Aluno: Luiz Rodrigo Lacé Rodrigues
DRE: 118049873

1.
A tabela abaixo foi obtida como resultado de um experimento relativo ao valor da temperatura T (em graus Celsius) com a posição x (em centímetros):

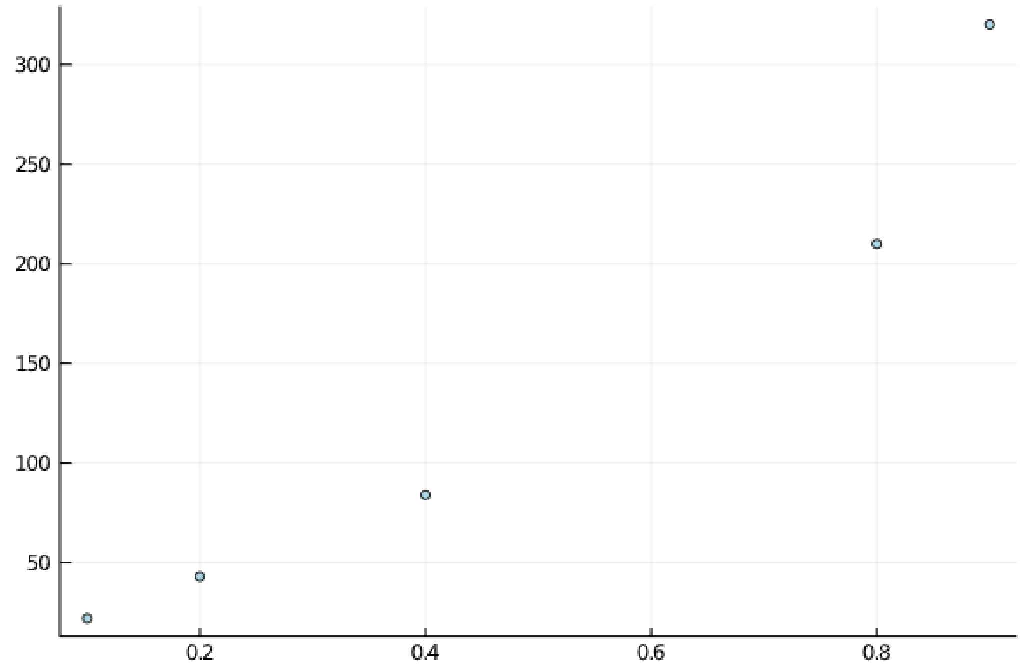
x	0.1	0.2	0.4	0.8	0.9
T(x)	22	43	84	210	320

Determine a curva da forma T(x) = c0*(x^c1) que melhor se ajusta aos dados da tabela com o método de mínimos quadrados com coeficientes não-lineares e use o modelo para calcular T(0.3) com três casas decimais.

Primeiramente no Julia vamos plotar os pontos normalmente para verificar o comportamento exponencial.

```
In [4]: x=[0.1; 0.2; 0.4; 0.8 ; 0.9]    #pontos de x
        y=[22;  43;  84;   210 ; 320 ] #pontos de y
        scatter(x, y ,c=:lightblue, ms=3, leg=false)
```

Out[4]:



Como queremos utilizar os pontos que temos para determinar a curva T(x), para que consigamos fazer a regressão e assim achar os coeficientes dessa curva, primeiramente precisamos linearizar-lá de forma com que fiquemos com:

T(x) = c0*(x^c1)

Tirando ln dos dois lados:

ln(T(x)) = ln(c0*(x^c1))

ln(T(x)) = ln(c0) + ln(c^c1)

ln(T(x) = ln(c0)+ c1 * ln(x)

Para o raciocínio da regressão Vamos nos aproveitar do método vandermonde, para criar as matrizes de vandermonde, que tem essa cara:

```
In [2]: function vandermonde(x,y,grau)
            n,=size(y) #pegando o tamanho do vetor y
            V=zeros(n, grau+1) #criando uma matriz de zeros
            for i=1:n #linhas
                for j=1:(grau+1) #colunas
                    V[i,j]=x[i]^(j-1) #calculando as resprectivas potencias do polinomio
                end
            end
            return V # retona a matriz de vandermonde
        end
```

