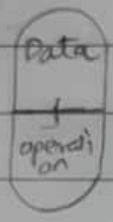
01/10/22

* Object Oriented Programming
  - bottom up approach.
  - o/p → process → i/p.

⇒ Encapsulation
  - Process to combine data & operation on data in a unit. (what data & operation to give to user)

construction of application



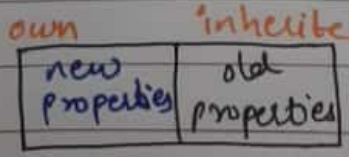
eg: food + containers + spoon + ...

⇒ Abstraction
  - Process to hide complexity & give access in simplest manner.
  - i.e hides the data & gives the access to the operation only.
  - It gives methods to access the data that is hidden using encapsulation.

abstracted backend (internal hidden)

GUI = access.

⇒ Inheritence
  - Process of creating new entity ~~with~~ by taking reference and properties from pre-existing entity.



⇒ **Polymorphism**

- ability to exist in multiple form
existing in different forms.

→ **overload**

- 'n' or multiple functions with single name
eg. ~~text~~ messages in the form of text/audio/video

→ **over-ride**

- adding new feature to preexisting by recoding.
- takes ref from existing, but the code is different & we write everything again.

eg. those messages can be now be reacted to.

⇒ **Class & Objects.**

Class	Object
- is virtual idea	- physical implementation of the idea
- gives framework	- implements / codes the framework
- has properties	- has values
- has operation on data / properties	- has methods to access values

human class → n objects

- gender →
- name → } values
- height →

setDetails()
display()
→ methods to access those values.

set
reset
edit

* 1 Class can have 'n' objects

→ Every property is created when you call a respective methods for it.

① Read in method.

```
class Human :                          → constructor → calls itself
                                       → reference of a caller
    def __init__(self):
        self.gender = input("gender :")

h = Human()  → input gender            //object h
h.gender     → printing
```

→ self - that activates the method
- reference of a caller (stores address of object)

self. x - goes to respective object

diff object has different reference id

* anything starting with & ending with double underscore is system or internal method.

__ __ ↑ system/ internal

★ **Abstraction** — can decide what access to give

3 access specifiers

- public → default → reflected & accessed
- private → _var/_method() → not reflected but accessed
- strong private → __var/__method() → not reflected & not accessed outside class but can be accessed by member of class.

```
class Human:
    gender = "female"   # public
    _name = "shruti"    # private
    __number = 123      # strong private
```

Humane.gender → female
Human._name → shruti
Human.__number → error - because not accessed outside class

→ to access strong private
→ member of Human

```
def access():
    print(Human.gender, Human._name,
          Human.__number)
```

Human.access() → prints everything

* def __init__(): Constructor that is called automatically.
called when object is created

* def __del__(): Finalizer called before object is removed.
called when object goes for garbage collection

* def __str__(): Prints details via returning string.
can only return
- str method would return string which gets internally called when we call from the print

★ Collection of object

→ list of objects

① create object ← loop
② append in list
③ continue till data exists

★ Membership

```
class Human :
    count = 0
    def __init__(self, name, gender):
        self.name = name
        self.gender = gender
        Human.count += 1
```

class member: Human.count

object member: self.name, self.gender

local variables: name, gender

→ class members can be accessed by objects maintained by class itself.

3/10/22

setattr
getattr } Null class
pass

Pass statement
- no operation statement
- used in class, function, loop, def of class, condition
- used when you have nothing to do as of now.
- It is used as filler, pass will be replaced later with needed logic.

Null class
- class without attributes i.e no properties.

* ⇒ setattr & getattr will affect the object who have called and not on any other objects.

⇒ setattr (obj, property, value)

class Human:
    pass

setattr (h1, name, "xyz")
setattr (h2, gender, "male")

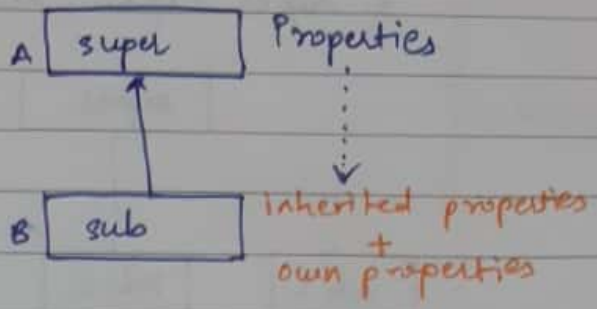
h1.name
h2.gender

- security is not concerned
- ∴ strong private variables are also accessed directly.

