MATI. Práctica 0

Introducción a Sage para grafos

Sage es un programa libre (open-source) de manipulación matemática.

Permite realizar las operaciones matemáticas más comunes (y muchas no tan comunes) como: integración (simbolica y numérica), Taylor, Gradiente y Hessiano, interpolación. Operaciones con matrices, etc. También admite programar para obtener más operaciones.

Para instrucciones, manuales y descargas se puede visitar http://www.sagemath.org (http://www.sagemath.org)

Para el uso de Sage en grafos podemos consultar el manual
http://www.sagemath.org/pdf/en/reference/graphs/graphs.pdf
(http://www.sagemath.org/pdf/en/reference/graphs/graphs.pdf)

En Sage un grafo no dirigido es un objeto de clase **Graph** y un grafo dirigido, de clase **DiGraph**. Podemos declarar un grafo que denominaremos G y un digrafo que denominaremos DG con las instrucciones:

In [6]:

```
G = Graph(); DG=DiGraph()  # grafos vacios, sin ningún vértice
```

In [ ]:

```

```

A partir del objeto G de tipo Graph, podemos obtener rápidamente un listado de métodos y atributos asociados a cualquier grafo sin más que teclear "G." y pulsar la tecla del tabulador . De forma similar para "DG. + ". La mayoría de los métodos y atributos son comunes para grafos y digrafos.

In [7]:

```
G.
```

```
  File "<ipython-input-7-4f4ee6e0d2ca>", line 1
    G.
      ^
SyntaxError: invalid syntax
```

In [ ]:

```
DG.
```

Existe una amplia base de datos de grafos y familias de grafos conocidos a la que podemos acceder sin más que teclear "**graphs**. + "

In [ ]:

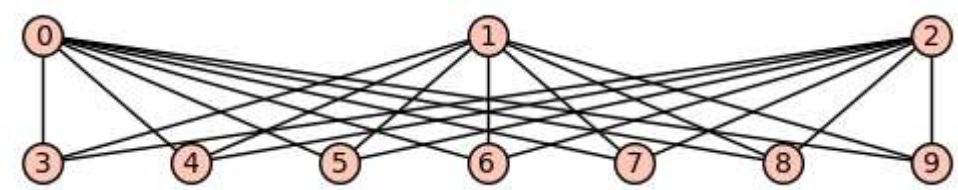
```
graphs.
```

Por ejemplo, vamos a asignar a la variable G el grafo bipartito completo $K_{3,7}$ y lo representamos gráficamente:

In [9]:

```
G = graphs.CompleteBipartiteGraph(3,7)
G.plot(figsize=5)
```

Out[9]:



Se pueden definir grafos propios siguiendo múltiples formatos. Para más información, ejecutar **Graph?**

Una de las formas más simples e intuitivas de definir un grafo es mediante un diccionario que describa las listas de adyacencias, donde las claves (keys) son los números asociados a los vértices y los valores (values) son la lista de vértices adyacentes. No es preciso añadir los recíprocos en las listas de adyacencias, ya que, según se trate de un grafo o un digrafo, se tomarán las recíprocas automáticamente o no.

In [10]:

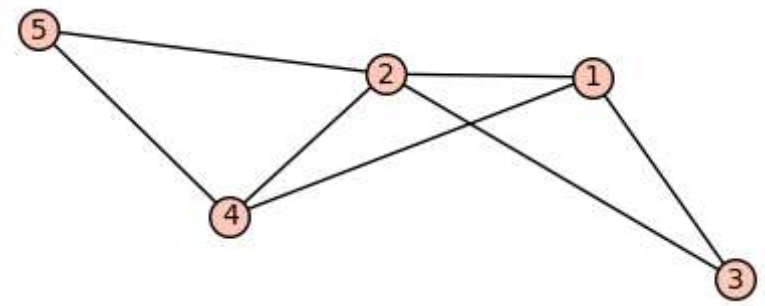
```
G1= Graph({1:[2,3,4],2:[3,4,5],4:[5]})
```

save_pos= True es una opción para fijar las posiciones de los vértices. Si no lo ponemos casa vez nos lo pinta de una manera.

