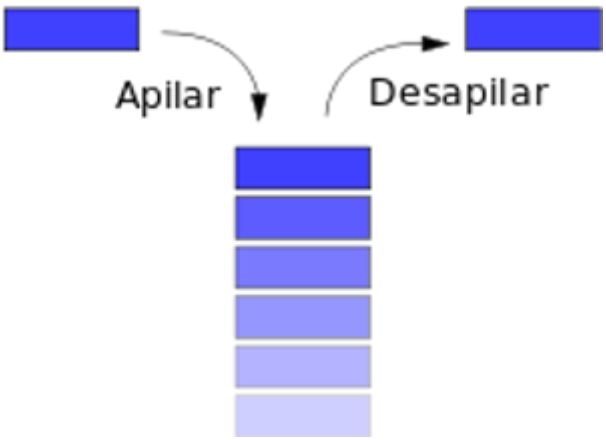
Pilas y Colas

Las pilas (stacks) y las colas (queues) son tipos abstractos de datos que nos permiten acceder a la información contenida en ellas de manera controlada, ya que solo se puede acceder a ella desde uno de los extremos (inicio o final). Tanto una pila como una cola pueden ser implementadas en una estructura de datos física como una lista enlazada o un vector, lo importante en cada caso es que las restricciones de acceso, inserción y borrado sean establecidas correctamente.

Pila (stack)

Out[3]:



Las pilas son estructuras de datos lineales donde la restricciones nos indican que la **inserción (apilar o push)** de un nuevo elemento solo debe realizarse desde la *cima* (o inicio) de la pila, y la **extracción o borrado (desapilar o pop)** de un elemento debe también realizarse desde la *cima* de la pila.

Este comportamiento hace que la pila sea una estructura tipo **LIFO** (*last in first out*, el último que entra es el primero que sale). En ciencias de la computación las pilas suelen usarse para:

- Evaluación de expresiones en notación postfija (notación polaca inversa).
- Reconocedores sintácticos de lenguajes independientes del contexto
- Implementación de algortimos recursivos.
- Etc.

Operaciones

Las operaciones principales de una pila son **apilar (push)** y **desapilar (pop)**. Usualmente se suelen agregar las funciones **cima (top)**, **tamaño (size)** y **vacio (is_empty)** como una manera de darle un mayor soporte al programador al momento de manejar la pila.

- **apilar (push):** inserta un elemento en la cima de la pila.
- **desapilar (pop):** retorna el elemento en la cima de la pila y lo elimina de la estructura.
- **cima (top):** retorna el elemento en la cima de la pila sin eliminarlo.

- **tamaño (size):** retorna la cantidad de elementos de la pila.
- **vacio (is_empty):** retorna verdadero si la pila no tiene elementos.

Además cuando una pila es implementada sobre un vector o arreglo se suele incluir una función que indique el tamaño máximo que puede tener la pila (**max_size**).

Ejemplo de pila implementada a partir de una lista enlazada con nodo centinela fuera de la lista.

In [1]:
```
class nodo:
    def __init__(self, dato):
        self.dato = dato
        self.sig = None

    def add(self, nuevo):
        nuevo.sig = self.sig # para que no se pierda el resto de la lista
        self.sig = nuevo

    def __str__(self):
        if(self.sig == None):
            return "({})->(None)".format(self.dato)

        return "({0})->{1}".format(self.dato, str(self.sig))

    def borrar(self, elem):
        if(self.sig == None):
            return None

        if(self.sig.dato == elem):
            self.sig = self.sig.sig
            return 1
        else:
            return self.sig.borrar(elem)

class lista_enlazada:
    def __init__(self):
        self.first = None # siempre la referencia al primero
        self.last = None # siempre la referencia al ultimo
        self.curr = None # a partir de esta referencia moverse en la lista
        self.num_elem = 0 # cantidad de elementos en la lista

    def __str__(self):
        # recorrer e imprimir
        return str(self.first)

    # ejemplo de uso de la referencia curr sobrecargando []
    def __getitem__(self, index):
        if(type(index) != type(int())): raise TypeError("Index must be inte
        if(index > self.num_elem): raise ValueError("Index out of range.")
        if(index < 0): raise ValueError("Index must be positive or zero.")

        self.curr = self.first
        curr_idx = 0
        while(curr_idx != index):
            curr_idx += 1
            self.curr = self.curr.sig

        return self.curr.dato

    def push(self, nuevo):
        if(self.first == None):
            self.first = nuevo
            self.last = nuevo
        else:
            nuevo.sig = self.first
```

```
            self.first = nuevo

        self.num_elem += 1

    def pop(self):
        if(self.first == None):
            return None
        else:
            aux = self.first
            self.first = self.first.sig
            self.num_elem -= 1
            return(aux)

    def top(self):
        return self.first

    def size(self):
        return(num_elem)

    def is_empty(self):
        if(self.num_elem == 0):
            return True
        else:
            return False
```