⇒ getattr (obj, property)
↳ o/p → its value.

⇒ # Inheritance

advantages
- faster development
- standardization
- Re use
- Enhancement



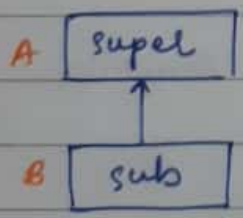
→ public & private properties will be inherited
→ but strong private properties will NOT get inherited

Python syntax:
write super class in bracket of sub class
A - super
B - sub
Class B(A): ↑ inherited
#

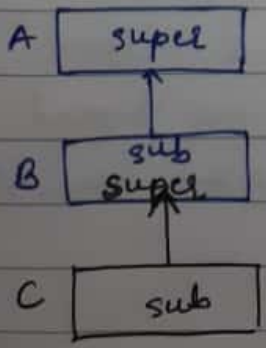
⇒ Types

1) Single level.

class A:
class B(A):



2) Multilevel.



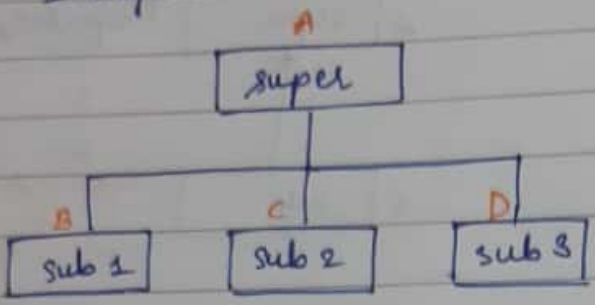
A created B
B created C

class A:
class B(A):
class C(B):

A is grandparent of C
∴ C has properties of B & A both

3) Hierarchical

1 super class, multiple sub class

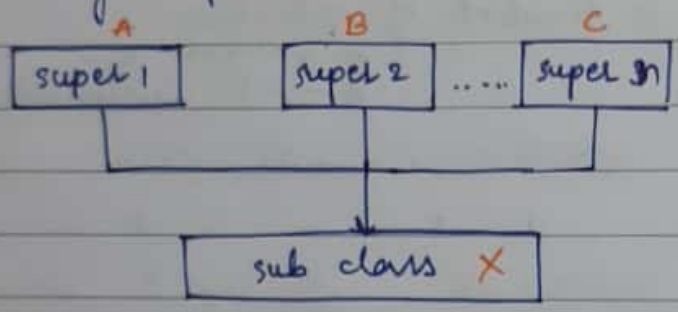



```
class A :
class B(A) :
class C(A) :
```

4) Multiple

many super class, one sub class

not supported by Java

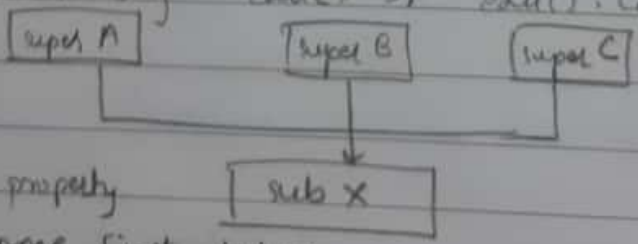



```
class A :
class B :
class C :
class X(A,B,C) :
```

Drawback: When 2 or more has same method, whose method to inherit?

- This is ambiguity
  - ↳ This is handled using FCFT (First come first taken)

eg edu() : Eng edu() : Dr edu() : CA



for common property

↳ First come First taken

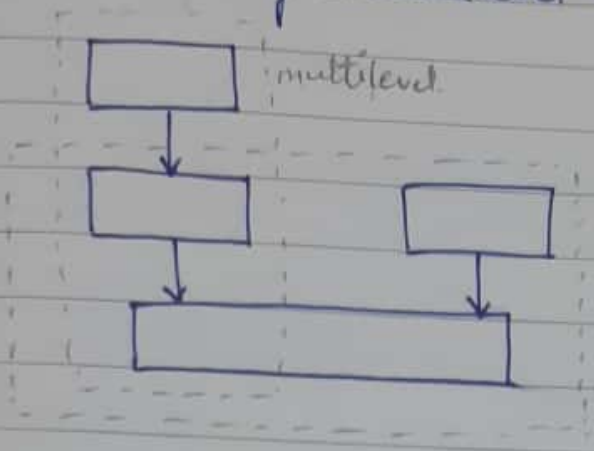
class X(A,B,C) : A taken, B & C skipped

class X(B,A,C) : B taken, A & C skipped

class X(C,A,B) : C taken, A & B skipped

5) Hybrid

combination of multilevel & multiple.




```
class A :
class B(A) :      } multilevel
class C(B) :
class X :

class Main (C, X) :
```

- multiple + multilevel C

* Override

- When sub class recodes one or more methods of parent / super class, it will reject inheritance & local methods will overtake / overrides parents methods.
- Codes own method, so it won't accept parent's methods.
- not changing every method, only upgrading the methods that needs to be change.

