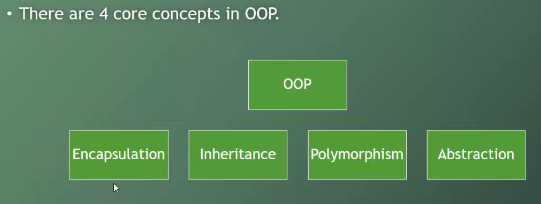
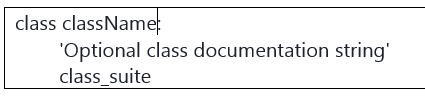
## OOP



### Class concept

পাইথনের ক্লাস হচ্ছে C++ এবং Modula-3 এর ক্লাস মেকানিজমের সংমিশ্রণ।পাইথনে ক্লাস ডিফাইন করার জন্য “class” কি-ওয়ার্ড ব্যবহার করা হয়।

ক্লাস এক্সপ্রেশনঃ



ক্লাসের ডকুমেন্টেশন স্ট্রিং কে চাইলে className.\_\_doc\_\_ দিয়ে এক্সেস করা যায়। class\_suite ব্লকে ক্লাসের সব কম্পোনেন্ট স্টেটমেন্টস লিখা থাকে।

**ক্লাস ভ্যারিয়েবল**

ক্লাসের যে ভ্যারিয়েবল গুলো ওই ক্লাসের সকল অবজেক্ট ব্যবহার করতে পারে সেগুলোই হল ক্লাস ভ্যারিয়েবল।ক্লাস ভ্যারিয়েবল ক্লাসের ভেতরে ডিফাইন করা হলে ও যেকোন মেথডের বাহিরে ডিফাইন করা হয়।

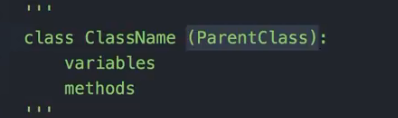
**ইন্সটেন্স ভ্যারিয়েবল**

যে ভ্যারিয়েবল ক্লাসের কোন মেথডের ভেতর ডিফাইন করা হয় তাকে ইন্সট্যান্স ভ্যারিয়েবল বলে।

**ডেটা মেম্বার**

ক্লাস ভ্যারিয়েবল এবং ইন্সট্যান্স ভ্যারিয়েবল কে ওই ক্লাসে ডেটা মেম্বার বলে।

কোন ক্লাস কোন ক্লাসের চাইল্ড ক্লাস বা কোনটি কোন ক্লাসের অবজেক্ট তা চেক করার জন্য issubclass() ও isinstance() দুটো বিল্ট ইন ফাংশন ব্যবহার করা হয়।



### Method concept

পাইথন ক্লাসের কোন ফাংশন যখন কোন অবজেক্ট দ্বারা কল করা হয় তখন সেটি মেথড হয়ে যায়। এখানে ফাংশন এবং মেথডের তফাৎ টা হচ্ছে মেথড ফাংশনের চাইতে একটি আর্গুমেন্ট বেশি নেয় এবং সেটি হচ্ছে সেই অবজেক্ট।**মেথডের এক্সপ্রেশন**

Class.method(object, \*args)

এই ব্যপারটাই ঘটে যখন আমরা কোন অবজেক্ট থেকে কোন মেথড কে কল করি।

object.method(\*args)

অর্থাৎ, কোন একটা ক্লাসের মেথডের প্রথম প্যারামিটার হচ্ছে সেই অবজেক্ট নিজেই।

কোন অবজেক্টে ক্লাস অ্যাট্রিবিউট কল করার সময় যদি সে অ্যাট্রিবিউটি একটি ফাংশন(মেথড) হয় তখন যে আর্গুমেন্ট গুলো পাস করা হয় সেগুলো সে অবজেক্টের মধ্যে অ্যাসাইন হয় না যথক্ষণ না আমরা self ব্যবহার করে তার মধ্যে ভ্যালু টা পাস করি। খুব গোল মেলে লাগছে নিশ্চয়ই!

### Object concept

কোন ক্লাসের অবজেক্ট হল সেই ক্লাসের সকল প্রপার্টি ধারন করা একটি ভ্যারিয়েবল। অবজেক্ট কে ইন্সট্যান্স ও বলা হয়। প্রত্যেকটি অবজেক্ট ই একটি নির্দিষ্ট টাইপের। ক্লাসের অবজেক্ট-কে দুই ভাবে তৈরি করা যায়। অ্যাট্রিবিউট রেফারেন্স ও ইন্সটেন্সিয়েশন। অ্যাট্রিবিউট হল কোন ক্লাসের ভেতরে থাকে এর প্রোপার্টিস, মেথডস ইত্যাদি।

অ্যাট্রিবিউট রেফারেন্স এর ক্ষেত্রে আমরা যখন কোন ক্লাসের অবজেক্ট তৈরি করি তখন সেটি একটি ব্ল্যাংক অবজেক্ট থাকে যতক্ষন পর্যন্ত না আমরা সে ক্লাসের কোন ফাংশন / মেথড / অ্যাট্রিবিউট কে কল না করি।

