```
from google.colab import drive
drive.mount('/content/drive')
```

```
    Drive already mounted at /content/drive; to attempt to forcibly remount, call drive.mount("/conte
```

```
from sympy import *
from polyclass import *
from libaldo_math import *
from libaldo_show import *
from physic_lib import *
from IPython.display import display, Math
init_printing()
```

```
x,y=symbols('x y')
P=mparticle()
P.add_forza(8*y,0)
P.add_forza(6*x+y,pi)
```

```
P.x_res()
```

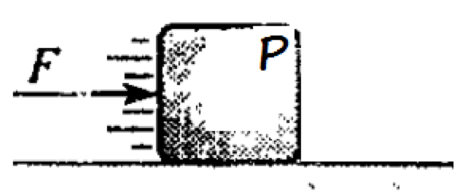
$$-6x+7y$$

Creating our Physics Object

Whit mparticle() class we can handle kinemtic, static, dynamic, work, energy,C.Gravity, etc

P = mparticle() .. class
We have : mass=m, g=grav
mu = CofFric
We add N1= normal fr = Rozam.

Find max value of F with m,g,mu




```
# Steep 1 : Creating sympy variables
m,g,mu,N1,F1,fr=symbols('m g mu N1 F1 fr')
```

```
# Steep 2 : Creating physic object P
P=mparticle()
```

```
# Steep 3 :
```

```
# function P.add_forza(value,angle respect x axis, pos x , pos y) default  pos x and y = 0
# adding weight
P.add_forza(m*g,-pi/2)
# adding F1
P.add_forza(F1,0)
# adding N1
P.add_forza(N1,pi/2)
# adding fr
P.add_forza(fr,pi)


# get resultamt in x and y :
P.x_res()
```

$$F_1 - fr$$

```
P.y_res()
```

$$N_1 - gm$$

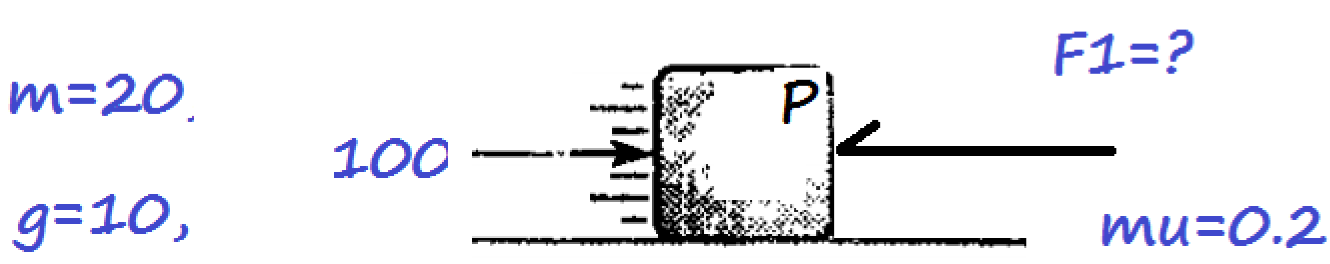
```
#  function P.store_val(symbol name,value with respect to other value)
# In this cas we know that N1=m*g and fr= mu*N1 then....
P.store_val(N1,m*g)
P.store_val(fr,N1*mu)


# see now what happend with x_res and y_res
P.x_res()
```

$$F_1 - gm\mu$$

```
P.y_res()
```

$$0$$

Find F1 value if P no move and in equilibrium...




```
# decrared sympy variables neccesary
N1,F1,fr =symbols('N1 F1 fr') # why N1 and not N .. because N is important notation in python
```

```
# Creating physic object P
P=mparticle()
```

```
# adding forzas
P.add_forza(100,0)
P.add_forza(F1,pi)
P.add_forza(N1,pi/2)
P.add_forza(10*20,-pi/2)
P.add_forza(fr,pi)
```

```
#  function P.store_val(symbol name,value with respect to other value)
# In this cas we know that N1=m*g and fr= mu*N1 then....
P.store_val(N1,20*10)
P.store_val(fr,N1*0.2)
```

```
# see now what happend with x_res and y_res
P.x_res()
```

