



**Institución Universitaria**  
Acreditada en Alta Calidad

## Inteligencia Artificial - IA184

### Instituto Tecnológico Metropolitano

#### Pedro Atencio Ortiz - 2018

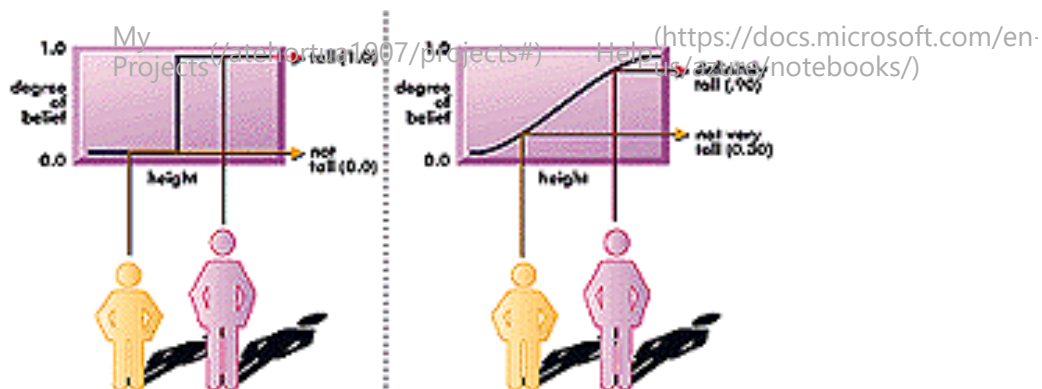
Parte de este material es autoría del profesor Cristian Guarnizo de la Universidad Tecnológica de Pereira. A él damos créditos.

En este notebook se aborda el tema de lógica difusa utilizando códigos ejecutables en Python:

- Conjuntos difusos.
- Funciones de pertenencia.
- Variable difusa.
- Fusificación (emborrosamiento).
- Base de reglas.
- Operadores difusos.
- Desfusificación.

---

## Módulo 3\_1: Logica Difusa

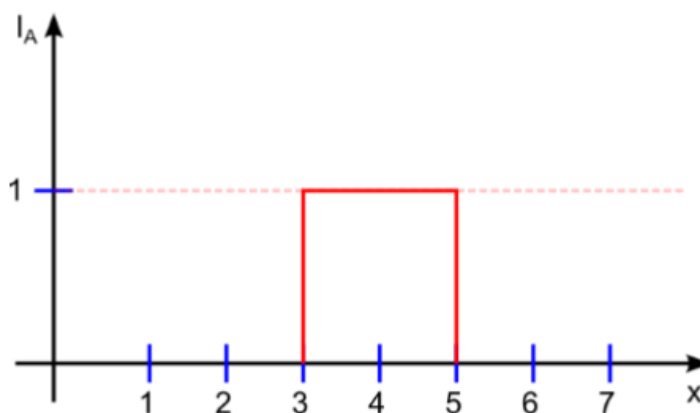


En términos reducidos, la lógica borrosa es un tipo de lógica que permite valores imprecisos (intermedios) para poder definir evaluaciones convencionales entre sí/no, verdadero/falso, negro/blanco, etc.

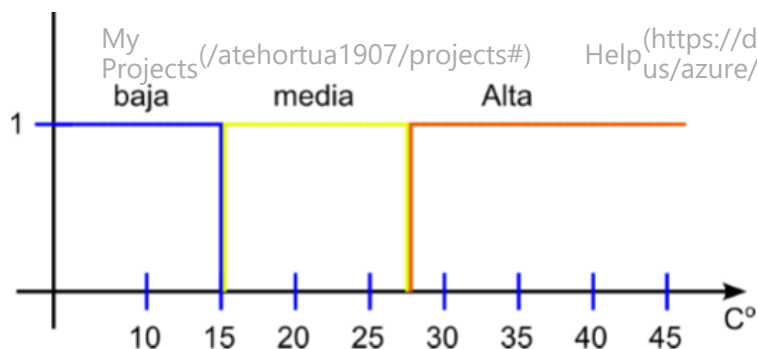
"El poder de la lógica difusa radica en que se puede analizar un sistema utilizando variables lingüísticas" [5].

## Conjunto Booleano

En el algebra de Boole(George Boole) las variables toman valores falso/verdadero (0, 1). Definamos un conjunto  $A = [3, 5]$  y  $x \in \{0, 7\}$ . Los valores de  $x$  que pertenecen a  $A$  reciben valor verdadero (1).



Por ejemplo, para el caso de la variable temperatura, si quisiéramos definir los valores temperatura baja, media y alta, utilizando lógica booleana, tendríamos:



En el ejemplo anterior, un elemento (valor de temperatura) tiene un grado de pertenencia a un conjunto o grupo, entonces, ¿A los 15 grados Celsius la temperatura es baja y repentinamente a los 16 es media?

In [10]:

```
"""
Implements a Boolean set.

Arguments:
x_domain -- domain of variable x (array)
x_set -- tuple of values that represents the set within domain of x

Returns:
parameters -- boolean (True, False) list of values inside x_set
"""
import numpy as np

def boolean_set(x_domain, x_set):
    val = np.logical_and(x_domain >= x_set[0], x_domain <= x_set[1])
    return val
```

Grafiquemos algunos conjuntos Booleanos para una variable  $x$  en el rango 0, 10

In [11]:

```
import matplotlib.pyplot as plt
```

```

limit1 = 0
limit2 = 100
domain = np.linspace(limit1, limit2, 500)

menor = (0.1,12)
bool_set_1 = boolean_set(domain,menor)

adolescente = (13,17)
bool_set_2 = boolean_set(domain,adolescente)

adulto = (18,90)
bool_set_3 = boolean_set(domain,adulto)

plt.fill(domain, bool_set_1)
plt.fill(domain, bool_set_2)
plt.fill(domain, bool_set_3)

plt.title("Three Boolean sets for variable x")
plt.show()

```

<Figure size 640x480 with 1 Axes>

In [12]:

```

import numpy as np

rango = np.linspace(0,20, 10)
print rango

[ 0.          2.22222222  4.44444444  6.66666667  8.88888889 11.11111111
 11
 13.33333333 15.55555556 17.77777778 20.          ]

```

## Trabajemos!

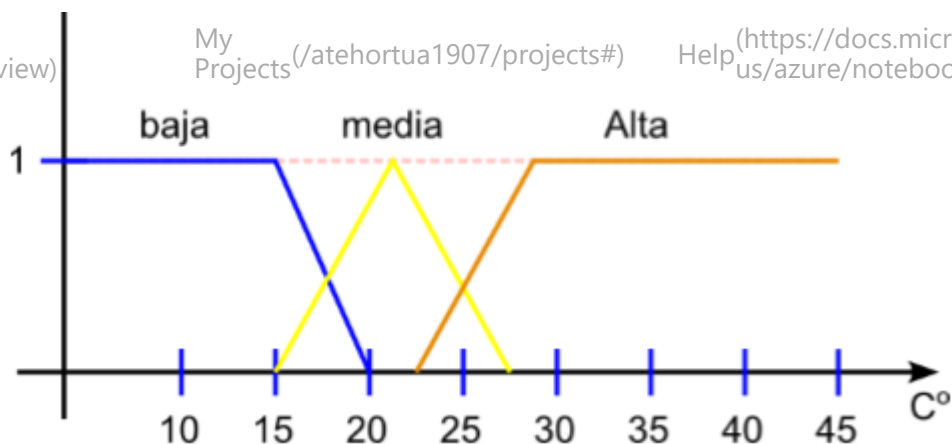
Implemente los conjuntos Booleanos que definen la mayoría de edad de una persona en Colombia.

## Conjunto difuso ( $\mu$ )

En un conjunto difuso, los grados de pertenencia son continuos y no binarios (falso, verdadero), de forma tal que una variable puede pertenecer en distinto grado a varios conjuntos.

