

# INVESTIGACIÓN OPERATIVA SUPERIOR

*nuevo módulo: planificación*

Virtual Sincrónica

## *segundo módulo: planificación*

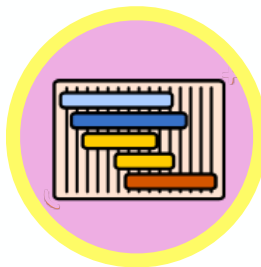
*programación de  
restricciones*



*presentación de  
resultados y reportes*

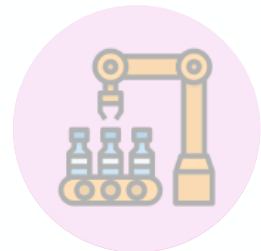


*planificación de la  
producción*



*planificación de  
proyectos*

*distintas bibliotecas  
para planificación*



*conexión de modelos a  
base de datos*

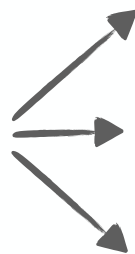
# *programación de restricciones*



*... pero, ¿qué son?*

*algoritmos de búsqueda basados en la satisfacción de las restricciones*

*¿cómo están  
formados?*



variables de  
decision

restricciones

funcional  
(opcional)

## nuestro simple problema ...

maximizar  $30x + 15y$

$x + 2y \leq 55$

$x \leq 50$

$y \leq 50$

digamos ahora que  $x$  e  $y$  son enteras



modelo de programación  
matemática

¿y en programación de  
restricciones?

```
from docplex.cp.model import CpoModel
```

```
m = CpoModel()
```

```
x = m.integer_var(name = 'x')
```

```
y = m.integer_var(name = 'y')
```

```
m.maximize(30*x + 15*y)
```

```
m.add_constraint(x + 2*y <= 55)
```

```
m.add_constraint(x <= 50)
```

```
m.add_constraint(y <= 50)
```

```
s = m.solve()
```

```
print(f"x
```

seguimos con  
la biblioteca:

**DOCPLEX**

DEMO 1

*¿por qué buscamos alternativas  
a la programación matemática?*

*tiempo de  
resolución*

*mejoras en la  
formulación*

*instrucciones  
especiales*

*algunas instrucciones  
especiales*

# contar valores en un vector

valor a contar  
↓  
`modelo.add(count(lista, v) == 1)`  
↑  
lista de python pero ...  
el tipo de variable debe ser  
compatible con CP

cargo la  
biblioteca

defino las  
variables

agrego la  
restricción

resuelvo el  
modelo

```
from docplex.cp.model import CpoModel  
m = CpoModel()  
(x = m.integer_var(min=0, max=100, name='x')  
y = m.integer_var(min=0, max=100, name='y'))  
m.add(m.count([x, y], 2) == 1)  
s = m.solve(TimeLimit=10)  
s.print_solution()
```

imprimo la  
solución

x = 2

y = 2

# valores diferentes

`modelo.add(all_diff(lista))`

↑  
*lista de python pero ...  
el tipo de variable debe ser  
compatible con CP*

*defino un  
array de  
enteros*

$$\sum k_i \leq 20$$

*restricción  
todos  
diferentes*

```
m = CpoModel()

w = m.integer_var(min=0, max=10, name="w")
x = m.integer_var(min=0, max=10, name="x")
y = m.integer_var(min=0, max=10, name="y")
z = m.integer_var(min=0, max=10, name="z")

k = [w, x, y, z]

m.add(sum(k) <= 20)

m.add(all_diff(k))
```



*luego de  
ejecutarlo*

```
m = CpoModel()

w = m.integer_var(min=0, max=10, name="w")
x = m.integer_var(min=0, max=10, name="x")
y = m.integer_var(min=0, max=10, name="y")
z = m.integer_var(min=0, max=10, name="z")

k = [w, x, y, z]

m.add(sum(k) == 20)

m.add(all_diff(k))
```



*nomenclatura  
muy importante*

```
m = CpoModel()

k = [m.integer_var(
    min=0,
    max=10,
    name="x" + str(i)
) for i in range(4)]

m.add(sum(k) == 20)

m.add(all_diff(k))
```

DEMO 2





## *ejemplo #1 sudoku*



## *reglas*

*completar los casilleros vacíos con números del 1 al 9 ...*

*no repetir  
números  
dentro de una  
columna*

5	3			7				
6			1	9	5			
	9	8					6	
8				6				3
4			8		3			1
7				2				6
	6					2	8	
			4	1	9			5
				8			7	9

*no repetir  
números  
dentro de una  
fila*

*no repetir  
números  
dentro de un  
cuadrante*

```
modelo = CpoModel()
```

```
k = modelo.integer_var(min=0, max=9, name="x")
```

**k: variable entera cuyo valor estará entre 0-9 y su nombre dentro del modelo será "x"**

					5	6	7	8
					5			
							6	
3	8				6			3
4	4			8		3		1
5	7				2			6
6		6					2	8
7				4	1	9		5
8					8		7	9

```
modelo = CpoModel()
```

```
k = [modelo.integer_var(min=0, max=9, name="x"+str(i)) for i in range(9)]
```

**k: vector (o lista) de 9 variables enteras con valores entre 0-9 con nombres dentro del modelo "x0, x1, x2, x3, x4, x5, x6, x7, x8"**

					5	6	7	8
					5			
							6	
								3
4	4			8		3		1
5	7				2			6
6		6					2	8
7				4	1	9		5
8					8		7	9

```
modelo = CpoModel()
```

```
k = [modelo.integer_var(min=0, max=9, name="x")+ str(i) + str(j))
for i in range(9)] for j in range(9)]
```

```
modelo.add(k[0][0] == 5)
```

```
modelo.add(k[0][1] == 5)
```

```
modelo.add(k[0][2] == 5)
```

```
modelo.add(k[1][0] == 5)
```

```
modelo.add(k[1][1] == 5)
```

```
modelo.add(k[1][2] == 5)
```

```
modelo.add(k[2][0] == 5)
```

```
modelo.add(k[2][1] == 5)
```

```
modelo.add(k[2][2] == 5)
```

```
modelo.add(k[3][0] == 5)
```

```
modelo.add(k[3][1] == 5)
```

```
modelo.add(k[3][2] == 5)
```

```
modelo.add(k[4][0] == 5)
```

```
modelo.add(k[4][1] == 5)
```

```
modelo.add(k[4][2] == 5)
```

```
etc ...
```

**k: matriz (o lista) de 9x9 variables enteras con valores entre 0-9 con nombres dentro del modelo "x00, x10, x20, ... , x88"**

		5	6	7	8
4	4				
5	7				
6		6			
7					
8					

*# elementos de la fila diferentes*

```
modelo.add(all_diff([k[0][0], k[0][1], k[0][2], k[0][3], k[0][4], k[0][5],
k[0][6], k[0][7], k[0][8], k[0][9]]))
```

**todos los elementos de la primera fila son diferentes ...**

*# elementos de la fila diferentes*

```
modelo.add(all_diff([k[0][j] for j in range(9)]))
modelo.add(all_diff([k[1][j] for j in range(9)]))
modelo.add(all_diff([k[2][j] for j in range(9)]))
modelo.add(all_diff([k[3][j] for j in range(9)]))
modelo.add(all_diff([k[4][j] for j in range(9)]))
modelo.add(all_diff([k[5][j] for j in range(9)]))
modelo.add(all_diff([k[6][j] for j in range(9)]))
modelo.add(all_diff([k[7][j] for j in range(9)]))
modelo.add(all_diff([k[8][j] for j in range(9)]))
modelo.add(all_diff([k[9][j] for j in range(9)]))
```

*# elementos de la fila diferentes*

```
for i in range(9): [k[0][j] for j in range(9)]))
    modelo.add(all_diff([k[i][j] for j in range(9)]))
```

*# elementos de la fila diferentes*

```
for i in range(9):  
    modelo.add(all_diff([k[i][j] for j in range(9)]))
```

*# elementos de la columna diferentes*

```
for j in range(9):  
    modelo.add(all_diff([k[i][j] for i in range(9)]))
```

*# elementos del cuadrante diferentes*

```
for x in [0, 3, 6]:  
    for y in [0, 3, 6]:  
        modelo.add(all_diff([k[i][j] for i in range(x, x + 3)  
                               for j in range(y, y + 3)]))
```

