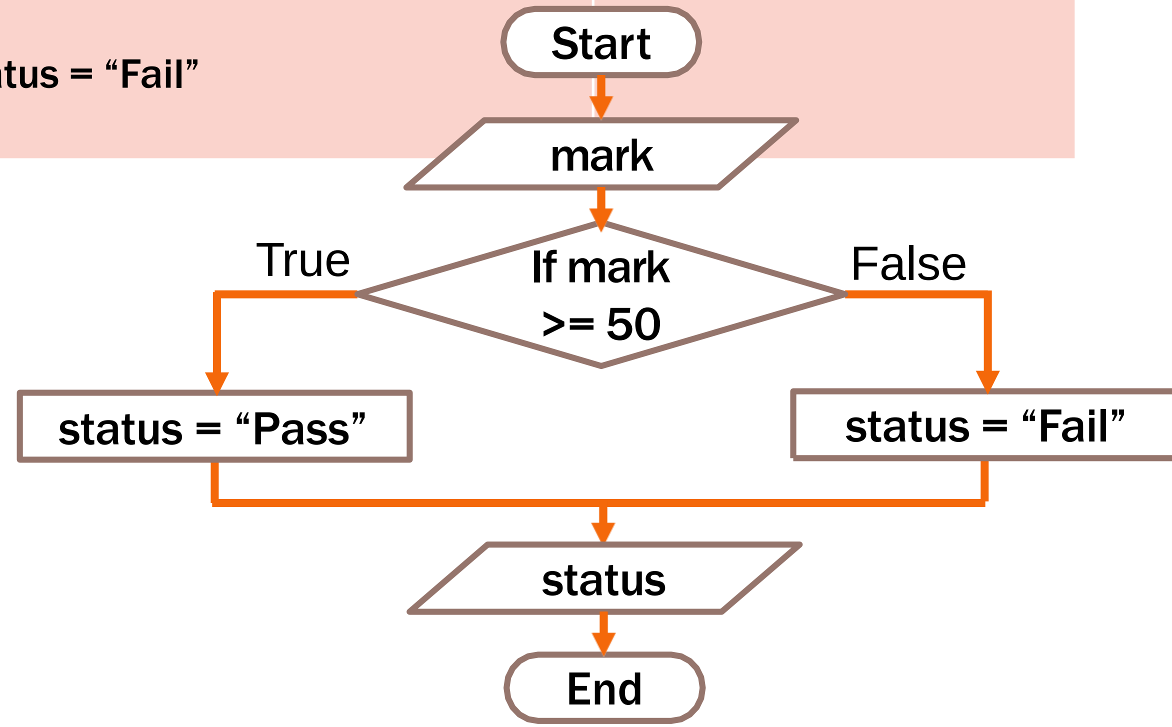
Example

- Student's status will be "Pass" if his/her mark is 50 and above.

Input	Process	Output

mark	If mark >= 50
	        status = "Pass"
	Else
	        status = "Fail"
	End If
		status

1. Start
2. Get mark
3. If mark >= 50
   status = "Pass"
   Else
   status = "Fail"
   End If
4. Print status
5. End



Multiple Condition

- Decision with multiple condition that lead to one action / set of actions
- Example:
  - If **Hour > 40 and Status = = "Permanent" then <u>Eligible for Bonus</u>**
    → One condition is about the Hour and another condition is about the Status
  - If **CGPA < 2.00 and CreditHour < 20 then <u>Terminated</u>**
    → One condition is about the CGPA and another condition is about the CreditHour
  - If **Point > 100 or IQ > 170 then <u>Join the League</u>**
    → One condition is about Point and another condition is about IQ

Example

- An employee is eligible for bonus when the hours worked is more than 40 and is a permanent staff.

Input	Process	Output

Hour
Status
	If Hour > 40 and Status == “Permanent”
	        Print “Eligible for bonus”
	Else
	        Print “Not eligible for bonus”
	End If	

1. Start
2. Get Hour
3. Get Status
4. If Hour > 40 and Status == “Permanent”
       Print “Eligible for bonus”
   Else
       Print “Not eligible for bonus”
   End If
5. End

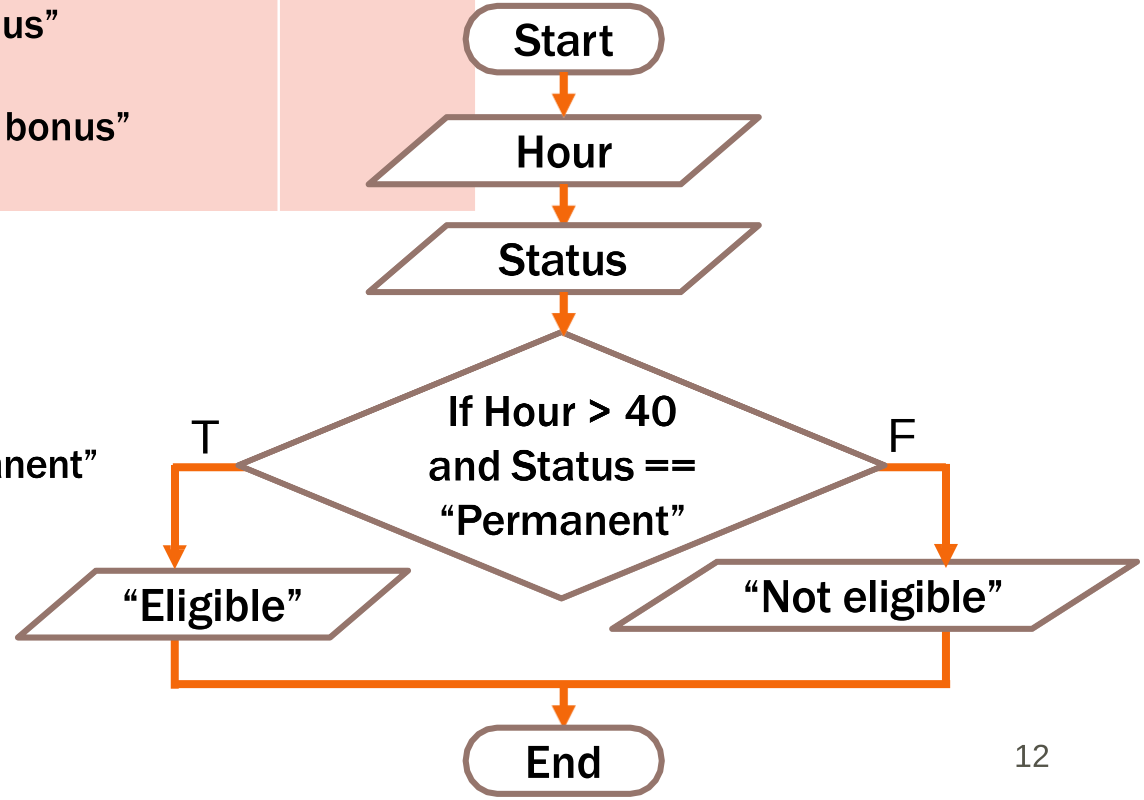
Example

- An employee is eligible for bonus when the hours worked is more than 40 and is a permanent staff.

