Programación—Certamen 2, jueves 14 de junio de 2012

Nombre: Rol: –

1. [30 %] Indique qué es lo que imprimen los siguientes programas.

```
a = [(1, 2), (3, 4), (5, 6)]
print len(a)
```

```
a = [[1], [2]]
a.append([3, 4, 5])
print len(a)
```

```
a = {'a': 'b', 'b': 'c', 'c': 'a'}
print a['a']
```

```
print len(set(range(5)) & set(range(3)))
```

```
d = {'a': 'b', 'b': 'c', 'c': 'a'}
a = ['d', 'e', 'a']
for c in d:
    if c in a:
        print d[c]
```

```
t = [(25, (1, 9)), (15, (2, 11))]
s = 0
for i in t:
    a, b = i
    c, d = b
    s += c
print s
```

```
def f(x):
    x = 18
    return x

x = 10
print f(f(f(x)))
```

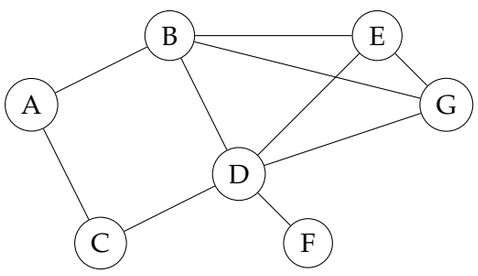
```
def f(a, b):
    return a * b

def g(a, b):
    return f(b, a + b)

a, b = (2, 1)
print f(g(b, a), g(a, b))
```

Programación—Certamen 2, jueves 14 de junio de 2012

Nombre: _____ Rol: _____ – _

2. **[35 %]** Anacleta, ciclista apasionada, planifica su próxima ruta usando mapas que descargó de internet. Cada mapa es un diccionario que asocia a cada ciudad el conjunto de sus ciudades vecinas. Dos ciudades son vecinas cuando hay un camino directo que las une.

Por ejemplo, la siguiente figura muestra un mapa con siete ciudades, junto con el diccionario que le corresponde:




```
mapa = {
   'A': {'B', 'C'},
   'B': {'A', 'D', 'E', 'G'},
   'C': {'A', 'D'},
   'D': {'B', 'C', 'E', 'G', 'F'},
   'E': {'B', 'D', 'G'},
   'F': {'D'},
   'G': {'E', 'B', 'D'},
}
```

Anacleta desea planificar una ruta (es decir, una secuencia de ciudades para visitar) y ver si es posible hacerla en mismo orden que a ella le interesa. No obstante, ella no se siente capacitada para hacer trayectos muy largos, por lo que en un día sólo le interesa ir de una ciudad a otra vecina. Sólo cuando se sienta especialmente vigorosa intentará seguir de largo para llegar a una próxima ciudad.

a) Escriba la función `son_vecinas(mapa, p, q)` que retorne un valor booleano indicando si las ciudades `p` y `q` son o no vecinas en el `mapa` (es decir, si se puede ir directamente de `p` a `q` sin pasar a través de ninguna otra ciudad).

```
>>> son_vecinas(mapa, 'A', 'B')
True
>>> son_vecinas(mapa, 'C', 'G')
False
```

b) Escriba la función `tienen_vecino_en_comun(mapa, p, q)` que retorne un valor booleano indicando si las ciudades `p` y `q` tienen o no un vecino en común en el `mapa` (es decir, si se puede ir de `p` a `q` pasando a través de una sola otra ciudad).

```
>>> tienen_vecino_en_comun(mapa, 'C', 'E')
True
>>> tienen_vecino_en_comun(mapa, 'A', 'F')
False
```

c) Escriba la función `existe_ruta(mapa, ruta)` que retorne un valor booleano indicando si es posible hacer la `ruta` en el orden indicado. El parámetro `ruta` es una lista de ciudades.

```
>>> existe_ruta(mapa, ['E', 'D', 'G', 'B', 'A'])
True
>>> existe_ruta(mapa, ['A', 'B', 'C', 'D', 'E'])
False
```

Conteste al reverso de esta misma hoja.

Programación—Certamen 2, jueves 14 de junio de 2012

Nombre: _____ Rol: _____–_

3. [35 %] Despues de llegar a su destino, Anacleta decidio abrir un café. Como ella anda en bicicleta no le gusta comprar comida de sobra, así que requiere que usted implemente un sistema de inventario que le permita calcular exactamente cuánta comida debe comprar para cada pedido.

Este sistema usa dos diccionarios: `inventario` y `recetas`. Las llaves del diccionario `inventario` son los ingredientes de las recetas de Anacleta, y sus valores son las cantidades que actualmente tiene en stock en el café:

```
inventario = {
    'huevos': 12,         # unidades
    'leche': 500,         # ml
    'papas': 3,           # unidades
    'margarina': 200,     # gr
    'azucar': 200,        # gr
    'sal': 50,            # gr
    'queso': 4,           # rebanadas
    'pan': 12,            # rebanadas
    'te': 5,              # bolsas
    # ...
}
```

Las llaves del diccionario `recetas` son los nombres de los platos que vende Anacleta, y los valores son listas de tuplas `(cantidad, ingrediente)` que requiere cada plato:

```
recetas = {
    'omelette': [(2, 'huevos'), (15, 'margarina'),
                 (1, 'sal'),    (10, 'leche')],
    'omelette con queso': [(2, 'huevos'), (10, 'margarina'),
                           (2, 'queso'),  (10, 'leche')],
    'pan con queso': [(2, 'pan'), (5, 'margarina'), (1, 'queso')],
    'te con leche':  [(1, 'te'), (10, 'leche'), (10, 'azucar')],
    'vaso de leche': [(500, 'leche')],
    # ...
}
```

Las recetas son siempre las mismas, por lo que `recetas` es una variable global.

a) Escriba la funcion `ingredientes(num_porciones, plato)`, que retorne un diccionario que indique cuánto se necesita de cada ingrediente para cocinar `num_porciones` del `plato`.

```
>>> ingredientes(2, 'omelette')
{'huevos': 4, 'leche': 20, 'margarina': 30, 'sal': 2}
>>> ingredientes(3, 'te con leche')
{'leche': 30, 'azucar': 30, 'te': 3}
```

b) Escriba la funcion `numero_de_porciones(plato, inv)`, que retorne el número máximo de porciones del plato que Anacleta puede cocinar usando el inventario `inv`.

```
>>> numero_de_porciones('omelette', inventario)
6
>>> numero_de_porciones('pan con queso', inventario)
4
```

c) Escriba la funcion `cocinar_juntos(pedido, inv)`, que retorne **`True`** si el pedido se puede cocinar con el inventario en `inv`, o **`False`** si no hay suficientes ingredientes. Un pedido es una lista de tuplas `(numero_de_porciones, plato)`.

```
>>> cocinar_juntos([(2, 'omelette'), (2, 'te con leche')], inventario)
True
>>> cocinar_juntos([(1, 'omelette con queso'),
...                 (1, 'pan con queso'),
...                 (500, 'vaso de leche')], inventario)
False
```

Conteste al reverso de esta misma hoja.