* Super() : method
super activate its immidieate super class

4/10/22

→ Error
- Some wrong input or step, or some glitch in code due to wrong input, logical mistake or due to some hardware failure.
One can't have normal flow of execution

report ↓

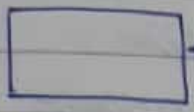
→ Exception
- Way of reporting error
- Mechanism to report
- Error caused system, creates report of it & it is called raising exception.

handle ↓

→ Exception Handling
- Trying either continue normal/alternate flow ffs or safe close in case of error.

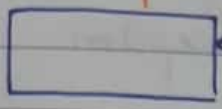
mandatory

try :

 ← code that can cause error

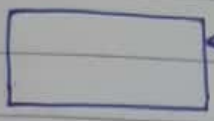
mandatory

except Exception :

 ← code to report & handle exception

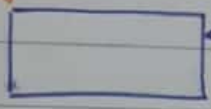
optional

else :

 ← code to run if & only if no error

optional

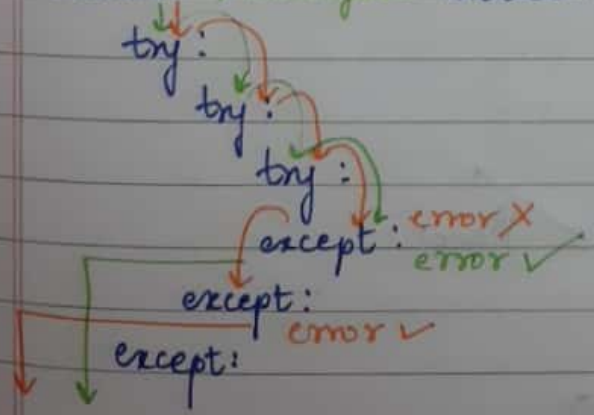
finally :

 ← code to run in any condition
(code to safeclose using file/ DB/ Device)

→ the code won't stop after exception handling unless we want to. even after exception handling, normal code can work.

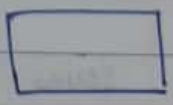
→ Try and finally only can also work.
- prints finally in any case → so it executes first, then finds for exception of the error

→ Follows Waterfall model.

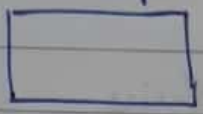


→ Multiple Exception

try :



except Exception 1 : most special



except Exception 2 : general exception

⋮

→ Raising own exception

(1) Create class that inherits exception
(2) Code __init__ to take value / message of exception
(3) Code __str__ to print message
(4) Use raise to raise own exception

✱ Pandas

Its a special package used over,

1D → Series
2D → Data frame
3D → Panel.

⇒ import pandas as (pd) → can be any variable its an alias.

⇒ Series

S = pd.Series([11, 22, 33])

- Series supports scalar operation on itself.
- S * 2 → multiples each element of series by 2.
- S / 2 → divides each element of series by 2.

(for list, it repeats)

- s = pd.Series([11, 2.2, 3.10, True, "abc"])
- s * 2

o/p →
22
4.4
6.2
2 ⟶ True is (1) → multiplied by 2 give (2)
abcabc ⟶ works like string.

→ We don't support negative index
s[-2] ⟶ gives error
- s[::-1] → slicing reverses the series.
- s[::] & s[2:4] also works.

⇒ Dataframes
- It is 2-D labled data structure ~~with or~~ in tabular form with different types.
- works with csv files → csv is comma-seperated values (comma delineated)
  ↳ can be created using any text editor (excel)
- It is like dictionary.

- .append adds new data & creates new instance

6/10/22

✱ Numpy.

import numpy as np → alias (can give any name)

- 1D list [ ]
- 2D list [[ ], [ ]]
-

np.array ( ) — to ~~Code~~ make array.

→ properties -

.shape → gives rows only. (always count each element in a row
iff rectangle, then gives rows & column

.ndim → n dimension 1D / 2D / 3D
iff rect / square (n×m)

.size → gives size
if Rect / square → n×m
else gives only Row

1D → a = np.array([10, 20, 30])
a.shape → 3

2D → a = np.array([[10, 20, 30], [40, 50, 60]])
a.shape → (2, 3)
rows cols