Input	Process	Output

Hour
Status
	If Hour > 40 and Status == “Permanent”
	Print “Eligible for bonus”
	Else
	Print “Not eligible for bonus”
	End If	

1. Start
2. Get Hour
3. Get Status
4. If Hour > 40 and Status == “Permanent”
   Print “Eligible for bonus”
   Else
   Print “Not eligible for bonus”
   End If
5. End

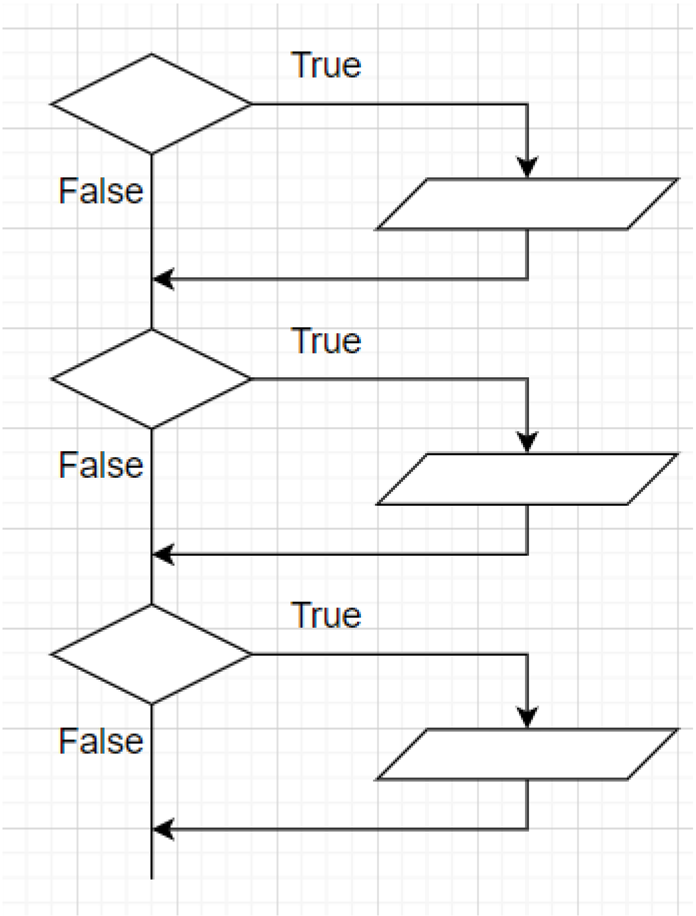


Multiple *If/Then/Else* Instruction

Type of Decision Logic		
Straight-through Logic	Positive Logic	Negative Logic
All decisions are processed sequentially one after another	Allows the flow of the processing to continue through the module when the resultant is true	Allows the flow of the processing to continue through the module when the resultant is false

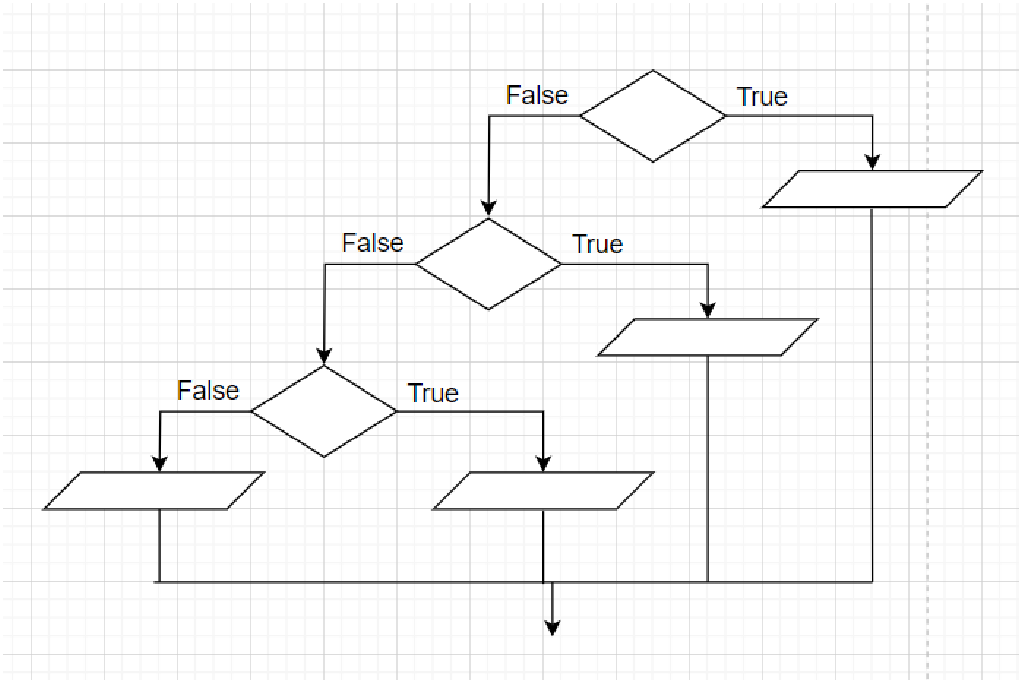
Straight-through Logic

- ALL decision/conditions must be processed
- Used in:
  - Data validation
  - Languages that have limited features



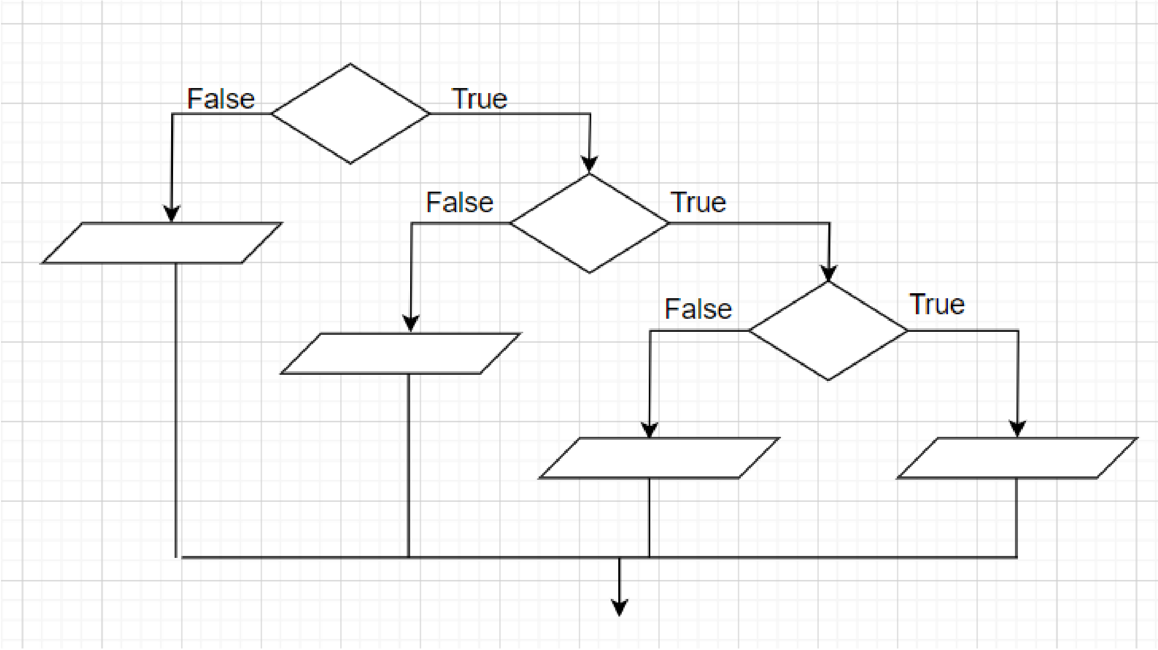
Positive Logic

- It tells the computer to perform a set of instructions and continue processing if the condition is *true*
- If *false* – the computer will process another decisions
- Fewer decision to be processed



Negative Logic

- It tells the computer to perform a set of instructions and continue processing if the condition is ***false***
- If ***true*** – the computer will process another decisions
- Fewer decision to be processed



Example

- The charge to enter the Zoo is listed in the table below:

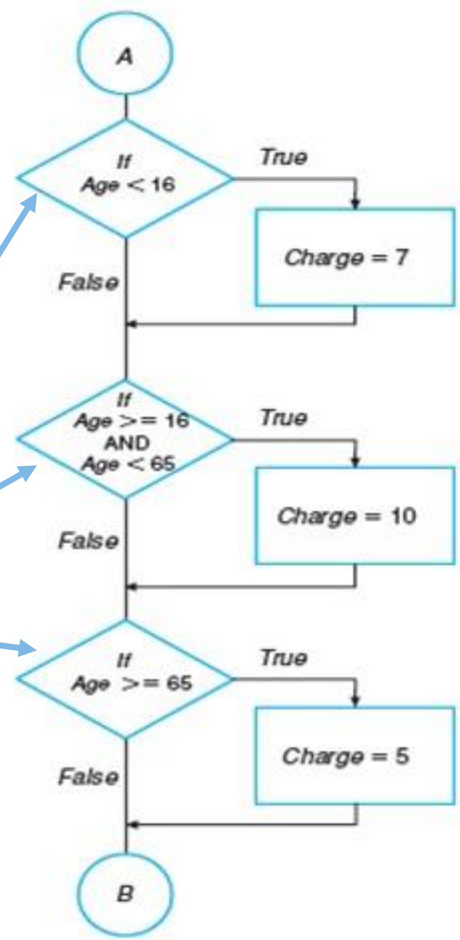
Age Range	Charge
Age < 16	$7
16 <= Age < 65	$10
Age >= 65	$5

Straight-through Logic

Algorithm

1. …
2. …
3. *If Age < 16 Then*
   Charge = 7
   Endif
4. *If Age >= 16 AND Age < 65 Then*
   Charge = 10
   Endif
5. *If Age >= 65 Then*
   Charge = 5
   Endif
6. *End*

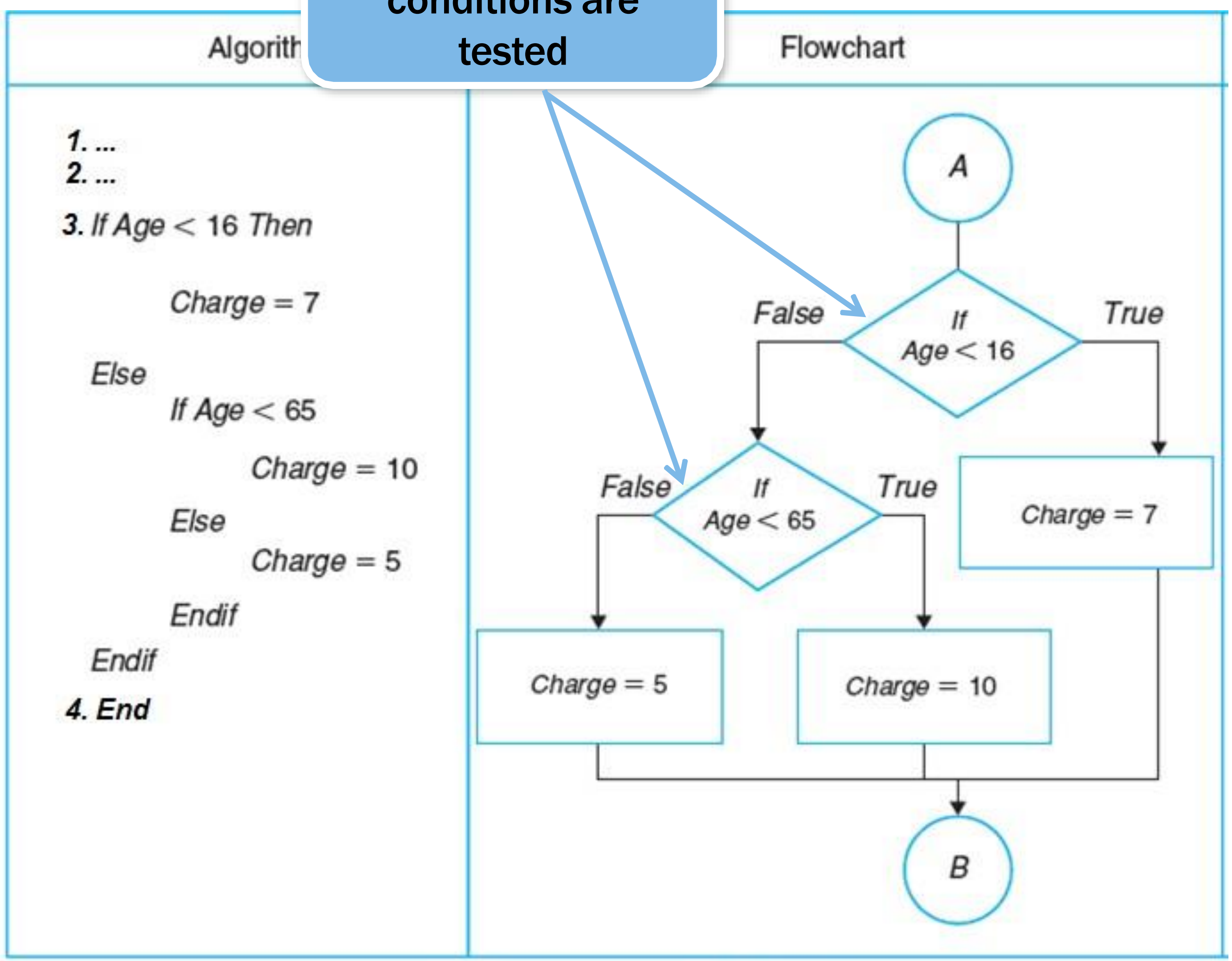
Flowchart



All 3 conditions are tested

Positive Logic

Only TWO conditions are tested



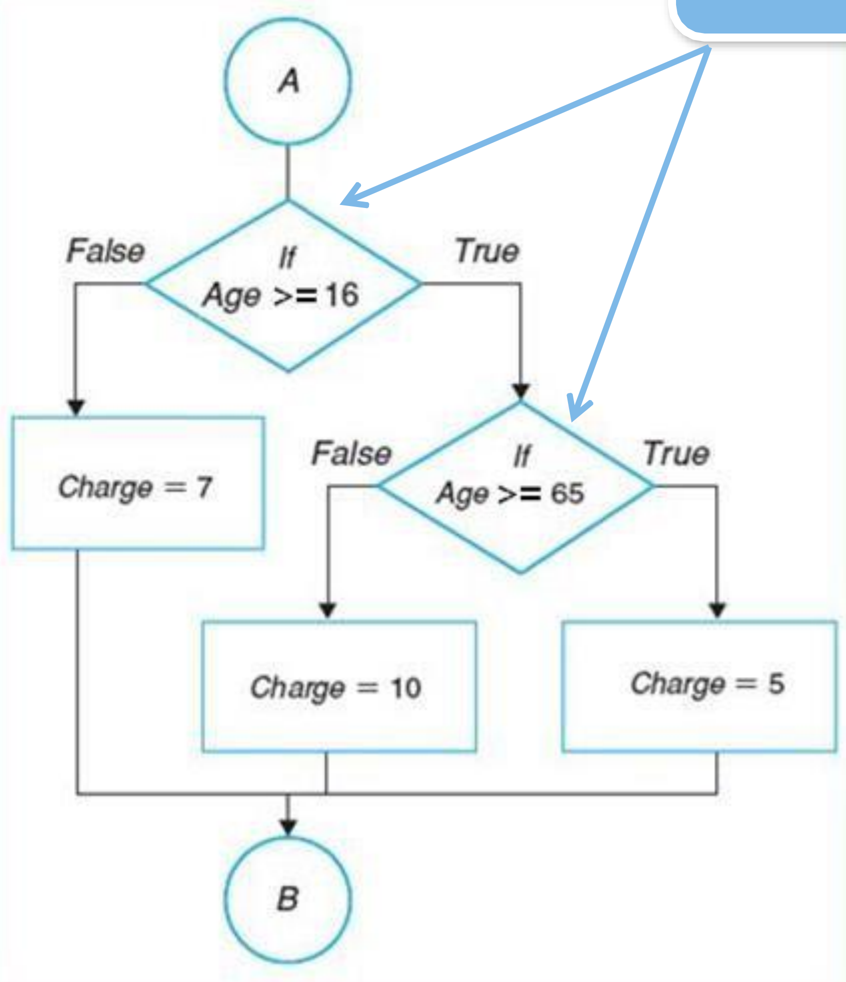
Negative Logic

Only TWO conditions are tested

Algorithm

```
1. ...
2. ...
3. If Age >= 16 Then
        If Age >= 65 Then
                Charge = 5
        Else
                Charge = 10
        Endif
   Else
        Charge = 7
   Endif
4. End
```

Flowchart



Exercise (Level 1)

1. Draw the flowchart for a program that will ask the user to enter the year he/she was born, and then it will display whether the year is a leap year or not.

Exercise (Level 2)

Draw a straight-through logic, positive logic and negative logic flowcharts and write the algorithm for each logic structures for the following problems.

1. Harrods is doing its annual sale. All items are on discount. Any item with Blue price tag will be given 25% discount, Red tag is 50% discount, and Green tag is 70% discount. Write the solutions to find the sale price of the item.

2. A program to determine the athlete's category for the standing long jump when the given input is the distance in centimeter with reference to this information.

Category	Distance in meter
Excellent	> 1.91
Average	1.91 – 1.62
Below average	< 1.62

Exercise (Level 3)

Draw a positive logic and negative logic flowcharts and write the algorithm for each logic structures for this problem.

Mr. Jones gave a test to his class. His test has 20 questions. He needs a program that will display the student's grade based on the best score. Assuming that the best score is 18, write the solution to find the grade for a random student.

Formula for Grade is as follows:

A will range from the best score, to the best score minus 2.

B will range from the best score minus 3, to the best score minus 5.

C will range from the best score minus 6, to the best score minus 8.

D will range from the best score minus 9, to the best score minus 11.

F will be anything below the best score minus 11.

If Statement

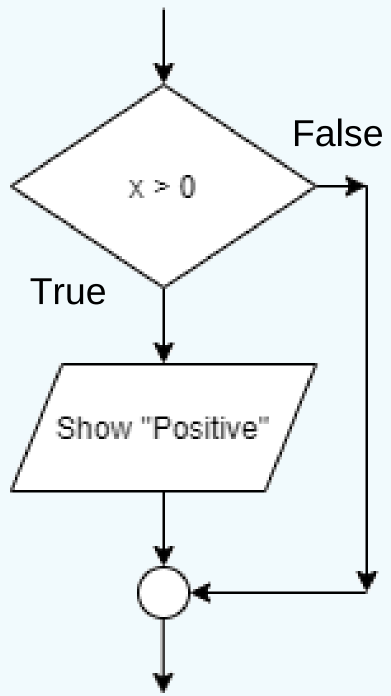
If Statement

```
if condition:
    indented block of statements
```

- The simplest selection structure to check a condition and change the behavior of the program accordingly.
- If `condition` is true, the `indented block of statements` will execute.
- A colon character (:) must follow the `condition`.

If Statement

Flowchart:

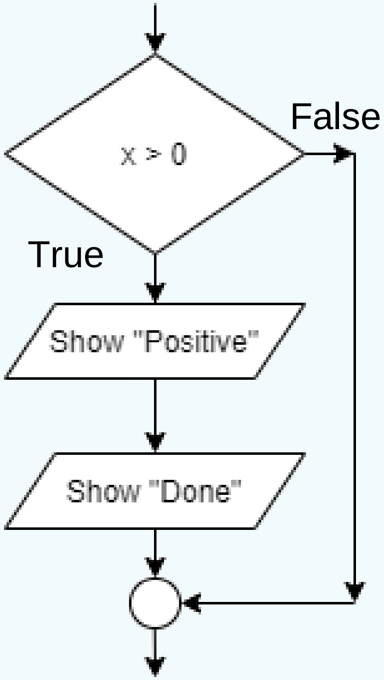


Code:

```
if x > 0:
   print('Positive')
```

If Statement

Flowchart:

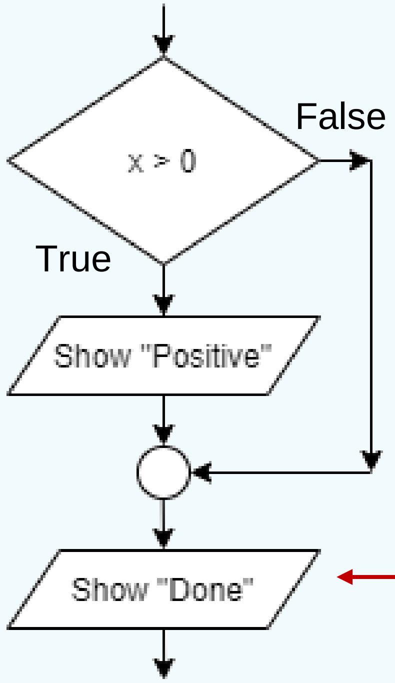


Code:

```
if x > 0:
    print('Positive')
    print('Done')
```

If Statement

Flowchart:



Code:

```
if x > 0:
    print('Positive')
print('Done')
```

NOTE: Indentation is important in Python

If-Else Statement

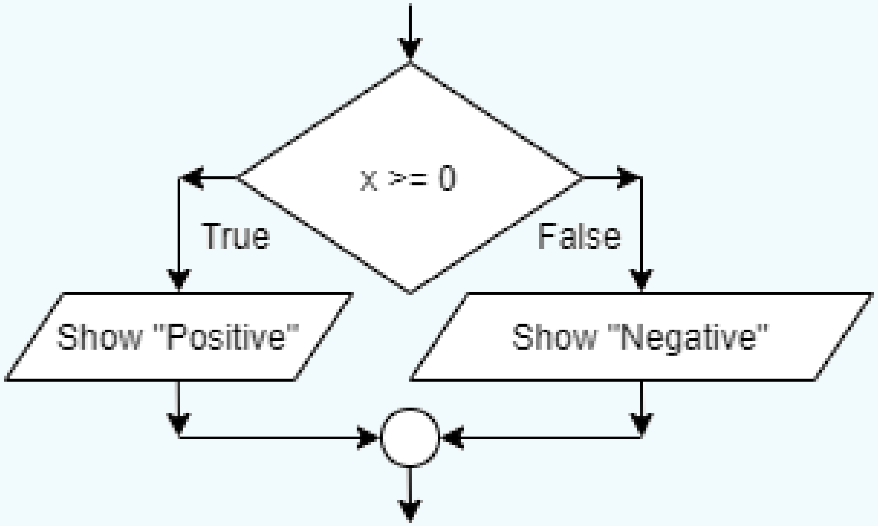
If-Else Statement

```
if condition:
    indented block of statements #1
else:
    indented block of statements #2
```

- If `condition` is true, the `indented block of statements #1` will execute.
- If `condition` is false, the `indented block of statements #2` will execute instead.
- A colon character (:) must follow the `else`.

If-Else Statement

Flowchart:

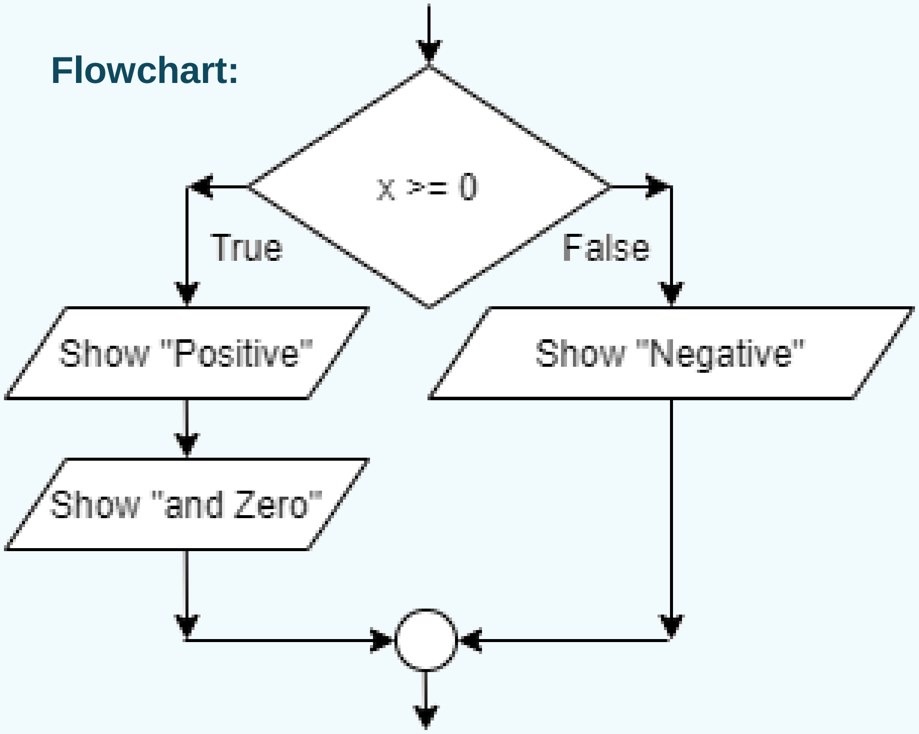


Code:

```
if x >= 0:
    print('Positive')
else:
    print('Negative')
```

If-Else Statement

Flowchart:

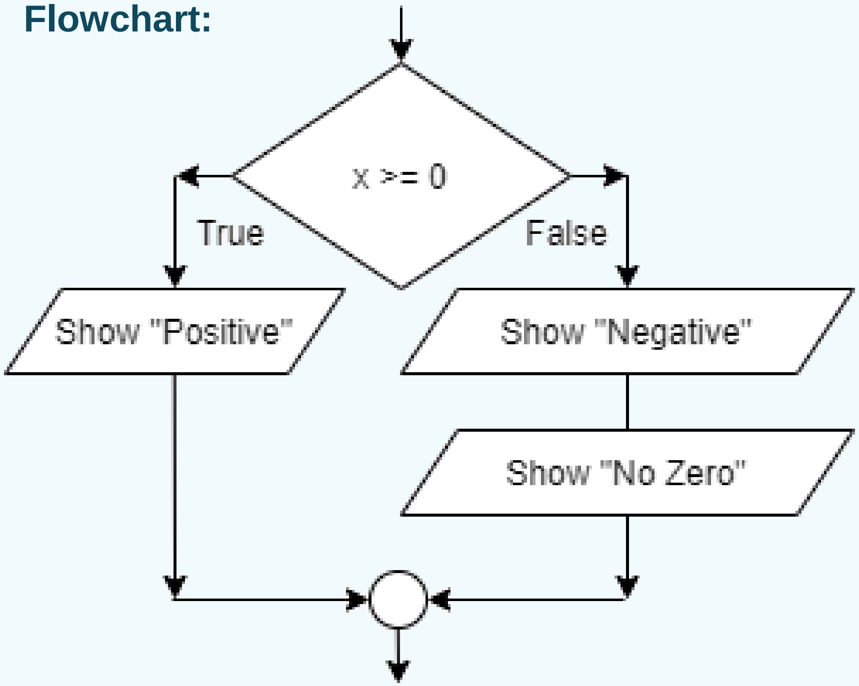


Code:

```
if x >= 0:
    print('Positive')
    print('and Zero')
else:
    print('Negative')
```

If-Else Statement

Flowchart:



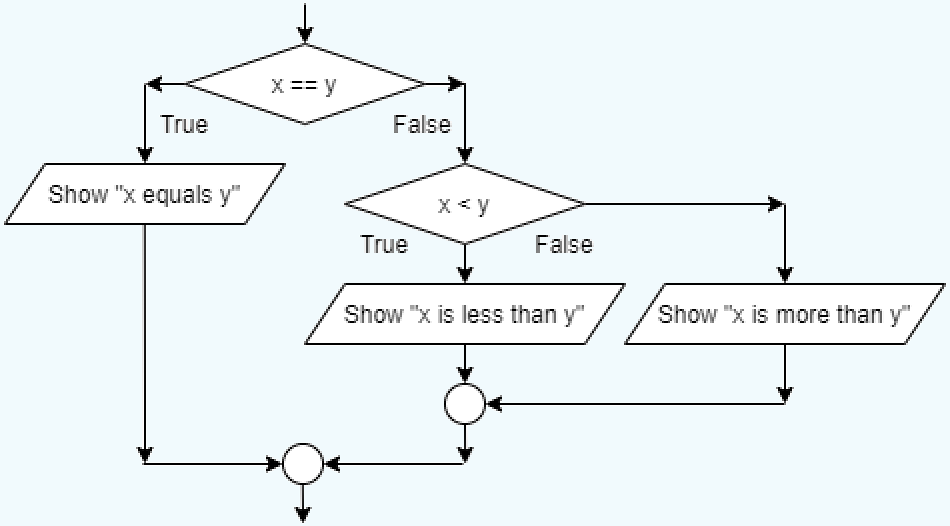
Code:

```
if x >= 0:
    print('Positive')
else:
    print('Negative')
    print('No Zero')
```

Nested *If* Statement

Nested *If* Statement

- It is possible to nest an if statement within another if statement.




```
if x == y:
   print('x equals y')
else:
   if x < y:
      print('x is less than y')
   else:
      print('x is more than y')
```

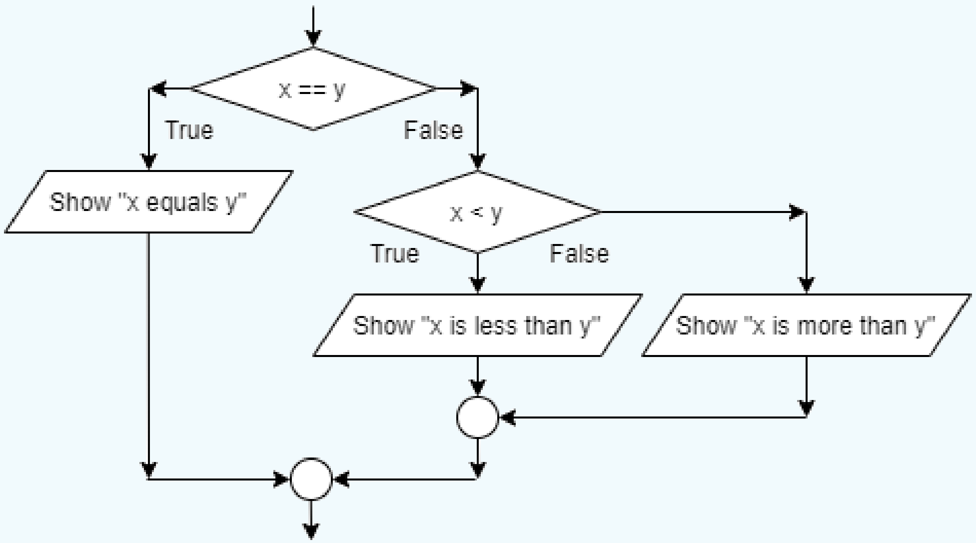
The *Elif* Clause

The *Elif* Clause

```
if condition #1:
   indented block of statements #1
elif condition #2:
   indented block of statements #2
else:
   indented block of statements #3
```

- The `elif` (short for “else if”) clause adds one additional condition to if-else statement.
- No limit imposed on the number of the `elif` clause.
- If there is an `else` clause, it has to be at the end, but it is optional.

The *Elif* Clause




```python
if x == y:
    print('x equals y')
elif x < y:
    print('x is less than y')
else:
    print('x is more than y')
```

Demo

marks-to-grade.py

Write a program that accepts a student's mark and outputs the grade according to this table.

Marks m	Grade
$80 \le m \le 100$	A
$60 \le m < 80$	B
$50 \le m < 60$	C
$0 \le m < 50$	F

Sample Runs:

```
C:\> python marks-to-grade.py
Enter marks: -0.1
Invalid marks
```

```
C:\> python marks-to-grade.py
Enter marks: 80
A
```

```
C:\> python marks-to-grade.py
Enter marks: 50
C
```

```
C:\> python marks-to-grade.py
Enter marks: 100.1
Invalid marks
```

```
C:\> python marks-to-grade.py
Enter marks: 79.9
B
```

```
C:\> python marks-to-grade.py
Enter marks: 49.9
F
```

Short-Circuit Evaluation

Short-Circuit Evaluation

- Occurs when the evaluation of a logical expression stops because the overall value is already known.
- Example:

```
y = 0
z = (1 <= y and y <= 10)
```

Since `y = 0`, `1 <= y` would yield *`False`*

Since `1 <= y` is *`False`*, `z` would be *`False`* regardless of the outcome of `y = 10`, therefore `y <= 10` was **never evaluated**.

Short-Circuit Evaluation

- A clearer (but weird) way to see Short-Circuit Evaluation in action.

short-circuit-1.py

```
y = int(input('Enter y: '))
z = (1 <= y and input('Hi: '))
```

Sample Run 1

```
C:\> python short-circuit-1.py
Enter y: 0
```

Sample Run 2

```
C:\> python short-circuit-1.py
Enter y: 1
Hi: a
```

Short-Circuit Evaluation

- Another similar example, but with the `or` operator.

short-circuit-2.py

```
y = int(input('Enter y: '))
z = (1 <= y or input('Hi: '))
```

Sample Run 1

```
C:\> python short-circuit-2.py
Enter y: 0
Hi: a
```

Sample Run 2

```
C:\> python short-circuit-2.py
Enter y: 1
```

Repetition Control Structure

Objective

1. Use problem-solving tools when developing solution using loop logic structure.
2. Use counters and accumulators in problem solution.
3. Use nested loop instructions to develop problem solution.
4. Distinguish different uses of three types of loop logic structures.

Loop Logic Structure

- To repeat instructions in a solution
- To return to the earlier point in the solution

Types of loop structure

Repeat/Until Loop	While/WhileEnd Loop	Automatic-Counter Loop

Also known as Do/While Loop

Loop Logic Structure

Standard types of task used in loop structure

Counting	Accumulating
Also called incrementing and decrementing	Also called calculating a sum or a total
The value is a constant	The value is a variable

The value of the variable is assigned to zero before starting the loop – *initializing the variable*

Counting: Incrementing/Decrementing

- A process of adding/subtracting a constant
- The variable must be initialized (set the value) before starting the loop

Counter = 0 ← **Initialize**
Counter = Counter + 1 ← **Incrementing**

x = 5 ← **Initialize**
x = x – 1 ←**Decrementing**

Accumulating

- A process of adding a variable to the value of another variable which hold the total or sum

Total = 0 ← Initialize the accumulator
Total = Total + Variable ← Increment the accumulator

TotalSales = 0 ← Initialize the accumulator
TotalSales = TotalSales + Sales ← Increment the accumulator

Accumulating

- A process of calculating the product of a series of number
- Two exceptions:
  - "+" sign is replaced with "*" sign
  - Product variable must be initialized. It can be any number but NOT 0.

Product = 1 ← **Initialize the accumulator**

Product = Product * Number ← **Equation to accumulate**

While/WhileEnd Loop

- It tells computer to repeat the sets of instructions while the condition is *True*

While <condition(s)>

Instruction

Instruction

...

...

WhileEnd

While/WhileEnd Loop

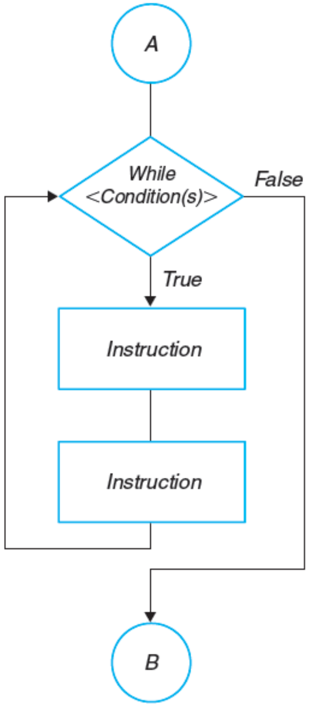
- It tells computer to repeat the sets of instructions while the condition is *True*

```
LapNumber = 0  ←Initialization of the counter
TotalApples = 0 ←Initialization of the accumulator

While LapNumber is less than 3  ← Condition that will tell you to continue or stop running
            Print LapNumber
            Run 1 Lap
            Pick up some Apples
            TotalApples = TotalApples + Apples  ←Accumulation of accumulator
            Print TotalApples
            LapNumber = LapNumber + 1  ← Incrementation of the counter
WhileEnd

End
```

While/WhileEnd Loop



Example

Which of these pseudocodes are correct for a program that will display Hello 3 times?

```
Start
Set times = 1
While (times <= 3)
        Print “Hello”
        times = times + 1
    WhileEnd
End
```

```
Start
Set times = 0
While (times <= 3)
        Print Hello
        times = times - 1
    WhileEnd
End
```

```
Start
Set times = 2
While (times > 0)
        Print “Hello”
        times = times -1
    WhileEnd
End
```

```
Start
Set times = 15
Print “Hello”
While (times > 13)
        Print “Hello”
        times = times -1
    WhileEnd
End
```

```
Start
Set times = 4
While (times > 1)
        Print “Hello”
        times = times -1
    WhileEnd
End
```

```
Start
Set times = 2
While (times >= 0)
        Print “Hello”
        times = times -1
    WhileEnd
End
```

Example *

Write a pseudocode for a program that ask a user to enter how many hello he/she wants to see on the screen.

```
Ask the user how many hello
Set the counter to 0
While counter < number of hello user want
        Print “Hello”
        Increment counter
While End
```

```
Start
Get howmany
Counter = 0
While Counter < howmany
        Print “Hello”
        Counter = Counter + 1
While End
End
```


```
Ask the user how many hello
While number of hello user want  > 0
        Print “Hello”
        Decrement number of hello user want
While End
```

```
Start
Get howmany
While howmany > 0
        Print “Hello”
        howmany = howmany - 1
While End
End
```

Example *

- Write a pseudocode to ask a user to enter 6 random numbers and the program will print the total of that 6 numbers.

Example

- Write a pseudocode to ask a user to enter the weekly price of petrol per liter and the program will print the average price of petrol for that month. Assume that the petrol price is updated every week and there are 4 weeks in a month.

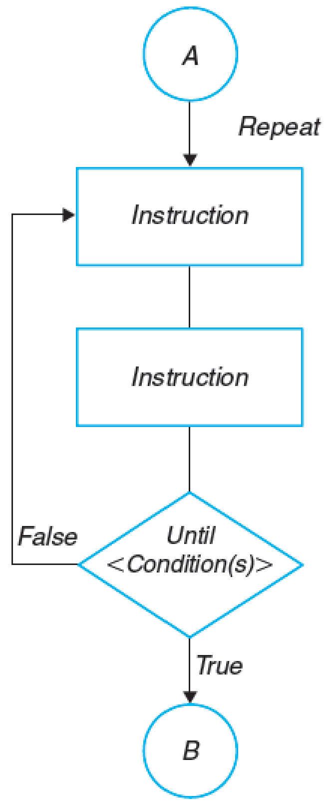
Repeat/Until Loop

- It tells computer to repeat the sets of instructions until the condition is True

While/WhileEnd	Repeat/Until
It will continue to loop as long as the resultant of the condition is True	It will stop the loop process when the resultant of the condition is True
The condition is processed at the beginning of the program	The condition is processed at the end of the program
Must initialize the data – resultant of the condition is True	The instruction in the loop are processed at least once

Repeat/Until Loop

Repeat
  Instruction
  Instruction
  .
  .
Until<Condition(s)>



Repeat/Until Loop

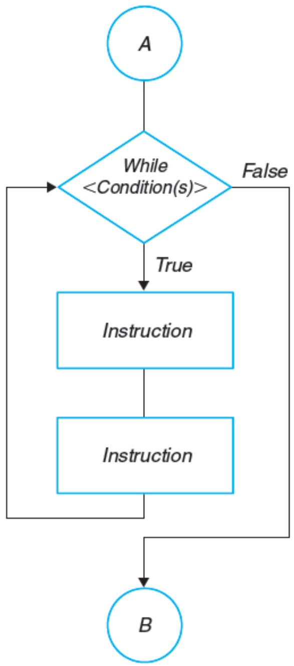
- It tells computer to repeat the sets of instructions until the condition is True

LapNumber = 0 ←Initialization of the counter
TotalApples = 0 ←Initialization of the accumulator

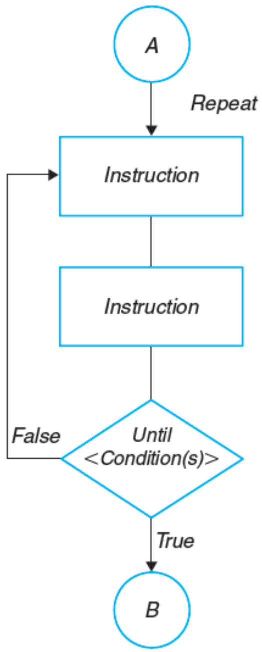
Repeat:
        Run 1 Lap
        Pick up some Apples
        TotalApples = TotalApples + Apples ←Accumulation of accumulator
        LapNumber = LapNumber + 1 ← Incrementation of the counter
Until LapNumber is more or equal than 5 ← Condition that will tell you to continue or stop running

Print TotalApples
End

Comparison



While/WhileEnd



Repeat/Until Loop

Example *

- Which of these pseudocodes are correct for a program that will display Hello 3 times?

```
Start
Set times = 3
Repeat
        Print "Hello"
        times = times -1
    Until (times >= 1)
End
```

```
Start
Set times = 3
Repeat
        Print "Hello"
        times = times -1
    Until (times < 1)
End
```

```
Start
Set times = 3
Repeat
        Print "Hello"
        times = times -1
    Until (times <=  0)
End
```

```
Start
Set times = 73
Print "Hello"
Repeat
        Print "Hello"
        times = times + 1
    Until (times >= 75)
End
```

```
Start
Set times = 73
Repeat
        Print "Hello"
        times = times + 1
    Until (times >= 76)
End
```

Example *

- Write a pseudocode to ask a user to enter how many hello he/she wants to see on the screen.

```
Ask the user how many hello
Set the counter to 0
Repeat:
            Print “Hello”
            Increment counter
Until counter >= number of hello user want
```

```
Start
Get howmany
Counter = 0
Repeat:
            Print “Hello”
            Counter = Counter + 1
Until Counter >= howmany
End
```


```
Ask the user how many hello
Repeat:
            Print “Hello”
            Decrement number of hello user want
Until number of hello user want  <= 0
```

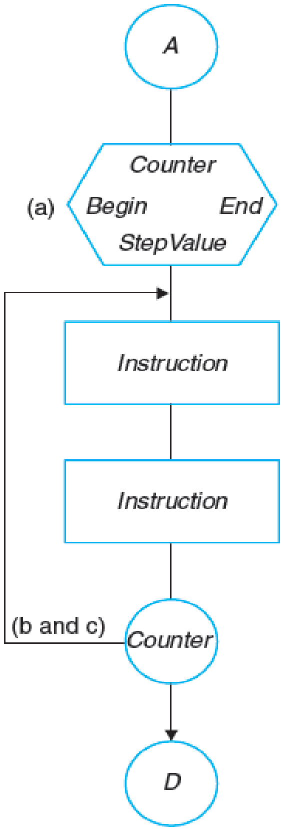
```
Start
Get howmany
Repeat:
            Print “Hello”
            howmany = howmany - 1
Until howmany <= 0
End
```

Automatic-Counter Loop

- It tells computer to increase or decrease the value of a variable every time the loop is repeated
- Variables is used as a counter which starts counting at a specified number and increase every time the loop is processed until it is greater than the ending number
- Cannot be changed during the processing of instruction in the loops

Automatic-Counter Loop

Loop: Counter = Begin To End Step StepValue
Instruction
Instruction
.
.
.
Loop-End: Counter



Begin- - beginning value
End – ending value
StepValue – increment value

Automatic-Counter Loop

- It tells computer to repeat the sets of instructions in a loop that has been predetermined.

```
TotalApples = 0 ←Initialization of the accumulator
Loop: LapNumber  = 0 to 4 Step 1  ← Predetermined loop
        Run 1 Lap
        Pick up some Apples
        TotalApples = TotalApples + Apples  ←Accumulation of accumulator
Loop-End: LapNumber
```

Example *

Which of these pseudocodes are correct for a program that will display Hello 3 times?

```
Start
Loop: times = 3 To 1 Step -1
        Print “Hello”
Loop-End: times
End
```

```
Start
Loop: times = 1 To 3 Step 1
        Print “Hello”
Loop-End: times
End
```

```
Start
Loop: times = 0 To 4 Step 2
        Print “Hello”
Loop-End: times
End
```

```
Start
Loop: times = 0 To 2 Step 1
        Print “Hello”
Loop-End: times
End
```

```
Start
Loop: times = 99 To 101 Step 1
        Print “Hello”
Loop-End: times
End
```

Example *

Which of these pseudocodes are correct for a program that ask a user to enter how many hello he/she wants to see on the screen?

```
Start
Get howmany
Loop: PrintingFrom  = 0 to howmany Step 1
            Print "Hello"
Loop-End: PrintingFrom
End
```

```
Start
Get howmany
Loop: PrintingFrom  = 0 to howmany-1  Step
1
            Print "Hello"
Loop-End: PrintingFrom
End
```

```
Start
Get howmany
Loop: PrintingFrom  = howmany to 0 Step -1
            Print "Hello"
Loop-End: PrintingFrom
End
```

```
Start
Get howmany
Loop: PrintingFrom  = howmany+1 to 2 Step -1
            Print "Hello"
Loop-End: PrintingFrom
End
```

The `while` Loop

The *`while`* Loop

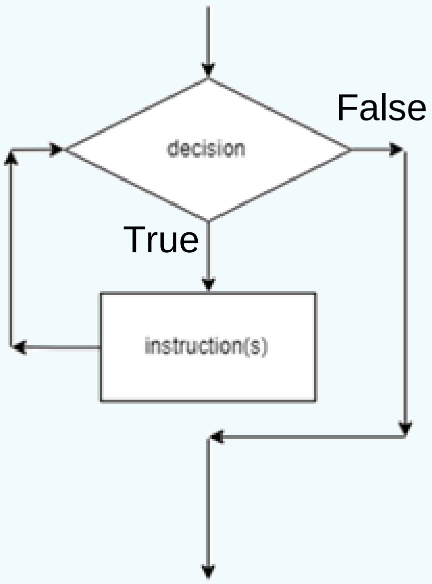
```
while condition:
    indented block of statements
```

Flow of execution:

1. Evaluate the condition, yielding *`True`* or *`False`*.
2. If *`condition`* is true, the *`indented block of statements`* will execute.
3. If *`condition`* is false, exit the *`while`* statement and execute the next statement after the *`while`* statement.

The *while* Loop

Flowchart:

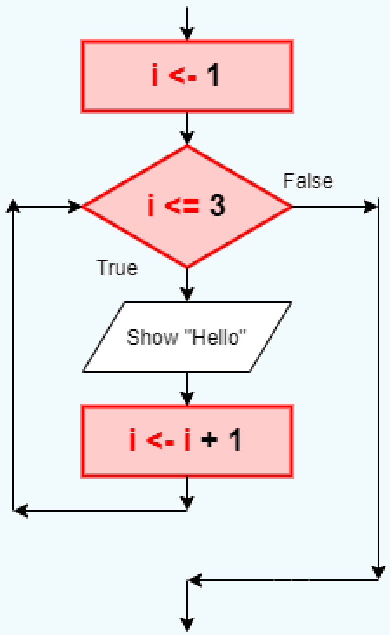


Code:

```
while decision:
    instruction(s)
```

The *while* Loop

Flowchart:

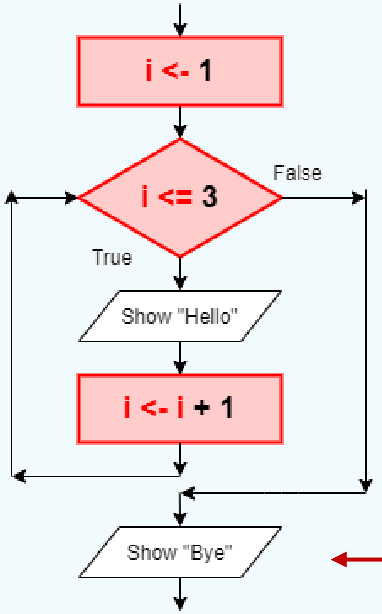


Code:

```
i = 1
while i <= 3:
    print('Hello')
    i = i + 1
```

The *while* Loop

Flowchart:



Code:

```
i = 1
while i <= 3:
    print('Hello')
    i = i + 1
print('Bye')
```

NOTE: Indentation is important in Python

Demo

`zero-to-n.py`

Using *`while`* loop, write a program that accepts a positive integer N, and display the sequence from 0 to N.

Sample Run 1

```
C:\> python zero-to-n.py
Enter n: 5
0 1 2 3 4 5
```

Sample Run 2

```
C:\> python zero-to-n.py
Enter n: 10
0 1 2 3 4 5 6 7 8 9 10
```

range() function

The *range()* function

```
range(start, stop, step)
```

- Generates an **integer sequence**.
- To show its values, the sequence must be **converted to list** using the *list()* function.

Interactive Mode:

```
>>> range(0, 3, 1)
range(0, 3)
>>> list(range(0, 3, 1))
[0, 1, 2]
```

The *range()* function

```
range(start, stop, step)
```

- Sequence in the form:

  *start, start + step, start + 2 * step, …*

- The last item is never equal to *stop*.

Interactive Mode:

```
>>> list(range(0, 3, 1))
[0, 1, 2]
>>> list(range(5, 10, 2))
[5, 7, 9]
```

The *range()* function

```
range(start, stop, step)
```

- *stop* is mandatory, *start* and *step* are optional.
- **Default value**: *start* = **0**, *step* = **1**

Interactive Mode:

```
>>> list(range(3))
[0, 1, 2]
>>> list(range(2, 5))
[2, 3, 4]
```

Equivalent to range(0, 3, 1)

Equivalent to range(2, 5, 1)

The *range()* function

range(start, stop, step)

- *start*, *stop*, and *step* can be **negative.**

Interactive Mode:

```
>>> list(range(-2, -5, -1))
[-2, -3, -4]
>>> list(range(3, -4, -2))
[3, 1, -1, -3]
>>> list(range(-2, 3))
[-2, -1, 0, 1, 2]
```

Equivalent to range(-2, 3, 1)

for Loop

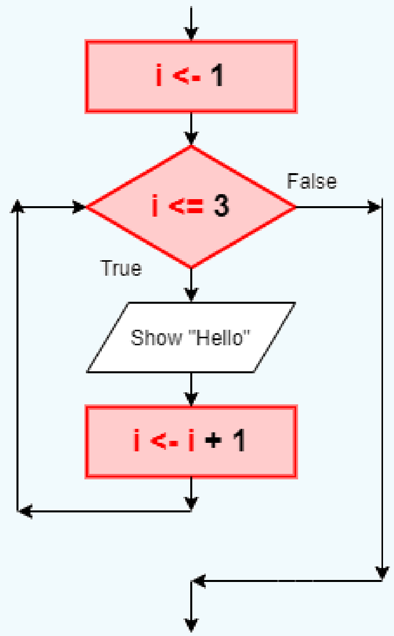
for Loop

```
for var in sequence:
    indented block of statements
```

- Designed to iterate through ***Sequence Type*** such as data returned from the *range()* function.
- Using *for* loop for other sequence type will be covered in next few lectures.

for Loop

Flowchart:



Code (using *while* loop):

```
i = 1
while i <= 3:
    print('Hello')
    i = i + 1
```

Code (using *for* loop):

```
for i in range(1, 4):
    print('Hello')
```

Demo

`zero-to-n.py`

Using *for* loop, write a program that accepts a positive integer N, and display the sequence from 0 to N.

Sample Run 1

```
C:\> python zero-to-n.py
Enter n: 5
0 1 2 3 4 5
```

Sample Run 2

```
C:\> python zero-to-n.py
Enter n: 10
0 1 2 3 4 5 6 7 8 9 10
```

break and *continue*

The *break* statement

- Terminates from anywhere in a loop body.
- Example:

break.py

```
while True:
    x = input('Enter x: ')
    if x == 0:
        break
    print(f'x = {x}')
```

Sample Run

```
C:\> python break.py
Enter x: 1
x = 1
Enter x: -2
x = -2
Enter x: 0
```

The *continue* statement

- Jumps to the top of the loop's header, skipping statements below it within the loop.
- Example (*while* loop):

continue-while.py

```
i = 0
while i < 3:
    if i == 1:
        continue
    print(i)
    i += 1
```

Similar to

Note:
This is NOT Python syntax.
This "fake code" is used to ease explanation

continue-while.py

```
i = 0
while i < 3:
    if i == 1:
        goto $
    print(i)
    i += 1
$
```

The *continue* statement

- Example (*for* loop):

continue-for.py

```
for i in range(3):
    if i == 1:
        continue
    print(i)
```

Similar to

Note:
This is NOT Python syntax.
This "fake code" is used to ease explanation

continue-for.py

```
for i in range(3):
    if i == 1:
        goto $
    print(i)
$
```

Question to Ponder

- Do these two code produces the same output?

continue-while.py

```
i = 0
while i < 3:
    if i == 1:
        continue
    print(i)
    i += 1
```

continue-for.py

```
for i in range(3):
    if i == 1:
        continue
    print(i)
```

Nested Loop

Nested Loop

- It is possible to nest a loop within another loop.

nested-loop.py

```
i = 1
while i <= 3:
    j = 1
    while j <= 5:
        print(j, end='')
        j = j + 1
    print()
    i = i + 1
```

Sample Run 1

```
C:\> python nested-loop.py
12345
12345
12345
```