In [11]:

```
G1.plot(figsize = 4,save_pos = True)
```

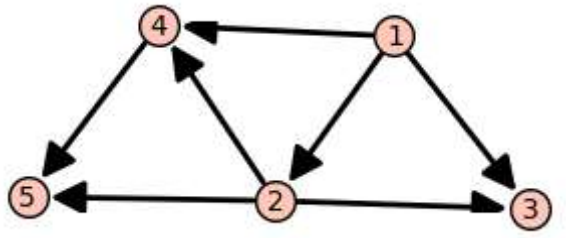
Out[11]:



In [12]:

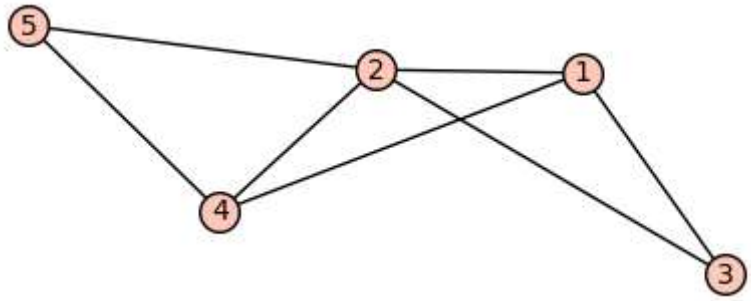
```
DG1=DiGraph({1:[2,3,4],2:[3,4,5],4:[5]})
plot(DG1,figsize = 3)
```

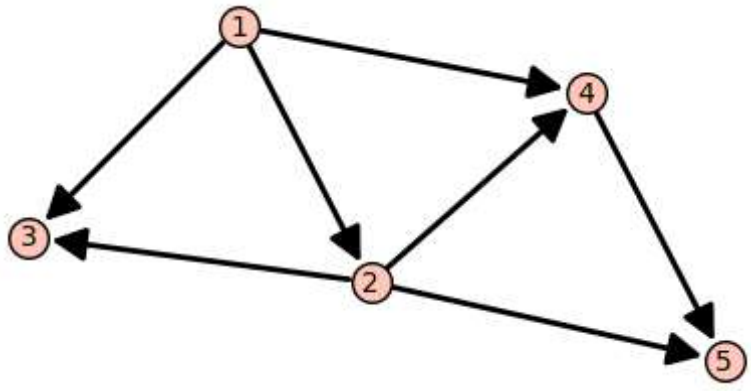
Out[12]:



In [13]:

```
G1.show()
DG1.show()
```





Con la siguiente instrucción creamos un grafo con el número de vértices que queramos. Luego le añadiremos aristas.

In [14]:

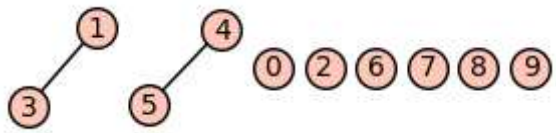
```
G2=Graph(10)
plot(G2, figsize=3)
```

Out[14]:

In [15]:

```
G2.add_edges({(1,3),(4,5)})
plot(G2, figsize=3)
```

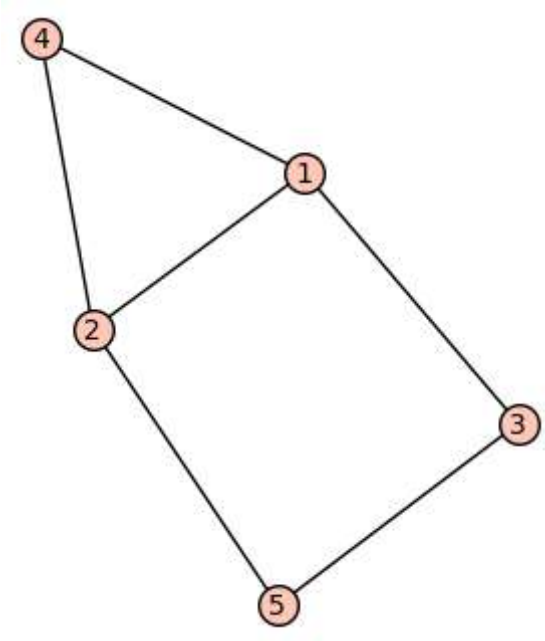
Out[15]:



Podríamos haberlo definido también a partir de la lista de sus aristas

In [16]:

```
G3=Graph()
l=[(1,2),(3,1),(1,4),(2,5),(2,4),(3,5)]
for a in l:
  G3.add_edge(a)
G3.show(figsize=5)
```



O a través de su matriz de adyacencia, para ello primero definimos M como un objeto "matriz"

In [17]:

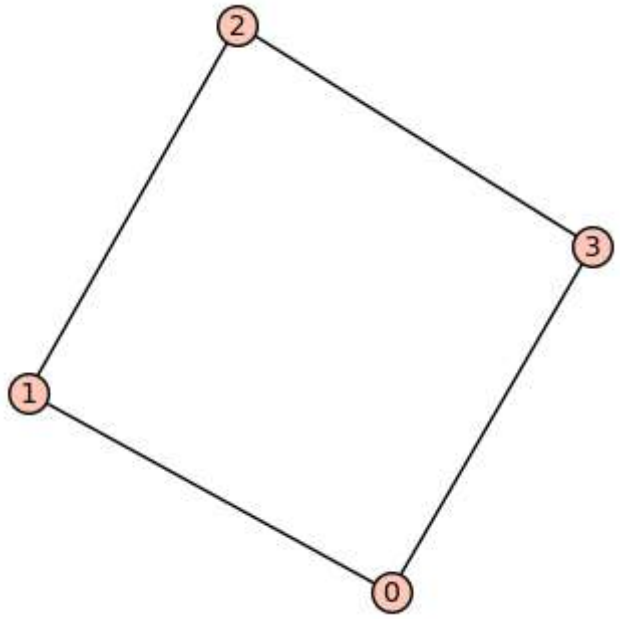
```
M=matrix([[0, 1, 0, 1],[1, 0, 1, 0],[0, 1, 0, 1],[1, 0, 1, 0]]);M
```

Out[17]:

```
[0 1 0 1]
[1 0 1 0]
[0 1 0 1]
[1 0 1 0]
```

In [18]:

```
G4=Graph(M);G4.show(figsize=5)
```



Una vez definido el grafo podemos pedir a sage múltiples caracteríticas y propiedades como la matriz de adyacencia:

In [19]:

```
print('la matriz de adyacencia de G1:')
print(G1.adjacency_matrix())
```

```
la matriz de adyacencia de G1:
[0 1 1 1 0]
[1 0 1 1 1]
[1 1 0 0 0]
[1 1 0 0 1]
[0 1 0 1 0]
```

la lista de grados:

In [20]:

```
G1.degree()
```

Out[20]:

```
[3, 4, 2, 3, 2]
```

In [21]:

```
DG1.in_degree()
```

Out[21]:

```
[0, 1, 2, 2, 2]
```

In [22]:

```
DG1.out_degree()
```

Out[22]:

```
[3, 3, 0, 1, 0]
```

los vecinos de un vértice

In [23]:

```
G.neighbors(2)
```

Out[23]:

```
[3, 4, 5, 6, 7, 8, 9]
```

número de aristas, etc.

In [24]:

```
G2.num_edges()
```

Out[24]:

```
2
```

Editor de grafos

El comando ***graph_editor*** no funciona en *Jupyter*

In [25]:

```
graph_editor();
```

```
---------------------------------------------------------------------------
ModuleNotFoundError                       Traceback (most recent call last)
<ipython-input-25-c8a1f676c54f> in <module>()
----> 1 graph_editor();

/opt/sagemath-9.1/local/lib/python3.7/site-packages/sage/misc/lazy_import.py
x in sage.misc.lazy_import.LazyImport.__call__ (build/cythonized/sage/misc/l
azy_import.c:3686)()
    351             True
    352         """
--> 353         return self.get_object()(*args, **kwds)
    354
    355     def __repr__(self):

/opt/sagemath-9.1/local/lib/python3.7/site-packages/sage/graphs/graph_edito
r.py in graph_editor(graph, graph_name, replace_input, **layout_options)
    106         sage: graph_editor(h, replace_input=False)  # not tested
    107     """
--> 108     import sagenb.notebook.interact
    109     if graph is None:
    110         graph = graphs.CompleteGraph(2)

ModuleNotFoundError: No module named 'sagenb'
```

~~También podemos editar un grafo que tenemos definido, (claro que no veremos las etiquetas).~~

In [ ]:

```
graph_editor(G);
```

Representando grafos

Los siguientes ejemplos muestran cómo cambiar el color de los vértices y su tamaño o el de las aristas. Para ello se crea un diccionario (una lista entre llaves) de modo que cada entrada de la lista sea una pareja separada por :, el elemento de la izquierda será el color y el de la derecha la lista de elementos a dibujar con ese color.