### Inheritance concept

অবজেক্ট ওরিয়েন্টেড প্রোগ্রামিং এর সবচেয়ে গুরুত্বপূর্ণ বিষয়টি হচ্ছে ইনহেরি -টেন্স। ইনহেরিটেন্স অর্থ উত্তরাধিকার। প্রোগ্রামিং এ একটি ক্লাসের ডেটা অন্য ক্লাস উত্তরাধিকার সূত্রে ব্যবহার করার পদ্ধতিকে ইনহেরিটেন্স বলে ৷ যে ক্লাসের ডেটা ব্যবহার করা হয় তাকে প্যারেন্ট ক্লাস বা বেজ ক্লাস বলে। আর যে ক্লাস ডেটা ব্যবহার করে তাকে চাইল্ড ক্লাস বা ডিরাইভড ক্লাস বলে। ইনহেরিটেন্স প্রোগ্রামে রিডান্ডেন্সি কমায়, প্রোগ্রামকে লাইট ওয়েট করে।

#### Types of inheritance

##### Single

##### Multiple

##### Multilevel

যদি একটি ডিরাইভড ক্লাসের ও আরও ডিরাইভড ক্লাস থাকে তখন একে মাল্টি লেভেল ইনহেরিটেন্স বলে।

class Father:

    def hero(self):

        print( "“Every father is a hero”")

class Mother:

    def warrior(self):

        print("“Every mother is a warrior")

class Child(Father,Mother):

    def life(self):

        print("“Child’s life is inherited from Father and Mother”")

we = Child()

we.life()

we.hero()

we.warrior()

class shape:

    def \_\_init\_\_(self,x,y):

        self.x=x

        self.y=y

    def area(self):

        print("i am area")

class triangle(shape):

    def area(self):

        area=.5\*self.x\*self.y

        print("area of triangle: ",area)

class rectangle(shape):

    def area(self):

        area=self.x\*self.y

        print("area of rectangle: ",area)

sos=triangle(2,3)

sos.area()

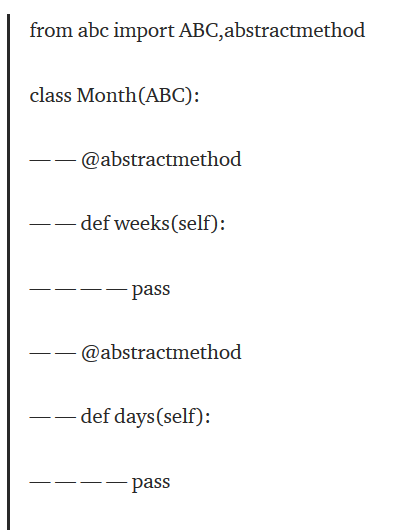
tos=rectangle(5,5)

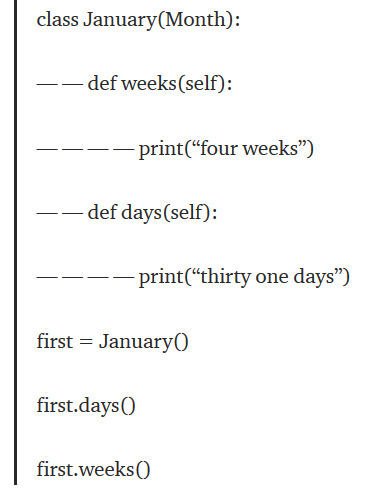
tos.area()

### Abstraction concept

পাইথনে অ্যাবস্ট্রাকশন মেনে কাজ করে জন্য অ্যাবস্ট্রাক্ট ক্লাস ও ইন্টারফেস ব্যবহার করতে হয়।

অ্যাবস্ট্রাকশন ক্লাস হল এক বা একাধিক অ্যাবস্ট্রাক্ট মেথড নিয়ে গঠিত ক্লাস যার মেথড গুলো সম্পূর্ন রুপে ইমপ্লিমেন্টেড থাকে না। অ্যাবস্ট্রাক্ট ক্লাসের মেথড ইমপ্লিমেন্টেশনের জন্য ওই ক্লাসের সাব ক্লাস তৈরি করতে হয়। পাইথনে বিল্ট-ইন কোন অ্যাবস্ট্রাক্ট বেজ ক্লাস নেই তবে (Abstract Base Classes ) abc মডিউলের মাধ্যমে প্রোগ্রামে অ্যাবস্ট্রাকশন ব্যবহার করা যায়।



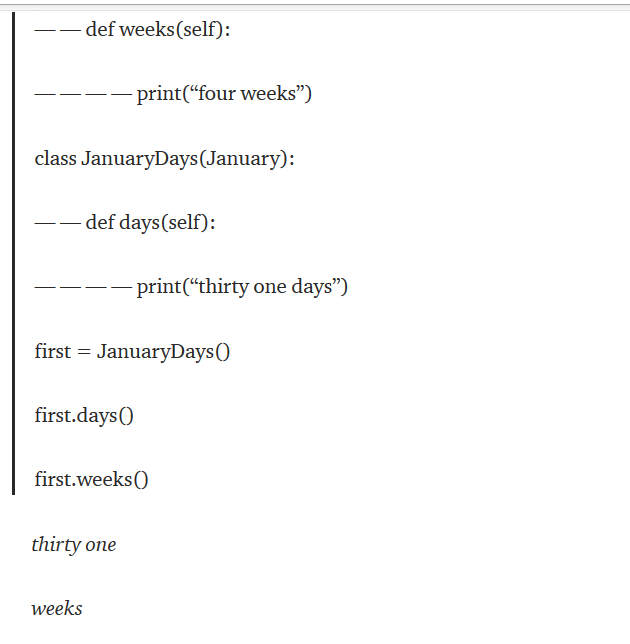


আমরা যদি অ্যাবস্ট্রাক্ট ক্লাসের days () মেথডটি January ক্লাসে ইমপ্লিমেন্ট না করি তাহলে আমাদের এরর দেখাবে। কারণ অ্যাবস্ট্রাক্ট ক্লাসের সবকটি মেথড ডিরাইভড অভাররাইড করা না হলে ডিরাইভড ক্লাসটির অবজেক্ট তৈরি করা যায় না। তাই আমরা এমন এরর পাব…

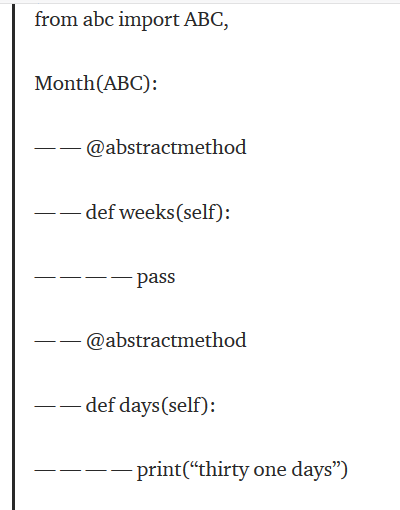
TypeError: Can’t instantiate abstract class January with abstract methods days

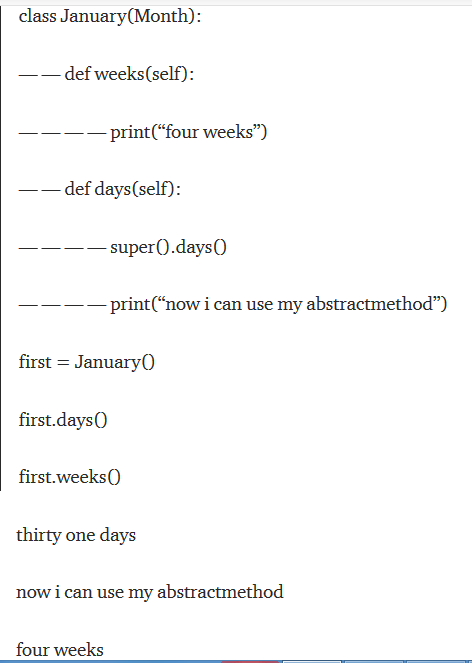
তবে আমরা চাইলে আলাদাভাবে একাধিক ক্লাস ইমপ্লিমেন্ট করতে পারি।

class January(Month):



আমরা অ্যাবস্ট্রাক্ট মেথড কে ও ইমপ্লিমেন্ট করতে পারি এবং সে মেথড কে ডিরাইভড ক্লাসের ভেতর ব্যবহার করার জন্য super() মেথড ব্যবহার করতে হয়। কেন করতে হয় সেটা পরে আলোচনা করা যাবে।





অ্যাবস্ট্রাক্ট ক্লাসের অবজেক্ট তৈরি করা যায় না। কারণ অ্যাবস্ট্রাক্ট ক্লাসের মেথড গুলো অসম্পূর্ণ। কোন অবজেক্ট তৈরি করা হয় তাহলে প্রকৃতপক্ষে তার কোন অ্যাট্রিবিউট থাকছে না যা ব্যবহারযোগ্যা।

a = Month()

TypeError: Can’t instantiate abstract class Month with abstract methods days, weeks

Jokon kono method er kono body or output show korar kono kisu take na ,seti abstract method and sey method Jodi kono class er vetore take sey class k abstract class bole . Given belo an example ..

class student:

    def \_\_init\_\_(self,name,roll):

        self.name=name

        self.roll=roll

    def fun1(self,year):

        self.year=year

        pass

class one(student):

    def fun1(self):

        print("name is : ",self.name , "roll is : ",self.roll)

sos=one("manhmud",123)

sos.fun1()

#on the above method fun1() is an example of abstruction method ,cause this method we did not use,just use only overloading funtion

from abc import ABC,abstractmethod

class shape(ABC):

    def \_\_init\_\_(self,x,y):

        self.x=x

        self.y=y

    @abstractmethod

    def area(self):

        print("i am area")

class triangle(shape):

    def area(self):

        area=.5\*self.x\*self.y

        print("area of triangle: ",area)

class rectangle(shape):

    def area(self):

        area=self.x\*self.y

        print("area of rectangle: ",area)

sos=triangle(2,3)

sos.area()

tos=rectangle(5,5)

tos.area()

from abc import ABC,abstractmethod

class father(ABC):

    @abstractmethod

    def fun1(self):

        #print("do some thing")

        pass

class son(father):

    def \_\_init\_\_(self,name):

        self.name=name

    def fun2(self):

        print(self.name)

    def fun1(self):

        print("yes father i am trying to do somethong ")

sos=son("mahmud")

sos.fun1()

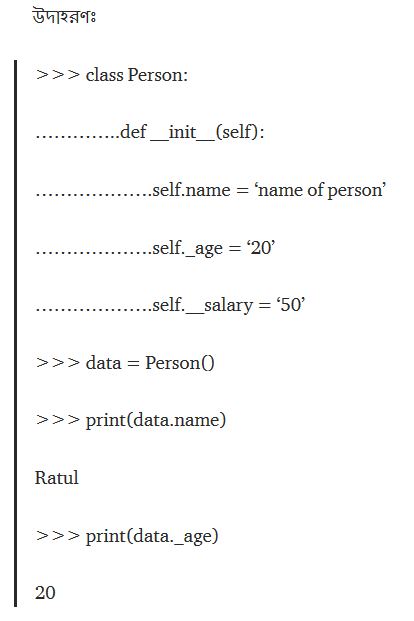
### Encapsulation concept

এনক্যাপসুলেশন অর্থ গোপনীয়তা। অন্য কোন অব্জেক্ট থেকে কোন ক্লাসের ডেটা লুকোনো বা সরাসরি ব্যবহার থেকে বিরত রাখা ই এনক্যাপসুলেশন। অর্থাৎ কোন ক্লাসের ডেটা বা মেথড কে প্রাইভেট করে রাখার প্রক্রিয়া-ই এনক্যাপসুলেশন । প্রাইভেট অ্যাট্রিবিউট ক্লাস নিজে যেকোন সময় এক্সেস করতে পারে তবে অন্য কোন ক্লাস সরাসরি এক্সেস করতে পারেনা বা পরিবর্তন করতে পারে না। এনক্যাপসুলেশন ক্লাসের ডেটার নিরাপত্তা বাড়ায়।

পাইথন ও অবজেক্ট ওরিয়েন্টেড প্রোগ্রামিং এর এনক্যাপসুলেশন নীতি ব্যবহার করে। যদিও পাইথনে ডেটা প্রাইভেট করার জন্য কোন প্রাইভেট কি-ওয়ার্ড নেই কিন্তু সিন্ট্যাক্স আছে। যদিও একে কেউ-ই সিন্ট্যাক্স বলা যায় না। সহজ করে বলার জন্য-ই বলা…

· একটি আইডেন্টিফায়ার যদি একটি আন্ডারস্কোর দিয়ে শুরু হয় তার মানে হচ্ছে আইডেন্টিফায়ার টি প্রাইভেট তবে চাইলে সহজেই এক্সেস করা যাবে।

· কোন আইডেন্টিফায়ার যদি দুটি আন্ডারস্কোর দিয়ে শুরু হয় তাহলে আইডেন্টিফায়ার টি প্রাইভেট তবে এক্সেস করা একটু কঠিন।



আমরা একটা Person ক্লাস তৈরি করেছি। এবং \_\_init\_\_() মেথডের তিনটি ভ্যারিয়েবলের ভেতরে name একটি পাব্লিক ভ্যারিয়েবল \_age প্রাইভেট, \_\_salary আরও বেশি প্রাইভেট ডেটা।

>>> print(data.\_\_salary)

Traceback (most recent call last):

File “<pyshell#16>”, line 1, in <module>

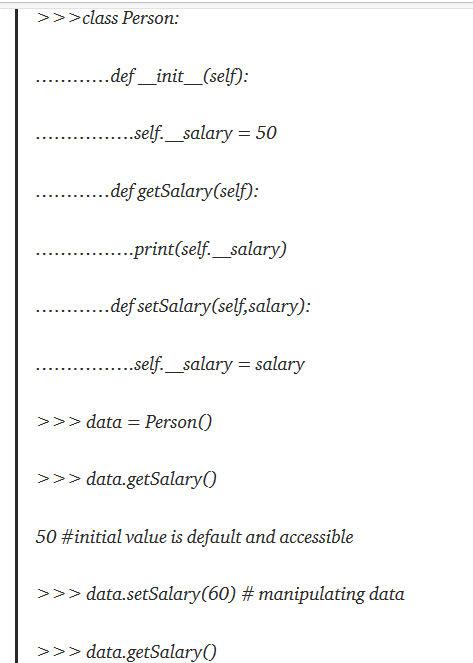
print(data.\_\_salary)

AttributeError: ‘Person’ object has no attribute ‘\_\_salary’

\_\_salary এতই প্রাইভেট যে পাইথন আমাদের বলছে যে এ নামে কোন অ্যাট্রিবিউটই নেই। কিন্তু \_age তো ঠিকই আউটপুট দেখাচ্ছে। একে তো প্রাইভেট মনে হচ্ছে না। এটি যে প্রাইভেট তা অনুভব করতে পারবে যখন ইন্ট্রারপ্রেটারে \_age এক্সেস করতে যাবে।

পাইথন ইন্টারপ্রেটার আমাদের শুধু মাত্র data অবজেক্টের একটি অ্যাট্রিবিউট সাজেস্ট করছে। যেন তার আর কোন অ্যাট্রিবিউট ই নেই।

এই প্রাইভেট অ্যাট্রিবিউট গুলো পরোক্ষ ভাবে এক্সেস বা পরিবর্তন করতে পাইথন getter এবং setter করে।

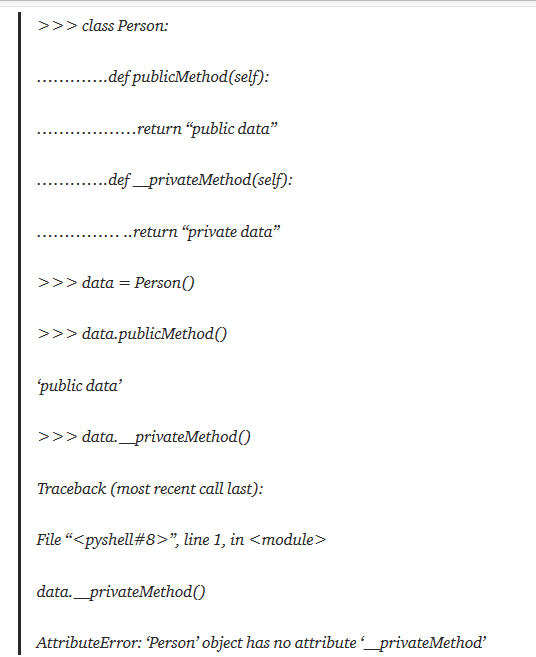


আমরা প্রথমে Person ক্লাসের দুটো মেথড ডিক্লেয়ার করেছি। getSalary() মেথড টি আমাদের \_\_salary ডেটা পাঠাবে এবং setSalary() মেথডটি তে আমরা \_\_salary ভ্যারিয়েবলে নতুন করে পাঠানো salary আর্গুমেন্টের ডেটা রেখে দিচ্ছি যা \_\_salary ভ্যারিয়েবলের ডিফল্ট ডেটা পরিবর্তন করছে।

এতক্ষণ আমরা প্রাইভেট ভ্যারিয়েবল কিভাবে ডিক্লেয়ার করে এক্সেস করে তা দেখলাম। এবার আমরা প্রাইভেট মেথডস নিয়ে আলোচনা করব। পাইথনে প্রাইভেট মেথড ডিক্লেয়ার করার জন্য নামের শুরুতে দুটো আন্ডারস্কোর দিতে হয়।

\_\_privateMethod():

pass



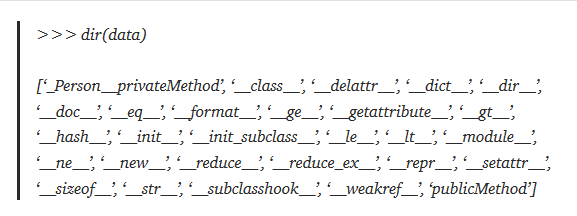
প্রাইভেট মেথড একাধিক ভাবে এক্সেস করা যায়। সাধারণত এভাবে কল করা যায়…

object.\_className.\_\_privateMethodName()

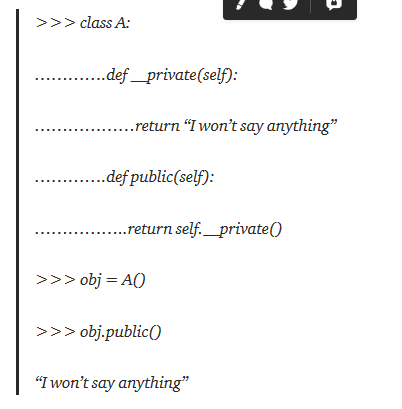
>>> data.\_Person\_\_privateMethod()

‘private data’

এ মেথড পাইথন নিজেই তৈরি করে নেয়। ইন্টারপ্রেটারে ক্লাসের ডিরেক্টরি দেখলে প্রথমেই মেথড দেখবে।



এছাড়া ও ক্লাসের ভেতরে পাব্লিক মেথড লিখে সেখানে প্রাইভেট মেথড কে কল করে পরোক্ষ ভাবে প্রাইভেট মেথড এক্সেস করা যায়।



পাইথনে আন্ডারস্কোর ব্যবহার করা ছাড়া accessify লাইব্রেরির @private, @protected ডেকোরেটরস ব্যবহার করে মেথড প্রাইভেট বা প্রোটেক্টেড করা যায়। এদের এক্সেস মডিফাইয়ারস বলে।

### Polymorphism concept

পলি অর্থ একাধিক এবং মরফিজম অর্থ কাঠামো। কোন কিছু কে একাধিক কাঠামো দেয়া কে পলিমরফিজম বলে। প্রোগ্রামিং এ পলিমরফিজম বলতে বোঝা য় একই ফাংশন কে ভিন্ন ভাবে ব্যবহার করা বা ভিন্ন আচরণ করা।

মনে কর + অপারেটর বা এর ফাংশন \_\_add\_\_() ইন্টিজার যোগ করার সময় এক রকম কাজ করে আবার স্ট্রিং যোগ করার সময় অন্য রকম.

পাইথনে দু’ভাবে পলিমরফিজম পদ্ধতি ব্যবহার করা যায়। মেথড ওভাররাইডিং ও মেথড ওভারলোডিং।

#### মেথড ওভাররাইডিং

কোন মেথড যদি প্যারেন্ট ক্লাসে ইম্পলিমেন্টেশন করা থাকে। তারপরও চাইল্ড ক্লাসে নিজের মত করে সে মেথড কে পরিবর্তন করে নেয়াকে মেথড ওভার রাইডিং বলে।

class one:

    def fun(self):

        print("hello")

class two(one):

    def fun(self):

        print("lamyaa")

sos=two()

sos.fun()

আমরা প্যারেন্ট ক্লাস one এর fun() মেথড কে ইমপ্লিমেন্ট করেছি। কিন্তু চাইল্ড ক্লাসে two মধ্যে আবার সে মেথড করে ওই ক্লাসের উপযোগী করে পরিবর্তন করে নিয়েছি এই প্রক্রিয়া কেই মেথড ওভাররাইডিং বলে।

#### মেথড ওভারলোডিং

একটি মেথড যখন ভিন্ন ভিন্ন কন্ডিশনে ভিন্ন ভিন্ন ভ্যালু রিটার্ন করে তখন তাকে মেথড ওভারলোডিং বলে।

Same method name but different output/result

class dog:

    def sound(self):

        print("ddddd")

class cat:

    def sound(self):

        print("cccccc")

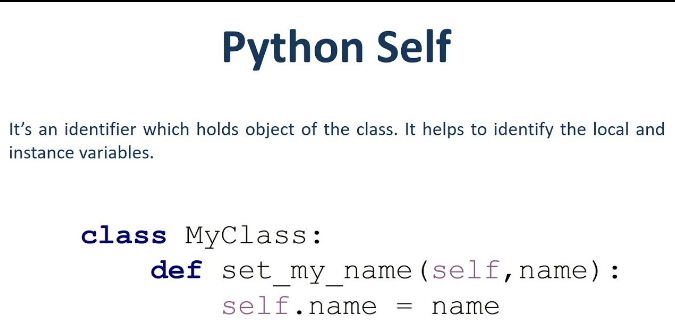
obj1=cat()

obj2=dog()

obj1.sound()

obj2.sound()

### Self-concept



পাইথন ক্লাসের কোন ফাংশন যখন কোন অবজেক্ট দ্বারা কল করা হয় তখন সেটি মেথড হয়ে যায়। এখানে ফাংশন এবং মেথডরে তফাৎ টা হচ্ছে মেথড ফাংশনের চাইতে একটি আর্গুমেন্ট বেশি নেয় এবং সেটি হচ্ছে সেই অবজেক্ট।

### Constractor concept

class one:

    name=''

    roll=''

    def \_\_init\_\_(self,name,roll):

        self.name=name

        self.roll=roll

        print("name:",name,"roll:",roll)

sos=one("mahmud",12345)

class one:

    name=''

    roll=''

    def \_\_init\_\_(self,name,roll):

        self.name=name

        self.roll=roll

    def fun(self):

        print("name:",self.name,"roll:",self.roll)

sos=one("mahmud",12345)

sos.fun()

### Magic method concept

পাইথনে কিছু বিশেষ ধরনের বিল্ট ইন মেথড আছে যেগুলোকে ম্যাজিক মেথড বলা হয়। এগুলার চেনার খুব সহজ উপায় হচ্ছে এদের নামের দুই পাসেই দুটো করে আন্ডারস্কোর সিম্বল থাকে। অর্থাৎ, \_\_init\_\_ মেথডের মত। এই মেথডের সঙ্গে ইতোমধ্যে আমাদের পরিচয় হয়েছে। কোন ক্লাসে এই ম্যাজিক মেথড ব্যবহার করলে এবং পরবর্তীতে সেই ক্লাসের ইন্সট্যান্স তৈরির সময় এই ম্যাজিক মেথডটি স্বয়ংক্রিয় ভাবেই কল হয় যাতে করে এর মাধ্যমে কিছু সেটআপ রিলেটেড কাজ করে নেয়া যায়।

এই মেথড গুলোকে ভাষায় প্রকাশ করার সময় একটু জটিলতা হয়। যেমন, "আন্ডারস্কোর আন্ডারস্কোর ইনিট আন্ডারস্কোর আন্ডারস্কোর" এভাবে বললে অদ্ভুত শোনায়। তাই এদেরকে সুন্দর ভাবে "dunders" তথা "ডাণ্ডার ইনিট" এভাবে বলা হয়ে থাকে।

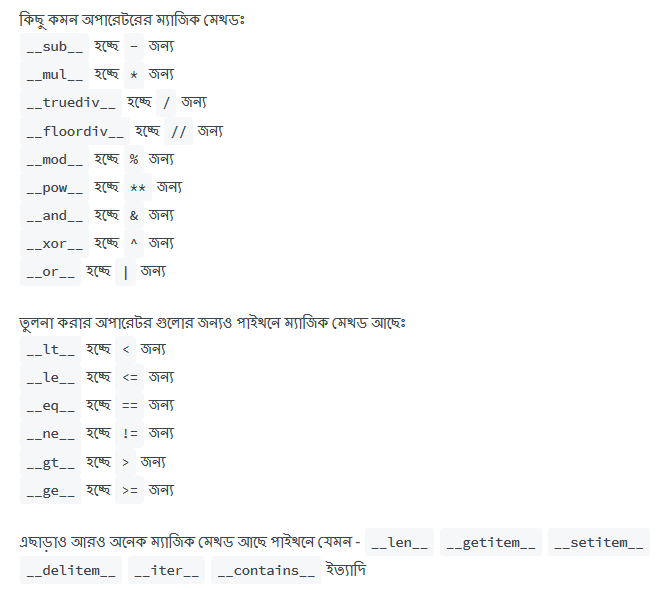
তো, এই \_\_init\_\_ মেথড বাদেও অনেক গুলো ম্যাজিক মেথড আছে পাইথনে। ম্যাজিক মেথডের খুব বহুল ব্যবহার দেখা যায় অপারেটর অভারলোডিং এর সময় যা পরের চ্যাপ্টারেই আলোচনা করা হয়েছে। প্রত্যেকটি অপারেটর এর জন্যই একটি ম্যাজিক মেথড আছে। যেমন, + অপারেটর এর জন্য ম্যাজিক মেথডটি হচ্ছে \_\_add\_\_

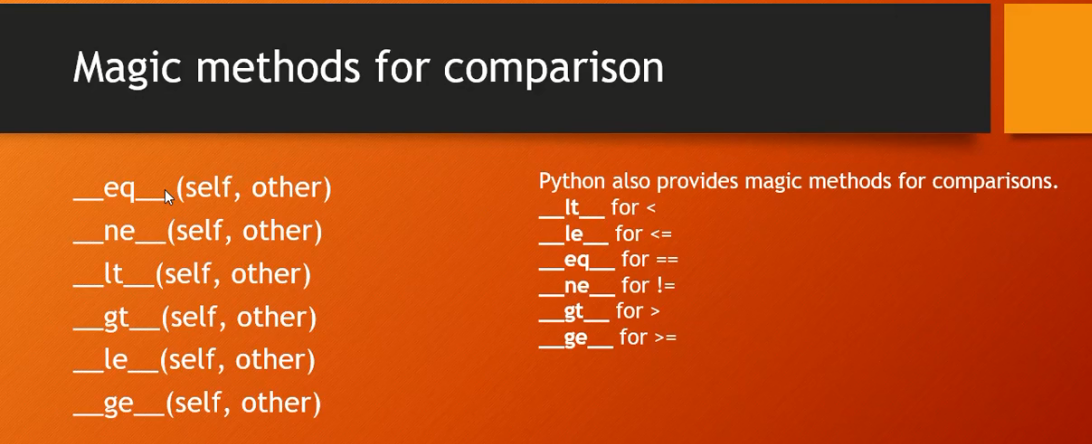
ঘটনাটি এভাবে ঘটে, যদি আমাদের এমন একটি এক্সপ্রেশন থাকে x+y এবং x বস্তুত K ক্লাসের ইন্সট্যান্স হয়। তখন পাইথন K ক্লাসের ডেফিনেশন চেক করবে। যদি K ক্লাসের একটি মেথড থাকে \_\_add\_\_ তাহলে সেটাকে কল করা হবে এভাবেঃ x.\_\_add\_\_(y)

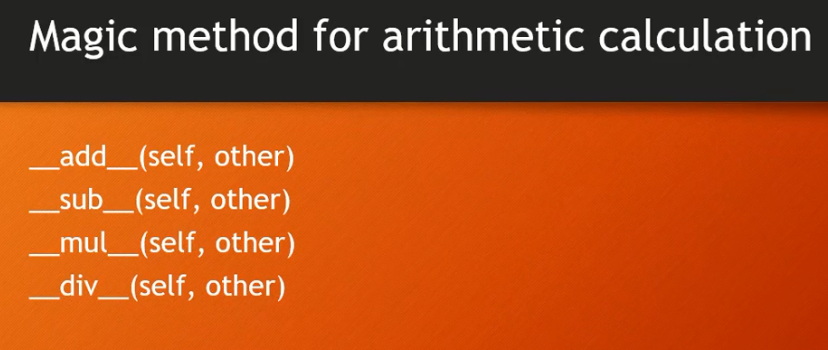
কিছু কমন অপারেটরের ম্যাজিক মেথডঃ \_\_sub\_\_ হচ্ছে - জন্য \_\_mul\_\_ হচ্ছে \* জন্য \_\_truediv\_\_ হচ্ছে / জন্য \_\_floordiv\_\_ হচ্ছে // জন্য \_\_mod\_\_ হচ্ছে % জন্য \_\_pow\_\_ হচ্ছে \*\* জন্য \_\_and\_\_ হচ্ছে & জন্য \_\_xor\_\_ হচ্ছে ^ জন্য \_\_or\_\_ হচ্ছে | জন্য

তুলনা করার অপারেটর গুলোর জন্যও পাইথনে ম্যাজিক মেথড আছেঃ \_\_lt\_\_ হচ্ছে < জন্য \_\_le\_\_ হচ্ছে <= জন্য \_\_eq\_\_ হচ্ছে == জন্য \_\_ne\_\_ হচ্ছে != জন্য \_\_gt\_\_ হচ্ছে > জন্য \_\_ge\_\_ হচ্ছে >= জন্য

এছাড়াও আরও অনেক ম্যাজিক মেথড আছে পাইথনে যেমন - \_\_len\_\_ \_\_getitem\_\_ \_\_setitem\_\_ \_\_delitem\_\_ \_\_iter\_\_ \_\_contains\_\_ ইত্যাদি







### Access modifier concept

#### Public:

All members in a Python class are **public** by default. Any member can be accessed from outside the class environment.

class one:

    def \_\_init\_\_(self,name,city):

        self.name=name

        self.city=city

        print("name:",self.name,"city:",self.city)

sos=one("mahmud","rajshahi")

tos=one("lamyaa","pabna")

You can access one class's attributes and also modify their values, as shown below.

#### Protected

Python's convention to make an instance variable **protected** is to add a prefix \_ (single underscore) to it. This effectively prevents it to be accessed, unless it is from within a sub-class.

## General

### Character class concept

import re

pat =r"[A-Z][1-9][a-z]"

if re.match(pat,"W1aA"):

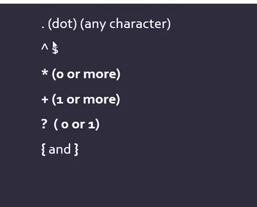
    print("match")

else:

    print("no match")

### Regular expression and Meta character

Meta character list:



### Import module concept

import datetime

time=datetime.datetime.now()

print("hour:",time.hour)

print("miniture:",time.minute)

print("second:",time.second)

print("microsecond",time.microsecond)

print("milisecond:",time.month)

print("milisecond:",time.year)

print("milisecond:",time.day)

print("presend year,month,day:",datetime.date.today())

#we can also using this formate, only for show date not time

from datetime import date

print("year,month,day:",date.today())

### Dictionary concept

lol={

    101:"mahmhud",

    "105":"lamyaa",

    "cool":"red"

}

print(lol[101])

print(lol["105"])

print(lol["cool"])

if call unknown value, above format show error, but using get ,we found good

lol={

    101:"mahmhud",

    "105":"lamyaa",

    "cool":"red"

}

print(lol.get(101))

print(lol.get("105"))

print(lol.get("cool"))

print(lol.get("co"))

we can also print a default value , if not found value

lol={

    101:"mahmhud",

    "105":"lamyaa",

    "cool":"red"

}

print(lol.get(101))

print(lol.get("105"))

print(lol.get("cool"))

print(lol.get("co", "not valid"))

### Keyboard input concept

def fun(x,y):

    sum=x+y

    print("result:",sum)

sos=int(input("enter x:"))

lol=int(input("enter y:"))

fun(sos,lol)

### List concept

lol=["ab","bc","ca"]

print(lol) #show a list items

print(lol[2:]) #start show item

print(lol[-2:]) #reversely show item

print("bcb" in lol) #use of in funtion

print("bcb" not in lol) #use of not in funtion

print(lol\*2) #multiply a list

lol.append("hello") #append funtion

print(lol)

lol.insert(3,"lamyaa") #insert funtion

print(lol)

lol.remove("ab") #remove funtion

print(lol)

print(len(lol)) #show lenght

lol.sort() #sorting funtion

print(lol)

lol.reverse() #reverse funtion

print(lol)

lol.pop() #pop funtion

print(lol)

lol.clear() #clear funtion

print(lol)

lol=["a","b","c"] #copy funtion

lol2=lol.copy()

print(lol2)

lol=[1,1,1,1,2,3,4,5,6]

sos=lol.index(2) #show position

print(sos)

sos=lol.count(1) #count funtion use to repert element count

print(sos)

### File handling concept



প্রোগ্রামিং করার সময় আমাদের নানা রকমের এররের সম্মুখীন হতে হয়।এদের প্রোগ্রামিং এর ভাষায় এক্সেপশন বলে। পাইথনে এক্সেপশন হ্যান্ডেল করার জন্য try-except নামের একটি স্টেট্মেন্ট আছে। হ্যান্ডেল মানে এই না যে এক্সেপশন দেখাবে না। সাধারণত পাইথন আমাদের প্রকৃত এররের সাথে আরও অনেক ডেটা প্রিন্ট করে। পাশাপাশি অনেক সময় প্রকৃত এরর টাকে ট্রেস করা যায় না। এক্সেপশন হ্যান্ডেলিং প্রোগ্রামের প্রকৃত এরর নির্ণয় করা হয় এবং প্রোগ্রামকে ক্র্যাশ করা থেকে রক্ষা করে।

**try-except**

try ব্লকে মূল কোড লিখা হয় এবং সে কোড থেকে যে এরর আশার সম্ভাবনা আছে সেগুলো লিখা হয় except ব্লকে।

যদি try-except স্টেট্মেন্ট ব্যবহার করা না হতো তাহলে প্রথম ইটারেশনে একবার এরর দেখিয়ে প্রোগ্রাম ক্র্যাশ করত। আমরা বুঝতে পারতাম না কোন ডেটার জন্য কি এরর দেখাচ্ছে। এভাবে আমরা প্রতিটি এরর আলাদাভাবে হ্যান্ডেল করতে পারি।

**Finally**

ফাইনালি স্টেটমেন্ট try-except এর পরে ব্যবহার করা হয়। try বা except যেকোন স্টেটমেন্ট এক্সিকিউট হওয়ার পর অবশ্যই finally স্টেটমেন্ট এক্সিকিউট হবে।

**try-except-else**

try-except এর সাথে আরেকটি স্টেটমেন্ট ব্যবহার করা হয় “else”। যখন কোডে কোন এরর পায় না তখনই কেবল else স্টেটমেন্ট এক্সিকিউট হয়৷

example:

def fun(x,y):

    try:

        result=x/y

        print(result)

    except:

        print("string not divided by an integer number")

    finally:

        print("lets try to another digit again")

fun("d",3)