En el siguiente ejemplo definimos tres conjuntos difusos  $\mu_{baja}$ ,  $\mu_{media}$  y  $\mu_{alta}$  para la variable temperatura, en el rango [0, 45] grados centígrados.

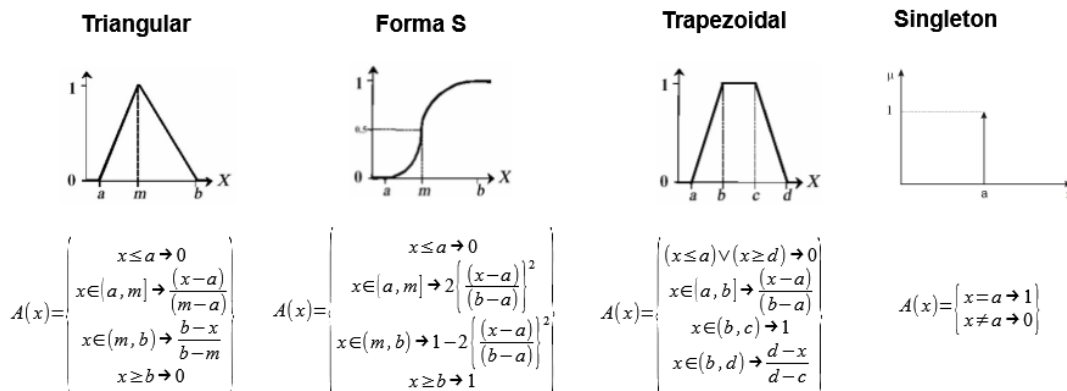




En el caso anterior, ¿un valor de temperatura igual a 17 grados será baja o media?

## Funciones de pertenencia

Un conjunto difuso  $\mu$  se define mediante una función de pertenencia. Dicha función define la forma en que se distribuye la pertenencia de una variable  $x$  a un conjunto  $S$  cambia. A continuación, algunas funciones de pertenencia y su definición algebraica:



Dependiendo de la naturaleza de la variable que se estudia, se define el número de conjuntos difusos y la función de pertenencia de cada uno.

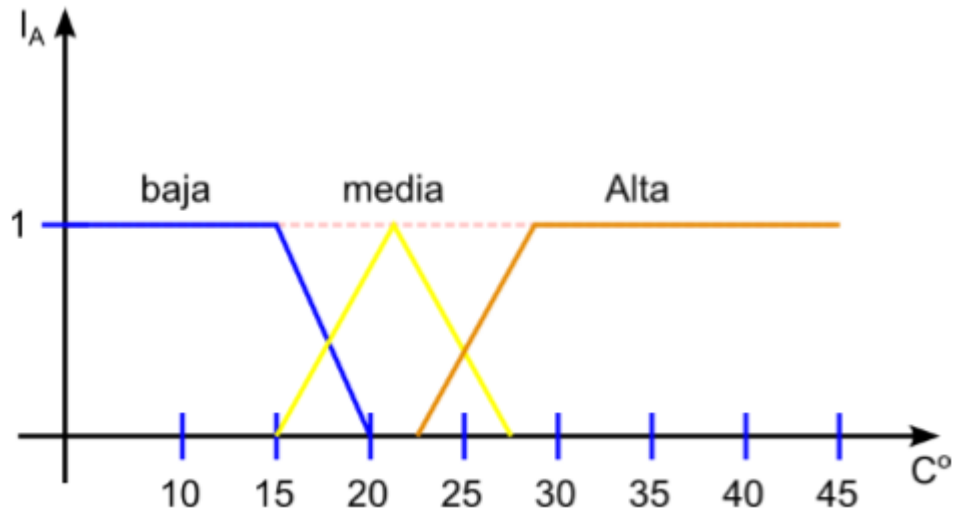
## Trabajemos!

Defina los conjuntos difusos para la variable edad, respecto a las categorías infante, adolescente, adulto, cenil.

## Variable difusa ( $x_{fuzzy}$ )

Una variable difusa está conformada por los valores de pertenencia de dicha variable respecto a cada conjunto. Entonces, la variable temperatura que estudiamos anteriormente se define como:

$$t_{fuzzy} = [\mu_{baja}(t), \mu_{media}(t), \mu_{alta}(t)]$$



## Fusificación

Esta operación consiste en transformar una variable normal (crisp) en una variable fuzzy. Para ello, se aplica calcula por cada conjunto difuso, el grado de pertenencia de dicha variable. Por ejemplo para el caso de la temperatura, supongamos el caso  $t = 10$ , entonces:

- $\mu_{baja}(t) = 1$
- $\mu_{media}(t) = 0$
- $\mu_{alta}(t) = 0$

Por lo tanto  $t_{fuzzy} = [1, 0, 0]$

```
In [13]: diccionario = {"menor":17, "mayor":20}
print(diccionario["menor"])
```

17

```
In [14]: """
Microsoft Implements a membership function
```

implements a membership function.

Arguments:

`x` -- crisp variable

`membership_function` -- string with the name of membership function: "trapezoid"

`parameters` -- dictionary of parameters for membership function

Returns:

`m_x` -- float membership of `x` with respect to `membership_function`

"""

`import numpy as np`

`def membership(x, membership_function, parameters):`

`m_x = 0`

`if(membership_function == "trapezoid"):`

`a = float(parameters["a"])`

`b = float(parameters["b"])`

`c = float(parameters["c"])`

`d = float(parameters["d"])`

`if(x <= a or x >= d):`

`m_x = 0`

`elif(x >= a and x <= b):`

`m_x = (x-a)/(b-a)`

`elif(x > b and x < c):`

`m_x = 1`

`elif(x >= c and x < d):`

`m_x = (d-x)/(d-c)`

`elif(membership_function == "triangular"):`

`a = float(parameters["a"])`

`m = float(parameters["m"])`

`b = float(parameters["b"])`

`if(x <= a):`

`m_x = 0`

`elif(x > a and x <= m):`

`m_x = (x-a)/(m-a)`

`elif(x > m and x < b):`

`m_x = (b-x)/(b-m)`

`else:`

`m_x = 0`

`return m_x`

## Trabajemos!

Utilizando las funciones de pertenencia implementadas, fusificar la variable

*temperatura* =  $16.7 \in T$ :

In [15]: #Completar: aproximadamente 3 lineas de codigo

Microsoft

parameters conjunto\_haia = {"a":0, "b":0, "c":15, "d":20}

```

parameters_conjunto_baja = {"a":0, "b":0, "c":15, "d":20}
parameters_conjunto_media = {"a":15, "b":21.5, "c":27.5}
parameters_conjunto_alta = {"a":21.5, "b":27.5, "c":40, "d":40}

t = 16.7

#completar: aproximadamente 3 lineas de codigo
miu_baja = membership(t, "trapezoid", parameters_conjunto_baja)
miu_media = membership(t, "triangular", parameters_conjunto_media)
miu_alta = membership(t, "trapezoid", parameters_conjunto_alta)

#completar la lista de pertenencias
t_fuzzy = [miu_baja, miu_media, miu_alta]

print "temperatura fuzzy: ", t_fuzzy

