Universidad Simón Bolívar
Departamento de Computación y Tecnología de la Información
CI-2692 - Laboratorio de Algoritmos y Estructuras II
Trimestre Enero-Marzo 2017

Introducción a la programación orientada a objetos. TADs Pila y Cola

Introducción

El objetivo del laboratorio es adquirir los conocimientos básicos en la programación orientada a objetos, así como la implementación de tipos abstractos de datos esenciales y para ello usará el código base adjunto a este enunciado.Tendrá que crear dos archivos llamados:

- **pila.py**. Módulo que contiene la implementación del tipo de dato que representa una **pila**.
- **cola.py**. Módulo que contiene la implementación del tipo de dato que representa una **cola**.

Mas adelante se le indicaran cuales métodos deberá implementar pero en primer lugar se definen los características básicas del paradigma orientado a objetos.

Objetos

Los objetos son contenedores de información, que toma la forma de atributos, llamados también propiedades, y métodos que manipulan operan con esos atributos. Toda esas interacciones que ocurran serán dentro del objeto y esa será su frontera. Aun cuando existe esa barrera, los objetos pueden relacionarse y crear referencias entre ellos. En primer lugar se hará un definición breve de los posibles objetos que pueden existir dentro del paradigma.

Clases

Las clases son quienes indican cuales atributos y métodos tendrá, y podrá disponer, una instancia. Definir una clase en *Python* sería como se muestra a continuación.

```
1    class Perro:
2
3        def __init__(self, nombre):
4            self.nombre = nombre
```

Esta clase representa a los perros y al momento de crear un representante, una instancia, se le asigna un nombre, el cuál puede ser preguntado a dicha instancia. El método `__init__` es especial porque es llamada justo despues de crear la instancia e inicializa los atributos con los parametros que le fueron pasados.

Instancias

Las instancias son los representantes de una clase, pueden ser visto como miembros de un conjunto o elementos pertenecientes a un tipo, ambos son puntos de vista válidos. A continuación se ilustra la obtención de un atributo a una instancia tomando en cuenta la clase *Perro* definida anteriormente.

```
1    p = Perro("Brownie") # Se crea una instancia de la clase Perro
2
3    aux_nombre = p.nombre # Se pregunta por el nombre y es asignado
4
5    print(aux_nombre) # Se imprime por consola el nombre del perro
```

Atributos y Métodos

Hay dos tipos de atributos y dos de tipos de métodos, aquellos que pertenecen a las instancias y los que pertenecen a las clases. Se separan de la siguiente manera:

- Atributo de instancia. Propiedad singular perteneciente a cada instancia de la clase.
- Atributo de clase. Propiedad común para todos las instancias dado que pertenecen a una misma clase.
- Método de instancia. Procedimiento perteneciente a la instancia y que puede manipular los atributos de instancia juntos a los atributos de clase.
- Método de clase. Procedimiento exclusivo para la clase y sólo puede acceder a las variables de clase.

Tomando la siguiente definición de clase con un atributo de clase, se indicará como se puede obtener el valor guardado.

```
1    class Gato:
2
3        genero = 'felino' # Atributo de clase
4
5        def __init__(self, nombre):
6            self.nombre = nombre # Atributo de instancia
```

Para acceder a la variable de clase se debe indicar la clase a consultar y el nombre del atributo, similar a las instancias.

```
1    g = Gato.genero # Se obtiene el genero de los gatos
2
3    print(g) # Imprime el genero
```

De igual forma se accede a los atributos de clase dentro de los métodos de instancia, lo que se han definido hasta el momento. Por otro lado los métodos de clase son definidos de la siguiente manera, usando la declaración `@classmethod`.

```
1   class Gato:
2
3       genero = 'felino'
4
5       def __init__(self,nombre):
6           self.nombre = nombre
7
8       @classmethod
9       def imprimir_genero(cls):
10          s = 'Genero: ' + cls.genero
11          print(s)
```

Una llamada a un método de clase sería de la siguiente forma.

```
1   Gato.imprimir_genero()
```

self

Hasta el momento se ha usado la variable `self` en todos los métodos y esto es porque es una variable especial, es una referencia directa a la instancia que permite vincularla con sus métodos. Con el objetivo de consultar y manipular los atributos de la misma.

Referencias

Por último, es relevante mencionar como es posible relacionar instancias singulares a través de referencias. Tomando como ejemplo la clase `Nodo` que se encuentra en el archivo **nodo.py** se puede crear una lista enlazada.

```
1   class Nodo:
2       def __init__(self, e, n):
3           self.element = e
4           self.next = n
5
6   lista1 = Nodo(1, Nodo(2, Nodo(3, Nodo(4, Nodo(5, None)))))
```

Como puede ver, es sencillo definir una lista enlazada por medio de clases y debe usarlas en el desarrollo del laboratorio. El uso de las listas de *Python* está prohibido.

Requerimientos específicos

Todos los archivos base para el desarrollo del laboratorio se encuentran adjuntos a este enunciado. Usted tendrá que crear los archivos **pila.py** y **cola.py**, con los siguientes métodos en cada archivo.

- **pila.py**
  - *is_empty*. Devuelve `True` si la pila está vacía, sino retorna `False`
  - *push*. Agrega un elemento en el tope de la pila
  - *pop*. Devuelve y remueve el elemento en el tope de la pila
  - *top*. Devuelve pero no remueve el elemento en el tope de la pila
  - *size*. Devuelve el número de elementos en la pila
- **cola.py**
  - *is_empty*. Devuelve `True` si la cola está vacía, sino retorna `False`
  - *enqueue*. Agrega un elemento al final de la cola
  - *dequeue*. Devuelve y remueve el primer elemento de la cola
  - $first$. Devuelve pero no remueve el primer elemento de la cola
  - *size*. Devuelve el número de elementos en la cola

Es importante resaltar el uso de la clase `Nodo` en el archivo **nodo.py**, esta debe ser importada en ambos archivos. Para probar el funcionamiento de su pila y cola hay dos archivos, **prueba_pila.py** y **prueba_cola.py**. La salida de cada archivo debe ser la siguiente respectivamente. Donde la primera columna es el método llamado, la segunda es el valor de retorno y la tercera es el estado actual de la estructura.

```
1     'p.push(27)'        None            [27]
2     'p.push(9)'         None           [9,27]
3      'p.size()'          2             [9,27]
4      'p.top()'           9             [9,27]
5      'p.pop()'           9              [27]
6     'p.push(3)'         None           [3,27]
7   'p.is_empty()'       False           [3,27]
8      'p.pop()'           3              [27]
9      'p.size()'          1              [27]
10     'p.pop()'           27             None
11  'p.is_empty()'        True           None
12     'p.pop()'           None           None
```

```
1     'c.enqueue(3)'      None            [3]
2     'c.enqueue(5)'      None           [3,5]
3       'c.size()'         2             [3,5]
4     'c.enqueue(9)'      None          [3,5,9]
5     'c.dequeue()'        3             [5,9]
6      'c.first()'         5             [5,9]
7     'c.is_empty()'      False          [5,9]
8     'c.dequeue()'        5              [9]
9     'c.dequeue()'        9              None
10      'c.size()'         0              None
11    'c.is_empty()'       True           None
12    'c.dequeue()'        None           None
```

Entrega del laboratorio

Al finalizar el desarrollo del laboratorio, deberá comprimir el código fuente en un archivo llamado *lab_sem_8_X_Y.tar.gz* donde X y Y corresponden al carné de cada integrante del grupo. En caso de no tener pareja llame el comprimido *lab_sem_8_X.tar.gz* donde X es su número de carné. La entrega será hasta las 4:30pm del jueves 2 de marzo de 2017.