😎 Grafos con NetworkX

NetworkX es un paquete de Python para la creación, manipulación y estudio de estructuras dinámicas y funciones de redes complejas.

Los grafos se utilizan para modelar situaciones, es una representación simplificada de la situación. Se aplican en Informática, Ciencias Sociales, Lingüística, Arquitectura, Comunicaciones, Física, Química, Ingeniería, etc.

Definición informal: conjunto de nodos unidos por aristas.

Definición formal: Un grafo es una terna **G = (V , A , Φ)**, dónde:

V: Conjuntos de vértices, dónde **V ≠ ∅**

A: Conjuntos de aristas.

Φ: Función de incidencia **Φ: A → $V^{(2)}$** Y **$V^{(2)}$** es el conjunto formado por subconjuntos de 1 ó 2 elementos de V, que son los extremos de la arista.

Ejemplo:

Sea el grafo **G = (V , A , Φ)**, siendo los conjuntos:

- **V = {v_1, v_2, v_3, v_4, v_5}**
- **A = {a_1, a_2, a_3, a_4, a_5}**

Y la función de incidencia:

Φ(a_1) = {v_1,v_2},

Φ(a_2) = {v_3},

Φ(a_3) = {v_4,v_2},

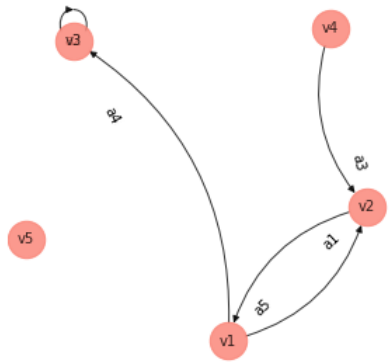
Φ(a_4) = {v_1,v_3},

Φ(a_5) = {v_2,v_1}

In [1]:
```
import networkx as nx
import matplotlib.pyplot as plt
import pandas as pd
import numpy as np
import warnings
warnings.filterwarnings('ignore')
```

In [2]:
```
fig, ax = plt.subplots(figsize=(5,5))

DG = nx.DiGraph([('v1','v2', {'label':'a1'}),('v3','v3', {'label':'a2'}), ('v4','v2',{'label':'a3'}),
                 ('v1','v3', {'label':'a4'}),('v2','v1', {'label':'a5'})])
DG.add_node('v5')
pos = nx.spring_layout(DG,k=4,scale=3.0)
labels = nx.get_edge_attributes(DG,'label')
nx.draw(DG, pos,  with_labels=True, connectionstyle="arc3,rad=0.3",
        node_color="salmon", alpha=0.8, font_size=10, width=1, node_size=700)
nx.draw_networkx_edge_labels(DG, pos, edge_labels=labels, font_size=10, label_pos=0.25)
plt.show()
```



Vértice aislado: v_5 es aislado, $v_i \Leftrightarrow \forall\ v_k \in V$: Si $v_i \neq v_k$: v_i no es adyacente a v_k. Significa que es un vértice que no es adyacente a ningún otro.

Aristas paralelas: a_1 y a_5 son paralelas, a_1 es paralela a $a_k \Leftrightarrow Φ\ (a_i) = Φ\ (a_k)$. Significa que son aristas comprendidas entre los mismos vértices.

Bucles o lazos: a_2 es bucle o lazo, a_i es bucle o lazo $\Leftrightarrow |\ Φ\ (a_i)\ | = 1$. Significa que es una arista con ambos extremos en el mismo vértice.

Crear un grafo

La clase (Graph) implementa un grafo no dirigido. Pueden generarse diferentes tipos de grafos con draw(), draw_circular, draw_espectral(), draw_shell(), etc.

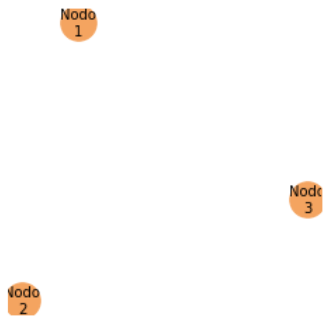
Diferencia entre draw() y draw_networkx()

- **draw()** : dibuja el gráfico como una representación simple sin etiquetas de nodo o de borde y utilizando el área de figura completa de Matplotlib sin etiquetas de eje de forma predeterminada.
- **draw_networkx()** : dibuja el gráfico con Matplotlib con opciones para posiciones de nodos, etiquetado, títulos y muchas otras características de dibujo.

Nodos

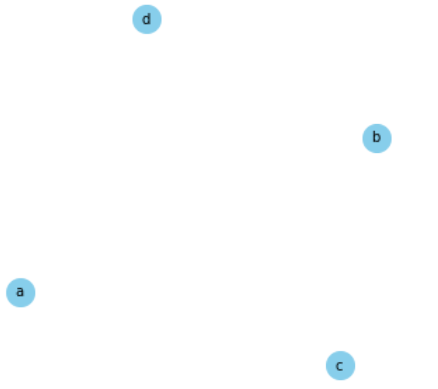
Agregar nodos con **add_node()** :

In [3]:
```
fig, ax = plt.subplots(figsize=(4,4))
G = nx.Graph()
G.add_node('Nodo\n1')
G.add_node('Nodo\n2')
G.add_node('Nodo\n3')
nx.draw_networkx(G, with_labels=True, node_size=650, font_size=10, node_color = 'sandybrown', ax=ax)
plt.axis('off')
plt.show()
```



Con **add_nodes_from()** : *se agregan nodos desde una lista.*

In [4]:
```
fig, ax = plt.subplots(figsize=(5,5))
L1 = ['a','b','c','d',]
G = nx.Graph()
G.add_nodes_from(L1)
nx.draw_networkx(G, with_labels=True, node_size=400, font_size=10, node_color = 'skyblue', ax=ax)
plt.axis('off')
plt.show()
```



Eliminar nodos

Con **remove_nodes_from()** o **remove_node()** se eliminan nodos._

In [5]:
```
G.remove_node('a')
fig, ax = plt.subplots(figsize=(5,4))
nx.draw_networkx(G, with_labels=True, node_size=400, font_size=10, node_color = 'skyblue', ax=ax)
plt.axis('off')
plt.show()
```

In [6]:
```
G.remove_nodes_from('b')
fig, ax = plt.subplots(figsize=(5,5))
nx.draw_networkx(G, with_labels=True, node_size=400, font_size=10, node_color = 'skyblue', ax=ax)
plt.axis('off')
plt.show()
```

Atributos de los nodos

In [7]:
```
print(f"nodes(), devuelve los nodos: {G.nodes()}\n\nnumber_of_nodes(), la cantidad: {G.number_of_nodes()}")
```

```
nodes(), devuelve los nodos: ['c', 'd']

number_of_nodes(), la cantidad: 2
```

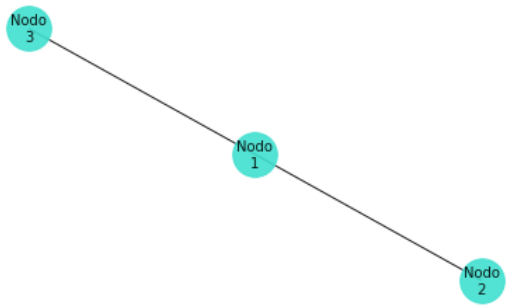
with_labels: *Visualiza las etiquetas del nodo.*
node_size: *Modifica el tamaño del nodo.*
node_color: *Cambia el color del nodo.*
alpha: *da transparencia.*
font_size: *Tamaño de la etiqueta.*
font_color: *Color de la fuente de la etiqueta.*
node_shape: *puede cambiarse la forma del nodo*

In [8]:
```
fig, ax = plt.subplots(figsize=(6,4))
nx.draw(G, node_color="salmon", with_labels=True, node_size=700, alpha=0.8, font_size=8,
        font_color='black', node_shape='^', ax=ax)
```

Aristas

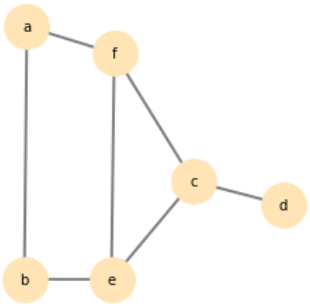
add_edge(): *enlaza nodos.*

In [9]:
```
fig, ax = plt.subplots(figsize=(7,4))
G = nx.Graph()
G.add_edge('Nodo\n1','Nodo\n2')
G.add_edge('Nodo\n3','Nodo\n1')
nx.draw(G, node_color="turquoise", with_labels=True, alpha=0.9, node_size=1000, font_size=10, ax=ax)
```



add_edges_from: *enlaza nodos desde una lista de tuplas*.