	0	1	2	3	4	5	6	7	8	9
0	5	3			7					
1	6			1	9	5				
2		9	8					6		
3	8				6				3	
4	4			8		3				1
5	7				2					6
6		6					2	8		
7				4	1	9				5
8						8		7	9	



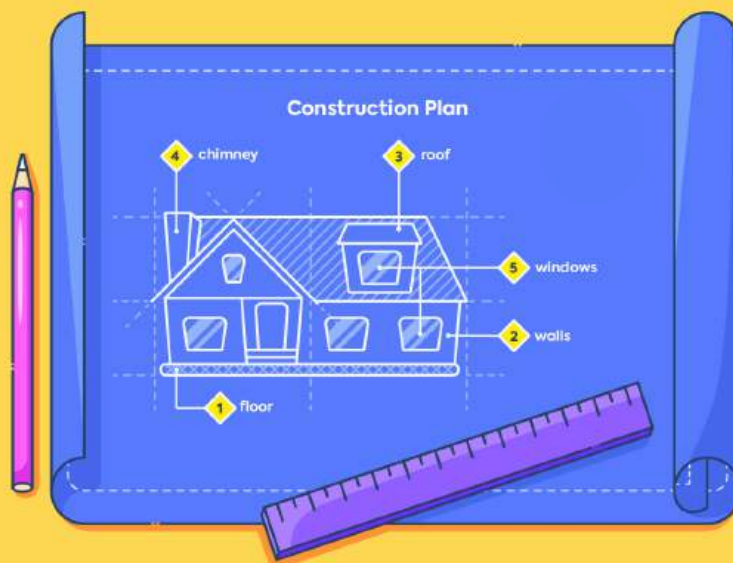
*¿preguntas?*

DEMO 3

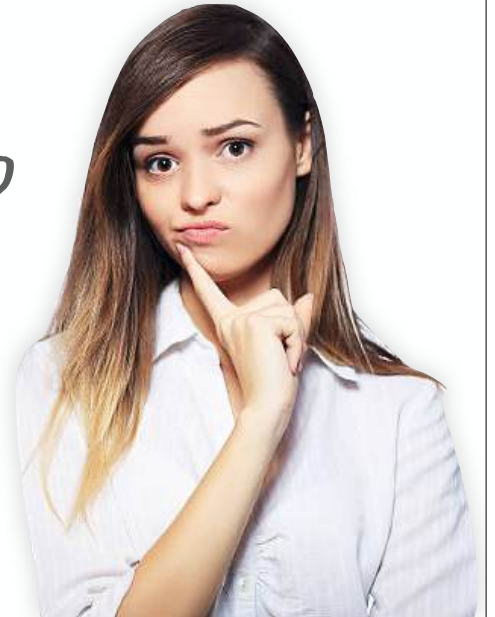
*pero, ¿y en qué es muy  
bueno programación  
de restricciones?*



*planificación*



*¿software "enlatado" o  
desarrollado "a medida"?*



**scheduling**

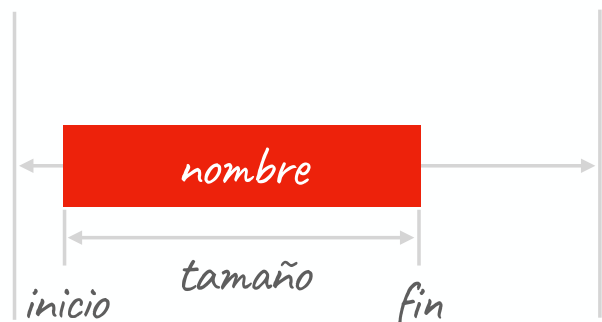
*(con constraint programming optimization)*

# intervalos

```
t = modelo.interval_var(  
    optional,  
    start,  
    end,  
    size,  
    name)
```

true si la  
tarea es  
opcional

## intervalo

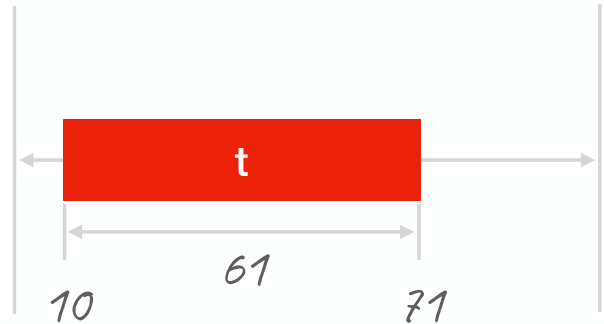




## intervalo

```
t = modelo.interval_var(  
    optional = False,  
    start = 10,  
    end = 71,  
    size = 61,  
    name = 't')
```

*false es la opción  
predeterminada*



*funciones para intervalos . . .*

## activación

```
t = modelo.interval_var(  
    optional = True,  
    start = 10,  
    end = 71,  
    size = 61,  
    name = 't')
```

```
modelo.add(modelo.presence_of(t)  
== True)
```



`modelo.presence_of(t) == True`

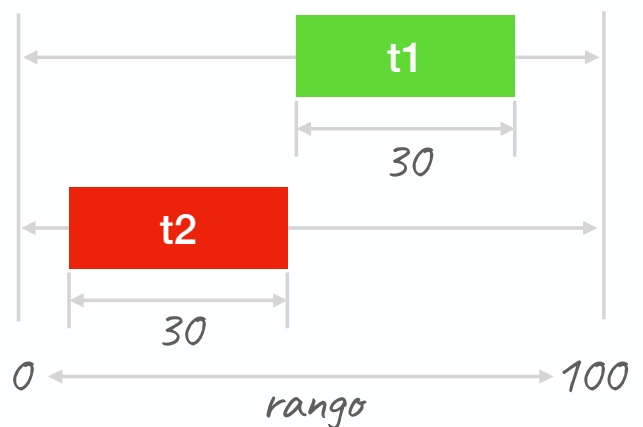
`modelo.presence_of(t) == False`

## activación

```
t1 = modelo.interval_var(  
    optional = True,  
    start = (0,100),  
    end = (0,100),  
    size = 30)
```

```
t2 = modelo.interval_var(  
    optional = True,  
    start = (0,100),  
    end = (0,100),  
    size = 30)
```

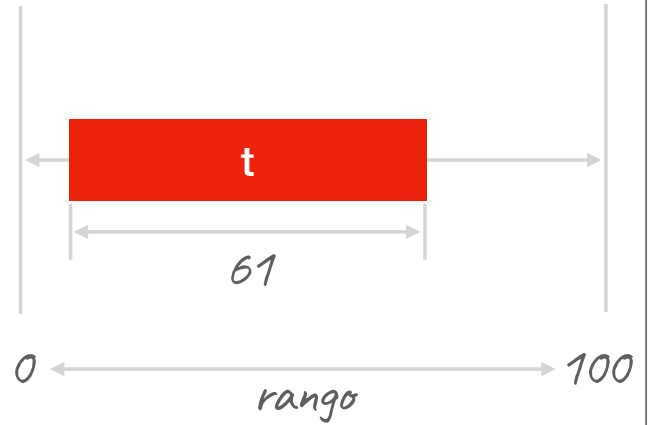
```
modelo.add(modelo.presence_of(t1)  
+ modelo.presence_of(t2) == 1)
```



```
t = modelo.interval_var(  
    start = (0,100),  
    end = (0,100),  
    size = 61,  
    name = 't')
```

```
modelo.add(modelo.start_of(t)==30)
```

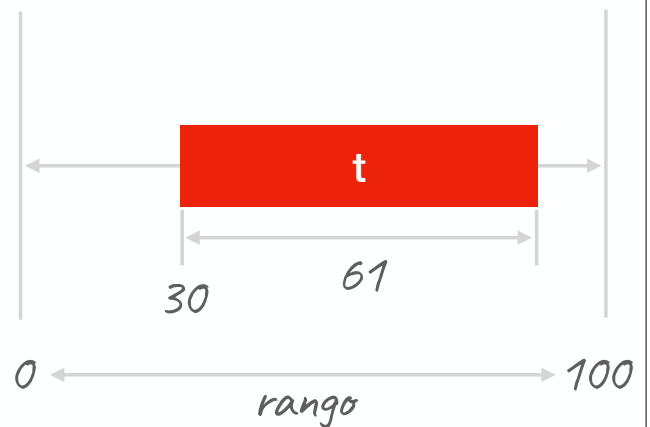
*start\_of*



```
t = modelo.interval_var(  
    start = (0,100),  
    end = (0,100),  
    size = 61,  
    name = 't')
```

```
modelo.add(modelo.start_of(t)==30)
```

*start\_of*



```
t = modelo.interval_var(  
    start = (0,100),  
    end = (0,100),  
    size = 61,  
    name = 't')
```

```
modelo.add(modelo.end_of(t)==100)
```

*end\_of*



```
t = modelo.interval_var(  
    start = (0,100),  
    end = (0,100),  
    size = 61,  
    name = 't')
```

```
modelo.add(modelo.end_of(t)==100)
```

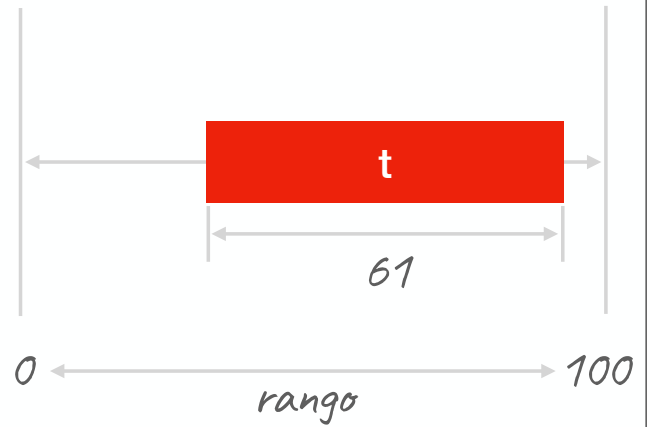
*end\_of*



```
t = modelo.interval_var(
    start = (0,100),
    end = (0,100),
size = 61,
    name = 't')
```

```
modelo.add(modelo.size_of(t)==30)
```

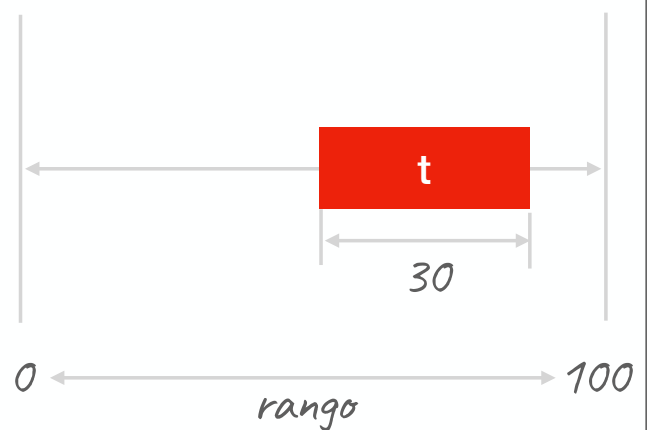
*size\_of*



```
t = modelo.interval_var(
    start = (0,100),
    end = (0,100),
    name = 't')
```

```
modelo.add(modelo.size_of(t)==30)
```

*size\_of*



```

t = modelo.interval_var(
    start = (0,100),
    end = (0,100),
    name = 't')

modelo.add(
    modelo.length_of(t)==30)

```

## length\_of



por ahora length = size  
pero más adelante  
veremos que no siempre  
es así

```

t = modelo.interval_var(
    optional, start,
    end, size, name)

modelo.start_of(t, x)
modelo.end_of(t, x)
modelo.size_of(t, x)
modelo.length_of(t, x)
modelo.presence_of(t)

```

## resumen

x es un valor optativo y  
si se utiliza será el  
valor que devuelva si el  
intervalo no esta  
presente

DEMO 4

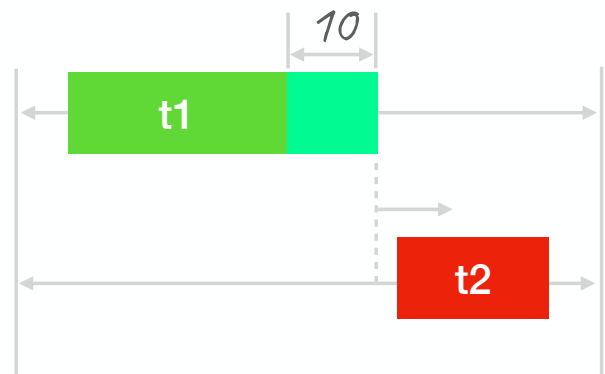
## relaciones de precedencia ...

```
modelo.add(modelo.end_before_start  
(t1, t2, 10))
```

