MANEJO DE UNA ESTRUCTURA PILA

```
#-*- coding: utf-8 -*-

# Inicializa una pila
pila = []

# ¿La pila está vacía?
print(len(pila) == 0)                # True

pila.append('a')                     # ['a']
pila.append('b')                     # ['a', 'b']
pila.append('c')                     # ['a', 'b', 'c']

print("Pila inicial:", *pila)        # Pila inicial: a b c
print(pila[-1])                      # c
print("pop:", pila.pop())            # pop: c
print(pila[-1])                      # b
print("Pila final:", *pila)          # Pila final: a b

# ¿La pila está vacía?
print(len(pila) == 0)                # False
```

MANEJO DE UNA ESTRUCTURA COLA

```
#-*- coding: utf-8 -*-

from collections import deque

# Inicializa una cola
cola = deque([])

# ¿La cola está vacía?
print(len(cola) == 0)                # True

cola.append('a')                     # ['a']
cola.append('b')                     # ['a', 'b']
cola.append('c')                     # ['a', 'b', 'c']

print("Cola inicial:", *cola)        # Cola inicial: a b c
print(cola[0])                       # a
print("popleft:", cola.popleft())    # popleft: a
print(cola[0])                       # a
print("Cola final:", *cola)          # Cola final: b c

# ¿La cola está vacía?
print(len(cola) == 0)                # False
```

MANEJO DE UNA ESTRUCTURA BICOLA

```
#-*- coding: utf-8 -*-

from collections import deque

# Inicializa una BiCola
bicola = deque([])

# ¿La bicola está vacía?
print(len(bicola) == 0)                          # True

bicola.append('b')                               # ['b']
bicola.append('c')                               # ['b', 'c']
bicola.append('d')                               # ['b', 'c', 'd']

print("BiCola inicial:", *bicola)                # BiCola inicial: b c d

bicola.appendleft('a')                           # ['a', 'b', 'c', 'd']
print(*bicola)                                   # a b c d
print("pop:", bicola.pop())                      # pop: d
print(*bicola)                                   # a b c
print("popleft:", bicola.popleft())              # popleft: a
print("BiCola final:", *bicola)                  # b c

print("{} - {}".format(bicola[0],bicola[-1]))    # b - c

# ¿La bicola está vacía?
print(len(bicola) == 0)                          # False
```

MANEJO DE UNA ESTRUCTURA COLA DE PRIORIDAD

```
#-*- coding: utf-8 -*-
from queue import PriorityQueue

# Inicializa la cola de prioridad
cp = PriorityQueue()

# ¿La cola de prioridad está vacía?
print(cp.empty())               # True

cp.put(100)                     # [100]
cp.put(10)                      # [100, 10]
cp.put(50)                      # [100, 50, 10]

print("Cola de Prioridad inicial: ", cp.queue)

print(cp.queue[0])              # 10
print("get:", cp.get())         # get: 10
print(cp.queue[0])              # 50

print("Cola de Prioridad final: ", cp.queue)

# ¿La cola de prioridad está vacía?
print(cp.empty())               # False

# Inicializa la cola de prioridad
clientes = PriorityQueue()

# ¿La cola de prioridad está vacía?
print()
print(clientes.empty())             # True

clientes.put((100, "a"))            # [<100, a>]
clientes.put((10, "b"))             # [<100, a>, <10, b>]
clientes.put((50, "c"))             # [<100, a>, <50, c>, <10, b>]

print("Cola de Prioridad (clientes) inicial: ", clientes.queue)

print(clientes.queue[0])            # (10, 'b')
print("get:", clientes.get())       # get: (10, 'b')
print(clientes.queue[0])            # (50, 'c')

print("Cola de Prioridad (clientes) final: ", clientes.queue)

# ¿La cola de prioridad está vacía?
print(clientes.empty())             # False
```