In [ ]:

```
P = graphs.PetersenGraph()
d = {'#FF0000':[0,5], '#FF9900':[1,6], '#FFFF00':[2,7], '#00FF00':[3,8], '#0000FF':[4,9]}
P.plot(vertex_colors=d,vertex_size=500).show(figsize=4)
```

¿Cómo quedará este grafo?. Dada una cadena de texto, se genera un grafo que une cada letra con la siguiente en el texto. Se incluyen opciones gráficas para etiquetar los vértices con las propias letras y se etiquetan y colorean las aristas de forma que todas las aristas con la misma etiqueta llevan el mismo color:

In [ ]:

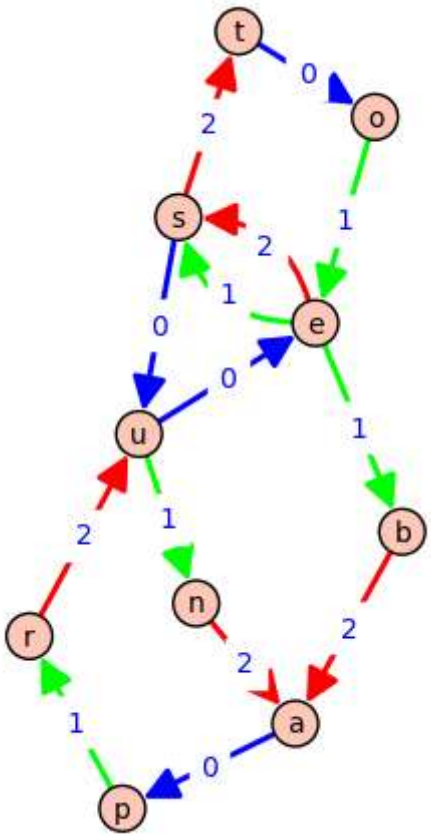
```
texto = 'estoesunaprueba'
g = Graph({}, loops=True, multiedges=True)
m=0
for i,j in [(texto[k], texto[k+1]) for k in range(len(texto)-1)]:
    m=m+1
    g.add_edge(i, j, mod(m,3))
g.plot(color_by_label=True,edge_labels=True, edge_style='solid').show(figsize=(7))
```

Grafo dirigido y/o etiquetado

Todo lo anterior es válido para grafos dirigidos, sin más que cambiar ***Graph*** por ***DiGraph***.

In [26]:

```
texto = 'estoesunaprueba'
g = DiGraph({}, loops=True, multiedges=True)
m=0
for i,j in [(texto[k], texto[k+1]) for k in range(len(texto)-1)]:
    m=m+1
    g.add_edge(i, j, mod(m,3))
g.plot(color_by_label=True,edge_labels=True, edge_style='solid').show(figsize=(7))
```



Generador de grafos libre de isomorfismos

Con la función graphs no solo accedemos a la base de datos de grafos sino que también disponemos de un potente "generador de grafos libre de isomorfismos". Ejecutar graphs? para más información. Y observa los siguientes ejemplos.

In [27]:

```
graphs?
```

Veamos algunos ejemplos de utilización:

Lista de todos los grafos (libre de isoformismos) con 5 vértices y 7 aristas:

In [28]:

```
grafos_5v7a = [gr for gr in graphs(5, size=7)]
print('nùmero de grafos = ',len(grafos_5v7a))
grafos_5v7a
```

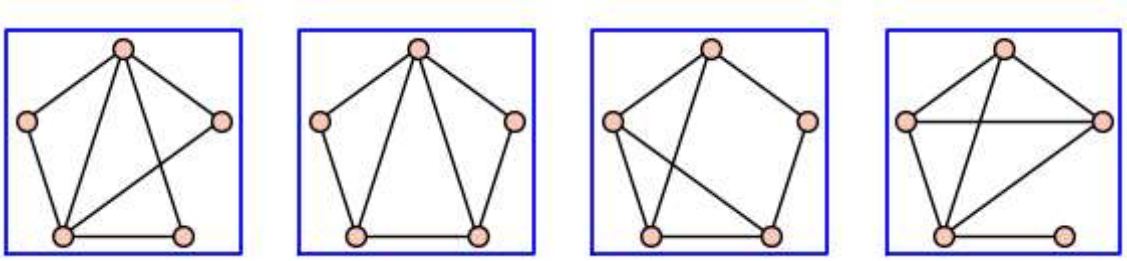
```
nùmero de grafos =  4
```

Out[28]:

```
[Graph on 5 vertices,
 Graph on 5 vertices,
 Graph on 5 vertices,
 Graph on 5 vertices]
```

In [29]:

```
graphs_list.show_graphs(grafos_5v7a)
```



Type *Markdown* and LaTeX: α^2

Cuestionario acerca de nociones básicas de Teoría de Grafos (Parte I)

(Hacer solo los ejercicios 1, 4, 5, 6, 7, 8, 9 y 10)

Ahora prácticaremos con algunos de los ejercicios de los cuestionarios e intentaremos ver si los sabemos resolver con SAGE.

Ejercicio 1.

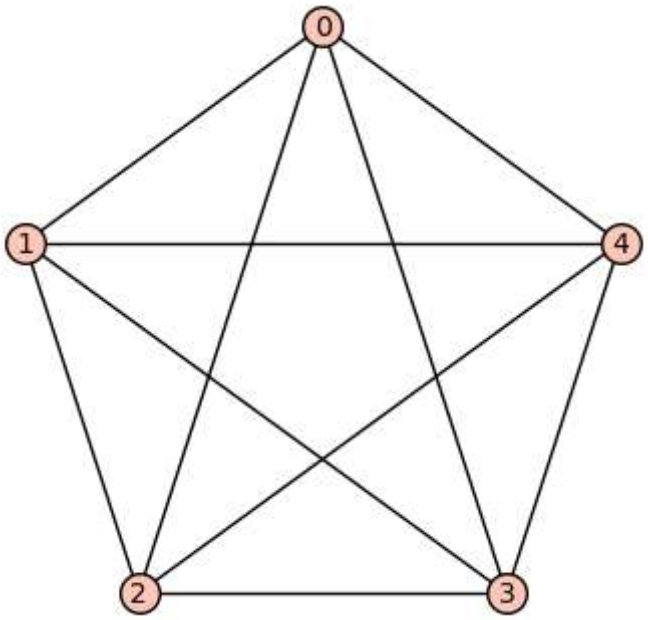
In [33]:

```
K5=graphs.CompleteGraph(5)
```

In [36]:

```
K5.plot(figsize=5)
```

Out[36]:



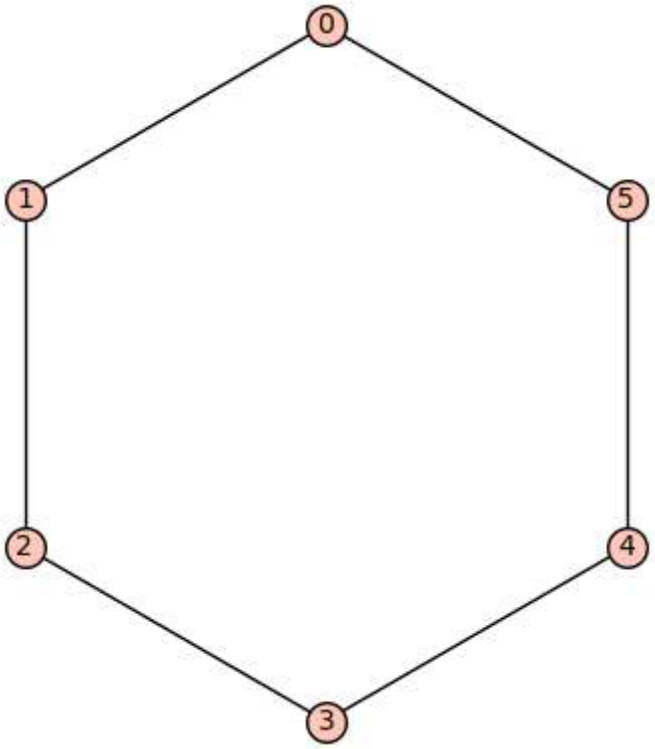
In [38]:

```
C6=graphs.CycleGraph(6)
```

In [39]:

```
C6.plot()
```

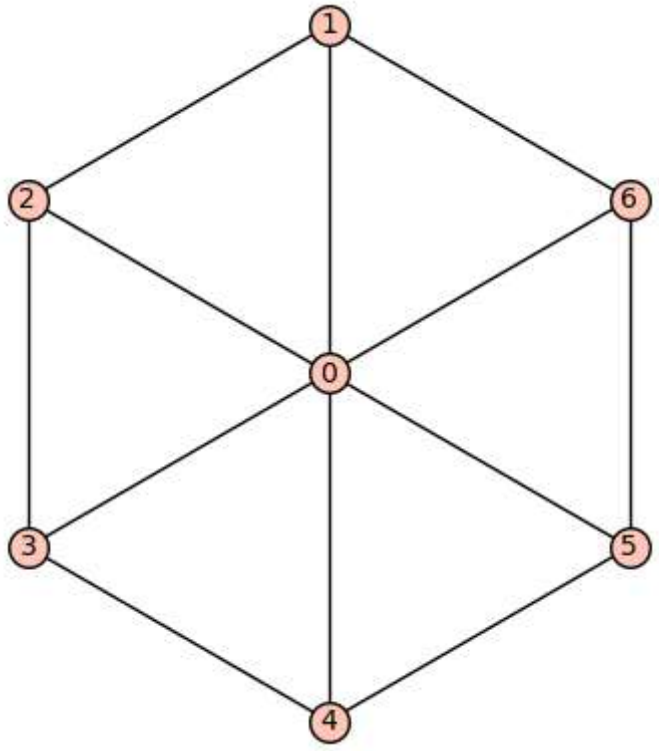
Out[39]:



In [42]:

```
W7=graphs.WheelGraph(7)
W7.plot()
```

Out[42]:



In [ ]:

```

```

Ejercicio 2. (Para las matrices que sean de adyacencia dibujad sus grafos.)

In [43]:

```
G.degree_sequence()
```

Out[43]:

```
[7, 7, 7, 3, 3, 3, 3, 3, 3, 3]
```

In [44]:

```
is_degree_sequence([7, 7, 7, 3, 3, 3, 3, 3, 3, 3])
```

```
---------------------------------------------------------------------------
NameError                                 Traceback (most recent call last)
<ipython-input-44-b7f47bee2aab> in <module>()
----> 1 is_degree_sequence([Integer(7), Integer(7), Integer(7), Integer(3),
 Integer(3), Integer(3), Integer(3), Integer(3), Integer(3), Integer(3)])

NameError: name 'is_degree_sequence' is not defined
```

Ejercicio 3. (Tendremos que crear una función, pues me lo pide para n)

In [ ]:

```
```

In [ ]:

```
```

Ejercicio 5. (Utiliza el generador de grafos)

In [53]:

```
graphs?
```

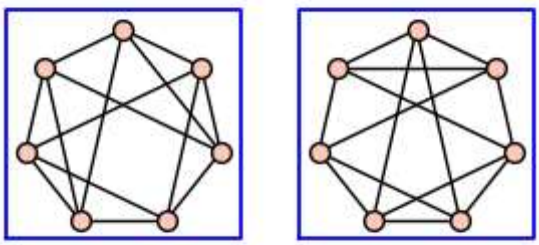
In [58]:

```
grafos_7v14a = [gr for gr in graphs(7, size=14) if gr.degree()==[4, 4, 4, 4, 4, 4, 4]]
print('nùmero de grafos = ',len(grafos_7v14a))
```

```
nùmero de grafos =  2
```

In [59]:

```
graphs_list.show_graphs(grafos_7v14a)
```



Ejercicio 6. (Utiliza el método o función adecuados para buscar subgrafos).

In [80]:

```
K47=graphs.CompleteBipartiteGraph(4,7)
C5=graphs.CycleGraph(5)

k47.subgraph_search_count(C5)
```

Out[80]:

```
0
```

In [79]:

```

```

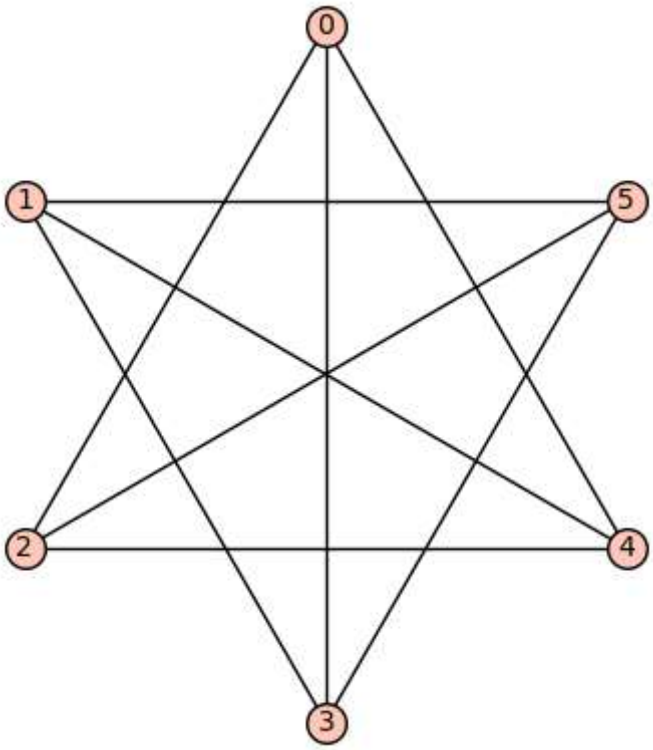
Out[79]:

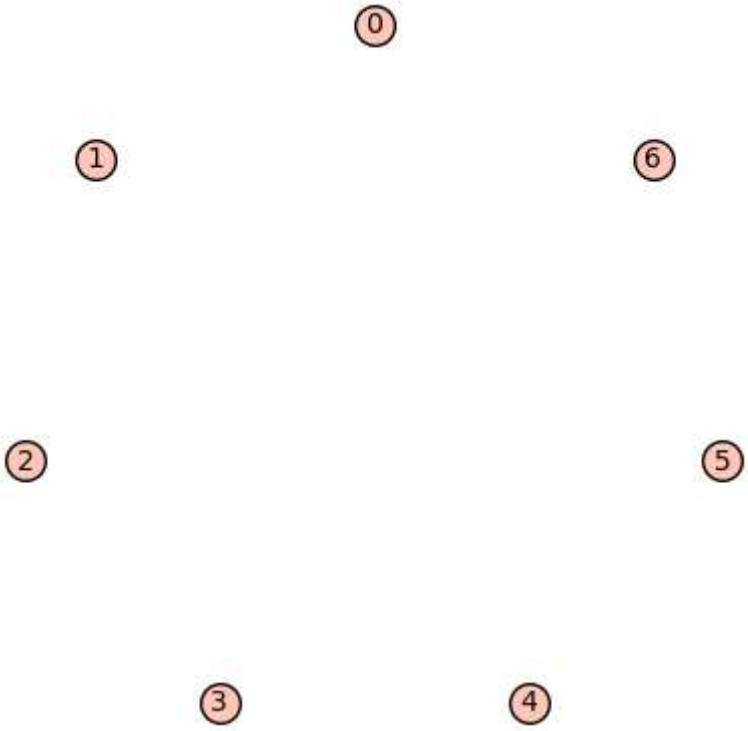
```
Subgraph of (Subgraph of (Complete bipartite graph of order 4+7)): Graph on
0 vertices (use the .plot() method to plot)
```

Ejercicio 7.

In [115]:

```
C6=graphs.CycleGraph(6)
C6.complement().show()
K7=graphs.CompleteGraph(7)
K7.complement().show()
```





Ejercicio 8. (Busca una orden de SAGE que te evalue si dos grafos son isomorfos y usala para crear tú una función que para cada grafo te diga si es o no autocomplementario.)

