4/4

1. Why? → To Solve problems

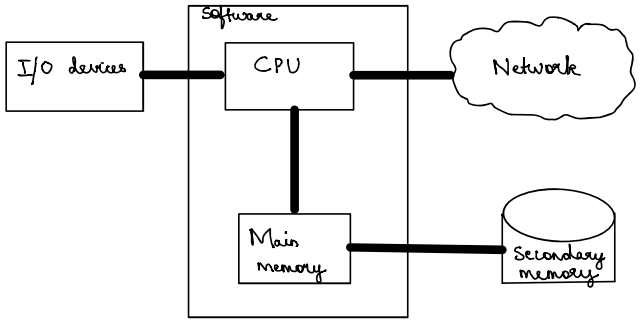
→ Choose programming language to solve required problem based on computational ability of the device, data Structures supported in the language & complexity of implementing algorithm.

→ Programmer's creativity & motivation.

art of programming ⟺ (dependent) Computer hardware Architecture

Program : Set of instructions written by programmer to solve problems by communicating with the CPU.

computer hardware architecture



Understanding Programming

→ look at info / data available & analyze it to build program to solve problem.

1. Syntax : grammar & vocabulary of the program
2. Algorithm : Step by step implementation of that idea to solve the problems.

Syntax : Characters, keywords & variables { Unlike other programming languages variables neednot be declared before its use in python }

Installing & working with python

→ Anaconda is a distribution of python & R programming languages for scientific computations that simplifies package mangement & deployment using package management system conda

→ Anaconda Navigator is desktop GUI include in Anaconda distribution to manage launch of applications & conda packages

→ Jupyter Notebook is browser based interactive computing environment for python programing

⇒ Python is command prompt with default editor IDLE = integrated development & learning environment which is the default IDE for python

Terminologies : Interpreters & Compilers

→ Python (high level language) —(interpreters)→ Machine level language

computers

Compiler : Translate the source code in high level programming language to executable files in machine level language.
source code extensions : .java .c .php
executable files extensions : .exe .dll

Interpreter : Parses code (does syntactic analysis) & interprets it immediately without the need for compilation.
i.e Each line of source code will give the output
Here, each line are logically related though they look independent.

5/4 2. Writing a Program
→ Source code extension for python file is .py
→ logically easier to debug ∴ we use python files for large programs

What is a Program

Program is a set of instructions intended to do some task. This task is a generic solution & can be used on different database & for different tasks.

Building blocks of a program
low-level conceptual structures to construct a programming language.

1. Input
2. Output
3. Execution → Sequential - from top
→ Conditional - if .... skips part of program based on condition
→ Repeated - for, while looping of the program
4. Reuse - Using functions

Mistakes while writing a program / Errors

1. Syntax Errors - Based on grammar
ex. Python is case-sensitive & prone to syntactical errors.

2. Logical Error - Grammar is correct but the program is not giving the intended output.
Not understanding the program correctly or typing mistakes

3. Semantic Error - Wrong variables, wrong use of operations and in wrong order.

4. Runtime error / Exceptions - wrong i/p & database connectivity problem
→ Solved by programmer i.e exception handling
ArithmeticError, FloatingpointError, EOFError (↳ end of file), MemoryError

3. Variables, Expressions and Statements

Values and types

→ Value is a basic thing like 1, 10.5, "hello"
→ type of the value is checked using the type function

```
>>> type("hello")
        <class 'str'>
>>> type(1)
        <class 'int'>
```

Variables

→ Variable is a named-literal that stores a value in the program
→ In python variable is not declared prior to use
→ assignment operator is used to assign a value to it.
→ Value assigned to the variable determines the type of variable

Variable names and keywords

→ Variable name should indicate its purpose
→ Python is case-sensitive
→ Variable names must not be keywords
→ Variable names can be uppercase, lowercase & numbers but must not start with numbers
→ '_' underscore is the only special character that can be used

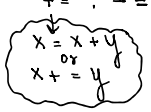
Statements

→ A statement is a small unit of code executed by python interpreter.
→ Program is a sequence of such statements

Operators & Operands

→ Operators are special symbols that perform specific tasks.
→ They work on single operand (Unary Operator) or two operands (binary Operators)

1. Arithematic operators + , − , * , / , % , //
   → Perform arithematic operations & give a value as output
2. Relational operator > , < , >= , <=
   → Compare & give Boolean value as output
3. logical Operators and , or , not , xor
   → perform logical operations & give 0 or 1 as o/p
4. Assignment Operator & Compound assignment
   = += , −=



Note
→ Python : can assign values of different types to multiple variables in a single statement.
→ Python support bitwise operations like &, |, ~, ^, >>, <<

→ Special operators of Python
identity Operator – is and is not
membership Operator – not and not in

Expressions → its a combination of values, variables and operators
→ Python interpreter evaluates simple expressions and gives results without print ( ).

Order of Operations [PEMDAS] - precedence rule

When 2 exponents are there its evaluated from right to left
>>> print (4 ** 3 ** 2) ; it is $4^{3^2} = 4^9 = 262144$
262144

(MD) → btwn mul & div whichever comes first is evaluated

(AS) → btwn add & sub whichever comes first is evaluated

String Operations → String concatenation is done using the + operator

```
>>> x = "32"
>>> y = "45"
>>> print (x + y)
3245
>>>  z = y + x
>>> print (z)
4532
>>> type (z)
Str
```

Asking user for an input

→ input ( ) is used for user input
→ a message can be inside
→ /n is used for a new line
This is assigned to a variable as
ex: Str1 = input (' Enter a sentence')
type (Str1)

→ The inputted value is always of the type string. To get other types the type is described at the start of the input

```
ex. >>Int_Value = int ( input (" Enter a number:"))
    Enter a number : 10
>>>type( int_value)
    int
```
→ int(), char(), float()

Comments
→ single line comments starts with #
→ Multiline comments use """ . . . . . """

mnemonic variable names
→ Memory aid.

Debugging → Correction of Errors.

3. Conditional Execution

Boolean Expressions : They belong to class bool and give output as True or False

bool
True
False

Difference between equality test and identity test
→ Based on memory location

identity test – is & is not does the identity test i.e two objects are same if they share the same memory location

Logical Operators and , or , not

6/4/21

3. Conditional Execution

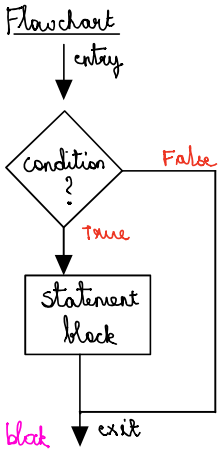
1. Basic level is if statement

Syntax

```
if condition :
    statement block
```

ex :
```
x = 10
if x < 40 :
    print ('Fail ")
```

Flowchart



note : → Pass can be used in statement block
→ Just passes over the execution.

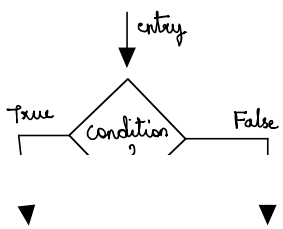
2. Alternative Execution : If - Else condition

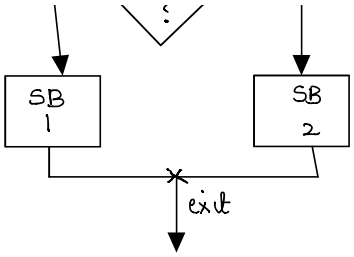
It has two branches / 2 statement blocks. If the condition is true one statement block is executed if its false other S.B will be executed.

Syntax :

```
if condition :
    SB1
else :
    SB2
```

flowchart





note: To check if input is even number

```
x = int( input (" Enter a number ")
if x%2 == 0 :
        print ("x: ", x , " is an even number ")
else :
        print ("x: ", x, " is not an even number ")
```

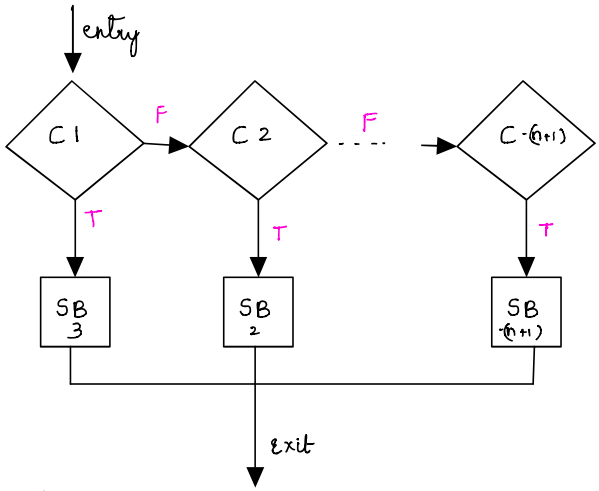
3. Chained Conditionals

ie more than one branch

Syntax

```
if condition1 :
        SB1
elif    C2 :
        SB2
elif    C3 :
        SB3
        .
        .
        .
elif    C-n :
        SB-n
else    C-(n+1)
        SB -(n+1)
```

flowchart




```
x = float (input ("Enter class marks")

if   x >= 80 :
        print (" first class & distinction")
elif   x >=60  and  x < 80 :
        print (" first class ")
elif   x >= 50  and  x < 60 :
        print (" Second class ")
elif   x >= 35  and  x < 50 :
        print (" third class ")
else :
        print (" Fail ")
```

4. Nested Conditionals

One condition statement is nested within another conditional statement
Its advised to use logical operators instead of conditional statement

ex:
```
x = float(input("Enter marks"))
if x>=60:
    if x>70:
        print("distinction")
    else:
        print("first class")
else:
    if x>50:
        print("second class")
    else:
        print("Work harder")
```

Catching expressions using try & except

Handling an exception like runtime error using try is called catching.

- → There is a suspicious code
- → Write the suspicious code within the try block
- → if an error occures it will go to except
- → Except block should contain remedy for the suspicious code.

→ Provision for ending a problematic code gracefully.

Short - circuit evaluation of logical Expressions

→ It saves computational time
→ It leads to a technique known as (gaurdian pattern)

⇓

Prevent error by strategically placing a gaurd evaluation before an evaluation that might cause error & using (and) logical operation.

Debugging : Traceback

Traceback is stack trace from the point of error occurence down to call sequence till the point of the call.

- → What kind of error it is
- → Where it occured

} needed for debugging.

5. Functions

Function is a sequence of instructions intended to perform a specific task.

a) Function call

→ function is given a specific 'name'

→ To use a function we call it by its name
→ Usually a function takes zero or more arguments and returns the result.

arguments : These are values given to the function.

b) Built - in functions

→ Programmer/ user need not know its internal working & only its purpose & name is required to use it.

ex max(), min(), len()

c) Type conversion function

Converts one type of value to another
* literal string conversion to int/ float is not allowed.

d) Random Numbers

deterministic function → pre-defined i/p from a range of inputs and a expected o/p
Non-deterministic function → Unexpected o/p

Generation of Pseudo - Random Numbers

module - random
functions → random() : gives a random floating point btwn 0 & 1 (No arg)
→ randint() : takes ↑ & ↓ No & gives random No btwn that (2 arg)
→ choice() : takes a list & gives one value from the list randomly (1 arg which is a list that can have many value)

e) Math functions

Module 'math' has a rich set of math functions
functions - sqrt(), pi(), log10() (of base 10), log() (of base e), sin(), cos() (radian value), pow(arg1, arg2) → (arg1)^arg2

To get correct value of trigonometric functions following is used.
to get sin(a) where a is in degree

```
math.sin(a * math.pi/180)
```

User - defined functions : Adding a new function

Syntax :

```
def <fname>(arg list):      } function header
    <Statement 1>           }
    <Statement 2>           } Body of the function
    :
```

:

< Statement n >
return <value> # optional statement

→ A function need to be defined before calling it
→ To call a function just use its name.

function definition creates an object of type function
↓
user defined variable

⇒ Execution of program is sequential.

Parameters & Arguments

→ A function may take arguments as input from calling functions.

parameter : is used while defining a function
It determines the variable type / memory space required

argument : Inputs given while calling the function
They are substituted / used in place of parameters in the actual function.

imp

Fruitful functions & Void Functions

Void function : Performs some tasks but doesn't return any value to the calling functions

Fruitful functions : Performs tasks & returns a value to the calling functions

imp

→ Void functions are called by just using their name & result will be seen.

→ If we call fruitful functions with just their name, we cannot see the result Though the function execution was successfull.

To see results we need to assign a variable & print that variable

Why functions ?

→ Easier to create, use, read & debug
→ Smaller program i.e eliminates repetitive code
→ Reuse.