Machine learning

28-6-22

Problem approach

Analyze → Think of solution → Write it down → Code it

(IMP)

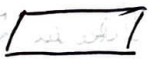
→ Method 1 : Flow chart

→ 2: Document designing

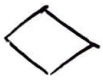
Flow chart - diagram, a solution to a problem

// break down the steps / process

Terminator / Beginning → 

I/P / O/P → 

Process → 

Decision → 

Flow → 

Python

print command :

print("...") / print('...') // only for text

By default it adds a new line in the end.

Variables:

// In jupyter, printing without
→ print without command
→ result of last line gets printed as o/p of a particular cell without print command.

print (a variable_name) → prints the stored value from the variable name

Naming convention -
→ should not start with digit.
→ start with (lower, upper) case, underscore.

// In Python, # - used for comments.

No need to assign datatype before name (like in Java).
Python automatically detects the datatype. It doesn't assign a storage

eg: a = 10;
a = 20

Both 10 & 20 are stored in different storage, just the address is changed after re-initializing.

a —address X→ [10]

after re initializing,

a —address ✓→ ['Hello']

type (variable) - returns the data type of a variable.

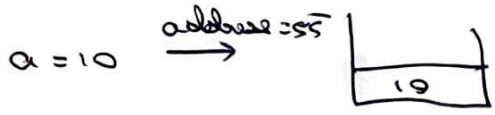
Data type:

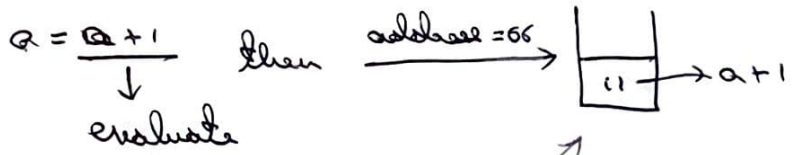
int, float, complex.

eg: a = 10
a = a + 1

a?

~~Here, if a is stored at (say) address (say 55)~~





Now for printing → it points to this address

So, the address itself is changed.

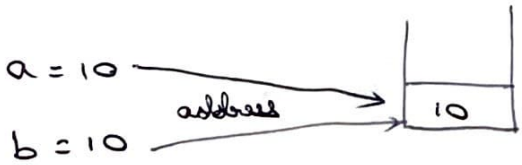
Proof - use id() → a unique number associated with storage.

Optimization:

if 2 ~~values have~~ variables have same value then,

both will have same id.

Since, both are referring to same location.



But it is not true all numbers, only for no. b/w -5 to 256.

because this range is common.

Arithmetic operators

+ - * /

// → integer division

** → exponential

% → modulus operator

Precedence

(), *, /, + / - ....

User input

input() → datatype(input())

eg: int(input())

any text inside it within quotes.

→ Usually gives o/p in a string format.

Boolean

value → true, false

True, False

Python correct format

Relation operators

>, <, >=, <=, ==, !=

Logical operators

combining multiple boolean inputs.

and - both have to be true.

T T → T

T F, F T → F

F F → F

or - one of them true.

T T → T

T F, F T → T

F F → F

not - negates the values.

opposite value

If - else

if condition:
    indentation (before) indentation

Same for else:
except condition is not required.

else not necessary for 'if'.

User can use multiple elif in b/w if & else.

if

Mandatory if block required.

Order matters of if - elif .... - else (optional)

Nested if

```
if {
    if...
    if...
    :
    else (optional)
  }
:
else (optional)
```

T, T ✓ {

F & T ⇒ F ✗

F & T & T

F & F ⇒ F ✗

C

while loops

→ repeated works.

```
while cond:
    - - -
    - - -
```

For loop

```
for i in x:
```

- `i` → iterating variable
- `x` → sequence (collection of multiple things)

range (start, stop, step) → prints a sequence of nos.
or increment

- step value / increment by a specific value
- starts from 'start' & ends at stop - 1

another way → range (stop) // by default start = 0 & step = 1

break

terminates a loop by an explicit cond other than the regular terminating cond.

Nested loop, break will terminate the inner loop.

Else

can be used with loops (while, for) in Python.

eg: while cond:
    ⋮

else: → when while cond becomes false / loop is over.
(runs only once)

Useful when using break inside a loop. Then, the else part will not be executed.

Continue

Used to skip an iteration.

Pass

Pass a loop 'pass'

Used inside loop, conditional stats.

No body instead write pass.

Functions -

allows create a block of code which can be called from anywhere

Syntax:

```
def fn_name():
    body
    ...
    return variable_name
```

→ i/ps

```
fn_name(user_i/p, whatever_u_give) // fn call
```

def fn_name(i/p) ≠ fn_call(i/p)

both are different

value is passed

Uses.

→ Improve readability

→ Avoid ~~repetition~~ repetition

→ Improve testing

fn calling working → FILO (first in last out)

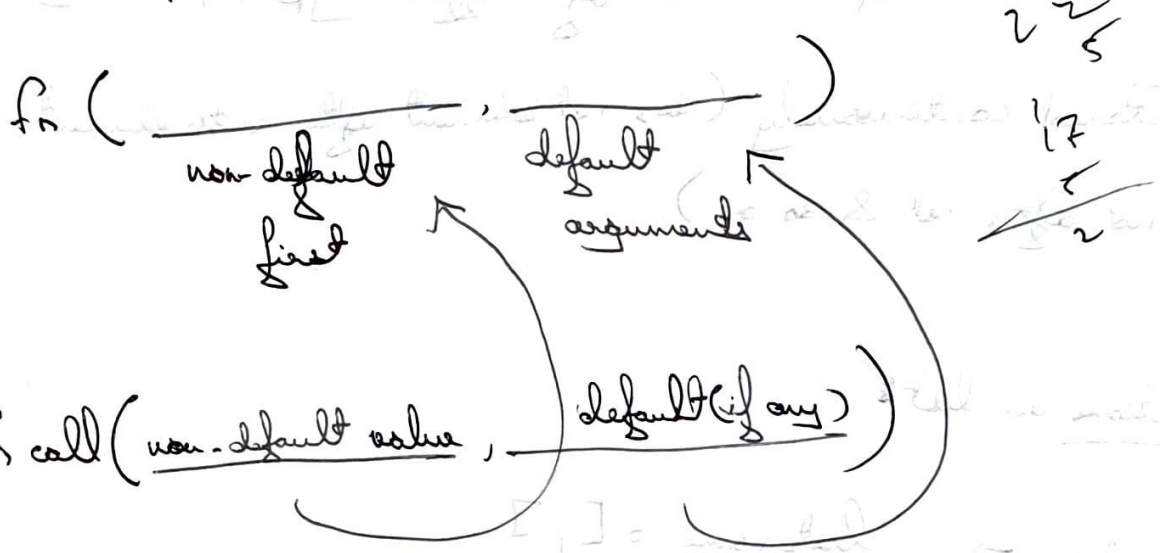
ie, 1st called fn exits in the last position.

Uses a call stack.

Scope & lifetime of a variable

- local variable - defined inside a fn. available only inside that fn.
- global variable - " outside " " , available anywhere

Default argument



Also, we can specify an argument which takes a value from fn call → fn def but we must ensure that the non-default arg are passed correctly.