$60.0 - F_1$

```
P.y_res()
```

0

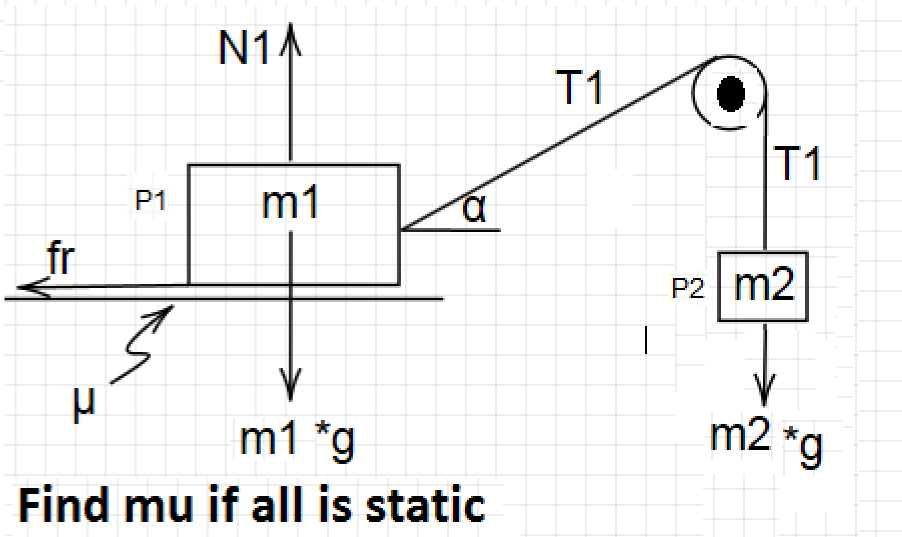
```
# New mat function  csolve( Equation that is = 0, variable to find, optional nice answer)
# we know that x resultant =0 then...
F1=csolve(P.x_res(),F1,'F1')
```

$$F1 = 60.0$$

```
F1
```

60.0

Working whit two Physic object



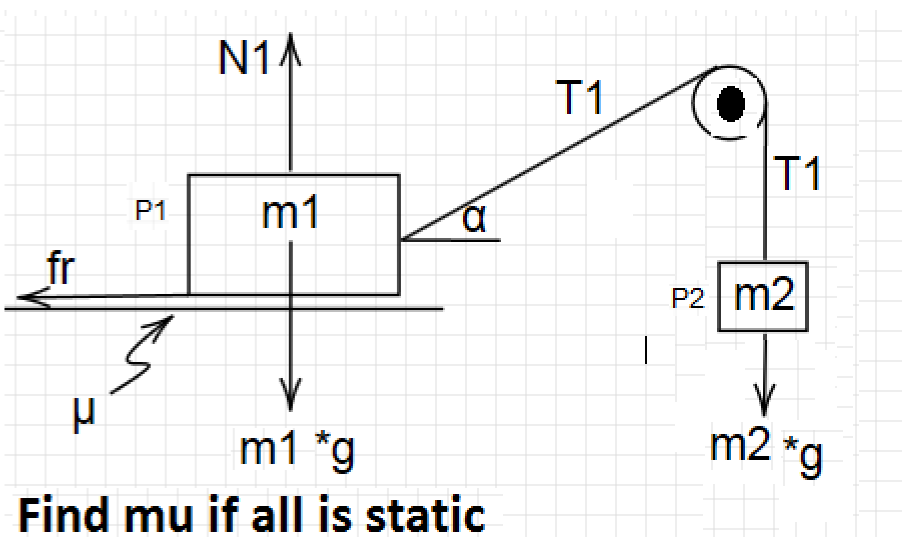

```
# decrared sympy variables neccesary
m1,m2,g,N1,fr,alpha,T1,mu =symbols('m1 m2 g N1 fr alpha T1 mu ')
```

```
# Creating physic object P2 and get T1 in m2,g function
P2=mparticle()
```

```
# is easy to know that T1 = m2*g but only for explained situation we will add forza and find
P2.add_forza(m2*g,-pi/2)
P2.add_forza(T1,pi/2)
```

```
# csolve T1  from P2.y_res=0 ans store in T1
T1=csolve(P2.y_res(),T1,'T1')
```