Ejemplo de uso de la pila

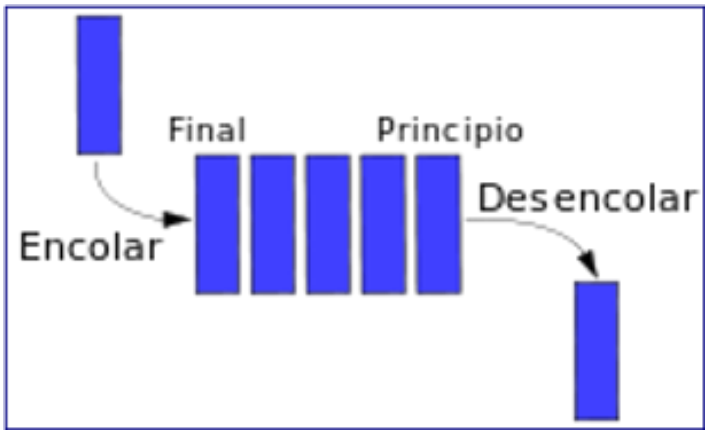
In [5]:
```
# construcción y llenado de la lista
head = lista_enlazada()
head.push(nodo("d1"))
head.push(nodo("d2"))
head.push(nodo("d3"))
head.push(nodo("d4"))
head.push(nodo("d5"))
head.push(nodo("d6"))

# uso del método sobrecargado str() y el borrado
print(str(head))
print(type(head.pop()))
print(str(head))
```

```
(d6)->(d5)->(d4)->(d3)->(d2)->(d1)->(None)
<class '__main__.nodo'>
(d5)->(d4)->(d3)->(d2)->(d1)->(None)
```

Cola (queue)

Out[5]:



Las colas son estructuras de datos lineales donde la restricciones nos indican que la **inserción (encolar o push)** de un nuevo elemento debe realizarse desde el final de la cola, y la **extracción o borrado (desencolar o pop)** de un elemento debe realizarse desde el inicio de la cola, es decir, ambas operaciones se realizan desde extremos opuestos de la estructura.

Este comportamiento hace que la cola sea una estructura tipo **FIFO** (*first in first out*, el primero que entra es el primero que sale). Las colas se utilizan en sistemas informáticos, transportes y operaciones de investigación (entre otros), dónde los objetos, personas o eventos son tomados como datos que se almacenan y se guardan mediante colas para su posterior procesamiento.

Operaciones

Las operaciones principales de una cola son **encolar (push)** y **desencolar (pop)**. Usualmente se suelen agregar las funciones **frente (front)**, **tamaño (size)** y **vacio (is_empty)** como una manera de darle un mayor soporte al programador al momento de manejar la cola.

- **encolar (push):** inserta un elemento al final de la cola.
- **desencolar (pop):** retorna el elemento del inicio de la cola y lo elimina de la estructura.
- **frente (front):** retorna el elemento del inicio de la cola sin eliminarlo.
- **tamaño (size):** retorna la cantidad de elementos de la cola.
- **vacio (is_empty):** retorna verdadero si la cola no tiene elementos.

Además cuando una cola es implementada sobre un vector o arreglo se suele incluir una función que indique el tamaño máximo que puede tener la pila (**max_size**).

Ejemplo de cola implementada a partir de una lista enlazada con nodo centinela fuera de la lista.

In [6]:
```
class nodo:
    def __init__(self, dato):
        self.dato = dato
        self.sig = None

    def add(self, nuevo):
        nuevo.sig = self.sig # para que no se pierda el resto de la lista
        self.sig = nuevo

    def __str__(self):
        if(self.sig == None):
            return "({})->(None)".format(self.dato)

        return "({0})->{1}".format(self.dato, str(self.sig))

    def borrar(self, elem):
        if(self.sig == None):
            return None

        if(self.sig.dato == elem):
            self.sig = self.sig.sig
            return 1
        else:
            return self.sig.borrar(elem)

class lista_enlazada:
    def __init__(self):
        self.first = None # siempre la referencia al primero
        self.last = None # siempre la referencia al ultimo
        self.curr = None # a partir de esta referencia moverse en la lista
        self.num_elem = 0 # cantidad de elementos en la lista

    def __str__(self):
        # recorrer e imprimir
        return str(self.first)

    # ejemplo de uso de la referencia curr sobrecargando []
    def __getitem__(self, index):
        if(type(index) != type(int())): raise TypeError("Index must be inte
        if(index > self.num_elem): raise ValueError("Index out of range.")
        if(index < 0): raise ValueError("Index must be positive or zero.")

        self.curr = self.first
        curr_idx = 0
        while(curr_idx != index):
            curr_idx += 1
            self.curr = self.curr.sig

        return self.curr.dato

    def push(self, nuevo):
        if(self.first == None):
            self.first = nuevo
            self.last = nuevo
        else:
            self.last.sig = nuevo
```

```
            self.last = nuevo

        self.num_elem += 1

    def pop(self):
        if(self.first == None):
            return None
        else:
            aux = self.first
            self.first = self.first.sig
            self.num_elem -= 1
            return(aux)

    def front(self):
        return self.first

    def size(self):
        return(num_elem)

    def is_empty(self):
        if(self.num_elem == 0):
            return True
        else:
            return False
```

Ejemplo de uso de la cola.

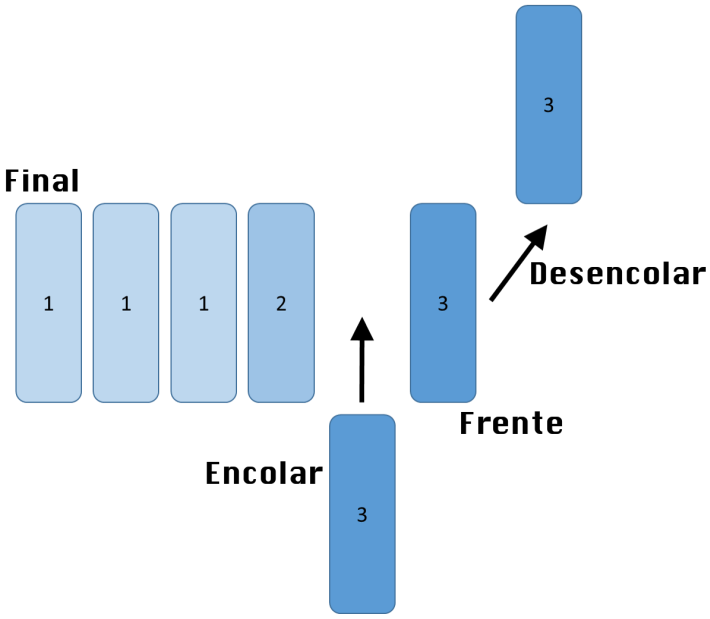
In [7]:
```
# construcción y llenado de la lista
head = lista_enlazada()
head.push(nodo("d1"))
head.push(nodo("d2"))
head.push(nodo("d3"))
head.push(nodo("d4"))
head.push(nodo("d5"))
head.push(nodo("d6"))

# uso del método sobrecargado str() y el borrado
print(str(head))
print(type(head.pop()))
print(str(head))
```

```
(d1)->(d2)->(d3)->(d4)->(d5)->(d6)->(None)
<class '__main__.nodo'>
(d2)->(d3)->(d4)->(d5)->(d6)->(None)
```

Cola de prioridad (priority queue)

Out[8]:



Una cola de prioridades es un tipo de dato abstracto similar a una cola en la que los elementos tienen adicionalmente, una prioridad asignada. En una cola de prioridades un elemento con mayor prioridad será desencolado antes que un elemento de menor prioridad. Si dos elementos tienen la misma prioridad, se desencolarán siguiendo el orden de cola.

Operaciones

Una cola de prioridad ha de soportar al menos las siguientes dos operaciones:

- **encolar (con prioridad):** se añade un elemento a la cola, con su correspondiente prioridad.
- **desencolar (con prioridad):** se retorna y elimina el elemento con mayor prioridad más antiguo que no haya sido desencolado de la cola.

Además suele implementarse una función frente (que habitualmente aquí se denomina encontrar-máximo o encontrar-mínimo), y que habitualmente se ejecuta en tiempo $O(1)$. Esta operación y su rendimiento en tiempo es crucial en ciertas aplicaciones de las colas de prioridades.

Ciertas implementaciones avanzadas pueden incluir operaciones más complejas para la inspección de los elementos de mayor o menor prioridad, borrar la cola o ciertos subconjuntos de la cola, realizar inserciones en masa, la fusión de dos colas en una, aumentar la prioridad de los elementos, etc.