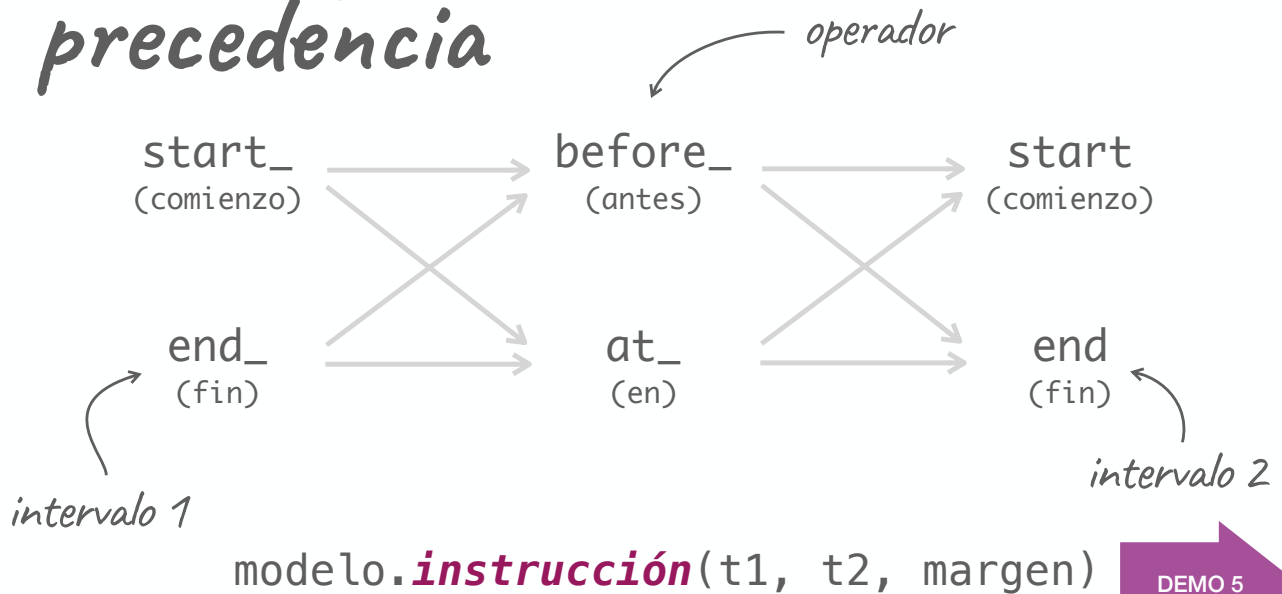
este parámetro  
es optativo

```
modelo.add(modelo.start_of(t2) >=  
modelo.end_of(t1) + 10)
```

## relaciones de precedencia

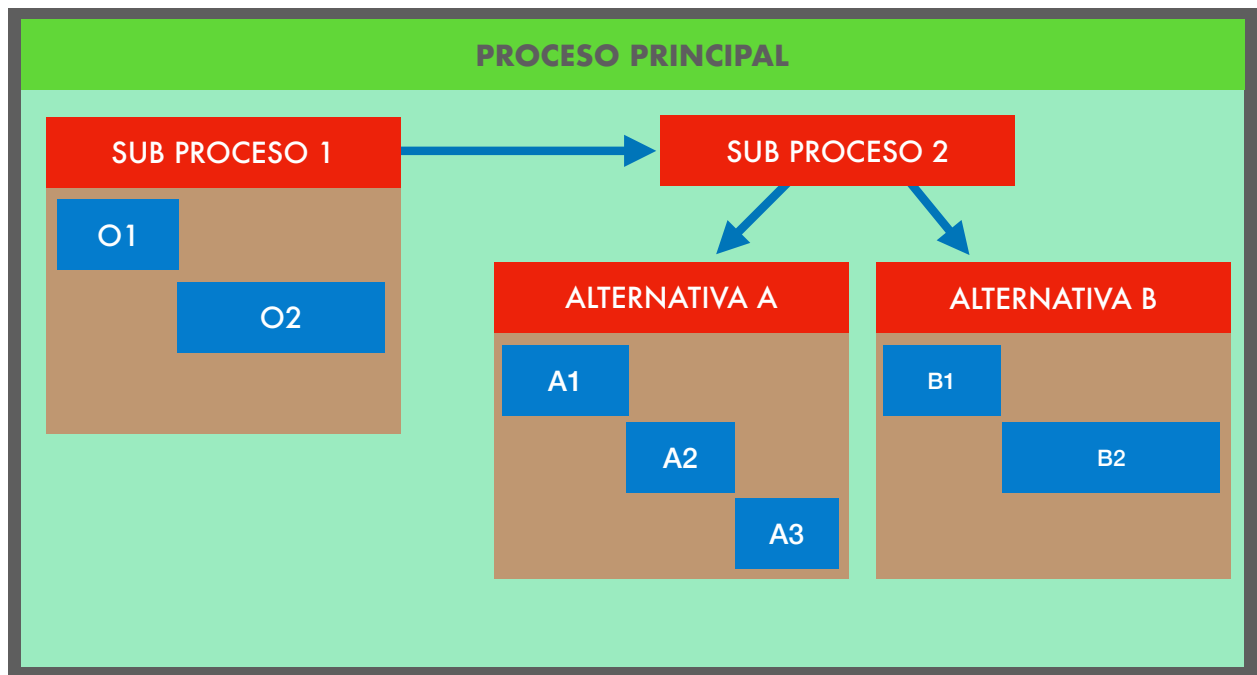


## relaciones de precedencia



relaciones de jerarquía ...





*span (incluir)*

$\swarrow$  actividad "padre"  
 modelo.*span*(ppal, [operaciones])

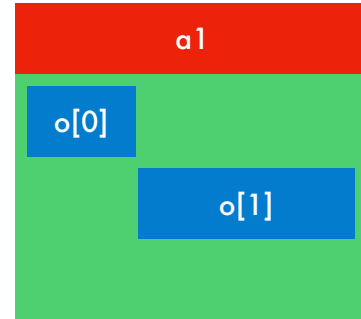


$\uparrow$   
 "array" de intervalos

si intervalo1 no esta  
 activo, tampoco lo  
 estarán los intervalos  
 hijos (o1 y o2)

## span (incluir)

```
a1 = modelo.interval_var  
(name = 'a1')  
  
o = [modelo.interval_var  
(name='op'+str(i)) for i in  
range(2)]  
  
o[0].set_size(12)  
o[1].set_size(40)  
  
modelo.add(modelo.end_before_start  
(o[0], o[1]))  
  
modelo.add(  
    modelo.span(a1, o)  
)
```

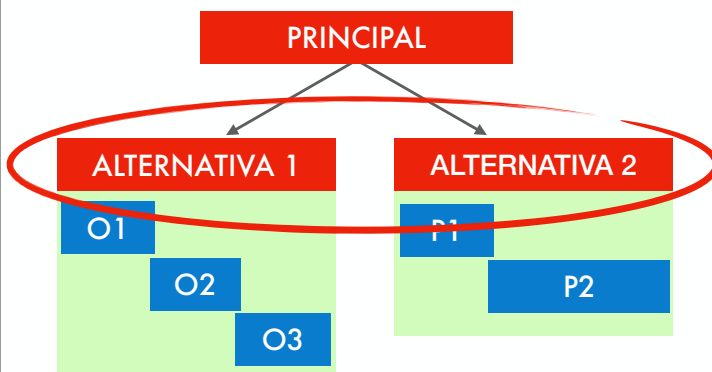


relaciones de  
precedencia y  
configuración

## alternativas

cantidad de alternativas  
aceptadas

modelo.**alternative**(ppal, [alter], **x**)



"array" de  
intervalos  
alternativos  
opcionales

si "principal" no esta  
activa, tampoco lo  
estarán las actividades  
alternativas

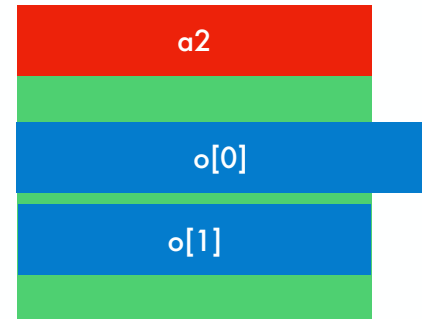
```
d = [12, 10]
```

```
a2 = modelo.interval_var(  
    name = 'a2')
```

```
o = [modelo.interval_var(  
    optional = True,  
    size = d[i],  
    name = 'o'+str(i))  
    for i in range(2)]
```

```
modelo.add(  
    modelo.alternative(a2, o)  
)
```

## alternativas



DEMO 6

*docplex tiene su  
propio visualizador ...*

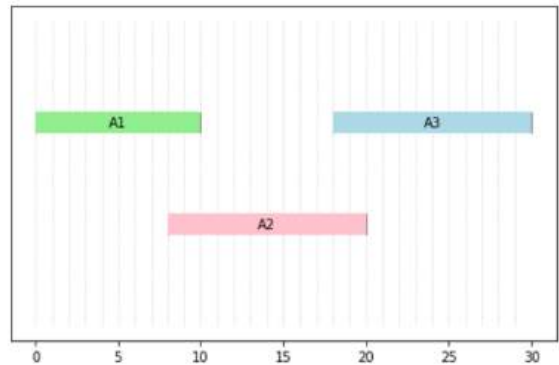
# visu diagramas de gantt

```
import docplex.cp.utils_visu as visu

visu.interval(0, 10, 'lightgreen', 'A1')
visu.interval(8, 20, 'pink', 'A2')
visu.interval(18, 30, 'lightblue', 'A3')

visu.show()
```

esto se puede  
reemplazar por las  
variables tipo  
intervalo



DEMO 7

floralwhite	turquoise	darkslateblue	white
oldlace	aquamarine	slateblue	whitesmoke
wheat	mediumaquamarine	ghostwhite	gainsboro
orange	mediumspringgreen	lavender	lightgray
moccasin	mintcream	midnightblue	lightgray
papayawhip	springgreen	blue	silver
blanchedalmond	mediumseagreen	mediumblue	darkgray
navajowhite	seagreen	darkblue	darkgray
antiquewhite	honeydew	navy	gray
tan	palegreen	royalblue	gray
burlywood	lightgreen	cornflowerblue	dimgrey
darkorange	limegreen	lightsteelblue	dimgrey
bisque	forestgreen	lightslateblue	black
linen	lime	lightslategray	lightpink
peru	green	slategray	pink
peachpuff	darkgreen	slategray	crimson
sandybrown	darkseagreen	dodgerblue	palevioletred
chocolate	lawngreen	aliceblue	lavenderblush
saddlebrown	chartreuse	steelblue	hotpink
seashell	greenyellow	lightskyblue	deeppink
sienna	darkolivegreen	skyblue	mediumvioletred
lightsalmon	yellowgreen	deepskyblue	orchid
orange	olivedrab	lightblue	magenta
coral	ivory	powderblue	fuchsia
darksalmon	lightyellow	cadetblue	violet
tomato	yellow	darkturquoise	plum
salmon	lightgoldenrodyellow	azure	thistle
mistyrose	beige	lightcyan	darkmagenta
snow	olive	paleturquoise	purple
red	darkkhaki	darkslategray	mediumorchid
lightcoral	palegoldenrod	darkslategray	darkviolet
indianred	lemonchiffon	cyan	darkorchid
rosybrown	khaki	aqua	indigo
firebrick	gold	darkcyan	blueviolet
brown	cornsilk	teal	rebeccapurple
darkred	goldenrod	mediumturquoise	mediumpurple
maroon	darkgoldenrod	lightseagreen	mediumslateblue

# *bibliografía y otros ...*

## **[Python] IBM CPO**

<https://www.ibm.com/products/ilog-cplex-optimization-studio/cplex-cp-optimizer>

<https://ibmdecisionoptimization.github.io/docplex-doc/cp/docplex.cp.modeler.py.html>

## **[Investigación Operativa 3] VIDEOS**



*clase de hoy  
pero con optimization  
studio*

*próxima clase pero  
en el video es con  
optimization studio*



*próxima clase:*

## *clase presencial*

*más de scheduling y presentación de  
resultados e informes*

# INVESTIGACIÓN OPERATIVA SUPERIOR

*¡muchas gracias!*

*[intervalo]*