$$T1 = gm_2$$




```
# Creating physic object P2 and get T1 in m2,g function
P1=mparticle()
```

```
# is easy to know that T1 = m2*g but only for explained situation we will add forza and find
```

```
# is easy to know that T1 = m2*g but only for explained situation we will add forza and find
P1.add_forza(m1*g,-pi/2)
P1.add_forza(N1,pi/2)
P1.add_forza(T1,alpha)
P1.add_forza(fr,pi)


# adding value that we know like...
P1.store_val(N1,m1*g)
P1.store_val(fr,N1*mu)


# and now res...??
P1.x_res()
```

$$-gm_1\mu + gm_2\cos(\alpha)$$

```
P1.y_res()
```

$$gm_2\sin(\alpha)$$

```
# we will use x_res to find mu
mu=csolve(P1.x_res(),mu,'mu')
```

$$mu = \frac{m_2\cos(\alpha)}{m_1}$$

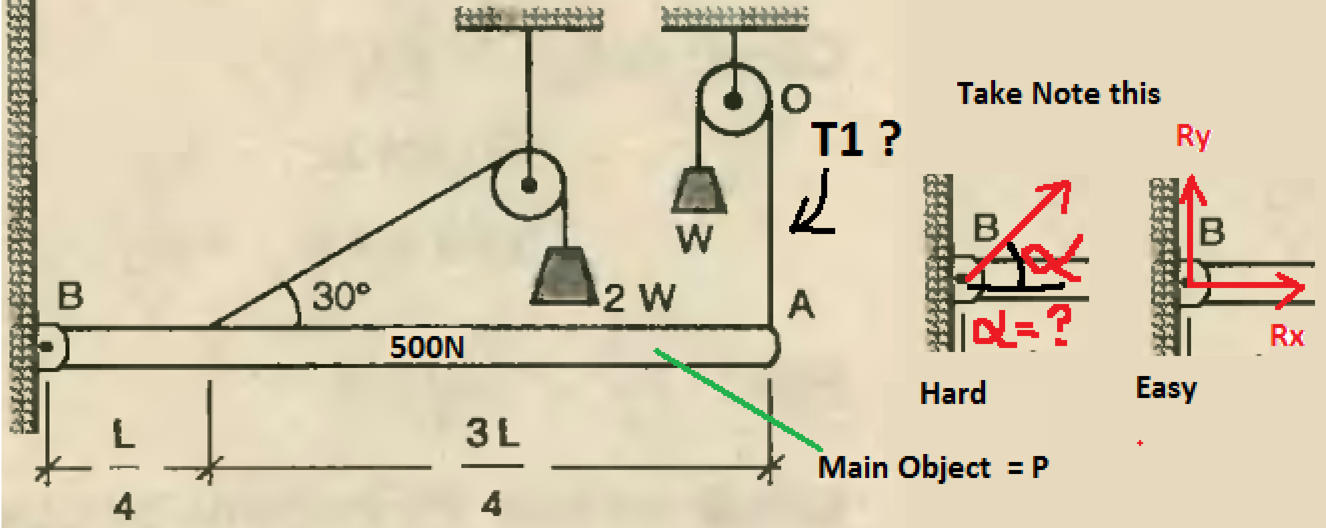
```
# new nice show function because i headddd Latex .. es una mierda el Latex
sE(['mu value is=',mu])  #sE= show expression
```

$$mu\ value\ is = \frac{m_2\cos(\alpha)}{m_1}$$

```
sE(['one=',1,',two=',2,'and','sum','one+two=',1+2 ])
```

$$one = \ 1\ , two = \ 2\ and\ sum\ one + two = \ 3$$

Working whit torque and adding forze whit x,y position neccesary for torque calculus




```
# decrared sympy variables neccesary
g,W,L,Rx,Ry =symbols('g W L Rx Ry ') # is better Rx+Ry instead R because ww not know  alpha
```

```
# Creating physic object P
P=mparticle()
```

```
# adding forza.. this time we will includedd x,y cordinate posss  B is (0,0)
P.add_forza(Rx,0,0,0) # Rx x comp in B
P.add_forza(Ry,pi/2,0,0) # Ry y comp in B
P.add_forza(2*W,pi/6,L/4,0) # Tension due 2W
P.add_forza(500,-pi/2,L/2,0) # weigh bar in midle pos
P.add_forza(W,pi/2,L,0) # T1=W in extremm bar poss
```

```
# Now due object ar in equilibrium ..the x_res=0, y_res=0, and Torque in wathever point=0
P.x_res() # first Eq
```

$$Rx + \sqrt{3}W$$

```
P.y_res()
```

$$Ry + 2W - 500$$

```
# Torque in B
P.torque()
```

$$\frac{5LW}{4} - 250L$$

```
# Torque in midle
P.torque(L/2,0) # to change torque applic.. pos  only included new x,y pos
```

$$-\frac{LRy}{2} + \frac{LW}{4}$$

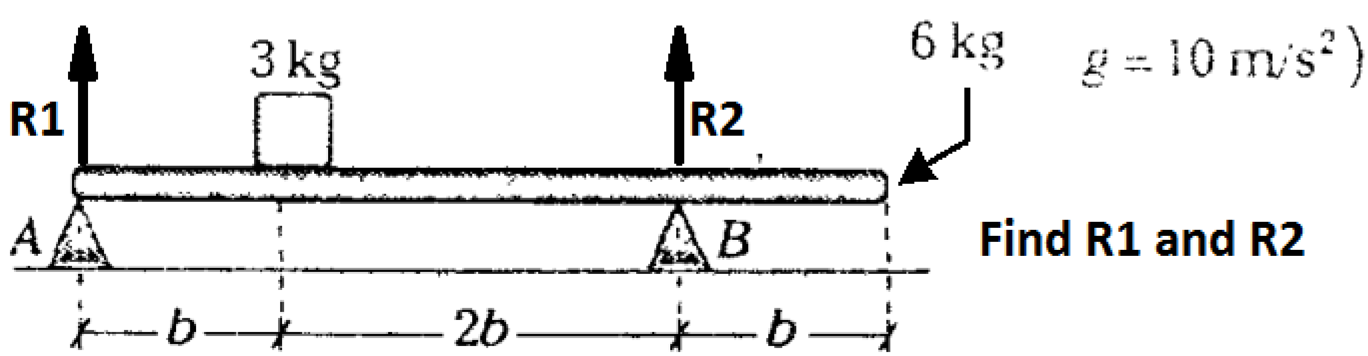
```
# Torque in extreme of bar
P.torque(L,0) # to change torque applic.. pos  only included new x,y pos
```

$$-LRy - \frac{3LW}{4} + 250L$$

```
# the only want to find W and not involved Rx and Ry is Torque in B because are caceleld .
# and now we will used csolve with torque=0 to find W or T1 that is the same...
W=csolve(P.torque(),W,'T1') # and voalla..
```

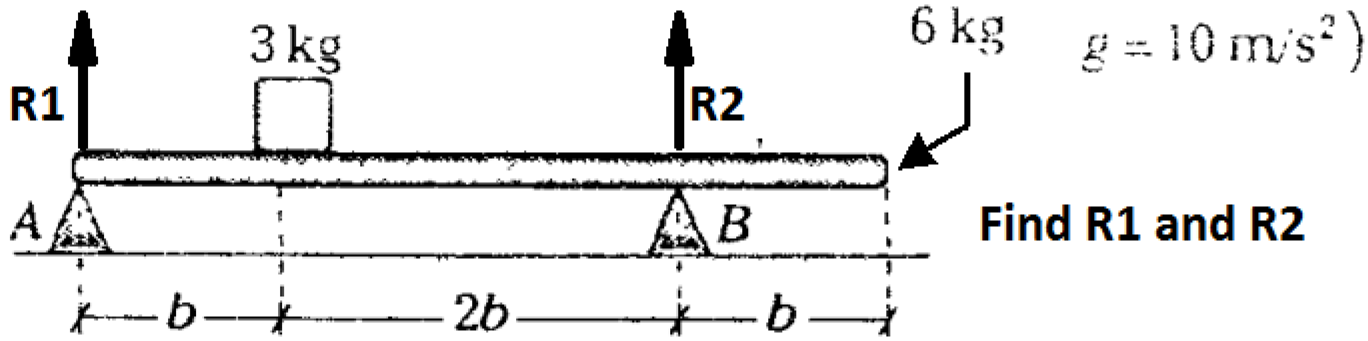
$$T1 = 200$$

More torque super easy sample... more advance come real nerd problems solve




```
# decrared sympy variables neccesary
b,R1,R2 =symbols('b R1 R2') # is better Rx+Ry instead R because ww not know  alpha
```

```
# Creating physic object P
P=mparticle()
```




```
# adding forza.. this time we will includedd x,y cordinate posss  B is (0,0)
P.add_forza(R1,pi/2,0,0) # R1
P.add_forza(3*10,-pi/2,b,0) # 3Kg weight
P.add_forza(6*10,-pi/2,2*b,0) # bar weight
P.add_forza(R2,pi/2,3*b,0) # weigh bar in midle pos
```

```
# find R2 using Torque = 0 in A
R2=csolve(P.torque(),R2,'R2')
```

$$R2 = 50$$

```
# find R1 using Torque = 0 in B
R1=csolve(P.torque(3*b,0),R1,'R1')
```

$$R1 = 40$$

---

working whit acceleration

```
# before we will try to understand mparticle()  class.....
```

internal variables that we can used into itself algoritm , always we need declared all variables because sympy library take two variables with same names but are different for itself.

```
class mparticle()
    self.x1=x1 # posicion x initial
    self.y1=y1 # posicion y initial
    self.x2=x2 # posicion x final
    self.y2=y2 # posicion y final
    self.v1=v1 # initial velocity
    self.v2=v2 # final velocity
    self.m=m   # masa
    self.v=v   # velocity,
    self.t=t  # time
```

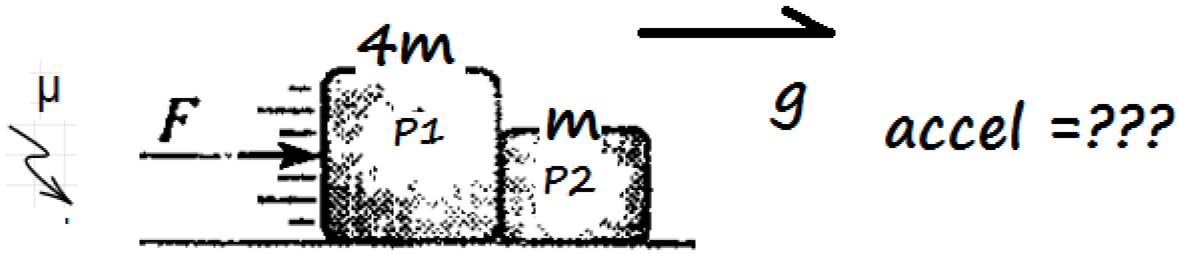

```
# first declaring variable
F1,m,g=symbols('F1 m g',positive=True)
# In advance we will include positive True in order to solve root equations
# also is better use F1 because F is used in several libraries...


P=mparticle(m=m) # now we declaring internal mass for used in other algoritm


P.add_forza(F1,0)
# see ress x
P.x_res()
```

