PONTIFICIA UNIVERSIDAD CATÓLICA DE CHILE
ESCUELA DE INGENIERÍA
DEPARTAMENTO DE CIENCIA DE LA COMPUTACIÓN

IIC2233 – Programación Avanzada
Interrogación 1

13 de Abril 2015

Duración: 2,5 horas. Sin consultas.

1. (15 pts) Usted tiene una fábrica de equipos de visión artificial y quiere sacar a la venta un nuevo sistema de cámaras IP que permita tomar una imagen panorámica usando varias cámaras simultáneamente.

   Una cámara IP está formada por:

   - Una cámara tradicional
   - Una tarjeta de red
   - Un método `leer_imagen()` que le permite enviar la imagen y la dirección IP de la cámara IP que la generó.

   Una cámara tradicional contiene:

   - Un número de serie
   - Modo de grabación
   - Tasa de captura (fps)
   - Un método llamado `capturar()` que permite adquirir la imagen que está en ese momento en el sensor de la cámara y devolverla como un string.

   Una tarjeta de red (NIC) contiene:

   - Dirección MAC de cada cámara IP
   - Dirección IP asignada como un string por un servidor
   - Un método llamado `enviar()` que permite enviar una tupla con la dirección de la tarjeta de red que generó el paquete y un strings de datos

   Uno de los desarrolladores tomó el excelente curso IIC2233 y determinó que el parámetro IP debería funcionar mejor usando properties.

   Considerar que el sistema:

   - Recibe un listado con los datos de las cámaras IP que serán activadas, en donde cada cámara IP es descrita usando un diccionario con sus atributos, como se sugiere a continuación:

```
listado_camaras = [
    {'serie':123, 'modo':'640x480', 'fps':30, 'mac':'AACC23'},
    {'serie':456, 'modo':'1024x720', 'fps':30, 'mac':'AACC89'}
]
```

- Tiene un método `asignar_ip()` que asigna una dirección ip a cada cámara en el listado.
- Tiene un método denominado `capturar()` que permite obtener una lista con todas las imágenes provenientes desde las cámaras agregadas al sistema.
- Posee un listado de direcciones ip que pueden ser asignados.

Asuma que ya están implementadas las clases Camara, NIC y CamaraIP con todos sus atributos y métodos descritos anteriormente. Escriba en Python el código correspondiente a la clase que representa el sistema de cámaras con los atributos y métodos necesarios según las especificaciones (haga los supuestos que sean necesarios).

2. a) (15 pts) Escriba una solución en Python que dada una lista de tuplas de dos valores, donde el primer valor se refiere a una entidad y el segundo valor se refiere a otra entidad que está contenida en la primera (Ej: (‘biblioteca’, ‘estante’) , (‘libro’,‘hojas’) , (‘cubiculo’, ‘mesa’) , (‘cubiculo’, ‘silla’)), escriba todas sus componentes (sin repetición) para una entidad dada. Por ejemplo, para la lista de tuplas [(‘biblioteca’, ‘estante’) , (‘biblioteca’, ‘cubiculo’) , (‘estante’, ‘libro’) , (‘libro’, ‘hojas’) , (‘cubiculo’, ‘mesa’) , (‘cubiculo’, ‘silla’), (‘cubiculo’, ‘puerta’)], si queremos las componentes de ‘biblioteca’, el output de su programa debería ser:

   Componentes(‘biblioteca’) $\rightarrow$ ‘estante’, ‘cubiculo’, ‘libro’, ‘hojas’, ‘mesa’, ‘silla’, ‘puerta’

b) (10 pts) Dado el siguiente código en Python explique lo que hace cada método y genere la salida descrita en el `main`:

```
class Nodo:
    def __init__(self, valor, izquierda=None, derecha=None):
        self.valor = valor
        self.izquierda = izquierda
        self.derecha = derecha

    def m1(self, f):
        r = ''
        if self.izquierda:
            r += f('pre', self.valor) + self.izquierda.m1(f)

        r += f('in', self.valor)

        if self.derecha:
            r += self.derecha.m1(f) + f('post' , self.valor)
        return r

class Arbol:
    def __init__(self, raiz):
        self.raiz = raiz
    def m2(self, f):
        if self.raiz: return self.raiz.m1(f)
        else:  return None
    def __repr__(self):
        def m3(s, valor):
            if s == 'pre':
                return '('
            elif s == 'in':
                return str(valor)
            else:
                return ')'
        return self.m2(m3)

if __name__ == '__main__':
    T = Arbol(Nodo('+', Nodo('-', Nodo(3), Nodo(4)), Nodo(2)))
    print(T)
```

3. a) (13 pts.) Defina una metaclase que asegure que cada vez que se llama a la clase para crear una nueva instancia, ésta retorne una nueva instancia sólo si ninguna se ha creado antes, en otro caso, se debería retornar la misma instancia creada la primera vez.

   b) (12 pts.) Defina una metaclase que asegure que la clase que se está creando no puede ser subclaseada, es decir, cualquier intento del programador de heredar de esta clase debería imprimir un error.

4. a) (5 pts.) Explique qué es un decorador y para qué puede servir.

   b) (10 pts.) Implemente la función **factorial** usando **reduce**.

   c) (10 pts.) Dadas dos listas de números L1 y L2, genere un código que imprima el producto de todos los números contenidos en la lista L3 cuyo elemento i-ésimo (L3[i]) es el que resulta de multiplicar L1[i]*L2[i]. Si L1 y L2 son de distinto tamaño, los elementos que sobran en la lista más larga deben no considerarse para la multiplicación. No puede usar **for** ni **if**.

   d) (10 pts.) Escriba la salida que el siguiente código genera con lo que se define en el main (justifique su respuesta):

```
def version(modo):
    def _version(cls):
        def _debug(self):
            attr = vars(self)
            print('MODO DEPURACION')
            for a, v in attr.items():
                print('{}: {} - {}'.format(a, v, type(v)))

        def _release(self):
            return None

        dict_modos = {'DEBUG': _debug, 'RELEASE': _release}
        setattr(cls, '__call__', dict_modos[modo])

        return cls
    return _version

#@version('RELEASE')
@version('DEBUG')
class Imagen:
    def __init__(self, size):
        self.size = size
        self._data = None

    @property
    def data(self):
        return self._data

    @data.setter
    def data(self, data):
        self._data = data

if __name__ == '__main__':
    img = Imagen((20,30))
    img.data = [255, 50, 100]
    img()
```