Esse método será utilizado no método regressão:

```
In [3]: function regressão(x,y,grau)
            V=vandermonde(x,y,grau) #montando a matriz de vondermonde
            c=V\y # resolve o  sistema linear. já faz minimos quadrados
            return c #retorna os coeficientes
        end
```

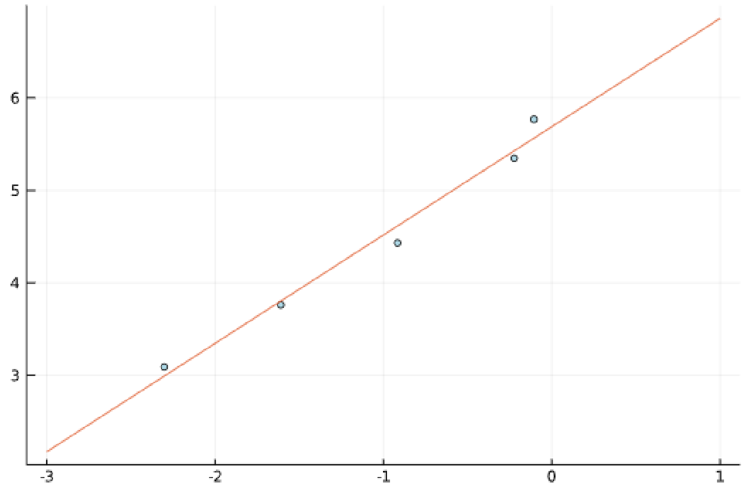
O que vamos fazer agora é calcular ln dos pontos x e y (x_barra e y_barra), fazer a regressão desses pontos para encontrar os coeficientes da reta(poli_barra). Também vamos plotar os pontos e a reta.

Teremos o seguinte resultado no julia:

In [24]:
```
#linearizando os pontos para encontrar os coeficientes
y_barra=log.(y)    #calculando o ln dos pontos de y
x_barra=log.(x)    #calculando o ln dos pontos de x

c_barra=regressão(x_barra, y_barra, 1)   #calculando os coeficientes por regressão de grau 1, pois linearizamos
poli_barra(x) = c_barra[1]+c_barra[2]*x  #Gerando a reta a partir dos coeficientes gerados anteriormente
scatter(x_barra, y_barra ,c=:lightblue, ms=3, leg=false)   #plotando os pontos linearizados
plot!(poli_barra,-3,1)   #plotando a reta
```

Out[24]:



Agora que linearizamos a curva e temos os coeficiente, podemos voltar para montar a curva T(x) = c0*(x^c1).

Entretanto o que temos é

ln(T(x) = ln(c0)+ c1 * ln(x)

O que encontramos então foi ln(c0) e c1, precisamos calcular o exponencial de ln(c0) para encontrar o c0.

Fazendo isso basta substituir em T(x) = c0*(x^c1) que teremos a nossa curva.

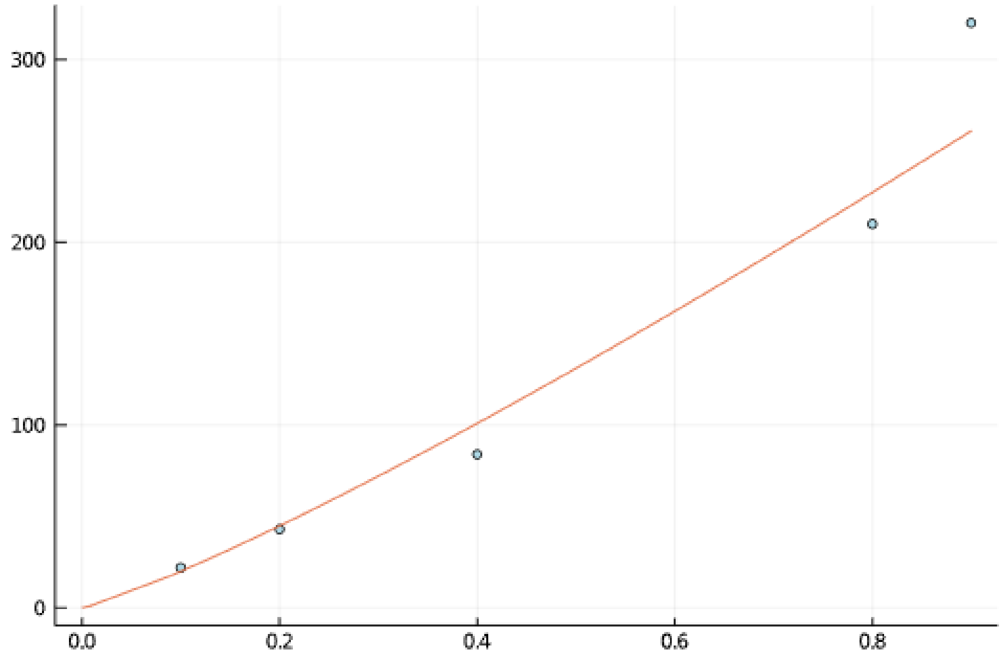
No Julia, teremos:

In [25]:
```
c_barra=regressão(x_barra, y_barra, 1)    #calculando os coeficientes por regressão de grau 1, pois linearizamos
c0 = exp(c_barra[1])                      #calculando exp(c0)
c1 = c_barra[2]                           #mantentdo o valor de c1
curva(x) = c0*(x^c1)                       #gerando a curva com os polinomios
```

Podemos agora plotar a curva com os pontos originais para verificar a aproximação:

```
In [26]: scatter(x, y ,c=:lightblue, ms=3, leg=false)
         plot!(curva,0,0.9)
```

Out[26]:



Como nosso objetivo era calcular T(0.3), basta substituir 0.3 em curva(x):

```
In [27]: curva(0.3)

Out[27]: 72.0611661989366
```

Como queremos apenas 3 casas decimais, basta adicionar digits = 3 no parâmetro:

```
In [28]: round(curva(0.3),digits=3)

Out[28]: 72.061
```

Chegamos então em **T(0.3) = 72.061**

2.
Em maio de 1992 o ônibus espacial Endeavour foi lançado na missão STS-49.
A tabela mostra o tempo e a velocidade do ˆonibus.

Tempo (s)	0	10	15	20	32	59	62	125
Velocidade (pé/s)	0	185	319	447	742	1325	1445	4151

(a)
Faça a integral numérica utilizando o método dos trapézios de t = 0 a t = 125 para estimar a altura atingida pelo ônibus 125s após o lançamento.

Nessa questão faremos uso do método trapézio que tem como parâmetros a função (f), os limites de integração(a,b) e a quantidade (n) de intervalos na integração, que tem essa cara:

```
In [17]: function trapezio(f,a,b,n) #calcular a integral f(x) de a até b
             h=(b-a)/n            #tamanho dos intervalos baseado no limite de integração e o numero de intervalos
             S=0.0                #soma da integral
             for i=1:(n-1)       #calcula o "meio"
                 x=a+i*h         #avanço de x
                 S+=2*f(x)       #incrementando a soma
             end
             S=h/2*(S+f(a)+f(b)) #calcula "as pontas"
             return S             #retorna a soma S
         end

Out[17]: trapezio (generic function with 1 method)
```

Entretanto, como não temos a função que traduz o comportamento desses pontos, vamos utilizar o método interpolação1 para gerar as retas entre cada intervalo dos pontos.

Esse método tem a seguinte cara:

```
In [30]: function interpolação1(x,y)
             #criar a matriz V
             V=[x.^0 x.^1]  #Matriz de vandermonde
             c=V\y   #vetor de coeficientes

             poli(x) = c[1]+c[2]*x  #polinomio montado a partir dos coeficientes encontrados

             return poli          #retorna o polinomio
         end

Out[30]: interpolação1 (generic function with 1 method)
```

Agora vamos juntar os dois métodos para que seja possível fazer a integração com o método do trapézio com todos os intervalos e assim achar a integral de 0 a 125 e assim podemos estimar a altura atingida pelo ônibus 125s após o lançamento.

O método em questão é o **interpola_trapezio(x,y,n)**, que recebe os pontos x e y e o número de intervalos

```
In [32]: function interpola_trapezio(x,y,n)
             S=0.0  #soma
             for i=1:n #loop com n intervalos
                 poli = interpolação1(x[i:i+1],y[i:i+1]) #gera um polinomio para cada intervalo
                 S+=trapezio(poli,x[i],x[i+1],1)          #faz o metodo do trapezio para o polinomio encontrado anteriormente
                                                          # e soma em S
             end
             return S
         end

Out[32]: interpola_trapezio (generic function with 1 method)
```

Nesse método temos a soma (S) da integral começando em 0. Depois entramos em um loop into de 1 até o número de intervalos entre os pontos, que no nosso caso é 7 (temos 8 pontos)

Para cada loop, uma reta (poli) será criada com o método da interpolação passando os pontos (x[i],y[i]) e (x[i+1],y[i+1]) em relação ao número do intervalo, dessa forma temos 2 pontos para criar a reta.

Com a reta (poli) criada, podemos passá-la no método trapézio para fazer a integração com os limites x[i] e x[i+1] em relação ao número do intervalo.

Agora vamos passar os pontos x e y e passando no parâmetro de junto com o número de intervalos n = 7, teremos:

```
In [33]: #Exercicio 2a

         x=[0; 10;   15;   20;    32;   59;     62;    125]
         y=[0; 185;  319;  447;   742;  1325;   1445;  4151]
```

```
In [34]: interpola_trapezio(x,y,7)

Out[34]: 219567.5
```

Logo a altura atingida em 125s é de **219567.5** pés.

(b) Qual informação sobre o ônibus espacial você precisa para estimar o erro máximo cometido no item anterior?

Sabendo que o erro do método do trapézio é dado por

$$\left| \int_a^b f(x) - (\text{método do trapézio}) \right| \leq \frac{h\hat{}3 \ * N * M}{12}$$

Se, somente se, | f''(x) | ≤ M , para a ≤ x ≤ b.

Logo, para determinar o erro dessa função, que é a função horária da velocidade, precisamos da sua derivada segunda, ou seja, precisaríamos saber o quanto a aceleração do ônibus espacial está variando. **E para isso teríamos que saber qual é a função horária da velocidade f(x).**

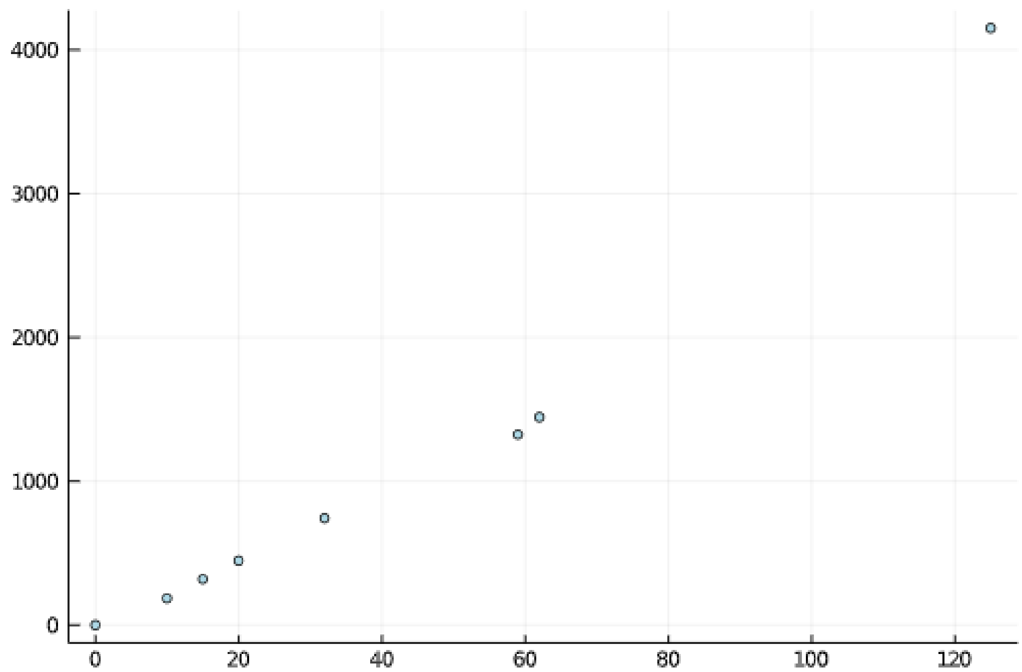
(c) Encontre a reta, p1(x) = c0*x+c1, no sentido dos mínimos quadrados, que melhor descreve a distribuição dos pontos. Estime a altura a partir dessa curva, ou seja, calcule $\int_0^{125}$ p1(x)dx.

Dessa vez vamos manter os pontos com seus valores originais.

Plotando, temos essa cara:

In [41]:
```
#Exercicio 2c
x=[0; 10;   15;   20;    32;   59;      62;    125]
y=[0; 185;  319;  447;   742;  1325;   1445;   4151]
scatter(x, y,c=:lightblue, ms=3, leg=false)
```

Out[41]:



Agora vamos fazer a regressão linear de grau 1 para encontrarmos os coeficientes que aproximem esses pontos de uma reta.

Utilizando o método regressão mostrado anteriormente, teremos:

```
In [42]: c=regressão(x,y,1)

Out[42]: 2-element Array{Float64,1}:
          -232.97096383346047
            32.438909320952575
```

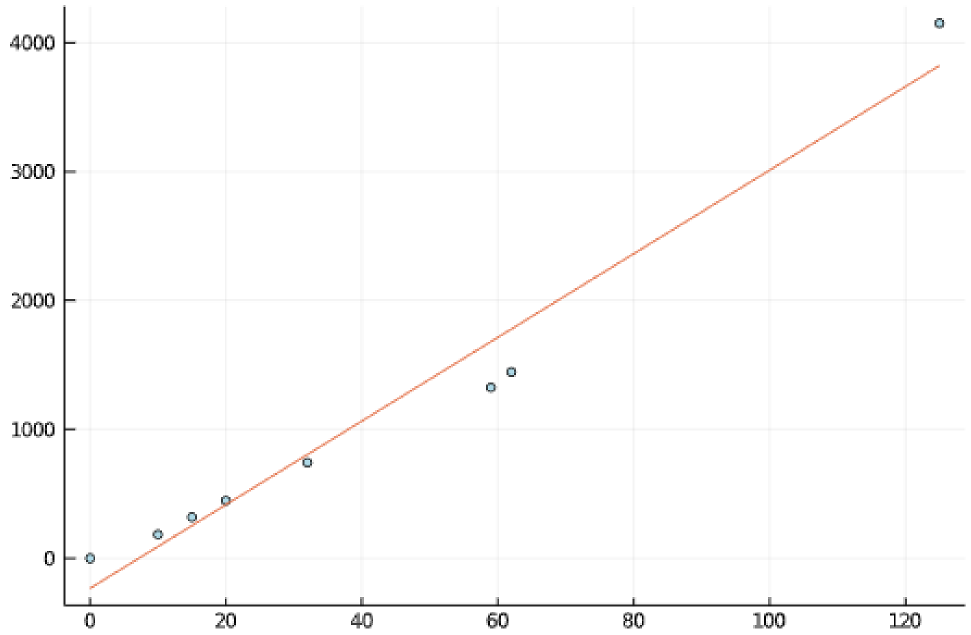
Agora com os coeficientes podemos montar a equação da reta e plotar para verificar a aproximação dos pontos:

```
In [45]: reta(x) = c[1] + c[2]*x

Out[45]: reta (generic function with 1 method)

In [46]: scatter(x, y,c=:lightblue, ms=3, leg=false)
         plot!(reta,0,125)

Out[46]:
```



Passando agora essa reta no método trapézio com o intervalo de integração de 0 a 125, com 7 intervalos, teremos:

```
In [40]: trapezio(reta,0,125,7)

Out[40]: 224307.6085907594
```

Logo, a altura estimada com essa estratégia é de **224307.608… pés**

3. A área do círculo $x^2 + y^2 = 1$ é igual a π.

(a) Determine uma aproximação para a área limitada por este círculo no primeiro quadrante usando o método de trapézio com h = 0.1 e determine uma estimativa para π a partir disto.

Sabendo que o $x^2 + y^2 = 1$, podemos isolar então y para passar como f(x) no método do trapézio

Logo teremos a função; $f(x) = y = \sqrt{1 - x^2}$

Como o círculo é de raio 1, logo nosso intervalo de integração será de 0 a 1

Pelo enunciado, é necessário que o tamanho do intervalo h seja de 0.1, nas outras questões estamos passando como parâmetro a quantidade de intervalos, agora vamos passar justamente o tamanho de cada intervalo, e também faremos uma pequena modificação para que também consigamos calcular a quantidade de intervalos.

Utilizaremos então o método trapezioh(f,a,b,h), que tem essa cara:

```
In [73]: #exercicio 3a
         function trapezioh(f,a,b,h) #calcular a integral f(x) de a até b
             #h=(b-a)/n    #tamanho do intervalo baseado no numero de intervalos
             n = (b-a)/h   #numero de intervalos baseado no tamanho do intervalo.
             S=0.0          #soma total da integral
             for i=1:(n-1)  #calcula o "meio"
                 x=a+i*h
                 S+=2*f(x)
             end
             S=h/2*(S+f(a)+f(b)) #calcula "as pontas"
             return S            #retorna a soma
         end

Out[73]: trapezioh (generic function with 1 method)
```

Perceba que a única diferença são os parâmetros passados, que agora temos o tamanho do intervalo h e a terceira linha, que calculamos a quantidade de intervalos a partir do tamanho do intervalo passado.

Nesse método então vamos passar a função que manipulamos, chamada de circulo(x), com o intervalo de 0 a 1 e h = 0.1

Teremos então:

```
In [75]: circulo(x) = sqrt(1-x^2)
         trapezioh(circulo,0,1,0.1)

Out[75]: 0.7761295815620796
```

Logo essa é a aproximação da área do primeiro quadrante.

Como queremos encontrar uma aproximação para pi, vamos lembrar da função da área do círculo, que é dada por:

área = π * raio ^ 2

Como temos apenas ¼ da área, logo dividimos tudo por 4:

$$\frac{área}{4} = \frac{\pi * raio^2}{4}$$

Substituindo o ¼ de área que temos

$$0.7761295815620796 = \frac{\pi * raio^2}{4}$$

Sabendo que o raio é 1, vamos manipular essa função até chegar em:

$$\pi = 4 * 0.7761295815620796$$

Calculando isso no julia, temos a aproximação de π:

```
In [96]: pi = 4 * trapezioh(circulo,0,1,0.1)

Out[96]: 3.1045183262483182
```

Logo, a aproximação de π para esse método é de **3.1045183262483182**

(b) Por que não podemos usar a fórmula do erro da regra da trapézio
para estimar o erro no item anterior?

Sabendo que o erro do método do trapézio é dado por

$$\left| \int_a^b \text{f(x) - (método do trapézio)} \right| \leq \frac{h\verb|^|3\ * N * M}{12}$$

Se, somente se, | f’’(x) | ≤ M , para a ≤ x ≤ b.

Nossa função é f(x) = y = $\sqrt{1 - x^2}$, logo precisamos calcular f’’(x)

Como f’’(x) = $\frac{-1}{(1 - x\verb|^|2)^{(3/2)}}$

Logo, a derivada segunda de f(x) não está determinada no intervalo de integração, visto que

$$\frac{-1}{(1 - 1\verb|^|2)^{(3/2)}} = \frac{-1}{0}$$

Como chegamos em uma divisão por 0, a derivada segunda não está determinada no intervalo de integração de 0 a 1, logo não podemos utilizar a forma do erro do trapézio para estimar o erro no item anterior.

4. (Bonus) Adapte a função de integral dupla da Aula 18 em Julia para
aproximar uma integral dupla $\int_a^b \int_{h(y)}^{g(y)}$ f(x,y) dxdy

Nessa função, basta mudarmos os limites de integração que foram passados no código em aula.

O método trapézio terá essa cara:

```
In [17]: function trapezio(f,a,b,n) #calcular a integral f(x) de a até b
             h=(b-a)/n            #tamanho dos intervalos baseado no limite de integração e o numero de intervalos
             S=0.0                #soma da integral
             for i=1:(n-1)       #calcula o "meio"
                 x=a+i*h         #avanço de x
                 S+=2*f(x)       #incrementando a soma
             end
             S=h/2*(S+f(a)+f(b)) #calcula "as pontas"
             return S             #retorna a soma S
         end

Out[17]: trapezio (generic function with 1 method)
```

Depois chamaremos um método Integral, que basicamente resolve o método trapézio mas com o número de intervalos pré determinados em 1000:

```
In [22]: function Integral(f,a,b)
             return trapezio(f,a,b,1000) # calculando uma integral com 1000 intervalos
         end
```

Agora para o método da Integral_Dupla(função,hy,gy,a,b) teremos:

```
In [28]: function Integral_Dupla(funcao,hy,gy,a,b) #integral dupla de funcao(x,y) de g(y) até g(y) no x e de a até b no y
             function g(y)
                 f(x)=funcao(x,y)             #função passada com 2 parametros
                 return Integral(f,hy(y),gy(y)) # integrando de h(y) até g(y) em relação a x
             end
             return Integral(g,a,b) #integrando agora o resultado de a até b em relação a y
         end

Out[28]: Integral_Dupla (generic function with 1 method)
```

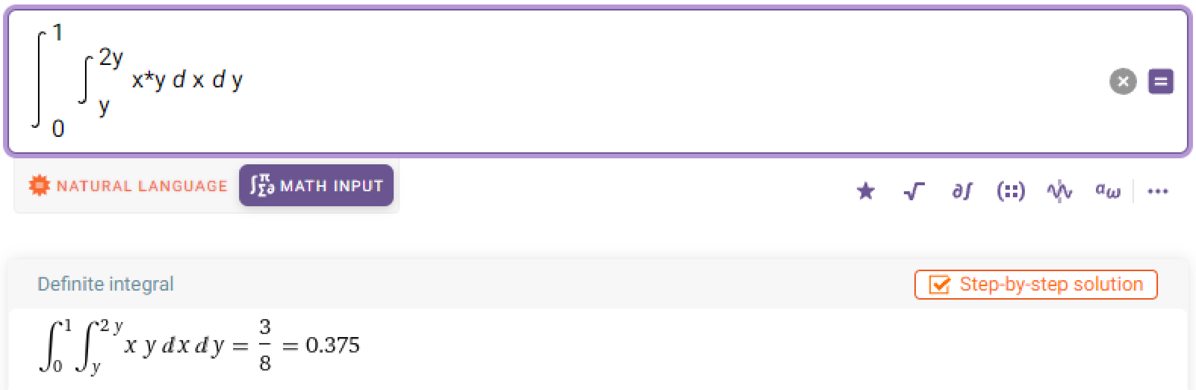
O que está acontecendo é, vamos passar como parâmetro a nossa função que queremos integrar, as funções h(y) e g(y) para serem os limites de integração da parte interior, a e b como os limites de integração da parte exterior.

Dentro do método Integral_Dupla, temos o método g(x) que vai retornar a “integral de dentro”, essa que será calculada com os limites de integração de hy(y) e gy(y) com o método Integral.

Com esse método retornar o resultado da Integral interior, vamos passá-lo para integrar com os limites de a até b.

Para testar se está realmente funcionando, vamos testar integrar a função f(x) = x*y , com os intervalos internos de h(y) = y até g(y) = 2y, e intervalos externos de 0 até 1

Tomando o wolframalpha como parâmetro para a função temos:



$$\int_0^1 \int_y^{2y} x\, y\, dx\, dy = \frac{3}{8} = 0.375$$

Agora utilizando nosso método:

```
In [29]: f(x,y) = x*y
         h(y) = y
         g(y) = 2y

Out[29]: g (generic function with 1 method)

In [30]: Integral_Dupla(f,h,g,0,1)

Out[30]: 0.3750003750000003
```

Podemos então verificar a validade do nosso método **Integral_Dupla**.