$$F_1$$

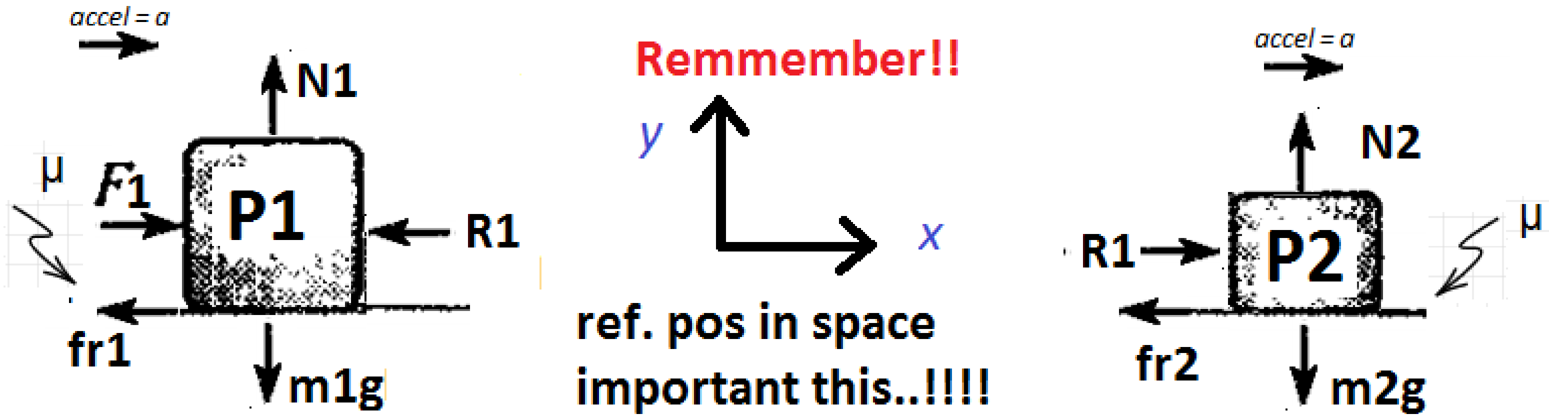
```
# Finding acceleration
P.simple_ac()
```

$$\frac{F_1}{m}$$

Finding default acceleration into Physic mparticle object class()...



first declaring al variable including Normals and F reaction between mass




```
from sympy import *
from polyclass import *
from libaldo_math import *
from libaldo_show import *
from physic_lib import *
from IPython.display import display, Math
init_printing()


F1,fr1,N1,M,g,R,N2,fr2,mu=symbols('F1 fr1 N1 M g R N2 fr2 mu',positive=True)


# creatin P1 with  mass = 4*M
P1=mparticle(m=4*M)
# adding forzas..
P1.add_forza(F1,0)
P1.add_forza(4*M*g,-pi/2)
P1.add_forza(R,pi)
P1.add_forza(N1,pi/2)
P1.add_forza(fr1,pi)


# change uknow Forces in know forces like
P1.store_val(N1,4*M*g) # Normal in P1 is weight
P1.store_val(fr1,N1*mu) # roz for = Normal * mu


# now get acceleration due x_res()
P1.simple_ac('x')
```

$$\frac{F_1 - 4Mg\mu - R}{4M}$$

```
# creatin P1 with  mass = 4*M
P2=mparticle(m=M)
# adding forzas..

P2.add_forza(M*g,-pi/2)
P2.add_forza(R,0)
P2.add_forza(N2,pi/2)
P2.add_forza(fr2,pi)


# change uknow Forces in know forces like
P2.store_val(N2,M*g) # Normal in P1 is weight
P2.store_val(fr2,N2*mu) # roz for = Normal * mu


# now get acceleration due x_res()
P2.simple_ac('x')
```

$$\frac{-Mg\mu + R}{M}$$

```
# with sympy.....???
solve(P2.simple_ac('x')-P1.simple_ac('x'),R)
```

$$[]$$

```
# with libaldo.....???
csolve(P2.simple_ac('x')-P1.simple_ac('x'),R)
```

$$[]$$

```
# but.....???
csolve(P2.simple_ac('x')-P1.simple_ac('x'),R,unifique=True)
```

$$\frac{F_1}{5}$$

```
P2.simple_ac('x')
```

$$\frac{-Mg\mu + R}{M}$$

```
# in object P2 we change all R by F1/5
P2.store_val(R,F1/5)


# after change call simple_ac again
P2.simple_ac('x')
```

$$\frac{\frac{F_1}{5} - Mg\mu}{M}$$

```
P1.simple_ac('x')
```

$$\frac{F_1 - 4Mg\mu - R}{4M}$$

```
# in object P1 we change all R by F1/5  and call again
P1.store_val(R,F1/5)


# after change call simple_ac again
P1.simple_ac('x')
```

$$\frac{\frac{4F_1}{5} - 4Mg\mu}{4M}$$