Delete element from dictionary : del d[x]
Get element from dictionary & defaut : d.get(x,0) [default value here is 0 i.e. when element not found]

Bisect:

a. **bisect.bisect_left :**
    1. lower bound (on left val<x, in right val>x)
    2. Input: array, value
    3. Output: index

b. **bisect.bisect_right/bisect.bisect:**
    4. upper bound (on left val<=x, in right val>x)
    5. Input: array, value
    6. Output: index

No need of flags in for loop:

```
for i in range(l+1,r):
   if days[i]<days[l] or days[i]<days[r]:
      l=i
      r=i+k+1
      break
else:
   ans=min(ans,max(days[l],days[r]))
   l,r=r,r+k+1
```

Itertools:

a. **itertools.product**
    1. Cartesian product
    2. Input: iterable/array, repeat=k (repeat k, by default 1)
    3. Output: iterable tuples
b. **Itertools.permutations**
    1. Permutations
    2. Input: iterable/array, r=k (k length tuples)
    3. Output: iterable tuples
c. **Itertools.combinations**
    1. Combination
    2. Input: iterable/array, r=k (k length tuples)
    3. Output: iterable tuple
d. **itertools.combination_with_replacement**
    1. Combination with replacement
    2. Input: iterable/array, r=k (k length tuples)

3. Output: iterable tuple

e. Itertools.accumulate

1. Presum
2. Input: iterable/array
4. Output: iterable (convert to list)

heapq:

```
import heapq

listForTree = [1,2,3,4,5,6,7,8,9,10,11,12,13,14,15]

heapq.heapify(listForTree) # For min heap

heapq._heapify_max(listForTree) # For max heap

heapq.heappop(minheap) # pop min

heapq._heappop_max(maxheap) # pop min
```

`heapq.heappush(`*heap*, *item*`)`

```
class MinHeap(object):
  def __init__(self): self.h = []
  def heappush(self,x): heapq.heappush(self.h,x)
  def heappop(self): return heapq.heappop(self.h)
  def __getitem__(self,i): return self.h[i]
  def __len__(self): return len(self.h)

class MaxHeap(MinHeap):
  def heappush(self,x): heapq.heappush(self.h,MaxHeapObj(x))
  def heappop(self): return heapq.heappop(self.h).val
  def __getitem__(self,i): return self.h[i].val

minh=MinHeap()
maxh=MaxHeap()
```

Queue

queue.**Queue**

queue.LifoQueue

queue.PriorityQueue

`q = queue.Queue()`

`Append → put : q.put(9)`

`Pop → get : q.get()`

`q.empty()`

`q.full()`