In [10]:
```
fig, ax = plt.subplots(figsize=(4,4))
L2 = [('a','b'),('a','f'),('b','e'),('c','e'),('c','d'),('e','f'),('f','c')]
G = nx.Graph()
G.add_edges_from(L2)
nx.draw(G, node_color="moccasin", font_size=10, width=2, with_labels=True, node_size=1000, edge_color='grey',ax=ax)
```



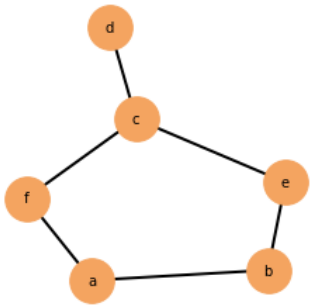
Ejercicio:

Crea un grafo simple, con el atributo node_shape elegido de la documentación de Networkx y considerando los siguientes enlaces: (1, 2), (1, 3), (1, 5), (2, 3), (3, 4), (4, 5)

Eliminar enlaces

remove_edge(): *elimina los enlaces.*

In [11]:
```
G.remove_edge('f','e')
fig, ax = plt.subplots(figsize=(4,4))
nx.draw(G, node_color="sandybrown", font_size=10, width=2, with_labels=True, node_size=1000)
```

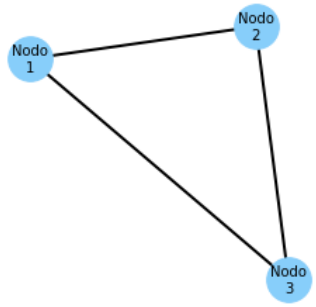


Atributos de los enlaces o aristas

Con **weight**: *se agrega peso a los enlaces.*

In [12]:

```
fig, ax = plt.subplots(figsize=(4,4))
G = nx.Graph()
G.add_edge('Nodo\n1','Nodo\n2', weight=5.0)
G.add_edge('Nodo\n2','Nodo\n3', weight=3.5)
G.add_edge('Nodo\n3','Nodo\n1', weight=1.5)
nx.draw(G, node_color="lightskyblue", font_size=10, width=2, with_labels=True,node_size=1000, ax=ax)
```



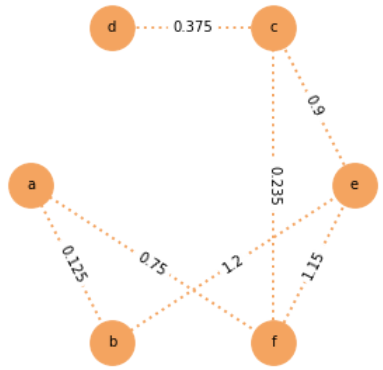
*Con **edge_color** se agrega color, con **style** se aplica un estilo. Con **edge_labels()** se pueden mostrar los pesos de los enlaces.*

In [13]:

```
fig, ax = plt.subplots(figsize=(5,5))

def emitoGraph(G, pos):
    nx.draw_networkx_nodes(G, pos, node_color = "sandybrown", node_size = 1000)
    nx.draw_networkx_labels(G, pos, font_size = 10, font_family = 'sans-serif')
    labels = nx.get_edge_attributes(G, 'weight')
    nx.draw_networkx_edges(G, pos, edge_color='sandybrown', width=2, arrowstyle= '<|-|>', arrowsize = 20,
                           style='dotted', ax=ax)
    nx.draw_networkx_edge_labels(G, pos, edge_labels = labels)

def cargoGraph(G):
    G.add_weighted_edges_from([('a','b',0.125),('a','f',0.75),('b','e',1.2),
                           ('c','e',0.90),('c','d',0.375),('e','f',1.15),('f','c',0.235)])
G = nx.Graph()
cargoGraph(G)
pos = nx.shell_layout(G)
emitoGraph(G, pos)
plt.axis('off')
plt.show()
```



Ejercicio:

Al grafo creado en el ejercicio anterior, agregar pesos a las aristas y emitirlo.

Consultar el grafo

In [14]:

```
print(f"Número de vértices: {G.order()}\nNúmero de vértices: {len(G)}")
print(f'Revisamos nodos: {G.nodes} \n')
print(f"Los enlaces son: {G.edges()}\n\nEl número de enlaces es:{G.number_of_edges()}")
print(f"\nneighbors() permite ver los nodos vecinos, en este caso de e: {list(G.neighbors('e'))}")
print("\nsize() devuelve el número de aristas", G.size())
```

```
Número de vértices: 6
Número de vértices: 6
Revisamos nodos: ['a', 'b', 'f', 'e', 'c', 'd']

Los enlaces son: [('a', 'b'), ('a', 'f'), ('b', 'e'), ('f', 'e'), ('f', 'c'), ('e', 'c'), ('c', 'd')]

El número de enlaces es:7

neighbors() permite ver los nodos vecinos, en este caso de e: ['b', 'c', 'f']

size() devuelve el número de aristas 7
```

Ejercicio:

Consultar el grafo creado emitiendo: los nodos, los enlaces, el número de vértices y enlaces.

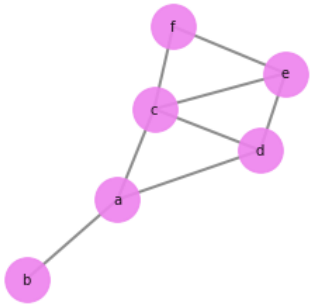
Funciones

In [15]:
```
# Excentricidad: distancia máxima de v a todos los otros vértices del grafo (por caminos mínimos)
print("Excentricidad:", nx.eccentricity(G))
print("Ruta más corta entre a y e: ", nx.algorithms.shortest_path(G, 'a', 'e'))
# Radio: mínimo número de excentricidad de todos los v en G
print("Radio:", nx.radius(G))
# Diámetro: máximo número de excentricidad de todos los v en G.
print("Diámetro:",nx.diameter(G))
# Centro: Conjunto de vértices de mínima excentricidad.
print("Centro:", nx.center(G))
# Periferia: Conjunto de vértices de máxima excentricidad.
print("Periferia:", nx.periphery(G))
```

```
Excentricidad: {'a': 3, 'b': 3, 'f': 2, 'e': 2, 'c': 2, 'd': 3}
Ruta más corta entre a y e:  ['a', 'b', 'e']
Radio: 2
Diámetro: 3
Centro: ['f', 'e', 'c']
Periferia: ['a', 'b', 'd']
```

Grafo Simple: G es simple si y sólo si **no** tiene aristas paralelas ni bucles:

In [16]:
```
fig, ax = plt.subplots(figsize=(4,4))
L1 = ['a', 'b', 'c', 'd', 'e', 'f']
L2 = [('a', 'b'), ('a', 'c'), ('c', 'd'), ('d', 'a'),('e', 'c'), ('e', 'd'), ('e', 'f'), ('f', 'c')]
G = nx.Graph()
G.add_nodes_from(L1)
G.add_edges_from(L2)
nx.draw(G, node_color="violet", edge_color="grey", font_size=10, width=2, with_labels=True, node_size=1000,
        alpha=0.9, ax=ax)
```



Ejercicio:

Es tu grafo creado anteriormente un grafo simple?.

El grado del nodo es el número de aristas adyacentes al nodo. El grado de nodo ponderado es la suma de los pesos de borde para los bordes que inciden en ese nodo.

Grado o valencia: Sea el grafo G = (V , A , Φ). Función grado: g: V → N_0. Dónde g(v_i) = cantidad de aristas incidentes en v_i, los bucles se cuentan dobles.

In [17]:
```
print(f"Grados de vértices: {G.degree()}\n\nGrado de vértices de c: \
{G.degree('c')}\n\nCantidad de vértices adyacentes a 'c': {G.degree['c']}")
```

```
Grados de vértices: [('a', 3), ('b', 1), ('c', 4), ('d', 3), ('e', 3), ('f', 2)]

Grado de vértices de c: 4

Cantidad de vértices adyacentes a 'c': 4
```

Propiedad: En todo grafo se cumple que la suma de los grados de los vértices es igual al doble de la cantidad de aristas.
En símbolos: ∑ g(v_i) = 2 | A |

adj: *Obtiene el objeto de adyacencia que contiene los vecinos de cada nodo.*

Vértices adyacentes: v_i es adyacente a v_j⇔ ∃(a_k) = {v_i,v_j}. Significa que son vértices que están unidos por alguna arista.

In [18]:
```
1 print("Vértices adyacentes a 'c'", list(G.adj['c']))
```

```
Vértices adyacentes a 'c' ['a', 'd', 'e', 'f']
```

Aristas adyacentes: a_i es paralela a $a_k \Leftrightarrow |\Phi(a_i) \cap \Phi(a_k)| = 1$. Significa que son aristas que tienen un único vértice en común.

In [19]:
```
1 print("Aristas adyacentes en 'c': ",G.degree['c'])
```

```
Aristas adyacentes en 'c':  4
```

Aristas incidentes en un vértice: a_i es incidente a $v_k \Leftrightarrow v_k \in \Phi(a_i)$. Significa que son las aristas que tienen a dicho vértice por extremo.

In [20]:
```
1 print("Aristas incidentes en 'd': ",G.edges('d'))
```

```
Aristas incidentes en 'd':  [('d', 'c'), ('d', 'a'), ('d', 'e')]
```

Ejercicio:

Consulta tu grafo y emite los vecinos de un nodo x elegido, el tamaño, el número de vértices,
el grado del nodo x, la cantidad de vértices adyacentes a x.

adjacency_matrix(): *Devuelve la matriz de adyacencia de G.*

Matriz de adyacencia: matriz booleana de n x n, Ma(G), cuyos elementos m_{ij} son 1 si v_i es adyacente a v_j, 0 si v_i no es adyacente a v_j. Significa que la matriz de adyacencia es una matriz cuadrada, las filas y las columnas representan los vértices, y los valores de los elementos son 1 si ambos vértices son adyacentes, y valen 0 en caso de no serlo.

In [21]:
```
1 Ma = nx.adjacency_matrix(G)
2 Ma.todense()
```

Out[21]:
```
matrix([[0, 1, 1, 1, 0, 0],
        [1, 0, 0, 0, 0, 0],
        [1, 0, 0, 1, 1, 1],
        [1, 0, 1, 0, 1, 0],
        [0, 0, 1, 1, 0, 1],
        [0, 0, 1, 0, 1, 0]], dtype=int32)
```

Networkx almacena las matrices de forma dispersa (posiciones no nulas y valor en éstas), y nosotros estamos acostumbrados a la forma densa (forma matricial).

incidence_matrix()

Matriz de incidencia: matriz booleana de n x n, Mi(G), cuyos elementos m_{ij} son 1 si v_i es extremo de a_j, 0 si v_i no es extremo de a v_j. Significa que la matriz de incidencia es una matriz rectangular, las filas representan los vértices, y las columnas representan las aristas, y los valores de los elementos son 1 si el vértice es extremo de la arista, y valen 0 en caso de no serlo.

In [22]:
```
1 Mi = nx.incidence_matrix(G)
2 Mi.todense()
```

Out[22]:
```
array([[1., 1., 1., 0., 0., 0., 0., 0.],
       [1., 0., 0., 0., 0., 0., 0., 0.],
       [0., 1., 0., 1., 1., 1., 0., 0.],
       [0., 0., 1., 1., 0., 0., 1., 0.],
       [0., 0., 0., 0., 1., 0., 1., 1.],
       [0., 0., 0., 0., 0., 1., 0., 1.]])
```

Ejercicio:

Crea la matriz de adyacencia y de incidencia del grafo de tu creación.

Caminos y ciclos en grafos

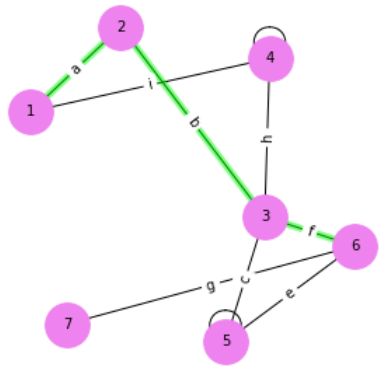
Camino: sucesión de aristas adyacentes distintas.

Un posible camino es: C1= (1, a, 2, b, 3, f, 6)

In [23]:

```
fig, ax = plt.subplots(figsize=(5,5))

G = nx.Graph([(1,2,{'label':'a'}),(1,4,{'label':'i'}),(2,3,{'label':'b'}),
              (4,3,{'label':'h'}),(3,5,{'label':'c'}),(3,6,{'label': 'f'}),
               (5,6,{'label':'e'}), (6,7,{'label':'g'})])
G.add_edge(4,4)
G.add_edge(5,5)
pos = nx.spring_layout(G,k=2,scale=3.0)
labels = nx.get_edge_attributes(G,'label')
nx.draw(G, pos,  with_labels=True, connectionstyle="arc3,rad=0.3",
        node_color="violet", font_size=10, width=1, node_size=1000)
nx.draw_networkx_edges(G, pos, connectionstyle='arc3,rad=0.1', edgelist = [(1,2),(2,3),(3,6)], width = 5,
                       alpha = 0.5, edge_color='lime')
nx.draw_networkx_edge_labels(G, pos, edge_labels=labels)
plt.show()
```



La longitud de este camino es: long[C1] = 3, es un camino simple porque no repite vértices

Longitud de un camino: cantidad de aristas que lo componen.

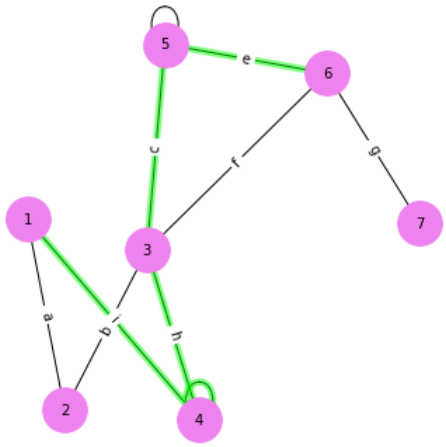
Camino simple: si todos los vértices son distintos.

Camino elemental: si todas las aristas son distintas.

Otro posible camino es: C2 = (1, i, 4, j, 4, h, 3, c, 5, e, 6)

In [58]:

```
fig, ax = plt.subplots(figsize=(5,5))
G = nx.Graph([(1,2,{'label':'a'}),(1,4,{'label':'i'}),(2,3,{'label':'b'}),
              (4,3,{'label':'h'}),(3,5,{'label':'c'}),(3,6,{'label': 'f'}),
               (5,6,{'label':'e'}), (6,7,{'label':'g'}), (4,4,{'label':'j'}), (5,5,{'label':'d'})])
pos = nx.spring_layout(G,k=2,scale=3.0)
labels = nx.get_edge_attributes(G,'label')
nx.draw(G, pos,  with_labels=True, connectionstyle="arc3,rad=0.3",
        node_color="violet", font_size=10, width=1, node_size=1000)
nx.draw_networkx_edges(G, pos, connectionstyle='arc3,rad=0.1', edgelist = [(1,4),(4,4),(4,3),(3,5),(5,6)], width = 5,
                       alpha = 0.5, edge_color='lime')
nx.draw_networkx_edge_labels(G, pos, edge_labels=labels)
plt.show()
```



La longitud de este camino es: long[C2] = 5, este camino NO es simple porque repite el vértice 4.

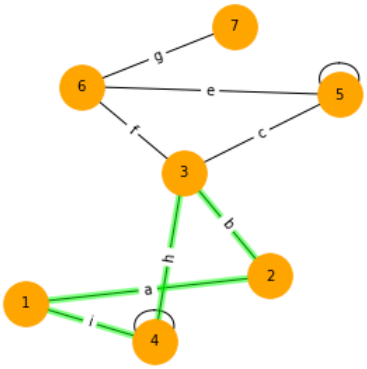
Ejercicio:

Tomando el grafo de tu creación, traza un camino, determina la longitud y si es simple o no.

Ciclo o circuito: camino cerrado (vértice inicial = vértice final)

Un posible ciclo es: C1 = (1, a, 2, b, 3, h, 4, i, 1)

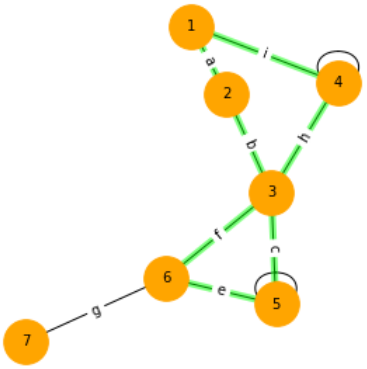
In [25]:
```
fig, ax = plt.subplots(figsize=(5,5))
G = nx.Graph([(1,2,{'label':'a'}),(1,4,{'label':'i'}),(2,3,{'label':'b'}),
              (4,3,{'label':'h'}),(3,5,{'label':'c'}),(3,6,{'label': 'f'}),
               (5,6,{'label':'e'}), (6,7,{'label':'g'})])
G.add_edge(4,4)
G.add_edge(5,5)
pos = nx.spring_layout(G,k=2,scale=3.0)
labels = nx.get_edge_attributes(G,'label')
nx.draw(G, pos,  with_labels=True, connectionstyle="arc3,rad=0.3",
        node_color="orange", font_size=10, width=1, node_size=1000)
nx.draw_networkx_edges(G, pos, connectionstyle='arc3,rad=0.1', edgelist = [(1,2),(2,3),(3,4),(1,4)], width = 5,
                       alpha = 0.5, edge_color='lime')
nx.draw_networkx_edge_labels(G, pos, edge_labels=labels)
plt.show()
```



La longitud de este ciclo es: long[C1] = 4, este ciclo es simple pues no repite vértices.

Otro posible ciclo es: C3 = (1, a, 2, b, 3, c, 5, e, 6, f, 3, h, 4, i, 1)

In [26]:
```
fig, ax = plt.subplots(figsize=(5,5))
G = nx.Graph([(1,2,{'label':'a'}),(1,4,{'label':'i'}),(2,3,{'label':'b'}),
              (4,3,{'label':'h'}),(3,5,{'label':'c'}),(3,6,{'label': 'f'}),
               (5,6,{'label':'e'}), (6,7,{'label':'g'})])
G.add_edge(4,4)
G.add_edge(5,5)
pos = nx.spring_layout(G,k=1,scale=3.0)
labels = nx.get_edge_attributes(G,'label')
nx.draw(G, pos,  with_labels=True, connectionstyle="arc3,rad=0.3",
        node_color="orange", font_size=10, width=1, node_size=1000)
nx.draw_networkx_edges(G, pos, connectionstyle='arc3,rad=0.1', edgelist = [(1,2),(2,3),(3,4),(1,4),(3,5),(3,6),(5,6)],
                       width = 5, alpha = 0.5, edge_color='lime')
nx.draw_networkx_edge_labels(G, pos, edge_labels=labels)
plt.show()
```



La longitud de este ciclo es: long[C3] = 7, este ciclo NO es simple porque repite el vértice 3.

Ejercicio:

Tomando el grafo de tu creación, traza un ciclo, determina la longitud y si es simple o no.

Grafos conexos

Un grafo conexo o conectado es un grafo en que todos sus vértices están conectados por un camino (si el grafo es no dirig ido) o por un semicamino (si el grafo es dirigido). Un grafo que no es conexo se denomina grafo disconexo o inconexo.

Dado un grafo G = (V, A, Φ), en el conjunto V se define la siguiente relación:

v_i R v_j ⇔ ∃ camino de v_i a v_j ∨ v_i = v_j

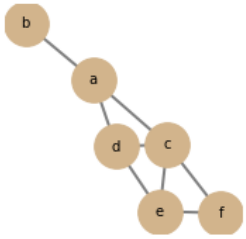
Esta relación es de equivalencia y por lo tanto pueden hallarse las clases de equivalencia, a las que se denomina componentes conexas.

Grafo conexo:

- **Un grafo es conexo si y sólo si tienen una única componente conexa.**
- **Un grafo es conexo si y sólo si existe algún camino entre todo par de vértices.**

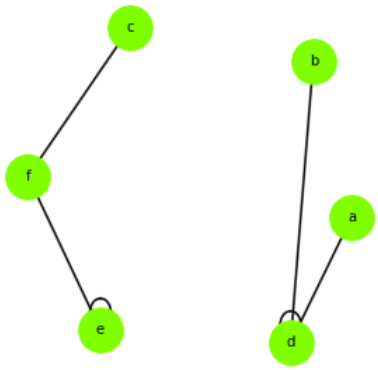
El siguiente grafo es conexo porque de cualquier vértice se puede llegar a cualquier otro a través de un camino.

In [27]:
```
fig, ax = plt.subplots(figsize=(3,3))
L1 = ['a', 'b', 'c', 'd', 'e', 'f']
L2 = [('a', 'b'), ('a', 'c'), ('c', 'd'), ('d', 'a'),('e', 'c'), ('e', 'd'), ('e', 'f'), ('f', 'c')]
G = nx.Graph()
G.add_nodes_from(L1)
G.add_edges_from(L2)
nx.draw(G, node_color="tan", edge_color="gray", font_size=10, width=2, with_labels=True, node_size=1000)
```



El siguiente grafo no es conexo porque, por ejemplo, no existe ningún camino entre los vértices a y c.

In [28]:
```
fig, ax = plt.subplots(figsize=(5,5))
G = nx.Graph([('a','d'), ('d','b'), ('d','d'), ('e','f'), ('f','c'), ('e','e')])
pos = nx.spring_layout(G,k=3,scale=2.0)
nx.draw(G, pos,  with_labels=True, connectionstyle="arc3,rad=0.3", node_color="chartreuse", font_size=10,
        edge_color="gray", width=2, node_size=1000)
nx.draw_networkx_edges(G, pos)
plt.show()
```



Sin embargo, está formado por dos subgrafos que cada uno de ellos sí es conexo, se llaman componentes conexas.

Caminos y ciclos especiales

Caminos y ciclos eulerianos

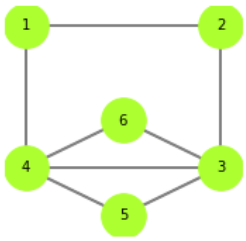
Camino de Euler: camino que pasa por todas las aristas.

La condición necesaria y suficiente para que en un grafo exista camino euleriano es: El grafo debe ser conexo, y todos los vértices deben tener grado par, o a lo sumo dos grados impar.

Ciclo de Euler: Ciclo que pasa por todas las aristas del grafo.

La condición necesaria y suficiente para que en un grafo exista ciclo euleriano es: El grafo debe ser conexo, y todos los vértices deben tener grado par.

In [29]:
```
fig, ax = plt.subplots(figsize=(3,3))
G = nx.Graph([(1,2),(2,3),(3,4),(4,6),(6,3),(3,5),(5,4),(4,1)])
pos_dict = {1:[ -0.1,1.4], 2: [0.1, 1.4],3: [ 0.1,0.8], 4: [-0.1,0.8], 6:[0.,1.], 5: [0.,0.6]}
positions=nx.spring_layout(G, pos=pos_dict)
nx.draw(G, pos_dict, with_labels=True, node_color="greenyellow", font_size=10, width=1, node_size=1000, ax=ax)
nx.draw_networkx_edges(G, pos_dict, edge_color='grey', width=2, ax=ax)
plt.show()
```



C = {1,2,3,4,6,3,5,4,1} es un ciclo euleriano.

Ejercicio:

El grafo de tu creación, tiene un camino o ciclo de Euler?.

Caminos y ciclos hamiltonianos

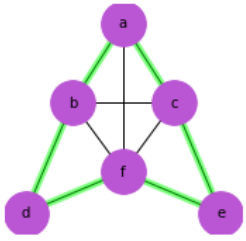
Camino de Hamilton: camino simple que pasa por todos los vértices.

Ciclo de Hamilton: Ciclo simple que pasa por todos los vértices.

Observación: no necesariamente va a pasar por todas las aristas, pues en muchos casos repetiría vértices y no sería hamiltoniano.

Un posible grafo hamiltoniano es: (a, b, d, f, e, c, a)

In [30]:
```
fig, ax = plt.subplots(figsize=(3,3))
G = nx.Graph([('a','b'),('a','c'), ('b','c'),('b','d'),('c','e'),('d','f'),('e','f'), ('a','f'), ('f','a'),
              ('c','f'), ('b','f')])
pos_dict = {'a':[ 0.,1.8], 'b': [-0.1,  1.6], 'c': [ 0.1,1.6], 'd': [-0.1,1.2], 'e': [0.1,1.2], 'f': [0.,1.]}
positions=nx.spring_layout(G, pos=pos_dict)
pos=nx.spring_layout(G,k=1,scale=3.0)
nx.draw(G, positions, with_labels=True, node_color="mediumorchid", font_size=10, width=1, node_size=1000)
nx.draw_networkx_edges(G, positions, connectionstyle='arc3,rad=0.1', edgelist = [('a','b'),('b','d'),('d','f'),
                                    ('f','e'),('e','c'),('c','a')], width = 5, alpha = 0.5, edge_color='lime')
plt.show()
```



Ejercicio:

El grafo de tu creación, tiene un camino o ciclo de Hamilton?.

Grafo regular

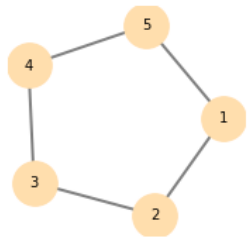
Un grafo regular es un grafo donde cada vértice tiene el mismo grado o valencia. Un grafo regular con vértices de grado k es llamado grafo k-regular o grafo regular de grado k.

Grafo K-regular: G es k-regular $\Leftrightarrow \forall v \in V : g(v) = k$ con $k \in N_0$

a_1 y a_5 son paralelas, a_1 es paralela a a_k Φ $(a_i) = (a_k)$. Significa que son aristas comprendidas entre los mismos vértices.

El siguiente grafo es 2-regular pues todos los vértices tienen grado 2.

In [31]:
```
fig, ax = plt.subplots(figsize=(3,3))
L1 = [1,2,3,4,5]
L2 = [(1,2),(2,3),(3,4),(4,5),(5,1)]
G = nx.Graph()
pos = nx.random_layout(G)
G.add_nodes_from(L1)
G.add_edges_from(L2)
nx.draw(G, node_color="navajowhite", edge_color="gray", font_size=10, width=2, with_labels=True, node_size=1000)
```



Isomorfismos de grafos

Dos grafos son isomorfos cuando tienen la misma estructura, es decir sus vértices están relacionados de igual forma, aunque estén dibujados de manera distinta.

Dados 2 grafos: $G_1 = (V_1 , A_1 , \Phi_1)$ y $G_2 = (V_2 , A_2 , \Phi_2)$.

Se dice que son isomorfos si y solo si existen dos funciones biyectivas: $f: V_1 \rightarrow V_2$ y $g: A_1 \rightarrow A_2$

Tales que: $\forall a \in A_1 : \Phi_2(g(a)) = f(\Phi_1(a))$

Si no hay aristas paralelas, entonces es suficiente: $\forall u,v \in V_1$: $\{u,v\} \in A_1 \rightarrow \{f(u), f(v)\} \in A_2$

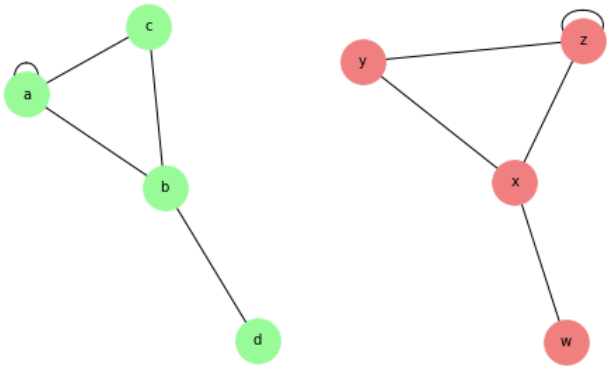
Condiciones necesarias para que 2 grafos sean isomorfos:
- Deben tener la misma cantidad de vértices.
- Deben tener la misma cantidad de aristas.
- Deben tener los mismos grados de los vértices.
- Deben tener caminos de las mismas longitudes.
- Si uno tiene ciclos, el otro también debe tenerlos.

Observación: las condiciones mencionadas son necesarias (es decir que sí o sí se deben cumplir para que los grafos sean isomorfos) pero no son suficientes (o sea que aunque se cumplan, puede ser que los grafos no sean isomorfos). Para estar seguros que dos grafos son isomorfos, una condición que es suficiente es que tengan la misma matriz de adyacencia.

In [32]:
```
plt.rcParams["figure.figsize"] = (8,5)
fig, axes = plt.subplots(nrows=1, ncols=2)
ax = axes.flatten()

G1 = nx.Graph([('a','b'),('a','c'),('b','c'),('b','d'), ('a','a')])
pos1 = nx.spring_layout(G1)
nx.draw(G1,pos1,with_labels=True, node_color="palegreen",font_size=10, width=1, node_size=1000, ax=ax[0])
ax[0].set_axis_off()

G2 = nx.Graph([('z','x'),('y','z'),('x','w'),('x','y'), ('z','z')])
pos2 = nx.spring_layout(G2)
nx.draw(G2,pos2,with_labels=True, node_color="lightcoral",font_size=10,width=1,node_size=1000, ax=ax[1])
ax[1].set_axis_off()
```



In [33]:
```
M1 = nx.adjacency_matrix(G1)
M1.todense()
```

Out[33]:
```
matrix([[1, 1, 1, 0],
        [1, 0, 1, 1],
        [1, 1, 0, 0],
        [0, 1, 0, 0]], dtype=int32)
```

In [34]:
```
M2 = nx.adjacency_matrix(G2)
M2.todense()
```

Out[34]:
```
matrix([[1, 1, 1, 0],
        [1, 0, 1, 1],
        [1, 1, 0, 0],
        [0, 1, 0, 0]], dtype=int32)
```

Ejercicio:

Tomando el grafo de tu creación, construye un grafo isomorfo y comprueba con las matrices.

Digrafo

Definición formal: Un digrafo es una terna G = (V , A , Φ), dónde:

V: Conjuntos de vértices, dónde V ≠ ∅

A: Conjuntos de aristas dirigidas.

Φ: Función de incidencia Φ: A → VXV

La función de incidencia Φ le hace corresponder a cada arista un par ordenado de vértices, al primero se lo llama extremo inicial de la arista, y el segundo es el vértice final.

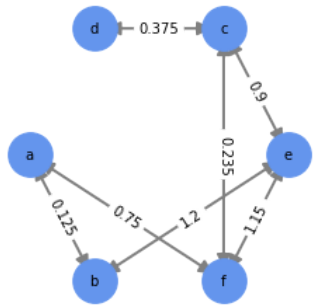
Los caminos y los ciclos se definen de la misma forma que para los grafos no dirigidos, pero hay que respetar el sentido de las aristas.

DiGraph()
to_directed()

In [35]:
```
fig, ax = plt.subplots(figsize=(4,4))

def emitoGraph(G, pos):
    nx.draw_networkx_nodes(G, pos, node_color = "cornflowerblue", node_size = 1000)
    nx.draw_networkx_labels(G, pos, font_size = 10, font_family = 'sans-serif')
    labels = nx.get_edge_attributes(G, 'weight')
    nx.draw_networkx_edges(G, pos, edge_color='gray', width=2, arrowstyle= '<|-|>', arrowsize = 20)
    nx.draw_networkx_edge_labels(G, pos, edge_labels = labels)

def cargoGraph(G):
    G.add_weighted_edges_from([('a','b',0.125),('a','f',0.75),('b','e',1.2),
                               ('c','e',0.90),('c','d',0.375),('e','f',1.15),('f','c',0.235)])
H = G.to_directed()
H = nx.DiGraph()
cargoGraph(H)
pos = nx.shell_layout(H)
emitoGraph(H, pos)
plt.axis('off')
plt.show()
```



to_directed() convierte el grafo no dirigido a uno dirigido o directamente se puede utilizar DiGraph()

Ejercicio:

Convierte tu grafo en un grafo dirigido.

Función grado de un digrafo

Grado positivo: cantidad de arcos que “entran” al vértice. Se denota g+(v)

Grado negativo: cantidad de arcos que “salen” del vértice. Se denota g-(v)

Grado total: suma de los grados positivo y negativo. Se denota g(v)

Grado neto: Diferencia entre grado positivo y negativo. Se denota $g_N(v)$

Propiedades:
$\Sigma\, g^+(v_i) = |\, A\,|$

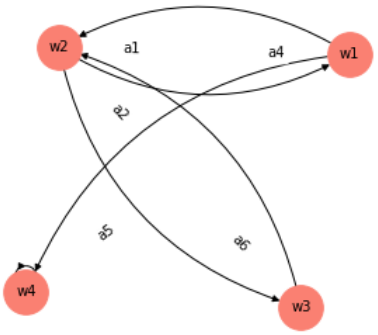
$\Sigma\, g^-(v_i) = |\, A\,|$

$\Sigma\, g(v_i) = 2\,|\, A\,|$

$\Sigma\, g_N(v_i) = 0$

Ejemplo:

In [36]:
```
fig, ax = plt.subplots(figsize=(5,5))
DG = nx.DiGraph([('w1','w2',{'label':'a1'}),('w1','w4',{'label':'a5'}),('w3','w2',{'label':'a2'}),
('w2','w3',{'label':'a6'}), ('w2','w1',{'label':'a4'}), ('w4','w4',{'label':'a3'})])
pos = nx.spring_layout(DG,k=4,scale=3.0)
labels = nx.get_edge_attributes(DG,'label')
nx.draw(DG, pos,  with_labels=True, connectionstyle="arc3,rad=0.3",
        node_color="salmon", font_size=10, width=1, node_size=1000)
nx.draw_networkx_edge_labels(DG, pos, edge_labels=labels, font_size=10, label_pos=0.25)
plt.show()
```



Grados positivos: $g^+(w_1) = 2$; $g^+(w_2) = 1$; $g^+(w_3) = 2$; $g^+(w_4) = 1$

Grados negativos: $g^-(w_1) = 1$; $g^-(w_2) = 3$; $g^-(w_3) = 0$; $g^-(w_4) = 2$

Grados totales: $g(w_1) = 3$; $g(w_2) = 4$; $g(w_3) = 2$; $g(w_4) = 3$

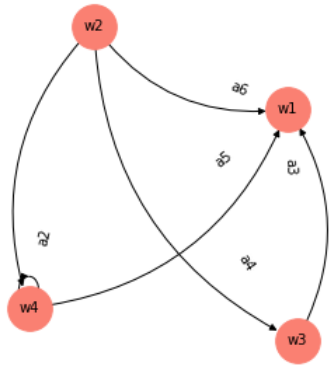
Grados netos: $g_N(w_1) = 1$; $g_N(w_2) = -2$; $g_N(w_3) = 2$; $g_N(w_4) = -1$

Pozo: es un vértice v tal que $g^-(v) = 0$, o sea, v no es extremo inicial de ninguna arista.

Fuente: es un vértice v tal que $g^+(v) = 0$, o sea, v no es extremo final de ninguna arista.

Ejemplo:

In [37]:
```
fig, ax = plt.subplots(figsize=(5,5))
DG = nx.DiGraph([('w4','w1',{'label':'a5'}),('w2','w4',{'label':'a2'}),
('w2','w1',{'label':'a6'}), ('w2','w3',{'label':'a4'}), ('w3','w1',{'label':'a3'}), ('w4','w4',{'label':'a3'})])
pos = nx.spring_layout(DG,k=4,scale=3.0)
labels = nx.get_edge_attributes(DG,'label')
nx.draw(DG, pos,  with_labels=True, connectionstyle="arc3,rad=0.3",
        node_color="salmon", font_size=10, width=1, node_size=1000)
nx.draw_networkx_edge_labels(DG, pos, edge_labels=labels, font_size=10, label_pos=0.25)
plt.show()
```



w1 es pozo, y w2 es fuente.

Ejercicio:

Crea un grafo dirigido con los siguientes datos:
[(0, 3), (1, 3), (2, 4), (3, 5), (4, 4), (3, 6), (4, 6), (5, 6)]

Grafo asociado a un digrafo

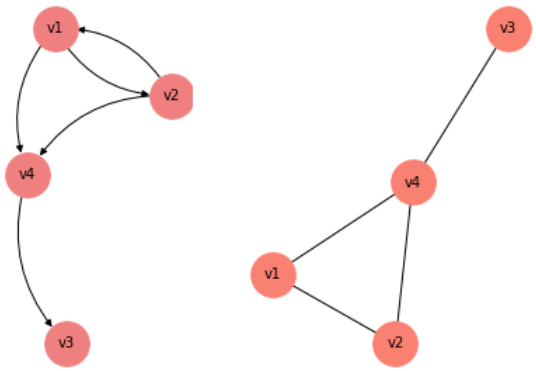
Dado un digrafo, si se cambian las aristas dirigidas por aristas no dirigidas, se obtiene el grafo asociado. Es decir hay que ignorar el sentido de las aristas. Si en el digrafo original hay aristas paralelas o antiparalelas, en el grafo asociado sólo se representa una de ellas.

Digrafo y grafo asociado

In [38]:
```
plt.rcParams["figure.figsize"] = (8,5)
fig, axes = plt.subplots(nrows=1, ncols=2)
ax = axes.flatten()

DG = nx.DiGraph([('v1','v2'),('v2','v4'),('v1','v4'),('v2','v1'), ('v4', 'v3')])
pos = nx.spring_layout(DG)
nx.draw(DG,pos,with_labels=True,connectionstyle="arc3,rad=0.3",node_color="lightcoral",
        font_size=10, width=1, node_size=1000, ax=ax[0])
ax[0].set_axis_off()

G = nx.Graph([('v1','v2'),('v2','v4'),('v1','v4'),('v4','v3')])
pos = nx.spring_layout(G)
nx.draw(G,pos,with_labels=True,node_color="salmon",font_size=10,width=1,node_size=1000, ax=ax[1])
ax[1].set_axis_off()
```



Ejercicio:

Con el último digrafo creado contruye su grafo asociado.

Digrafos conexos

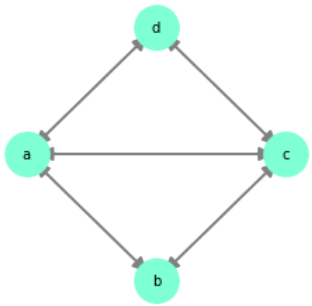
Es todo aquel cuyo grafo asociado sea conexo.
Dígrafo fuertemente conexo: es todo aquel en el que exista algún camino entre todo par de vértices.

In [39]:

```
fig, ax = plt.subplots(figsize=(4,4))

def emitoGraph(G, pos):
    nx.draw_networkx_nodes(G, pos, node_color = "aquamarine", node_size = 1000)
    nx.draw_networkx_labels(G, pos, font_size = 10, font_family = 'sans-serif')
    labels = nx.get_edge_attributes(G, 'weight')
    nx.draw_networkx_edges(G, pos, edge_color='gray', width=2, arrowstyle= '<|-|>', arrowsize = 20)
    nx.draw_networkx_edge_labels(G, pos, edge_labels = labels)

def cargoGraph(G):
    G.add_edges_from([('a','b'),('b','c'),('c','d'),
                          ('a','d'),('a','c')])
DG = nx.DiGraph()
cargoGraph(DG)
pos = nx.shell_layout(DG)
emitoGraph(DG, pos)
plt.axis('off')
plt.show()
```



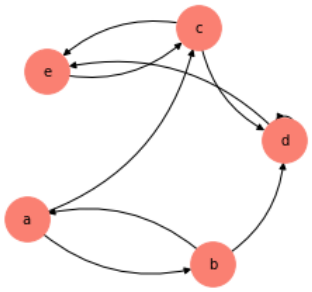
El digrafo anterior es conexo y además es fuertemente conexo.

Ejercicio:

Es tu digrafo conexo?.

In [40]:

```
fig, ax = plt.subplots(figsize=(4,4))
DG = nx.DiGraph([('a','b',{'label':'a1'}),('b','a',{'label':'a5'}),('a','c',{'label':'a2'}),
('e','c',{'label':'a6'}), ('c','e',{'label':'a4'}), ('d','e',{'label':'a3'}), ('c','d',{'label':'a4'}),
                ('b','d',{'label':'a4'}), ('d','d',{'label':'a4'})])
pos = nx.spring_layout(DG,k=4,scale=3.0)
labels = nx.get_edge_attributes(DG,'label')
nx.draw(DG, pos,  with_labels=True, connectionstyle="arc3,rad=0.3",
        node_color="salmon", font_size=10, width=1, node_size=1000)
plt.show()
```



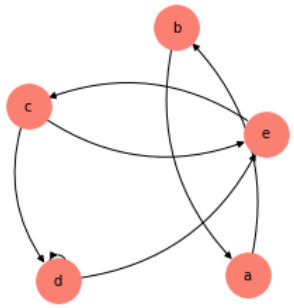
Este digrafo si bien es conexo, no es fuertemente conexo, ya que por ejemplo no existe camino alguno que salga del vértice C y llegue al vértice B.

Ejercicio:

Es tu digrafo fuertemente conexo?.

In [41]:

```
fig, ax = plt.subplots(figsize=(4,4))
DG = nx.DiGraph([('a','b',{'label':'a1'}),('b','a',{'label':'a5'}),('e','c',{'label':'a6'}), ('c','e',{'label':'a4'}),
                 ('d','e',{'label':'a3'}), ('c','d',{'label':'a4'}),('d','d',{'label':'a4'})])
pos = nx.spring_layout(DG,k=4,scale=3.0)
labels = nx.get_edge_attributes(DG,'label')
nx.draw(DG, pos,  with_labels=True, connectionstyle="arc3,rad=0.3",
        node_color="salmon", font_size=10, width=1, node_size=1000)
plt.show()
```



Lo que sí hay, son dos componentes fuertemente conexas.

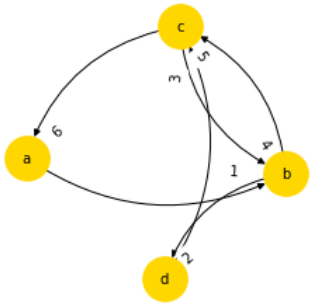
Caminos de Euler y Hamilton en digrafos

Se definen de forma similar que para grafos no dirigidos, pero hay que respetar el sentido de las aristas.

Condición necesaria y suficiente para que exista ciclo de Euler en un digrafo: ∀ v ∈V : g+(v) = g-(v)

In [42]:

```
fig, ax = plt.subplots(figsize=(4,4))
DG = nx.DiGraph([('a','b',{'label':'1'}),('b','d',{'label':'2'}),('d','c',{'label':'3'}), ('c','a',{'label':'6'}),
                 ('b','c',{'label':'5'}), ('c','b',{'label':'4'})])
pos = nx.spring_layout(DG,k=3,scale=2.0)
labels = nx.get_edge_attributes(DG,'label')
nx.draw(DG, pos, with_labels=True, connectionstyle="arc3,rad=0.3", node_color="gold", font_size=10,
        width=1, node_size=1000)
nx.draw_networkx_edge_labels(DG, pos, edge_labels=labels, label_pos=0.2)
plt.show()
```



En este digrafo existe ciclo de Euler: C = (A,1,B,2,D,3,C,4,B,5,C,6,A)
y un posible ciclo de Hamilton: C = (A,1,B,2,D,3,C,6,A)

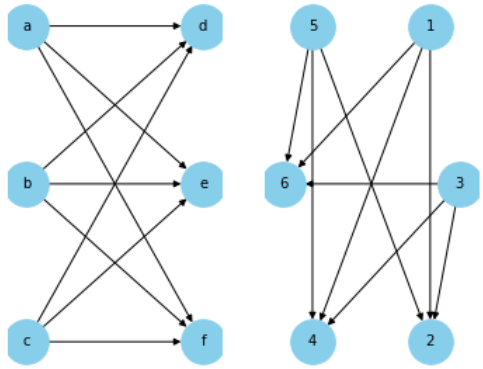
Isomorfismos de dígrafos

Es lo mismo que para grafos, pero hay que tener en cuenta el sentido de las aristas.

In [43]:
```
plt.rcParams["figure.figsize"] = (6,5)
fig, axes = plt.subplots(nrows=1, ncols=2)
ax = axes.flatten()

D1 = nx.DiGraph([('a','d'),('a','e'),('a','f'),('b','d'),('b','e'),('b','f'),('c','d'),('c','f'),('c','e')])
pos_dict = {'a': [-0.1, 0.9], 'b': [-0.1,0.7], 'c': [-0.1,0.5], 'd': [0.1,0.9], 'e': [0.1,0.7],'f': [0.1,0.5]}
nx.draw(D1, pos_dict, with_labels=True, node_color="skyblue", font_size=10, width=1, node_size=1000, ax=ax[0])
ax[0].set_axis_off()

D2 = nx.DiGraph([('3','4'),('3','6'),('3','2'),('5','4'),('5','6'),('1','6'),('1','2'),('1','4'),('5','2')])
pos_dict = {'5': [-0.1, 0.9], '1': [0.1,0.9], '4': [-0.1,0.5], '2': [0.1,0.5], '6': [-0.15,0.7],'3': [0.15,0.7]}
nx.draw(D2, pos_dict, with_labels=True, node_color="skyblue", font_size=10, width=1, node_size=1000, ax=ax[1])
ax[1].set_axis_off()
```



Si definimos la función: f tal que: f(1) = A ; f(2) = D ; f(3) = B ; f(4) = E ; f(5) = C ; f(6) = F, y construimos las matrices de adyacencia:

In [44]:
```
M1 = nx.adjacency_matrix(D1)
M1.todense()
```

Out[44]:
```
matrix([[0, 1, 1, 1, 0, 0],
        [0, 0, 0, 0, 0, 0],
        [0, 0, 0, 0, 0, 0],
        [0, 0, 0, 0, 0, 0],
        [0, 1, 1, 1, 0, 0],
        [0, 1, 1, 1, 0, 0]], dtype=int32)
```

In [45]:
```
M2 = nx.adjacency_matrix(D2)
M2.todense()
```

Out[45]:
```
matrix([[0, 1, 1, 1, 0, 0],
        [0, 0, 0, 0, 0, 0],
        [0, 0, 0, 0, 0, 0],
        [0, 0, 0, 0, 0, 0],
        [0, 1, 1, 1, 0, 0],
        [0, 1, 1, 1, 0, 0]], dtype=int32)
```

Como las matrices son iguales entonces los digrafos son isomorfos.

Grafos bipartitos

Los grafos bipartitos son grafos cuyo conjunto de vértices está particionado en dos subconjuntos: V1 y V2 tales que los vértices de V1 pueden ser adyacentes a los vértices de V2 pero los de un mismo subconjunto no son adyacentes entre sí.

Sea un grafo simple: G = (V , A , Φ) con V = $\{v_1.....v_n\}$ y A = $\{a_1.....a_m\}$

G es bipartito ⇔ V = V_1 U V_2 con $V_1 \neq \emptyset \wedge V_2 \neq \emptyset \wedge V_1 \cap V_2 = \emptyset \wedge \forall a_i \in A$: Φ $(a_i) = \{v_j, v_k\}$ con $v_j \in v_1 \wedge v_k \in v$

En el siguiente grafo, cuyo conjunto de vértices es: V = { 1, 2, 3, 4, 5 }
Si consideramos los subconjuntos: V1 = { 1, 2, 3 } V2 = { 4, 5 }
Vemos que todas las aristas que hay, tienen un extremo en V1 y el otro en V2, por lo tanto es bipartito.

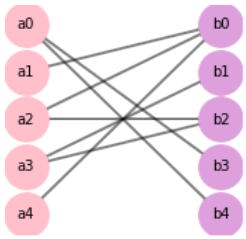
Nota: la definición no exige que deba haber arista entre todo par de vértices (uno de V1 y el otro de V2) sino que dice que las aristas que existan deben estar comprendidas entre un vértice de cada subconjunto. En este ejemplo, no hay arista entre 2 y 4, lo cual estaba permitido.

bipartite: Los grafos bipartitos tienen dos conjuntos de nodos y bordes que solo conectan nodos de conjuntos opuestos.

In [46]:

```
fig, ax = plt.subplots(figsize=(3,3))

mat = [ [0, 0, 0, 1, 1],
        [1, 0, 0, 0, 0],
        [1, 0, 1, 0, 0],
        [0, 1, 1, 0, 0],
        [1, 0, 0, 0, 0]]
G = nx.Graph()
a = ["a"+str(i) for i in range(len(mat))]
b = ["b"+str(j) for j in range(len(mat[0]))]
G.add_nodes_from(a, bipartite=0)
G.add_nodes_from(b, bipartite=1)
for i in range(len(mat)):
    for j in range(len(mat[i])):
        if mat[i][j] != 0:
            G.add_edge(a[i], b[j])
pos_a = {}
const = 0.100
x = 0.100
y = 1.0
for i in range(len(a)):
    pos_a[a[i]] = [x, y-i*const]
pos_b = {}
x = 0.500
for i in range(len(b)):
    pos_b[b[i]] = [x, y-i*const]
nx.draw_networkx_nodes(G, pos_a, nodelist=a, node_color="pink", node_size=1000)
nx.draw_networkx_nodes(G, pos_b, nodelist=b, node_color="plum", node_size=1000)
pos = {}
pos.update(pos_a)
pos.update(pos_b)
nx.draw_networkx_labels(G, pos, font_size=10)
nx.draw_networkx_edges(G, pos, width=2, alpha=0.5)
plt.axis('off')
plt.show()
```



Grafos completos Kn

En teoría de grafos, un grafo completo es un grafo simple donde cada par de vértices está conectado por una arista.

Sea n ∈ N: K_n = (V, A, Φ) tal que: ∀ v,w ∈ V: v ≠ w ⇔ ∃ a ∈ A: Φ (a) = {v, w}

O sea, los Kn son grafos simples de n vértices en los cuales cada vértice es adyacente a todos los demás. Ejemplos:

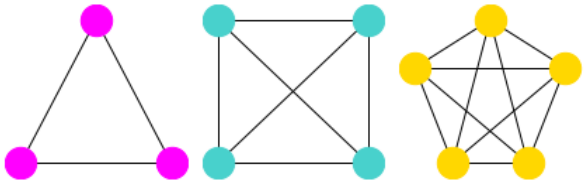
In [47]:

```
plt.rcParams["figure.figsize"] = [6,2]
plt.rcParams["figure.autolayout"] = True
fig, axes = plt.subplots(nrows=1, ncols=3)
ax = axes.flatten()

G1 = nx.Graph([('a','b'),('a','c'), ('b','c')])
pos_dict = {'a':[ 0.,0.8], 'b': [-0.1,  0.5], 'c': [ 0.1,0.5]}
nx.draw(G1, pos_dict, node_color="magenta", font_size=10, width=1, node_size=500, ax=ax[0])
ax[0].set_axis_off()

G2 = nx.Graph([('a','b'),('b','c'), ('c','d'),('d','a'),('a','c'),('b','d')])
pos_dict = {'a': [-0.05, 0.8], 'b': [0.05,0.8], 'c': [-0.05,0.5], 'd': [0.05,0.5]}
nx.draw(G2, pos_dict, node_color="mediumturquoise", font_size=10, width=1, node_size=500, ax=ax[1])
ax[1].set_axis_off()

G3 = nx.Graph([('a','b'),('a','c'),('a','d'),('a','e'),('b','c'),('b','d'),('c','e'),('d','e'),('b','e'),('c','d')])
pos_dict = {'a':[ 0.,1.4], 'b': [-0.1,  1.2], 'c': [ 0.1,1.2], 'd': [-0.05,0.8], 'e': [0.05,0.8]}
nx.draw(G3, pos_dict, node_color="gold", font_size=10, width=1, node_size=500, ax=ax[2])
ax[2].set_axis_off()
```



Grafos bipartitos completos $k_{n,m}$

En teoría de grafos, un grafo bipartito completo es un grafo bipartito en el que todos los vértices de uno de los subconjuntos de la partición están conectados a todos los vértices del segundo subconjunto, y viceversa.

Son grafos bipartitos de n + m vértices con todas las aristas posibles. La cantidad de aristas de un grafo $K_{n,m}$ es n * m

Ejemplos siguientes:

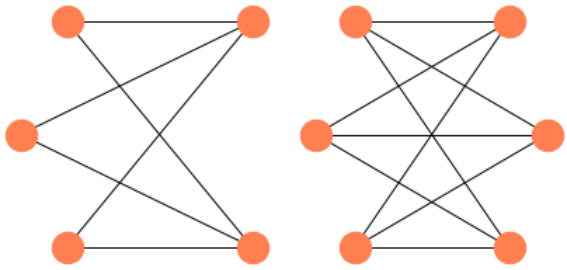
- Primer caso $K_{3,2}$
- Segundo caso $K_{3,3}$

In [48]:
```
plt.rcParams["figure.figsize"] = [6,3]
plt.rcParams["figure.autolayout"] = True
fig, axes = plt.subplots(nrows=1, ncols=2)
ax = axes.flatten()

G1 = nx.Graph([('a','b'),('b','c'),('c','d'),('d','a'),('b','e'),('d','e')])
pos_dict = {'a': [-0.1, 0.9], 'b': [0.1,0.9], 'c': [-0.1,0.5], 'd': [0.1,0.5], 'e': [-0.15,0.7]}
nx.draw(G1, pos_dict, node_color="coral", font_size=10, width=1, node_size=500, ax=ax[0])
ax[0].set_axis_off()

G2 = nx.Graph([('a','b'),('b','c'),('c','d'),('d','a'),('b','e'),('d','e'),('a','f'),('c','f'),('e','f')])
pos_dict = {'a': [-0.1, 0.9], 'b': [0.1,0.9], 'c': [-0.1,0.5], 'd': [0.1,0.5], 'e': [-0.15,0.7],'f': [0.15,0.7]}
nx.draw(G2, pos_dict, node_color="coral", font_size=10, width=1, node_size=500, ax=ax[1])
ax[1].set_axis_off()

```



Ejemplo de aplicación

In [49]:
```
aeropuertos = pd.read_csv('archs/aeropuertos.csv', encoding='latin-1')
df_a = pd.DataFrame(aeropuertos)
df_a.head(5)
```

Out[49]:

	COD	CIUDAD	AEROPUERTO	PROVINCIA
0	AEP	Ciudad de Buenos Aires	Jorge Newbery	Buenos Aires
1	EZE	Ezeiza	Ministro Pistarini	Buenos Aires
2	JNI	Junín	Junín	Buenos Aires
3	LPG	La Plata	La Plata	Buenos Aires
4	MDQ	Mar del Plata	Astor Piazzolla	Buenos Aires

In [50]:
```
vuelos = pd.read_csv("archs/combi_precios.csv", encoding='latin-1')
df_b = pd.DataFrame(vuelos)
df_b.head(5)
```

Out[50]:

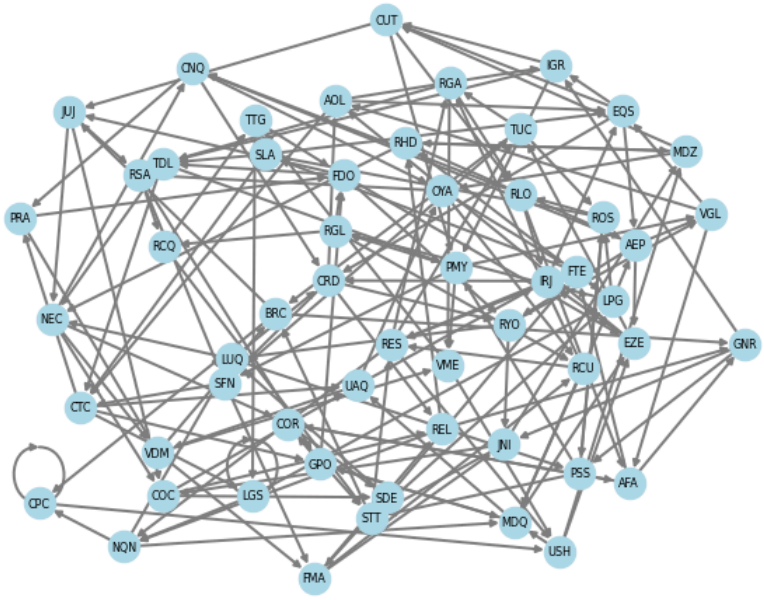
	Origen	Destino	Duracion	Precio
0	AEP	CNQ	95.45	680.0
1	EZE	IRJ	39.50	4780.0
2	JNI	COC	51.44	1160.0
3	LPG	AEP	66.26	7580.0
4	MDQ	GPO	18.85	720.0

In [51]:
```
DG = nx.DiGraph()
for i in range(0, len(df_a)):
    DG.add_node(df_a.iloc[i]['COD'])
    i = i + 1
```

In [52]:
```
for i in range(0, len(df_b)):
    DG.add_edge(df_b.iloc[i]['Origen'], df_b.iloc[i]['Destino'])
    i = i + 1
```

In [53]:

```
fig, ax = plt.subplots(figsize=(9,7))
DG.nodes(data=True)
nx.draw(DG, node_color="lightblue", edge_color="gray", font_size=8, width=2, with_labels=True,
                  node_size=500)
plt.show()
```



In [54]:

```
df_b = df_b.loc[:, ['Origen', 'Destino', 'Precio']]
df_b
```

Out[54]:

	Origen	Destino	Precio
0	AEP	CNQ	680.0
1	EZE	IRJ	4780.0
2	JNI	COC	1160.0
3	LPG	AEP	7580.0
4	MDQ	GPO	720.0
...	...	...	...
171	CTC	UAQ	1470.0
172	RES	RGA	2980.0
173	CRD	RYO	9260.0
174	EQS	AEP	2210.0
175	PMY	SFN	3780.0

176 rows × 3 columns

In [55]:

```
DG = nx.DiGraph()
DG.add_weighted_edges_from([tuple(x) for x in df_b.values])
# DG.edges()
```

In [56]:

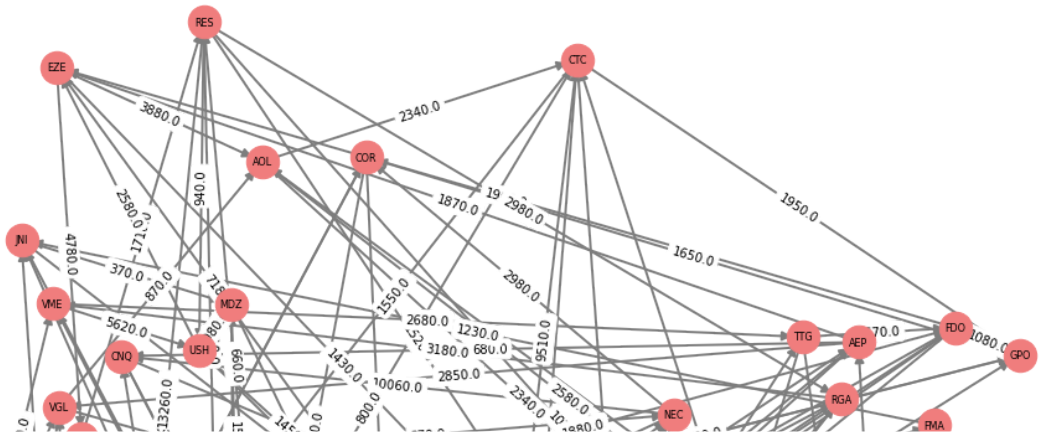
```
DG.get_edge_data('AEP','CNQ')
```

Out[56]:

```
{'weight': 680.0}
```

In [57]:

```
fig, ax = plt.subplots(figsize=(15,15))
pos = nx.random_layout(DG)
nx.draw_networkx_nodes(DG, pos, node_color="lightcoral", node_size=700)
nx.draw_networkx_labels(DG, pos, font_size=8, font_family='sans-serif')
labels = nx.get_edge_attributes(DG, 'weight')
nx.draw_networkx_edges(DG, pos, width=2, arrowstyle= '-|>', arrowsize = 15, edge_color='grey')
nx.draw_networkx_edge_labels(DG, pos, edge_labels=labels)
plt.axis('off')
plt.show()
```



¿Qué clase de grafo necesito representar?

Clase Networkx	Tipo	Autoloops permitidos	Bordes paralelos permitidos	Documentación
Graph	no dirigido	si	No	https://networkx.org/documentation/stable/reference/classes/graph.html (https://networkx.org/documentation/stable/reference/classes/graph.html)
DiGraph	dirigido	si	No	https://networkx.org/documentation/stable/reference/classes/digraph.html (https://networkx.org/documentation/stable/reference/classes/digraph.html)
MultiGraph	no dirigido	si	si	https://networkx.org/documentation/stable/reference/classes/multigraph.html (https://networkx.org/documentation/stable/reference/classes/multigraph.html)
MultiDiGraph	dirigido	si	si	https://networkx.org/documentation/stable/reference/classes/multidigraph.html (https://networkx.org/documentation/stable/reference/classes/multidigraph.html)

Ver más ejemplos (EjemplosNetworkx)