```

```
temperatura fuzzy: [0.6600000000000001, 0.26153846153846144, 0]
```

## Obtengamos y visualicemos todo el conjunto fuzzy utilizando la función anterior

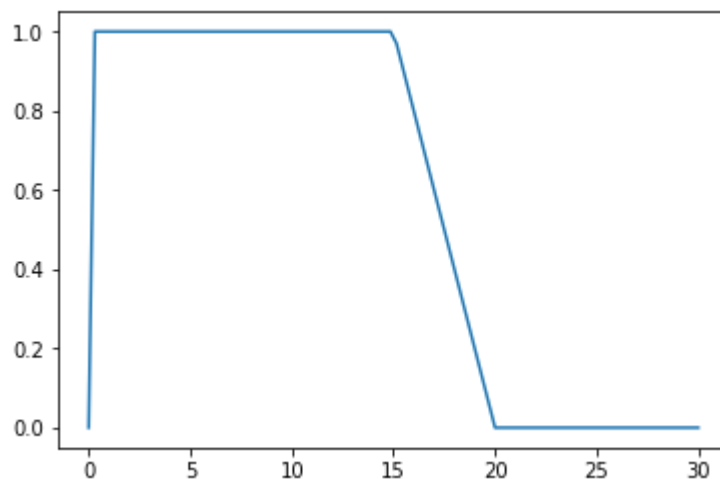
```

In [16]: resolution = 100
T = np.linspace(0, 30, resolution)

#utilicemos comprehension de python para generar todo el vector del conjunto
conjunto_baja = np.array([membership(t, "trapezoid", parameters_conjunto_baja) for t in T])

plt.plot(T, conjunto_baja)
plt.show()

```



```

In [17]: print T
          print conjunto_baja

```



Azure Notebooks (/#)	Preview (/help/preview)	My Projects (/atehortua1907/projects#)	Help (https://docs.microsoft.com/en-us/azure/notebooks/)
52	0.3030303	0.6060606	0.9090909
33	1.8181818	2.1212121	2.4242424
15	3.6363636	3.9393939	4.2424242
97	5.4545454	5.7575757	6.0606060
79	7.2727272	7.5757575	7.8787878
61	9.0909090	9.3939393	9.6969697
42	10.9090909	11.2121212	11.5151515
24	12.7272727	13.0303030	13.3333333
06	14.5454545	14.8484848	15.1515151
88	16.3636363	16.6666667	16.9696969
7	18.1818181	18.4848484	18.7878787
52	20.0000000	20.3030303	20.6060606
33	21.8181818	22.1212121	22.4242424
15	23.6363636	23.9393939	24.2424242
97	25.4545454	25.7575757	26.0606060
79	27.2727272	27.5757575	27.8787878
61	29.0909090	29.3939393	29.6969697
42	30.0000000	30.3030303	30.6060606
24	31.8181818	32.1212121	32.4242424
06	33.6363636	33.9393939	34.2424242
88	35.4545454	35.7575757	36.0606060
7	37.2727272	37.5757575	37.8787878
52	39.0909090	39.3939393	39.6969697
33	40.9090909	41.2121212	41.5151515
15	42.7272727	43.0303030	43.3333333
97	44.5454545	44.8484848	45.1515151
79	46.3636363	46.6666667	46.9696969
61	48.1818181	48.4848484	48.7878787
42	50.0000000	50.3030303	50.6060606
24	51.8181818	52.1212121	52.4242424
06	53.6363636	53.9393939	54.2424242
88	55.4545454	55.7575757	56.0606060
7	57.2727272	57.5757575	57.8787878
52	59.0909090	59.3939393	59.6969697
33	60.9090909	61.2121212	61.5151515
15	62.7272727	63.0303030	63.3333333
97	64.5454545	64.8484848	65.1515151
79	66.3636363	66.6666667	66.9696969
61	68.1818181	68.4848484	68.7878787
42	70.0000000	70.3030303	70.6060606
24	71.8181818	72.1212121	72.4242424
06	73.6363636	73.9393939	74.2424242
88	75.4545454	75.7575757	76.0606060
7	77.2727272	77.5757575	77.8787878
52	79.0909090	79.3939393	79.6969697
33	80.9090909	81.2121212	81.5151515
15	82.7272727	83.0303030	83.3333333
97	84.5454545	84.8484848	85.1515151
79	86.3636363	86.6666667	86.9696969
61	88.1818181	88.4848484	88.7878787
42	90.0000000	90.3030303	90.6060606
24	91.8181818	92.1212121	92.4242424
06	93.6363636	93.9393939	94.2424242
88	95.4545454	95.7575757	96.0606060
7	97.2727272	97.5757575	97.8787878
52	99.0909090	99.3939393	99.6969697
33	100.9090909	101.2121212	101.5151515
15	102.7272727	103.0303030	103.3333333
97	104.5454545	104.8484848	105.1515151
79	106.3636363	106.6666667	106.9696969
61	108.1818181	108.4848484	108.7878787
42	110.0000000	110.3030303	110.6060606
24	111.8181818	112.1212121	112.4242424
06	113.6363636	113.9393939	114.2424242
88	115.4545454	115.7575757	116.0606060
7	117.2727272	117.5757575	117.8787878
52	119.0909090	119.3939393	119.6969697
33	120.9090909	121.2121212	121.5151515
15	122.7272727	123.0303030	123.3333333
97	124.5454545	124.8484848	125.1515151
79	126.3636363	126.6666667	126.9696969
61	128.1818181	128.4848484	128.7878787
42	130.0000000	130.3030303	130.6060606
24	131.8181818	132.1212121	132.4242424
06	133.6363636	133.9393939	134.2424242
88	135.4545454	135.7575757	136.0606060
7	137.2727272	137.5757575	137.8787878
52	139.0909090	139.3939393	139.6969697
33	140.9090909	141.2121212	141.5151515
15	142.7272727	143.0303030	143.3333333
97	144.5454545	144.8484848	145.1515151
79	146.3636363	146.6666667	146.9696969
61	148.1818181	148.4848484	148.7878787
42	150.0000000	150.3030303	150.6060606
24	151.8181818	152.1212121	152.4242424
06	153.6363636	153.9393939	154.2424242
88	155.4545454	155.7575757	156.0606060
7	157.2727272	157.5757575	157.8787878
52	159.0909090	159.3939393	159.6969697
33	160.9090909	161.2121212	161.5151515
15	162.7272727	163.0303030	163.3333333
97	164.5454545	164.8484848	165.1515151
79	166.3636363	166.6666667	166.9696969
61	168.1818181	168.4848484	168.7878787
42	170.0000000	170.3030303	170.6060606
24	171.8181818	172.1212121	172.4242424
06	173.6363636	173.9393939	174.2424242
88	175.4545454	175.7575757	176.0606060
7	177.2727272	177.5757575	177.8787878
52	179.0909090	179.3939393	179.6969697
33	180.9090909	181.2121212	181.5151515
15	182.7272727	183.0303030	183.3333333
97	184.5454545	184.8484848	185.1515151
79	186.3636363	186.6666667	186.9696969
61	188.1818181	188.4848484	188.7878787
42	190.0000000	190.3030303	190.6060606
24	191.8181818	192.1212121	192.4242424
06	193.6363636	193.9393939	194.2424242
88	195.4545454	195.7575757	196.0606060
7	197.2727272	197.5757575	197.8787878
52	199.0909090	199.3939393	199.6969697
33	200.9090909	201.2121212	201.5151515
15	202.7272727	203.0303030	203.3333333
97	204.5454545	204.8484848	205.1515151
79	206.3636363	206.6666667	206.9696969
61	208.1818181	208.4848484	208.7878787
42	210.0000000	210.3030303	210.6060606
24	211.8181818	212.1212121	212.4242424
06	213.6363636	213.9393939	214.2424242
88	215.4545454	215.7575757	216.0606060
7	217.2727272	217.5757575	217.8787878
52	219.0909090	219.3939393	219.6969697
33	220.9090909	221.2121212	221.5151515
15	222.7272727	223.0303030	223.3333333
97	224.5454545	224.8484848	225.1515151
79	226.3636363	226.6666667	226.9696969
61	228.1818181	228.4848484	228.7878787
42	230.0000000	230.3030303	230.6060606
24	231.8181818	232.1212121	232.4242424
06	233.6363636	233.9393939	234.2424242
88	235.4545454	235.7575757	236.0606060
7	237.2727272	237.5757575	237.8787878
52	239.0909090	239.3939393	239.6969697
33	240.9090909	241.2121212	241.5151515
15	242.7272727	243.0303030	243.3333333
97	244.5454545	244.8484848	245.1515151
79	246.3636363	246.6666667	246.9696969
61	248.1818181	248.4848484	248.7878787
42	250.0000000	250.3030303	250.6060606
24	251.8181818	252.1212121	252.4242424
06	253.6363636	253.9393939	254.2424242
88	255.4545454	255.7575757	256.0606060
7	257.2727272	257.5757575	257.8787878
52	259.0909090	259.3939393	259.6969697
33	260.9090909	261.2121212	261.5151515
15	262.7272727	263.0303030	263.3333333
97	264.5454545	264.8484848	265.1515151
79	266.3636363	266.6666667	266.9696969
61	268.1818181	268.4848484	268.7878787
42	270.0000000	270.3030303	270.6060606
24	271.8181818	272.1212121	272.4242424
06	273.6363636	273.9393939	274.2424242
88	275.4545454	275.7575757	276.0606060
7	277.2727272	277.5757575	277.8787878
52	279.0909090	279.3939393	279.6969697
33	280.9090909	281.2121212	281.5151515
15	282.7272727	283.0303030	283.3333333
97	284.5454545	284.8484848	285.1515151
79	286.3636363	286.6666667	286.9696969
61	288.1818181	288.4848484	288.7878787
42	290.0000000	290.3030303	290.6060606
24	291.8181818	292.1212121	292.4242424
06	293.6363636	293.9393939	294.2424242
88	295.4545454	295.7575757	296.0606060
7	297.2727272	297.5757575	297.8787878
52	299.0909090	299.3939393	299.6969697
33	300.9090909	301.2121212	301.5151515
15	302.7272727	303.0303030	303.3333333
97	304.5454545	304.8484848	305.1515151
79	306.3636363	306.6666667	306.9696969
61	308.1818181	308.4848484	308.7878787
42	310.0000000	310.3030303	310.6060606
24	311.8181818	312.1212121	312.4242424
06	313.6363636	313.9393939	314.2424242
88	315.4545454	315.7575757	316.0606060
7	317.2727272	317.5757575	317.8787878
52	319.0909090	319.3939393	319.6969697
33	320.9090909	321.2121212	321.5151515
15	322.7272727	323.0303030	323.3333333
97	324.5454545	324.8484848	325.1515151
79	326.3636363	326.6666667	326.9696969
61	328.1818181	328.4848484	328.7878787
42	330.0000000	330.3030303	330.6060606
24	331.8181818	332.1212121	332.4242424
06	333.6363636	333.9393939	334.2424242
88	335.4545454	335.7575757	336.0606060
7	337.2727272	337.5757575	337.8787878
52	339.0909090	339.3939393	339.6969697
33	340.9090909	341.2121212	341.5151515
15	342.7272727	343.0303030	343.3333333
97	344.5454545	344.8484848	345.1515151
79	346.3636363	346.6666667	346.9696969
61	348.1818181	348.4848484	348.7878787
42	350.0000000	350.3030303	350.6060606
24	351.8181818	352.1212121	352.4242424
06	353.6363636	353.9393939	354.2424242
88	355.4545454	355.7575757	356.0606060
7	357.2727272	357.5757575	357.8787878
52	359.0909090	359.3939393	359.6969697
33	360.9090909	361.2121212	361.5151515
15	362.7272727	363.0303030	363.3333333
97	364.5454545	364.8484848	365.1515151
79	366.3636363	366.6666667	366.9696969
61	368.1818181	368.4848	

```
In [80]: def generate_fuzzy_set(X, membership_function, parameters):
         fuzzy_set = np.array([membership(x, membership_function, parameters)
         return fuzzy_set
```

## Grafiquemos todos los conjuntos fuzzy de la variable $T$

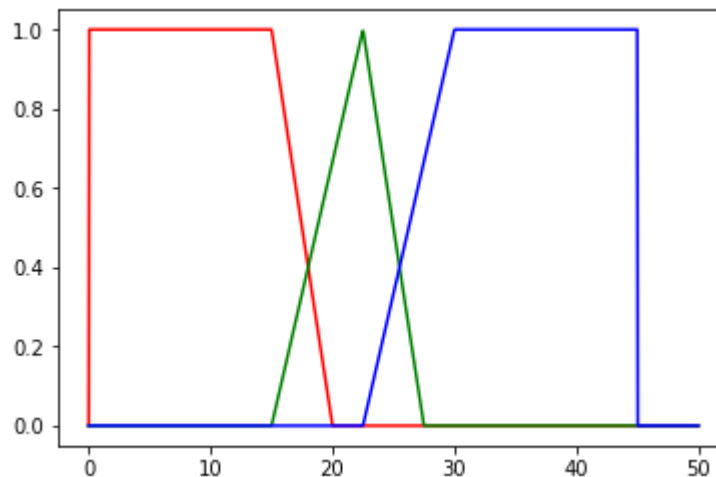
Utilicemos para ello la funcion generate\_fuzzy\_set

```
In [19]: params_conjunto_baja = {"a":0, "b":0, "c":15, "d":20}
         params_conjunto_media = {"a":15, "m":22.5, "b":27.5}
         params_conjunto_alta = {"a":22.5, "b":30, "c":45, "d":45}

         resolucion = 1000
         T = np.linspace(0, 50, resolucion)

         conjunto_baja = generate_fuzzy_set(T, "trapezoid", params_conjunto_baja)
         conjunto_media = generate_fuzzy_set(T, "triangular", params_conjunto_med
         conjunto_alta = generate_fuzzy_set(T, "trapezoid", params_conjunto_alta)

         plt.plot(T,conjunto_baja, 'r')
         plt.plot(T,conjunto_media, 'g')
         plt.plot(T,conjunto_alta, 'b')
         plt.show()
```



## Trabajemos!

Implementar una función generica fusificacion que calcule automáticamente la fusificación de una variable x respecto a n conjuntos difusos:

```
In [20]: """
         Implements a fuzzify function
```

Azure Notebooks (/#) Preview (/help/preview) My Projects (/atehortua1907/projects#) Help (https://docs.microsoft.com/en-us/azure/notebooks/)

implements a fuzzy function.

Arguments:

```
x -- crisp variable
fuzzy_sets -- dictionary with next form: {"set_name":(membership_function,
membership_function: string with the name of membership_function: "t",
parameters -- dictionary of parameters for membership function
```

Returns:

```
f_x -- dictionary of membership values of x with respect to each set in
"""
```

```
def fuzzify(x, fuzzy_sets):
    ##CODE HERE
    #f_x = []
    f_x = {} #lista de valores de pertenencia por cada conjunto difuso

    keys = fuzzy_sets.keys()

    for k in keys:
        #CODE HERE: utilizar la funcion membership(x, membership_function,
        #m = membership(x, fuzzy_sets[k]..., fuzzy_sets[k]...)
        m = membership(x, fuzzy_sets[k][0], fuzzy_sets[k][1])
        #f_x.append(k)
        f_x[k]=m

    return f_x
```

```
In [21]: lista = [1, "hola", 3]
dic = {"1":1, "a":2}
tupla = (1,2)
```

Utilicemos la función genérica para fusificar la variable  $t=16.7 \in T$

```
In [22]: params_conjunto_baja = {"a":0, "b":0, "c":15, "d":20}
params_conjunto_media = {"a":15, "m":22.5, "b":27.5}
params_conjunto_alta = {"a":22.5, "b":30, "c":45, "d":45}

funcion_baja = "trapezoid"
funcion_media = "triangular"
funcion_alta = "trapezoid"

key_baja = "baja"
key_media = "media"
key_alta = "alta"

t = 16.7
fuzzy_sets = {key_baja:(funcion_baja, params_conjunto_baja), key_media:(

print(fuzzify(t, fuzzy_sets))

{'baja': 0.6600000000000001, 'media': 0.22666666666666665, 'alta': 0}
```

```
In [23]: EDAD = np.linspace(0,50, 1000)
edad = 13
```

```

params_conjunto_infante = {"a":0, "b":0, "c":12, "d":16}
params_conjunto_adolescente = {"a":12, "m":16, "b":21}
params_conjunto_adulto = {"a":16, "b":21, "c":50, "d":50}

funcion_infante = "trapezoid"
funcion_adolescente = "triangular"
funcion_adulto = "trapezoid"

f_set_infante = generate_fuzzy_set(EDAD, funcion_infante, params_conjunt
f_set_adolescente = generate_fuzzy_set(EDAD, funcion_adolescente, params
f_set_adulto = generate_fuzzy_set(EDAD, funcion_adulto, params_conjunto_

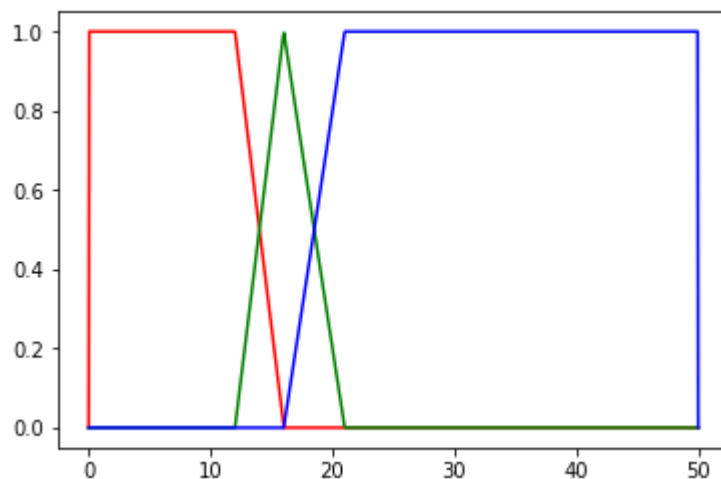
plt.plot(EDAD, f_set_infante, 'r')
plt.plot(EDAD, f_set_adolescente, 'g')
plt.plot(EDAD, f_set_adulto, 'b')
plt.show()

conjunto_infante = "infante"
conjunto_adolescente = "adolescente"
conjunto_adulto = "adulto"

diccionario_parametros = {conjunto_infante:(funcion_infante, params_conj
edad_fuzzy = fuzzify(edad, diccionario_parametros)

print("la variable concreta: ", edad, " fue fusificada como: ", edad_fuz

```



```

('la variable concreta: ', 13, ' fue fusificada como: ', {'adulto': 0,
'adolescente': 0.25, 'infante': 0.75})

```

In [24]: edad\_fuzzy["adolescente"]

Out[24]: 0.25

**Salida esperada:** [0.6600000000000001, 0.22666666666666657, 0]

## Base de reglas

Estas reglas especifican el conocimiento del **experto humano** en forma de condicionales que utilizan los nombres de los conjuntos difusos que tienen las variables difusas. Es decir, estas reglas expresan la relación entre los valores lingüísticos de la entrada con respecto a la salida.

Por ejemplo, utilizando este tipo de reglas y las variables difusas, podemos realizar preguntas en lenguaje natural, tales como: ¿Está la temperatura alta?

Las reglas se construyen con instrucciones IF-THEN:

IF  $x_1$  es "bajo" and  $x_2$  es "medio" entonces  $y$  es medio

Estas reglas requieren de operadores AND y OR específicamente diseñados para la lógica difusa.

## Operadores

Autor/Operador	A AND B	A OR B	NOT A
Zadeh	$\min(\mu_A, \mu_B)$	$\max(\mu_A, \mu_B)$	$1 - \mu_A$
Probabilístico	$\mu_A \cdot \mu_B$	$\mu_A + \mu_B - \mu_A \cdot \mu_B$	$1 - \mu_A$

Dependiendo de la definición de las reglas, los antecedentes se pueden combinar con los operadores AND y OR, o cambiar su valor con la negación NOT.

## Resolución de reglas y obtención de la salida

La resolución de reglas se refiere a como se interpreta la regla desde la lógica difusa. El proceso es el siguiente:

1. Seleccionar la pertenencia a la cuál hacen referencia los operandos de las reglas:

If  $\mu_{bajo}(x_1)$  AND  $\mu_{medio}(x_2)$  entonces  $Y_{bajo}$

2. Resolver los operadores difusos del antecedente:

$$K_{r1} = \text{AND}(\mu_{bajo}(x_1), \mu_{medio}(x_2))$$

3. Agrupar reglas con el mismo consecuente y aplicar operador OR:

$$A_{bajo} = \text{OR}(K_{r1} \in y_{bajo}, K_{r2} \in y_{bajo}, \dots, K_{rn} \in y_{bajo})$$

$$A_{medio} = \text{OR}(K_{r1} \in y_{medio}, K_{r2} \in y_{medio}, \dots, K_{rn} \in y_{medio})$$

$$A_{alto} = \text{OR}(K_{r1} \in y_{alto}, K_{r2} \in y_{alto}, \dots, K_{rn} \in y_{alto})$$

4. Resolver el consecuente de la regla mediante implicación (MIN) para cada conjunto de la salida.

$$S_{bajo} = \text{MIN}(A_{bajo}, \mu_{bajo}(y))$$

$$S_{medio} = \text{MIN}(A_{medio}, \mu_{medio}(y))$$

$$S_{alto} = \text{MIN}(A_{alto}, \mu_{alto}(y))$$

5. Obtener el área de respuesta aplicando mediante AGREGACION utilizando el operador MAX entre todos los resultados de las reglas ya implicadas.

$$Y_{fuzzy} = \text{MAX}(S_{bajo}, S_{medio}, S_{alto})$$

6. Obtener el valor final de la regla mediante DESFUSIFICACION.

$$salida = \text{desfusi}ficacion(Y_{fuzzy})$$

## 2. Resolver operador difuso

In [25]:

```
"""
Implements Zadeh's Fuzzy Operators.

Arguments:
operator -- string with name of operator: "AND", "OR", "NOT"
parameters -- dictionary with parameters of operator

Returns:
k -- operators value
"""

def fuzzy_operator(operator, parameters):
    k = 0
    if(operator == "AND"):
        a = parameters["a"]
        b = parameters["b"]
        k = min(a, b)
    elif(operator == "OR"):
        a = parameters["a"]
        b = parameters["b"]
        k = max(a, b)
    elif(operator == "NOT"):
        a = parameters["a"]
        k = 1 - a
    else:
        print("Invalid operator.")

    return k
```

In [26]:

```
parameters = {"a": 0.6}
print(fuzzy_operator("NOT", parameters))
```

### 3. Agrupar reglas (mismo consecuente)

Reglas que tengan un mismo consecuente, son agrupadas como un solo valor utilizando el operador OR == MAX (ZADEH), debido a que por lógica tradicional si dos antecedentes conducen a un mismo consecuente, es lo mismo que una disyunción entre dichos antecedentes, es decir:

si  $a < b$  entonces  $c$  y si  $a < f$  entonces  $c$

entonces, es lo mismo que:

si  $a < b$  OR  $a < f$  entonces  $c$

In [27]:

```
"""
Implements common consequent grouping.

Arguments:
rules -- list of values from resolved antecedents of rules with same consequent

Returns:
val -- max from rules
"""

def fuzzy_group(rules):
    val = np.max(rules)
    return val
```

In [28]:

```
r1 = 0.2
r2 = 0.1
r3 = 0.6

rules = [r1, r2, r3]

print fuzzy_group(rules)

0.6
```

### 4. Resolver consecuente mediante implicación

Lo primero que debemos hacer es generar el conjunto de salida al que hace referencia la regla o el conjunto de reglas ya agrupada.

Para ello utilizamos la función **generate\_fuzzy\_set** que construimos anteriormente.

supongamos un conjunto fuzzy  $A$  de una variable  $Y \in [0, 100]$ , de tipo triangular con parámetros  $a = 10$ ,  $m = 20$  y  $c = 30$ , realicemos una implicación sobre dicho

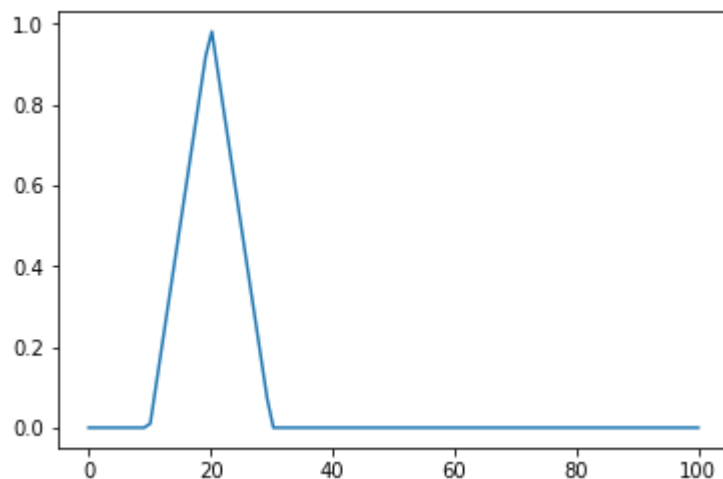
conjunto con valor de  $r = 0.4$

In [29]:

```
resolucion = 100
Y = np.linspace(0,100,resolucion)
params_set_A = {"a":10, "m":20, "b":30}

fuzzy_set_A = generate_fuzzy_set(Y, "triangular", params_set_A)

plt.plot(Y, fuzzy_set_A)
plt.show()
```



La implicación en lógica borrosa, se realiza mediante el operador **MIN**. Implementemos una función para ello:

In [30]:

```
"""
Implements fuzzy implication (MIN).

Arguments:
r -- scalar from rules solving and grouping.
fuzzy_set -- A fuzzy set (array of values).

Returns:
s -- implication result. An array of numeric values.
"""

def fuzzy_implication(r, fuzzy_set):
    val = np.minimum(r, fuzzy_set)
    return val
```

In [31]:

```
r = 0.4
implication_set = fuzzy_implication(r, fuzzy_set_A)
```

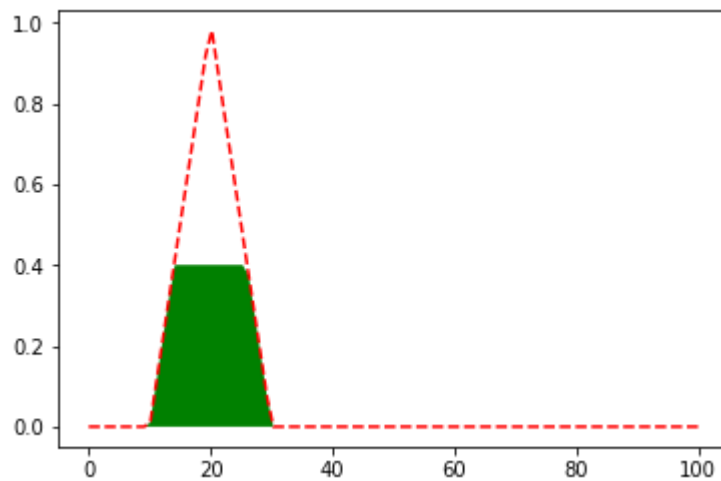


[Previous \(/help/previous\)](#)
[Microsoft Projects \(/atehortua1907/projects#\)](#)
[Help \(https://docs.microsoft.com/en-us/azure/notebooks/\)](#)

```

implication_set = fuzzy_implication(r, fuzzy_set_A)
plt.plot(Y, fuzzy_set_A, 'r--')
plt.fill(Y, implication_set, 'g')
plt.show()

```



Supongamos que tenemos tres conjuntos:  $A$  de tipo trapezoidal,  $B$  de tipo triangular y  $C$  de tipo triangular para la variable de salida  $Y \in [0, 50]$ , y que hemos tenemos los valores de los antecedentes de tres reglas que hacen referencia a cada conjunto así:  $r_A, r_B, r_C$ .

```

In [32]: resolution = 100
Y = np.linspace(0,50,resolucion)

params_set_A = {"a":0, "b":10, "c":15, "d":20}
params_set_B = {"a":10, "m":20, "b":30}
params_set_C = {"a":25, "m":30, "b":45}

Y_A = generate_fuzzy_set(Y, "trapezoid", params_set_A)
Y_B = generate_fuzzy_set(Y, "triangular", params_set_B)
Y_C = generate_fuzzy_set(Y, "triangular", params_set_C)

#1. apliquemos implicacion
r_A = 0.2
r_B = 0.5
r_C = 0.7

A_implicated = fuzzy_implication(r_A, Y_A)
B_implicated = fuzzy_implication(r_B, Y_B)
C_implicated = fuzzy_implication(r_C, Y_C)

```

```

In [33]: plt.plot(Y, Y_A, 'r--')
plt.plot(Y, Y_B, 'g--')

```

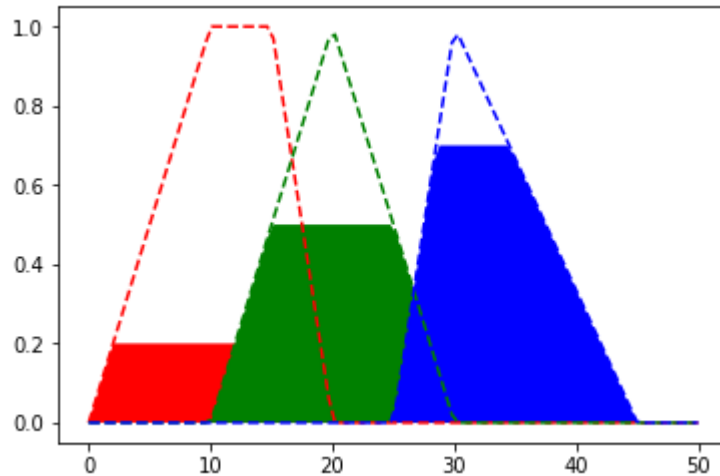
Azure  
Notebooks  
(/#)

Previous  
(/help/review)

Projects  
(/atehortua1907/projects#)

Help  
(https://docs.microsoft.com/en-us/azure/notebooks/)

```
plt.plot(Y, Y_C, 'b--')
plt.fill(Y, A_implicated, 'r')
plt.fill(Y, B_implicated, 'g')
plt.fill(Y, C_implicated, 'b')
plt.show()
```



## 5. Obtener el área de respuesta aplicando mediante AGREGACION utilizando el operador MAX entre todos los resultados de las reglas ya implicadas

In [34]:

```
"""
Implements Aggregation fuzzy operator using MAX.

Arguments:
fuzzy_sets -- A list with fuzzy_sets of output variable. All sets must h

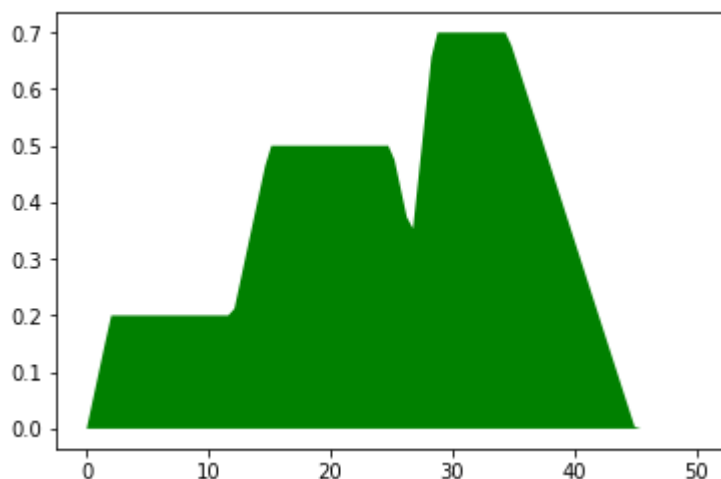
Returns:
val -- Aggregation result. An array of values.
"""
def fuzzy_aggregation(fuzzy_sets):
    val = np.zeros([fuzzy_sets[0].shape[0]])

    for s in fuzzy_sets:
        val = np.maximum(val, s)

    return val
```

