if is dangerous class notes

if else and elif is branching of the execution flow. when they are simple the conditional code is not too confusing. its okay but when the conditonal variables count becomes too high or the range of them is too vast the code becomes confusing for our future self as well as fellow mebers. to avoide this the code must follow the concept of **perceived linearity** that is the main flow of code execution must be perceived as linear. the conditonas required for the logic of algorithm will still remain but only that they are hanndled in a more readable and boservably linear manner.

methods used for them are moving the code along the repo or transform it.

in case of the conditonal variables being catagorical if normal if else we use ==. here we create a register/map/dict of the conditon as key to the related code to be executed as value. then the conditon hanling method need only use this register as a way of handling the various categories.

e.g.,

```
class reg/map:
    pass in key and functionality corresponding to it.they are stored as key value pair


Reg/Map compute:
    ADD:addresstoaddmethod()
    subtract:addresstosubtractmethod()
    .
    .
    .

key =input operator
list = operand lists
reg/Map.compute([key])(list) will select the method to be applide on list of inputs
```

another way when handling categories that share handling logic is to split the categories according to the handling logic first and then itteraeate over this split data and apply the handling logic on them seperatly than conditonally applying them to each value accoring to their category

```
for value in all_values:
    if value in cat_1:
            handling statements for 1
    elif value in cat_3:
            handling statements for 3
    elif value in cat_2:
            handling statements for 2
```

this becomes bug pron as the categories as well as the handling steps increas so rahter use.

```
for value in all value:
    split to the respective categories 1,2,3....

for element in category 1:
        handling statements for 1
for elemnt in cat_3:
            handling statements for 3
for element in cat_2:
            handling statements for 2


.
.
.
```

for countable finite type variables for the conditonal split we can use above methods or when the count is too large use seperate function to handle the condional check by which we move the branching of flow from the main flow of execution and make it a sub module that can be eaisly explored on its on. normally one would need to use == as well as </>= checks with these type of operators in if.

for infinite type variables like float on would need to use </>= as well as isclose like methods. these too are moved into a seperate sub module or fucntion to imporve readability. and independant exploration.