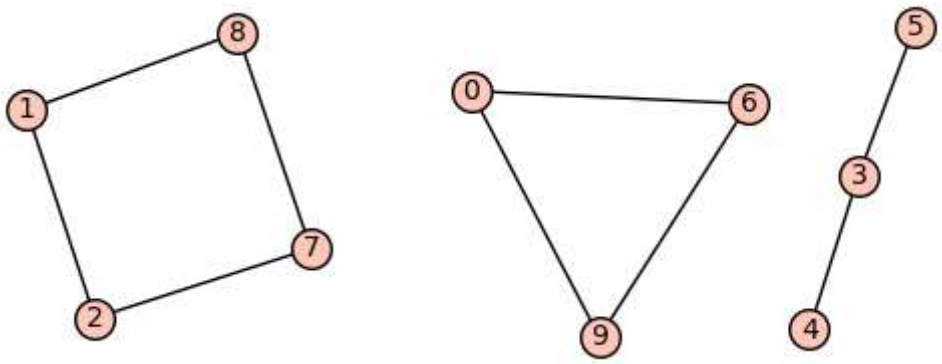
In [103]:

```
def is_auto_comp(g):
    return g.is_isomorphic(g.complement())
```

Ejercicio 9.

In [104]:

```
A=Graph()
l=[(0, 6), (1, 2), (1, 8), (0, 9), (2, 7), (3, 4), (3, 5), (6, 9), (7, 8)]
for n in l:
  A.add_edge(n)
A.show(figsize=5)
CC= A.connected_components_number()
```



Ejercicio 10.

In [110]:

```
g=Graph({1:[3,4,7,8],2:[5,6,9,10], 3:[4,11], 4:[11], 5:[6], 6:[11], 7:[8,12], 8:[12], 9:[10
g.diameter()
```

Out[110]:

4

Para entregar EN ENSEÑANZA VIRTUAL

antes del miércoles 21 octubre a las 13:30 GMT

Hay que resolver el examen y entregar la actividad

Incumplir las instrucciones significará la no evaluación de la práctica

El número x son las dos últimas cifras de tu DNI sumadas a 50. Incluye el número de tu DNI en el fichero PDF

Ejercicio:

(Un pasatiempo del número 6 de “Le Monde 2”)
¿Cuál es el tamaño del mayor conjunto $S \subseteq [0, \ldots, x]$ de forma que no contiene dos enteros i, j tal que $|i - j|$ es un cuadrado?
¿Cuál es el tamaño del mayor conjunto $T \subseteq [0, \ldots, x]$ de forma que dados dos enteros i, j de T, $|i - j|$ es un cuadrado?

In [ ]:

```
Creamos los conjuntos S y T, compuestos de los enteros de 0 a 102.
Buscaremos que siendo i,j un entero del conjunto S, recogemos el elemento de la lista que s
|i-j| no sea cuadrática, y lo contrario para el conjunto T.
```

In [288]:

```
C=[] #Conjunto [0.....x]
S=[] #Conjunto S
for a in range(50+53):
    C.append(a)
S.append(C[0])
valido=false
for i in range(1,len(C)-1):
    valido=true
    #print(i)
    for j in S:
        #print('\n|',i,'-',j,'|',j,' es ',sqrt(abs(i-j)), ", es entero: ",sqrt(abs(i-j)).is
        if sqrt(abs(i-j)).is_integer() == true :
            valido=false
    if valido :
        S.append(i)

print(S)
```

```
[0, 2, 5, 7, 10, 12, 15, 17, 20, 22, 34, 39, 44, 52, 57, 62, 65, 67, 72, 85,
95]
```

In [289]:

```
len(S) #Tamaño del Conjunto S
```

Out[289]:

```
21
```

In [302]:

```
C=[] #Conjunto [0.....x]
T=[] #Conjunto T
for a in range(50+53):
    C.append(a)
T.append(C[0])
valido=false
for i in range(1,len(C)-1):
    valido=true
    #print(i)
    for j in T:
        #print('\n|',i,'-',j,'|',j,' es ',sqrt(abs(i-j)), ", es entero: ",sqrt(abs(i-j)).is
        if sqrt(abs(i-j)).is_integer() == false :
            valido=false
    if valido :
        T.append(i)

print(T)
```

```
[0, 1]
```

In [303]:

```
len(T) #Tamaño del Conjunto T
```

Out[303]:

```
2
```

In [ ]:

In [ ]:

In [ ]:

In [ ]:

In [ ]:

In [ ]:

In [ ]:

In [ ]:

In [ ]:

In [ ]:

In [ ]: