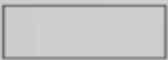
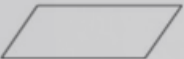


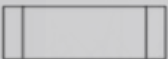
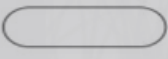
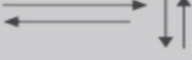
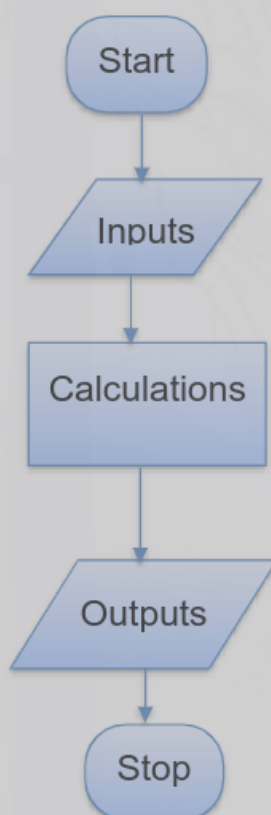


01 - Develops algorithms to solve problems and uses python programming language to encode algorithms

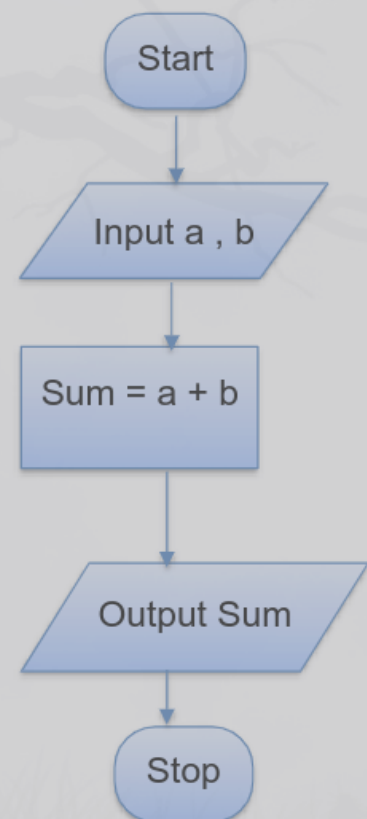
## Flowcharts

### Symbols commonly used in flowcharting...

Symbol	Name	Function
	Process	Indicates any type of internal operation inside the Processor or Memory
	input/output	Used for any Input / Output (I/O) operation. Indicates that the computer is to obtain data or output results
	Decision	Used to ask a question that can be answered in a binary format (Yes/No, True/False)
	Connector	Allows the flowchart to be drawn without intersecting lines or without a reverse flow.
	Predefined Process	Used to invoke a subroutine or an interrupt program.
	Terminal	Indicates the starting or ending of the program, process, or interrupt program.
	Flow Lines	Shows direction of flow.



Eg :



## Pseudocode

## Rules for writing Pseudocodes

- **Write one statement per line.**
- **Capitalize Initial Keywords:** Primarily **READ, WRITE, IF, ELSE, ENDIF, WHILE, ENDWHILE, REPEAT, UNTIL** are the keywords that must be written in **capital letters**.
- **Indent to show hierarchy:** Indent lines to make the Pseudocode easy to read and understand.
- **End multiline structures:** Every **IF** must be end with an **ENDIF**. Every **DO, DO WHILE** must end with **ENDDO**.
- **Keep statements language independent:** This rule does not mean that we can write our pseudocode in English, French, Russian, Chinese or whatever languages we know. **It refers to programming language.** Try to avoid the use of words peculiar (abnormal) to any programming language.

## ADVANTAGES AND DISADVANTAGES

### Pseudocode Disadvantages

- It's not visual
- There is no accepted standard, so it varies widely from company to company

### Pseudocode Advantages

- Can be done easily on a word processor
- Easily modified
- Implements structured concepts well

### Flowchart Disadvantages

- Hard to modify
- Need special software (not so much now!)
- Structured design elements not all implemented

### Flowchart Advantages

- Standardized: all pretty much agree on the symbols and their meaning
- Visual (but this does bring some limitations)

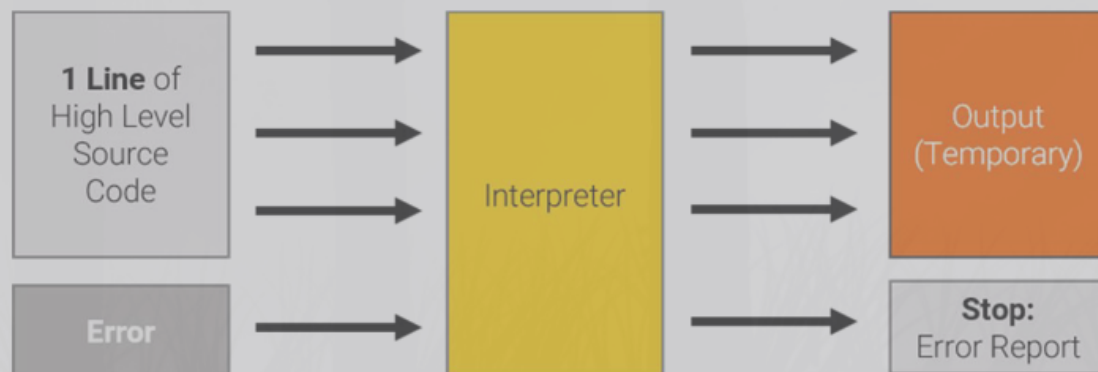
## Language translators

## Assembler

- Assemblers are used to translate a program written in a low-level assembly language into a machine code (object code) file.
- so, it can be used and executed by the computer.
- Once assembled, the program file can be used again and again without re-assembly.

## Interpreter

- Interpreter programs are able to read, translate and execute one statement at a time from a high-level language program.
- The interpreter stops when a line of code is reached that contains an error.
- Interpreters are often used during the development of a program. They make debugging easier as each line of code is analyzed and checked before execution.
- Interpreted programs will launch immediately, but your program may run slower than a compiled file.
- No executable file is produced. The program is interpreted again from scratch every time you launch it.



## Compiler

- Compilers are used to translate a program written in a high-level language into machine code (object code).
- Once compiled (all in one go), the translated program file can then be directly used by the computer and is independently executable.
- Compiling may take some time but the translated program can be used again and again without the need for recompilation.
- An error report is often produced after the full program has been translated.
- Errors in the program code may cause a computer to crash.
- These errors can only be fixed by changing the original source code and compiling the program again.



Translator	Examples
Compiler	Microsoft Visual Studio GNU Compiler Collection (GCC) Common Business Oriented Language (COBOL)
Interpreter	OCaml List Processing (LISP) Python
Assembler	Fortran Assembly Program (FAP) Macro Assembly Program (MAP) Symbolic Optimal Assembly Program (SOAP)

Compiler	Interpreter	Assembler
Translate high – level languages into machine code	Temporarily executes high – level languages, one statement at a time	Translate low-level assembly code into machine code
An executable file of machine code is produced (object code)	No executable file of machine code is produced (no object code)	An executable file of machine code is produced (object code)
Compiled program no longer need the compiler	Interpreted programs cannot be used without the interpreter	Assembled programs no longer need the assembler
Error report produced once entire program is compiled. These errors may cause your program to crash	Error message produced immediately (and program stops at that point)	One low-level language statement is usually translated into one machine code instruction
Compiling may be slow, but the resulting program code will run quick (directly on the processor)	Interpreted code is run through the interpreter (IDE), so it may be slow, e.g. to execute program loops	
One high-level language statement may be several lines of machine code when compiled		

## Linkers

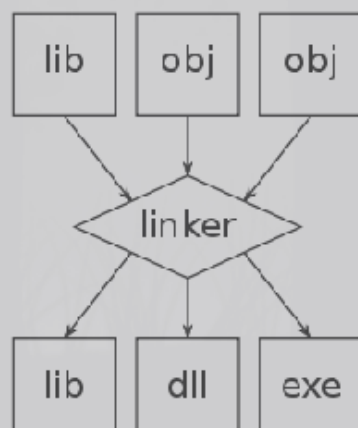
## Linker

A **linker** is a computer program that takes one or more object files generated by a compiler and combines them into one, executable program.

- Computer programs are usually made up of multiple modules that span separate object files, each being a compiled computer program.
- The program as a whole refers to these separately compiled object files using symbols.
- The linker combines these separate files into a single, unified program; resolving the symbolic references as it goes along.

### Functions of Linker

- For most compilers, each object file is the result of compiling one input source code file.
- When a program comprises multiple object files, the linker combines these files into a unified executable program.



## Syntax

# Operators

- Arithmetic

Operator	Name	Example
+	Addition	$x + y$
-	Subtraction	$x - y$
*	Multiplication	$x * y$
/	Division	$x / y$
%	Modulus	$x \% y$
**	Exponentiation	$x ** y$
//	Floor division	$x // y$

- Assignment

Operator	Example	Same As
=	$x = 5$	$x = 5$
+=	$x += 3$	$x = x + 3$
-=	$x -= 3$	$x = x - 3$
*=	$x *= 3$	$x = x * 3$
/=	$x /= 3$	$x = x / 3$

<code>%=</code>	<code>x %= 3</code>	<code>x = x % 3</code>
<code>//=</code>	<code>x //= 3</code>	<code>x = x // 3</code>
<code>**=</code>	<code>x **= 3</code>	<code>x = x ** 3</code>
<code>&amp;=</code>	<code>x &amp;= 3</code>	<code>x = x &amp; 3</code>
<code> =</code>	<code>x  = 3</code>	<code>x = x   3</code>
<code>^=</code>	<code>x ^= 3</code>	<code>x = x ^ 3</code>
<code>&gt;&gt;=</code>	<code>x &gt;&gt;= 3</code>	<code>x = x &gt;&gt; 3</code>
<code>&lt;&lt;=</code>	<code>x &lt;&lt;= 3</code>	<code>x = x &lt;&lt; 3</code>

- Bitwise

<https://www.youtube.com/watch?v=PyfKCvHALj8>

Operator	Name	Description	Example
<code>&amp;</code>	AND	Sets each bit to 1 if both bits are 1	<code>(a &amp; b)</code> (means 0000 1100)
<code> </code>	OR	Sets each bit to 1 if one of two bits is 1	<code>(a   b) = 61</code> (means 0011 1101)
<code>^</code>	XOR	Sets each bit to 1 if only one of two bits is 1	<code>(a ^ b) = 49</code> (means 0011 0001)
<code>~</code>	NOT	Inverts all the bits	<code>(~a) = -61</code> (means 1100 0011 in 2's complement form due to a signed binary number.)
<code>&lt;&lt;</code>	Zero fill left shift	Shift left by pushing zeros in from the right and let the leftmost bits fall off	<code>a &lt;&lt; 2 = 240</code> (means 1111 0000)
<code>&gt;&gt;</code>	Signed right shift	Shift right by pushing copies of the leftmost bit in from the left, and let the rightmost bits fall off	<code>a &gt;&gt; 2 = 15</code> (means 0000 1111)

# NOT

```
>>> a = 60
>>> ~a
-61
```

# Left shift

```
>>> a = 5
>>> b = 3
>>> a << b
40
```



```

bin_5 -> 101.0000
left_shift_by_3 -> 101'000'.0 # Moving the number 3 bits to the left (adding 3 0s)
                -> 101000.0

>>> int("1011000", 2)
40

```

```

# Right shift
>>> a = 15
>>> b = 3
>>> a >> b
1

bin_15 -> 1111.0
right_shift_by_3 -> 1.'111'0 # Moving the number 3 bits to the right (removing 3 b
                  -> 1.0

>>> int("1", 2)
1

```

- Precedence

Operator	Description
**	Exponentiation (raise to the power)
~ + -	Complement, unary plus and minus (method names for the last two are +@ and -@)
* / % //	Multiply, divide, modulo and floor division
+ -	Addition and subtraction
>> <<	Right and left bitwise shift

&	Bitwise 'AND'
^	Bitwise exclusive 'OR' and regular 'OR'
<= < > >=	Comparison operators
< > == !=	Equality operators
= %= /= //= -= += *= **=	Assignment operators
is is not	Identity operators
in not in	Membership operators
not or and	Logical operators

# Lists

- List methods

Method	Description
<code>append()</code>	Adds an element at the end of the list
<code>clear()</code>	Removes all the elements from the list
<code>copy()</code>	Returns a copy of the list
<code>count()</code>	Returns the number of elements with the specified value
<code>extend()</code>	Add the elements of a list (or any iterable), to the end of the current list
<code>index()</code>	Returns the index of the first element with the specified value
<code>insert()</code>	Adds an element at the specified position
<code>pop()</code>	Removes the element at the specified position
<code>remove()</code>	Removes the item with the specified value
<code>reverse()</code>	Reverses the order of the list
<code>sort()</code>	Sorts the list

```
>>> a = [1,2,3]
>>> b = a.copy()
>>> b
[1, 2, 3]
---
>>> a.extend([4,5])
>>> a
[1, 2, 3, 4, 5]
---
>>> c = ["a","b","c"]
>>> c.index("b")
1
---
>>> c.insert(1,"d")
>>> c
['a', 'd', 'b', 'c']
---
>>> c.pop(1)
'd'
>>> c
['a', 'b', 'c']
---
>>> c.remove("a")
>>> c
```

```
['b', 'c']
---
>>> h = [5,2,6,1]
>>> h.sort()
>>> h
[1, 2, 5, 6]
>>> g = ["b", "a", "B", "A"]
>>> g.sort()
>>> g
['A', 'B', 'a', 'b']
```

# Tuples

- Tuple methods

Method	Description
count()	Returns the number of times a specified value occurs in a tuple
index()	Searches the tuple for a specified value and returns the position of where it was found

# Dictionaries

- Dictionary methods

Method	Description
clear()	Removes all the elements from the dictionary
copy()	Returns a copy of the dictionary
fromkeys()	Returns a dictionary with the specified keys and values
get()	Returns the value of the specified key
items()	Returns a list containing the a tuple for each key value pair
keys()	Returns a list contianing the dictionary's keys
pop()	Removes the element with the specified key
popitem()	Removes the last inserted key-value pair
setdefault()	Returns the value of the specified key. If the key does not exist: insert the key, with the specified value
update()	Updates the dictionary with the specified key-value pairs
values()	Returns a list of all the values in the dictionary

```
>>> a = {"name": "kavi", "age": 19}
>>> a.get("name")
'kavi'
>>> a['name']
'kavi'
---
>>> for i in a.items():
...     print(i)
...
('name', 'kavi')
('age', 19)
---
```

## File handling

- Python has several functions for creating, reading, updating, and deleting files.
- Uses basic file operations (open, close, read write and append)
- The key function for working with files in Python is the `open()` function.

The `open()` function takes two parameters; *filename*, and *mode*.

There are four different methods (modes) for opening a file:

"r" - Read - Default value. Opens a file for reading, error if the file does not exist

"a" - Append - Opens a file for appending, creates the file if it does not exist

"w" - Write - Opens a file for writing, creates the file if it does not exist

"x" - Create - Creates the specified file, returns an error if the file exists

In addition you can specify if the file should be handled as binary or text mode

"t" - Text - Default value. Text mode

"b" - Binary - Binary mode (e.g. images)

- Example

```
f = open("n1.txt", "rt")
```

## DB connection

```
import mysql.connector

mydb = mysql.connector.connect(
    host="localhost",
    user="yourusername",
    passwd="yourpassword"
)

mycursor = mydb.cursor()

mycursor.execute("CREATE DATABASE school")
```

- Insert data

```
import mysql.connector

mydb = mysql.connector.connect(
    host="localhost",
    user="yourusername",
    passwd="yourpassword",
    database="school"
)
mycursor = mydb.cursor()
sql = "INSERT INTO student (regNo, name, address, contactNo, dob) VALUES (%s, %s, %s, %s, %s)"
val = ("1200", "John", "Highway 21", "0715623410", "2000-10-21")
mycursor.execute(sql, val)
```

To insert multiple rows into a table, use the *executemany()* method. The second parameter of the executemany() method is a list of tuples, containing the data that want to insert.

```
sql = "INSERT INTO student (regNo, name, address, contactNo, dob) VALUES (%s, %s, %s, %s, %s)"

val = [
    ('1201', 'Peter', 'Lowstreet 4', '0715874510', '2000-06-21'),
    ('1202', 'William', 'Central st 954', '0775857410', '2000-03-11'),
    ('1203', 'Viola', 'Sideway 1633', '0710055210', '2000-08-09')
]

mycursor.executemany(sql, val)

mydb.commit()
```

Note that for any command that has to update or add info, you need to use the `mysql.commit()` function to write the changes to the db

- Retrive data

```
import mysql.connector
```

```
mydb = mysql.connector.connect(  
    host="localhost",  
    user="yourusername",  
    passwd="yourpassword",  
    database="school"  
)
```

```
mycursor = mydb.cursor()  
mycursor.execute("SELECT * FROM student")  
myresult = mycursor.fetchall()
```

```
for x in myresult:  
    print(x)
```