In [35]:

```
agg_set = fuzzy_aggregation([A_implicated, B_implicated, C_implicated])
plt.fill(Y, agg_set, 'o')
```



In [36]: `print(agg_set)`

```
[0.          0.05050505 0.1010101  0.15151515 0.2          0.2
 0.2          0.2          0.2          0.2          0.2          0.2
 0.2          0.2          0.2          0.2          0.2          0.2
 0.21212121 0.26262626 0.31313131 0.36363636 0.41414141 0.46464646
 0.5          0.5          0.5          0.5          0.5          0.5
 0.5          0.5          0.5          0.5          0.5          0.5
 0.5          0.5          0.5          0.5          0.5          0.5
 0.5          0.5          0.47474747 0.42424242 0.37373737 0.35353535
 0.45454545 0.55555556 0.65656566 0.7          0.7          0.7
 0.7          0.7          0.7          0.7          0.7          0.7
 0.7          0.7          0.7          0.67676768 0.64309764 0.60942761
 0.57575758 0.54208754 0.50841751 0.47474747 0.44107744 0.40740741
 0.37373737 0.34006734 0.30639731 0.27272727 0.23905724 0.20538721
 0.17171717 0.13804714 0.1043771  0.07070707 0.03703704 0.003367
 0.          0.          0.          0.          0.          0.
 0.          0.          0.          0.          ]
```

## 6. Obtener el valor final de la regla mediante DESFUSIFICACION

Para ello podemos aplicar diferentes métodos. El más conocido es el del CENTROIDE o centro de área, definido como:

$$y = \frac{\int_{i \in Y} i \mu_{\text{aggregated}}(i) di}{\int_{i \in Y} \mu_{\text{aggregated}}(i) di}$$

Otro método popular es el del BISECTOR definido como: "el valor que separa el área

bajo la curva en dos sub-áreas iguales":

$$y \mid \int_{i \in (-\infty, y) \in Y} i \mu_{aggregated(i)} di \approx \int_{i \in [y, \infty) \in Y} i \mu_{aggregated(i)} di$$

Implemetemos el método del centroide y apliquemos el mismo sobre el resultado anterior:

In [37]:

```
"""
Implements defuzzification (centroid, bisector).

Arguments:
Y -- array with range of output variable Y
fuzzy_set_output -- A fuzzy_set of output variable.
method -- string with name of defuzzification method. can be "centroid",

Returns:
val -- scalar value of crisp output variable.
"""
def fuzzy_defuzzy(Y, fuzzy_set_output, method="centroid"):

    if(method == "centroid"):
        val = np.sum(Y * fuzzy_set_output) / np.sum(fuzzy_set_output)
    elif(method == "bisector"):
        val = None

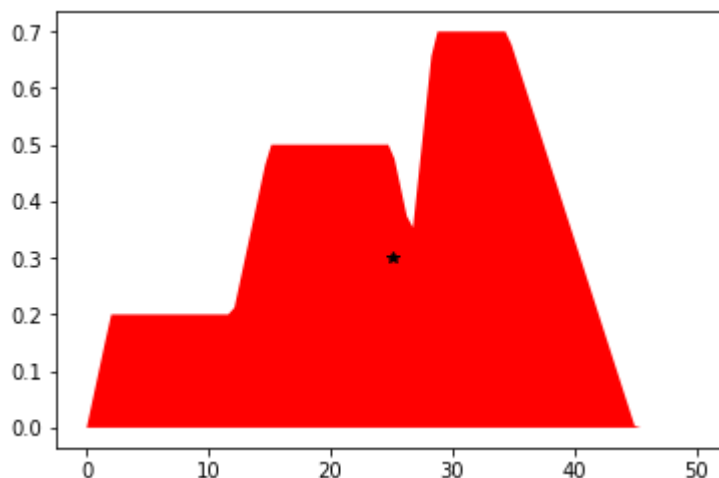
    return val
```

In [38]:

```
output = fuzzy_defuzzy(Y, agg_set, "centroid")
print("salida del sistema: " + output)
```

```
print('salida del sistema: ', output,
agg_set = fuzzy_aggregation([A_implicated, B_implicated, C_implicated])
plt.fill(Y, agg_set, 'r')
plt.plot(output, 0.3, 'k*')
plt.show()
```

('salida del sistema: ', 25.03236833717073)



## Taller en clase

### Recordemos! Hemos implementado las siguientes funciones:

- generate\_fuzzy\_set
- fuzzify
- fuzzy\_operator
- fuzzy\_group
- fuzzy\_implication
- fuzzy\_aggregation
- fuzzy\_defuzzy

In [39]: `??generate_fuzzy_set`

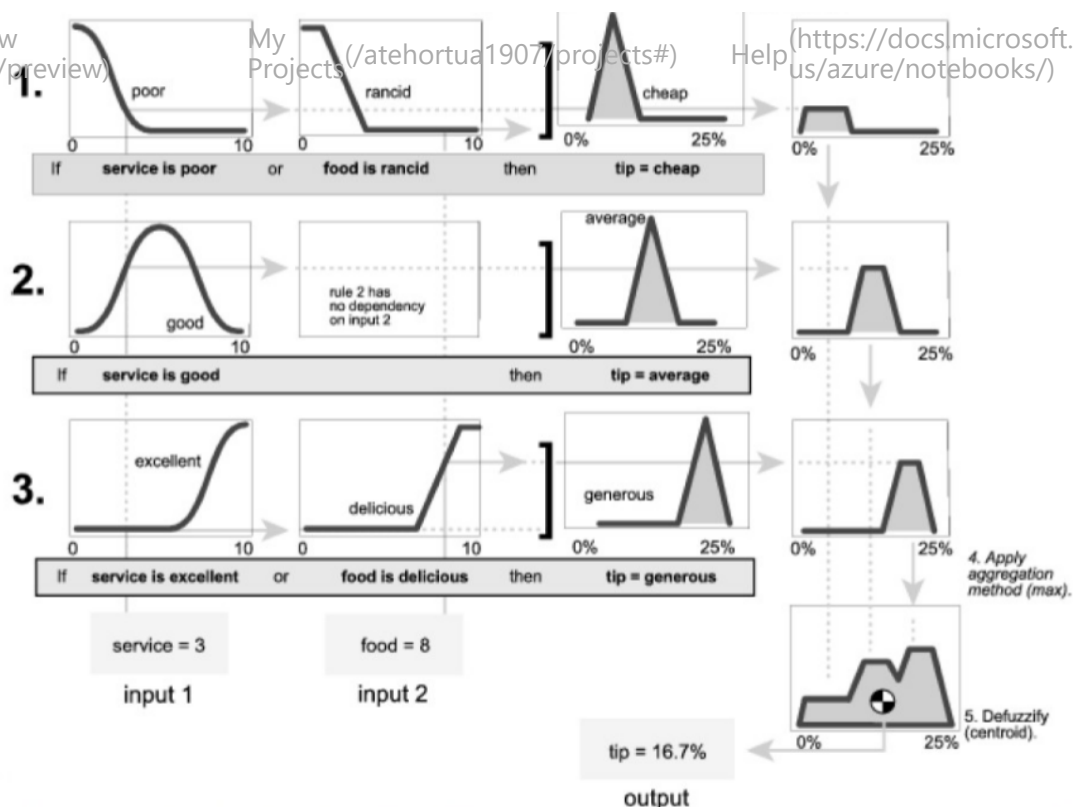
### 1. Implementemos el siguiente ejemplo y verifiquemos si nuestra implementación arroja un resultado similar:

Azure  
Notebooks  
(/#)

Preview  
(/help/preview)

My Projects  
(/atehortua1907/projects#)

Help  
(https://docs.microsoft.com/en-us/azure/notebooks/)



In [82]: #1. Inicializar parametros

```

service_poor = "poor"
service_good = "good"
service_excellent = "excellent"
funcion_service_poor = "trapezoid"
funcion_service_good = "triangular"
funcion_service_excellent = "trapezoid"
params_poor = {"a":0, "b":0, "c":2, "d":5}
params_good = {"a":2, "m":5, "b":8}
params_excellent = {"a":5, "b":7, "c":10, "d":10}

food_rancid = "rancid"
food_delicious = "delicious"
funcion_food_rancid = "trapezoid"
funcion_food_delicious = "trapezoid"
params_rancid = {"a":0, "b":4, "c":7, "d":4}
params_delicious = {"a":6, "b":8, "c":10, "d":10}

tip_cheap = "cheap"
tip_average = "average"
tip_generous = "generous"
funcion_tip_cheap = "triangular"
funcion_tip_average = "triangular"
funcion_tip_generous = "trapezoid"
params_cheap = {"a":0, "m":6, "b":12.5}
params_average = {"a":6, "m":12.5, "b":20}
params_generous = {"a":12.5, "b":20, "c":25, "d":30}

```

In [83]: #2 fusificar

Microsoft

```

service = 9
Food = 9

fuzzy_sets_servicio = {service_poor:(funcion_service_poor, params_poor),
service_fuzzy = fuzzify(service, fuzzy_sets_servicio)

fuzzy_sets_food = {food_rancid:(funcion_food_rancid, params_rancid), food
food_fuzzy = fuzzify(food, fuzzy_sets_food)

print(service_fuzzy)
print(food_fuzzy)

{'poor': 0, 'good': 0, 'excellent': 1}
{'delicious': 1, 'rancid': 0}

```

In [71]: `??generate_fuzzy_set`

In [84]: *#3. Resolver reglas*

```

r1 = fuzzy_operator("OR", {"a":service_fuzzy["poor"], "b":food_fuzzy["rancid"]
r2 = service_fuzzy["good"]
r3 = fuzzy_operator("OR", {"a":service_fuzzy["excellent"], "b":food_fuzzy["delicious"]

print(r1,r2,r3)

(0, 0, 1)

```

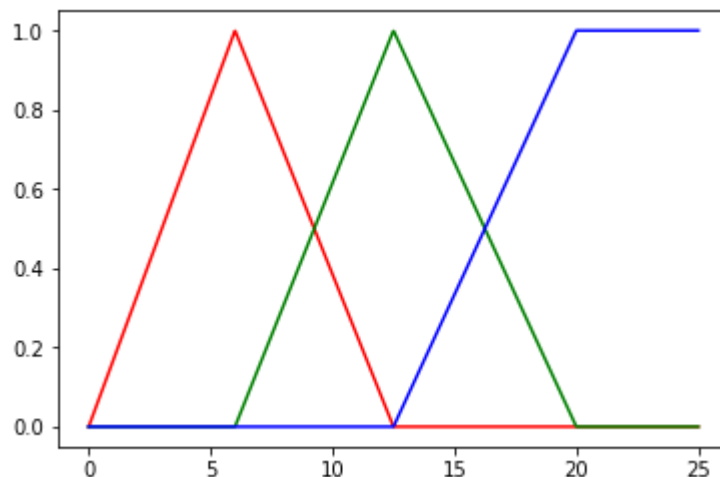
In [85]: *#4. Implicacion*

```

T = np.linspace(0,25, 1000)
fuzzy_set_cheap = generate_fuzzy_set(T, funcion_tip_cheap, params_cheap)
fuzzy_set_average = generate_fuzzy_set(T, funcion_tip_average, params_average)
fuzzy_set_generous = generate_fuzzy_set(T, funcion_tip_generous, params_generous)

plt.plot(T, fuzzy_set_cheap, 'r')
plt.plot(T, fuzzy_set_average, 'g')
plt.plot(T, fuzzy_set_generous, 'b')
plt.show()

```

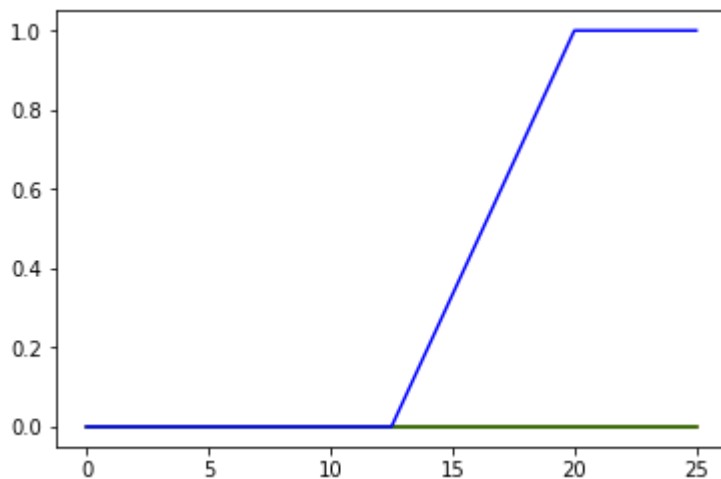


In [45]: `??fuzzy_implication`

Azure In [86]: Preview My Projects (https://docs.microsoft.com/en-  
 Notebooks (/help/preview) (/notebooks/1907/projects/#) Help Us (/azure/notebooks/)  
 (/#)

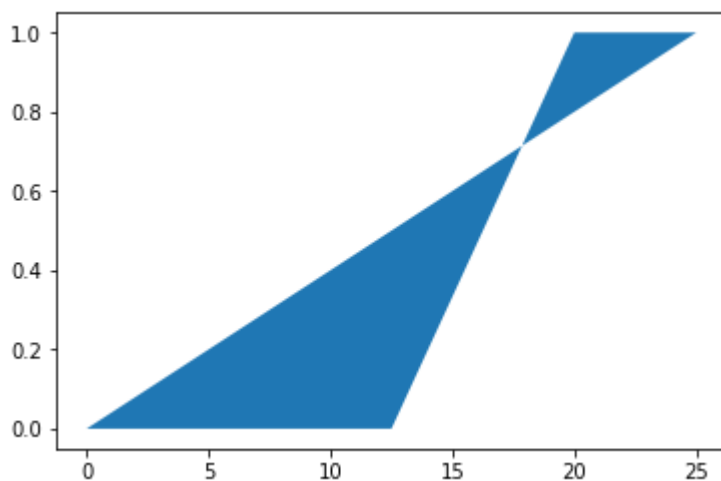
```
fs_cheap_truncated = fuzzy_implication(r1, fuzzy_set_cheap)
fs_average_truncated = fuzzy_implication(r2, fuzzy_set_average)
fs_generous_truncated = fuzzy_implication(r3, fuzzy_set_generous)

plt.plot(T, fs_cheap_truncated, 'r')
plt.plot(T, fs_average_truncated, 'g')
plt.plot(T, fs_generous_truncated, 'b')
plt.show()
```



In [47]: `??fuzzy_aggregation`

In [87]: `#5. Agregacion`  
`aggregated_tip = fuzzy_aggregation([fs_cheap_truncated, fs_average_truncated, fs_generous_truncated])`  
`plt.fill(T, aggregated_tip)`  
`plt.show()`



In [55]: `??fuzzy_defuzzy`

In [88]: `#6. desfusificar`  
`calida = fuzzy_defuzzy(T, aggregated_tip, "centroid")`



-----  
Azure Preview: <https://docs.microsoft.com/en-us/azure/notebooks/>  
Notebooks (/help/preview) Projects (atenhortua1907/projects#) Help  
(/#)

```
salida = fuzzy_defuzzy(f, aggregated_ip, control_ip,
print("al señor se le recomienda pagar: ", salida)
('al se\x3\xb1or se le recomienda pagar: ', 20.3637754869411)
```

In [ ]: